Report Contents

1. INTRODUCTION

For my project, I aimed to enhance the gaming experience, by integrating real-time heart rate data from a programmable open-source smart watch into gameplay. The objective was to create a more immersive and personalized gaming environment by leveraging the physiological data captured by the heart rate sensor. This integration would allow the game to dynamically respond to the player's physiological state, adapting its challenges and interactions accordingly.

In order to achieve this goal, I extensively researched various wearable devices that would be compatible with C# programming and provide open development capabilities. After careful consideration, I selected the Bangle.js watch as the platform for my project. The Bangle.js watch is an open-source smartwatch equipped with a heart rate sensor, making it an ideal choice for integrating physiological data into gameplay.

To begin the project, I dedicated time to thoroughly understand the architecture and functioning of the Blexer middleware. This involved studying the documentation, exploring code samples, and reviewing tutorials. Through this process, I gained a solid understanding of the middleware's core components, APIs, and interfaces. I familiarized myself with the existing codebase and identified the areas where the heart rate data retrieval and transmission functionality needed to be implemented.

With my knowledge of C# programming, I started developing the new function within the Blexer middleware. The goal was to retrieve real-time heart rate data from the Bangle.js watch and transmit it to the game. I followed best practices and utilized the available APIs to establish a stable Bluetooth Low Energy (BLE) connection between the watch and the game. However, despite my efforts, I encountered challenges in establishing a consistent connection due to hardware issues and the inherent instability of BLE connections.

To address these challenges, I engaged in troubleshooting activities, seeking solutions to stabilize the BLE connection. I collaborated with the project team, including the professor and teammates, to explore different approaches and identify potential causes of the connection issues. However, despite our collective efforts, the connection problem persisted, indicating a possible hardware limitation or an inherent instability in the BLE implementation.

Although I was unable to establish a stable connection between the Bangle.js watch and the game, I extensively documented my troubleshooting process, including the steps taken, the potential solutions considered, and the lessons learned. This documentation will serve as a valuable resource for future development or for other researchers who might encounter similar challenges.

Despite not achieving the desired outcome, this project provided valuable insights and learnings. It highlighted the complexities involved in integrating hardware devices and the importance of considering potential limitations and instabilities in the development process. Additionally, I further honed my problem-solving skills, adaptability, and ability to work within a team.

Moving forward, the experience gained from this project will be invaluable in future endeavours. It has deepened my understanding of hardware integration challenges and reinforced the importance of thorough testing and troubleshooting. While the connection issue remains unresolved, this project has laid the foundation for potential future improvements and serves as a reminder of the inherent complexities and uncertainties that can arise in software development projects.

2. CONTEXT

During my project, I had the opportunity to work in the software development department at the UPM center. This department specializes in creating innovative solutions and integrating cutting-edge technologies into various applications. As part of the team, I collaborated with fellow students, researchers, and mentors who provided guidance and support throughout the project.

The lab where I worked was equipped with state-of-the-art development tools and resources, creating an ideal environment for exploring and implementing new technologies. The team fostered a collaborative atmosphere, encouraging knowledge sharing and brainstorming sessions to solve complex challenges. This collaborative approach allowed me to leverage the expertise of experienced professionals and gain valuable insights into software development best practices.

Throughout the project, I actively engaged with my team members, seeking their expertise and guidance to overcome the connection issues between the Bangle.js watch and the game. We held regular meetings where we discussed the challenges faced, shared ideas, and proposed potential solutions. These interactions not only helped me enhance my technical skills but also provided me with valuable insights into effective teamwork and collaboration.

In addition to the internal interactions, I also reached out to external communities and forums to seek advice and solutions for the connectivity problem. By participating in discussions and sharing my experiences, I received valuable feedback and suggestions from other developers who had encountered similar issues. These interactions expanded my network and exposed me to a broader range of perspectives and ideas.

Furthermore, I actively utilized online resources, documentation, and tutorials related to the Bangle.js watch and the Blexer middleware. These resources provided valuable insights into the underlying technologies, architecture, and best practices for development. I explored code samples, studied documentation, and followed tutorials to deepen my understanding of the middleware and its integration with the watch.

In the lab, I had access to a range of equipment and software necessary for the project. I utilized, debugging tools, the Bangle.js watch, and programming environments to implement and test my code. The team provided me with the necessary software licenses, development kits, and hardware components, ensuring that I had the required resources to carry out the project effectively.

Throughout my interaction with the lab and the team, I fostered a proactive approach to problem-solving and learning. I actively sought guidance and feedback from my supervisor, who played a vital role in shaping my project and providing valuable insights. My supervisor's expertise and guidance helped me navigate through the challenges and provided a structured approach to troubleshooting the connection issues. Their continuous support and feedback greatly contributed to my growth as a software developer.

In summary, the UPM centre and the collaborative environment within the software development department provided a conducive setting for me to carry out my project. The lab's resources, the support of my team members, and the guidance of my supervisor all played integral roles in my development as a professional. This context allowed me to maximize my learning opportunities, enhance my technical skills, and gain valuable insights into effective teamwork and problem-solving approaches.

3. PROJECT OBJECTIVES AND TASKS

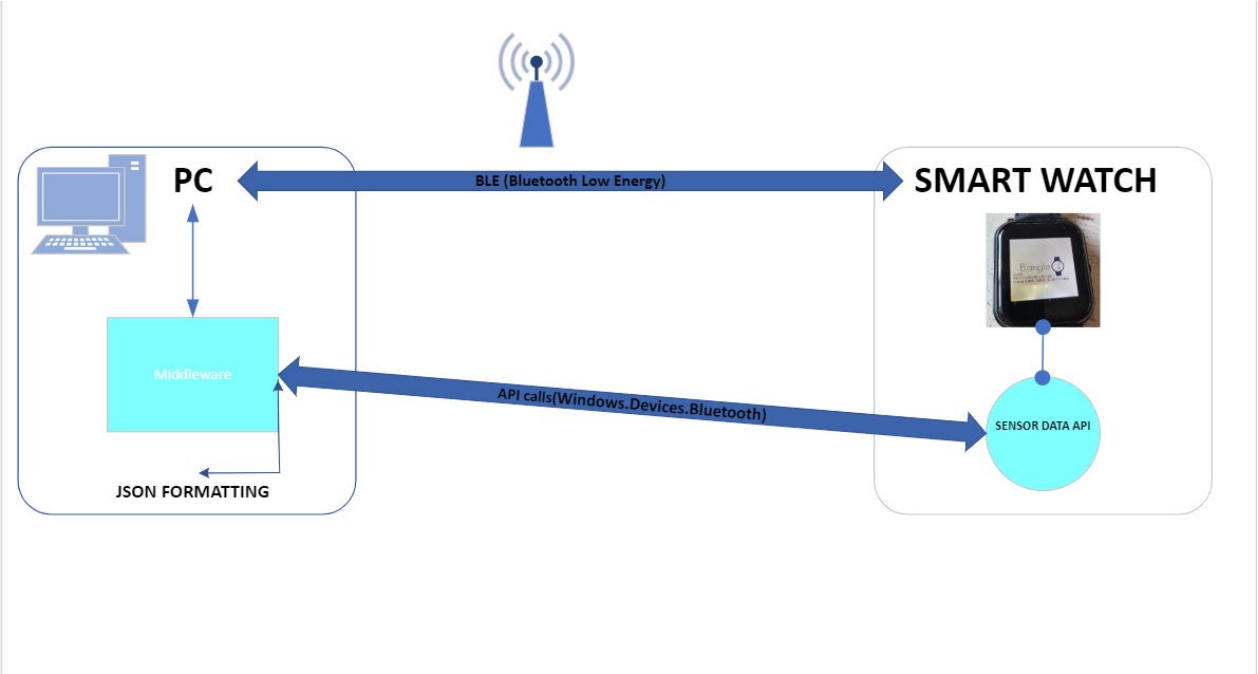
The primary objective of my project was to understand the architecture and functionality of the Blexer middleware and add a new function that would enable the retrieval and transmission of heart rate sensor data from the smartwatch to the game. This objective was crucial for enhancing the gameplay experience and incorporating real-time physiological data into the game dynamics. Throughout the project, I completed several tasks that contributed to the fulfilment of these objectives while encountering various challenges and implementing effective solutions.

