# 언어치료환자를 위한 구강운동촉진기기 제작 매뉴얼

## 1.하드웨어 설계

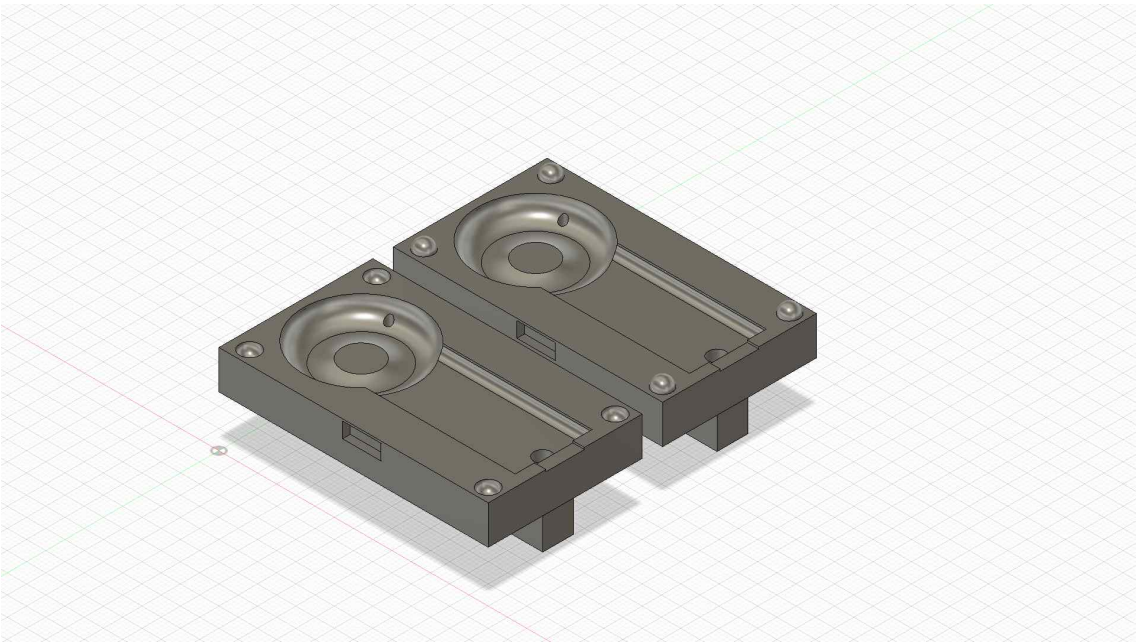
### 1-1 실리콘 금형 틀 제작

구강내 삽입할 수 있는 센서의 제작을 위해 실리콘 금형 틀이 필요합니다.

금형틀 디자인은 Abilex사의 Oral Motor Exerciser를 참고하여 제작하였습니다.

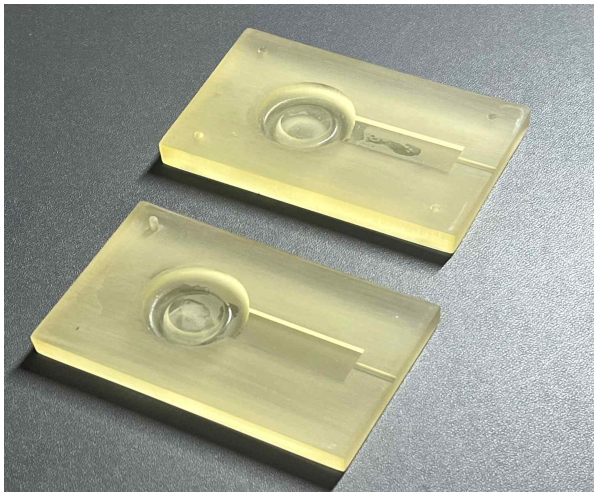
금형틀 제작 프로그램은 Autodesk 회사의 fusion360이라는 프로그램을 사용했습니다.

프로그램을 이용하여 Abilex사의 제품을 참고하여 금형틀을 제작합니다.



금형틀의 재료는 높은 열 높은 내구성을 가지는 고온에 강한 하이 템프 레진을 사용하여 레진 금형을 제작합니다.

