Interface for Real-Time Feedback of EEG Data

An Advanced Project-1 in Data Engineering

Abhishek Mathur ***Ejsi Veshaj***

Contents:

Introduction:

The goal of this project is to create Brain-Computer Interface for real-time neurofeedback from EEG (electroencephalogram) signals for psychologists or experts and the participants. They can review and analyze the average band power as a very relevant metric to track brain activities of participants such as for sleep research to differentiate between the different sleep stages. The application could be an integral component of the Mental Health Management of participants. The user of the application is a physiatrist and data are collected from participants' activities (brain impulses tracked as data source). The aim is to study the brain activity of the participants so that psychiatrists could recommend correct solutions to the clients and manage their mental health improvements better.

The web application thrives on data collected from brain activity (converting EEG signals into data). This data is organized, retrieved, and analyzed through our application.

Data engineering is the practice of designing and building systems for collecting, storing, and analyzing data at scale. It is a broad field with applications in just about every industry. Hence, as Data Engineers, for our Advanced Project 1, we decided to build this web application using the trending high-level language 'Python'. And to challenge ourselves, we decide to work on real-time data and process it to yield the results for further analyses. To do so, we come across the opportunity to work on a psychological project to study brain activities via neurofeedback EEG signals.

Methods:

The real-time neurofeedback data is collected from the brain using a Muse headset. To do so, Muse 2 device and Petal Metrics application are used. The live data is then collected and filtered based on channel number using Python code and stored in a temporary file 'mytext.txt'. From this file, using python code, data points are converted into Power Spectral Density and Frequency and the outcome is seen by visualizations also created using Python. The major statuses are PSD vs Frequency graph and relative Band Powers (Delta, Theta, Alpha, and Beta).

Technologies Used:

- Python language
- Flask framework
- Visual Studio Code
- HTML
- pylsl
- Java Script
- Matplotlib

Installation:

These are the steps to set up your computer

- Installing Python and required packages:

Python is a high-level scripting language that has been widely adopted in many fields. It is open, free, simple to read, and has extensive standard libraries. Many packages can also be downloaded online to complement its features. Below are the packages that are needed to be installed for our project-

a) Flask
b) Pylsl
c) Scipy
d) Numpy
e) Seaborn
f) Os
g) Matplotlib

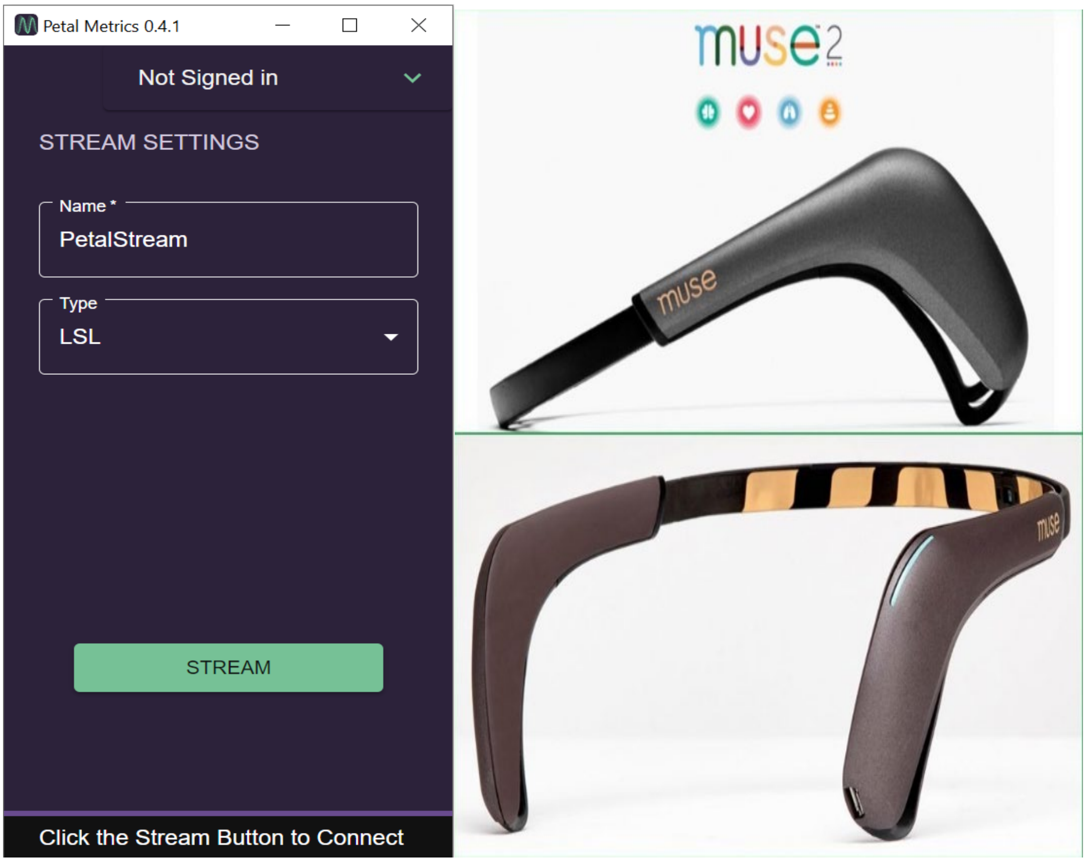
About Muse and its functioning:

- **Muse EEG Headset device:**

  Muse 2 EEG headband is a great piece of starter to technology to get into modern-day mind control. The main attraction of the Muse 2 is its **electroencephalography (EEG)** electrodes; they're what measure your brainwaves. EEG is a neuroimaging technique that allows us to listen to the complicated firing of neurons in the brain using voltage signals.

- **Connectivity with computer:**

  Pairing the Muse EEG headset with Petal Metrics. To do so, download the Petal Metrics application, install it and open it. Turn on the computer's Bluetooth, then chose type 'LSL' in the Petal app and switch on the Muse device and click on 'STREAM', as shown below-



```python
if request.method == 'POST' and 'channel' in request.form:

    print("looking for an EEG stream...")
    streams = resolve_stream('type', 'EEG') # create a new inlet to read # from the stream
    inlet = StreamInlet(streams[0])

    ch=request.form.get('channel')
    if ch != 'Channel Number':
        ch=int(ch)
        url='/static/images/plot.png'

    print('selected channel: ',ch)

    i=0
    lst=[]

    while True:

        if i==500:
            break
        else:
            sample, timestamp = inlet.pull_sample()
            lst.append(sample)
            print(sample)
        i=i+1

    eeg_data_n=[item[ch] for item in lst]
```

With the help of Muse device and Petal Metrics app, EEG signals are captured with the help of python library ‘pylsl’. Channel numbers can be selected by the user and the corresponding data points then are collected and stored in a text file.

