Faculty of Computer and
Artificial Intelligence

Benha University

Epilepsy Seizures Detection Device

Computer science department

Project Team

1-Nadeen Ashraf

2-Mennatullah Hany

3-Hager Hamdy

4-Ghada Ahmed

5-Amaal Abdelazeem

6-Dina Yousry

7-Radwa Sayed

8-Alaa Ahmed

9-Karim Ehab

Under Supervision

Dr. Hamada Nayel Eng. Mohamed Mostafa

Contents

Chapter One

“Introduction”

Introduction

Epilepsy is a disorder of the central nervous system that causes brain activity to become abnormal.

Epilepsy seizure occurs when burst of electrical impulses escape their normal limits and spread to neighboring areas and create uncontrolled storm of electrical activity. Electrical impulses can be transmitted to muscles. The contraction process begins when electrical impulses reach at intervention zone. Intervention zone is neuromuscular junction.

Epilepsy Types:

Partial seizures: these seizures occur in one area of the brain and cause symptoms that are not dangerous.

Generalized seizures: these seizures occur in all parts of the brain causing symptoms that are dangerous.

Epilepsy seizures detection device detects three types of generalized seizures atonic, colonic, and myoclonic seizures as these types cause dangerous symptoms such as spasms that cause disability to control body that may lead to internal bleeding.

When the device detects seizure it sends alarm to patient's family and nearest hospital using mobile application.

We use **Electromyogram** (**EMG)** sensor to predict seizure by detecting muscle contraction.

An **EMG** sensor is a device that measures the electrical activity of muscles at rest and during contraction.

EMG captures action potential (electrical potential that muscles produce when they are activated neurologically or electrically) using two or more electrodes placed on IZ intervention zone to achieve better quality. IZ is in the middle of muscle.

There are two main types of EMG sensors:

Surface EMG sensors and intramuscular EMG sensors and we will use surface EMG.

1.1 Problem Description

When seizure occurs, it causes recurring spasms and movements due to muscle contraction which causes losing ability to control our body.

These spasms can lead to dangerous injuries such as: -

- Fractures are a common injury in people with epilepsy.
- Bruises.
- Physical injuries.
- Internal bleeding.
- Accidents.
- In some cases, if seizure duration is long, it causes death.

1.2 Current Solution

Magnetic Resonance Imaging (MRI) can show if there is any scar, tumor, or lesion in the brain that may be causing seizures.

An **Electroencephalogram (EEG)** can show irregular activity in the brain that can indicate seizures.

Seiz Alarm app for application watch but it was not preferred as it gives alarm of seizure for normal movements.

1.3 Proposed Solution

When the device detects seizure, it sends alarm to patient's family and nearest hospital using mobile Application.

It differentiates between normal movements and seizures using machine learning.

When device detects seizure and sends alarm, Application has option not seizure (false prediction), so device learns from feedback.

1.4 Project Scope

Project include device detects three types of generalized seizures atonic, colonic, and myoclonic.

Project exclude Partial seizures.

Chapter Two

“Planning”

Gant chart

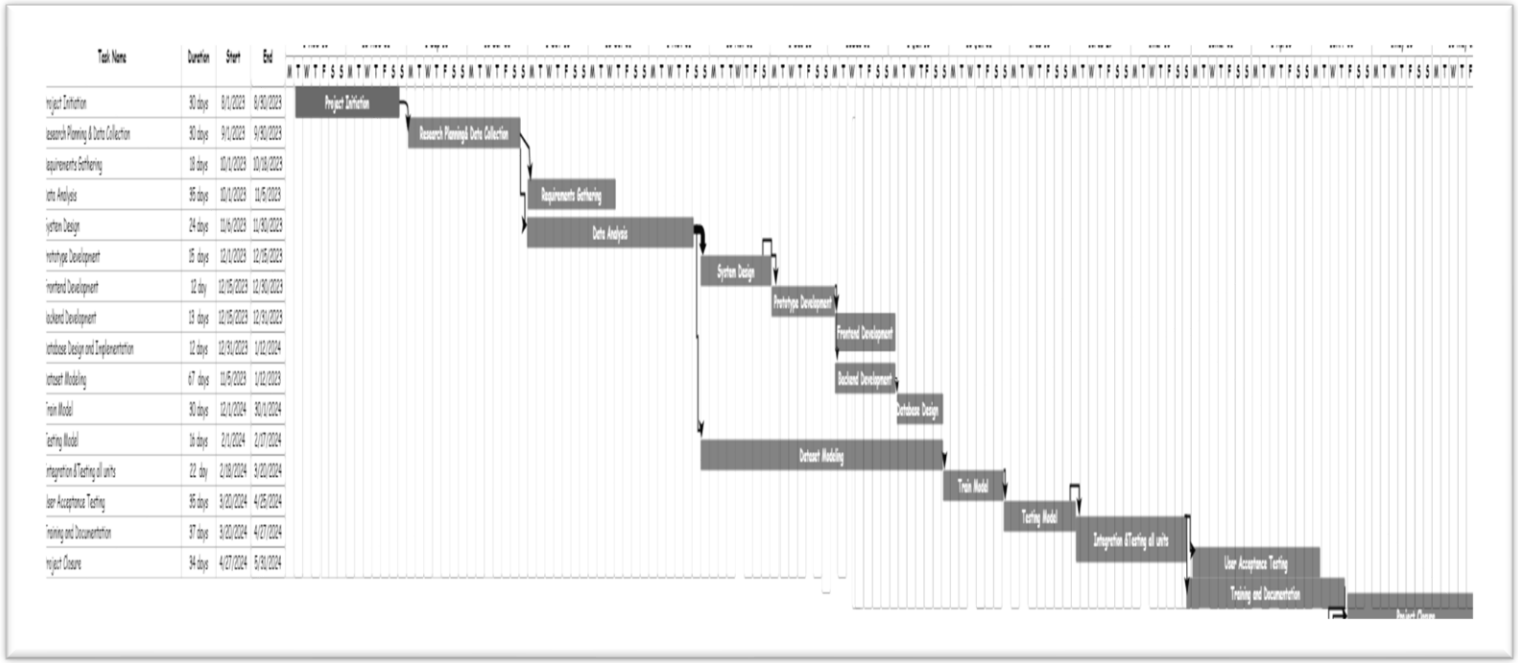


Figure 1:Gant Chart

n	Task Name	Duration	Start	End	Dependence
1	Project Initiation	30 days	8/1/2023	8/30/2023	
2	Research Planning & Data Collection	30 days	9/1/2023	9/30/2023	1
3	Requirements Gathering	18 days	10/1/2023	10/18/2023	2
4	Data Analysis	35 days	10/1/2023	11/5/2023	2
5	System Design	24 days	11/6/2023	11/30/2023	4
6	Prototype Development	15 days	12/1/2023	12/15/2023	5
7	Frontend and Backend Development	12 days	12/15/2023	12/30/2023	6
8	Database Design and Implementation	12 days	12/31/2023	1/12/2024	8
9	Dataset Modelling	67 days	11/5/2023	1/12/2024	5
10	Train Model	30 days	12/1/2024	30/1/2024	10
11	Testing Model	16 days	2/1/2024	2/17/2024	11
12	Integration &testing all units	22 days	2/18/2024	3/20/2024	12
13	User Acceptance Testing	35 days	3/20/2024	4/25/2024	13
14	Training and Documentation	37 days	3/20/2024	4/27/2024	13
15	Project Closure	34 days	4/27/2024	5/31/2024	15

Table1 : Main Tasks

2.1 User Requirements

1. Real-Time Monitoring:

The system should supply real-time monitoring to promptly alert caregivers or emergency services in case of a seizure to his parents, doctor, **911 Emergency**.

2. User Registration and Authentication & User Interface:

- User Profile: Users would be able to create and manage their profiles.
- Authentication: Secure login and authentication mechanisms.
- The **UI** would be user-friendly, especially for individuals who may have cognitive impairments during or after seizures.

3. Privacy and Security:

- Data Encryption: Ensure that all sensitive health data is encrypted during transmission and storage.
- If the user already has an account, he can login with his data name and email address.
- If the data is correct, go to his account with his data about his condition when it's bad or good.

4. Alert Mechanisms:

- Consider different alert mechanisms such as alarms, notifications on mobile devices, and even automated calls to caregivers or Emergency Services.
- Alerts must be easily distinguishable and not easily ignored to ensure the user's safety.

5. Cost:

Consider the affordability of the system to make it accessible to a wider range of users for the full charge of the sensors and other instruments.

6. Ease of Maintenance:

The system would be easy to maintain, with simple procedures for updating software, replacing batteries, and troubleshooting issues.

8. Portable and Wearable Device:

- Users may want a system that is portable and can be worn discreetly to maintain their privacy.

- The system should be lightweight, comfortable, and designed to be worn for extended periods.

9. Alert System:

- **Real-time Notifications** supply immediate alerts to relatives, doctors, and emergencies when a seizure is detected.

- **Vibration or Sound Alerts** allow users to choose between vibration, sound, or both for notifications. If a user has seizure many times monthly warning him with messages to take.

2.2 Functional Requirements

- Provide a unique identification for each user for login.
- Send an alarm to patient`s family when a seizure occurs.
- Provide stop alarm option for fault tolerance.
- Provide advice about how to deal with the patient when seizure occurs.
- Provide information about how many times seizure occurs per week or month.

2.3 Non-Functional Requirements

- **Availability:**
  - The system should be available all time.
- **Reliability:**
  - The outcome is dependable and true.
  - The system must be without errors or crashes.
- **Security:**
  - System should have high security.
- **Usability:**
  - Application must be user friendly, simple, and interactive.
  - The user interface is easy to use.
  - The device is easy to use and comfortable.

2.4 Methodology

Software Requirement:

- **Programming Languages**: Proficiency in programming languages such as **Python**, **C**, or depending on the software stack and microcontroller used.
- **Machine Learning**: Knowledge of frameworks like **scikit-learn** or **TensorFlow**.
- **Signal Processing**: Skills in signal processing for extracting relevant features from physiological signals. Knowledge of libraries like **SciPy** and Biopsy for signal processing in Python.
- **Embedded Systems Programming:** Familiarity with embedded systems programming for microcontrollers (e.g., **Arduino, ESP32, Raspberry Pi)** to implement real-time algorithms.
- **Data Analysis**: Proficient in data analysis techniques, including statistical analysis and visualization, to analyze and interpret sensor data.
- **Real-Time Processing**: Understanding of real-time processing requirements for wearable devices, including minimizing latency and optimizing algorithm efficiency.
- **Mobile App Development**: Wearable device communicates with a mobile app, skills in mobile app development (**Flutter**).

Hardware Requirement:

- **Electronics and Circuit Design**: Understanding of electronics and circuit design for developing the hardware components of the wearable device.

- **Microcontroller Programming**: Proficiency in programming microcontrollers (e.g., Arduino, ESP32, Raspberry Pi) to interface with sensors, process data, and control the device.
- **Sensor Integration**: Knowledge of integrating and interfacing various sensors (EMG sensor, EDA sensor) with the microcontroller.
- **Power Management**: Skills in power management for optimizing energy consumption and selecting suitable power sources for wearable devices.
- **Wireless Communication**: Understanding of wireless communication protocols (e.g., Bluetooth, WIFI) for data transmission between the wearable device and Mobile App.
- **Prototyping**: Ability to create physical prototypes of the wearable device for testing and iteration.
- **Embedded Systems**: Understanding of embedded systems principles for designing devices that operate with limited resources.
- **Hardware Debugging**: Skills in debugging hardware issues, including troubleshooting connections, sensor malfunctions, or power-related problems.

2.5 Software

- Operating System: Windows 10 or higher Platform:
- IOT Cloud IDE
- Arduino IDE
- Anaconda IDE
- Android Studio IDE

2.6 Hardware

- Microcontroller: ESP32
- EMG Sensors
- WI-FI Module
- Amplifier
- Signal processing unit.
- Charging module
- Lithium batteries

2.7 Cost Estimating & Budget.

Identify the budget amount allocated by key budget category (e.g., project milestone or standard cost categories such as personnel, Materials), including the period that may constrain use of the budget Total budget for project 5000 L.E.

- Hardware: - 5000 L.E.

Chapter Three

“System Analysis”

3.1 Use case Diagram

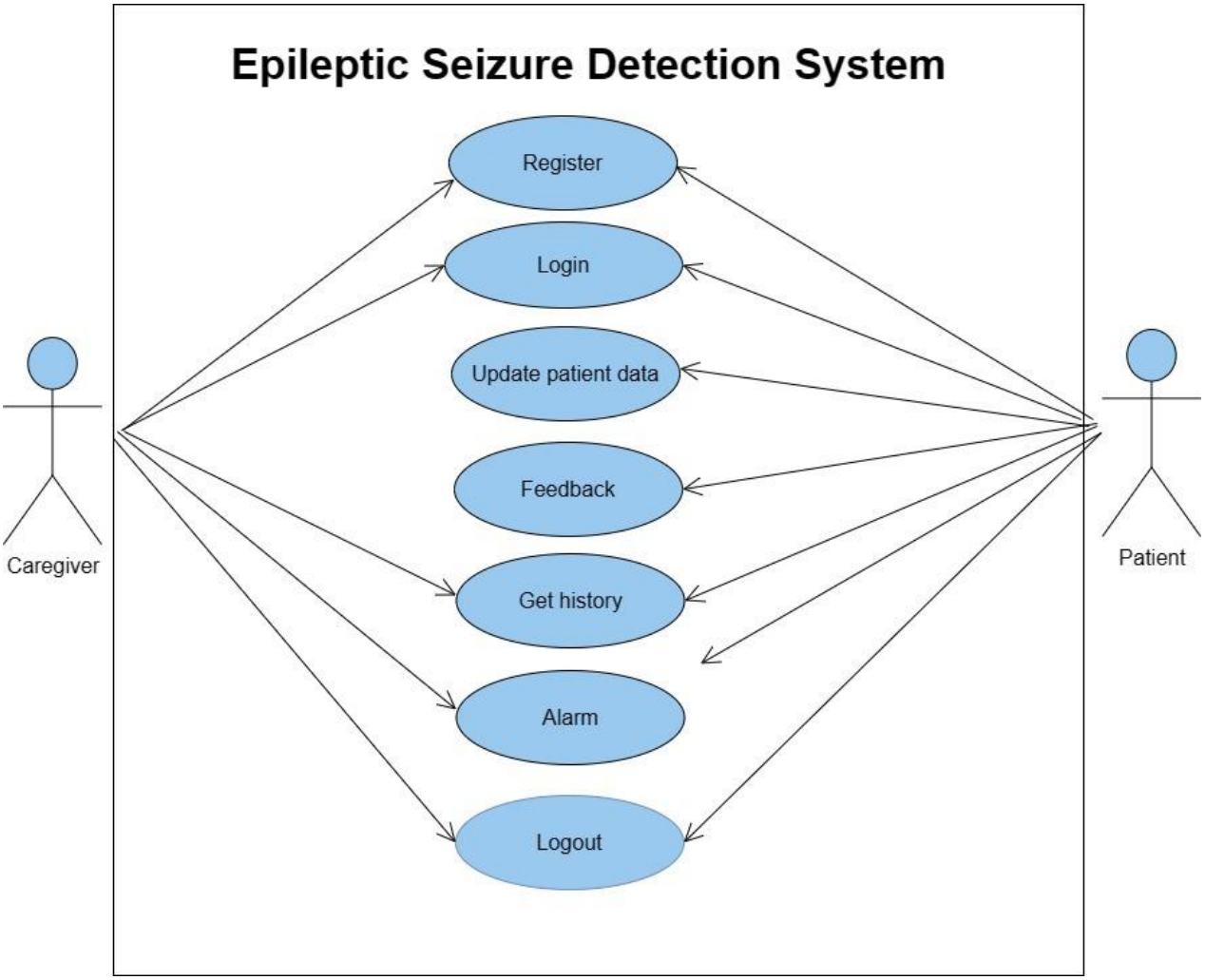


Figure 2: Use case Diagram.

