



## Table of Contents

<b>Overview</b>	<b>1</b>
<b>Sending Raw Data to Back End</b>	<b>3</b>
<b>Sending Episodes and Duration to Back End</b>	<b>5</b>
<b>Data Processing Algorithms</b>	<b>5</b>

# Overview

This document explains the process for sending the data to the backend. There are two types of data you have to send, the “Raw Data”, which consists of the sensor values, and the “Episodes and Duration”, which consists of the number of episodes, episode time ranges, and total episode duration. 

The function below shows the overall function that handles the data to send it to the backend. This is what it does:

- Segments the data by day. The device may have data from multiple days of data and only the data for a singular day should be processed at once. A day is also defined differently in our case as denoted by the dayAdjust function included further below. 
- Within each segment
  - Calls the function to calculate the number of episodes, episode time ranges, and total episode duration and to remove useless raw data
  - Calls the function to send the Raw Data to the backend
  - Sends the Episodes and Duration to the backend

```
processSync = async () => {
  console.log("process sync hit")
  if(this.syncDataBuffer.length == 0){
    return
  }
  this.syncDataBuffer.sort(this.sampleOrder)
  let currentDate = this.dayAdjust(this.syncDataBuffer[0].ts)
  let dayData = []
  let analyzedReport = 0

  console.log("dataBuffer: ")
  console.log(JSON.stringify(this.syncDataBuffer[0]))

  for(let x = 0; x<this.syncDataBuffer.length; x++){
    let sampleDate = this.dayAdjust(this.syncDataBuffer[x].ts)
    if(sampleDate != currentDate){
      analyzedReport = this.packageAnalyzed(analyze(dayData))
      await this.sendToBackend(analyzedReport.processedData)
```

```

        this.errorLog(analyzedReport.sessionEpisodes+" Episodes
"+analyzedReport.sessionDuration+" Seconds")
        this.socket.emit('report', {
            user: this.state.userID,
            Date: new Date(this.fixedDateCheck(new Date(dayData[0].ts))),
            sessionEpisodes: analyzedReport.sessionEpisodes,
            sessionDuration: analyzedReport.sessionDuration,
            episodeList: analyzedReport.episodeList
        });
        dayData = []
        dayData.push(this.syncDataBuffer[x])
        await delay(100)
        currentDate = sampleDate

    }
    else{
        dayData.push(this.syncDataBuffer[x])
    }
}
analyzedReport = this.packageAnalyzed(analyze(dayData)
await this.sendToBackend(analyzedReport.processedData)
console.log(analyzedReport.sessionEpisodes+" Episodes
"+analyzedReport.sessionDuration+" Seconds")
this.errorLog(analyzedReport.sessionEpisodes+" Episodes
"+analyzedReport.sessionDuration+" Seconds")
this.socket.emit('report', {
    user: this.state.userID,
    Date: new Date(this.fixedDateCheck(new Date(dayData[0].ts))),
    sessionEpisodes: analyzedReport.sessionEpisodes,
    sessionDuration: analyzedReport.sessionDuration,
    episodeList: analyzedReport.episodeList
});
dayData = []
this.syncDataBuffer = []

```

## Function to adjust the day

Anytime before 5 pm is counted as the previous day.

```

dayAdjust = (date) =>{
  let dateConv = new Date(date)
  // console.log('initial date to convert: '+dateConv)
  let hour = dateConv.getHours()
  if(hour>=17){
    // console.log('dateConv: '+dateConv.getDay())
    return dateConv.getDay()
  }
  else{
    dateConv.setDate(dateConv.getDate()-1)
    //console.log('dateConv: '+dateConv.getDay())
    return dateConv.getDay()
  }
}

```

## Sending Raw Data to Back End

**Websocket name:** grindRatio

Object to send

JSON format

```

{
  user: String,
  data: [{
    gr: [{sensor1: Number, sensor2: Number}]
    ts: Date
  }],
}

```

The “data” field is an array of sensor data timestamped with Date values. Date values are formatted as Javascript Date objects.

Additionally, the data must be all from the same hour. For example, data from 9:35 am and data from 10:05 am must be in different documents.



Useless raw data must be removed prior to sending to the backend by using the data processing algorithms.



Here is some Javascript code showing how I send the raw data to the backend:

```
sendToBackend = async (dataArray) => {
  if(!dataArray){
    return
  }
  if(dataArray.length == 0){
    return
  }
  let datePackage = []
  let currentHour = dataArray[0].ts.getHours()
  datePackage.push(dataArray[0])
  for(let x = 1; x < dataArray.length; x++){
    if(dataArray[x].ts.getHours() != currentHour){
      this.socket.emit('grindRatio',{
        user: this.state.userID,
        data: datePackage,
      })

      await delay(100)

      datePackage = []
      datePackage.push(dataArray[x])
      currentHour = dataArray[x].ts.getHours()
    }
    else{
      datePackage.push(dataArray[x])
    }
  }
  this.socket.emit('grindRatio',{
    user: this.state.userID,
    data: datePackage,
  })
}
```

## Sending Episodes and Duration to Back End

**Websocket name:** report

Object to send

```
{
  user: String,
  Date: Date,
  sessionEpisodes: Number,
  sessionDuration: Number,
  episodeList: [{start: Date,
                  end: Date}]
}
```

The “episodeList” field is an array of start and end times. Date values are formatted as Javascript Date objects.

## Data Processing Algorithms

The functions to process analyze and process the data can be found in the “Data Processing Algorithms” folder.

These algorithms:

- Remove useless raw data
- Calculate number of episodes and total duration
- Return time windows for each episode

The “zeroFill.js” file contains the **analyze** function, which is the overall function that takes in the *arr* parameter, which is the array of timestamped raw data received from the device. The *noiseThreshold* parameter can be ignored. It also contains the **zeroFill** function called by the. The device only collects active bruxism events and ignores down time. The **zeroFill** function fills in these gaps in time with zeroes.

The “DBSCAN.js” file contains the **findEpisodes** function called by **analyze** and any functions that it calls to calculate the number of episodes, episode time ranges, and total episode duration.

The “filter.js” file contains the **highpass** function called by **zeroFill** to remove any useless data.