완성된 금형틀의 모습입니다.

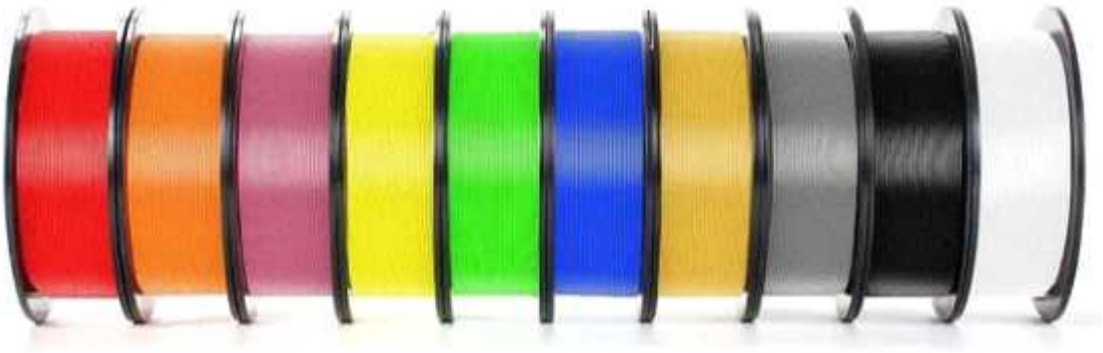


## 1-2 조음운동촉진기기 핸들 손잡이 제작

센서모듈의 손잡이 부분을 제작하기 위해 프로그램은 Autodesk 회사의 fusion360 프로그램을 사용했습니다.

손잡이 부분은 따로 모형 기준은 따로 없으며 손에 잡기 쉬운 그립을 모형으로 제작합니다.

3D 프린터를 이용하여 제작하며
재료는 3D 프린터를 이용할 때 사용하는 PLA를 사용합니다.



제작이 완료된 조음운동촉진기기의 손잡이입니다.

## 1-3 조음운동촉진기기 실리콘 센서부 제작

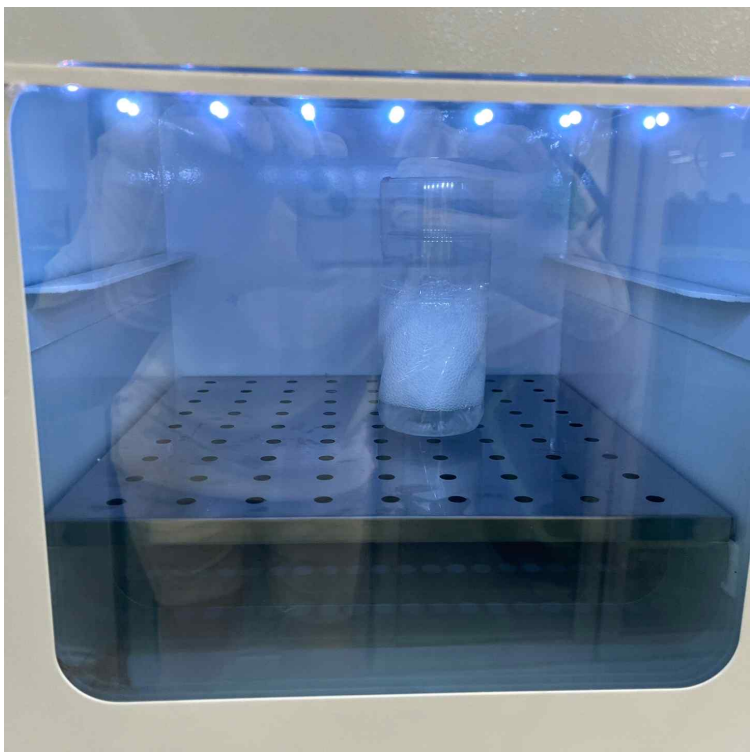
실리콘 센서부을 제작하기 위한 재료는 Sylgard-184입니다.


