VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Nina Barnabishvili

# VISUALIZATION OF ENVIRONMENTAL NOISE

A sound map of Technobothnia Laboratory

Technology and Communication
2020

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

# ABSTRACT

| | |
|---|---|
| Author | Nina Barnabishvili |
| Title | Visualization of Environmental Noise |
| Year | 2020 |
| Language | English |
| Pages | 26 |
| Name of Supervisor | Gao Chao |

Nowadays the problem of noise pollution occurs in megapolises and coworking areas. Exceeding the noise level of recommended 40 dB leads to faster cognitive decline, hearing loss, sleep disturbances, and other harmful and disturbing effects. /1, 2/

The main goal of the thesis was to develop an application, which helps with the noise data visualization for a better understanding of the problem and the possible ways of solutions. Providing this product in coworking areas, for instance, could decrease the harmful consequences.

The front-end side of the application was implemented with the React JavaScript framework. The back-end part was developed using the Raspberry Pi 3B with WebSocket module. The server was written on Python 3 and deployed with the front-end part to Heroku.

The application can be use in coworking areas where it can decrease harmful consequences of noise. The system proved to be reliable during the testing process. Moreover, the application works correctly on mobile devices that provide easy access to the data even re-motely.

Keywords React JS, Raspberry Pi 3, WebSocket, Python 3, Heroku

# CONTENTS

## LIST OF FIGURES AND TABLES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| Amazon's EC2 | Amazon's Elastic Compute Cloud |
| CSS | Cascading Style Sheets |
| DOM | Document Object Model |
| HDMI | High-Definition Multimedia Interface |
| HTTP | Hypertext Transfer Protocol |
| MIME | Multipurpose Internet Mail Extensions |
| PaaS | Platform as a service |
| PHP | Hypertext Preprocessor |
| RPi3 | Raspberry Pi 3 |
| TCP/IP | Transmission Control Protocol / Internet Protocol |
| UI | User Interface |
| USB | Universal Serial Bus |

# 1 INTRODUCTION

Noise pollution is a significant problem nowadays. The nature is not quiet itself: weather, species and other – all produce million different sounds. Animals use acoustic signals for communication, navigation and warning each other. However, all these natural sounds are not harmful to life forms. The most widespread harmful noises are anthropogenic. The human impact on natural soundscapes started with the growth of population on Earth – around 2.6 million years ago. From the Stone Age to the invention of the gunpowder in China, it proliferated. It became a real problem after the industrial revolution and invention of the steam engine. /3/

Acoustic overexposure affects humans as well as animals. It may cause permanent or temporary hearing damage or even deafness. Moreover, people may suffer from sleeping problems, chronic stress, cognitive disorders and regress of work performance. For animals, it might cause hearing problems too, or even physical damage. The species, which hearing range is not the same that people able to hear, may still have behavioural defects. Overall, the problems of acoustic climate-changing have to arise. /3/

It is crucial nowadays to follow noise hygiene and track soundscape. Thus, the application "Sound map of Technobothnia" was implemented to show the volume levels in the laboratory. The thesis demonstrates React.js and Python development, which provides the visualisation of noise with the help of Raspberry Pi nodes.

The purpose during this project was to gain full-stack development skills, learning the WebSocket library usage and implement a real-working solution for the actual problem.

# 2 APPLICATION REQUIREMENTS

## 2.1 Application Overview

The use-case diagram below (**Figure 1**) shows the application structure.

It helps to understand the relationship between different components of the system. The diagram does not show all the details of the application and only exposes the core parts of the project implementation.

The point where a user interacts with the application is the User Interface written with the help of React JS library. The React client communicates with the server by establishing a WebSocket connection. The server receives necessary data from Raspberry Pi clients and sends average and maximum volume levels, IP and node order number of the RPi clients to the React client.