To begin with, I conducted extensive research on the architecture and functionality of the Blexer middleware. I studied the existing codebase, reviewed documentation, and explored relevant APIs and interfaces. This deep dive into the middleware's inner workings allowed me to gain a comprehensive understanding of its structure and the different components involved. By familiarizing myself with the existing codebase, I was better equipped to add the new function. In addition to studying the middleware, I also delved into the workings of the UDP (User Datagram Protocol) protocol. I researched its principles, advantages, and use cases in data transmission. Understanding how UDP operates helped me design an efficient and reliable communication channel between the middleware, watch, and PC. Furthermore, I acquired knowledge on transforming data into JSON (JavaScript Object Notation) format. JSON is a widely used data interchange format due to its simplicity and compatibility with various programming languages. I explored different techniques and libraries for encoding and decoding data in JSON, ensuring that the transmitted data is properly formatted and structured. During my research journey, I found several valuable resources that aided my understanding of UDP and JSON. Here are some of the key sources I referred to:

1. "UDP (User Datagram Protocol)" - GeeksforGeeks: This article provided a comprehensive overview of UDP, its features, and how it differs from TCP. It also explained the UDP packet structure and various applications of UDP in networking. [Link: https://www.geeksforgeeks.org/user-datagram-protocol-udp/]
2. "Introduction to JSON" - MDN Web Docs: This guide from Mozilla Developer Network (MDN) introduced the fundamentals of JSON, its syntax, and common use cases. It provided clear examples and practical insights into working with JSON data. [Link: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON]

To select the most suitable smartwatch for the project, I conducted extensive research on various available options. I evaluated their features, compatibility with the chosen middleware, and their programmability in C#. After careful consideration, I decided to focus on the Bangle.js smartwatch due to its numerous advantages. The Bangle.js smartwatch stood out as a highly capable device that aligned well with the project requirements. It offers Bluetooth connectivity, a heart rate sensor, a step counter and an accelerometer/gyroscope. These features make it a versatile and powerful tool for data collection and analysis (Bangle.js 2 - Espruino). Furthermore, the Bangle.js watch provides an SDK (Software Development Kit) that facilitates the development process. To gather more information about the Bangle.js watch and explore its capabilities, I referred to the official Bangle.js website [Bangle.js 2 Software Reference - Espruino]. This provided me with detailed documentation, tutorials, and examples to better understand its programming model and how to leverage its functionalities effectively. During my research, I also considered other smartwatches such as PineTime, Pebble Time, Garmin Forerunner 35, Garmin Vivoactive 3, Samsung Galaxy Watch. However, based on a thorough evaluation, I determined that the Bangle.js

smartwatch was the most suitable choice for this project, given its extensive feature set, compatibility with the chosen middleware, and the availability of resources for development.



The diagram illustrates the communication and data flow between the smartwatch, middleware, and the PC. It highlights the various components involved in this process. Starting with the smartwatch, it communicates with the PC through a BLE (Bluetooth Low Energy) connection. The smartwatch collects sensor data such as heart rate. The middleware acts as the central component in this architecture. It interfaces with both the smartwatch and the PC. It receives the sensor data from the smartwatch through communication interfaces. These interfaces enable the exchange of data between the watch and the middleware. Once the middleware receives the sensor data, it utilizes a Sensor Data API to process and manage the data. The middleware employs API calls to extract relevant information from the sensor data and perform necessary operations. The middleware communicates with the PC application, enabling seamless data transfer between the two. Middleware handles tasks such as formatting the data into JSON (Java Script Object Notation) files. JSON files provide a standardized format for storing and exchanging structured data. The game utilizes the JSON-formatted data for various purposes, such as real-time monitoring.

During the initial phase, I embarked on my journey with the Bangle.js watch by studying the small applications provided on the official Bangle.js website. These examples served as an invaluable resource for me to understand the watch's programming language, which is JavaScript. By thoroughly studying these applications, I gradually gained a deeper comprehension of how to interact with the watch's features and functionalities. One of the notable applications I extensively studied was the "A Battery Widget (with percentage) - Mod by Hank." This widget is a remarkable addition to the Bangle.js watch as it displays the current battery level and charging status in the top right corner of the clock, along with the charge percentage. The widget boasts a simple design that doesn't require any additional settings. Here are some key characteristics of the battery widget:

- Indicates red color when the battery level falls below 30%
- Displays blue color when the watch is being charged

- Compact size of 40 pixels wide

Optional high-level marker, which is a small bar at the 100% battery level point, that can be toggled in the settings. The battery widget seamlessly integrates with the Bangle.js watch, enhancing its functionality. Additionally, I also explored the "Digital clock widget" that can be placed in the bottom widget area. This widget not only shows the time but also provides an opportunity to test the less frequently used widget area on the watch. It is important to note that the battery widget will not be visible when a clock app is active on the watch screen. The widget is a forked project from the official BangleApps repository, and the GitHub link is: https://github.com/espruino/BangleApps/tree/master/apps/widclk. To utilize the widget, you can simply upload the widget file to your Bangle.js watch and then open any compatible app (excluding clock/watch face apps) that supports displaying widgets, especially in the bottom widget area. The battery widget will become visible, providing real-time information about the battery status. By studying this small application, I not only became more familiar with the Bangle.js watch but also honed my programming skills in JavaScript. To test and experiment with the application and widget for the Bangle.js watch, I utilized the online emulator provided by Espruino. The emulator, accessible at https://www.espruino.com/ide/?emulator , offered a convenient platform to simulate the Bangle.js watch environment. Using the emulator, I was able to upload and run the applications I studied, including the battery widget and the digital clock widget, without the need for a physical Bangle.js watch. This allowed me to thoroughly explore the functionalities and features of the applications and gain a deeper understanding of their behaviour. The online emulator provided an excellent opportunity to simulate real-world scenarios and interactions with the Bangle.js watch. It enabled me to test different configurations, visualize the widget placement, and observe the real-time effects of the battery level and charging status on the battery widget. However, it's important to note that the online emulator only supported coding in JavaScript, not C#. Therefore, while I could experiment and write applications in JavaScript using the emulator, I would need a physical Bangle.js watch and the appropriate development environment to program in C# for the watch. Overall, the emulator offered a reliable and convenient way to study and experiment with the Bangle.js watch applications.