**Biocompatible Silicone Sylgard-184**
**(Dow Corning, USA)**

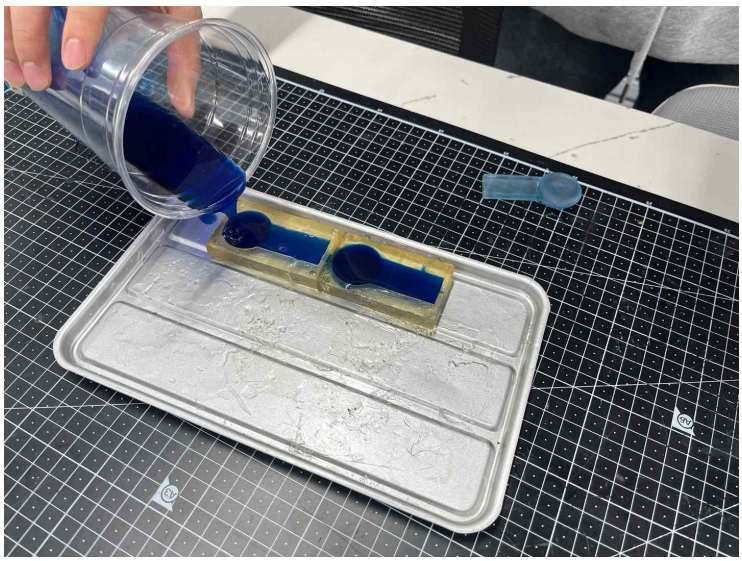
실리콘과 경화제의 비율은 실리콘 10, 경화제 1이며 보통 실리콘 40g에 경화제 4g으로 맞추면 약 1개의 실리콘센서를 제작할 수 있습니다.

만들어진 실리콘안에는 아직 기포가 많아 이 기포를 빼줘야하며, 챔버를 이용하면 기포를 쉽게 제거 할 수 있습니다.

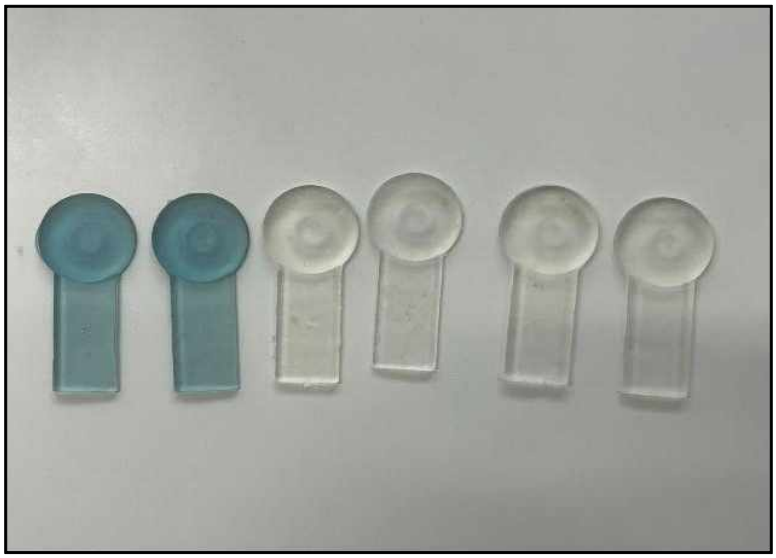
챔버를 정상적으로 잘 동작시키면 사진과 같이 기포가 빠지는 것을 확인 할 수 있습니다.



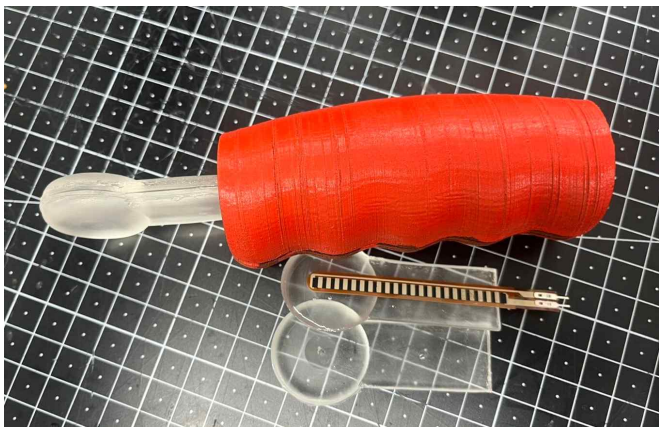
챔버를 이용하여 기포를 빼는 시간은 3시간 + α 정도며 길게 기포를 뺄수록 실리콘에 기포자국이 더욱 안보입니다.
기포를 뺀 실리콘을 목업 금형틀위에 부어줍니다.