Data gathering, filtering, and processing using Python libraries:

By using Python libraries mentioned above and Muse device signals, the app collects the raw EEG signals from all the channels and then as per the user’s request, filters out only one channel’s data points to be considered. Signal noises are removed using some data cleaning methods and then this data is stored in a text file (myfile.txt) within the application.

```
myfile.txt ×    app.py
application > myfile.txt
-633.30078125
-1000.0
-1000.0
227.5390625
-999.51171875
-669.43359375
-1000.0
-1000.0
-684.5703125
545.41015625
-524.4140625
-1000.0
-1000.0
-874.51171875
425.29296875
-482.91015625
-1000.0
-968.26171875
-999.51171875
180.6640625
-459.9609375
-1000.0
-940.4296875
-1000.0
```

Using this text file, PSD (Power Spectral density) and its corresponding frequency bands namely Delta (1.5 – 4 Hz), Theta (4 – 8 Hz), Alpha (8 – 14 Hz), and Beta (14 – 30 Hz) are extracted and plotted on a graph. Using the PSD and Frequencies values, Relative band Powers for each frequency band are calculated.

Total Frequency bands power: 1942191.0558219904 V^2 / Hz

Total Delta band power: 150835.89261741296 V^2 / Hz || Relative Delta power: 7.766274701207236 %

Total Theta band power: 640937.7913914968 V^2 / Hz || Relative Theta power: 33.00075908959604 %

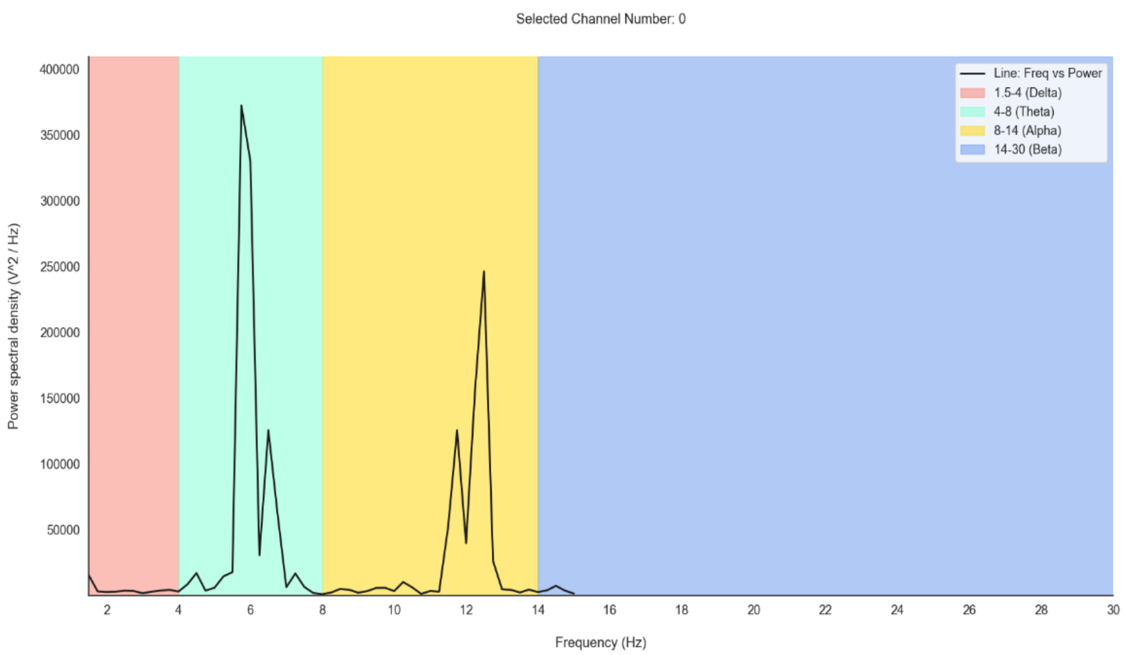
Total Alpha band power: 1132992.132019879 V^2 / Hz || Relative Aplha power: 58.33577127356118 %

Total Beta band power: 17425.23979320161 V^2 / Hz || Relative Beta power: 0.8971949356355549 %

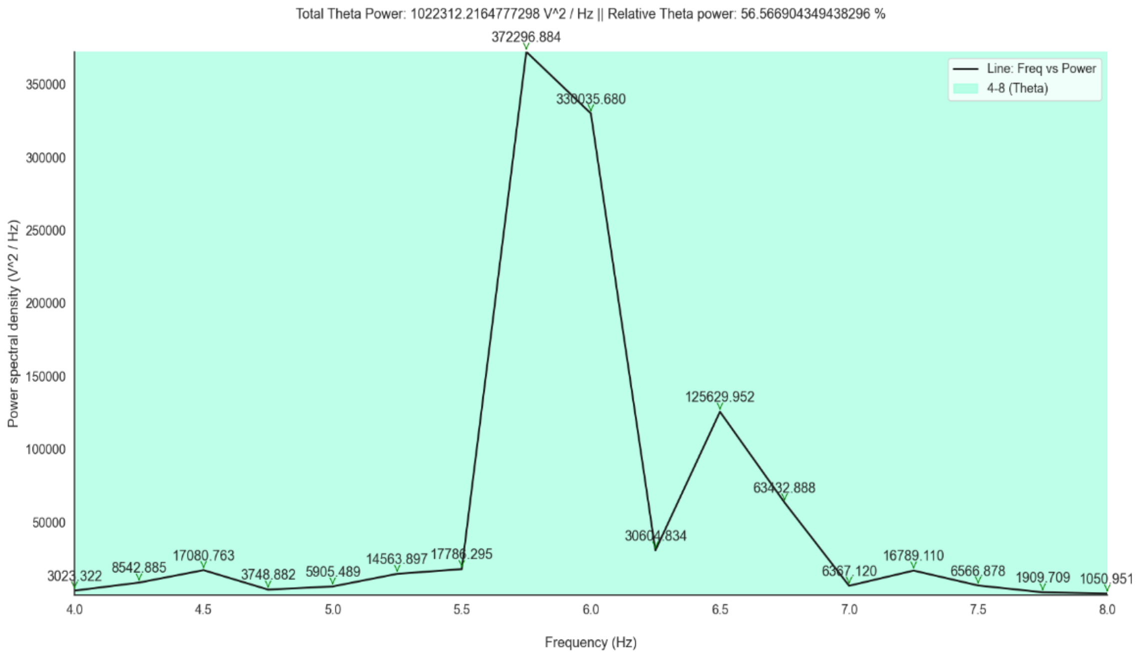
When the user needs to check the PSD data points in detail for a particular frequency band, the app creates a graph with details for that individual frequency band.

The processed data in this project is divided as:

- For all frequency bands - PSD vs Frequency graph



- For particular frequency band PSD data - PSD vs Frequency graph(band specific)



These graphs and relative band powers are live data which change with every new choice of the user and refresh button. Processing the data including, extraction, cleaning, and filtering follows further in this document.