After familiarizing myself with the basic applications and widgets for the Bangle.js watch, I shifted my focus towards exploring examples of applications that leverage the heart rate sensor. Concurrently, I delved into the documentation to understand the data flow and how to access heart rate data from the watch's sensor. One noteworthy application I came across is the "Bluetooth Heart Rate Monitor" app, which overrides Bangle.js's built-in heart rate monitor with an external Bluetooth one. When this app is installed, it searches for a heart rate monitor via Bluetooth, establishes a connection, and sends data back to the watch using the Bangle.on('HRM') event, as if it originated from the onboard heart rate monitor. This makes it compatible with various Bangle.js apps, such as the Heart Rate Widget and Heart Rate Recorder. However, it's important to note that it is not compatible with the Heart Rate Monitor app, as it requires live sensor data rather than just BPM readings. To use the Bluetooth Heart Rate Monitor app, I simply install it and then install an app that utilizes the heart rate monitor. While the heart rate monitor was available for Bluetooth pairing, I accessed the app's settings to scan for devices. The Bluetooth Heart Rate Monitor app offers different modes, including Off, Default, Both, and Custom. The Both mode requires explicit activation by an app that intends to use the Bluetooth heart rate monitor, and it operates independently from the internal monito. It serves as a useful tool for debugging or comparing data between the Bluetooth and internal monitors. The resulting "merged" heart rate data can be recorded using the default heart rate monitor recorder. To access the Bluetooth Heart Rate Monitor app, you can visit the following link: https://banglejs.com/apps/?c=bluetooth .

After becoming familiar with the Bangle.js watch, I started developing a console application that

establishes a Bluetooth connection with the watch. To achieve this, I conducted research to find suitable libraries and functions for Bluetooth connectivity in the Windows environment. During my search, I discovered the Windows Bluetooth API, which is a comprehensive set of tools and functionalities provided by Microsoft for working with Bluetooth devices on Windows platforms. The official documentation for the Windows Bluetooth API offered detailed information about the classes, methods, and events available for Bluetooth communication in C#. Importantly, the Windows Bluetooth API supports both traditional Bluetooth and Bluetooth Low Energy (BLE) devices, making it compatible with the BLE capabilities of the Bangle.js watch. To ensure compatibility with the Bangle.js watch's BLE features, I focused on exploring the Windows Bluetooth API and its capabilities. By leveraging this API, I was able to establish a Bluetooth connection with the Bangle.js watch, once. The following link provides valuable information about the Windows Bluetooth API: Windows.Devices.Bluetooth.GenericAttributeProfile Namespace - Windows UWP applications | Microsoft Learn. One of the main motivations for choosing the Windows Bluetooth API as the library for my console application was its official support and compatibility with the Windows platform. Being developed and maintained by Microsoft, the Windows Bluetooth API is a reliable and well-documented tool for working with Bluetooth devices. Additionally, the Windows Bluetooth API provided extensive functionality and flexibility, allowing me to establish a Bluetooth connection with the Bangle.js watch and perform data exchange seamlessly. Its support for both traditional Bluetooth and BLE devices was a crucial factor, as the Bangle.js watch utilizes BLE technology. This compatibility ensured that I could leverage the full potential of the watch's BLE capabilities in my application. Moreover, the Windows Bluetooth API offered a range of classes, methods, and events specifically designed for Bluetooth communication in C#. The comprehensive documentation provided by Microsoft allowed me to understand and utilize these resources effectively, making the development process smoother and more efficient.

After familiarizing myself with the Windows Bluetooth API and its compatibility with the Bangle.js watch, I proceeded to develop the code for data transmission between the watch and the middleware. This stage marked a crucial step in the project, as it involved establishing a reliable connection and enabling seamless communication between the two devices. In the subsequent sections, I will delve into the logic behind the code that I have implemented, explaining the key components, functions, and processes involved in the data transmission. By providing a detailed analysis of the code, I aim to offer insights into the inner workings of the application and showcase the steps taken to ensure efficient and accurate data exchange between the Bangle.js watch and the middleware.

```
using System;
using System.IO;
using System.Linq;
using System.Net.Sockets;
using Windows.Devices.Bluetooth;
using Windows.Devices.Bluetooth.GenericAttributeProfile;
using Windows.Networking.Sockets;
using Newtonsoft.Json;

namespace BangleApp
{
    0 references
    class Program
    {
        0 references
        static async void Main(string[] args)
        {
            // MAC address of the Bangle.js watch
            string bangleMacAddress = "de:c1:32:07:9c:a6"; // Replace with actual MAC address

            // Code to upload to Bangle.js
            string BANGLE_CODE = @"
            Bangle.setHRMPower(1)
            Bangle.on('HRM',function(hrm) {
              var d = [
                'H',
                hrm.bpm,
                hrm.confidence
                ];
              Bluetooth.println(d.join(','));
            })
            ";

            // Create a Bluetooth device object from the MAC address
            var device = await BluetoothLEDevice.FromBluetoothAddressAsync(ulong.Parse(bangleMacAddress));

            // Check if the device is found
            if (device == null)
            {
                Console.WriteLine("The watch is not found.");
                return;
            }
            else
            {
                Console.WriteLine("The watch is found.");
            }
```

```
    // Get the GATT services for the device
    var gattServicesResult = await device.GetGattServicesAsync();

    // Check if any services are found
    if (gattServicesResult.Status != GattCommunicationStatus.Success || gattServicesResult.Services.Count == 0)
    {
        Console.WriteLine("No GATT services are found.");
        return;
    }

    // Find the UART service by its UUID
    var uartService = gattServicesResult.Services.FirstOrDefault(s => s.Uuid == Guid.Parse("6E400001-B5A3-F393-E0A9-E50E24DCCA9E"));

    // Check if the UART service is found
    if (uartService == null)
    {
        Console.WriteLine("The UART service is not found.");
        return;
    }

    // Check if the connection is successful
    if (client.Connected)
    {
        Console.WriteLine("The watch is connected.");
        // Get the network stream from the client
        NetworkStream stream = client.GetStream();

        // Create a binary reader and writer to communicate with the watch
        BinaryReader reader = new BinaryReader(stream);
        BinaryWriter writer = new BinaryWriter(stream);

        // First, reset the Bangle.js
        writer.Write("reset();\n");

        // Wait for it to reset itself
        System.Threading.Thread.Sleep(1500);

        // Now upload our code to it
        writer.Write("\x03\x10if(1){" + BANGLE_CODE + "}\n");

        Console.WriteLine("Ready...");
```

```
            // Define a function to update the chart with heart rate data
            void updateChart(int hr)
            {
                // TODO: implement chart logic here using CanvasJS or similar library
                Console.WriteLine($"Heart rate: {hr} bpm");
            }

            // Define a function to handle the data received from the watch
            void onLine(string line)
            {
                Console.WriteLine("RECEIVED:" + line);
                var d = line.Split(",");
                if (d.Length == 3 && d[0] == "H")
                {
                    // we have an HR monitor reading
                    var hrm = new
                    {
                        bpm = int.Parse(d[1]),
                        confidence = int.Parse(d[2])
                    };
                    // Update the chart with the heart rate data
                    updateChart(hrm.bpm);
                }
            }

            // Handle the data we get back, and call 'onLine'
            // whenever we get a line
            string buf = "";

            while (true)
            {
                // Read data from the stream
                string data = reader.ReadString();

                // Append data to buffer
                buf += data;

                // Split buffer by newline characters
                var lines = buf.Split("\n");

                // Keep the last line in buffer for next iteration
                buf = string.Join("\n", lines); // Use string.Join instead of lines.Join

                // Process each line except the last one
                for (int i = 0; i < lines.Length - 1; i++)
                {
                    onLine(lines[i]);
                }
```

```
                // Check if user pressed any key to exit loop
                if (Console.KeyAvailable) break;

            }

            // Close the connection
            Console.WriteLine("Closing the connection...");

        }

    }
}
```