금형틀 제작 후에는 즉시 경화 작업을 진행해야 하며, 이를 위한 두 가지 방법 중 하나는 자연경화입니다. 이 방법은 금형틀에 실리콘을 부었을 때 생기는 작은 기포들이 자연스럽게 빠져나갈 시간을 제공하여 더 높은 품질의 센서를 제작할 수 있게 합니다. 단점은 시간이 오래 걸리며 보통 경화시간은 24시간 + α 정도입니다. 완전하게 굳지 않은 상태에선 끈적거리는 특징을 보입니다.
두 번째 방법으론 고온으로 빠른 경화를 유도하는 방법입니다. 챔버에는 탈포기능만 있는 것이 아니라 오븐기능도 있는데 150도로 설정 후  경화를 기다립니다. 4시간 정도의 시간이 소요되며, 빠른 경화를 하는 대신 실리콘 센서에 기포자국이 생길 가능성이 높습니다.
완성된 실리콘입니다.



완성된 조음운동촉진기기의 실리콘 센서부와 핸들 손잡이 부분



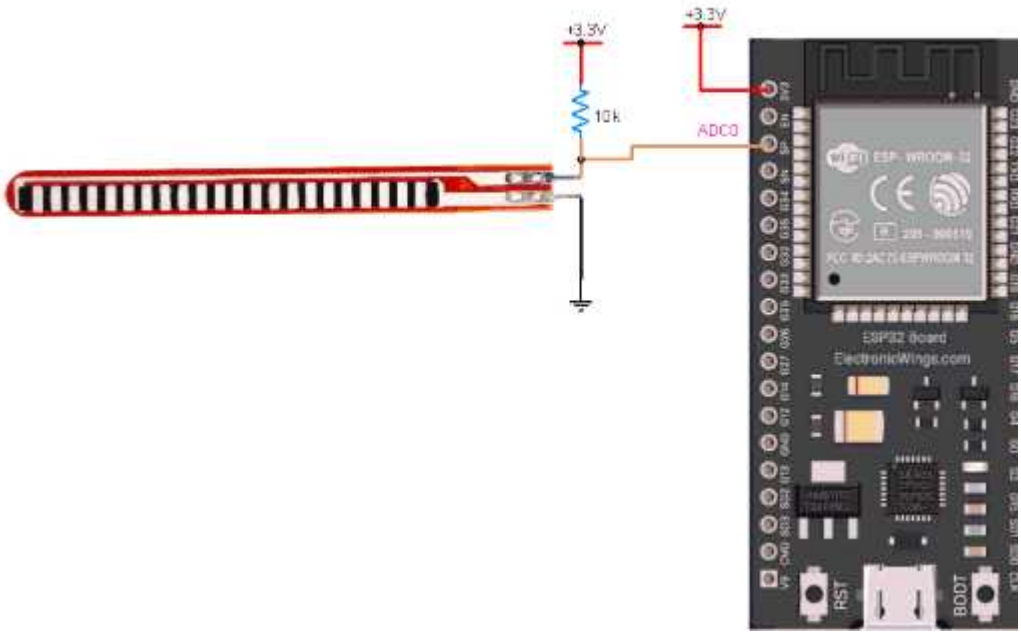
## 1-4 조음운동촉진기기 하드웨어 연결

FLEX SENSOR는 휘어짐에 강도에 따라 전압으로 나타내는 센서입니다.