3.2 Activity Diagram

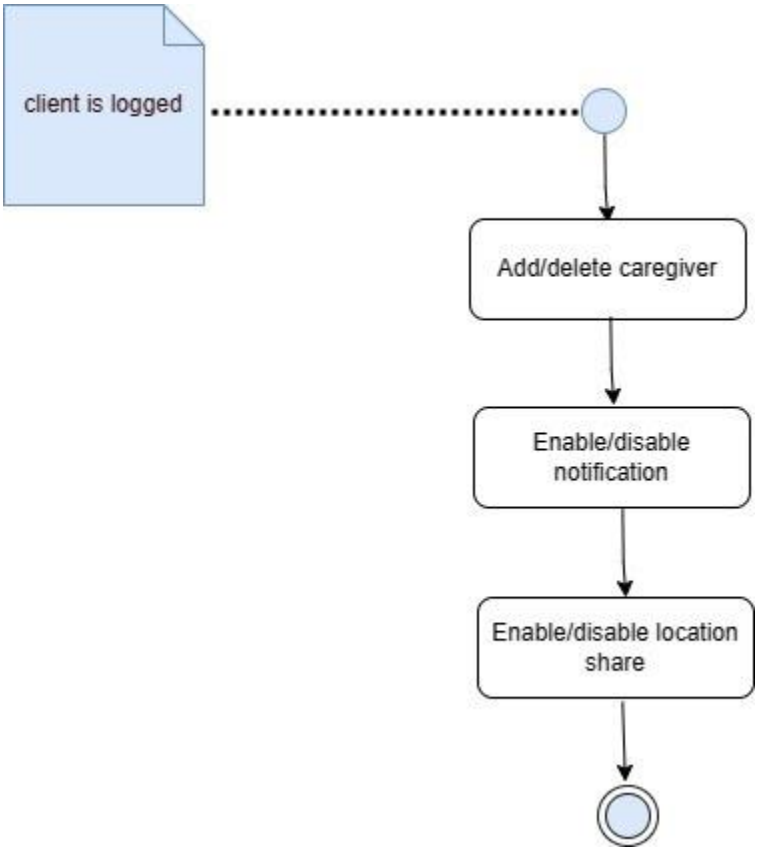


Figure 3: Activity Diagram 1

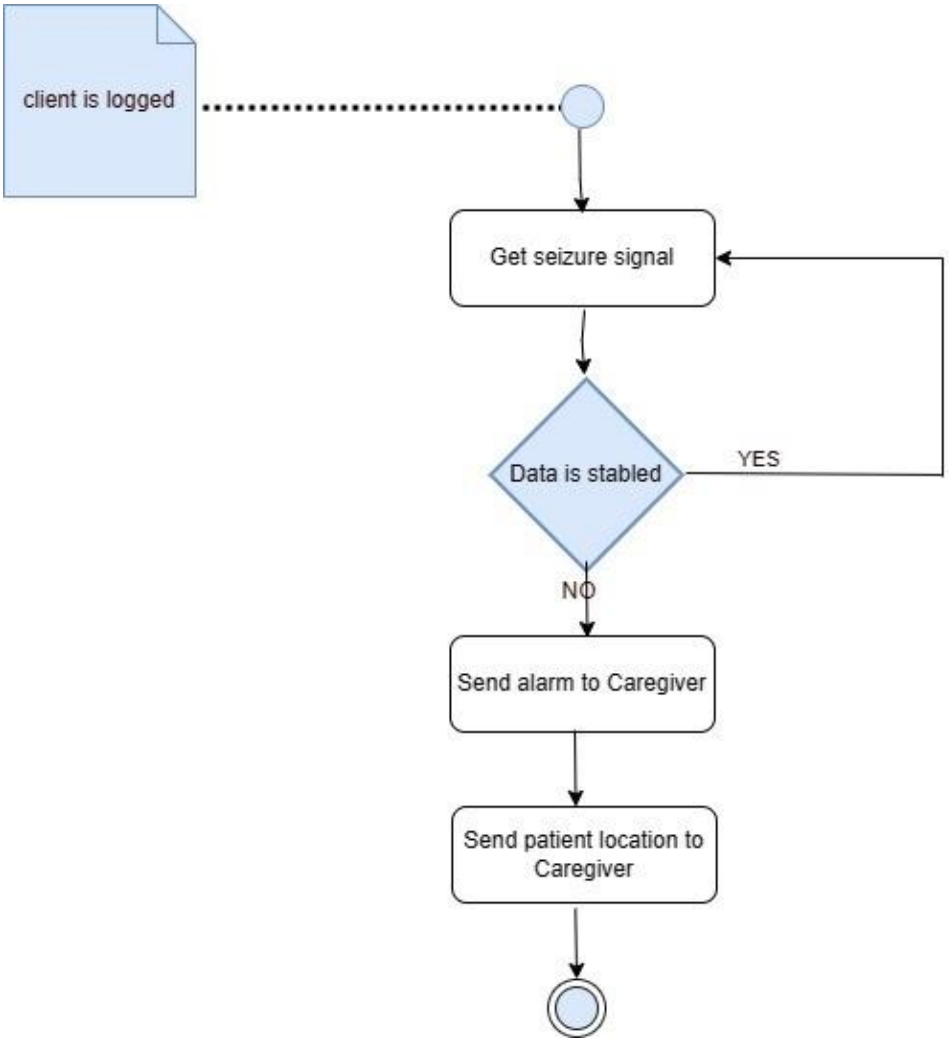


Figure 4: Activity Diagram 2

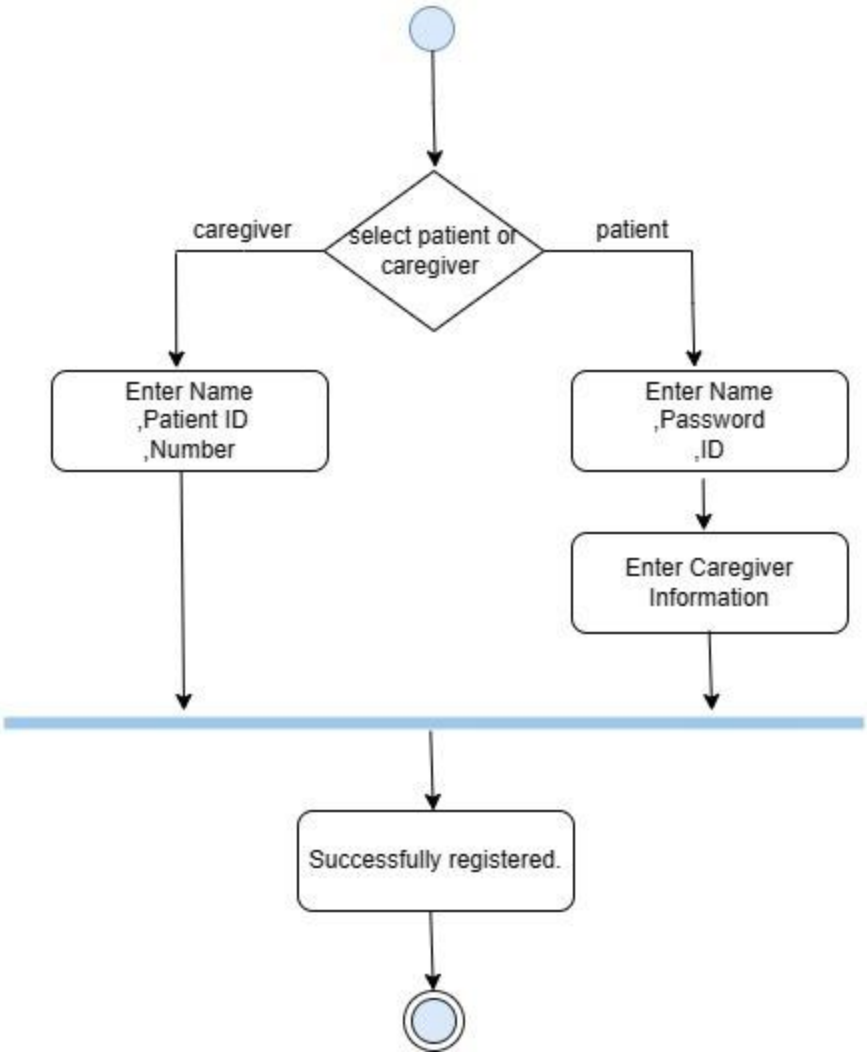


Figure 5: Activity Diagram 3

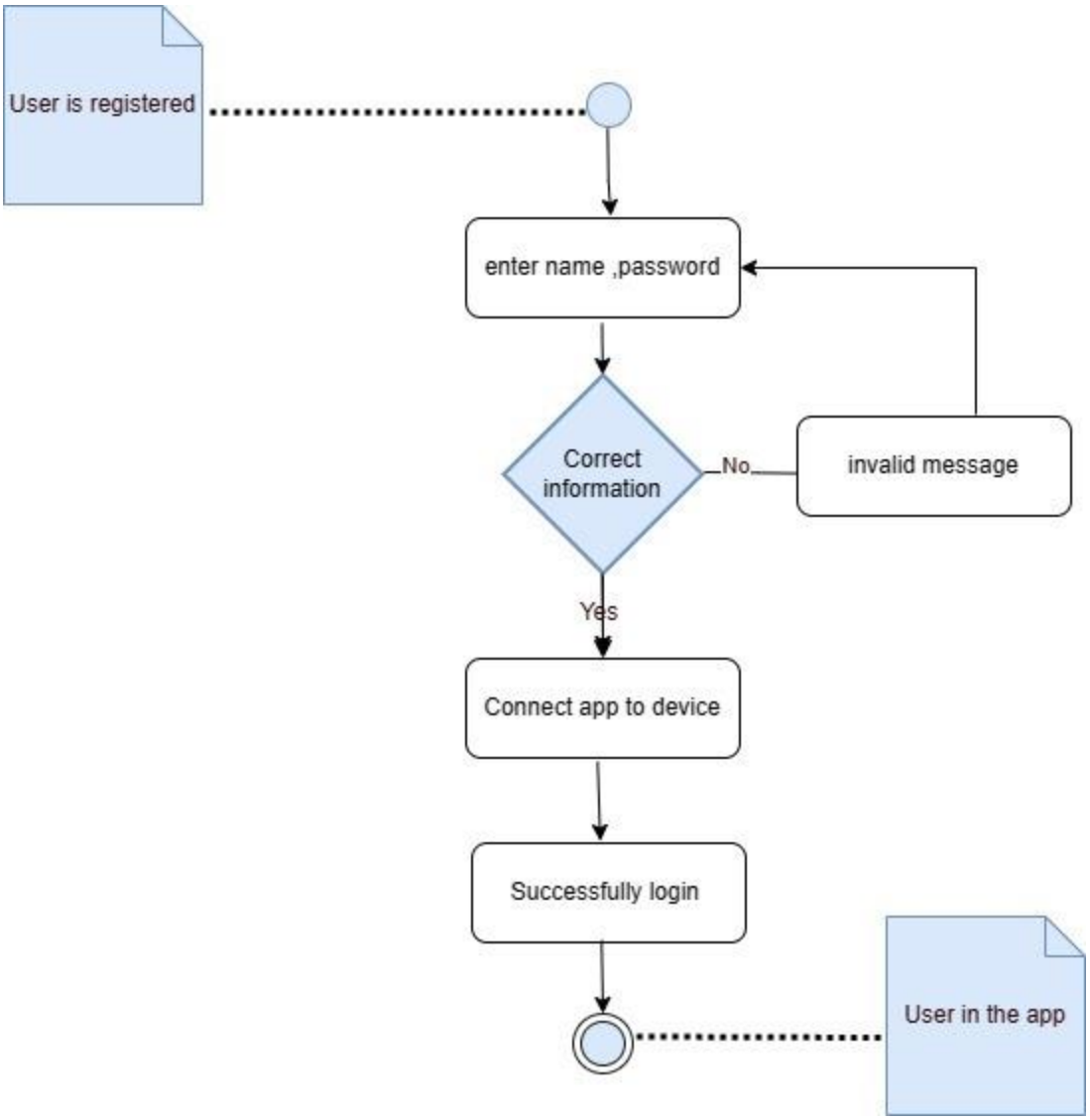


Figure 6: Activity Diagram 4

3.3 Class Diagram

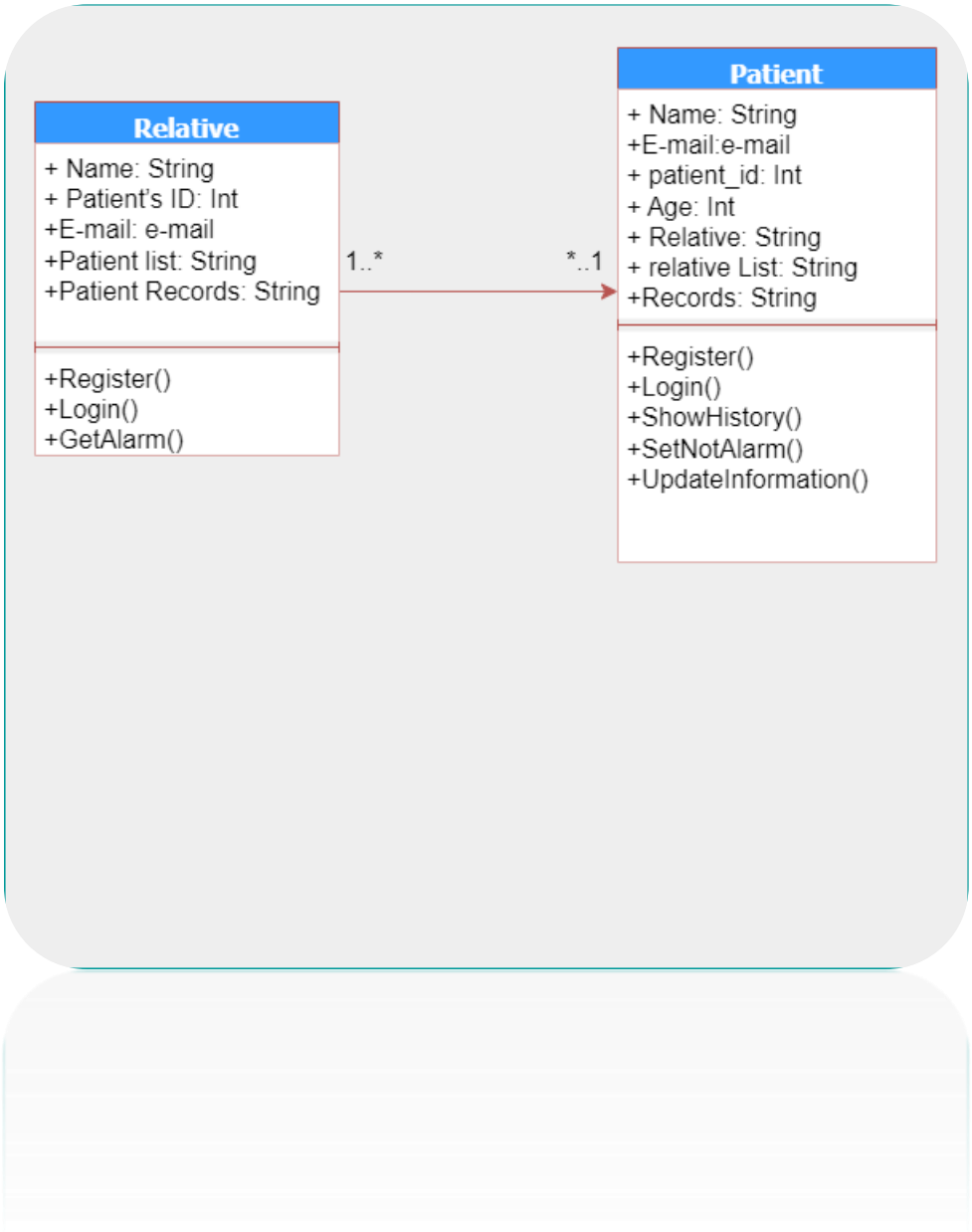


Figure 7: Class Diagram

3.4 Sequence Diagram

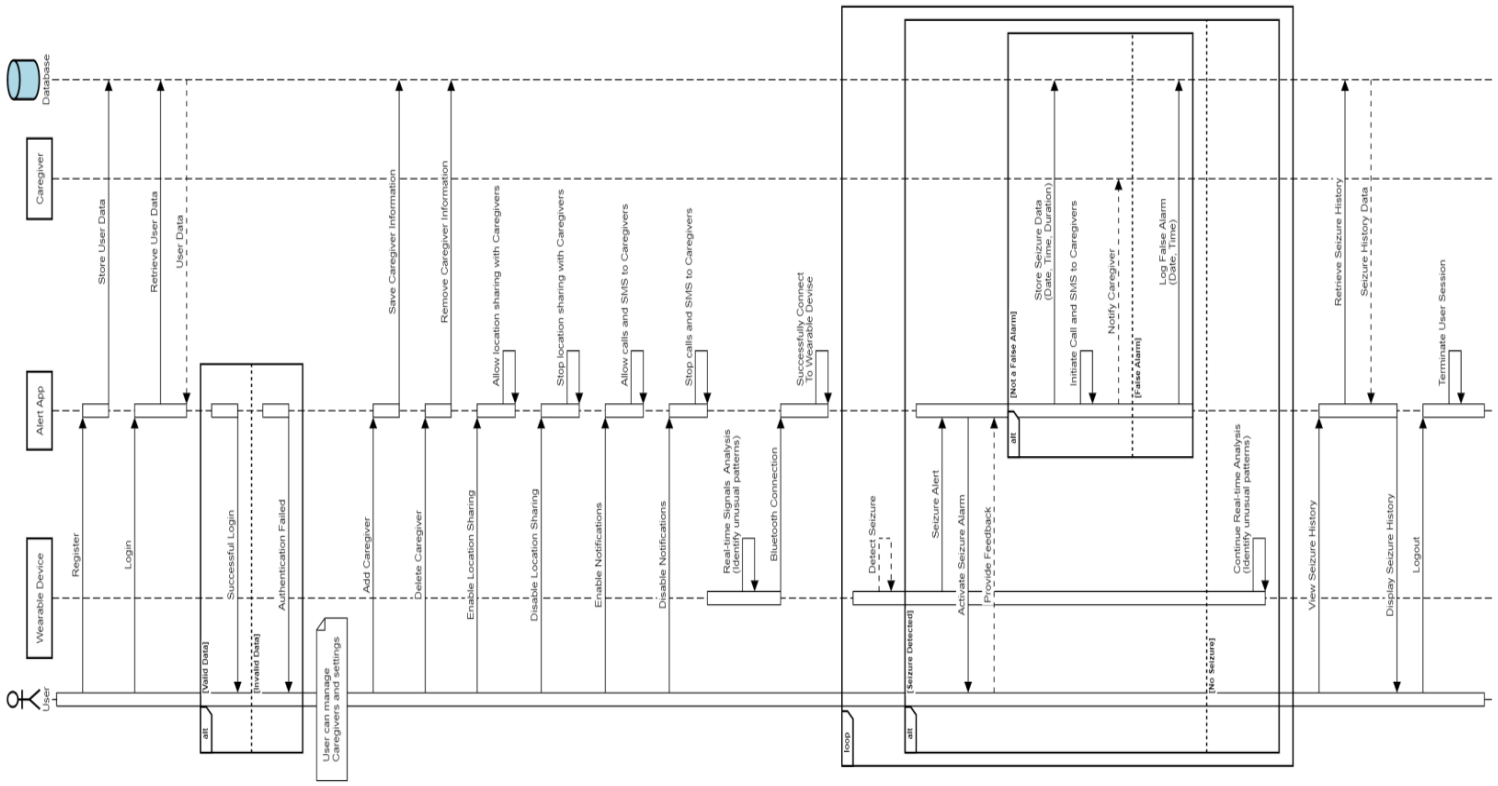


Figure 8: Sequence Diagram

3.5 Block Diagram

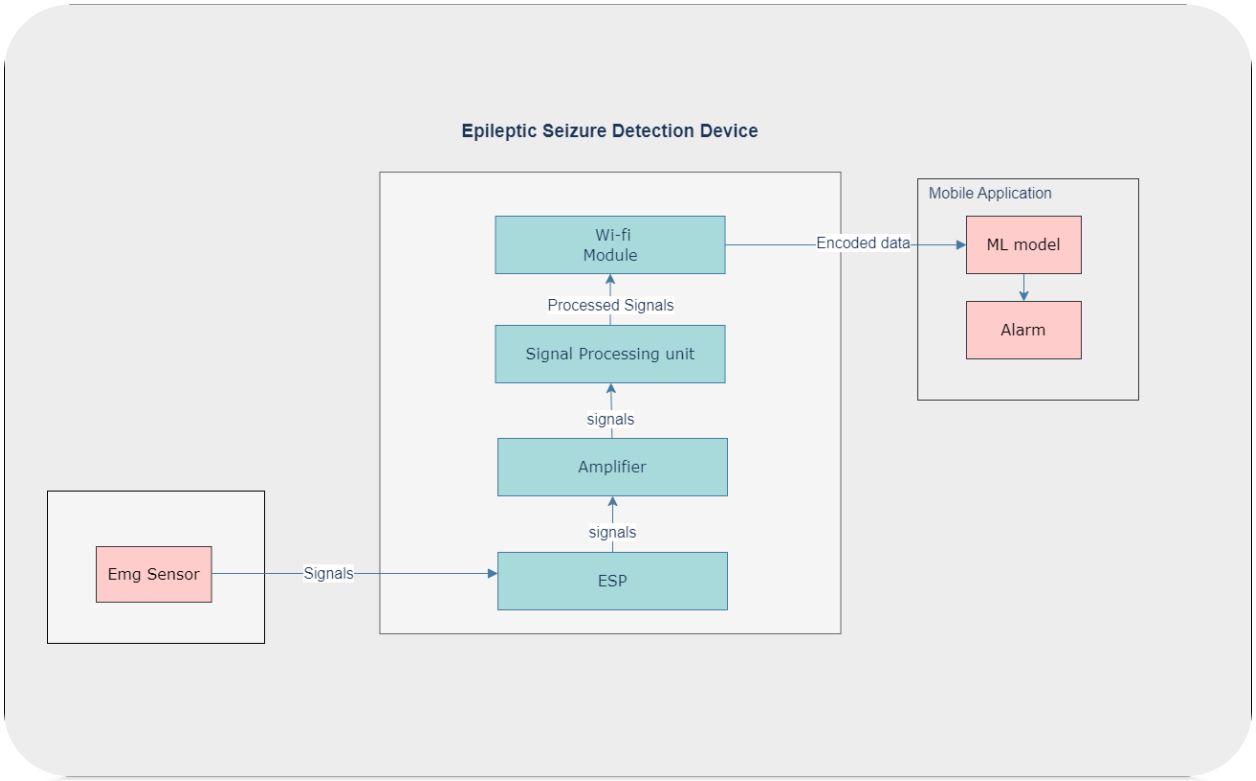


Figure 9: Block Diagram

3.6 DFD Diagram

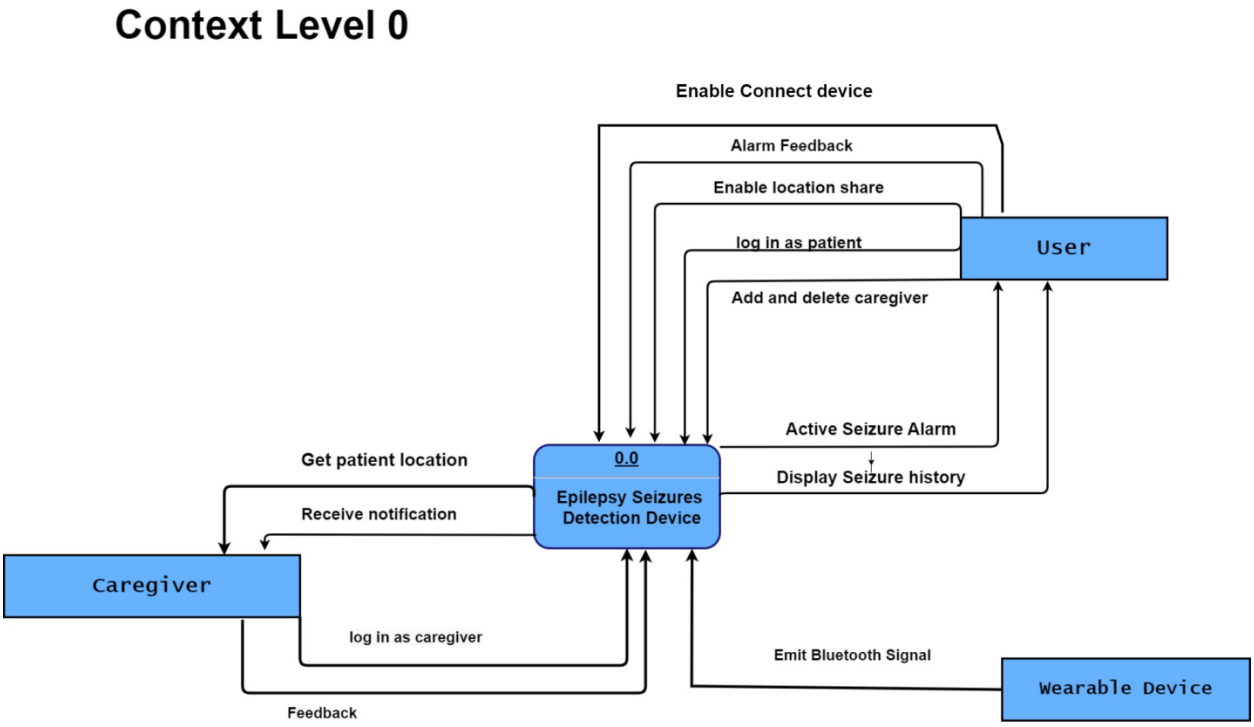


Figure 10: Context Diagram

Context Level 1

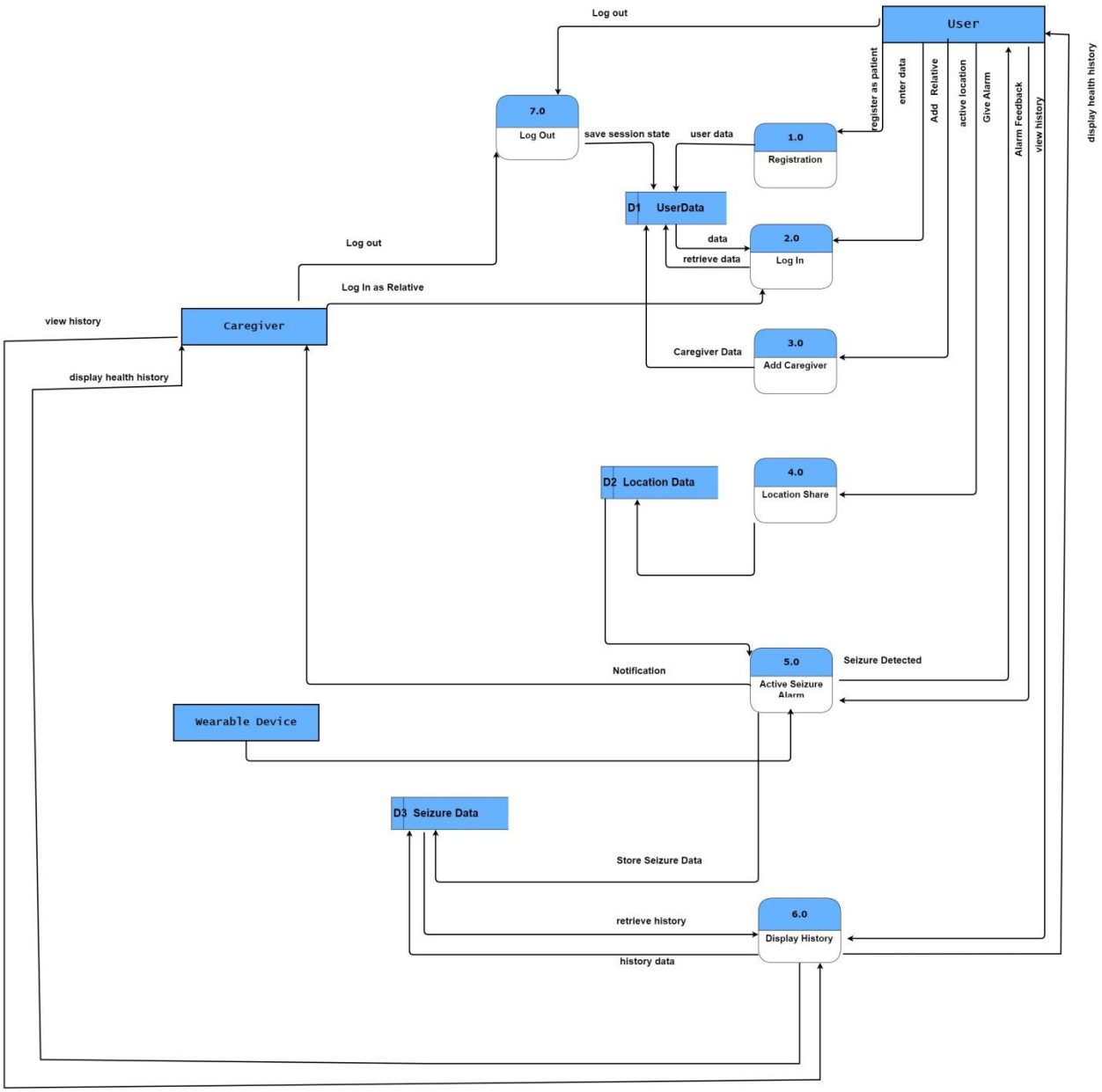


Figure 11: DFD Level 1 Diagram

3.7 ERD Diagram

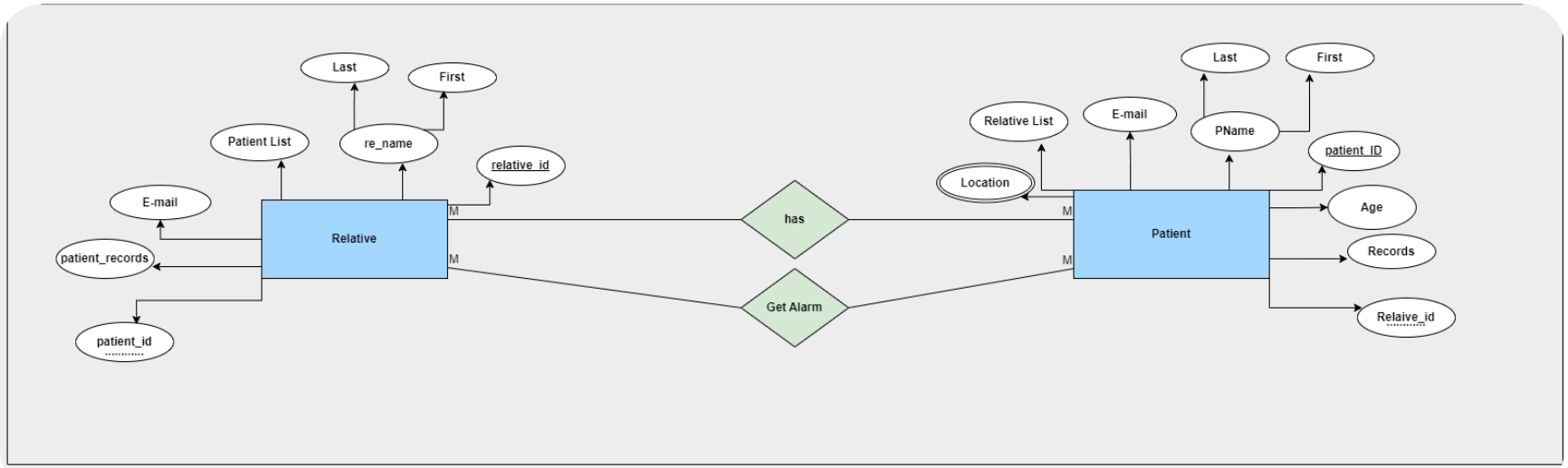


Figure 22: ERD Diagram.

Chapter Four

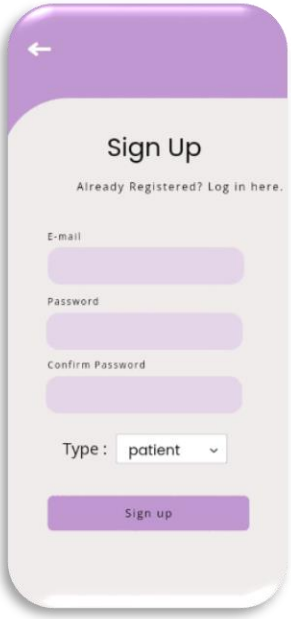
“Prototype”

4.1 Common Pages

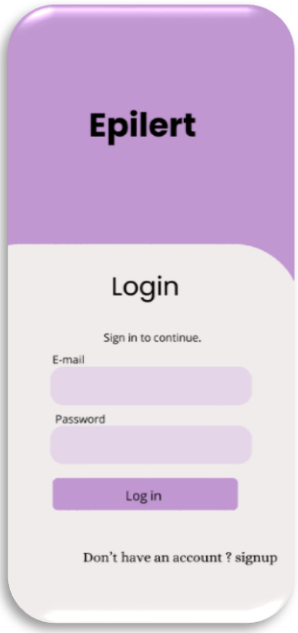
These are the mutual pages between patient and relative.