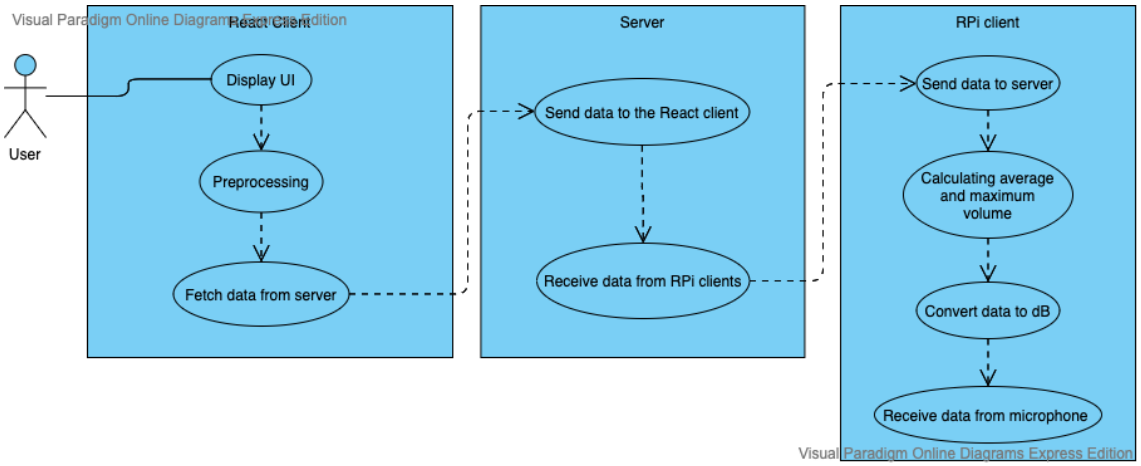


**Figure 1.** Use-case Diagram

## 2.2 Application Functionality

### 2.2.1 Volume and Position Display

The main screen of the client application displays the RPi clients' positions, their IP addresses, average volumes and maximum volumes. From it, a user can identify the location of the nodes and noise intensity in the area around them. This screen serves as

the main point of interaction for the user and provides access for other functionalities of the application.

In the picture below (**Figure 2**), there are multiple points; each represents the node measuring volume levels in the allocated area. An IP address of connected RPis accompanies every node. The inner area of the circles (in dark blue) represents the maximum volume, and the outer area (in light blue) displays the average volume of the noise in the area.

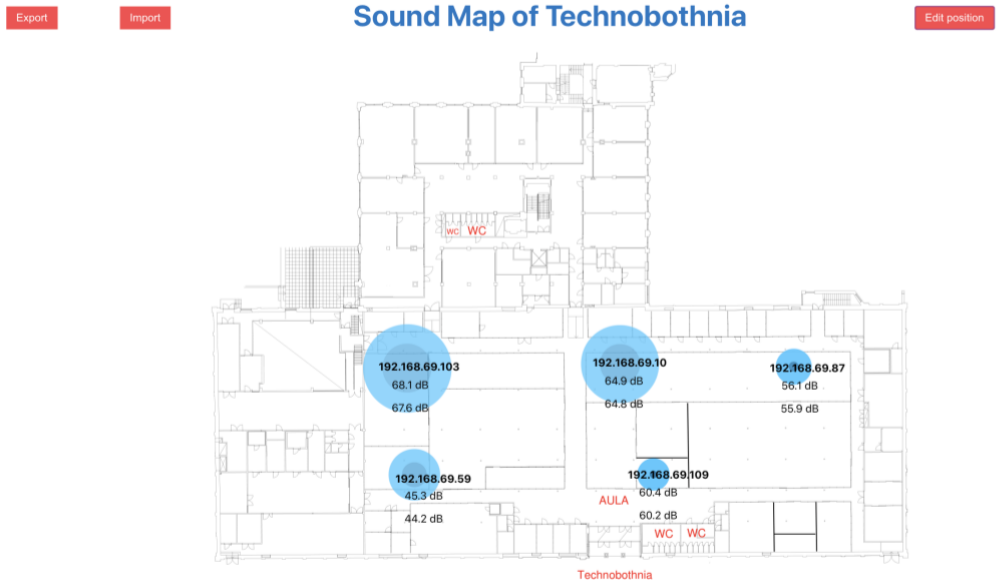


**Figure 2.** Main screen

### 2.2.2 Export Nodes

The "Export" button on the top left corner of the screen provides the export functionality. It allows a user to save the current positions of nodes inside the JSON file. This feature might serve useful to users who want to save the node positions for future use temporarily.