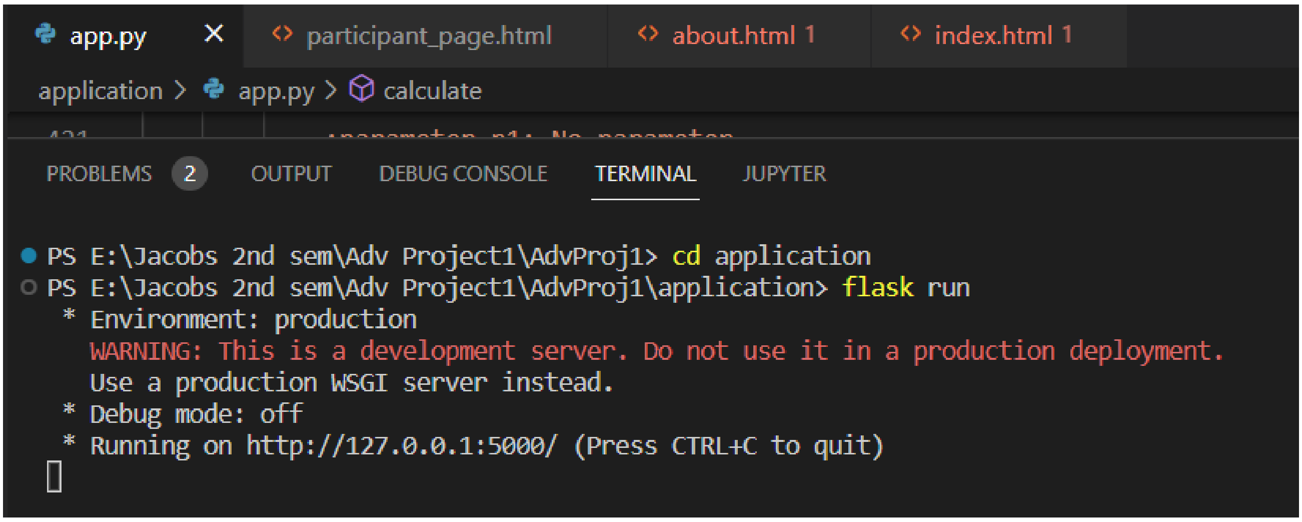
Visual Studio Code, Flask, Python libraries and HTML pages:

application > app.py > ...

```
from flask import Flask, render_template,request
from pylsl import StreamInlet, resolve_stream # first resolve an EEG # stream on the lab network
from scipy import signal
import numpy as np
import os
import seaborn as sns
import matplotlib.pyplot as plt

app = Flask(__name__)

# @app.route('/about', methods = ['POST','GET'])


@app.route('/', methods = ['POST','GET'])
```

Visual Studio Code is used to design this web application. It aims to provide just the tools a developer needs for a quick code-build-debug cycle and leaves more complex workflows to fuller featured IDEs. We use multiple Python libraries for this project like flask, pylsl, scipy, and so on. We have created app.py where app.py is the entry point for our Python applications. All the functions related to the HTML pages and their functionalities are written in this app.py.

The application can run the flask application on VS Code by using the command ‘flask run’ in the terminal and will be able to access the contents on http://127.0.0.1:5000.



Following are the functions that we have created to use:

- **Calculate():** This function calls the home page (index.html) where the user can see an interesting and user-friendly web page. Here user can select the channel number and get the desired PSD vs Frequency graph along with relative band powers. Here you can also go to the ‘About’ page by clicking on the About menu at the top left corner.

  Below is the code for the same-about.html 1 | app.py | participant_page.html | index.html 1

pplication > app.py > calculate

```python

def calculate():
    """
        calculate() function do the cumputation where as per the chosen channel number of device by the user, the resultant graph
        of Power Spactral Density vs Frequency gets displayed. It also calculates individual total band powers and their relative band powers.

            :parameter p1: No parameter.
            :return: 'index.html',
                                channel=[{'noc': 'Channel Number'}, {'noc': 0}, {'noc': 1}, {'noc': 2}, {'noc': 3}, {'noc': 4}],
                                url,
                                 tot,
                                sum_delta,
                                sum_theta,
                                sum_alpha,
                                sum_beta,
                                delta_band_power,
                                theta_band_power,
                                alpha_band_power,
                                beta_band_power

    """

    url=''

    tot=0,
    sum_delta=0,
    sum_theta=0,
    sum_alpha=0,
    sum_beta=0,
    delta_band_power=0.0,
    theta_band_power=0.0,
    alpha_band_power=0.0,
    beta_band_power=0.0

    if request.method == 'POST' and 'channel' in request.form:

        print("looking for an EEG stream...")
        streams = resolve_stream('type', 'EEG') # create a new inlet to read # from the stream
        inlet = StreamInlet(streams[0])

        ch=request.form.get('channel')
        if ch != 'Channel Number':
            ch=int(ch)
            url='/static/images/plot.png'

        print('selected channel: ',ch)

        i=0
        lst=[]

        while True:

            if i==500:
                break
            else:
                sample, timestamp = inlet.pull_sample()
                lst.append(sample)
                print(sample)
            i=i+1
```

app.py × index.html 1 patient_page.html about.html 1

application > app.py > calculate

```python
            val=str(i)+'\n'
            file1.write((val))

        file1.close()

        data = np.loadtxt('myfile.txt')
        # Define sampling frequency and time vector
        sf = 30
        time = np.arange(data.size) / sf

        # Define window length (4 seconds)
        win = 4 * sf
        freqs, psd = signal.welch(data, sf, nperseg=win)

        sns.set(font_scale=1.2, style='white')
        plt.figure(figsize=(20, 10))
        plt.plot(freqs, psd, color='k', lw=2)
        plt.xlabel('Frequency (Hz)', labelpad=20)
        #plt.vlines([1.5,4,8,14,30], 1.5,175000, linestyles='dashed', colors='red')
        plt.axvspan(1.5, 4, color='salmon', alpha=0.5)
        plt.axvspan(4,8, color='aquamarine', alpha=0.5)
        plt.axvspan(8,14, color='gold', alpha=0.5)
        plt.axvspan(14,30, color='cornflowerblue', alpha=0.5)
        #plt.vlines([12], 0, 1000, linestyles='dashed', colors='blue')
        plt.ylabel('Power spectral density (V^2 / Hz)',labelpad=20)
        plt.ylim([1.5, psd.max() * 1.1])
        plt.title(f"Selected Channel Number: %d" % ch, y=1.05)
        plt.xticks(np.arange(0, len(freqs)+1, 2))
        plt.xlim([1.5, 30])  #plt.xlim([0, freqs.max()])
        plt.legend(['Line: Freq vs Power', '1.5-4 (Delta)','4-8 (Theta)','8-14 (Alpha)','14-30 (Beta)'],
                bbox_to_anchor=(1,1), loc= 'upper right')
        sns.despine()

        # fig_(fig)

        plt.savefig('static/images/plot.png')

        ###To get total power of all ferq bands:
        tot=0
        for i in range(6,len(psd)):

            tot=tot+psd[i]

        #print(max(freqs),'  max freq', ' index of psd:',(len(psd)+1), ' total:',tot)

        if max(freqs)>=4:
            #For Delta graph
            lst=[]
            sns.set(font_scale=1.2, style='white')
            plt.figure(figsize=(20, 10))
            plt.plot(freqs, psd, color='k', lw=1,  marker='o')

            for x,y in zip(freqs,psd):

                label = "{:.2f}".format(y)
```

