RMIT UNIVERSITY

ENHANCITE

TECHNICAL REPORT

Version 1.0

Start Date: 22/05/2024

Authors : Kelvin Young, s3899733
Benjamin Merkli, s3896757
Tianlang Huang, s3886233
Samuele Andre Reyes, s3910311
Richard Edbert Hariyanto, s3868156

Document Control

Version #	Implemented By	Implementation Date	Reviewed By	Approval Date	Reason
1					
2					
3					
4					
5					

TABLE OF CONTENTS

Contents

1 Executive Summary

Enhancite is a cutting-edge software solution that revolutionises mental health treatment by combining real time data visualisation with virtual reality (VR) technology. This innovative product allows clinicians to monitor and quantify their patients' physiological responses during therapeutic sessions, enabling a more personalised and effective approach to treatment.

Enhancite's core functionality is its ability to display live graphs and visualisations of data collected from body sensors worn by patients. These sensors track vital signs such as heart rate, body temperature, and electrodermal skin response, providing clinicians with valuable insights into their patients' emotional and physiological states.

Enhancite is designed for mental health professionals, including psychologists, therapists, and counsellors, who specialise in treating conditions such as post-traumatic stress disorder (PTSD), anxiety disorders, and phobias. The product caters to both in person and telehealth settings, allowing said professionals to remotely monitor and treat patients from the comfort of their homes.

The mental health industry is rapidly growing, with an increasing demand for innovative and effective treatment solutions. Enhancite positions itself at the forefront of this market, offering a cutting-edge approach to exposure therapy that has the potential to significantly improve patient outcomes.

Our product represents a groundbreaking innovation in the field of mental health treatment, integrating modern technologies into a comprehensive, easy-to-use platform tailored for professional use. With its unique value proposition, strong market potential, and robust technical foundation, Enhancite is poised to revolutionise the way mental health professionals treat their patients.

2 INTRODUCTION

In the rapidly evolving landscape of mental health treatment, there is a growing need for innovative solutions that can provide personalised, quantitative, and effective care[1]. Traditional approaches to therapy rely heavily on qualitative observations and self-reported assessments[2], which can be influenced by various factors and may not provide a complete or even accurate picture of one's condition. Furthermore, the increasing demand for telehealth services has highlighted the importance of remote monitoring and treatment capabilities[3].

Enhancite, our cutting-edge software solution, was developed to address these challenges and revolutionise the way mental health professionals approach exposure therapy. The primary objective of this project is to create a comprehensive platform that seamlessly integrates real time data visualisation with virtual reality (VR) technology, enabling clinicians to monitor and quantify their patients' physiological responses during therapy sessions.

The problem that Enhancite aims to solve is the lack of objective, data-driven insights into a patient's emotional and physiological state during exposure therapy[2]. By incorporating

biometric data from body sensors, Enhancite offers a quantitative approach to therapy, allowing clinicians to make more informed decisions based on real time physiological indicators.

Moreover, the integration of VR technology addresses the need for immersive and controlled exposure environments[4]. Patients can be safely exposed to simulated traumatic situations while clinicians monitor their vital signs and reactions, enabling a more personalised and effective treatment approach[4].

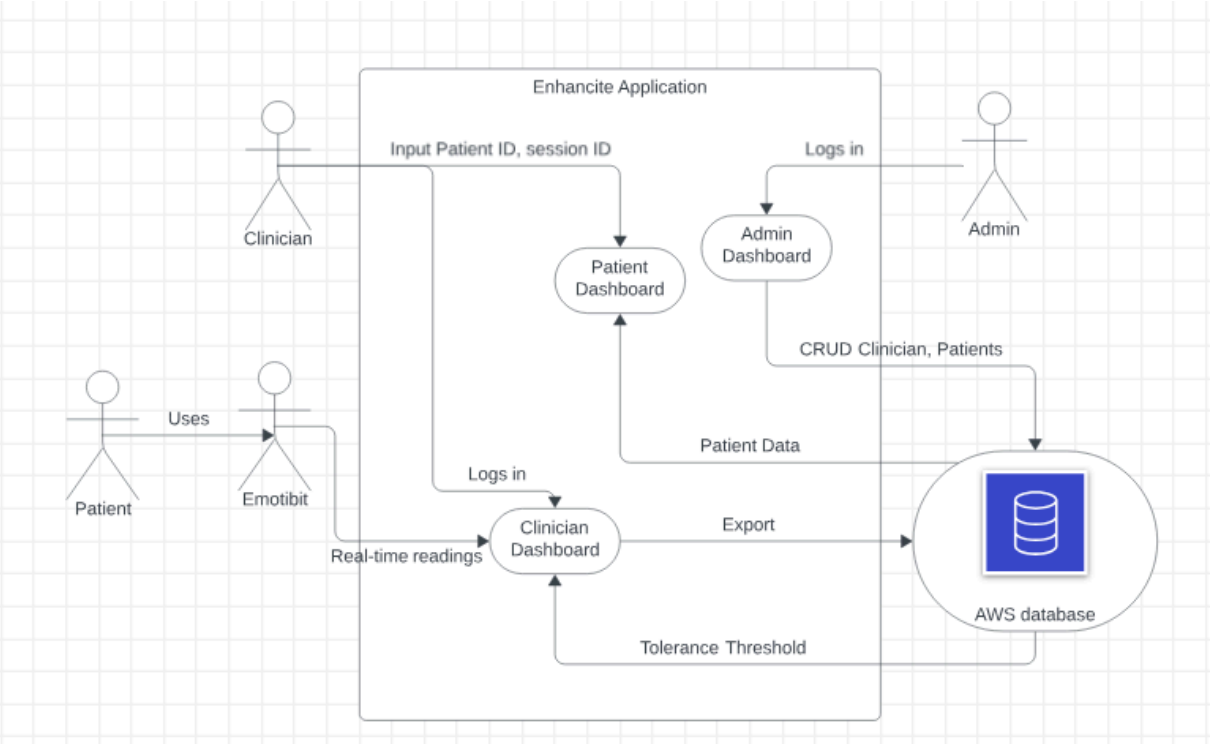
The stakeholders involved in this project include mental health professionals, such as psychologists, therapists, and counsellors, who specialise in treating conditions like post-traumatic stress disorder (PTSD), anxiety disorders, and phobias. Additionally, patients seeking effective and innovative mental health treatment solutions are key beneficiaries of this product.

Furthermore, Enhancite caters to the growing telehealth industry, enabling remote monitoring and treatment of patients from the comfort of their homes. This addresses the increasing demand for accessible and convenient mental health services, particularly in underserviced or remote areas[5].

By combining new technologies, Enhancite aims to revolutionise the mental health treatment landscape, providing clinicians with a powerful tool for quantitative and personalised therapy. The integration of real-time data visualisation and VR exposure therapy offers a unique and comprehensive solution that addresses the limitations of traditional approaches, ultimately improving patient outcomes and advancing the field of mental health care.

3 REQUIREMENTS

3.1 USE CASE DIAGRAM



3.2 FUNCTIONAL REQUIREMENTS SPECIFICATION

Req. ID	Description	Priority
R1	As a clinician, I want to see biometric data so I can better treat my patients.	High
R2	As a clinician, I want to see more biometric data so I can better understand the different measures my patients show.	High
R3	As a clinician, I want to export a patient's data so other people can use the data.	Medium
R4	As a clinician, I want to see a patient's readings appear in real-time so I can respond to my patient's reactions quickly	High
R5	As a clinician, I want to be alerted when a patient's vitals exceed a threshold, so I do not overexpose my patient	Medium
R6	As a clinician, I want to have the sensor connect to the Enchanite application without third-party programs, so it is simple to use.	Medium
R7	As a clinician, I want a viewport next to the real-time data so I can see what the patient is experiencing.	Low

3.3 NON-FUNCTIONAL REQUIREMENTS (NFR) SPECIFICATION

NFR	DESCRIPTION	?

AVAILABILITY	Solution must be operable and accessible when required for use	Application must work when used by Clinicians and Admins when requested.
RELIABILITY	The product will perform is required functions under an extended period of time	Application must be reliable enough to last an extended session, up to an hour.
USABILITY	Ease of use by those who do not have great IT skills	Clinicians and admins must be able to easily use the application.
SECURITY	Logins required to restrict users to their specific actions and to connected users	Admins only able to delete clinicians, admins require login to access this action.
MAINTAINABILITY	The product must have the flexibility to be updated with ease	Use of appropriate coding practices with comments and documentation allowing future users to understand code.
FUNCTIONALITY	The solution must execute the required functions in the scope	Meeting the client's requirements.
PORTABILITY	The product must be able to be run on different platforms	Emotibit able to run on Mac and Windows. Website runs on old systems.
SCALABILITY	The product is to be setup with the ability of future development/ expansion	Flexible infrastructure built on node.js to allow for expansion.
COMPATIBILITY	The solutions operations must be compliant to the client's environment	The Enhancite application must have the ability to run on standard operating systems in medical setting.