Home Page

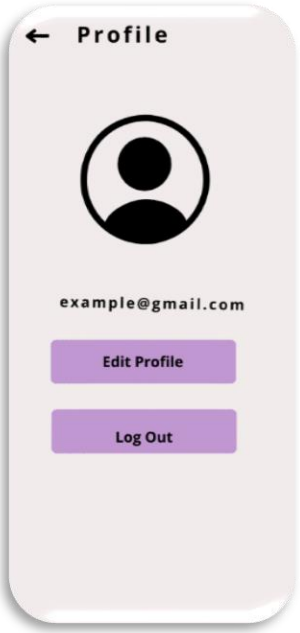
Sign Up Page



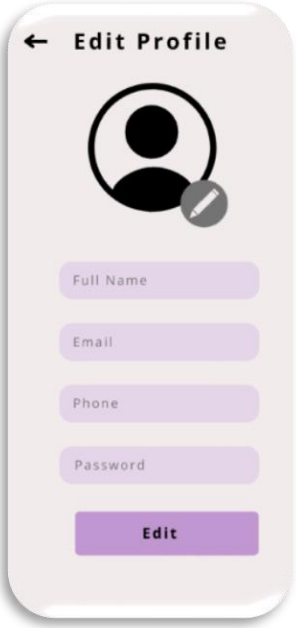
Login page



Profile page

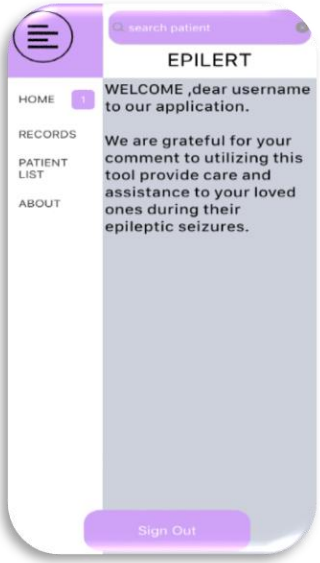


Edit Profile Page

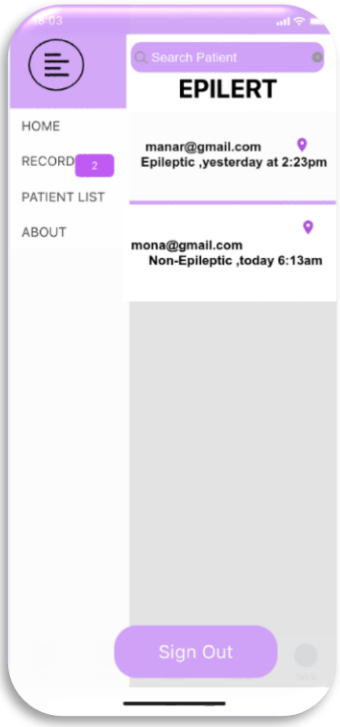


4.2 Relative Pages

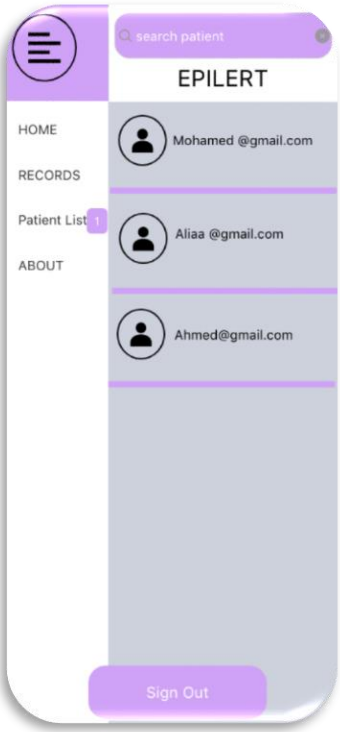
Home Page



Records Page



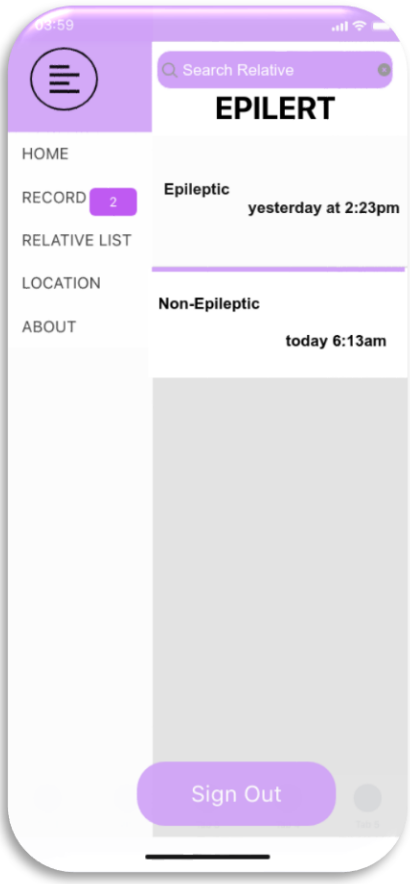
Patient List Page



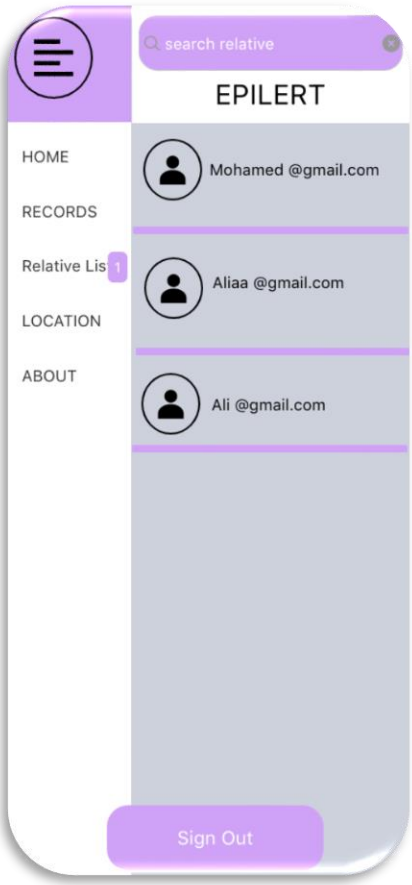
4.3 Patient Page

Home Page

Records Page



Relative List Page



Chapter Five

“Tools Required and Methodology”

5.1 Technologies

The Internet of Things (IOT) is a system of interrelated computing devices, people. That are marked by unique identifiers and can transfer data over a network. The Mobile apps connect to multiple smart devices through Wi-Fi. The focus is made on such modules, so that the connectivity is simple and easy to use. To communicate with IOT devices like sensors and actuators. Deep Learning Models that manipulate and process the data received from the sensor.