application > app.py > calculate

```python
            plt.plot(freqs, psd, color='k', lw=1, marker='o')

            for x,y in zip(freqs,psd):

                label = "{:.2f}".format(y)

                plt.annotate(label,
                    (x,y),
                    textcoords="offset points",
                    xytext=(0,10),

                    ha='center',
                    arrowprops=dict(arrowstyle="->", color='green'))

            plt.xlabel('Frequency (Hz)', labelpad=20)
            plt.axvspan(1.5, 4, color='salmon', alpha=0.5)
            plt.legend(['Line: Freq vs Power', '1.5-4 (Delta)'],
            bbox_to_anchor=(1,1), loc= 'upper right')
            plt.xlim([1.5, 4])
            first=int((np.where(freqs == 1.5))[0])
            last=int((np.where(freqs == 4.0))[0])
            print(first,'1st ',last,' last')
            #to get the band power for delta range
            sum_delta=0

            for i in range(first,last+1):
                sum_delta=sum_delta+psd[i]
                #print(i, '', psd[i])
            delta_band_power=(sum_delta/tot)*100
            print('sum_delta ',sum_delta,' sum_delta/tot= ',sum_delta/tot, ' ',delta_band_power)

            for i in range(first,last+1):
                lst.append(psd[i])
            #print(max(lst), '  max Delta')
            plt.ylim([1.5, max(lst)+5])  #plt.xlim([0, freqs.max()])
            plt.ylabel('Power spectral density (V^2 / Hz)', labelpad=20)

            plt.title(f"Total Delta Power: %s V^2 / Hz || Relative Delta power: %s %% "
                % (str(sum_delta), str(delta_band_power)), y=1.05)
            sns.despine()
            plt.savefig('static/images/Delta.png')

    #For Theta graph
        if max(freqs)>=8:

            lst=[]
            sns.set(font_scale=1.2, style='white')
            plt.figure(figsize=(20, 10))
            plt.plot(freqs, psd, color='k', lw=2)

            for x,y in zip(freqs,psd):

                label = "{:.3f}".format(y)

                plt.annotate(label,
```

application > app.py > calculate

```python

                plt.annotate(label,
                    (x,y),
                    textcoords="offset points",
                    xytext=(0,10),

                    ha='center',
                    arrowprops=dict(arrowstyle="->", color='green'))

            plt.xlabel('Frequency (Hz)', labelpad=20)
            plt.axvspan(4,8, color='aquamarine', alpha=0.5)
            plt.legend(['Line: Freq vs Power', '4-8 (Theta)'],
            bbox_to_anchor=(1,1), loc= 'upper right')
            plt.xlim([4,8])
            first=int((np.where(freqs == 4.0))[0])
            last=int((np.where(freqs == 8.0))[0])

            #to get the band power for Theta range
            sum_theta=0

            for i in range(first+1,last+1):
                sum_theta = sum_theta + psd[i]
                #print(i, '', psd[i])
            theta_band_power= (sum_theta/tot)*100
            print('sum_theta ',sum_theta,' sum_theta/tot= ',sum_theta/tot, ' ',theta_band_power)

            for i in range(first,last+1):
                lst.append(psd[i])
            #print(max(lst), '  max Theta')
            plt.ylim([1.5, max(lst)+5]) #plt.xlim([0, freqs.max()])
            plt.ylabel('Power spectral density (V^2 / Hz)')
            plt.title(f"Total Theta Power: %s V^2 / Hz || Relative Theta power: %s %%"
                % (str(sum_theta), str(theta_band_power)), y=1.05)

            sns.despine()
            plt.savefig('static/images/Theta.png')

    #For Alpha graph
        if max(freqs)>=14:
            #For Alpha graph
            lst=[]
            sns.set(font_scale=1.2, style='white')
            plt.figure(figsize=(20, 10))
            plt.plot(freqs, psd, color='k', lw=2)

            for x,y in zip(freqs,psd):

                label = "{:.3f}".format(y)

                plt.annotate(label,
                    (x,y),
                    textcoords="offset points",
                    xytext=(0,10),
                    # and the text label
                    ha='center',
```

application > app.py > calculate

```python
                    # and the text label
                    ha='center',
                    arrowprops=dict(arrowstyle="->", color='green'))

            plt.xlabel('Frequency (Hz)', labelpad=20)
            plt.axvspan(8,14, color='gold', alpha=0.5)
            plt.legend(['Line: Freq vs Power', '8-14 (Alpha)'],
            bbox_to_anchor=(1,1), loc= 'upper right')
            plt.xlim([8,14])
            first=int((np.where(freqs == 8.0))[0])
            last=int((np.where(freqs == 14.0))[0])

            #to get the band power for Alpha range
            sum_alpha=0

            for i in range(first+1,last+1):
                sum_alpha = sum_alpha + psd[i]
                #print(i, '', psd[i])
            alpha_band_power = (sum_alpha/tot )*100
            print('sum_alpha ',sum_alpha,' sum_alpha/tot= ',sum_alpha/tot, ' ',alpha_band_power)


            #to set Y limit as per highest value of Y in X- range
            for i in range(first,last+1):
                lst.append(psd[i])
            #print(max(lst), '  max aplha')
            plt.ylim([1.5, max(lst)+5])
            plt.ylabel('Power spectral density (V^2 / Hz)', labelpad=20)
            plt.title(f"Total Alpha Power: %s V^2 / Hz || Relative Alpha power: %s %%"
                % (str(sum_alpha), str(alpha_band_power)), y=1.05)

            sns.despine()
            plt.savefig('static/images/Alpha.png')

    #For Beta graph
        if max(freqs)<=30 and max(freqs)>14:
            #For Beta graph
            lst=[]
            sns.set(font_scale=1.2, style='white')
            plt.figure(figsize=(20, 10))
            plt.plot(freqs, psd, color='k', lw=2)

            for x,y in zip(freqs,psd):

                label = "{:.2f}".format(y)

                plt.annotate(label,
                    (x,y),
                    textcoords="offset points",
                    xytext=(0,10),

                    ha='center',
                    arrowprops=dict(arrowstyle="->", color='green'))

            plt.xlabel('Frequency (Hz)', labelpad=20)
```

application > app.py > calculate

```python

                    ha='center',
                    arrowprops=dict(arrowstyle="->", color='green'))

            plt.xlabel('Frequency (Hz)', labelpad=20)
            plt.axvspan(14,30, color='cornflowerblue', alpha=0.5)
            plt.legend(['Line: Freq vs Power', '14-30 (Beta)'],
            bbox_to_anchor=(1,1), loc= 'upper right')
            plt.xlim([14,30])
            first=int((np.where(freqs == 14.0))[0])
            last=int((np.where(freqs == max(freqs)))[0])

            sum_beta=0

            for i in range(first+1,last+1):
                sum_beta = sum_beta + psd[i]
                #print(i, '', psd[i])
            beta_band_power = (sum_beta/tot)*100
            #print('sum_beta ',sum_beta,' sum_beta/tot= ',sum_beta/tot, ' ',beta_band_power)


            #to set Y limit as per highest value of Y in X- range
            for i in range(first,last+1):
                lst.append(psd[i])
            #print(max(lst), '  max aplha')
            plt.ylim([1.5, max(lst)+5])
            plt.ylabel('Power spectral density (V^2 / Hz)', labelpad=20)
            plt.title(f"Total Beta Power: %s V^2 / Hz || Relative Beta power: %s %%"
                % (str(sum_beta), str(beta_band_power)), y=1.05)

            sns.despine()
            plt.savefig('static/images/Beta.png')

        elif max(freqs)>30:
            #For Beta graph
            lst=[]
            sns.set(font_scale=1.2, style='white')
            plt.figure(figsize=(20, 10))
            plt.plot(freqs, psd, color='k', lw=2)


            for x,y in zip(freqs,psd):

                label = "{:.3f}".format(y)

                plt.annotate(label, # this is the value which we want to label (text)
                    (x,y), # x and y is the points location where we have to label
                    textcoords="offset points",
                    xytext=(0,10), # this for the distance between the points
                    # and the text label
                    ha='center',
                    arrowprops=dict(arrowstyle="->", color='green'))

            plt.xlabel('Frequency (Hz)', labelpad=20)
            plt.axvspan(14,30, color='cornflowerblue', alpha=0.5)
```

