



SE 101 Introduction to Methods of Software Engineering

Fall 2022

Course Project Final Report

4FUN

4 GAINS

Member Name	userID	Username
Norman Chen	20990817	njchen@uwaterloo.ca
Emma Huang	20997368	e8huang@uwaterloo.ca
Benjamin Ng	20996253	bymng@uwaterloo.ca
Peter Wang	21024648	p25wang@uwaterloo.ca
Alex Zhu	21001761	a3zhu@uwaterloo.ca

Introduction

The reopening of public gyms has attracted an influx of fitness lovers as the world gradually recovers from the effects of the pandemic. In particular, we saw a massive wave of fitness craze among our peers while transitioning into university as workout sessions became integral to many students' daily routines. As frequent gym-goers, we were inspired by this phenomenon to create "4Gains"—a product that targets the growing fitness demographic and improves the gym experience for all fitness lovers, beginners, and pro athletes alike.

Easily mounted onto any weight stack, the 4Gains unit provides immediate feedback to the user on the quality of their workout sessions through its various sensors. Upon a button press on the device, the program begins to track a variety of metrics, including session duration, number of reps performed, movement distance, acceleration, and stability. The program then offers the user a visual representation of their fitness progress, which they can access through a web application. We envision 4Gains as a tool that makes fitness more accessible to novice gym-goers while providing valuable personalized suggestions to veterans. Demo video: [Introducing 4GAINS | 4FUN - 2022 SE101 Project.mp4](#)

Background Research

Hardware

We found out it was not possible to store data that the Arduino sent to the Serial port unless you used PuTTY, which we did not think was viable, as it is difficult to run an instance of PuTTY on demand. Instead, we decided to send the data to a local server at a set IP address, and then scrape that. Another option would have been to use a remote server hosting service that ran instances of code 24/7 and waited for an update, but it would have been more difficult to set up, less debuggable if something went wrong because it wouldn't be our code, and expensive in the long run.

Backend

When researching what frameworks to use for the API, we considered using Express.js or Flask. After researching to compare both, besides the differences in the programming language, we found that the main distinction between the two is that Express is faster due to the v8 engine but was more complicated to set up. After evaluating what the features of the API would have to be, we realized that the data being sent through the API would be relatively small if we leveraged lists and that the number of user requests would be very small for this project. Additionally, as we had already decided to use Python for the data parsing and filtering due to the greater number of open-source libraries for data analysis, we decided upon using Flask so that the language would be consistent for the backend to improve the readability of the code.

Frontend

When designing 4Gains, we envisioned an interactive frontend application that will let users access their statistics from previous workouts on a mobile device. Upon research, we decided to use the React Javascript library to develop a web application for our front end. It integrates reusable UI components, which reduces redundant code and speeds up our development process. In addition, React is ideal for applications that require a high level of user interaction, as it supports stateful components and takes over the management of low-level algorithms. Our application requires all displayed graphs to update automatically with new data after each set is completed by the user. React implements a virtual DOM in memory that allows the program to re-render only the components which require updating. This system increases the speed of reloads so that the user can see details and statistics about their workout faster after each session. The distance, acceleration, stability, reps, and session length data are then stored as states within the graph components, which automatically re-render once new data is pushed to the database.

To style the website, TailwindCSS was chosen because of the ease of styling components. In particular, the in-line styling of Tailwind is closer to plain English. Additionally, the ability to style components for different display sizes using grids allowed us to make the app responsive for tablet, phone, or computer. Although the customizability of conventional CSS is better than Tailwind, for the purpose of this app we prioritized the appearance of the website and ease of learning.

We looked into several JavaScript libraries for the graph display and ultimately selected the Victory library for its interactivity and ability to work dynamically with updating state variables. Charts created using Victory can integrate smoothly with React's re-rendering of stateful components to automatically generate updated graphs when new data points become available.

Implementation

The 4Gains personal trainer comprises three key modules: a hardware component that consists of sensors collecting data from the user's workouts, backend programs and scripts that retrieve process, and store the collected data, and a frontend React web application that displays the processed data through graphical representations. In addition, the custom APIs we have designed and implemented allow these components to work together seamlessly.