The information gained from the Model is displayed in the application. The application sends an alarm if seizure detected.

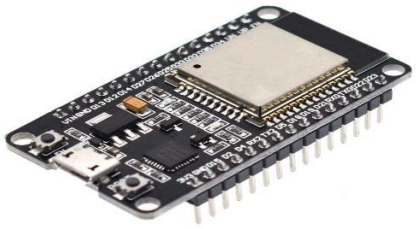
5.2 Tools

- ESP32 microcontroller
- 9V batteries and connectors
- Rechargeable lithium polymer battery
- Jumper Wires
- Breadboard
- Micro USB Lithium Battery Charging Module TP4056 with Battery Protection
- EMG sensor kit
- Micro USB cable
- Hand band

5.2.1 ESP32 microcontroller

The ESP32 is a microcontroller that is designed to provide wireless connectivity and processing power for a wide range of applications. It is based on the Xtensa LX6 processor, which is a dual-core 32-bit processor with a clock speed of up to 240 MHz's,448 KB ROOM,520 kB SRAM,16 KB SRAM NRTC.

The ESP32 also includes a variety of peripherals, such as Wi-Fi, Bluetooth, and Ethernet interfaces, as well as GPIO pins and Analog-To-Digital converter.



5.2.2 9V batteries

Primary battery 9 V, capacity 190 mA.

Design = flat

Type = 9V block

Dimensions = 25.5 x 48.5 x 16 mm

Manufacturer = Westinghouse

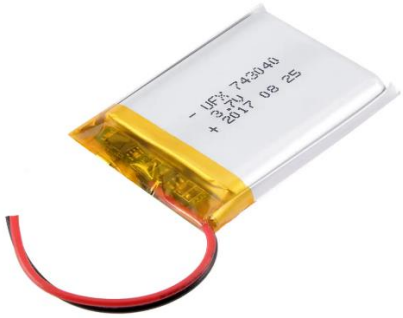
Sold by 1 pc = 1 battery.

5.2.3 Rechargeable lithium polymer battery

These are very slim, extremely light weight batteries based on the new Polymer Lithium-Ion chemistry. This is the highest energy density currently in production. Each cells outputs a nominal 3.7V at 1200mAh & Dimensions.

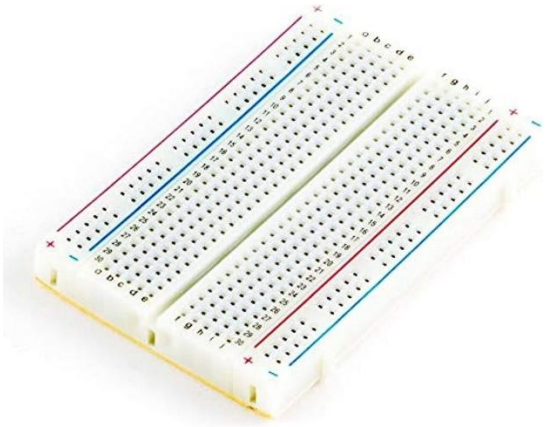
These batteries require special charging. Do not attempt to charge these with anything but a specialized LiPo charger.

Battery includes built-in protection against over voltage, over current, and minimum voltage.



5.2.4 Breadboard and Jumper Wires

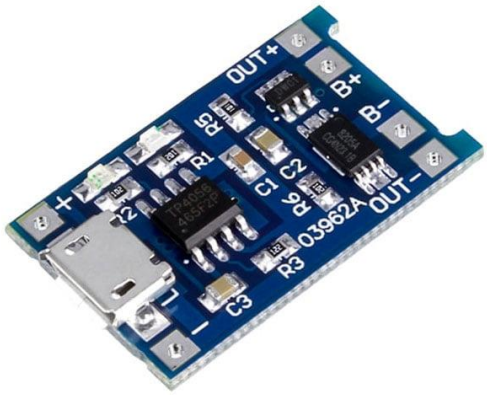
A breadboard is a construction base used to build semi-permanent prototypes of electronic circuits. Unlike a Perfboard or stripboard, breadboards do not require soldering or destruction of tracks and are hence reusable.



5.2.5 Micro USB Lithium Battery Charging Module TP4056 with Battery Protection

This TP4056 1A Li-Ion Battery Charging Board Micro USB with Current Protection is a tiny module, perfect for charging single cell 3.7V 1 Ah or higher lithium ion (Li-Ion) cells such as 16550s that don't have their own protection circuit. Based on the TP4056 charger IC and DW01 battery protection IC this module will offer 1A charge current then cut off when finished.

Furthermore, when the battery voltage drops below 2.4V the protection IC will switch the load off to protect the cell from running at too low of a voltage – and protects against over-voltage and reverse polarity connection (it will usually destroy itself instead of the battery).

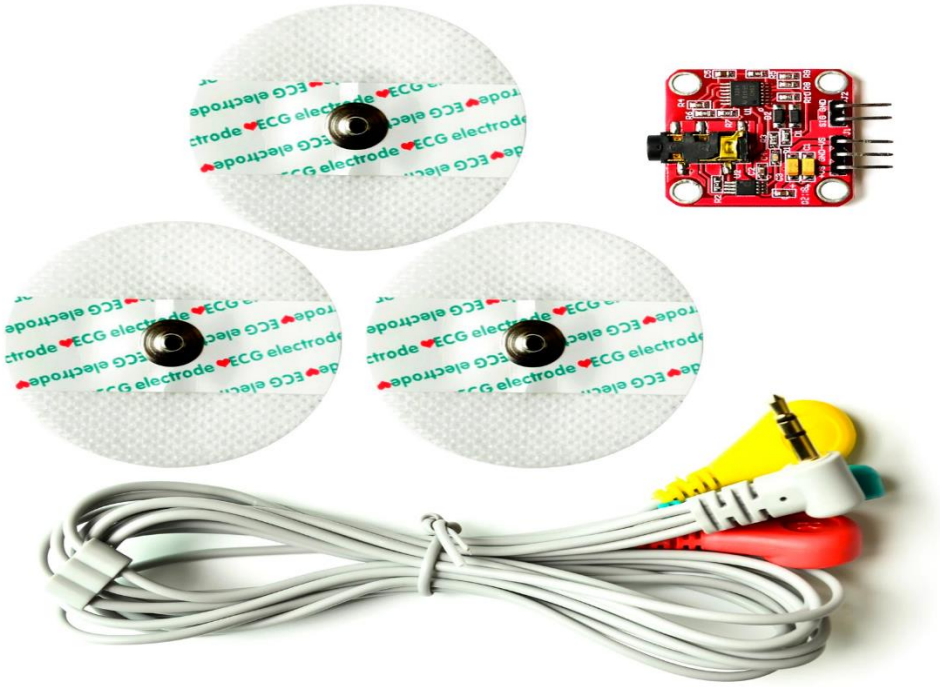


5.2.6 EMG sensor kit

The Advance Technologies EMG Muscle Sensor with Cable and Electrodes will measure the filtered and rectified electrical activity of a muscle outputting 0-Vs Volts depending on the amount of activity in the selected muscle, where Vs signifies the voltage of the power source. Power supply voltage: min. +-3.5V.

This Muscle Sensor from Advancer Technologies measures**,** filters, rectifies, and amplifies the electrical activity of a muscle and produces an Analog output signal that can easily be read by a microcontroller, enabling novel, muscle-controlled interfaces for my project.

Sensor has 3 (Green, Red and Yellow) electrodes connect to the module for transmitting electrical signals from the muscle motion.



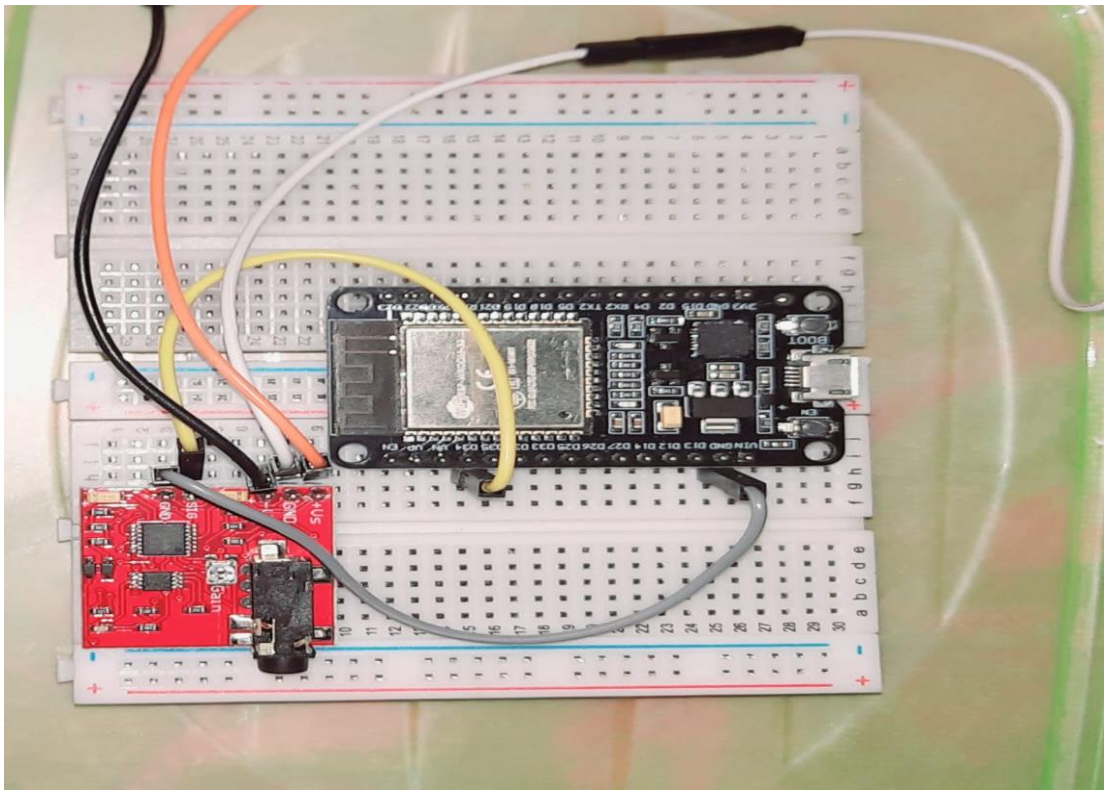
5.2.7 Micro USB Cable and Headband

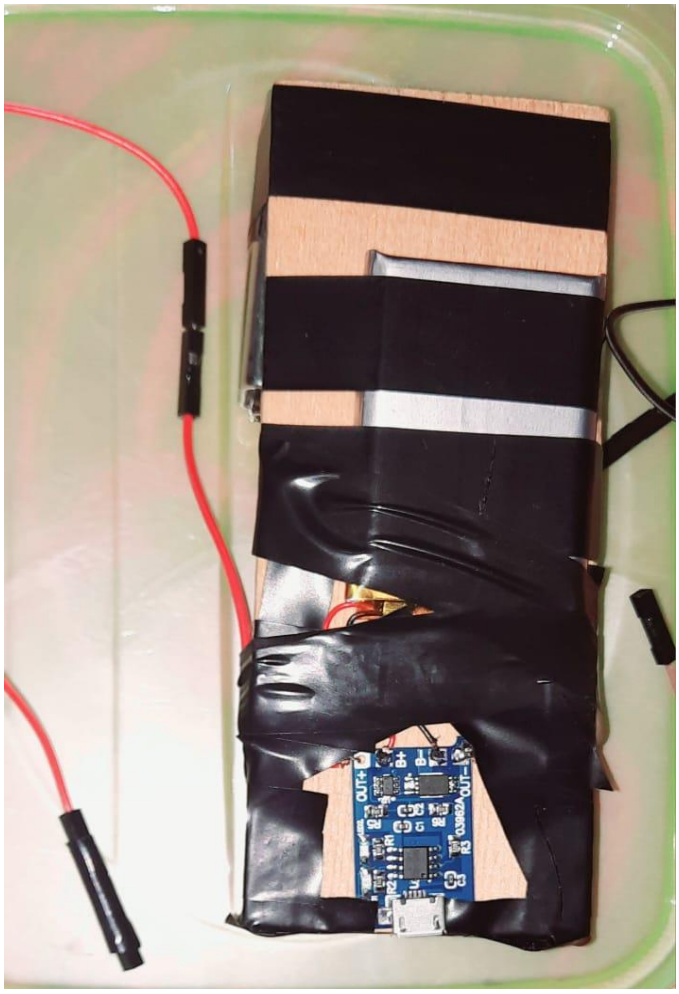
Chapter Six

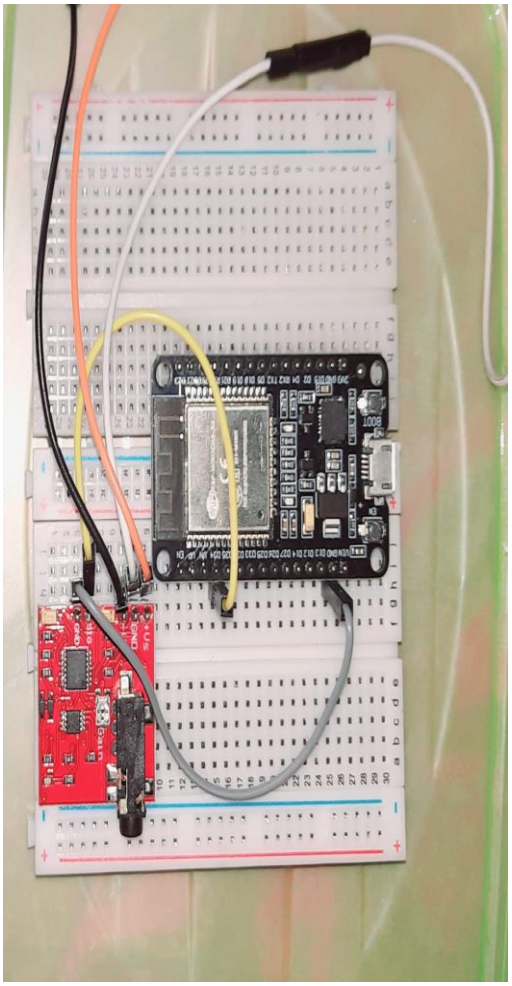
“Implementation”