4 ARCHITECTURE

The project architecture is designed to provide a multi-user platform catering to Clinicians (User A), and the potential for administrators (User B) to manage user accounts. Each user type has a dedicated dashboard, each serving specific functions and interaction privileges.

1. Clinician Dashboard:
   - Upon login, Clinicians have access to a personalized dashboard.
   - The dashboard displays real-time data captured by an Emotibit device attached to a patient's body.
   - Clinicians can access and interpret this data to make informed decisions during therapy sessions.
   - An alert system is in place; if a Patient records a heart rate exceeding a predefined threshold, the system immediately notifies the Clinician. The table will also flash in red.

- An export button can be pressed to send all the data that was captured in a session to a database in AWS RDS.

2. Patient Dashboard:
   - Patients have personalized dashboards
   - Their dashboard focuses on guiding Patients on how to use the Emotibit device correctly, providing a step-by-step guide to prepare for their sessions.
   - It is meant to contain a feature that calls the RDS through API to show the session data of this patient, so Clinicians can see the past sessions of a patient.

3. Admin Page:
   - Admin Page is implemented as a vital security feature.
   - Admins are responsible for managing user accounts, specifically granting Clinicians access to their dedicated dashboards.
   - User registration for Clinician accounts is restricted; only those authorized by the administrator can access Clinician features.
   - The admin dashboard is kept separate from the Clinician dashboard, ensuring the separation of administrative and user functionalities.

Key Components of the Architecture:

1. User Authentication: A secure login system is in place to ensure that each user type accesses their respective dashboard.

2. Real-time Data Integration: The architecture integrates with the Emotibit device to capture and present real-time data. The data is accessible only to Clinicians for analysis.

3. Alert System: An alert mechanism is implemented to notify Clinicians when a patient has a heart rate above a specific threshold, enabling rapid responses.

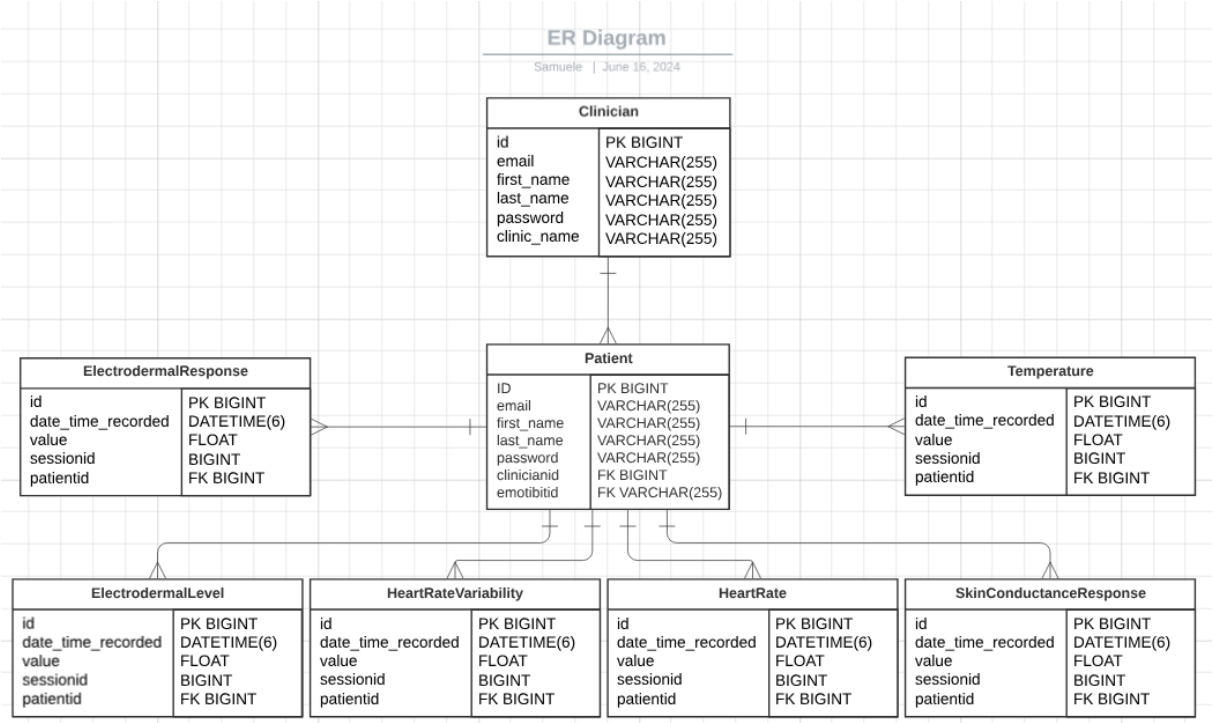
4. User Guidance: The patient dashboard includes guidance on using the Emotibit device, ensuring Patients are adequately prepared for their sessions.

5. Scalability: The architecture is designed to support future expansion and integration of VR/XR projects, enhancing the platform's capabilities.

6. Administrator Controls: Only Administrators have access to deleting users, ensuring the security and integrity of user accounts.

The project's architecture focuses on providing a secure, efficient, and user-specific platform for Clinicians, Patients, and potential administrators, allowing for real-time data interpretation and future growth in functionality.

Backend Architecture: Entity Relationship Diagram



The AWS RDS tables follow the above structure. Each datastream has a many to one relationship with patient, where each record of a datastream is linked to a singular patient and a patient can have many records for each datastream. A patient is assigned to a singular clinician while a clinician can have multiple patients, explaining their one-to-many relationship.

5 TECHNICAL FRAMEWORK

Real-time Graphs Showing Health Data Graphs that show health data in real time

Using the udpport.py tool to get live data, the real-time graphs in our project are made to show health data that changes all the time. Recharts, a charting tool built on React components that can be put together in different ways, was used to make these graphs. Through a set of defined components, Recharts lets you make different kinds of charts. It works with many types of charts, like Line Charts, Bar Charts, and Pie Charts. It also works with extras like tooltips, labels, and axes.

We use line charts to show real-time health data for our unique use case. JavaScript XML (JSX) is used to change the designs. JSX is a syntax extension that lets us write HTML-like code right in JavaScript. This makes it simple to create and change the way data is shown graphically in a declarative way. Using JSX makes it easier to change the chart on the fly as new information comes in.

Charts for Setting Styles and Heart Rate Alerts System

The way our real-time graphs look is controlled by JavaScript XML (JSX), which is like HTML as we already said. Because of its flexibility, CSS lets us set the style of the line charts, including

things like line colour, size, and placement. This makes sure that the graphs are not only useful, but also nice to look at and simple to understand.

Our system comes with a Heart Rate Alert System in addition to simple styling. The goal of this system is to let people know when the heartbeat data hits important levels. When this kind of threshold is passed, the heart rate line graph's style changes in real time to draw attention to the strange event. JavaScript is used to make this alert system work, which lets us change the style of the plots in real time based on the data we get.

Taking charge of udpport.py Script

To make sure that the udpport.py script only works in the right situation, we built control methods into the script itself. This method makes sure that udpport.py only runs on the Clinic Dashboard pages, so it doesn't run in other situations where it's not needed.

Checking the current page and only collecting and sending data when the ClinicDashboard is live is what the control logic in udpport.py is for. This built-in feature gets rid of the need for a separate control script, which speeds up the whole process.

udpport.py also uses the Flask framework because it is easy to use and works well. Flask lets us make a simple web app that stores control logic directly in the script and makes it easier to handle. With Flask, we can set up routes and manage how data moves between the frontend and server parts without any problems. This setup makes sure that udpport.py is only run when it's needed, which saves resources and keeps the real-time data show accurate.

6 IMPLEMENTATION

1.Frontend Overview: The front end of the ClinicDashboard is made with React, a flexible JavaScript tool that is known for making dynamic user interfaces possible. The ClinicDashboard component is at its core the main controller. It brings together different React hooks, WebSocket connections, and third-party libraries to improve usefulness and user interaction.

2.State Management: The useState hook in React is a big part of how the ClinicDashboard handles state. This basic hook lets the dashboard keep track of and change clinical metrics, like heart rate, skin conductance reaction, accelerometer readings, and electrodermal activity. Each measure is controlled by two state variables: one that updates in real time and the other that keeps an array of data points from the past. This two-state method ensures that the dashboard shows the current values of the metrics and interesting historical trends through interactive graphs.

3.WebSocket Connection: When the ClinicDashboard is first started, it sets up a persistent WebSocket link with the backend server so that real-time data updates can happen without any problems. The ability to receive continuous streams of clinical data is made possible by this link. When the frontend gets data messages through WebSocket, it quickly handles them. This makes sure that the most up-to-date measure values are shown correctly in the

dashboard's graphs and numbers. The WebSocket integration makes the ClinicDashboard more responsive by giving users instant updates and a view of important data in real time.