First and foremost, the hardware component consists of an Arduino Uno connected to an HC-SR04 ultrasonic distance sensor, an ESP8266 Wi-Fi module, a LED-light indicator, and a digital button through a BB830 solderless breadboard. The ultrasonic sensor collects distance over time data in intervals of 100-milliseconds and the collected data is first processed locally to count the

number of repetitions completed during the set. The number of repetitions and the distance data is then concatenated into a singular string and sent to a web server on the local network through the Wi-Fi module and concluding the hardware component.

The string sent to the local web server is then parsed by the python script `ParseGraphForDistance.py`, which scrapes the HTML web page for distance values over time in an integer variable length array by utilizing the BeautifulSoup python library. It reads the specified text from the HTML page and stores it in a list, which gets uploaded to the MongoDB distributed database which is then fetched through a custom API by the `dataProcessing.py` python file. The file iterates through the parsed list and implements data smoothing, removing any outlier points and reducing the overall noise of the dataset through an anomaly recognition algorithm. The data is then iterated through again and the velocity, acceleration, and stability over time datasets are determined by taking derivatives. Additionally, the distance data is parsed to count “good” reps. The clean distance, velocity, acceleration, stability, and rep data is reuploaded to the MongoDB database using dynamic paths in Flask to save each set of data in its respective table, able to be fetched by the front-end component at any instance to display for the user.

Finally, once the data parsing and processing have been completed by the python scripts and the backend components, the front-end module fetches the final dataset from the MongoDB database through another API and displayed it to the user on a React-based web application hosted on one of our group member’s personal domains. The distance, acceleration, and stability data are displayed graphically to the users through the Victory wrapper component, and the web application is made much more visually appealing to the user through Tailwind.css.

Hardware

4Gains unit comprises an Arduino Uno, an ultrasonic distance sensor, an LED light indicator, and a wi-fi module connected through a BB830 solderless breadboard (See Fig. 1).

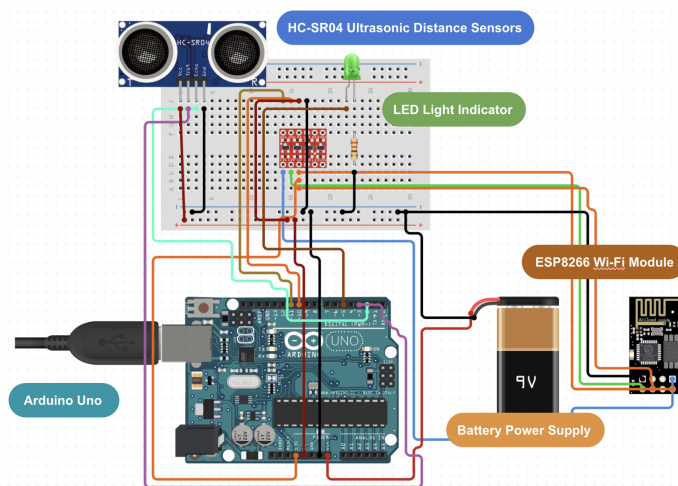


Fig.1. Structural Diagram of the 4Gains Hardware Unit

Arduino Uno

The core of the 4Gains unit is an Arduino Uno, which controls all the connected hardware components and runs on a wireless battery power supply. The Arduino Uno is programmed through C++ and implements several functionalities:

1. Establishes Wi-Fi connection through the module and invokes data transfer to a local network
2. Interprets echo data provided by the ultrasonic distance sensors into displacement from the unit’s initial position. If it identifies a displacement exceeding a given range, a rep is counted.
3. Detects the completion of a set by measuring idle time. If the unit remains relatively stationary for over 2.5 seconds, the set is marked as complete.
4. Reflects the start and competition of sets and reps by flashing the LED light indicator.

HC-SR04 Ultrasonic Distance Sensors

The HC-SR04 sensor module consists of an ultrasonic transmitter and a receiver. From the top of the weight stack, the transmitter sends an ultrasonic signal upwards, until it is deflected back by the ceiling to the receiver. The sensor records the time between sound echoes and uses the formula $distance = \frac{time * speed\ of\ sound}{2}$ to determine the distance between the unit and the ceiling. Repeating this process in 100-millisecond intervals, the HC-SR04 module tracks the unit’s displacement continuously throughout the session, which the Arduino program interprets into an array of heights representing the user’s movement during the workout.