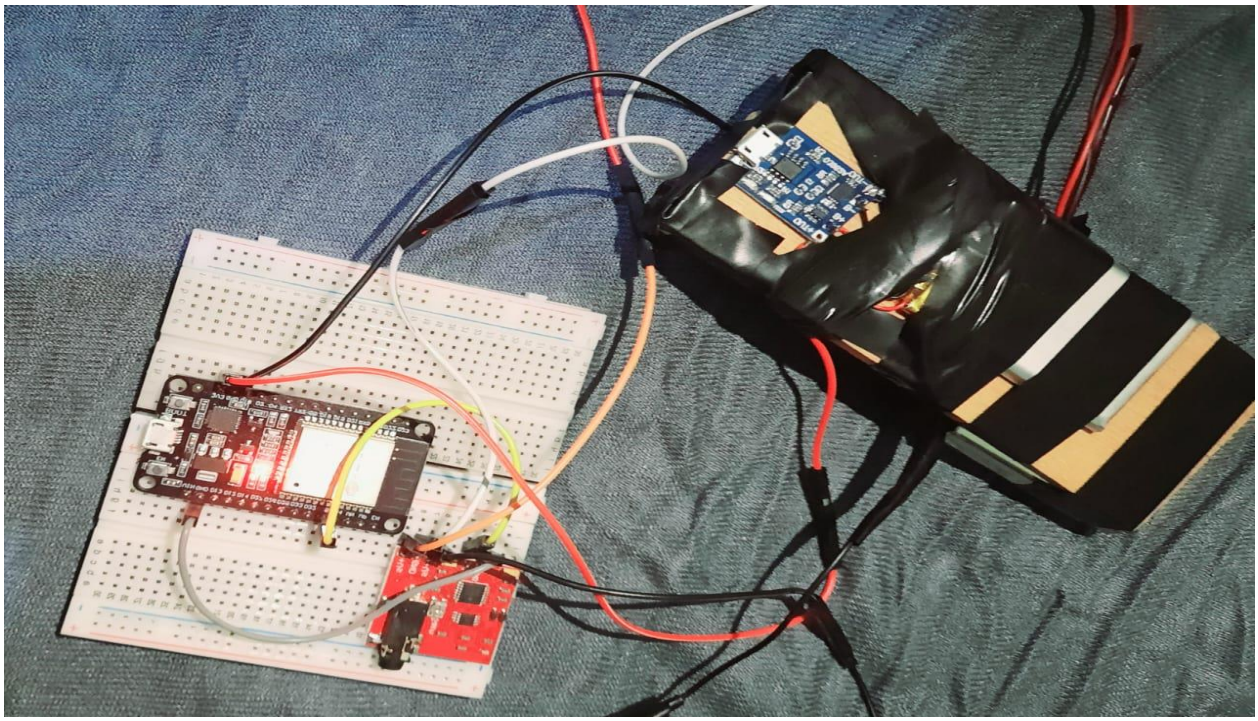
5.1 Hardware

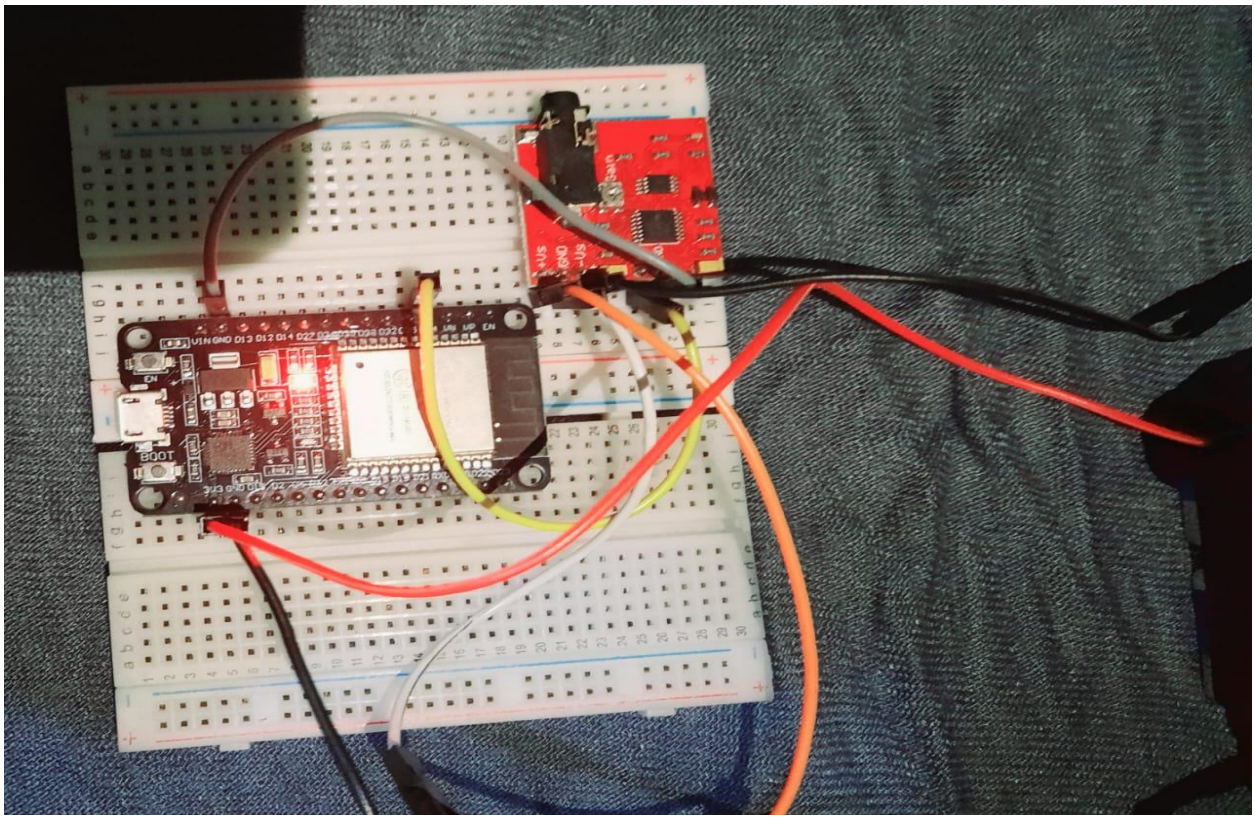
5.1.1 ESP32 – Microcontroller and EMG sensor

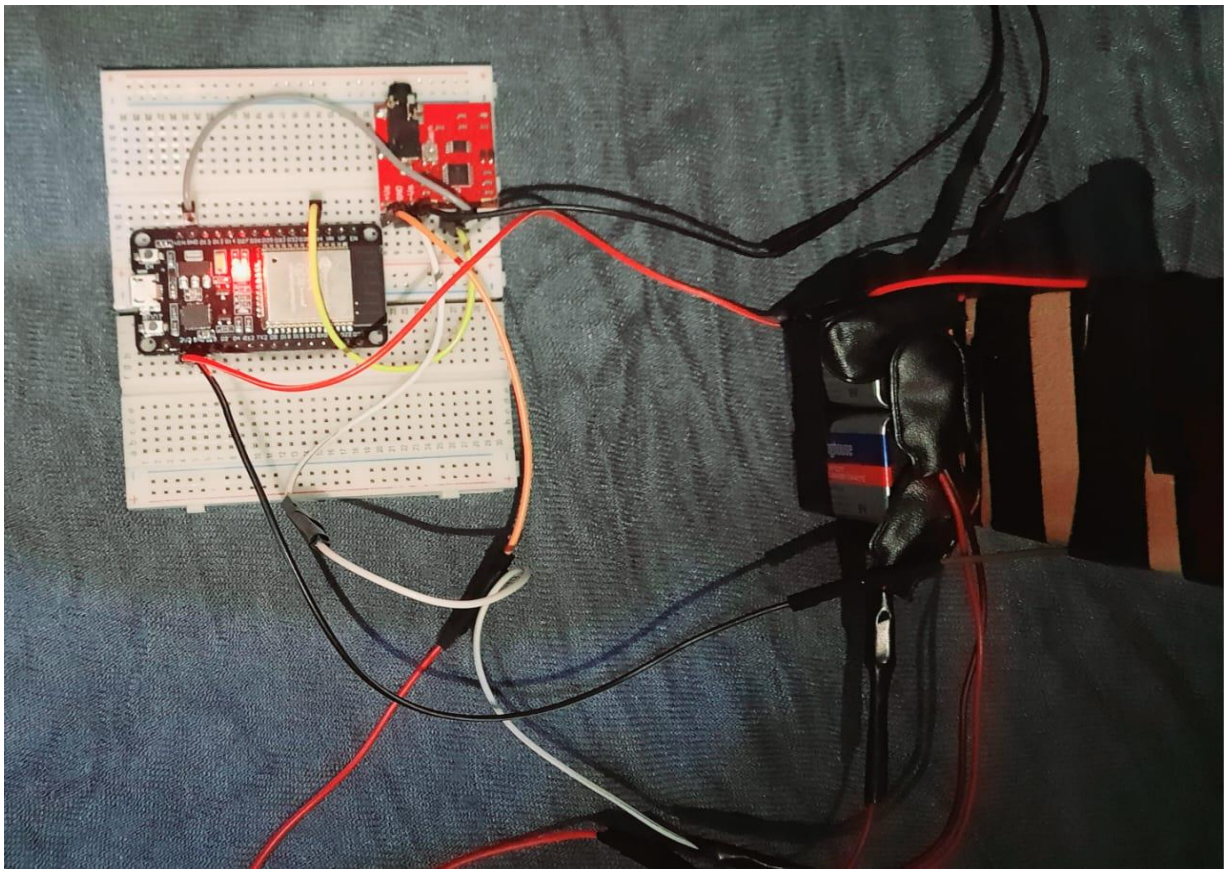






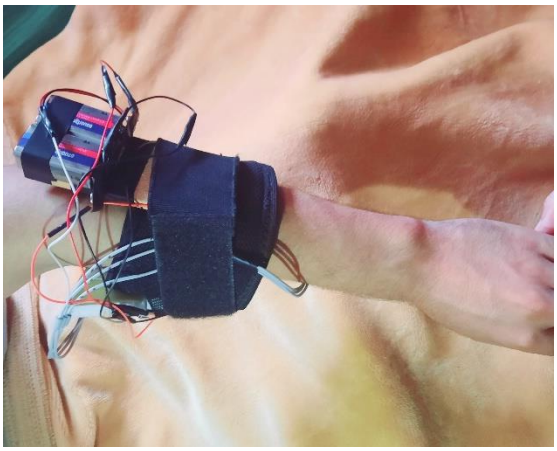


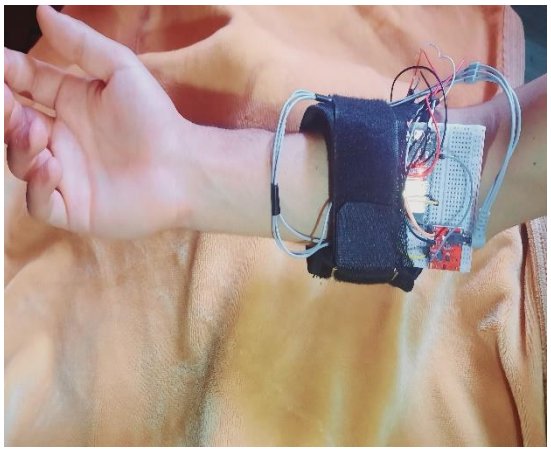




1) Connect ESP32 to the breadboard.
2) Connect EMG sensor to the breadboard.
3) Connect the positive terminal of the first 9V battery to the +Vs pin on your sensor.
4) Connect the negative terminal of the first 9V battery to the positive terminal of the second 9V battery. Then connect to the GND pin on your sensor.
5) Connect the negative terminal of the second 9V battery to the –Vs pin of your sensor.
6) Connect the GND pin of your sensor to a GND pin on the ESP32.
7) Connect the SIG pin of your sensor to an analog pin (D34). on the ESP32.
8) Connect ground of the lithium battery to the GND of the ESP32
9) Connect +3V of the battery to +3V of the ESP32.
10) Connect the two 9V batteries in serial connection to get 18V for the EMG Sensor

5.1.2 Final Result





5.2 Mobile Application

There are common screens between relative and patient, ex: sign up, login, profile, about us, advice, and feedback.

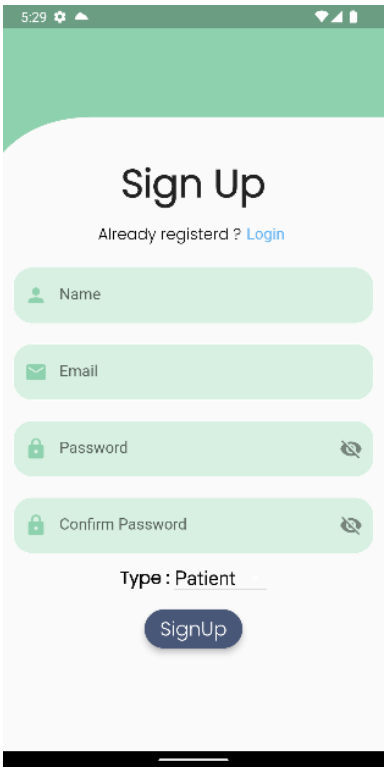
We used Nilsons'8 golden rules to maintain usability and user experience with the application.

5.2.1 Common screens

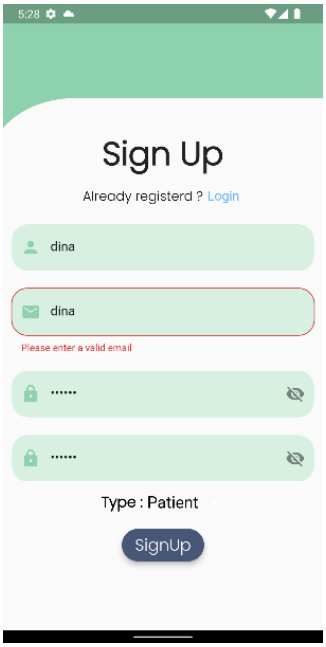
Sign up.

Users can choose between a patient or relative before signing up.

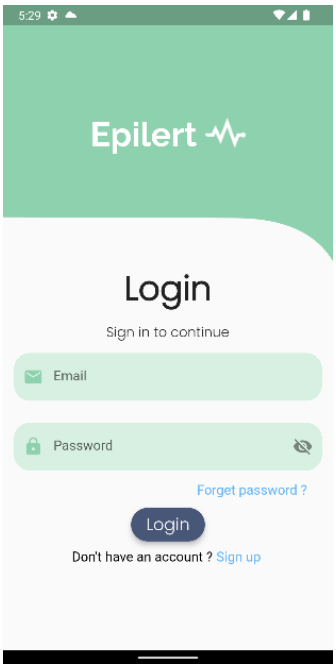
Here we used some rules to keep user engagement with the application.



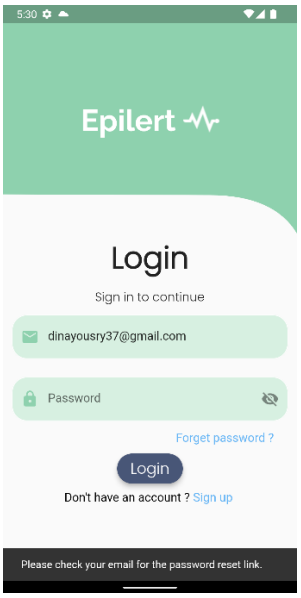
1-prevent errors. 2-strive for consistency.



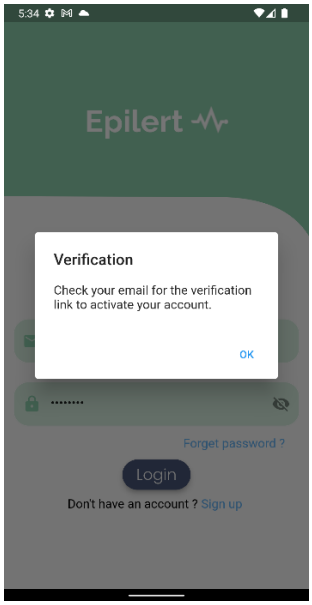
Login.



1-give formative feedback.



2-Design dialogs to yield closure.



Advice.

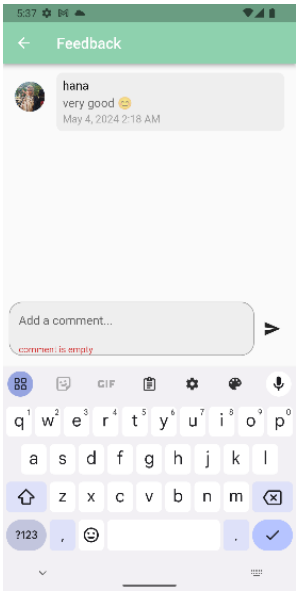
Here patients and users can have some guidance about how to deal with epilepsy seizure and how to avoid it.

Feedback.

Here users can give their opinions about

the application and give us ideas to improve usability**.**

1-pervent errors.



About us.

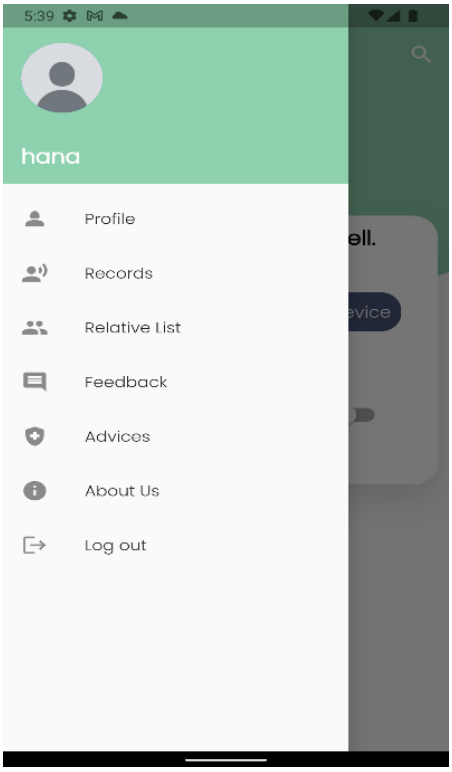
Here users can know a brief about us.

5.2.2 Patient Screens

Screens that are available to patients like drawer lists, connect device, record patient, relative list, and search relative.

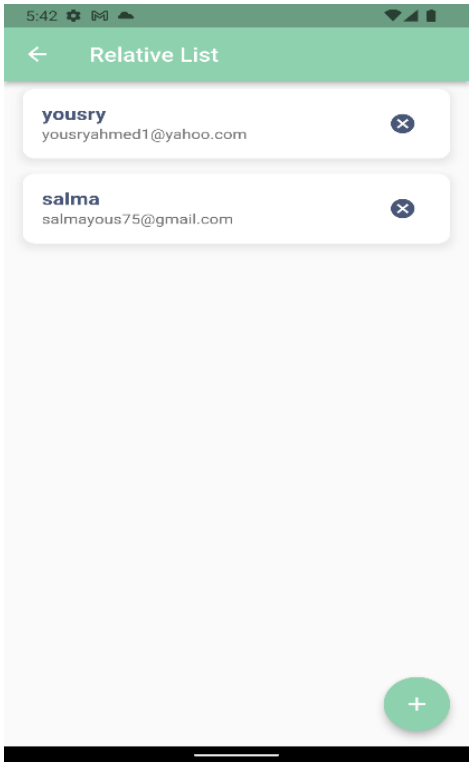
Drawer List.

There are some options on the list.



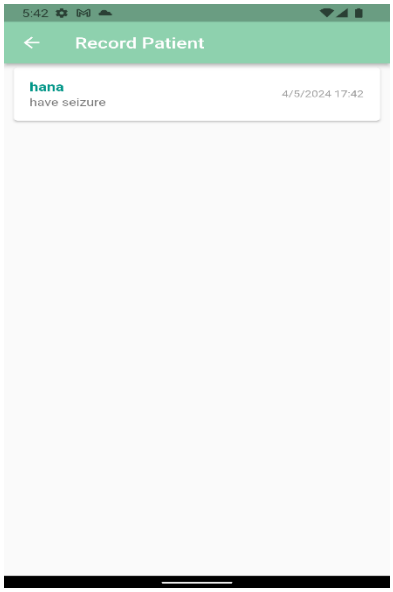
Relative List.