4.Data Processing: The code in the ClinicDashboard carefully reads and sorts incoming data messages from the WebSocket. Different kinds of measure data are handled by separate parsing functions, which makes sure that the data is correctly extracted and added to the dashboard's state management system. One important part of processing data is making the best use of resources by keeping only the most current data points for each metric. This method not only saves memory but also keeps the speed of the dashboard's real-time graphs, which show how metrics change over time.

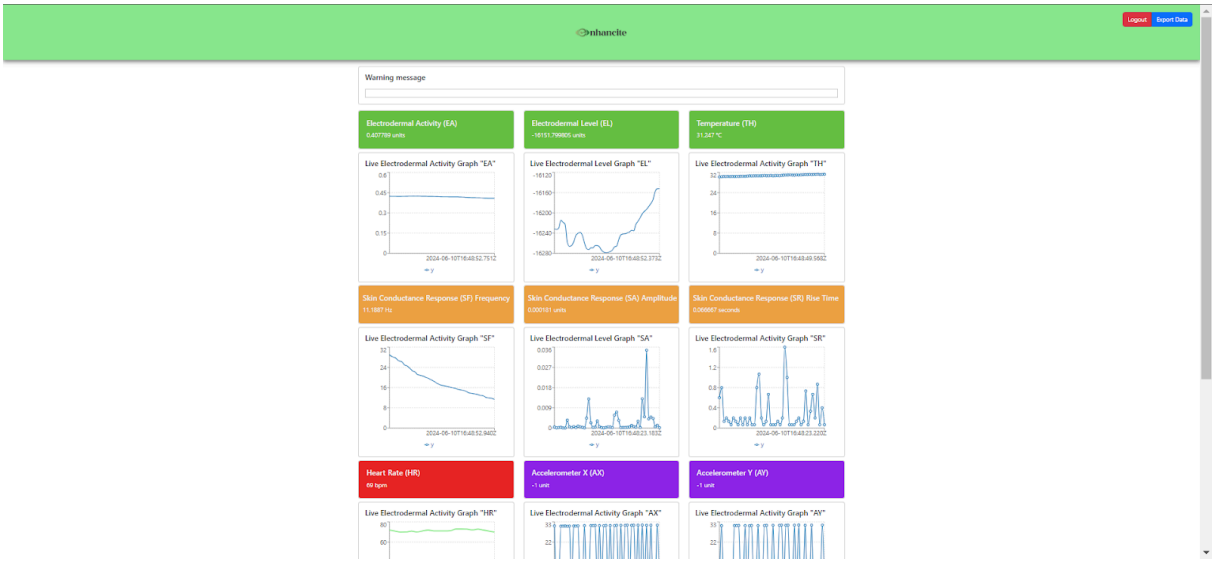
5.Real-Time Data Visualization: Recharts is a complex charting library made just for React apps that lets you see how metrics are changing over time. Recharts' RealTimeLineChart component is used by the ClinicDashboard to show past metric data in interactive line charts. This part can be easily changed to fit your needs. It can take parameters like chart titles, dataset details, Y-axis key values, and stroke colours. Using Recharts, the ClinicDashboard shows clinical metrics in a way that is both visually appealing and useful. This lets users keep a close eye on changes and trends.

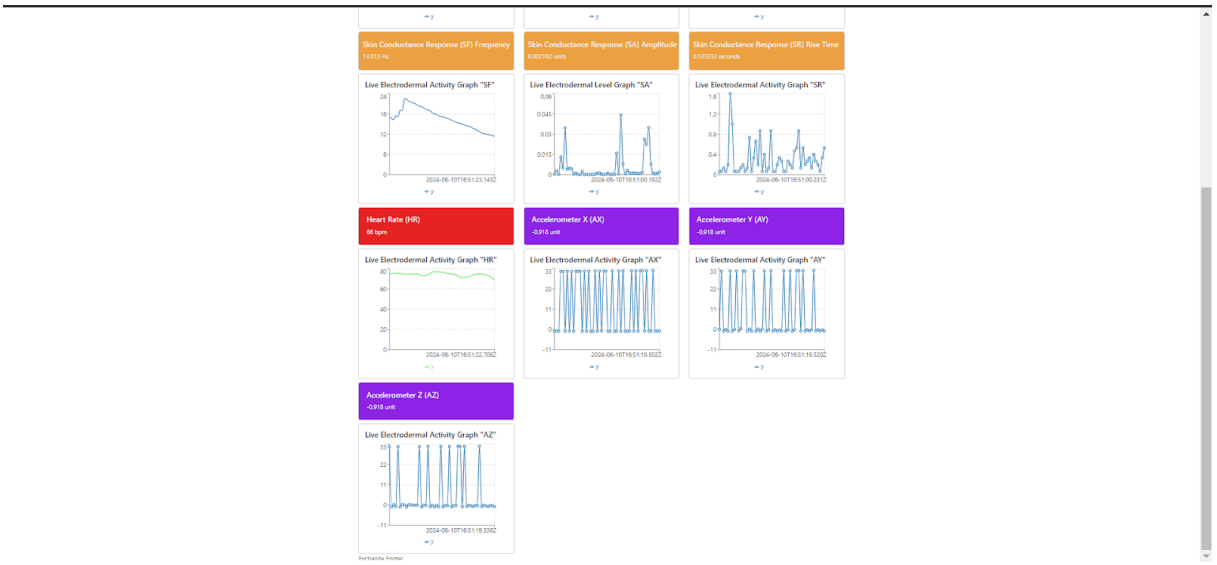
6.User Interface and Layout: Bootstrap is a responsive CSS framework known for its grid system and UI components. It was used to carefully organise the ClinicDashboard's user interface. This framework makes sure that the dashboard works perfectly on all kinds of devices and screen sizes, making it easier for everyone to reach and use. There are organised rows and columns in the layout, and each one holds an easy-to-understand card that shows real-time plots, numbers, and other relevant information. The header and footer parts are clearly defined to help users find their way around and get to the most important dashboard features, making the whole experience simple and effective.

7. API Integration: Axios is a strong HTTP client that works with promises to make interaction with the backend server easy. This is done through RESTful API requests. There are important tasks that the ClinicDashboard does with Axios, like starting and stopping the UDP script that sends data and saving session data. These API interactions happen when the user does something in the dashboard, like click on an interactive button or pick a certain choice. Axios makes sure that contact with the backend is reliable, which lets the ClinicDashboard have full control over how data is sent and how sessions are managed.

Real-time graphs:

The main results of our project are located on the ClinicDashboard page. The ClinicDashboard shows 10 real-time line graphs, EA, EL, TH, SF, SA, SR, HR, AX, AY, AZ.





Heart rate alert system

In the alert system, special attention is given to the visualization of heart rate data to ensure that it effectively communicates critical information. The implementation includes dynamic line color changes in the real-time heart rate graph and a blinking effect to highlight abnormal heart rate values. These features are designed to draw immediate attention to potentially critical conditions.

Dynamic Line Color for Heart Rate

The line color of the heart rate graph is dynamically updated based on the value of the heart rate. This visual cue helps in quickly identifying abnormal heart rate readings:

Normal Range: When the heart rate is within the normal range (between 50 and 100 beats per minute), the line is displayed in green.

High Heart Rate: If the heart rate exceeds 100 beats per minute, the line color changes to red, indicating a potential issue that requires attention.

Low Heart Rate: If the heart rate drops below 50 beats per minute, the line color changes to blue, signaling a potential concern.

This dynamic coloring is achieved by monitoring the heart rate values and updating the stroke color of the line chart accordingly.

Blinking Effect for Abnormal Heart Rate

In addition to changing the line color, a blinking effect is implemented to further emphasize abnormal heart rate values:

High Heart Rate Blinking: When the heart rate exceeds 100 beats per minute, the heart rate display starts blinking in red. This visual effect draws immediate attention to the high heart rate, indicating that immediate action may be required.

Low Heart Rate Blinking: When the heart rate falls below 50 beats per minute, the heart rate display blinks in blue. This serves as an alert to check for possible issues related to low heart rate.

The blinking effect is implemented using conditional CSS classes that are applied based on the heart rate value. These classes toggle the blinking animation to create a noticeable visual effect.