application > app.py > calculate

```python
            plt.xlabel('Frequency (Hz)', labelpad=20)
            plt.axvspan(14,30, color='cornflowerblue', alpha=0.5)
            plt.legend(['Line: Freq vs Power', '14-30 (Beta)'],
            bbox_to_anchor=(1,1), loc= 'upper right')
            plt.xlim([14,30])
            first=int((np.where(freqs == 14.0))[0])
            last=int((np.where(freqs == 30.0))[0])

            sum_beta=0

            for i in range(first+1,last+1):
                sum_beta=sum_beta+psd[i]
                #print(i, '', psd[i])
            beta_band_power=(sum_beta/tot)*100
            #print('sum_beta ',sum_beta,' sum_beta/tot= ',sum_beta/tot, ' ',beta_band_power)


            #to set Y limit as per highest value of Y in X- range
            for i in range(first,last+1):
                lst.append(psd[i])
            #print(max(lst), '  max Beta')
            plt.ylim([1.5, max(lst)+5])
            plt.ylabel('Power spectral density (V^2 / Hz)', labelpad=20)
            plt.title(f"Total Beta Power: %s V^2 / Hz || Relative Beta power: %s %%" % (str(sum_beta), str(beta_band_power)), y=1.05)

            plt.savefig('static/images/Beta.png')

    #
        print(freqs)
        print(psd)
        print(max(freqs),'  max freq', ' index of psd:',(len(psd)+1), ' total:',tot)

    return render_template('index.html',
                           channel=[{'noc': 'Channel Number'}, {'noc': 0}, {'noc': 1}, {'noc': 2}, {'noc': 3}, {'noc': 4}],
                           url=url,
                            tot=tot,
                           sum_delta=sum_delta,
                           sum_theta=sum_theta,
                           sum_alpha=sum_alpha,
                           sum_beta=sum_beta,
                           delta_band_power=delta_band_power,
                           theta_band_power=theta_band_power,
                           alpha_band_power=alpha_band_power,
                           beta_band_power=beta_band_power)
```

__Index.html page__: calculate() function calls index.html page where first it takes the input from the user for channel number and then, as a result, returns PSD vs Frequency graph and relative band-powers.

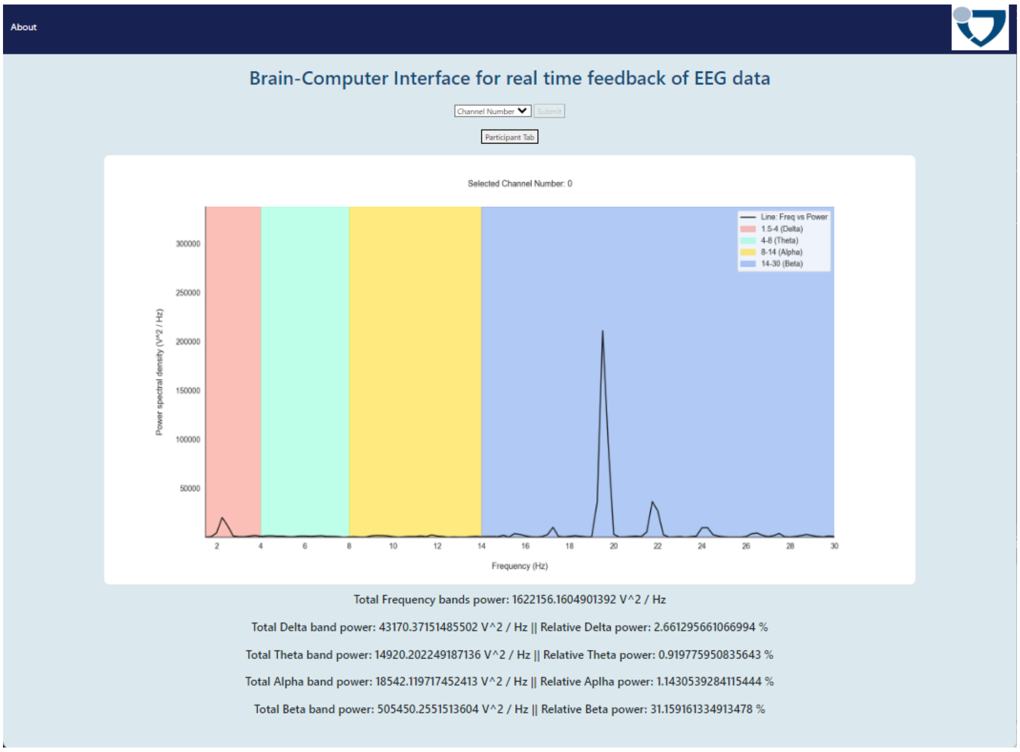
index.html 1 | app.py | patient_page.html | about.html 1