There are relatives that patient added



Record Patient.

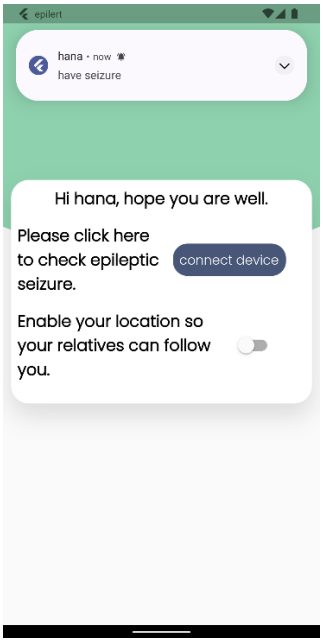
In which seizure date and details to help doctor in diagnoses.



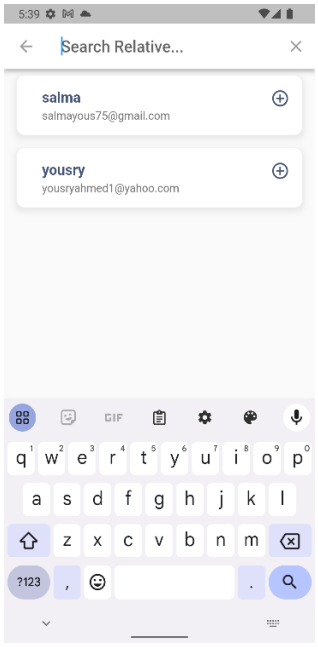
Connect device.

1-keep users in control.

2-Offer informative feedback.



Search Relative.

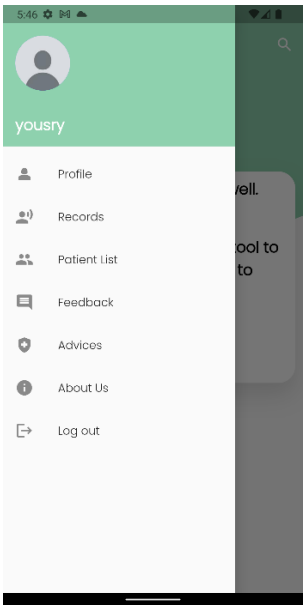


5.2.3 Relative Screens

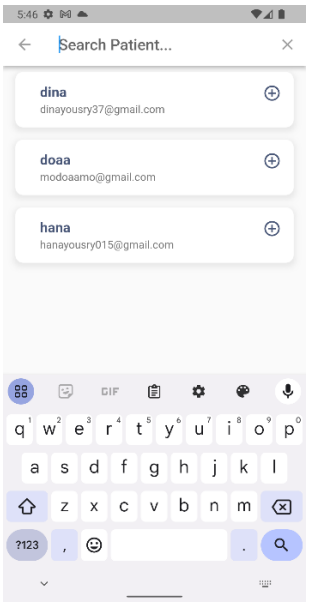
Screens that available to Relative like drawer list, record patient, Patient list, and search patient.

Drawer List.

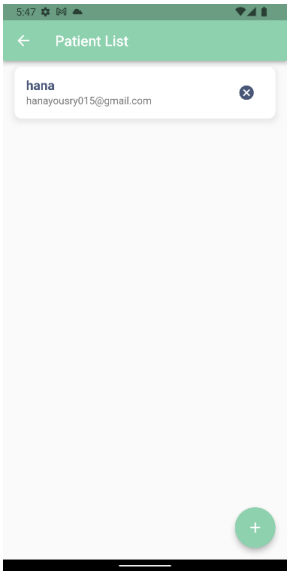
The same as patient screens.



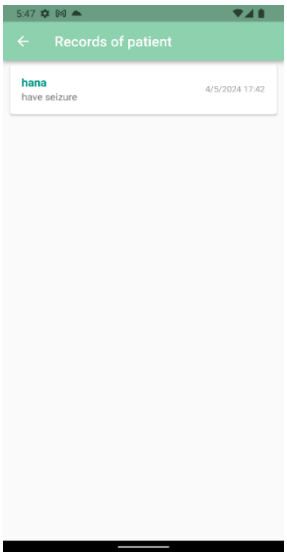
Search Patient.



Patient List.

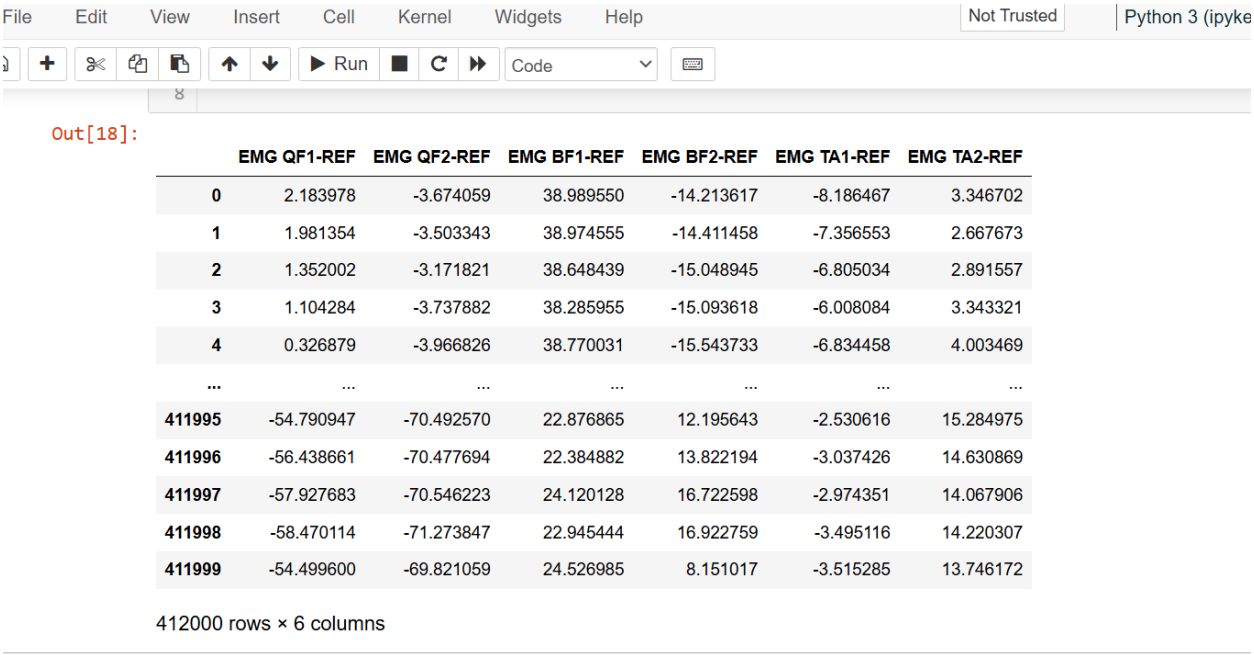


Record Patient.



5.3 Machine Learning

5.3.1 Dataset



Out[18]:

	EMG QF1-REF	EMG QF2-REF	EMG BF1-REF	EMG BF2-REF	EMG TA1-REF	EMG TA2-REF
0	2.183978	-3.674059	38.989550	-14.213617	-8.186467	3.346702
1	1.981354	-3.503343	38.974555	-14.411458	-7.356553	2.667673
2	1.352002	-3.171821	38.648439	-15.048945	-6.805034	2.891557
3	1.104284	-3.737882	38.285955	-15.093618	-6.008084	3.343321
4	0.326879	-3.966826	38.770031	-15.543733	-6.834458	4.003469
...	...	...	...	...	...	...
411995	-54.790947	-70.492570	22.876865	12.195643	-2.530616	15.284975
411996	-56.438661	-70.477694	22.384882	13.822194	-3.037426	14.630869
411997	-57.927683	-70.546223	24.120128	16.722598	-2.974351	14.067906
411998	-58.470114	-71.273847	22.945444	16.922759	-3.495116	14.220307
411999	-54.499600	-69.821059	24.526985	8.151017	-3.515285	13.746172

412000 rows × 6 columns

Dataset contains 6 columns (Channels of EMG raw signal) and 412000 rows for each file

We have 15 file. Each file contains EMG signals for specific patient.

5.3.2MODELS

```
import mne
import pandas as pd
from scipy.signal import butter, filtfilt
from tqdm import tqdm
import numpy as np

from keras.regularizers import l2
from keras.callbacks import EarlyStopping, ModelCheckpoint
from imblearn.over_sampling import RandomOverSampler
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import LSTM, Dropout, Dense, Bidirectional, Conv1D, MaxPooling1D, Flatten, BatchNormalization, Attent
from keras.optimizers import Adam
```

Python

Functions

```python
def load_emg_data(signals):
    data = mne.io.read_raw_edf(signals, preload=True, verbose=False, encoding='latin1')
    data.resample(1000)
    return data
```

They receive the path of the signal, load the signal file either with extension edf or csv then resample it to 1024 sample per second and return raw EMG signal. * For csv, the data is reshaped into 6 channels first.

```python
def emg_data_preprocessing(signals):
    # Convert to DataFrame and select relevant columns
    raw_data = signals.to_data_frame()
    raw_data = raw_data.loc[:, ~raw_data.columns.str.startswith('time')]
    raw_data = raw_data.iloc[:, -6:].to_numpy(dtype='float64')

    # Bandpass filter (10-450 Hz)
    low_band = 20 / 500
    high_band = 450 / 500
    a, b = butter(2, [low_band, high_band], btype='band')
    emg_filtered = filtfilt(a, b, raw_data, method='gust')

    # Rectify the signal
    emg_rectified = np.abs(emg_filtered)
    # Normalize the data
    emg_normalized = (emg_rectified - np.mean(emg_rectified, axis=0)) / np.std(emg_rectified, axis=0)

    return emg_normalized
```

It receives the raw EMG signal, removes channels which do not contain signals, for the classification of new signal, it takes the last 6 channels only as the model is trained with 6 channels, then apply band pass filter and band stop filter then rectify the signal.

```python
def get_datalabels(non_data, seizure_data, insert_col):
    seizure_labels = [1] * len(seizure_data)
    non_labels = [0] * len(non_data)

    all_data = np.concatenate((non_data, seizure_data), axis = 0)
    all_label = non_labels + seizure_labels

    data_label = np.insert(all_data, insert_col, all_label, axis=1)
    np.random.shuffle(data_label)

    data_label = pd.DataFrame(data_label)

    print(data_label)
    sig , label = data_label.iloc[:, :-1], data_label.iloc[:, -1]

    return sig , label
```

It takes the data, features filtered signal for EMG, extracted from the original signal. It creates labels for these data and concatenate data and labels to shuffle them, so the model won't learn from one patient and test on another. Then separate them again.

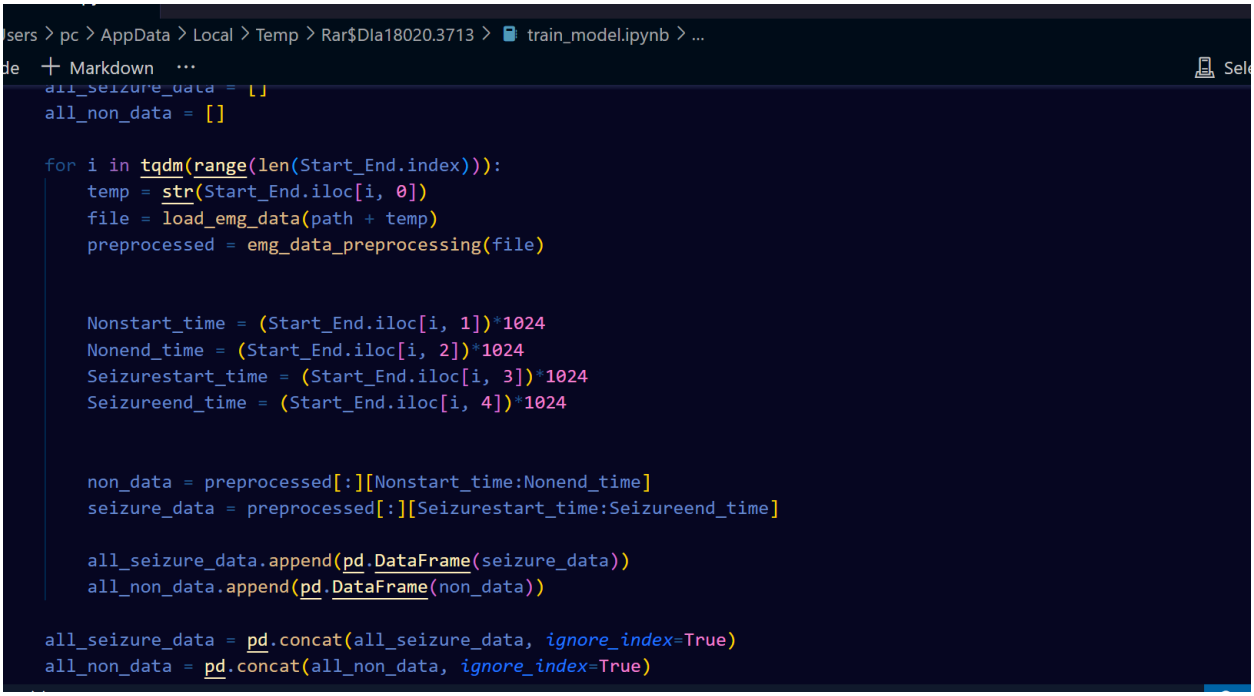
```python
def splitting_data(sig , label):
    oversampler = RandomOverSampler(
        sampling_strategy=1.0, random_state=42
    )
    data_input_balanced, data_output_balanced = oversampler.fit_resample(
        sig, label
    )


    '''
    Split the data into training, validation and testing with ratios 60:20:20 respectively
    '''
    X, X_test, y, y_test = train_test_split(
        data_input_balanced, data_output_balanced, test_size = 0.20, random_state = 42
    )

    X_train, X_val, y_train, y_val = train_test_split(
        X, y, test_size = 0.25, random_state = 0
    )

    return X_train, y_train, X_val, y_val, X_test, y_test
```

It takes the signals and its labels, apply random over sample on them to increase the number of samples of class 1 to become equal to number of samples of class 0, then splits it to train, validation and test with ratios 60:40:40.

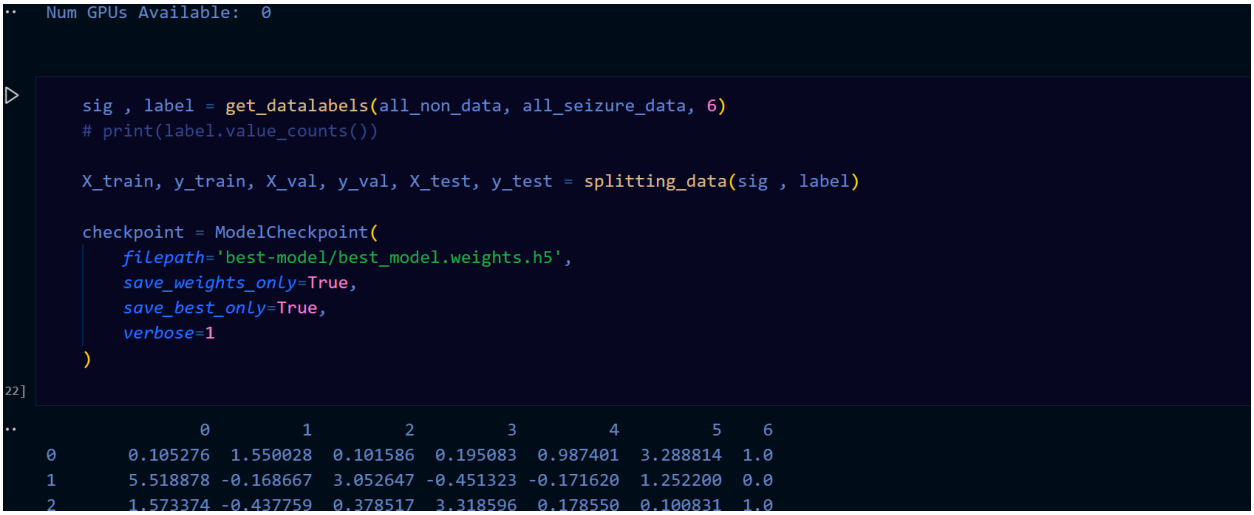
```python
Start_End=pd.read_csv('Start_End.csv')
Start_End=Start_End.loc[:,~Start_End.columns.str.startswith('Unnamed: 0')]
Start_End
```

	Signal	Non_start	Non_end	Seizure_start	Seizure_end
0	P1.1.edf	120	150	54	68
1	P1.2.edf	100	120	39	56
2	P1.3.edf	126	140	65	76
3	P1.4.edf	130	150	69	82
4	P10.1.edf	116	130	53	66
5	P10.2.edf	105	135	47	58
6	P11.1.edf	50	70	26	48
7	P11.2.edf	130	170	42	58

Start-End file which includes that name of the file with the start and end for the seizure and non-seizure parts to be used in the feature extraction.



It uses the information of Start-End file to first read the file and apply pre-processing techniques on it then extract the seizure and non-seizure part to be used in training.