The main goal of the code is to establish a connection between the console application and the Bangle.js watch via Bluetooth. Once connected, the application uploads JavaScript code to the watch, which enables the watch's heart rate monitor functionality. The code begins by specifying the MAC address of the Bangle.js watch and defining the JavaScript code to upload. It then creates a BluetoothLEDevice object using the MAC address to establish the Bluetooth connection with the watch. Next, the code checks if the device is found and retrieves the GATT services available on the watch. It specifically looks for the UART service, which allows bidirectional communication between the watch and the computer. Once the connection is established, the code sends commands to reset the watch and uploads the JavaScript code. It then enters a loop to continuously receive data from the watch. In each iteration of the loop, the code reads the received data and processes it line by line. It calls the onLine function to handle each line of data, extracting the heart rate information and updating a chart or display. The code uses buffering techniques to ensure complete lines of data are processed, and it checks if the user has pressed any key to exit the loop. Finally, it closes the connection with the watch. Overall, the code establishes a Bluetooth connection, uploads the necessary code to enable heart rate monitoring on the Bangle.js watch, and handles the data received from the watch for further processing or visualization. In the following paragraphs, I will provide detailed explanations for each of the functions used in the code. Now, I will take each function that I wrote in the code and start to explain it. Firstly, let's begin with the main function:

- "static async void Main(string[] args)": The Main method is the entry point of the program and is marked as async to allow the use of await for asynchronous operations.

  "string bangleMacAddress = "de:c1:32:07:9c:a6";" -This line initializes a string variable

bangleMacAddress with the MAC address of the Bangle.js watch.

“string BANGLE_CODE = @"..."; // JavaScript code to upload to Bangle.js” -This line defines a string variable BANGLE_CODE that holds the JavaScript code to be uploaded to the Bangle.js watch. This code sets up the heart rate monitor and handles heart rate data received from the watch.

“vardevice=awaitBluetoothLEDevice.FromBluetoothAddressAsync(ulong.Parse(bangleMacAddress));” -This line establishes a Bluetooth Low Energy (LE) connection with the Bangle.js watch. It uses the MAC address to identify and connect to the watch.

“if (device == null)” - This condition checks if the device object is null, indicating that the Bangle.js watch was not found or could not be connected. If the watch is not found, a message is printed to the console, and the program returns.

“var gattServicesResult = await device.GetGattServicesAsync();” -This line retrieves the GATT (Generic Attribute) services of the connected Bluetooth device (Bangle.js watch). GATT services provide functionality and data available on the device.

“If (gattServicesResult.Status!= GattCommunicationStatus.Success || gattServicesResult.Services.Count == 0)” -- This condition checks if the GATT services retrieval was successful (gattServicesResult.Status != GattCommunicationStatus.Success) and if any services were found (gattServicesResult.Services.Count == 0). If either condition is true, it indicates that no GATT services were found on the Bangle.js watch or there was an error communicating with the watch. In such cases, a message is printed to the console, and the program returns.

“var uartService = gattServicesResult.Services.FirstOrDefault(s => s.Uuid == Guid.Parse("6E400001-B5A3-F393-E0A9-E50E24DCCA9E"));” -This line searches for the UART (Universal Asynchronous Receiver/Transmitter) service in the list of GATT services obtained from the Bangle.js watch. UART is a common protocol used for serial communication over Bluetooth. The provided UUID corresponds to the Nordic UART Service.

“if (uartService == null)”- This condition checks if the uartService object is null, indicating that the UART service was not found on the Bangle.js watch. If the UART service is not found, a message is printed to the console, and the program returns.

The bangleMacAddress variable stores the MAC address of the Bangle.js watch.

The BANGLE_CODE variable stores the JavaScript code that will be uploaded to the Bangle.js watch.

The code starts by establishing a Bluetooth connection with the Bangle.js watch using the BluetoothLEDevice.FromBluetoothAddressAsync method.

It then checks if the device was found and connected successfully. If not, it prints a message and exits the program.

Next, it retrieves the GATT services of the connected device (Bangle.js watch) using device.GetGattServicesAsync().

It checks if GATT services were found. If not, it prints a message and exits the program.

The code searches for the UART service in the list of GATT services using the UUID 6E400001-B5A3-F393-E0A9-E50E24DCCA9E.

If the UART service is not found, it prints a message and exits the program.

Finally, it sets up a TCP client, which will be used to connect to the UART service and communicate with the Bangle.js watch.

- **updateChart** function: "void updateChart(int hr)" –"void updateChart(int hr)" -This line declares the updateChart function, which takes an integer parameter hr representing the heart rate. "Console.WriteLine($"Heart rate: {hr} bpm");" -This line prints the heart rate value received as an argument to the console. It uses string interpolation to include the heart rate value within the printed message. The updateChart function is responsible for updating the chart or displaying the received heart rate data. The function contains a placeholder implementation that simply prints the heart rate value to the console using Console.WriteLine. This placeholder implementation was chosen because the primary goal was to ensure that the program had a functional flow and was able to receive and process the heart rate data correctly. The intention behind using a placeholder implementation was to focus on the core functionality of receiving and processing the heart rate data from the Bangle.js watch. By temporarily using Console.WriteLine to display the heart rate value, it allowed for quick verification and testing of the overall program flow without the need for an extensive charting library or visualization tool implementation. The plan was to later enhance the updateChart function with the actual chart logic using a suitable library such as CanvasJS or a similar one. However, due to some unforeseen issues or constraints, the desired chart logic implementation could not be completed or integrated into the code before testing. Therefore, the placeholder implementation remained in place for the purpose of demonstrating the functionality and verifying the heart rate data reception. It is important to note that the placeholder implementation can be replaced with the specific chart logic as per the requirements of the project. This allows for a more visually appealing and interactive representation of the heart rate data, providing a better user experience.
- **onLine** function: "void onLine(string line)": The onLine function is responsible for handling the data received from the Bangle.js watch. It receives a line of data as a parameter and performs processing based on the content of that line. The onLine function is implemented with the purpose of extracting heart rate data and updating the chart or performing any necessary actions based on that data. Upon receiving a line of data, the function first prints the received line to the console for debugging purposes. It then splits the line by commas to separate the different data elements. The expected format of the line is H,bpm,confidence, where bpm represents the heart rate value and confidence represents the confidence level associated with that value. The function checks if the split line contains three elements and if the first element is equal to "H". This ensures that the line corresponds to the heart rate monitor reading data. If these conditions are met, the function extracts the heart rate value (bpm) and confidence level from the line and creates an anonymous object (hrm) to hold this information. At this point, the function would typically update the chart with the extracted heart rate data using the updateChart function. However, as mentioned earlier, the updateChart function is left as a placeholder implementation in the provided code. Therefore, in its current state, the onLine function calls the updateChart function, passing the extracted heart rate value (hrm.bpm) as an argument. It's important to note that the implementation of the onLine function can be customized according to specific application requirements. Depending on the needs of the project, additional processing or actions can be performed within this function, such as storing the heart rate data, triggering alerts based on specific conditions, or integrating with external systems or APIs for further analysis or visualization. By leveraging the onLine function, the