application > templates > index.html > html > body > body > form > img#plot

```html
<html>

    <link rel="stylesheet" href=
        "https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">

        <style>
                .navbar-custom {
                    background-color: rgb(24, 34, 82);

            }
            /* Modify brand and text color */

            .navbar-custom .navbar-brand,
            .navbar-custom .navbar-text {
                color: rgb(255, 255, 255);
            }

            img {
            display: block;
            }

            #logo{
            margin-left: 90%;
            }

            #plot{
            margin-right:auto;
            margin-left: auto;
            border-radius: 11px;
            }

            h1 {
            text-align: center;
            }

                body {
                    background-color: rgb(219, 233, 238);
            }

        </style>


        <body>


            <div class="navbar-item">

                <nav class="navbar navbar-custom">
                    <a class="navbar-brand" href="{{ url_for('about') }}" class="navbar-item" >About</a>
                    <img id="logo" align ='margin-right' src='/static/images/jacob_logo.jpeg' width=120" height="100" >
                </nav>
            </div>

            <br>

            <h1 style="color: rgb(22, 76, 120)">Brain-Computer Interface for real time feedback of EEG data</h1>

        <body>


        <form action="/" method="POST">

            <center>
                <br>
            <select name="channel" id="lst" class="Input" onChange="enable(this)">
                {% for o in channel %}
                <option value="{{ o.noc }}">{{ o.noc }}</option>
```

```html
            <center>
                <br>
            <select name="channel" id="lst" class="Input" onChange="enable(this)">
                {% for o in channel %}
                <option value="{{ o.noc }}">{{ o.noc }}</option>
                {% endfor %}
            </select>

            <button disabled="true" id="btn" type="submit" value="submit";">Submit</button>

            <br>
            <br>
            <button type="button" onclick="window.location.href='{{ url_for( 'participant_page') }}';">Participant Tab</button>

            <script type='text/javascript'>

                function enable(channels)
                    {
                    var btn=document.getElementById("btn");
                    if(channels.value == 'Channel Number')
                    {
                        btn.disabled = true
                    }
                    else
                    {
                        btn.disabled = false
                    }
                    }

            </script>


            </center>

            <br>

            <img id="plot" src={{url}} width=1700" height="900" alt=" Graph will be displayed here. Choose the channel Number then
                    click on Submit button and please wait...">

            <center>
                    <div style="width: 1500px; height: 250px;">
                        <p style="text-align:justify;">
                            <h4>
                                Total Frequency bands power: {{tot}} V^2 / Hz
                                <br><br>
                                Total Delta band power: {{sum_delta}} V^2 / Hz    ||    Relative Delta power: {{delta_band_power}} %
                                <br><br>
                                Total Theta band power: {{sum_theta}} V^2 / Hz    ||    Relative Theta power: {{theta_band_power}} %
                                <br><br>
                                Total Alpha band power: {{sum_alpha}} V^2 / Hz    ||    Relative Aplha power: {{alpha_band_power}} %
                                <br><br>
                                Total Beta band power: {{sum_beta}} V^2 / Hz   ||    Relative Beta power: {{beta_band_power}} %
                            </h4>
                        </p>
                    </div>
            </center>

        </form>

    </body>

</html>
```

Once the Channel Number is selected and ‘Submit’ is clicked, the resultant page looks like the below-



- **Participant_page():** This function calls the participant page after clicking on the ‘Participant Tab’ button on the home(index.html) page, where, the user can select a particular frequency band to analyze the band power in detail. Here you can go back to the Home page by clicking on the Nav bar at the top left or you can see the about page by clicking on the about menu at the top left corner.

about.html 1 | app.py | participant_page.html | index.html 1

application > app.py > calculate

```python


@app.route("/participant_page/", methods = ['POST','GET'])

def participant_page():

    """
        participant_page() function computes .

            :parameter p1: No parameter.
            :return: "participant_page.html", band, url

    """


    url_=''
    if request.method == 'POST' and 'band' in request.form:

        band=request.form.get('band')
        print('selected Frequency band: ',band)
        if band ==  'Delta: 0-4 Hz':
            url_='/static/images/Delta.png'
        if band ==  'Theta: 4-8 Hz':
            url_='/static/images/Theta.png'
        if band ==  'Alpha: 8-14 Hz':
            url_='/static/images/Alpha.png'
        if band ==  'Beta: 14-30 Hz':
            url_='/static/images/Beta.png'

    return render_template('participant_page.html',
                                band=[{'band': 'Select Band Name'}, {'band': 'Delta: 0-4 Hz'}, {'band': 'Theta: 4-8 Hz'}, {'band': 'Alpha: 8-14 Hz'}, {'band': 'Beta: 14-30 Hz'}],
                                url_=url_
                               )

```

***P**articipant _page.html page*: participant_page() function calls the participant _page.html where it takes a specific frequency band range for which a detailed result needs to be obtained and then it shows the PSD vs Frequency graph for that particular frequency band range.

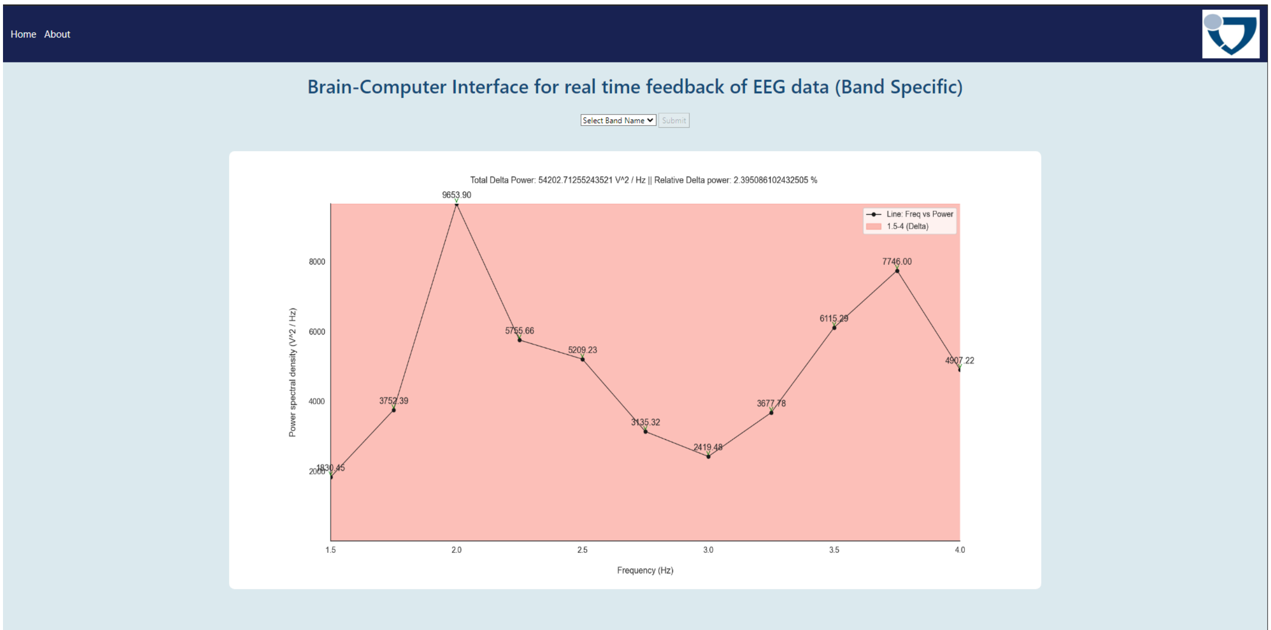
participant_page.html × | about.html 1 | app.py | index.html 1