LED Light Indicator

The LED light indicator serves the purpose of giving the user feedback and also for debugging the Arduino code when it is not connected to the serial terminal on a laptop. The LED light blinks after each step of calibration, such as when it connects to the Wi-Fi, when a set/rep has been completed, and when the data has been successfully sent to the server.

ESP8266 Wi-Fi Module

The ESP8266 Wi-Fi MCU grants the Arduino Uno the ability to connect and establish a web page on the local Wi-Fi network. The Arduino code takes in the Service Set Identifier (SSID) of the network and the password and establishes the connection through a series of ATtention commands (See Fig.2).

Establishing Wi-Fi Connection

```
189 void wifi_init() //send AT commands to module
190 {
191     establishConnection("AT", 100); // Tests the AT start up
192     digitalWrite(ledPin, HIGH);
193     delay(125);
194     digitalWrite(ledPin, LOW);
195     delay(125);
196     establishConnection("AT+QMODE=3", 100); // Sets the Wi-Fi mode of operation to Station+SoftAP
197     digitalWrite(ledPin, HIGH);
198     delay(125);
199     digitalWrite(ledPin, LOW);
200     delay(125);
201     delay(1000);
202     establishConnection("AT+QIAP", 100); // Disconnects the ESP8266 from an Access Point
203     digitalWrite(ledPin, HIGH);
204     delay(125);
205     digitalWrite(ledPin, LOW);
206     delay(125);
207     delay(1000);
208     establishConnection("AT+RST", 5000); // Restarts the Wi-Fi module
209     digitalWrite(ledPin, HIGH);
210     delay(125);
211     digitalWrite(ledPin, LOW);
212     delay(125);
213     delay(1000);
214     findIp(5000); // A helper function that determines the IP address of a connected local network
215     if (!No_IP) {
216         Serial.println("Connecting Wi-Fi...");
217         establishConnection("AT+QIAP=" + SSID + "\",\"\" + PIN + "\",\"\", 7000); // Takes in Wi-Fi username and password
218         // The QIAP command connects the module to an Access Point
219     } else {
220     }
221     Serial.println("Wi-Fi Connected");
222     showIP();
223     establishConnection("AT+CIPMUX=1", 100); // Enables multiple TCP Connections.
224     establishConnection("AT+CIPSERVER=1,80", 100); // Creates a TCP Server
225 }
226
227
```

Sending Data to the Server

```
228 void sendData(String server1) //send data to module
229 {
230     int p = 0;
231     while (1) {
232         unsigned int l = server1.length();
233         // Serial.print("AT+CIPSEND=0,");
234         comm.print("AT+CIPSEND=0,");
235         // Serial.println(l + 2);
236         comm.println(l + 2);
237         delay(100);
238         // Serial.println(server1);
239         comm.println(server1);
240         while (comm.available()) {
241             //Serial.print(Serial.read());
242             if (comm.find("OK")) {
243                 p = 11;
244                 break;
245             }
246         }
247         if (p == 11)
248             break;
249         delay(100);
250     }
251 }
```

Fig.2. Code snippets of the establish Wi-Fi connection function

In the main loop of the Arduino code, a string is continuously updated by appending displacement values from the ultrasonic sensors. Once the entire set has been completed, the post data is pushed to the local server; available to be parsed by any device on the local network.

Data Processing

All the data processing occurs in the dataProcessing.py file. The function of this file is to process the data from the Arduino to displayable data on the front-end. The program is given a list of distance values that are 0.1s apart from each other. During testing, we found that there are occasionally anomalous points in the distance data (e.g. 48, **1800**, 48); the function *filterData* will iterate through the list and will delete such anomalous points. A *movingAverage* function was also created to filter out additional anomalous points by comparing 2 sets of the average of 3 consecutive points, and deleting the outlier point if it is greater than a predefined threshold.