Implementation Details

The heart rate value is continuously monitored through the WebSocket connection, which provides real-time updates. As new heart rate data is received:

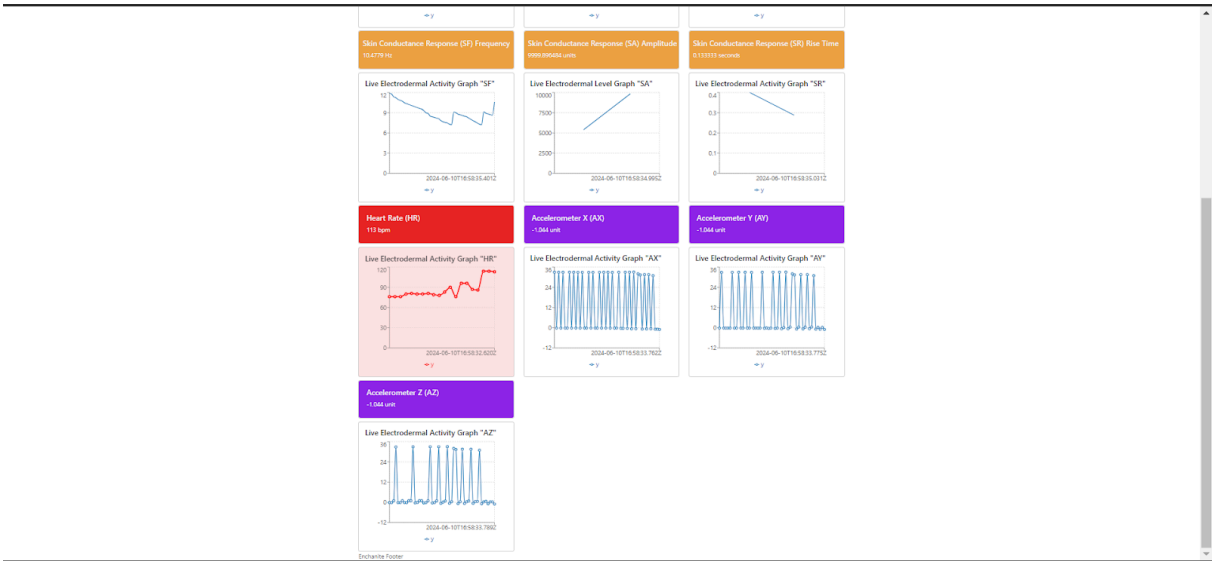
Value Check: The heart rate value is checked against predefined thresholds.

State Update: Based on the value, the state is updated to reflect the appropriate line color and apply the blinking effect.

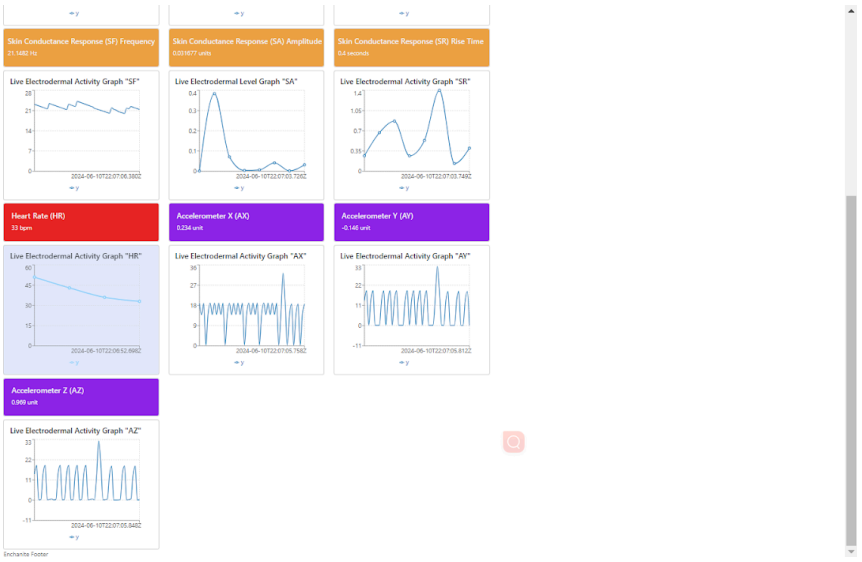
Conditional Rendering: The updated state triggers re-rendering of the heart rate display and graph with the new styles.

This approach ensures that the dashboard provides immediate visual feedback on the patient's heart rate, enabling quick identification of potential issues and facilitating timely interventions.

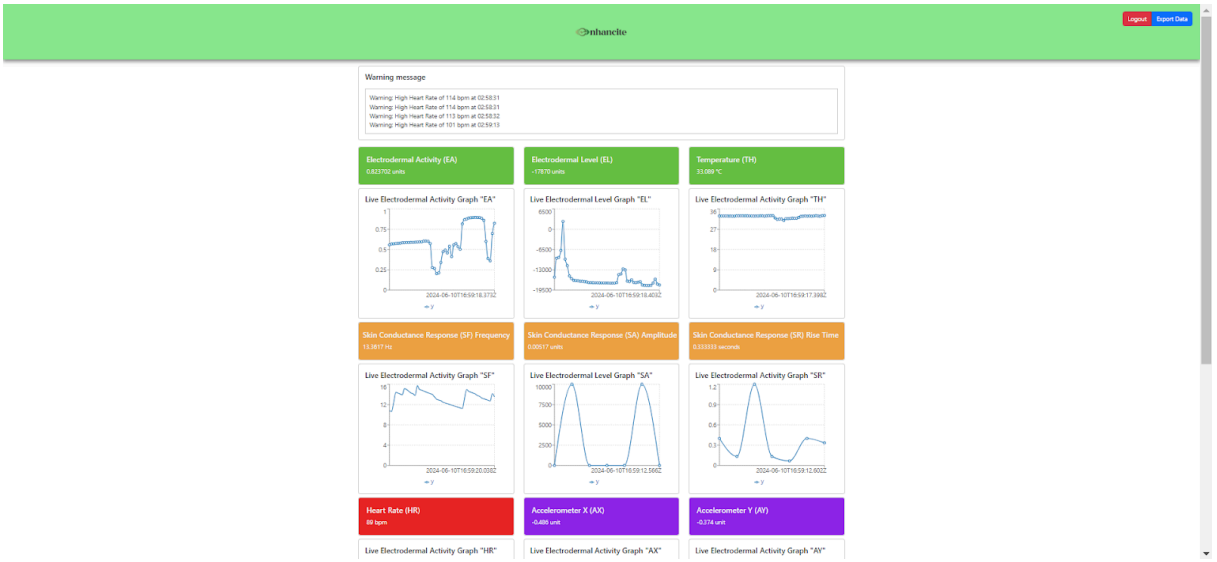
When the heart rate is over 100, the real-time heart rate graph’s line will change to red. And the heart rate graph shows red blinks.



When the heart rate is below 50 or no heart rate data, the real-time heart rate graph's line will change to blue. And the heart rate graph shows red blinks.



When a low heart rate or high heart rate appears, a warning message will show as well.



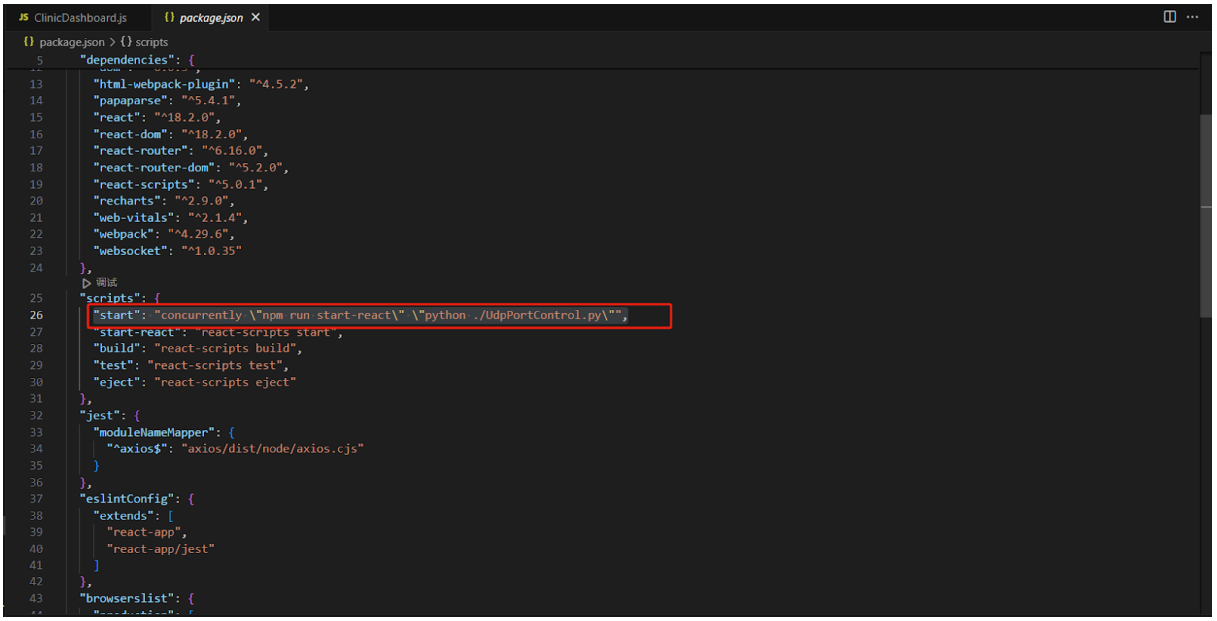
Udpport.py script

Flask is used to set up and manage the execution of udpport.py by the web API for handling the UDP script. Flask's lightweight structure makes it easy to start and stop the script without any problems, allowing it to work independently and efficiently handle real-time data.