application > templates > participant_page.html > html > body > form

```html
<html>

    <head>

        <link rel="stylesheet" href=
        "https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">

        <style>
             .navbar-custom {
            background-color: rgb(24, 34, 82);
            }
            /* Modify brand and text color */

            .navbar-custom .navbar-brand,
            .navbar-custom .navbar-text {
                color: rgb(255, 255, 255);
            }

            img {
            display: block;

            }

            h1 {
            text-align: center;
            }

            h2 {
            text-align: center;
            }

            body {
                background-color: rgb(219, 233, 238);
            }

            #logo{
            margin-left: 90%;
            }

            #plot{
            margin-right:auto;
            margin-left: auto;
            border-radius: 11px;
            }
        </style>

    </head>

        <body>

            <nav class="navbar navbar-custom">
                <a class="navbar-brand" href="{{ url_for('calculate') }}"  >Home</a>
                <a class="navbar-brand" href="{{ url_for('about') }}" >About</a>
                <img id="logo" align ='margin-right' src='/static/images/jacob_logo.jpeg' width=120" height="100" >
            </nav>

            <br>

            <h1 style="color: rgb(22, 76, 120)">Brain-Computer Interface for real time feedback of EEG data (Band Specific)</h1>

            <br>


        <form action="/participant_page" method="POST">

            <center>
```

```
            <center>



                <select name="band" id="lst" class="Input" onChange="enable(this)">
                    {% for o in band %}
                    <option value="{{ o.band }}">{{ o.band }}</option>
                    {% endfor %}
                </select>

                <button disabled="true" id="btn" type="submit" value="submit";">Submit</button>

                <script type='text/javascript'>

                    function enable(bands)
                        {
                        var btn=document.getElementById("btn");
                        if(bands.value == 'Select Band Name')
                        {
                            btn.disabled = true
                        }
                        else
                        {
                            btn.disabled = false
                        }
                        }
                </script>

            </center>

            <br>
            <br>

            <img id="plot" src={{url_}} width=1700" height="900" alt="NOTE: Graph will be displayed here...  Please connect the muse device
            then select channel number and then click on submit button. Then chose the band range to get the specific band information...">

        </form>

    </body>

</html>
```

Once the Band name is selected and ‘Submit’ is clicked, the resultant page looks like below-



- **about():** This function calls the about.html page that shows the description about the application and a note below. Here you can go back to Home page by click on the Nav bar at the top left or by clicking on the ‘Home’ button below the image.

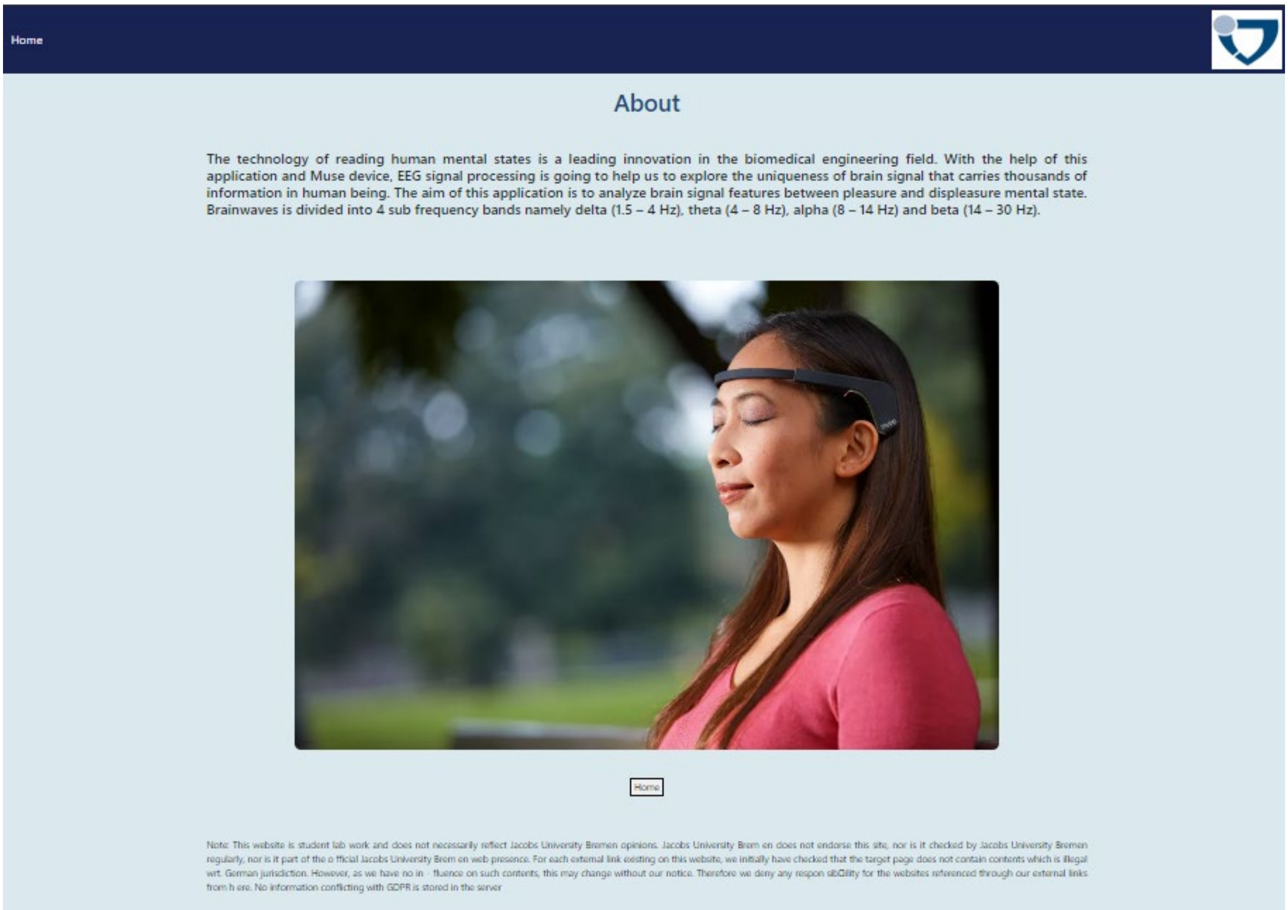
```python
@app.route("/about/")

def about():

    """
        about() function shows the about page where a brief discription of the application is mentioned.

            :parameter p1: No parameter.
            :return: "about.html",url

    """

    return render_template("about.html",url='/static/images/muse_pic.png')
```

about.html page: About page is called by about() function . Code is given as below:

```html
application > templates > about.html > html > body > style > #plot

<html>
    <body>

        <link rel="stylesheet" href=
            "https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">


        <style>
            .navbar-custom {
                background-color: rgb(24, 34, 82);
            }
                /* Modify brand and text color */

            .navbar-custom .navbar-brand,
            .navbar-custom .navbar-text {
                color: rgb(255, 255, 255);
            }

            img {
                display: block;
            }

            h1 {
                text-align: center;
            }

            body {
                background-color: rgb(219, 233, 238);
            }

            #logo{
                margin-left: 90%;
            }

            #plot{
                margin-right:auto;
                margin-left: auto;
                border-radius: 11px;
            }

        </style>



        <div class="navbar-item">

            <nav class="navbar navbar-custom">
                <a class="navbar-brand" href="{{ url_for('calculate') }}" class="navbar-item" >Home</a>
                <img id="logo" align ='margin-right' src='/static/images/jacob_logo.jpeg' width=120" height="100" >
            </nav>
        </div>
```

```html
application > templates > about.html > html > body > style > #plot
            <nav class="navbar navbar-custom">
                <a class="navbar-brand" href="{{ url_for('calculate') }}" class="navbar-item" >Home</a>
                <img id="logo" align ='margin-right' src='/static/images/jacob_logo.jpeg' width=120" height="100" >
            </nav>
        </div>


        <br>

        <h1 style="color: rgb(22, 76, 120)">About</h1>

        <br><br>

        <center>
            <div style="width: 1500px; height: 200px;">
                <h4>
                    <p style="text-align:justify;">
                    The technology of reading human mental states is a leading innovation in the biomedical engineering field.
                    With the help of this application and Muse device, EEG signal processing is going to help us to explore the uniqueness of brain signal that carries thousands of information in human being.
                    The aim of this application is to analyze brain signal features between pleasure and displeasure mental state.
                    Brainwaves is divided into 4 sub frequency bands namely delta (1.5 – 4 Hz), theta (4 – 8 Hz), alpha (8 – 13 Hz) and beta (13 – 30 Hz).
                    </p>
                </h4>
            </div>


            <br>


            <img id="plot" src={{url}} width=1200" height="800">

            <br>
            <br>

            <button type="button" onclick="window.location.href='{{ url_for('calculate') }}';">Home</button>

            <br>
            <br>
            <br>
            <br>
        </center>

        <center>
            <div style="width: 1500px; height: 250px;">
                <p style="text-align:justify;">
                        This website is student lab work and does not necessarily reflect Jacobs University Bremen opinions.
                Jacobs University Brem en does not endorse this site, nor is it checked by Jacobs University Bremen
                regularly, nor is it part of the o fficial Jacobs University Brem en web presence.
                For each external link existing on this website, we initially have checked that the target page
                does not contain contents which is illegal wrt. German jurisdiction. However, as we have no in -
                fluence on such contents, this may change without our notice. Therefore we deny any respon sibility for the websites referenced through our external links from h ere.
                No information conflicting with GDPR is stored in the server
                </p>
            </div>
        </center>
    </body>

</html>
```