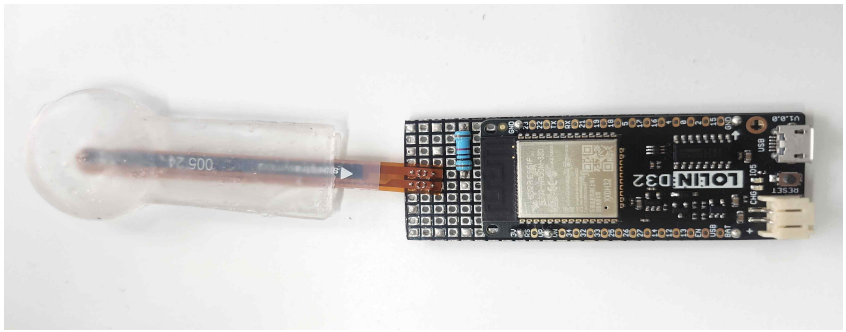
신호의 처리를 위해서는 EPS32 Lolin 보드가 필요합니다. ESP32는 BLE 통신과 ADC 변환을 위해 사용합니다.

또한, 조음운동촉진기기는 무선통신을 위해 3.7V 리튬폴리머배터리를 이용합니다.

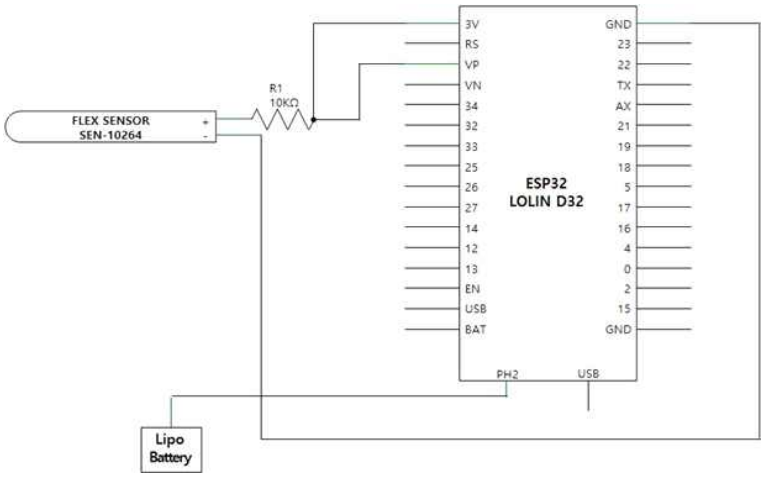

플렉스 센서와 신호처리 보드연결도는 아래와 같습니다.



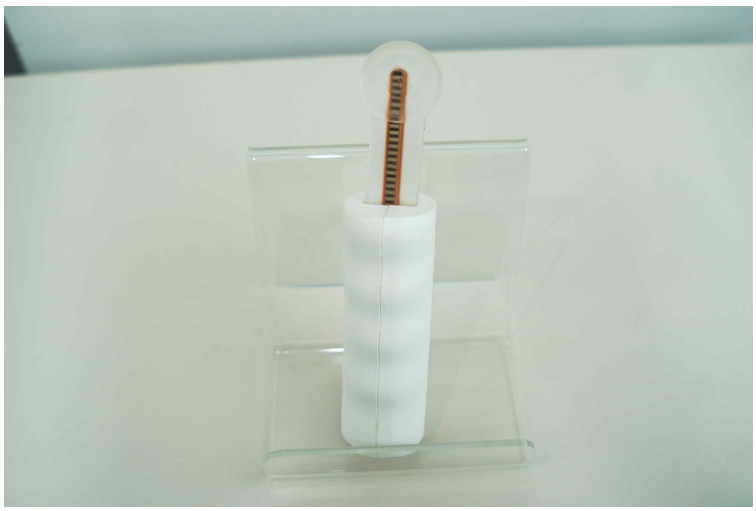

만들어진 실리콘에 플렉스 센서를 고정후 ESP32와 연결한 모습입니다.



연결된 TTB와 배터리를 아래 도면도를 참고하여 연결합니다.




최종 패키징된 조음운동촉진기기의 모습은 아래의 그림과 같습니다.