Flask Application Setup: Flask was picked because it makes making web APIs easy and gives you a lot of options. It makes setting up web routes and handling HTTP requests easy, which makes it perfect for managing data transmission and running the udpport.py script.

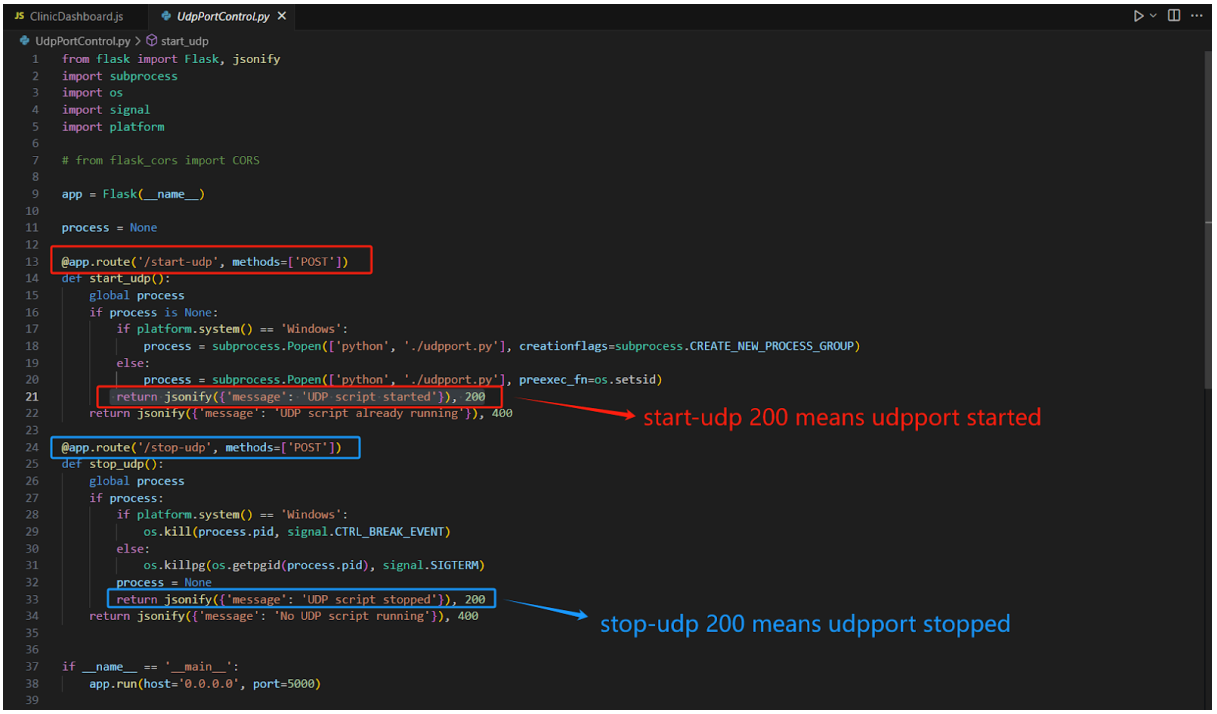
Global Process Management: To make sure it runs smoothly, udpport.py runs as a separate process that is controlled by Python's subprocess module. A global variable keeps track of the script's status as it runs, so the programmed can check on it and stop it if needed. This method makes the best use of resources and improves the dependability of sending important clinical data.

UdpPortControl.py will start when the project starts.



```
{
  "dependencies": {
    "html-webpack-plugin": "^4.5.2",
    "papaparse": "^5.4.1",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router": "^6.16.0",
    "react-router-dom": "^5.2.0",
    "react-scripts": "^5.0.1",
    "recharts": "^2.9.0",
    "web-vitals": "^2.1.4",
    "webpack": "^4.29.6",
    "websocket": "^1.0.35"
  },
  "scripts": {
    "start": "concurrently \"npm run start-react\" \"python ./UdpPortControl.py\"",
    "start-react": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "jest": {
    "moduleNameMapper": {
      "^axios$": "axios/dist/node/axios.cjs"
    }
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
```

UdpPortControl.py

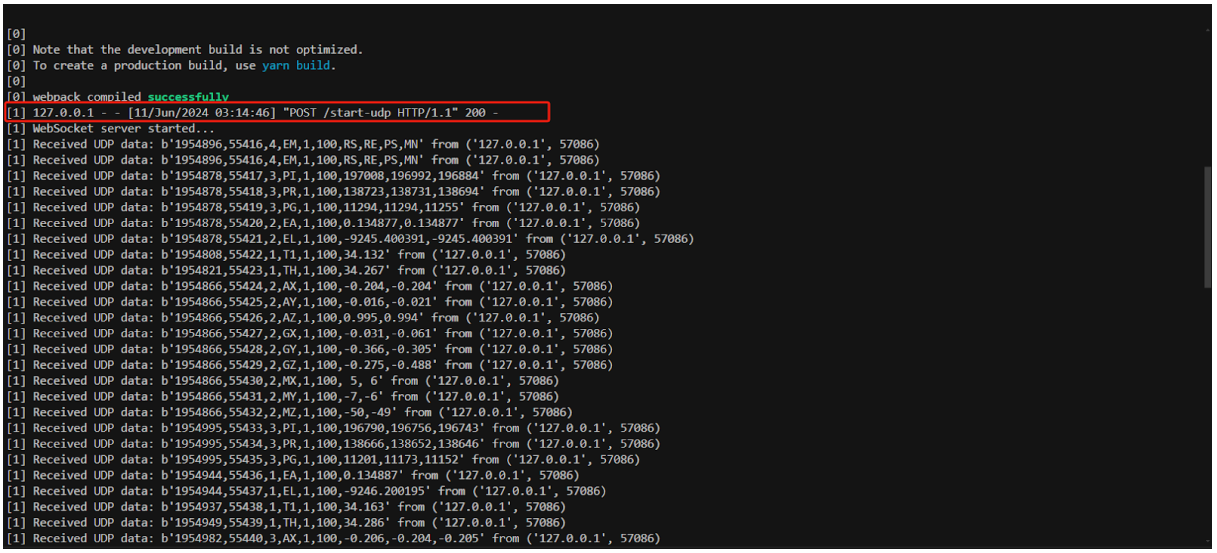


```
from flask import Flask, jsonify
import subprocess
import os
import signal
import platform

# from flask_cors import CORS

app = Flask(__name__)

process = None

@app.route('/start-udp', methods=['POST'])
def start_udp():
    global process
    if process is None:
        if platform.system() == 'Windows':
            process = subprocess.Popen(['python', './udpport.py'], creationflags=subprocess.CREATE_NEW_PROCESS_GROUP)
        else:
            process = subprocess.Popen(['python', './udpport.py'], preexec_fn=os.setsid)
        return jsonify({'message': 'UDP script started'}), 200
    return jsonify({'message': 'UDP script already running'}), 400

@app.route('/stop-udp', methods=['POST'])
def stop_udp():
    global process
    if process:
        if platform.system() == 'Windows':
            os.kill(process.pid, signal.CTRL_BREAK_EVENT)
        else:
            os.killpg(os.getpgid(process.pid), signal.SIGTERM)
        process = None
        return jsonify({'message': 'UDP script stopped'}), 200
    return jsonify({'message': 'No UDP script running'}), 400


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

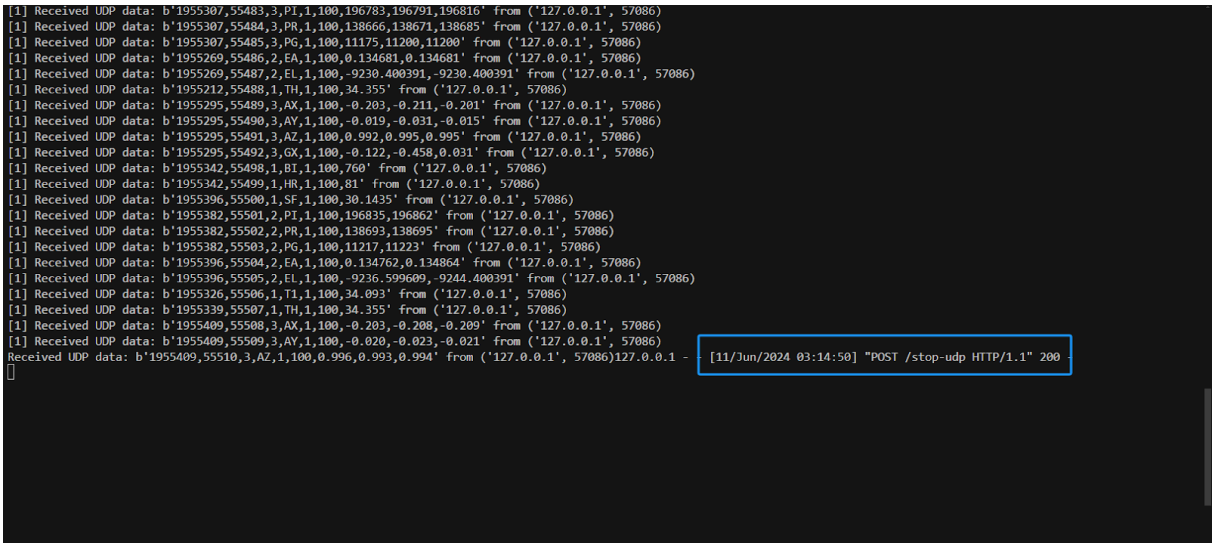
start-udp 200 means udpport started

stop-udp 200 means udpport stopped

When you go to the ClinicDashboard page, the start-udp 200 will show, which means udpport.py started



When you go back to the main page, the stop-udp will show, which means udpport.py stopped.



Data export

Navigating to the Clinician Dashboard creates a sessionId with the current time and a random number, making it unique. When each data is received from UDP, it also creates a timestamp and appends this to the data. When clicking the data export button, this sends each of the datastream's data and sessionId to AWS with a JSON.