When ‘About’ link is clicked, the resultant page looks like below-



Product description:

From the report, we see that a part of 'Interface for Real-Time Feedback of EEG Data'

is implemented with the following scope:

- Secure features including all the buttons, dropdown list and links for the pages. With these, there will be less chance of human error in the web page and the process.
- Python functions calculate(), participant_page() and about() are created to perform dedicated tasks on the HTML pages.
- EEG signal functions are implemented to get the results in form of PSD vs Frequency and relatives band powers of all frequency bands.
- Data cleaning is applied where noise is removed from (0-1.5) Hz.
- Filters are used to select a particular channel number from the device and then to choose the frequency band from which detailed results can be analyzed.

Links that are working with the current implementation are tested and verified operations by running the app on Windows System (Win11, AMD Ryzen7, 16GB RAM). In terms of psychology knowledge, we covered the basic principles behind the use of electroencephalography signals in modern BCI applications: properties of the raw EEG time series, extraction of band power features based on individual frequency bands (Delta (1.5 – 4 Hz), Theta (4 – 8 Hz), Alpha (8 – 14 Hz), and Beta (14 – 30 Hz)).

So far, our application does the job of choosing the Muse device channel number and getting the real-time data for all frequency bands and individual frequency bands. The future Implementations shall include:

- Login security for both Psychologists and participants.
- To include live signal graphs for all frequency bands using python.
- Include a visual drop or increase in the power of frequencies.
- Implementing the same on a cloud-based server.
- Maintain a database periodically using Python and DBMS once it is implemented in the cloud.

Project's individual evaluations:

Abhishek:

Areas of ownership-

- Establish the Bluetooth connection with the computer and stream the real-time brain signals.
- Performing FFT transformation on the above brain signals, to convert the time to frequency (the required parameter).
- Build the user interface for the participant.

Being Data Engineer, to hone my data engineering skills, I wanted to work on a challenging project where the topic and the programming language should be new to me. I am grateful to get an interesting project to work on which is based on human psychology using real-time neurofeedback from the brain using a device called Muse, which I had never heard of. Secondly, I find the topic of 'mental health highly fascinating and so I am personally drawn to the opportunity of making an impact in this highly relevant field in today's field. The opportunity to make an impact in the field of mental health is greatly inspiring for me.

To work on the project-

- I researched the Muse device first, what it does and how it can be utilized to analyse brain activities. Once it was done, I implemented the setup to connect the device with the computer and successfully was able to connect and get the brain signals using Python language.
- After that, I was successfully able to convert the signals into data points and get the frequency bands and their amplitude in form of Powe Spectral Density.
- Then I was able to create web pages to build Brain-Computer Interface successfully where users can get the data and can track brain activities based on that.

I am very much satisfied with my results, however, was not able to reach the target of 100%. My initial goal was to get a real-time graph. However, in the final deliverable, the real-time data is captured, and the final visualization is a static graph based on the data. Hence, it's broken into 2 steps where live data is coming in parts instead of a continuous stream. Nevertheless, I am highly motivated and satisfied with my learning in a short time span with new technology and a project topic that was totally challenging yet interesting and new to me.

The most challenging parts were:

- To research the Muse- device connectivity with the computer and to study how to perform the transformation of data points from it to get the desired result in form of power and frequency. Change in the plan came when at the initial point I had to decide which application to use to connect the Muse device with the computer. I worked on the BlueMuse application first, but despite fulfilling all its requirements, it was not able to connect to the device via Bluetooth. Then I switched to the Petal Metrics application as a connector of the device with the computer, and it worked successfully.

- To get a continuously changing graph with a continuous stream of data. The challenges here that I faced were, that I have never worked on a Python library like 'pylsl' before and the language itself is new to me altogether. Secondly, the topic itself was new to me and the process was not handy anywhere I can research. Here, I had to change my plan. Instead of getting a continuous graph, I decided to break the process into half; first, collect the live data, store it, and through the result with a click of a button. After analysing the first static graph, the user can again get the real-time data and get the results.

Learning:

It was a great experience working on a complex topic with a programming language that is new to me. I believe, that now I am able to handle and deal with challenging projects. I have also improved my Python skills along with HTML that I used in the project. Also, I got a chance to learn about the neurofeedback system and how my skills can help in analysis in this domain. In the end, I am confident in my data management and python skills through this project. Also, my project management skills to meet client requirements and deliver impactful work.

Ejsi:

Areas of ownership-

- Determining the corresponding electrodes for each signal value and building the plot (amplitude vs time).
- Applying some data cleaning methods to remove some signal noises and produce the respective output based on the signal pattern.
- Build the user interface for the psychology expert.

The first reason why I decided to work on this topic is that I really like unique projects regardless of the challenges they pose. Secondly, psychology represents one of my hobbies and I found it quite interesting to apply my software and data skills in this field. During the implementation of the different components, I encountered a lot of difficulties. This is because I only had 3 months of experience in Python, acquired during the first semester of my master's degree. While the processing of signals in real-time is a very complicated task. However, by constantly searching for information about this topic and testing different methods, we managed to build a good product at the end of this project. I also tried to implement some components in MATLAB as an additional alternative. But then I decided to use only Python, avoiding combinations between these programming languages. Being quite ambitious, of course, I am satisfied only if I achieve 100% of the goal of a project. On the other hand, I am satisfied with the significant progress of my skills during this project and with the fact that, despite the lack of knowledge, I managed to reach a significant % of my aims. The part that remained unrealized belongs to the processing of real-time signals with negligible delay. But I am currently trying to master my skills in the combination of fields used by this project, and I highly consider improving it during the 3rd semester. All in all, I would recommend such a topic to any student who is a senior in Python or MATLAB or anyone else who has a high level of ambition and does not stop until they overcome every challenge that this project presents. Everyone's CV is greatly enriched after including such a project on it.