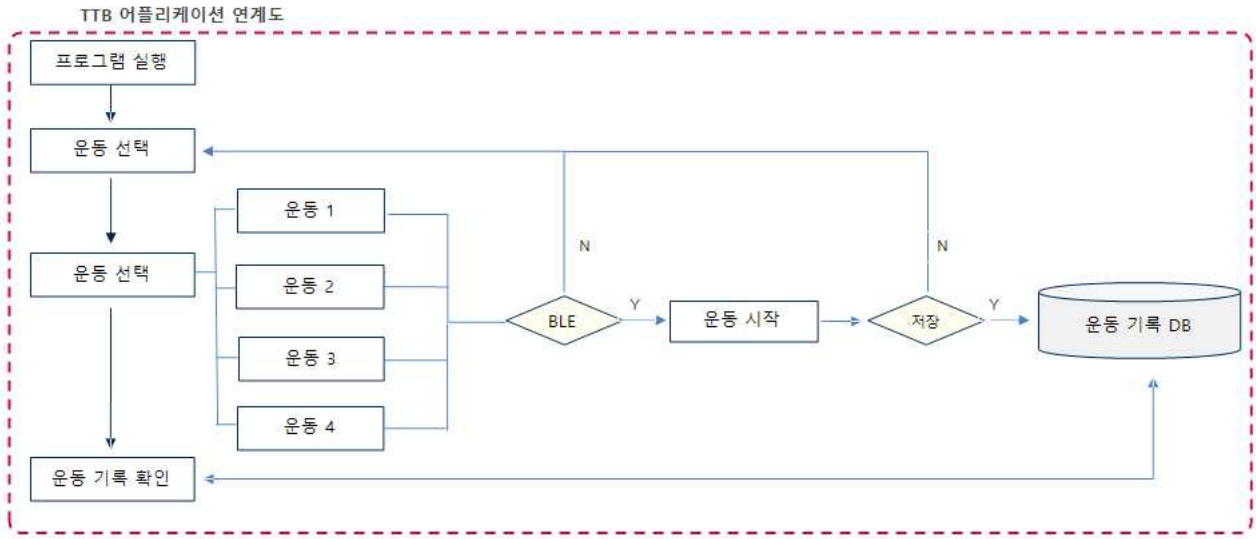


필요에 따라 배터리를 부착할 수 없을 경우 유선으로도 제작이 가능합니다.

Wired Version

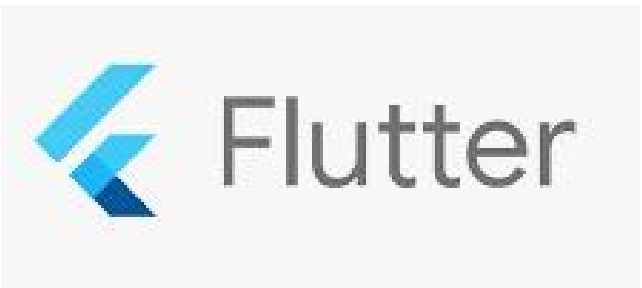
## 2. 소프트웨어 설계

먼저 소프트웨어의 기본적으로 실행되는 연계도입니다.




아두이노 IDE를 이용하여 ESP32를 제어하였습니다.


또한 APP을 만들기 위해 Flutter를 이용하여 구현했습니다.



아래의 기능들을 동작시키기 위해 ESP32 제어 코드와 Flutter 코드가 필요합니다.

## 2-1 ESP32 Code

BLE를 이용할려면 아래와 같은 코드를 사용해야 하며 이 작업을 통해 BLE 장치 간의 데이터 통신 구조를 정의하여 데이터를 사용할 수 있게 됩니다.

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
```

```
// 블루투스 서비스 UUID 및 특성 UUID 정의
#define SERVICE_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
```

또한 아래 코드는 BLE 연결 상태를 모니터링하고 동작을 관리할 수 있는 코드입니다.

```
// BLE 특성 콜백 클래스 정의
class MyCallbacks : public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic *pCharacteristic) {
    String value = pCharacteristic->getValue().c_str(); // 특성에 쓰인 값을 가져옴
    if (value.length() > 0) {
      Serial.print("Received Value: ");
      for (size_t i = 0; i < value.length(); i++) {
        char receivedChar = value[i];
        Serial.println(receivedChar); // 각 문자를 시리얼 모니터에 출력
      }
    }
  }
};
```

BLE를 제어 할수 있는 코드는 아래와 같습니다.