After the data has been sieved by the *filterData* function, a *derivative* function was created to take the derivative of discrete data points and used three times to find the first, second, and third derivatives. Finally, the *goodReps* function was made, which counts a rep based on the distance change between the local minimums and maximums and if it is above a certain threshold value. In the end, the lists distanceX, distanceY, velocityX, velocityY, accelX, accelY, thirdDerX, thirdDerY, and variable goodReps are returned to the database. As a proof of concept for the derivative function, the first 13 integer data points of the function $y = x^2$ were inputted into the function and the following graphs outputted attest to the accuracy of the derivative function (See Fig.3).

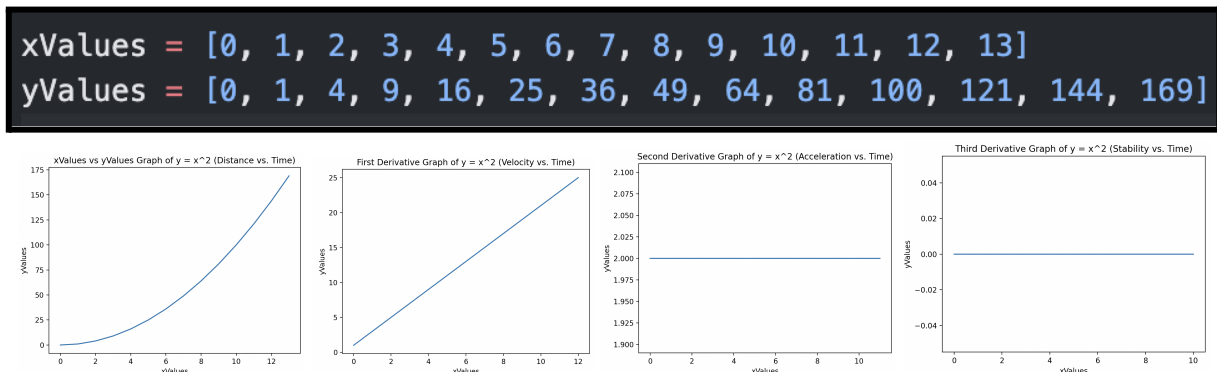


Fig.3. Input x, y values and outputted graph for the first second, and third derivative

Backend

API: Fetching & Posting Data

When the API fetches data from the MongoDB Atlas database, it converts the data into a JSON format which can be accessed by a URL (See Fig.4).

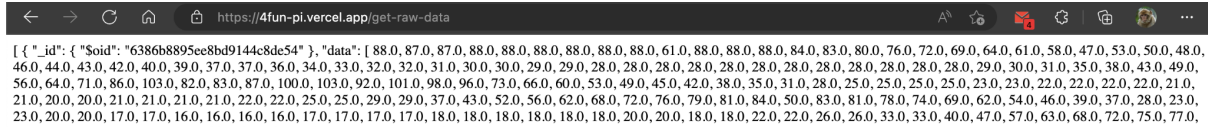


Fig.4. Raw data format stored in a .json file

One of the problems that we ran into while building the API was that only a single array could be stored into an item on MongoDB. This was a problem because we had several sets of filtered distance, velocity, acceleration, and consistency data which all corresponded to a single set of raw data. To work around this challenge, we utilized dynamic paths in Flask which allowed us to POST and GET data to a particular collection of the database by changing the last part of the path (See Fig.5).

```
@app.route('/push-data/<dataType>', methods=['POST'])
def sendFilteredData(dataType):
```

Fig.5. Code snippets of the dynamic path reallocation in Flask

Utilising this in the data processing file, we were able to simplify our code by creating a function which takes the parameters of the array and name to send to the database (See Fig.6).

```
def postData(array, dataType):
    url = f'https://4fun-pi.vercel.app/push-data/{dataType}'
    item = {"data": array}
    x = requests.post(url, json=item)
```

Fig.6. Code snippets of the data sending function to MongoDB

Frontend

The frontend of 4Gains is developed primarily in JavaScript using the React Library. The Victory library is integrated for the creation of interactive graph displays, and the Tailwind CSS library is used for styling the user interface of the application. The application is hosted on a domain and can be accessed anywhere by the user through their mobile devices. App: <https://4fun.benjaminng.ca/>