```
case 'EL':
  if (!isNaN(data.value1)) {
    const electrodermalLevelPayload = {
      sessionId: sessionId,
      dateTimeRecorded: timestamps[data.id],
      patientId: 1, // Assuming patient ID is always 1 (TODO: CHANGE TO GET PATIENT_ID)
      value: data.value1
    };
    await sendDataToTable('edl', electrodermalLevelPayload)
  }
  break;
```

```
async function sendDataToTable(tableName, payload) {
  try {
    await axios.post(`http://3.27.158.164:8080/datastream/${tableName}`, payload);
    console.log(`${tableName} data sent successfully:`, payload);
  } catch (error) {
    if (axios.isAxiosError(error)) {
      // Handle Axios errors
      console.error('Request failed:', error.message);
      console.error('Request config:', error.config);
      if (error.response) {
        console.error('Response status:', error.response.status);
        console.error('Response data:', error.response.data);
      }
    } else {
      // Handle other errors
      console.error(`Error sending ${tableName} data:`, error.message);
    }

  }
}
```

The current implementation has a hard-coded value for patientId. This is because the current version does not include a textbox to select the patient that is using the Emotibit device. To extend this, patientId's value should correspond to the patientId of the patient using the device.

Backend

The project is built on a Spring Boot framework. Thus, the codebase is built on a conventional Spring Boot structure. The system uses a controller-service-repository pattern. This is where the controller handles HTTP requests, the service calls the repository and processes it to send to the controller, and the repository holds the data.

Controller Layer: This layer handles incoming HTTP requests and orchestrates some interactions within the application. It does not contain any business logic and relies on the service layer for executing code. This is done for modularity and maintainability.

Service Layer: This layer contains business logic. It orchestrates the communication between the controller and repository layers for database interactions. It also sends this information to the controller layer when requested. This separates the business logic from the other layers such that there is a clear distinction between data processing and interaction with the database.

Repository Layer: The repository layer focuses on database interactions. With Java Persistence API (JPA) and DTOs (Data Transfer Objects), they define the structure of the database tables. The project maintains a symmetrical relationship between DTOs and database tables. For example, the Patient class acts as a data transfer object for sending and receiving data in JSON format and a blueprint for the Patient table for the JPA database. This dual role of DTOs streamlines data transmission while ensuring consistent and well-structured data storage in the database.

This architecture enhances the project's maintainability and comprehensibility while adhering to best practices in Spring Boot application design. For those familiar with Spring Boot, navigating and expanding the project should be straightforward.

The update to the DTOs comes with the inclusion of a sessions id. This sessionId is sent through within the JSON to indicate the recording session of the data. Multiple sessions can occur within the day and are separated from one another. Since all the datastream data includes this need to have a session id, this is done within the abstracted class “DataStream”.

Backend Extension

The backend was extended with the addition of new datastreams: Electrodermal response, Electrodermal level and Skin conductance response frequency. Each datastream requires 3 key methods: 1. saving, 2. get the latest value of a specific patient, 3. get the complete history of values of a specific patient for a given session. A detailed guide on implementing this extension is shown as:

1. **Create a New DTO (Data Transfer Objects) Class:**

Create a new DTO class named after a new datastream, like “ElectrodermalLevel”. This class creates a new table in the database for “ElectrodermalLevel” and facilitates transmission of JSON data. Also create a container to keep a list of the new datastream, like “ElectrodermalLevels”.

2. **Establish a Repository Interface:**

Create a new repository class named after a new datastream, like “ElectrodermalLevelRepository”. This interface handles interactions with the database and contains the queries to access and manipulate data in the database.

3. **Update DataStreamService and DataStreamServiceImpl:**

Update the "DataStreamServiceImpl" class to include the 3 essential methods discussed earlier – one for saving data, including the timestamp of when it was received, one for retrieving the latest data and one for accessing the complete data history. These methods interact with the database for the new data stream. Additionally, update the “DataStreamService” interface to include the definitions of these new methods.

4. **Update DataStreamController:**

In the DataStreamController class, these contain the API endpoints. Incorporate 2 GET methods for the controller. These methods reflect the functionality of the service layer methods. Their purpose is to invoke the corresponding methods in DataStreamService, retrieve that requested data, and return an HTTP OK status alongside the JSON object

containing the data. The endpoints reflect the methods, such as "/edl/all/{patientId}" gets all the patient's Electrodermal Level Data.

5. **Leverage Custom Exceptions:**

The project includes a selection of custom exceptions and error messages for your use. Utilise these predefined exceptions to enhance error handling and maintain consistency within the application.

Examining the existing the other datastreams can explain how to easily extend the current implementation for more datastreams. This a systematic procedure and is a process requiring minimal time and effort. As such, this guide is simple and does not explain every bit of code required.

7 Deployment Instructions

7.1 RUNNING THE APPLICATION

Equipment: A computer 16 GB RAM Windows 10, 11/ MAC, Emotibit device

Network: WiFi2.4GHz

Software: VSCODE, Nodejs, EmotiBit Oscilloscope, Python, Python extension in vscode

The file named How to launch the Enhancite project frontend, Emotibit software and device.docx will show all the steps to launch Enhancite project frontend and Emotibit software, device.

7.2 BACKEND DEPLOYMENT

The backend and database are currently deployed in AWS through an Amazon EC2 instance and Amazon RDS respectively.

It is an EC2 instance named as "EmotibitAPI" and is hosted in “ap-southeast-2", which is Sydney. It has been configured with the implementation of security protocols and a controller-service-repository structure. Currently, the security groups are more permissive due to testing purpose. This can easily be updated to be stricter when the official product is released. To access the EC2 instance and make changes to the backend, a PuTTY private key named “emotibit-key-api” is needed. It is recommended to liaise with Kristy to establish the requisite account access. Current details for reference are:

Public IPv4 Address (as of 15/4/2024): 3.27.158.164

Port: 8080 (Note: This is the port on which the Java application operates, not the EC2 instance itself).

The current iteration of the backend API is deployed on this EC2 instance. The steps needed to make modifications to the backend API are as follows:

1. **Prepare release**

Update the source code within the EmotibitBackend repository. The code is written in Java and has a controller-service-repository architecture to access the AWS RDS database. A more detailed explanation is provided in section 6.

2. **Package Backend**

Package the backend into a JAR file with maven by executing “mvn package”. The current implementation in the EC2 instance service file checks for the JAR file “EmotiBit-1.0-SNAPSHOT”. Instructions to change this is in step 5.

3. **Access the EC2 instance**

Establish a SSH connection to the EC2 instance to stop the daemon process. This was done with PuTTY. The host's name is currently “ubuntu@ec2-3-27-158-164.ap-southeast-2.compute.amazonaws.com” but can change if the EC2 instance is restarted. The format of the host is “ubuntu@<<public IPv4 address where “.” Is replaced with “-”>>.ap-southeast-2.compute.amazonaws.com”. The port is 22. Under connection->SSH->Auth, the private key should be “emotibit-key-api.ppk”.

4. **Stop the Service**

Navigate to the directory with the service file with "cd /etc/systemd/system". Use the command “sudo systemctl stop emotibit-api.service” to stop the service. Do not restart the service until you have uploaded the new JAR, as outlined in the next step. Keep this terminal tab open for the upcoming step

5. **Upload the JAR File**

Utilize an FTP or PTP service such as FileZilla to upload the new JAR file from your local machine to the EC2 instance. Replace the "EmotiBit-1.0-SNAPSHOT.jar" with the new JAR file in the "/home/ubuntu/emotibit/" directory. If the name of the JAR file is different from the one in “/etc/systemd/system/emotibit-api.service”, use “nano emotibit-api.service” and change the line “ExecStart=/usr/bin/java -jar /home/ubuntu/emotibit/EmotiBit-1.0-SNAPSHOT.jar” to “ExecStart=/usr/bin/java -jar /home/ubuntu/emotibit/<<New JAR name>>”.

6. **Restart the Service**

After the JAR is successfully uploaded, the service needs to be started. Go back to the repository with the emotibit service file "/etc/systemd/system" and use the "sudo systemctl start emotibit-api.service" command to start the service

7. API Operations

The service is now live and runs the JAR. The API will be running and ready for testing. More explanation for the backend can be found in the "BackendDocumentation.pdf" file.

Database