program can handle the received data from the Bangle.js watch and process the heart rate readings in a customized manner, enabling further analysis or utilization of the heart rate data.

- The while loop in the Main method represents the last lines of code and serves as the program's termination condition. This condition in the Main method is responsible for continuously reading data from the network stream and processing it. This loop ensures that the program keeps receiving data from the Bangle.js watch and performs the necessary actions based on the received data. Within the while loop, the following steps are executed: Read data from the network stream: The reader.ReadString() method is used to read data from the network stream. It reads a string of characters until a newline character is encountered. The read data is stored in the data variable. Append data to buffer: The data string is appended to the buf string, which acts as a buffer for incoming data. This is done to ensure that partial lines of data received from the watch are combined and processed correctly. Split buffer by newline characters: The buf string is split into multiple lines using the newline character ("\n") as the delimiter. The Split method is used to split the buf string into an array of lines. Keep the last line in the buffer for the next iteration: The last line in the lines array is stored back in the buf string. This is done to handle cases where a line of data is only partially received, and the complete line is expected to be received in the next iteration. Process each line except the last one: A for loop is used to iterate over each line in the lines array, except for the last line. This is done to process complete lines of data received from the watch. The onLine function is called for each line to handle the received data. Check if the user pressed any key to exit the loop: The Console.KeyAvailable property is checked to determine if the user has pressed any key. If a key is pressed, it indicates the intention to exit the loop and terminate the program. In such a case, the loop is broken, and the program proceeds to close the connection. This while loop ensures that the program continuously receives and processes data from the Bangle.js watch until the user decides to exit the loop. Each complete line of data is passed to the onLine function for further processing, which can include updating the chart, performing calculations, or triggering additional actions based on the received data. It's important to note that this loop operates asynchronously, allowing other parts of the program to execute concurrently. This enables the program to handle data reception and user input simultaneously, providing a responsive and interactive experience.

During the implementation and testing of the code on the Bangle.js watch, I encountered several challenges that affected the application's functionality. In the subsequent section, I will detail the specific issues I encountered, as well as the steps I took to identify the source of the problem. Additionally, I will describe the modifications I attempted to address these issues.

First, I encountered a significant problem related to the smartwatch's connection with my laptop. I encountered difficulties when attempting to scan for the watch within the application. Despite following the necessary steps, the watch was not being detected by the scanning process. Upon further investigation, I discovered that the issue extended beyond the application itself. It became evident that the watch was unable to establish a connection with the laptop through the standard Bluetooth interface. This discovery added an additional layer of complexity to the problem, as it indicated a potential compatibility or configuration issue between the watch and my laptop's Bluetooth functionality. To troubleshoot this issue, I performed the following steps: Verified the MAC address: I double-checked the MAC address of the Bangle.js watch to ensure it was correctly entered in the code. I confirmed that the MAC address provided in the code matched the actual MAC address of the watch. Bluetooth settings: I checked the Bluetooth settings on my laptop to ensure that Bluetooth was enabled and discoverable. I

also made sure that the Bangle.js watch was in pairing mode and visible to other devices. To address the connectivity issue between the Bangle.js watch and my laptop, I embarked on a debugging process to identify the root cause. As part of this process, I decided to test the functionality of my application by searching for other Bluetooth devices, such as my smartphone and Bluetooth headphones, using different MAC addresses. Surprisingly, this approach yielded positive results. I was able to successfully detect and establish connections with these devices, like my phone or the headphones, using my application.

I conducted further tests with my colleague's laptop running on macOS and a PC. To establish a Bluetooth connection on the PC, I initially used an older version of a dongle that supported the previous version of the Bluetooth protocol. However, the connection did not work. I then switched to a newer version of the dongle, which supported the latest version of the Bluetooth protocol, but still encountered connectivity issues. In an attempt to pair the Bangle.js with a mobile device, I followed the instructions provide by the Bangle.js Gadgetbridge mobile app (available at https://www.espruino.com/Gadgetbridge). The app is designed to enable smartwatch-style notifications and health monitoring without the need for a proprietary application or web service. Despite following the documentation diligently, I encountered difficulties in establishing a stable connection between the Bangle.js and the mobile device. The documentation that I have been following, for the Bangle.js Gadgetbridge app can be found at the following links: Official Gadgetbridge Documentation: https://gadgetbridge.org / Gadgetbridge GitHub Repository: https://github.com/Freeyourgadget/Gadgetbridge. The Gadgetbridge app offers various setup options to optimize the connection. I made sure to disconnect the Bangle.js from the computer and started the Gadgetbridge app on the mobile device. However, even with the recommended settings enabled, the connection remained unreliable. I also encountered issues with Gadgetbridge disconnecting from the Bangle.js intermittently, which could be related to battery optimization settings on the mobile device. I explored additional settings in Gadgetbridge to enhance the functionality. These settings include enabling automatic reconnection when Bluetooth is turned on, syncing time for calendar events, allowing internet access for Bangle.js apps, and granting necessary permissions in the Android device settings. Despite enabling these features, I was unable to establish a consistent connection between the Bangle.js and the mobile device. Furthermore, I attempted to use the Bangle.js Gadgetbridge app, specifically designed for Bangle.js with internet access capabilities. This version of the app can facilitate HTTP requests and interactions with Android apps. However, even with this specialized app, I encountered difficulties in establishing a stable connection and performing desired actions. In conclusion, despite testing the Bangle.js app on various devices and following the documentation, I encountered persistent connectivity issues. The Bluetooth connection with the PC did not work effectively, and the Gadgetbridge app on the mobile device failed to establish a stable and reliable connection with the Bangle.js. These challenges suggest potential compatibility or configuration issues that need further investigation and troubleshooting.