### 2.2.3 Import Nodes

The “Import” button on the top left corner refers to the data import functionality. A user can upload the saved node positions from the local machine. The server saves the

positions of points from the previous browser session; however, a user may import them from another machine.

### 2.2.4 Adding Nodes

Additional nodes append to the main screen by pressing the “Edit” button. A user can select the node from the drop-down list, and it will appear in the centre of the screen. Afterwards, the position of the node can be changed.

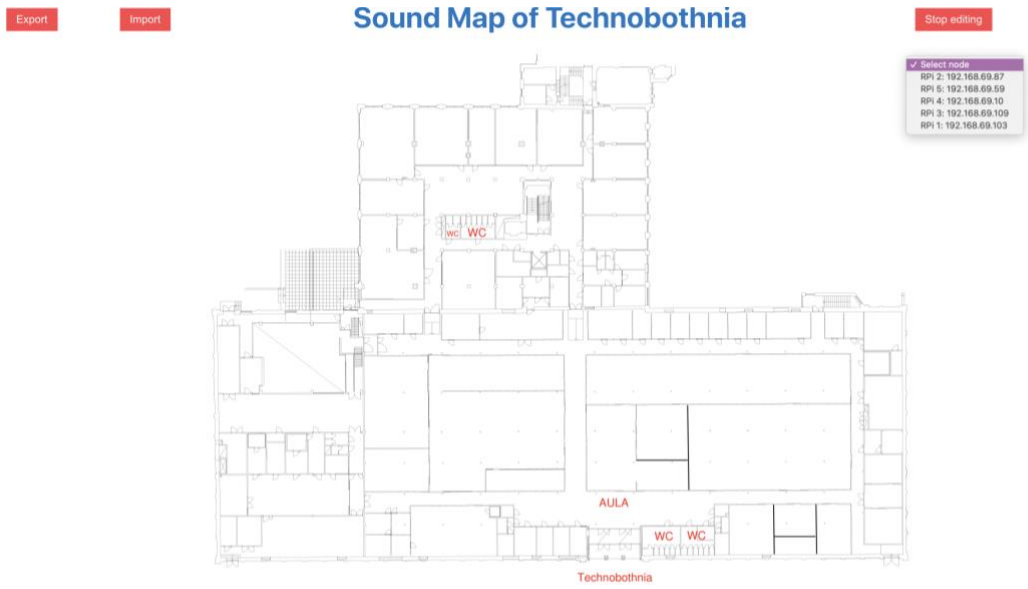


**Figure 3**. Editing mode

### 2.2.5 Editing Nodes

A user might edit nodes by clicking the “Edit” button. The position of nodes can be changed by dragging the points and dropping them in the assigned area. The location should be equivalent to the position of RPi on the premises. The exit from the editing mode implements by clicking the “Stop editing” button.

It is impracticable to change the order number of the node via the main screen. If this is needed, it can be done by editing the configuration file of RPi. Thus, the physical label on the RPi cover has to be changed as well.

### 2.2.6 Removing Nodes

The deletion of the node can be achieved by clicking the “Edit” button and pressing the cross icon (in red) on the top right corner of the outer circle of a particular node. All the data will be removed from the server, likewise. Once a user removes the node, after adding it anew, it will appear in the centre of the main screen.