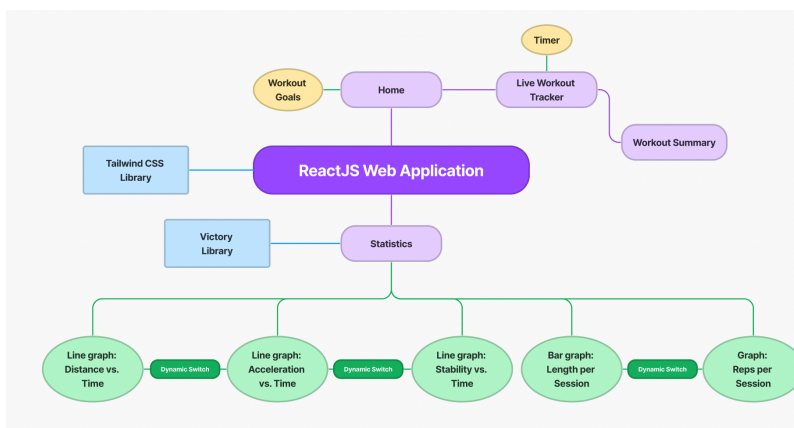


Fig.7. Block diagram of our front-end interface

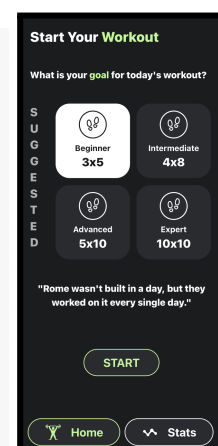


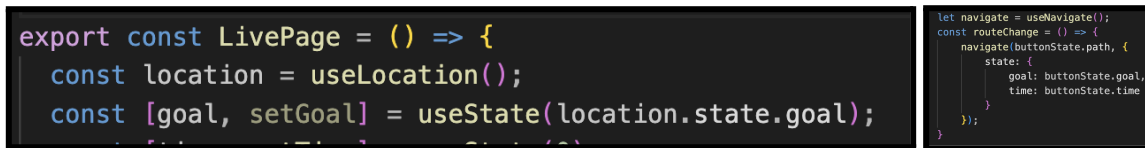
Fig.8. Home page

We designed the UI of the application in [Figma](#), and then recreated the design using CSS and the Tailwind CSS library. Tailwind CSS replaces the need for elaborate CSS stylesheets by offering in-line shorthand properties, which we then customized to fit our theme through tailwind.config.js.

At the center of the home page, we created a *Suggested* panel that prompts the user to select a goal for their workout session among four preset options implemented through the *Goal* component. Once the user presses on a goal, the *Goal* component calls a function provided by *Suggested.js* and passes its unique ID to the parent file. The selected ID is then passed to

and maintained as a state in *Home.js* and updated whenever the user changes their selection. Underneath the *Suggested* panel is a strip containing a motivational message that is randomly generated from a list and refreshed every 3 seconds. Through the goal feature and motivational message, we hope to encourage the user in challenging themselves, before their workout session begins.

All the pages of the application are implemented as navigable routes. Once the “Start” button is pressed, the *Live Workout* page with an animated timer appears, which can be controlled by the “Pause/Resume” button below. The timer updates the “time” state in one-second intervals and running it through a timeFormatter helper function to achieve a HH:MM:SS display. Above the timer, the user can view their selected goal for the session, which has been passed from the *Home* page as a navigation state and then retrieved through the *useLocation* function (see code snippets below).



```
export const LivePage = () => {
  const location = useLocation();
  const [goal, setGoal] = useState(location.state.goal);
  // ...
}

let navigate = useNavigate();
const routeChange = () => {
  navigate(buttonState.path, {
    state: {
      goal: buttonState.goal,
      time: buttonState.time
    }
  });
};
```

Fig.9. Using navigation states and location to pass state variables across pages

When the “Exit” button is pressed, the user is taken to the *Exit* page where they are presented with a quick summary of their session, including the goal they had selected and the length of the workout. This page is also responsible for sending the timestamp and length of the user’s workout session to the MongoDB database. This transfer of data is achieved through invoking the post method of the API into the workout-summary database:

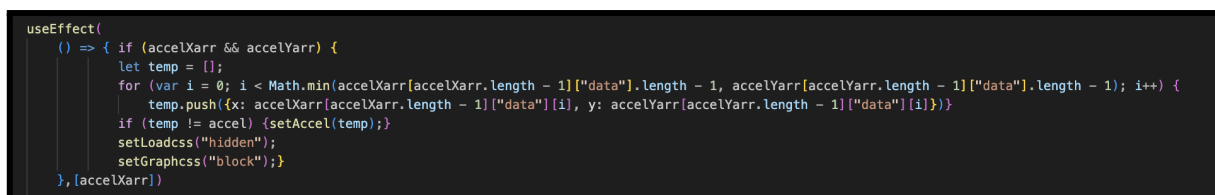


```
useEffect(() => {
  if (!ran) {
    const requestOptions = {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        "workoutLength": time,
        "timestamp": Date.now()
      })
    };
    fetch('https://4fun-pi.vercel.app/send-workout-summary', requestOptions)
      .then(response => response.json())
      .then(data => setPostId(data.id))
      .catch(error => {
        console.log(error);
      }).finally(setLoading(false));
    ran = true;
  }
}, [()])
```

Fig.10. Sending data through the API using the POST method

Fig. 11. Stats page

The second main feature of the application is the *Statistics* page, consisting of three sections: a *GraphDisplay* component, a *BarDisplay* component, and a suggestion box. The *GraphDisplay* component generates three graphs, each representing the unit’s movement distance over time, acceleration over time, and stability over time throughout the session. Each graph fetches the corresponding array of data from the database through the fetch method of the API, and then parses the array into x and y values for each datapoint on the graph:



```
useEffect(
  () => { if (accelXarr && accelYarr) {
    let temp = [];
    for (var i = 0; i < Math.min(accelXarr[accelXarr.length - 1]["data"].length - 1, accelYarr[accelYarr.length - 1]["data"].length - 1); i++) {
      temp.push({x: accelXarr[accelXarr.length - 1]["data"][i], y: accelYarr[accelYarr.length - 1]["data"][i]});
    }
    if (temp !== accel) {setAccel(temp);}
    setLoadcss("hidden");
    setGraphcss("block");
  }, [accelXarr])
```

Fig.11. Parsing the fetched arrays of data into x and y values displayed on the graph

The data array is then passed into a *VictoryChart*, which produces a line graph through the *VictoryLine* component. The *VictoryChart* is then styled by customizing its labels, axes, and ticks so that the user can easily comprehend the presented information. The user can switch between the different graphs through the buttons below. This feature is handled by a *Switch* component that manages the states of each button and toggles between the three graphs accordingly. Similarly to *GraphDisplay*, the *BarDisplay* contains two bar graphs representing the length of the user’s recent workout sessions and the number of reps performed. Finally, the suggestion box provides an overall recommendation on whether the current workout attempted by the user is too difficult, too easy, or appropriate. This suggestion is based on the amount of variation in the acceleration, calculated by taking the standard deviation in the user’s stability throughout the session.

Group Members' Contribution

- **Alex:** market research at the gym, wiring the hardware, writing Arduino C code for wifi, accelerometer, LED, distance sensor, building the physical box, writing Python code for graphing for prototype presentation, parsing data from server and sending to database, storyboarding, directing and editing the final video
- **Emma:** creating UI design with Figma, writing Arduino C code for wifi calibration and accelerometer, developing frontend application with graphical displays and tracker using ReactJS and Tailwind CSS, design & painting of physical container
- **Ben:** Creation of API with flask, deploying frontend and backend on Netlify and Vercel, parsing data from arduino using BeautifulSoup, connecting frontend and backend algorithms to database, planning and filming video
- **Peter:** Implementation and integration of hardware components: ESP8266 Wi-Fi module, MPU6050 accelerometer, HC-SR04 ultrasonic distance sensor, LED light indicator and digital button. Development of Arduino.ino code for the aforementioned hardware components and wrote a repetition counting algorithm to determine the number of reps completed each set.
- **Norman:** Researching and implementing hardware: ESP8266 and the MPU6050 accelerometer. Writing data processing code in python (finding the derivative of a dataset, filtering of data of anomalous data, creating the moving average, and creating data that can be sent back to the database)