We send the extracted features to get-datalabels along with the number of the last column to add the labels on it. Then we drop the first columns (0, 8, 16, 24, 32, 40, 48, 56) as they refer to the number of the items.

```
input_shape=(1, 6)

model = Sequential()
# Bidirectional LSTM layers
model.add(Bidirectional(LSTM(128, return_sequences=True)))
model.add(Dropout(0.3))
model.add(Bidirectional(LSTM(128)))
model.add(Dropout(0.3))

# Dense layer with L2 regularization
model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.01)))
model.add(Dropout(0.3))

model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.001), metrics=['accuracy', 'AUC'])


history = model.fit(X_train_reshaped, y_train_reshaped, epochs=10, batch_size=32, validation_data=(X_val_reshaped, y_val
```

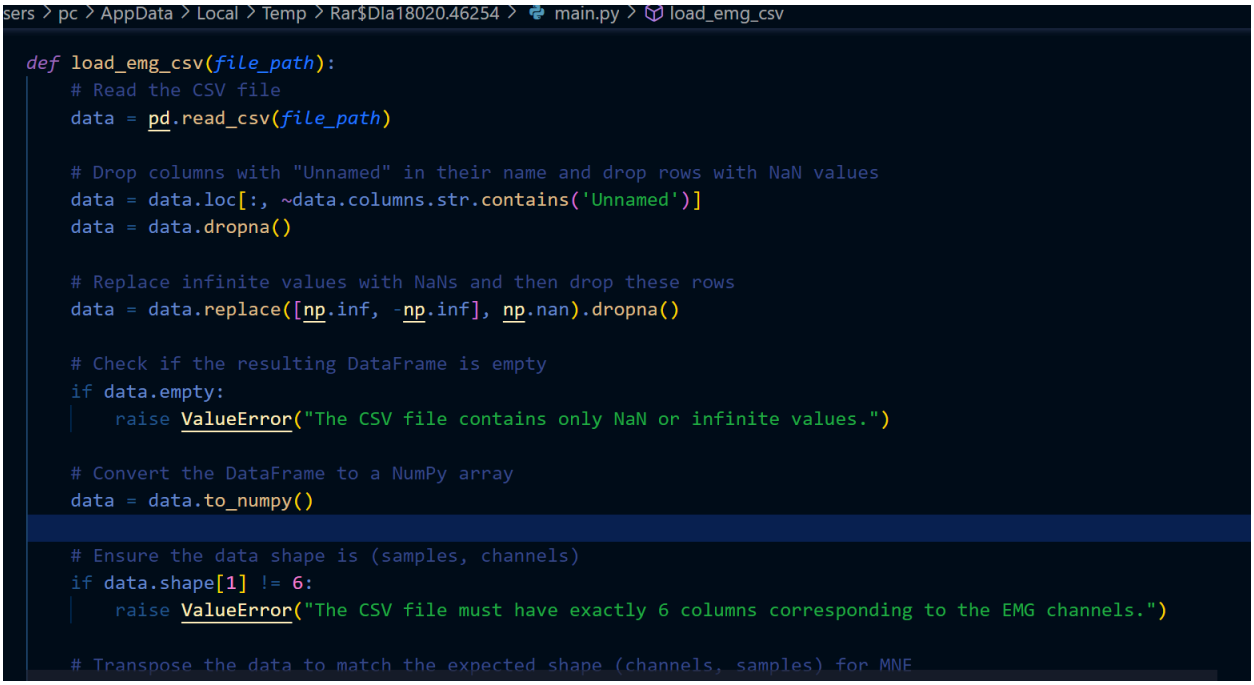
Creating and training a sequential model using Bidirectional LSTM layers in Keras.

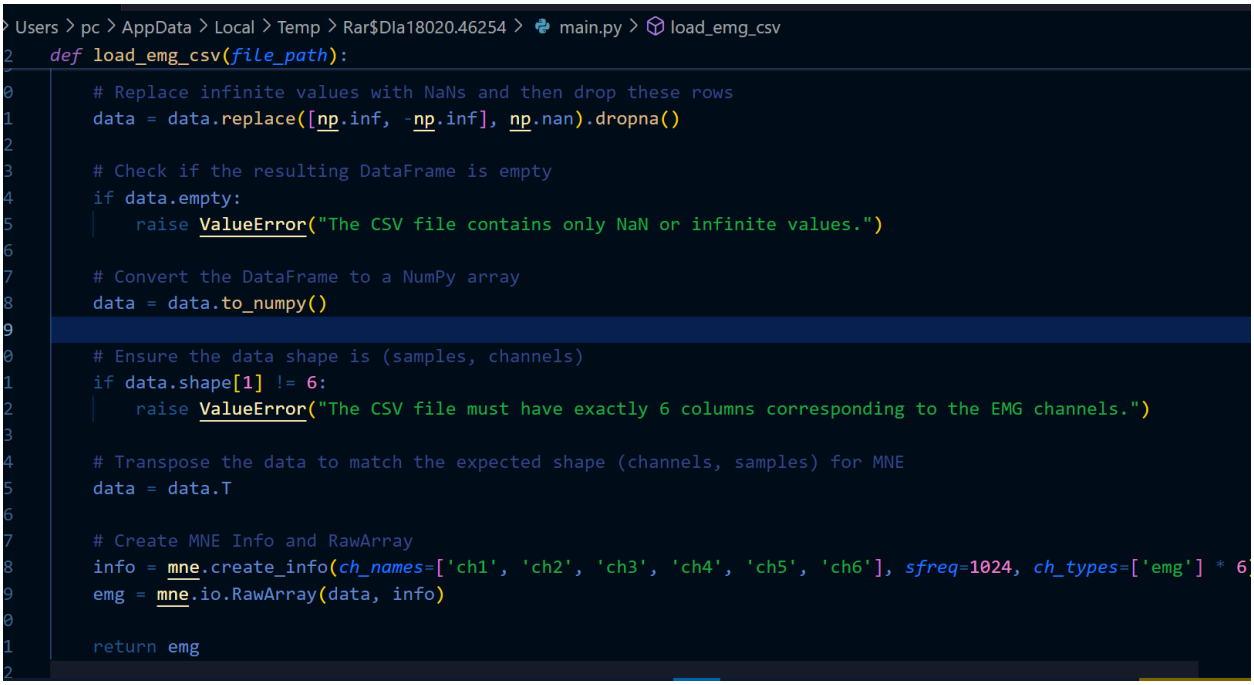
```
# Evaluate the model
loss, accuracy, auc = model.evaluate(X_test_reshaped, y_test_reshaped)
print(f'Test Loss: {loss}')
print(f'Test Accuracy: {accuracy}')
print(f'Test AUC: {auc}')
```

[25] Python

```
4775/4775 [==============================] - 9s 2ms/step - loss: 0.1711 - accuracy: 0.9481 - auc: 0.9814
Test Loss: 0.1711205542087555
Test Accuracy: 0.9480628967285156
Test AUC: 0.9814354181289673
```

Evaluating the model and printing the test loss, accuracy, and AUC.





function `load_emg_csv` is designed to load EMG data from a CSV file, preprocess it, and convert it into an MNE `RawArray`.

```python
class EMGClassifier:

    def classify_emg(self, signal):
        if isinstance(signal, str):
            # Handle file paths
            signal_name = signal
            if signal_name.endswith(".edf"):
                EMG_signal = load_emg_data(signal)
            elif signal_name.endswith(".csv"):
                EMG_signal = load_emg_csv(signal)
            else:
                raise ValueError("Unsupported file format. Please provide a .edf or .csv file.")
        else:
            # Handle file-like objects

            signal_name = werkzeug.utils.secure_filename(signal.filename)

            if signal_name.endswith(".edf"):
                with open("EMG_file.edf", "wb") as save_file:
                    signal.save(save_file)
                EMG_signal = load_emg_data("EMG_file.edf")

            elif signal_name.endswith(".csv"):
                with open("EMG_file.csv", "wb") as save_file:
```

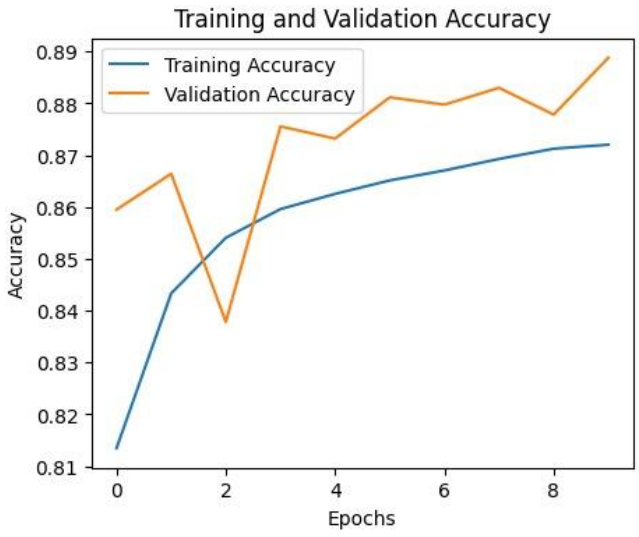
```python
class EMGClassifier:
    def classify_emg(self, signal):
                with open("EMG_file.edf", "wb") as save_file:
                    signal.save(save_file)
                EMG_signal = load_emg_data("EMG_file.edf")

            elif signal_name.endswith(".csv"):
                with open("EMG_file.csv", "wb") as save_file:
                    signal.save(save_file)
                EMG_signal = load_emg_csv("EMG_file.csv")
            else:
                raise ValueError("Unsupported file format. Please provide a .edf or .csv file.")

        preprocessed_emg = emg_data_preprocessing(EMG_signal)

        # Reshape data to fit the model input shape
        preprocessed_reshaped = preprocessed_emg.reshape((preprocessed_emg.shape[0], 1, preprocessed_emg.shape[1]

        # Model prediction on entire sequence or fixed-size windows
        prediction = self.model.predict(preprocessed_reshaped)
        mean_prediction = np.mean(prediction)

        return mean_prediction
```

```python
class EMGClassifier:

    @staticmethod
    def classification(y_pred):
        class_name = 'Seizure' if y_pred >= 0.70 else 'No Seizure'
        prob_Seizure = y_pred
        prob_no_Seizure = 1 - y_pred
        result = {
            'Probability of Seizure': (prob_Seizure * 100),
            'Probability of No Seizure': (prob_no_Seizure * 100),
            'Class Name': class_name
        }

        return result
```

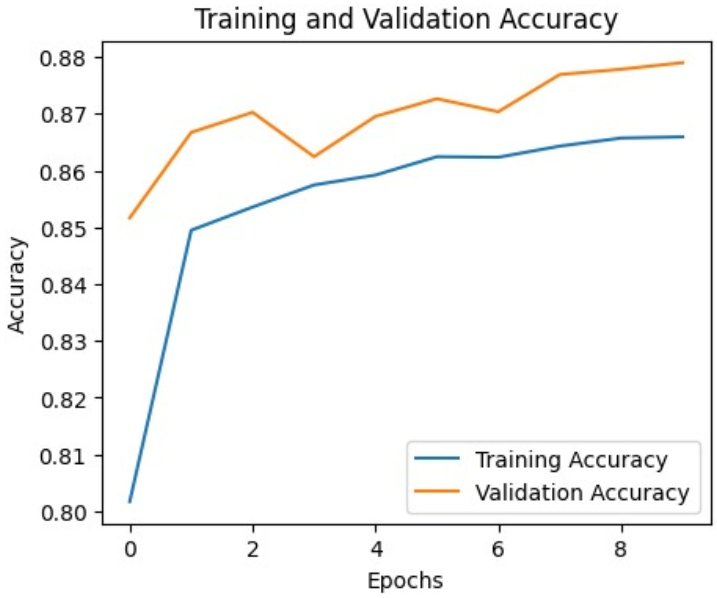
function `classify_emg` handles the classification of EMG signals from file paths.

Models Accuracy:

1- Bidirectional LSTM 64 layer

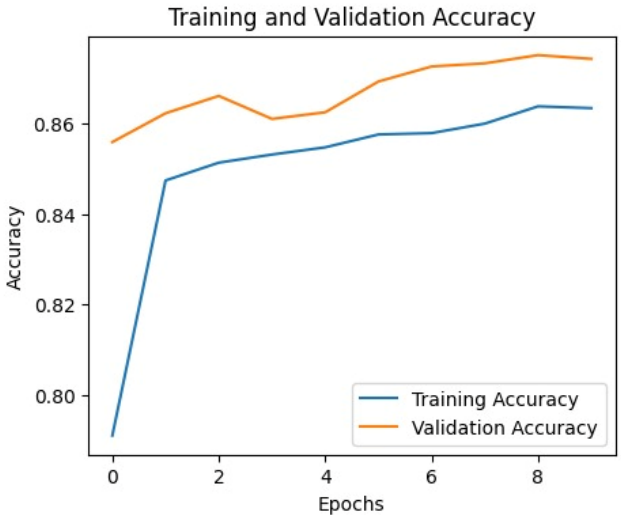


2- GRU 64 layer

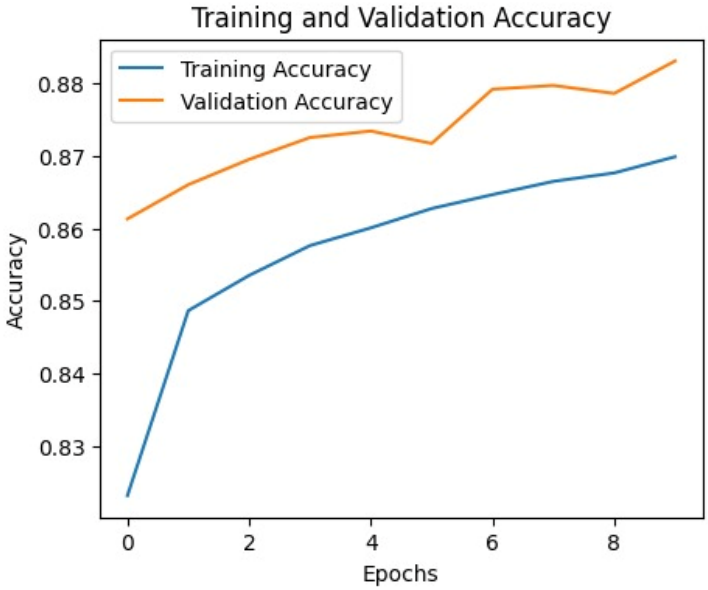


3- GRU 128 layer

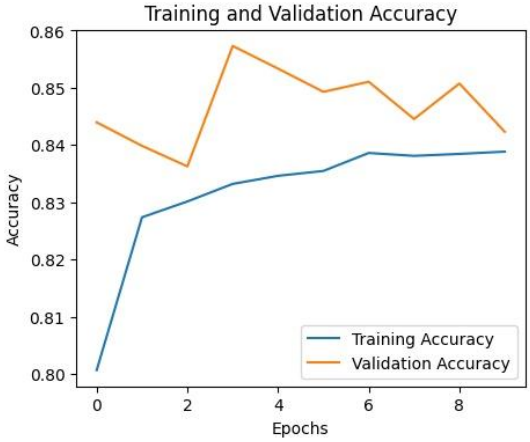
4- LSTM 64 layer



5- LSTM 128 layer



6- LSTM (recurrent regulariser)



5.3.3 Flask API

Flask App Initialization

The Flask application is initialized and configured to handle file uploads and integrate with Firebase for storing detection results.

Key Components:

- **Flask App:** The main web application framework.
- **Firebase Admin SDK:** Used to interact with Firebase Realtime Database.
- **EMG Classifier:** A class from the main module responsible for classifying EMG signals.

Initialization Code:

- Initialize Firebase with service account credentials.
- Set up the file upload directory.

Routes and Functionality

The application provides a web interface to upload EMG signal files and get classification results.

index

- **Route:** /
- **Method:** GET
- **Description:** Renders the homepage of the web application.
- **Returns:** An HTML page (index.html).

Classify EMG

- **Route:** /classify EMG
- **Method:** POST
- **Description:** Handles the upload of EMG signal files, processes the files using the LSTM model, and stores the results in Firebase.
- **Parameters:**
    - File (file): The uploaded EMG signal file.

- **Returns:**
    - JSON: A JSON response containing the classification results:
        - Class Name: Either 'Seizure' or 'No Seizure'.
        - Probability of Seizure: The probability that the signal is a seizure.
        - Probability of No Seizure: The probability that the signal is not a seizure.
        - Timestamp: The timestamp when the classification was performed.
- **Error Handling:**
    - Returns an error message if no file is provided or if the file is not selected.

Firebase Integration

The application uses the Firebase Admin SDK to store the classification results in Firebase Real-time Database. This allows the Flutter app to access the latest detection results in real-time.

Firebase Setup

- **Credentials:** Service account key (key.json) is used to authenticate and initialize Firebase.
- **Database Reference:** A reference to the /seizures path in the Firebase Real-time Database is created to store the classification results.

Data Storage

- **Structure:** The classification results, including the class name, probabilities, and timestamp, are stored as JSON objects in Firebase.

Conclusion

This Flask application facilitates real-time epilepsy seizure detection using EMG signals. By integrating with Firebase, the application ensures that the results are easily accessible to other platforms, such as a Flutter app, enabling effective monitoring and timely intervention

.

Chapter Seven

“Testing”

7.1.1 Software Components:

- **MOSQUITTO Broker**: This is a message broker that will facilitate communication between the ESP32 and other devices using the MQTT protocol (lightweight messaging protocol). we installed MOSQUITTO on a computer as a local server.

- **Wi-Fi:** This will enable the ESP32 to connect to the internet and communicate with the MOSQUITTO broker and other services to provide IOT system network interaction.

- **Flask API:** This Python web framework will create a RESTFUL API that the ESP32 and Flutter application can interact with. The API will likely handle tasks like receiving data from the ESP32, processing it through the deep learning model, and storing results in the Firebase database.

- **Deep Learning Model:** This is a machine learning model trained to identify seizure patterns from EMG data. we explored a pre-trained models or train your own model using EMG datasets. TensorFlow is popular deep learning frameworks.

- **Firebase Realtime Database:** This cloud-based NoSQL database from Google will store the processed EMG data and seizure alerts in real-time.

- **Flutter Application (Epilert):** This mobile application developed with the Flutter framework will display seizure alerts received from the Firebase database and potentially provide additional functionalities like notification or emergency contact options.

Additional Considerations:

- **Data Pre-processing**: We need to pre-process the EMG data collected by the sensor before feeding it into the deep learning model. This might involve filtering, normalization, or segmentation.

- **Model Training:** We trained **LSTM** deep learning model and needed a suitable EMG dataset and computational resources to train this model effectively.

- **Security**: We Considered implementing security measures to protect sensitive health data collected by the EMG sensor.