In AWS, the RDS instance named "emotibit-data" contains the clinicians and patients' login data which are encrypted. It also contains the tables for each of the datastreams. It has been created with all the necessary security protocols. The EC2 instance does not share this security due to testing purposes. When the product is officially released, the database should only be accessible to the websites and relevant parties. The current implementation can be accessed through:

Endpoint: emotibit-data.cxx7fbhxbyig.ap-southeast-2.rds.amazonaws.com
Port: 3306

8 TEST SPECIFICATIONS

Test Case ID	Req. covered	Test Objective	Preconditions	Steps	Expected result	Actual result

T1	R2	Test getLatestEDRofPatient()	id = 9;
			patient_id = 1;
			session_id = 9;
			date = 20/4/2021;
			value = 10.1;
				1. Create EDR with preconditions
				2. Mock MVC to call the behaviour
				3. Check if the return values are the same the EDR from preconditions
					Latest EDR is the EDR that has the same values as preconditions	Latest EDR is the EDR that has the same values as preconditions

T2	R2	Test getAllEDRHistoryofPatient()	id = 9 - 12;
			patient_id = 1;
			session_id = 9;
			date = 20/4/2021;
			value9 = 68.8;
			value10 = 69.8;
			value11 = 70.8;
			value12 = 79.1;
				1. Create EDRs with preconditions
				2. Mock MVC to call the behaviour
				3. Check if the return values are the same as the EDRs from preconditions
					ArrayList with all EDRs from preconditions	ArrayList with all EDRs from preconditions

T3	R2	Test getLatestEDLofPatient()	id = 9;
			patient_id = 1;
			session_id = 9;
			date = 20/4/2021;
				1. Create EDL with preconditions	Latest EDL is the EDL that has the same values as preconditions	Latest EDL is the EDL that has the same values as preconditions

			value = 10.1;	2. Mock MVC to call the behaviour 3. Check if the return values are the same the EDL from preconditions		

T4	R2	Test getAllEDLHistoryofPatient()	id = 9 - 12;
			patient_id = 1;
			session_id = 9;
			date = 20/4/2021;
			value9 = 68.8;
			value10 = 69.8;
			value11 = 70.8;
			value12 = 79.1;
				1. Create EDRs with preconditions
				2. Mock MVC to call the behaviour
				3. Check if the return values are the same as the EDRs from preconditions
					ArrayList with all EDLs from preconditions	ArrayList with all EDLs from preconditions

T5	R2	Test getLatestSCRFofPatient()	id = 9;
			patient_id = 1;
			session_id = 9;
			date = 20/4/2021;
			value = 10.1;
				1. Create SCRF with preconditions
				2. Mock MVC to call the behaviour
				3. Check if the return values are the same the SCRF from preconditions
					Latest SCRF is the SCRF that has the same values as preconditions	Latest SCRF is the SCRF that has the same values as preconditions

T6	R2	Test getAllSCRFHistoryofPatient()	id = 9 - 12;
			patient_id = 1;
			session_id = 9;
			date = 20/4/2021;
			value9 = 68.8;
			value10 = 69.8;
			value11 = 70.8;
			value12 = 79.1;
				1. Create SCRFs with preconditions
				2. Mock MVC to call the behaviour
				3. Check if the return values are the same as the SCRFs from preconditions
					ArrayList with all SCRFs from preconditions	ArrayList with all SCRFs from preconditions

T7	R1	Test if ‘Live Electrodermal Level Graph "EL"’ is in Clinician Dashboard	ClinicianDashboard is rendered	1. getByText the element that should appear on screen 2. expect it to be on the document	EL text should be in the document	EL text is on the document

T8	R1	Test if ‘Electrodermal Level (EL)’ is in Clinician Dashboard	ClinicianDashboard is rendered	1. getByText the element that should appear on screen 2. expect it to be on the document	EL header text should be in the document	EL header text is on the document
T9	R1	Test when no data in EL value	ClinicianDashboard is rendered	1. getByText to get header, nextElementSibling to get the value of EL	EL value should be ‘No data’	EL value is ‘No data’
T10	R4	Test non-null data in EL value	ClinicianDashboard is rendered. testELvalue = 80	1. open mock websocket 2. send stringified EL data 3. getByText to get header, nextElementSibling to get the value of EL	EL value should be ‘80 seconds’	EL value is ‘80 seconds’
T11	R4	Test EL line chart component	ClinicianDashboard is rendered. testELData = { 70, 75}	1. open mock websocket 2. send stringified EL data for both 3. getByText the datapoint on the graph	Both EL values should appear on the line chart component	Both EL values appear on the line chart component
T12	R1	Test if ‘Live Electrodermal Response Graph "ER"’ is in Clinician Dashboard	ClinicianDashboard is rendered	1. getByText the element that should appear on screen 2. expect it to be on the document	ER text should be in the document	ER text is on the document
T13	R1	Test if ‘Electrodermal Response (ER)’ is in Clinician Dashboard	ClinicianDashboard is rendered	1. getByText the element that should appear on screen 2. expect it to be on the document	ER header text should be in the document	ER header text is on the document
T14	R1	Test when no data in ER value	ClinicianDashboard is rendered	1. getByText to get header, nextElementSi	ER value should be ‘No data’	ER value is ‘No data’

				bling to get the value of ER		
T15	R4	Test non-null data in ER value	ClinicianDashboard is rendered. testERvalue = 80	1. open mock websocket 2. send stringified ER data 3. getByText to get header, nextElementSibling to get the value of ER	ER value should be ‘80 seconds’	ER value is ‘80 seconds’
T16	R4	Test ER line chart component	ClinicianDashboard is rendered. testERData = { 70, 75}	1. open mock websocket 2. send stringified ER data for both 3. getByText the datapoint on the graph	Both ER values should appear on the line chart component	Both ER values appear on the line chart component
T17	R1	Test if ‘displays Live Skin Conductance Response Frequency Graph "SF"’ is in Clinician Dashboard	ClinicianDashboard is rendered	1. getByText the element that should appear on screen 2. expect it to be on the document	SF text should be in the document	SF text is on the document
T18	R1	Test if ‘Skin Conductance Response Frequency (SF)’ is in Clinician Dashboard	ClinicianDashboard is rendered	1. getByText the element that should appear on screen 2. expect it to be on the document	SF header text should be in the document	SF header text is on the document
T19	R1	Test when no data in SF value	ClinicianDashboard is rendered	1. getByText to get header, nextElementSibling to get the value of SF	SF value should be ‘No data’	SF value is ‘No data’
T20	R4	Test non-null data in SF value	ClinicianDashboard is rendered. testSFvalue = 80	1. open mock websocket 2. send stringified SF data 3. getByText to get header, nextElementSi	SF value should be ‘80 seconds’	SF value is ‘80 seconds’

				bling to get the value of SF		
T21	R4	Test SF line chart component	ClinicianDashboard is rendered. testSFData = { 70, 75}	1. open mock websocket 2. send stringified SF data for both 3. getByText the datapoint on the graph	Both SF values should appear on the line chart component	Both SF values appear on the line chart component
T22	R1	Test if ‘displays Live Heart Rate Graph "HR"’ is in Clinician Dashboard	ClinicianDashboard is rendered	1. getByText the element that should appear on screen 2. expect it to be on the document	HR text should be in the document	HR text is on the document
T23	R1	Test if ‘Heart Rate (HR)’ is in Clinician Dashboard	ClinicianDashboard is rendered	1. getByText the element that should appear on screen 2. expect it to be on the document	HR header text should be in the document	HR header text is on the document
T24	R1	Test when no data in HR value	ClinicianDashboard is rendered	1. getByText to get header, nextElementSibling to get the value of HR	HR value should be ‘No data’	HR value is ‘No data’
T25	R4	Test non-null data in HR value	ClinicianDashboard is rendered. testHRvalue = 80	1. open mock websocket 2. send stringified HR data 3. getByText to get header, nextElementSibling to get the value of HR	HR value should be ‘80 bpm’	HR value is ‘80 bpm’
T26	R4	Test HR line chart component	ClinicianDashboard is rendered. testHRData = { 70, 75}	1. open mock websocket 2. send stringified HR data for both 3. getByText the datapoint on the graph	Both HRvalues should appear on the line chart component	Both HR values appear on the line chart component

9 TEST RESULTS

The test results show that we have implemented most of the features stated through our functional requirements, R1-R5. However, we were unable to implement R6 as there are limitations to the application being on node.js. We do not have kernel level access which prevents us from starting other applications like Oscilloscope. We also did not implement R7 as we did not receive any of the VR licensing. Bravemind was the software to be used alongside Enhancite. As such, we are unable to do any testing for a viewport.

10 CYBER SECURITY

In the context of a medical software product, cybersecurity is of paramount importance due to the sensitive nature of the data involved and the potential impact on patient privacy and well-being. This section outlines key cybersecurity considerations to ensure the confidentiality, integrity, and availability of the system and its data.

DATA PRIVACY AND CONFIDENTIALITY

Ensuring the confidentiality and privacy of patients' biometric data and personal health information is crucial. Implementing robust encryption mechanisms for data transmissions and storage is essential. Adhering to relevant data protection regulations, such as HIPAA in the United States or GDPR in the European Union, is mandatory.

SECURE COMMUNICATION CHANNELS

Establishing secure communication channels between the patient's sensors, the clinician's device, and the software product is critical. Implementing secure protocols for data transmission over internet is necessary. Ensuring the integrity and authenticity of data during transmission is vital.

ACCESS CONTROLS AND AUTHENTICATION

Implementing strong access controls and authentication mechanisms to ensure only authorised clinicians can access patient data is essential. Utilising multi factor authentication and role-based access controls to limit access based on user roles and permissions is recommended.