Alex Zhu Peter Wang Ben Ng Emma Shang Norman Chen

Design Trade-offs

As the SE101 design project was the first time that many of us had worked on a project involving both significant quantities of software and hardware components. The design process of our project saw over-optimism in our group's ability to physically construct and integrate the vision that we had in mind. It was not until our prototype was eventually completed that we realized we would have to make various design trade-offs in a number of modules in order to have the overall system work together without changing the original goal significantly.

Hardware

Initially, our group planned on integrating both the MPU6050 accelerometer and the HC-SR04 ultrasonic distance sensor in conjunction to collect as much data as possible to be processed by the backend component. However, due to the two sensors sharing different baud rates (speed of communication), our group decided to utilize only the ultrasonic distance sensor as it provided more consistent data and had significantly lower power usage compared to the accelerometer. Furthermore, the data sampling rate of the ultrasonic distance sensor had to be lowered to every 0.1 seconds in order to reduce the overcollection of data, which often clogged the already limited random access memory of the Arduino Uno. The sound sensor which we had originally planned to implement to detect the drop of metal plates to determine the end of a set eventually became outdated as its functionality was already included in that of the ultrasonic distance sensor.

The initial idea of utilizing Bluetooth to communicate the data between the Arduino Uno and the backend components was also quickly scrapped due to the limited physical range of Bluetooth communication and its unreliability in metal-containing environments. Moreover, market research on the price of these two sensors indicated that the Bluetooth module cost significantly more than the ESP8266 Wi-Fi module, further demonstrating its inability to compete with the Wi-Fi-based communication system.

Backend

The preliminary concept of data transfer between the Arduino Uno and the backend components was to collect the data directly from the Serial terminal of the Arduino; however, that was soon rendered impossible as background research indicated that it was impossible to export data from the Arduino Serial monitor. Consequently, our group decided to upload the concatenated string with the data values to a local web server to be parsed by the ParseGraphForDistance.py Python script utilizing the BeautifulSoup library.

The database portion of the backend module was originally set to be offline, on a local computer or mobile device. However, as we reduced the amount of data we are collecting and sharing by removing the accelerometer and sound sensors, the MongoDB Atlas made for a suitable cloud storage option, as it allows the python script to collect the data anywhere in the world and made the database hosting process much simpler. The wide range of open-source libraries that came with MongoDB, including PyMongo that we ended up using also made the API creation process easier.

Frontend

For the prototype, the graphical display of the data was achieved through the implementation of the D3.js data visualization Javascript library. However, due to its lower compatibility with the React Javascript library and its inability to adapt to live and changing data, the alternative of the Victory Javascript library was implemented instead. The Victory library allowed for automatic updates when new data is retrieved from the backend and also provided interactive graphs for the users.

Citing the Wi-Fi connection delays mentioned in the hardware trade-offs section, the inability to receive data from the sensors in real time rendered the concept of a live repetition counter impossible. Hence, the idea was replaced by a stopwatch which helps the users keep track of the length of their individual sets as well as the overall workout.

Future Work

On the backend, one of the primary improvements that could be made is automating the process of data transfer between the Arduino and the backend components which would allow for real-time suggestions for the user during their workout. With our current implementation of the backend, the data is only being processed at the end of the workout by utilizing an array of data that is received from the Arduino which prevents the users from seeing the number of reps that they have done in real-time or the consistency of their reps. Although the current method of sending data is much faster and requires less computational power, it reduces user experience as they would not be able to receive real-time feedback.

Another feature that we hope would come in a later update of the *4Gains* unit would be the implementation of personalized workout feedback. In the current version of our software, we simply visualize the collected physical data for the users and do not provide any qualitative or quantitative feedback that would be beneficial to help users improve. In the future, we would like to build personal artificial intelligence models based on individual users' physical attributes, such as height and weight as well as their experience competing in certain sports. These models would allow users to set goals for themselves that the *4Gains* unit would interpret and help the user set workout plans to achieve their goals in the shortest time possible.