Putting it all Together:

1. The EMG sensor captures muscle activity signals.
2. The ESP32 processes the signals and transmits them to the Mosquitto broker using Wi-Fi and MQTT.
3. The Flask API receives the data from the broker.
4. The API preprocesses the data and feeds it into the deep learning model.
5. The model analyses the data and generates a seizure prediction.
6. The API stores the prediction and any relevant data (timestamps, etc.) in the Firebase database.
7. The Flutter application retrieves data updates from the Firebase database and displays seizure alerts to the user.

7.2.1 Testing Levels Testing levels

refer to the different stages or phases of software testing that are conducted to ensure the quality and reliability of a software product. There are various testing levels based on the specificity of the test.

1. **Unit Testing:** This is the first level of testing where individual units or components of the software are tested in isolation to ensure that they function as intended.

2. **Integration Testing:** In this level, multiple units or components are combined and tested together to ensure that they work seamlessly with each other.

3. **System Testing:** This level involves testing the entire system, including all its components and subsystems, to ensure that it meets all functional and non-functional requirements.

4. **Performance testing:** Performance testing is the process of determining the speed or effectiveness of a computer, network, software program or device such as response time or millions of instructions per second etc.

5. **Acceptance Testing:** This is the final level of testing where the software is tested by end-users or stakeholders to ensure that it meets their expectations and requirements

7.2.1 Test Cases

A test case is a set of test data, preconditions, expected results and post conditions, developed for a test scenario to verify compliance against a specific requirement. We have designed and executed a few test cases to check if the project meets the functional requirements.

7.2.2 Hardware Test Cases

7.2.2.1 Introduction

This chapter provides a comprehensive overview of the hardware testing process for the EMG sensor used in our seizure detection system. The aim is to ensure the sensor's accuracy, reliability, and performance under various conditions, validating its suitability for real-time seizure monitoring.

7.2.2.2 Objectives

The objectives of the hardware test are:

- To verify the functional performance of the EMG sensor.
- To assess the sensor's sensitivity and specificity in detecting muscle activity associated with seizures.
- To evaluate the sensor's reliability and stability over prolonged usage.
- To identify and mitigate any potential sources of error or interference.

7.2.2.3 Test Setup

1- Hardware Components

- EMG Sensor: Muscle Sensor v3
- Microcontroller: ESP32 WROOM
- Power Supply: 5V DC power adapter
- Signal Processing Unit: Integrated within the microcontroller

- Data Acquisition System: Computer with USB interface
- Test Subjects: Simulated muscle contractions and live human trials

2- Software Tools

- Arduino IDE: For programming the microcontroller

- Anaconda Platform: For data analysis and visualization

- Mobile APP: Developed for real-time monitoring and logging of EMG data

1- Test Environment

- Controlled laboratory setting with minimal electrical noise
- Temperature: 22-25°C
- Humidity: 40-60%

2- Test Procedure

1-Calibration

- **Initial Setup:** Connect the EMG sensor to the ESP32 and power supply.
- **Baseline Measurement:** Record baseline EMG readings with no muscle activity to establish a reference.
- **Calibration:** Adjust sensor sensitivity and amplification settings to optimize signal detection.

2- Functional Testing

1. **Simulated Muscle Contractions:** Use a signal generator to simulate muscle contractions at different frequencies and amplitudes.

- Test low, medium, and high frequency contractions.
- Vary the amplitude to simulate different muscle strengths

2. Live Testing: Attach the sensor to a human subject's muscle (for arm) and record EMG signals during voluntary muscle contractions.

- Ensure ethical guidelines are followed and obtain informed consent from subjects.
- Monitor and record EMG signals during rest, mild contractions, and intense contractions.

3- Reliability and Stability Testing

1. **Continuous Monitoring:** Operate the sensor continuously for 24 hours to assess stability.
   Record any drift or noise in the baseline readings.

2. **Temperature and Humidity Variations:** Test the sensor's performance under different environmental conditions.
   - Increase/decrease temperature by 5°C.
   - Increase/decrease humidity by 20%.

4- Interference Testing

1. Electrical Interference: Introduce common sources of electrical noise (ex: nearby electronic devices) and observe the sensor's performance.

2. Motion Artifacts: Simulate movements and vibrations to test the sensor's resistance to motion artifacts.

5- Data Analysis

- **Signal Quality Assessment:** Analyze the signal-to-noise ratio (SNR) of the recorded EMG data.
- **Accuracy and Precision:** Calculate the accuracy and precision of the sensor in detecting known simulated and live contractions.
- **Error Analysis:** Identify and quantify any errors or anomalies in the data.

6- Results

- **Calibration Results**
  - Baseline EMG readings.
  - Optimal sensitivity and amplification settings.

- **Functional Test Results**
  - EMG signal characteristics for different frequencies and amplitudes.
  - Comparison of simulated vs. live muscle contraction data.

- **Reliability and Stability Results**
  - Stability of baseline readings over 24 hours.
  - Performance variations under different environmental conditions.

- **Interference Test Results**
  - Impact of electrical noise on EMG signal quality.
  - Effectiveness of the sensor in rejecting motion artifacts.

Test title	Test conditions	Preconditions	Test steps	Expected output	Actual output	Status
Powering EMG	The EMG is powering.	The battery is charged (2 batteries 9v).	Wearing electrodes Of EMG And ESP32 is powered on.	Serial receive signals range from 300 to 5000.	Serial received signals in written range.	PASS
	Delay while powering EMG.				Signal imbalanced.	PASS
	The EMG doesn't work.				No signal received.	FAIL
Check ESP32	ESP32 is working.	The battery is charged (3.7v).	Connect Lithium battery.	ESP32 led on.	ESP32 led on.	PASS
	ESP32 not working.				ESP32 led off.	FAIL
Check Lithium battery and charge module	Lithium battery is charging	Connection established and charger with 5v and 1A.	Connect lithium battery to charger.	Charger module blue led is on.	Blue led on.	PASS
	Lithium battery is not charging.				Blue led off	FAIL

7.2.3 Connection Test Cases

Test title	Test conditions	Preconditions	Test steps	Expected output	Actual output	Status
Check serial connection	If Baud rate is 9600.	UART is enabled.	Code executed on ESP32.	Data received on the server and can be visualized in serial monitor.	Data received.	PASS
	If Baud rate is not 9600.				Data not received.	FAIL
Check Wi-Fi connection	If Wi-Fi enabled in pc.	Knowing Ip configuration of available network.	Enable hotspot and ESP32 is powered.	Print in serial WIFI connected and IP of network.	Print in serial WIFI connected and IP of network.	PASS
	If Wi-Fi not enabled in pc.				Print “.”until connection retrieved.	FAIL
Check MQTT connection	If MOSQUITTO broker is enabled.	Establishing MOSQUITTO libraries in pc.	Prepare MOSQUITTO files with TEST.CONF file that contains type of connection not secure, port 1883).	Data received in MQTTCLI with topic.	Data received in MQTT CLI with topic.	PASS
	If MOSQUITTO broker isn’t enabled				No data received in MQTT CLI	FAIL
Check PYTHON_MQTT subscriber	if topic is found.	Topic is founded and python. PAHO library is installed.	Identify same topic in MQTT then port of MQTT (1883) with	Data received in python and created EMG	Data received in python and created EMG	PASS

			IP of network to receive data.	RECORDS.	RECORDS.	
	If topic not found.				Data not received in python.	FAIL
Check app.py file (model connection)	If new EMG RECORD found.	Flask API connected to PYTHON_MQTT subscriber.	Installed required libraries for Flask API.	Create new record if data were sent to model and send results of model to firebase.	File (processed in model) created then sent to firebase.	PASS
	If new EMG RECORD not found.				No file (processed in model) created or sent to firebase.	FAIL

7.2.4 Application Test Cases

7.2.4.1 Login

Test Case ID	Description	Steps	Expected Result
1.1	Valid Login	1. Navigate to the login page . 2. Enter a valid username and password . 3. Click on the login button.	User is successfully logged in and redirected to dashboard.
1.2	Invalid Login (Incorrect Password)	1. Navigate to the login page . 2. Enter a valid username and an incorrect password. 3. Click on the login button.	User receives an error message indicating incorrect password.

Test Case ID	Description	Steps	Expected Result
1.3	Invalid Login (Non-existent Username)	1. Navigate to the login page. 2. Enter a non-existent username and any password. 3. Click on the login button.	User receives an error message indicating username doesn't exist.
1.4	Empty Credentials	1. Navigate to the login page. 2. Leave the username and password fields empty. 3. Click on the login button.	User receives an error message indicating both fields are required.

7.2.4.2 Sign Up

Test Case ID	Description	Steps	Expected Result
2.1	Valid Sign Up	1. Navigate to the sign-up page. 2. Enter valid information in all required fields. 3. Click on the sign-up button.	User is successfully registered and redirected to login page.
2.2	Invalid Sign Up (Existing Username)	1. Navigate to the sign-up page. 2. Enter an already existing username. 3. Fill in other required fields and click sign-up.	User receives an error message indicating username is taken.
2.3	Password Mismatch	1. Navigate to the sign-up page. 2. Enter different passwords in the password and confirm password fields. 3. Click on the sign-up button.	User receives an error message indicating passwords do not match.

7.2.4.2 Profile

Test Case ID	Description	Steps	Expected Result
3.1	View Profile	1. Log in to the application. 2. Navigate to the profile page.	User can view their profile information.
3.2	Edit Profile	1. Log in to the application. 2. Navigate to the profile page. 3 Click on the edit button. 4. Update profile information. 5 Click save.	Profile information is updated successfully.

7.2.4.3 Connect Device

Test Case ID	Description	Steps	Expected Result
4.1	Connect New Device	1. Log in to the application. 2. Navigate to the devices page. 3. Click on "Add Device" button. 4. Enter device details and click connect.	Device is successfully connected and listed in user's devices.
4.2	Disconnect Device	1. Log in to the application. 2. Navigate to the devices page. 3. Select a connected device and click "Disconnect" button.	Device is successfully disconnected and removed from user's devices.

7.2.4.4 Search Patient

Test Case ID	Description	Steps	Expected Result
5.1	Search Patient by Name	1. Log in to the application. 2. Navigate to the patient search page. 3. Enter a patient's name in the search field and click search.	Search results display patients matching the entered name.
5.2	Search Patient by ID	1. Log in to the application. 2. Navigate to the patient search page. 3. Enter a patient's ID in the search field and click search.	Search results display the patient with the matching ID.
5.3	Search Patient with No Results	1. Log in to the application. 2. Navigate to the patient search page. 3. Enter a random name or ID that does not exist and click search.	No results found and an appropriate message is displayed.

Heuristic Evaluation:

The Eight Golden Rules of Interface Design

Rule	Question considers	Mark
Strive for consistency	Icons consistency in home page and other pages and same design	☑
Seek universal usability	Use universal language (English)	☑
Offer informative feedback.	If change password it sends messages that new link to update password is sent and check it	☑
design dialogs to yield closure	When user want to log in app make sure he wants it by sending verification message in his email .	☑
Prevent errors	If user entered email without @. give message to help him that invalid email	☑
Permit easy reversal of actions.	If user want to go back from any action in system	☑
Keep users in control	No in system.	☒
Reduce short-term memory load	No	☒

Nielsen's 10 Usability Heuristics

Heuristic	Question considers	Mark
Visibility of system status	No	✗
Match between system and the real world	Icons of pages like real world	☑
User control and freedom	Navigate easily between pages with back arrows.	☑
Consistency and standards	Icons consistency in home page and other pages and same design	☑
Prevent errors	If user entered email without @. give message to help him that invalid email	☑
Recognition rather than recall	No	✗
Flexibility and efficiency of use	No	✗
Aesthetic and minimalist design	Design is simple and recognized easily and clear.	☑
Help users recognize, diagnose, and recover from errors	Error messages to help user to add. Password instructions and Email	☑
Help and documentation	No	✗

4.Heuristic Evaluation Report

Heuristic and Rules does not apply in application.

1- Visibility of system status: The application doesn't have many statuses, but it gives feedback "alarm or message" that confirm update will processing. Like verification to reset password.

2- Recognition rather than recall: we didn't have many tabs to navigate so we didn't add bar with pass. It is provided on some screens, and some do not have.

3- Flexibility and efficiency of use: the app is only for normal people, and we don't have many modes.

4- Help and documentation: the application functionality is easy and simple, so user doesn't have to read documentation to help him navigate inside the app.

7.2.5 Functional Test

Test case	Procedure	Expected output	Actual Output	Status
Power and Connectivity	Connect batteries to the EMG sensor and the ESP32	ESP32 powers up, connects to Wi-Fi and The EMG read signals.	ESP32 powers up, connects to Wi-Fi and The EMG read signals.	PASS
EMG Signal Acquisition	Attach EMG electrodes and perform muscle contractions.	Accurate EMG signals displayed on serial monitor.	Accurate EMG signals displayed on serial monitor.	PASS
Data Transmission via MQTT	Transmit EMG data to Flask application.	Data packets received without delays or losses.	Data packets received without delays or losses.	PASS
Data Processing and Storage	Process data in Flask application and store results in Firebase.	Data processed correctly and securely stored.	Data processed correctly and securely stored.	PASS
User Interface Verification	Access app and check real-time and historical data display.	Accurate display of real-time and historical data with location	Accurate display of real-time and historical data without location	FAIL
Battery and Charging Test	Run system on battery, then connect charging module.	System operates correctly on battery; smooth charging process.	System operates correctly on battery; smooth	PASS
Overall System Functionality	Simulate a seizure event.	System detects event, processes data, and generates alerts.	System detects event, processes data, and generates alerts.	PASS

Chapter eight

“Discussion and conclusion”

8.1 Limitations

We Develop Seizure Detection Application Using EMG Sensor, ESP32 Microcontroller and Communication via MQTT and WIFI to Anaconda Platform using flask API Subsequence Between model and firebase. The System has some Limitations that impacts reliability , usability and accuracy of system.

1. User should connect to Wi-Fi so Application can send notifications, the system is IOT oriented so any failure in network causes data loss.

2. Both EMG sensor and ESP32 are battery dependent, need frequent charging that mean long term monitoring.

3. Users follow guidelines to wear the device correctly. Right emplacement of the electrodes on the muscle.

4. Processing EMG data in real time for several users may lead to high data load on the local server.

5. Deep learning algorithm needs continuous improvements.

8.2 Recommendation and Future Work

Looking forward, there are several promising directions for advancing the seizure detection system, each of which can contribute to improving accuracy, reliability, user experience, and scalability.

1. Exchange the local server with cloud-based server like AWS IOT core (Provide paid services)
    - Device shadow prevents data loss.
    - AWS S3 provides storage services.
    - Amazon dynamo DB.
    - Automatic registration for device provisioning.
    - Amazon simple notification (Amazon SNS).
    - AWS lambda function.

2. Use alternative power solutions like: - rechargeable batteries.
3. Use additional sensors such as ECG, EEG to create multi model detection system.

By pursuing these avenues for future work, the seizure detection system can evolve into a more robust, accurate, and user-friendly solution, offering significant benefits to individuals with epilepsy and their caregivers. Through proposed solutions the system can contribute to improved seizure management and quality of life for affected individuals.

8.3 Conclusion

The design and implementation of the seizure detection system using EMG sensors, the ESP32 microcontroller, and an IOT-based architecture represents a significant step forward in health monitoring technology. This system allows users to monitor their health parameters, specifically focusing on seizure detection, which can greatly aid in managing epilepsy and improving patient safety. By providing real-time alerts and the ability to share health data with medical professionals, this system empowers patients to seek timely medical assistance when necessary.

Our project demonstrates the practical application of IOT in health monitoring. The seizure detection system measures muscle activity using EMG sensors, processes the data using the ESP32 microcontroller, and transmits the information via MQTT over Wi-Fi to a Python-based Flask application. The processed data is then stored securely in Firebase, from where it can be accessed through a user-friendly mobile. This setup ensures that health data is securely stored and accessible to both patients and their relatives, enabling remote health monitoring and timely interventions.

The system is particularly beneficial for individuals with epilepsy, allowing them to monitor their condition continuously. This continuous monitoring is crucial for managing chronic conditions and can help in reducing the risk of severe seizure episodes by providing early warnings and facilitating prompt medical response. The system can also be adapted for other health monitoring applications, making it a versatile tool for a wide range of patients, including those with chronic illnesses, elderly patients, and individuals with other health conditions that require regular monitoring.

From an engineering perspective, this project has effectively applied concepts from computer science and embedded systems, demonstrating the integration of hardware and software to create a functional and reliable health monitoring solution. Knowledge of electric circuit analysis was utilized in the design and fabrication of the sensor modules, while expertise in electromagnetic fields and wireless communication was essential for the data transmission between the microcontroller and the server. The software programming aspect was crucial for developing the microcontroller firmware and the server-side application, ensuring seamless operation of the entire system.

In conclusion, this IOT-based seizure detection system showcases the potential of integrating advanced technologies to improve health monitoring and patient care. By addressing current limitations and pursuing future enhancements, this system can become a vital tool in managing epilepsy and other health conditions, ultimately contributing to better health outcomes and quality of life for patients.

9.References:

- Mosquitto broker: https://mosquitto.org/.
- ESP32: https://www.espressif.com/en/products/devkits.
- Flask: https://flask.palletsprojects.com/en/2.3.x/.
- TensorFlow: https://www.tensorflow.org/.
- Firebase Realtime Database: https://firebase.google.com/docs/database.
- Flutter: https://flutter.dev/.
- ESP32 Board : https://microohm-eg.com/product/esp-wroom-32/.
- EMG Module – Muscle Signal Sensor Kit: https://microohm-eg.com/product/emg-muscle-sensor-v3-kit/.
- Rechargeable lithium polymer battery: https://microohm-eg.com/product/polymer-li-ion-single-cell-battery-3-7v-1200mah-5030mm/.
- Micro USB Lithium Battery Charging Module TP4056 with Battery Protection: https://microohm-eg.com/product/tp4056-1a-li-ion-battery-charging-board-micro-usb-with-current-protection/.
- Breadboard: https://microohm-eg.com/product/400-tie-points-contacts-mini-circuit-experiment-solderless-breadboard/.