During my troubleshooting process to address the connectivity issue with the smartwatch, I followed a comprehensive set of steps provided by the manufacturer. I wanted to ensure that I covered all potential problems and solutions related to Bluetooth connection. To do this, I visited the dedicated support webpage on the manufacturer's website, which specifically addresses Bluetooth connectivity issues. The webpage, titled "Troubleshooting (Bluetooth LE)," provides an extensive list of potential problems and solutions related to Bluetooth connections. The webpage is at the following link http://www.espruino.com/Troubleshooting+BLE#getting-connected-- . I carefully went through each step mentioned on the webpage, paying close attention to the details and instructions provided. The troubleshooting guide covered various scenarios, including problems related to Web Bluetooth, compatibility issues with different operating systems, connection errors, device-specific concerns, and

more. One of the first issues addressed was the availability of Web Bluetooth in the Web IDE connection options. To resolve this problem, the guide suggested ensuring that I had an up-to-date version of Google Chrome (at least version 51) and a Bluetooth LE-capable adapter (at least Bluetooth 4.0) for my PC. Additionally, it provided specific requirements for different operating systems such as Android, Windows and macOS. Another step involved troubleshooting connection problems in Windows, where the guide mentioned that Bluetooth LE serial devices like my watch are not treated as serial port devices by Windows. Therefore, if the connection menu in the IDE showed devices beginning with the word "COM," they were not my device, and connecting to them would not work. Furthermore, I explored the troubleshooting steps for various other scenarios, such as not being able to see the device in the IDE, being unable to reconnect, or encountering difficulties connecting with Chrome. The guide recommended performing a series of actions, including resetting the device, restarting the computer, and clearing the browser cache. It also mentioned the possibility of conflicts with other Bluetooth devices and suggested temporarily disabling them to check if the issue persisted. It also mentioned the possibility of conflicts with other Bluetooth devices and suggested temporarily disabling them to check if the issue persisted. Lastly, I reviewed the troubleshooting steps for connecting with Chrome. The guide emphasized the importance of running the Chrome browser with the necessary privileges by launching it from the command line with the "--no-sandbox" flag. It also mentioned other potential issues related to browser permissions and advised checking the Chrome developer console for any error messages. Despite diligently following each step for troubleshooting provided on the manufacturer's webpage, the connection issue with my watch still persists. It seems that the problem lies beyond the scope of the troubleshooting guide's suggestions.

After exhausting the troubleshooting steps provided by the manufacturer's webpage, I expanded my research to understand the Bluetooth Low Energy (BLE) disconnections. The article titled "Understanding BLE Disconnections" from Argenox provides in-depth insights into debugging and investigating BLE connections, particularly disconnections. The article can be accessed at the following link: https://www.argenox.com/blog/understanding-ble-disconnections/ . The article highlights the significance of connections in BLE and emphasizes the need for debugging and comprehending connections to ensure a reliable and low-power product. However, it acknowledges that the Bluetooth Low Energy protocol itself can be challenging to debug due to its wireless nature and the way the protocol is defined. When it comes to BLE disconnections, the article explains that there are specific error codes that can indicate the reason for a terminated connection. The LL_TERMINATE_IND packets, which contain the error code, are crucial to analyse when sniffing the connection or examining HCI captures. The article provides a list of common error codes such as authentication failure, remote user termination, connection termination by the local host, low resources, power off, unsupported remote feature, pairing with unit key not supported, and unacceptable connection parameters. The author notes that 0x13 and 0x16 are the most common error codes observed in disconnection scenarios. While these codes may not provide specific reasons, they often appear due to the protocol's specification requirements. However, some devices or product developers may provide more specific error codes to indicate the cause of disconnection. Furthermore, the article explores intentional disconnections initiated by either the central or peripheral device through the HCI_Disconnect command. It discusses potential reasons for intentional disconnections, such as failed connection parameter negotiation or insufficient channels for connection. It also mentions that disconnections can occur when devices detect that the connection has dropped, often due to the expiration of the supervisor timeout. In terms of debugging BLE devices, the article highlights the importance of a well-designed hardware setup, considering factors such as crystal accuracy, interference from other devices in the 2.4GHz band, signal strength, sensitivity, and output power. It advises starting with a good design and conducting proper validation. To debug BLE devices effectively,

the article suggests using a wireless sniffer and capturing HCI messages between the BLE host and controller. It also mentions the usefulness of utilities like BlueNox BLE Scan to assess signal levels and UART output for embedded devices. After thoroughly studying the "Understanding BLE Disconnections" article from Argenox, I proceeded to implement the recommended steps to troubleshoot the connection issue with my Bangle.js watch. Here is a detailed account of each step I followed: Addressing interference: I took several measures to mitigate potential interference in the 2.4GHz band. I tested the Bangle.js watch in environments with minimal interference, such as remote locations or by temporarily disabling nearby Bluetooth and Wi-Fi devices. Despite these efforts, the connection issue persisted. Checking signal strength: To ensure sufficient signal strength, I verified that the Bangle.js watch received an acceptable signal level. Using tools like BlueNox BLE Scan, I measured the signal strength of advertising packets from the watch. The signal strength appeared to be within acceptable ranges (-85dBm or better), indicating that signal strength was not the primary factor contributing to the connection problem. Therefore, it was unlikely that these parameters were responsible for the connection issue. Despite diligently following each step recommended in the article, I did not find any significant issues. In addition to the steps mentioned earlier, it's important to note that I did not open the Bangle.js watch to inspect its internal hardware components. I wanted to exhaust all possible troubleshooting options without tampering with the device's internals. While opening the watch could potentially reveal any underlying hardware problems, I preferred to explore software and configuration-related solutions before resorting to hardware examination. By relying solely on external analysis and following the recommended steps from the Argenox article, I aimed to identify any issues that could be resolved without physically accessing the watch's internal components. However, since the connection problem persisted despite these efforts, I may need to consider inspecting the hardware components of the Bangle.js watch as a next step in the troubleshooting process. It's important to approach hardware examination with caution, especially if the device is under warranty or if opening it may void any existing warranty or support agreements. Therefore, I have carefully weighed the potential benefits of inspecting the hardware against any associated risks or limitations and I have made the decision not to proceed with inspecting the device's hardware at this time.

Moreover, I decided to switch libraries for the Bluetooth connection in my code. I opted to use the InTheHand.Net.Bluetooth library as an alternative to the windows library. I integrated the InTheHand.Net.Bluetooth and InTheHand.Net.Sockets, namespaces into my code, hoping that this change would help resolve the connectivity issue with the Bangle.js watch. After switching to the InTheHand.Net.Bluetooth library, I proceeded to follow the same steps outlined in my previous attempts to debug the application. Despite my efforts and the library switch, I encountered the same result. The Bluetooth connection remained unreliable, and I was unable to establish a consistent and stable communication channel with the Bangle.js watch. It became apparent that the issue may not be specific to the library being used, but rather related to other factors.

After exhausting all available troubleshooting options and not finding a resolution to the connectivity issue with my Bangle.js watch, I decided to reach out to the manufacturer for further assistance. I contacted them via email, providing detailed information about the problem I was experiencing and the steps I had already taken to troubleshoot it. Additionally, I posted a question on the official Bangle.js forum (https://forum.espruino.com/conversations/387126/) seeking guidance from the community and fellow users. While waiting for a response, I received feedback from other forum users who attempted to help me troubleshoot the issue. However, their suggestions and recommendations did not significantly differ from the steps I had already followed based on the research that I have made and other reliable sources. Despite the collective efforts and insights shared by the community, the root cause of the connectivity problem remained elusive.