```
// BLE 서버 콜백 클래스 정의
class MyServerCallbacks : public BLEServerCallbacks {
  void onConnect(BLEServer *pServer) {
    deviceConnected = true;
    Serial.println("BLE 연결 성공");
  }
  void onDisconnect(BLEServer *pServer) {
    deviceConnected = false;
    Serial.println("BLE 연결 실패");

    // BLE 광고 재시작
    pServer->startAdvertising();
    Serial.println("BLE 광고 다시 시작");

    oneDataVector.clear(); // 기록된 데이터 초기화
    twoDataVector.clear();

    Serial.println("모든 동작이 초기화되었습니다.");
  }
};
```

아래코드는 개인 맞춤형 운동을 위해 Moving Filter를 적용한 코드입니다. 아래코드를 통해 사용자마다의 운동 특성을 반영하여 제공할 수 있습니다.

```
double movingFilter(int readings[], int size) {
  int window_size = 10;
  int moving_point = 1;

  // 첫 번째 윈도우 평균 계산
  double sum = 0;
  for (int i = 0; i < window_size && i < size; i++) {
    sum += readings[i];
  }
  double avg = sum / window_size;

  // 알파 계산
  double alpha = (double)(window_size - moving_point) / window_size;

  // 이동 평균 계산
  for (int i = moving_point; i <= size - window_size; i += moving_point) {
    if (i + window_size > size) {
      break; // 남은 데이터가 윈도우 크기보다 작으면 종료
    }

    // 현재 윈도우 평균 계산
    sum = 0;
    for (int j = i; j < i + window_size; j++) {
      sum += readings[j];
    }
    double current_avg = sum / window_size;

    // 이동 평균 업데이트
    avg = alpha * current_avg + (1 - alpha) * avg;
  }

  // 최종 이동 평균 값을 반환
  return avg;
}
```

## 2-2 Application

어플리케이션을 구현하기 위해 flutter를 사용했으며 아래 코드는 어플리케이션에서 ESP32로 문자열을 송신하는 함수입니다.

```
Future<void> sendData(String dataToSend) async {
  if (selectedDevice != null) {
    List<BluetoothService> services = await selectedDevice!.discoverServices();
    for (BluetoothService service in services) {
      for (BluetoothCharacteristic characteristic in service.characteristics) {
        if (characteristic.properties.write) {
          await characteristic.write(utf8.encode(dataToSend));
        }
      }
    }
  }
}
```