For the hardware components, we would like to see improvements in the computational power of the Arduino Uno to support faster data transfer and more connected sensors. Furthermore, the ESP32 Wi-Fi module was an option that was considered to improve the stability and reliability of the Wi-Fi connection, which would support the greater flow of data per unit of time and the faster setup process that comes before each workout. Finally, the mounting system of the box would be redesigned and printed using a 3D printer, allowing it to be customizable to different workout machines and mountable onto free weights.

Final Product Evaluation

The group members unanimously agreed that the *4Gains* project was an overall success despite the many challenges and setbacks that we faced. Even though our final product is significantly different from what we had in mind when we designed our theoretical model, the general idea of wireless data transfer, repetition counting, and graphical feedback was maintained. Furthermore, from the prototype version all the way up until the final product, the hardware components saw visible aging and decreases in performance and discrepancy between the actual performance of the sensors compared to the advertised versions when they were purchased. Finally, the time constraint of the project and the lack of experience in soldering resulted in loose connections between the various hardware components.

Despite the various challenges and difficulties that we faced, overcoming each and every one of them brought invaluable experience in the realm of hardware, frontend, data processing/analysis, and backend for every single group member. The three-month-long project helped us form career-long connections with like-minded peers within our program and sharpened our already diverse array of soft skills, including communication, teamwork, and initiative. I firmly believe that every group member of *4Fun* had much more takeaways from this project than they had expected, and the pride that every one of us feels at the conclusion of this project made it all worth it.

References

- Arduino - mysql: Arduino tutorial. Arduino Getting Started. (n.d.). Retrieved December 2, 2022, from <https://arduinogetstarted.com/tutorials/arduino-mysql>
- askSensors, & Instructables. (2020, July 21). *Connecting Arduino WIFI to the cloud using ESP8266*. Instructables. Retrieved December 2, 2022, from <https://www.instructables.com/Connecting-Arduino-WiFi-to-the-Cloud-Using-ESP8266/>
- Chris, Terenc, Sinha, C., Akindele, L., Dejan, Green, J., Richards, A., Manuel, Martin, Feroce-Lapin, Omar, & Chess, E. (2022, February 18). *Arduino and MPU6050 accelerometer and gyroscope tutorial*. How To Mechatronics. Retrieved December 2, 2022, from <https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/>
- Dmitry, Santos, R., Andy, Santos, S., Singh, R., Steve, Ma, S., Jimmy, Patil, A., Bonta, Pranav, Ramon, Benemerito, R. A., García, M. G., O, J., TheOldBrick, Layne, Pranay, André, ... Far. (2019, April 2). *Guide for microphone sound sensor arduino*. Random Nerd Tutorials. Retrieved December 2, 2022, from <https://randomnerdtutorials.com/guide-for-microphone-sound-sensor-with-arduino/>
- Dr Manab. (2020, April 23). *Numerical differentiation of discrete data in python | complete tutorial* [Video]. Youtube. https://www.youtube.com/watch?v=9G16HYUGENo&ab_channel=DrManab
- Last Minute Engineers. (2022, June 11). *How HC-SR04 ultrasonic sensor works & how to interface it with Arduino*. Last Minute Engineers. Retrieved December 2, 2022, from <https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/>
- Leon, C. D., Hamza, Kumar, S., Noob, Rastogi, A., Vip, Chakraborty, S., Disturbedbrown1, Microcontrollers Lab, Manasa, Saqib, Morteza, Fakler, F., & Petemango. (2022, May 17). *Data receiving on webpage from Arduino using ESP8266*. Microcontrollers Lab. Retrieved December 2, 2022, from https://microcontrollerslab.com/data-receiving-webpage-arduino-esp8266/#ESP-01_Wi-Fi_Module_Introduction
- Sanjeev, A. (2022, December 3). *How to Interface Arduino and the MPU 6050 sensor: Arduino*. Maker Pro. Retrieved December 2, 2022, from <https://maker.pro/arduino/tutorial/how-to-interface-arduino-and-the-mpu-6050-sensor>