In conclusion, after conducting extensive research, following recommended troubleshooting steps, exploring alternative libraries, and engaging with the manufacturer and the online community, the persistent connectivity issue with the Bangle.js watch remains unresolved, requiring further investigation. Considering the information gathered throughout this process, it is reasonable to suspect that the problem could be attributed to either a hardware-related issue or the inherent instability of Bluetooth Low Energy (BLE) connections. While I exercised caution in not opening the watch to inspect its hardware, the possibility of a hardware malfunction cannot be entirely ruled out. There might be internal components or antenna-related problems that are impacting the stability of the Bluetooth connection. To definitively determine if hardware is the culprit, a thorough examination by a qualified technician may be necessary. On the other hand, the inherent instability of BLE connections should also be taken into account. Despite efforts to implement different libraries and adhere to best practices for BLE development, intermittent connectivity issues are not uncommon. Bluetooth technology, particularly BLE, can be influenced by various environmental factors, interference, or protocol limitations, which can lead to sporadic disconnections.

4. TECHNOLOGIES AND EQUIPMENT

During the project, I utilized various technologies, equipment, software, and tools to accomplish the objectives. These resources were instrumental in developing the functionality of the Blexer middleware and integrating it with the Bangle.js watch. Additionally, I made use of specific algorithms and libraries to handle data retrieval and transmission. Here's an overview of the technologies and equipment employed in the project:

Bangle.js Watch:

The Bangle.js watch played a central role in the project. I initially chose this watch due to its compatibility with writing code in C#. However, upon further investigation, it was clarified that the watch is primarily programmed using JavaScript. The Bangle.js watch features a heart rate sensor that captures real-time heart rate data. This sensor was essential for achieving the project's objectives, particularly in the context of the game. The heart rate capturing function works by leveraging the watch's built-in heart rate sensor. JavaScript code is written to enable the heart rate monitoring functionality on the watch. The code registers an event listener for heart rate updates and retrieves the heart rate data whenever a new reading is available. The heart rate data is then formatted as a stream of values, including the heart rate measurement and confidence level. To transmit the heart rate data from the Bangle.js watch to the PC, a Bluetooth connection is established between the watch and the computer. In the code, the Windows-specific libraries Windows.Devices.Bluetooth and Windows.Devices.Bluetooth.GenericAttributeProfile are utilized to communicate with the watch over Bluetooth. The data is transmitted from the watch to the PC as a string of comma-separated values, where each value represents a different aspect of the heart rate reading, such as the heart rate measurement and confidence level. Once the heart rate data is received on the PC, it can be further processed and utilized in the game. The data is transmitted from the Bangle.js watch to the PC, and can be transformed in JSON format, which provides a structured representation of the heart rate readings and associated information. This JSON data can then be parsed and extracted on the PC side to obtain the heart rate measurement and confidence level.

Bluetooth Low Energy (BLE):

To establish a connection between the Bangle.js watch and the middleware, I utilized Bluetooth Low

Energy (BLE) technology. BLE provides a power-efficient wireless communication protocol for devices such as watches, smartphones, and other IoT devices. It allows for data transmission over short distances, making it ideal for wearable devices. The BLE technology facilitated the communication between the watch and the middleware, enabling the retrieval and transmission of heart rate data. I relied on the Windows.Devices.Bluetooth.GenericAttributeProfile namespace, which is an integral part of the Windows Runtime (WinRT) API provided by the Windows operating system. This namespace offers a comprehensive set of classes and functions specifically designed to interact with Bluetooth devices using the Generic Attribute Profile (GATT). To begin with, I utilized the BluetoothLEDevice class, which allowed me to represent a Bluetooth Low Energy (BLE) device and initiate a connection with a specific device based on its Bluetooth address. By calling the BluetoothLEDevice.FromBluetoothAddressAsync method and providing the MAC address of the Bangle.js watch, I should be able to establish a connection to the desired device. Once the connection was established, I proceeded to retrieve the available GATT services on the Bangle.js watch using the GattDeviceService class. By invoking the GetGattServicesAsync method on the BluetoothLEDevice instance, I obtained a GattDeviceServicesResult object that contained information about the retrieved services. To ensure the successful retrieval of GATT services, I checked the Status property of the GattDeviceServicesResult object. By comparing it with the GattCommunicationStatus.Success enumeration value, I could determine if the services were successfully obtained. If the status indicated success and the Services collection of the GattDeviceServicesResult object contained at least one service, I could proceed with further operations. Within the GATT services, I needed to find the UART (Universal Asynchronous Receiver-Transmitter) service, which facilitates serial communication. To accomplish this, I utilized the GattCharacteristic class and its FirstOrDefault method on the Services collection. By specifying the UUID (Universally Unique Identifier) of the UART service as Guid.Parse("6E400001-B5A3-F393-E0A9-E50E24DCCA9E"), I could obtain the desired GattDeviceService object representing the UART service on the Bangle.js watch. By leveraging these Windows.Devices.Bluetooth.GenericAttributeProfile components and functions, I have tried to established a connection between the Bangle.js watch and the middleware. It's important to note that the Windows.Devices.Bluetooth.GenericAttributeProfile namespace and its associated classes and functions are specific to the Windows platform. These APIs are part of the Windows Runtime (WinRT) API, providing developers with a standardized and efficient way to interact with Bluetooth LE devices. By utilizing these APIs, I could seamlessly integrate Bluetooth functionality into my application.

C# Programming Language:

I leveraged the C# programming language to develop the functionality within the Blexer middleware. C# is a versatile and powerful programming language widely used for developing various applications, including desktop and web applications. It provided me with a familiar and efficient environment to write the code for the new function. With C#, I was able to implement the necessary logic for retrieving the heart rate data from the Bangle.js watch and transmitting it to the game.

Blexer Middleware:

The Blexer middleware served as the backbone of the project. It provided the infrastructure and functionality to facilitate the communication between the game and the Bangle.js watch. By understanding the architecture and functionality of the Blexer middleware, I was able to identify the areas where the new function could be integrated. I extended the middleware by adding the necessary code to retrieve and transmit the heart rate data. The middleware acted as an intermediary layer between the game and the

watch, ensuring seamless data flow.

Software Libraries and APIs:

To facilitate the implementation of the project, I made use of several software libraries and APIs. One notable library is Newtonsoft.Json, which provided robust support for parsing and serializing JSON data. This library was utilized to handle the data received from the watch and convert it into a usable format within the middleware. Additionally, the Windows.Devices.Bluetooth and Windows.Devices.Bluetooth.GenericAttributeProfile namespaces provided the necessary classes and methods to interact with the Bluetooth LE device.

In conclusion, the project involved the utilization of the Bangle.js watch, BLE technology, the C# programming language, the Blexer middleware, and various software libraries and APIs. These technologies and equipment were instrumental in achieving the project's objectives of retrieving and transmitting heart rate data. Despite encountering challenges related to connectivity, I utilized the available resources effectively to implement the necessary algorithms and functionality. Through this project, I gained hands-on experience with these technologies and equipment, further enhancing my skills in software development and IoT integration.