```dart
Future<List<double>> receiveRecord() async {
  final Completer<List<double>> completer = Completer();
  final List<double> receivedDataList = [];
  const Duration timeoutDuration = Duration(seconds: 1);
  Timer? timeoutTimer;
  if (selectedDevice != null) {
    List<BluetoothService> services = await selectedDevice!.discoverServices();
    for (BluetoothService service in services) {
      for (BluetoothCharacteristic characteristic in service.characteristics) {
        if (characteristic.properties.notify || characteristic.properties.indicate) {
          await characteristic.setNotifyValue(true);
          final StreamSubscription<List<int>> subscription = characteristic.lastValueStream.listen((List<int> value) {
            if (value.isNotEmpty && value.length >= 4) {
              int intValue = value[0] | (value[1] << 8) | (value[2] << 16) | (value[3] << 24);
              double data = intValue.toDouble();
              print('수신된 데이터: $data');
              receivedDataList.add(data);
              timeoutTimer?.cancel();
              timeoutTimer = Timer(timeoutDuration, () {
                if (!completer.isCompleted) {...}
              });
            }
          }, onDone: () {...}, onError: (error) {
            if (!completer.isCompleted) {...}
          });
          completer.future.whenComplete(() {...});
          break;
        }
      }
      if (completer.isCompleted) {...}
    }
  } else {...}
  timeoutTimer = Timer(timeoutDuration, () {...});
  return completer.future;
}
```

또한 환자의 운동데이터를 수신하는 함수는 아래에 있습니다. 이를 통해 사용자의 운동 진행 상황을 실시간으로 확인하고 기록을 저장할 수 있습니다.

## 2-2 서버

제작된 조음운동촉진기기는 범용 서버를 사용하여 운동을 기록할 수 있게 구현했습니다.

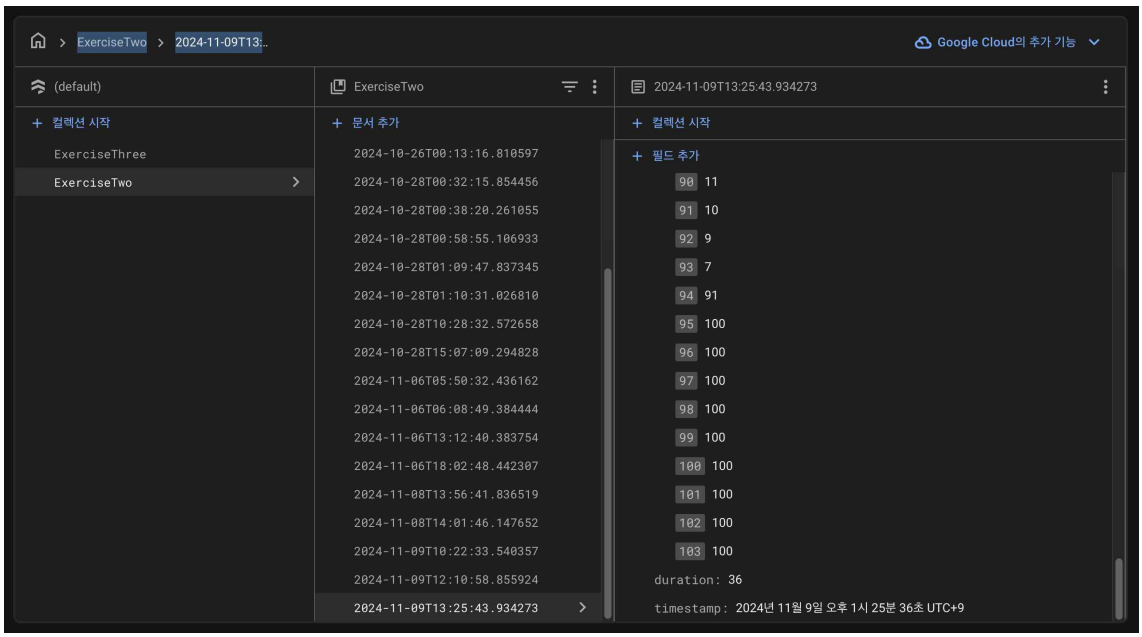
아래 코드는 운동 기록을 수신 받아 리스트 배열에 저장하는 함수이며, Firebase함수를 호출하여 저장된 리스트 배열을 전달합니다.

```
void saveRecord(int exerciseDurationInSeconds) async {
  final globalState = Provider.of<GlobalState>(context, listen: false);
  try {
    List<double> data = await globalState.receiveRecord();
    if (mounted) {
      setState(() {
        record.addAll(data);
      });
    }
    await saveRecordToFirestore(exerciseDurationInSeconds);
    save = true;
    if (save && mounted) {
      Navigator.of(context).popUntil((route) => route.isFirst);
      save = false;
    }
  } catch (e) {
    print("Error: $e");
  }
}
```

이 코드는 각각의 운동 이름으로 서버 컬렉션을 지정하여 해당 운동의 컬렉션을 찾아 기록을 저장하는 코드입니다. 운동 기록, 시간, 날짜를 저장할 수 있습니다.

```
Future<void> saveRecordToFirestore(int exerciseDurationInSeconds) async {
  final String documentId = DateTime.now().toIso8601String();
  final DocumentReference document = _firestore.collection('ExerciseTwo').doc(documentId);
  await document.set({
    'data': record, // 수신한 데이터
    'duration': exerciseDurationInSeconds, // 운동 시간(초)
    'timestamp': FieldValue.serverTimestamp(),
  }, SetOptions(merge: true));
  print('Exercise record saved');
}
```

서버를 구현하여 저장한 데이터들은 아래 사진에 있습니다.




아래의 사진은 소프트웨어를 종합적으로 합치고 완성된 결과물입니다.