TELEHEALTH SECURITY

Addressing the unique security challenges associated with telehealth services, such as remote patient monitoring and virtual consultations, is crucial. Implementing secure video streaming solutions and ensuring the privacy of virtual consultations is essential. Providing guidance and best practices for patients using the product from home environments is recommended.

INCIDENT RESPONSE AND DISASTER RECOVERY

Developing and implementing an incident response plan to effectively respond to and mitigate potential security incidents is necessary. Establishing disaster recovery procedures to ensure the availability and continuity of the service in case of system failures or cyber-attacks is crucial.

An access control and authentication solution has been implemented into the current working version of Enhancite. The cybersecurity considerations presented are intended to inform future development of the project.

11 IMPACT ON STAKEHOLDERS

The Enhancite product has the potential to impact various stakeholders and users in the following ways:

Impact on Stakeholders:

1. Mental Health Professionals (Clinicians):
    - Enhancite provides clinicians with an innovative data-driven approach to exposure therapy, enabling them to make more informed decisions based on real-time physiological indicators from patients.
    - The ability to monitor patients' vital signs and physical reactions during simulated traumatic situations allows clinicians to personalise treatment and adjust therapeutic processess accordingly.
    - Remote monitoring capabilities through Enhancite's telehealth features expand the reach of clinicians and improve accessibility to mental health services.
2. Patients:
    - Patients undergoing exposure therapy with Enhancite can benefit from a more personalised and effective treatment approach tailored to their physiological responses.
    - The integration of VR technology creates immersive and controlled exposure environments, potentially enhancing the efficacy of the therapeutic process.
    - Telehealth capabilities enable patients to receive treatment from the comfort of their homes, improving convenience and accessibility.
3. Healthcare Providers and Institutions:
    - Enhancite offers an innovative solution that can potentially improve patient outcomes and treatment efficacy, aligning with the goals of healthcare providers and institutions.
    - The product's telehealth features can help expand the reach of mental health services, addressing the growing demand for accessible care.

Impact on Users:

1. Clinicians (as Users of the Product):
    - Enhancite provides clinicians with a user-friendly interface to monitor and analyse real time biometric data from patients during therapy sessions.
    - The ability to visualise patients' physiological responses and observe their experiences in the VR environment enhances the clinician's understanding and decision-making process.
    - Clinicians can leverage Enhancite's features to streamline their workflow and improve the overall quality of care they provide.
2. Patients (as Users of the Product):
    - Patients using Enhancite can benefit from a more engaging and immersive therapy experience through the integration of VR technology.

- The ability to receive treatment remotely through telehealth features can improve accessibility and convenience for patients.
- Patients may experience a sense of empowerment and control over their treatment process by actively participating in the quantitative approach enabled by Enhancite.

Impact on the Wider Society:

1. Advancement in Mental Health Treatment:
    - Enhancite represents a technological innovation in the field of mental health treatment, potentially setting new standards and paving the way for further research and development in this area.
    - The integration of data visualisation and VR technology in mental health therapy can inspire and encourage other researchers and developers to explore similar approaches.
2. Improved Access to Mental Health Services:
    - Enhancite's telehealth capabilities can help address the growing demand for mental health services, particularly in underserviced or remote areas where access to in-person treatment may be limited.
    - By improving accessibility to mental health care, Enhancite cna contribute to improving the overall well-being of individuals and communities.
3. Destigmatisation of Mental Health Issues:
    - The development and adoption of innovative solutions like Enhancite can help raise awareness about the importance of mental health and the need for effective treatment options.
    - As more people benefit from Enhancite's quantiative and personalised therapeutic approach, it may contribute to destigmatising mental health issues and encourage more individuals to seek help.

Overall, Enhancite has the potential to positively impact various stakeholders, users, and the wider community by revolutionising the way mental health treatment is approached, improving accessibility, and contributing to the advancement of the field.

12 OTHER CONSIDERATIONS

While the current prototype of Enhancite demonstrates the potential of integrating real time data visualisation and VR technology in mental health treatment, there are several areas for improvement and future considerations to be made to enhance the product's proposed functionality and user experience. These considerations are presented independently of the current state of the prototype, which may or may not fully mirror the full intended functionality described in this report.

Client-Server Architecture and Remote Connectivity

One of the key limitations of the current prototype is its inability to facilitate remote connections between clinicians and patients. To address this, a future iteration of Enhancite should adopt a client-server architecture, where the application is hosted on a central server independently accessible to both clinicians and patients.

This approach would enable clinicians to send and receive connection requests to patients, allowing them to view the patient's biometric data and VR screen output remotely. The server would act as a hub, managing the exchange of sensor data and video streams between the connected parties.

Implementing a client-server architecture would require significant architectural changes and the development of secure communication protocols to ensure the privacy and integrity of sensitive patient data.

Dedicated VR Application

The current prototype relies on running multiple programs simultaneously, including a VR application and the Enhancite software, which may pose usability challenges for patients. To streamline the user experience, a dedicated VR application for Enhancite should be developed.

This VR application could be designed specifically for all-in-one platforms like the Meta Quest, leveraging the device's Android based operating system and Bluetooth connectivity[6]. By integrating the VR environment and biometric data collection directly into a single application, patients would benefit from a more seamless and user-friendly experience.

The development of a dedicated VR application would involve close collaboration with VR hardware manufacturers and the integration of their SDKs to ensure optimal performance and compatibility.

Ethical Considerations and Professional Consultation

As a student team at RMIT University, it is crucial to acknowledge the limitations of our expertise and seek professional consultation regarding the ethical implications and potential risks associated with the Enhancite product.

While the concept of using technology to quantify and personalise mental health treatment holds promise, it is essential to consider the potential ethical concerns surrounding data privacy, patient safety, and the responsible use of VR technology in therapeutic settings.

Prototype Limitations and Ongoing Development

It is important to recognise that the current Enhancite prototype is a proof-of-concept and may not fully meet the envisioned functionality and user experience. Ongoing development efforts should focus on addressing the identified limitations and refining the product based on user feedback and real-world testing.

Continuous iteration and improvement are crucial to ensure that Enhancite remains at the forefront of technological advancements in mental health treatment and continues to provide clinicians and patients with a powerful and effective tool for quantitative and personalised therapy.

13 REFERENCES

1. Insel, T. R. (2017). Digital phenotyping: Technology for a new science of behavior. JAMA, 318(13), 1215-1216. https://doi.org/10.1001/jama.2017.11295
2. Kazdin, A. E. (2015). Treatment as usual and routine care in research and clinical practice. Clinical Psychology Review, 42, 168-178. https://doi.org/10.1016/j.cpr.2015.08.008
3. Totten, A. M., Womack, D. M., Eden, K. B., McDonagh, M. S., Griffin, J. C., Grusing, S., & Hersh, W. R. (2016). Telehealth: Mapping the evidence for patient outcomes from systematic reviews. Agency for Healthcare Research and Quality. https://www.ncbi.nlm.nih.gov/books/NBK379320/
4. Bell, I. H., Nicholas, J., Alvarez-Jimenez, M., Thompson, A., & Valmaggia, L. (2020). Virtual reality as a clinical tool in mental health research and practice. Dialogues in Clinical Neuroscience, 22(2), 169-177. https://doi.org/10.31887/DCNS.2020.22.2/lvalmaggia
5. Thomas, N., McDonald, C., de Leau, V., Haag, D., Barson, M., Obermair, H., et al. (2022). Evaluation of a public mental health service transition to digitally-enabled care in the COVID-19 pandemic: Stratified, prospective observational study. Journal of Medical Internet Research, 24(3), e34668. https://www.jmir.org/2022/3/e34668
6. Meta. (n.d.). Meta Quest [Product page]. Retrieved June 12, 2024, from https://www.meta.com/au/quest/