5. DEVELOPED COMPETENCES AND SKILLS

Throughout the course of this project, I have had the opportunity to develop and enhance several key competences and skills. The tasks I performed not only allowed me to apply the theoretical knowledge acquired during my studies but also provided me with practical experience in various areas. The following sections outline the competences and skills that I have developed throughout this project:

Technical Proficiency: This project required a solid understanding of programming languages, specifically JavaScript and C#. By working with the Bangle.js watch and the middleware, I gained proficiency in writing code to capture real-time heart rate data, establish a connection via Bluetooth Low Energy (BLE), and transmit data to a computer. These technical skills have greatly enhanced my programming abilities and prepared me for future projects involving IoT devices and wearable technology.

Problem Solving: Throughout the project, I encountered various challenges, such as connectivity issues and data retrieval inconsistencies. To overcome these obstacles, I employed problem-solving techniques, including debugging, troubleshooting, and researching potential solutions. This experience strengthened my ability to analyze problems, think critically, and devise effective strategies to resolve technical issues. I learned the importance of perseverance and adaptability when faced with complex problems, and this skillset will undoubtedly prove invaluable in my future endeavors.

Communication and Collaboration: Collaboration played a significant role in this project, as I worked closely with team members and sought guidance from my professor. Clear and effective communication was essential for coordinating tasks, sharing progress updates, and discussing potential solutions. Through regular meetings, group discussions, and presentations, I honed my ability to convey complex technical concepts in a concise and understandable manner. This experience improved my interpersonal skills, fostering effective teamwork and collaboration.

Project Management: I learned how to set clear objectives, create a timeline, allocate resources, and track progress. I also gained valuable experience in prioritizing tasks, managing deadlines, and adapting to changing requirements. This project helped me understand the importance of proper planning and organization in achieving successful outcomes.

Research and Continuous Learning: Throughout the project, I engaged in extensive research to explore existing technologies, understand the functionality of the Bangle.js watch, and investigate best

practices for integrating heart rate data into gameplay. This research-oriented approach enhanced my ability to gather relevant information, analyze different perspectives, and make informed decisions. Additionally, I cultivated a habit of continuous learning by staying updated with advancements in wearable technology, IoT devices, ensuring that my knowledge remains relevant and up to date.

6. CONCLUSIONS

The completion of this project has significantly contributed to my training and personal growth. Through the guidance and support of my supervisor, I have been able to overcome challenges, acquire new knowledge and skills, and gain valuable insights into the field of software development. In this section, I will provide an evaluation of how the project has contributed to my training and the influential role of my supervisor in this process. First and foremost, this project has provided me with a practical application of the theoretical concepts and principles I learned during my studies. It served as a bridge between classroom learning and real-world implementation. By working on a project that integrated real-time heart rate data into gameplay, I gained hands-on experience in programming and data processing. This project not only solidified my technical skills but also improved my problem-solving abilities, critical thinking, and creativity. The guidance and expertise of my supervisor played a crucial role in this project. Their extensive knowledge and experience in the field of software development provided valuable insights and direction throughout the project lifecycle. From the initial planning stages to the final implementation, my supervisor provided continuous support, offering valuable advice, suggesting alternative approaches. Their feedback and constructive criticism helped me refine my ideas, improve my coding practices, and enhance the overall quality of the project. The supervisor encouraged me to conduct in-depth research, stay updated with the latest advancements, and explore innovative techniques and technologies. The collaboration with my supervisor also enhanced my communication and interpersonal skills. Regular meetings, progress updates, and discussions fostered effective communication and allowed for valuable knowledge exchange. The feedback provided by my supervisor helped me refine my presentation skills, enabling me to effectively convey complex technical concepts to both technical and non-technical stakeholders. Additionally, the collaborative environment provided a platform for exchanging ideas, sharing experiences, and learning from each other's expertise. In conclusion, the completion of this project has been an enriching experience that has significantly contributed to my training as a student. The practical application of theoretical knowledge, the development of technical skills, and the cultivation of problem-solving abilities have prepared me for future challenges in the field of software development. The mentorship has not only shaped my technical skills but also fostered a mindset of continuous learning, effective communication, and collaboration. I am grateful for the opportunity to work on this project and for the invaluable guidance provided by my supervisor, which has greatly influenced my personal and professional development.

7. PROJECT LOG

Week 1 (15.03 - 21.03):

In the initial week of the project, I focused on conducting extensive research on available smartwatches and selecting the most suitable option for the development of the new function. After evaluating various options, I chose the Bangle.js smartwatch. Additionally, I studied the middleware, technologies, and protocols relevant to the development of the smartwatch app. This included familiarizing myself with the JSON format for data exchange. This research phase laid the foundation for the subsequent development work.

Week 2 (22.03 - 28.03):

In the second week, I dedicated my time to an in-depth study of the middleware used. I have thoroughly examined all the functions provided by the middleware and analyzed their relevance to the new function requirements. Through collaborative discussions, I have determined the ideal placement of the new function within the existing middleware structure.

Additionally, I focused on further exploring the JSON format and its implementation for seamless data transmission between the app and the smartwatch. By gaining a comprehensive understanding of this format, I ensured that the data exchange process would be efficient and compatible with the chosen middleware.

Week 3 (29.03 - 04.04):

During this week, while awaiting the arrival of the smartwatch, I took the opportunity to enhance my familiarity with the C# programming language. I dedicated time to review documentation and online resources, deepening my understanding of its syntax, features, and best practices.

By immersing myself in C# documentation, I familiarized myself with the language's core concepts, data types, control structures, and object-oriented programming principles. I also explored C# libraries and frameworks relevant to the development of smartwatch applications, gaining insights into their usage and potential integration with the upcoming project.

Week 4-6 (05.04 - 25.04):

In the first week of the development phase, I familiarized myself with the Bangle.js smartwatch by creating small watch apps. This helped me understand its capabilities and ecosystem.

Afterward, I started writing the code for the new function for the Bangle.js. I focused on designing the logic, implementing algorithms, and ensuring compatibility with the existing codebase.

Throughout the development process, I did testing and debugging of the function, to improve the code.

Week 7-9 (26.04 - 16.05):

During these weeks, a significant portion of my time was dedicated to troubleshooting and resolving connectivity issues related to the Bangle.js smartwatch. I have tried to identify the root cause of the intermittent connectivity problems and worked on finding solutions. I explored different strategies and approaches, aiming to establish a stable and reliable connection between the smartwatch and the new function. Throughout this process, I meticulously documented the troubleshooting steps and observations, with a particular focus on documenting the Bluetooth Low Energy (BLE) connection. Although a definitive solution remained elusive, the documentation serves as a valuable resource for future investigation and potential improvements.

Week 10-12 (17.05 - 14.06):

During the final three weeks of the project, my primary focus was on continuing the troubleshooting efforts to resolve the connectivity issues with the Bangle.js smartwatch. I remained committed to identifying the underlying cause of the intermittent connectivity problems and explored additional approaches to rectify the issue.