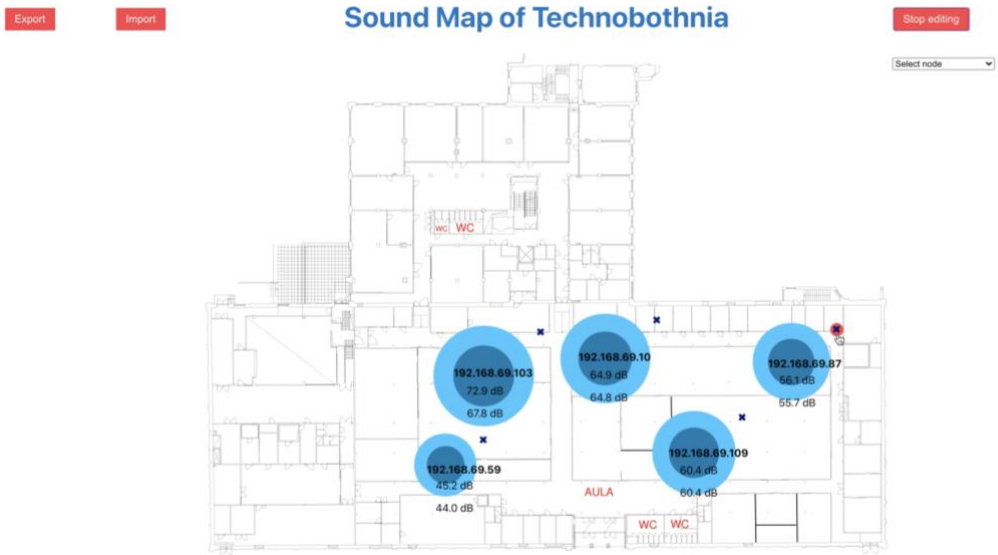


**Figure 4.** Removing nodes

# 3 TECHNOLOGIES AND TOOLS

## 3.1 Raspberry Pi 3

Developed as a low-cost and small-sized single-board computer, Raspberry Pi 3 Model B, hereinafter RPi3, is a proper solution for maintaining a small node client to transfer data and send requests to a server. RPi3 includes a 1.2 GHz 64-bit quad-core processor, onboard Wi-Fi, Bluetooth and Universal Serial Bus boot capabilities. To establish the working set, a user has to have a Micro SD card and a Micro USB power supply separately. To configure RPi3, a user should have NOOBS installed to an SD card and have monitor, High-Definition Multimedia Interface cable, keyboard and mouse. /4/

## 3.2 WebSocket

According to Mozilla Developer Network Web Documents, WebSocket is “an advanced technology that makes it possible to open a two-way interactive communication session between the user’s browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply”. /5/

This technology enables two-way communication which is used to provide the functionality of updating the state of the application with a response to the changes on other clients. The server communicates with the clients continuously by utilizing WebSocket technology. This way, it is possible to keep track of the most relevant data in real-time.

## 3.3 React JS

React is an open-source JavaScript library for UI building. Maintained by Facebook, it provides a base for the development of single-page or mobile applications, rendering data to the Document Object Model. Reconciliation allows to write code while the webpage is rendered on each change, and the React libraries only render subcomponents that change. It works by creating an in-memory cache, computes the resulting differences, and then updates the displayed DOM efficiently. React affords a significant performance boost and keeps the effort of recalculating the Cascading Style Sheets, layout and rendering for the page. /6/

### 3.4 Python 3

Being developed by Guido van Rossum and first released in 1991, Python is a high-level programming language that is interpreted and general-purpose. The main design philosophy aim of Python is to help programmers write clear and logical code for various size projects. Moreover, it supports multiple programming paradigms, such as structured, object-oriented, and functional programming. Released in 2008, Python 3.0 was a significant revision of the language that much Python 2 code does not run unmodified on Python 3. Its sizeable standard library provides tools suited to many tasks: standard formats and protocols (Multipurpose Internet Mail Extensions and HTTP) for Internet-facing applications, modules for creating graphical user interfaces, connecting to databases, generating pseudorandom numbers, arithmetic with arbitrary-precision decimals, manipulating regular expressions, and unit testing. /7/

### 3.5 Visual Studio Code

Visual Studio Code is a lightweight, robust source code editor which is available for various operating systems. It has built-in support and extensions for many programming languages: JavaScript, TypeScript, Node.js, C++, Java, Python, Hypertext Preprocessor and many others /8/. Features involve support for debugging, syntax highlighting, snippets, code refactoring, and pre-installed Git system. Users can install extensions that add additional functionality. Alternatively of a project system, VS Code allows users to open one or more directories and save them in workspaces for later reuse. This feature will enable it to operate as a language-agnostic code editor for any language. /9/

### 3.6 Heroku

Heroku is a Platform as a service that allows developers to build, deploy and manage monitor and scale apps. In June 2007 it supported only the Ruby programming language, but now includes Java, Node.js, Scala, Clojure, Python, PHP, and Go. Running by Heroku. Applications have a unique domain, which routes HTTP requests to the right application container. All the domains form a "dyno grid" or "domain grid" which consists of several servers. Heroku's own Git server manages application repository pushes from

authorized users. Amazon's Elastic Compute Cloud platform hosts all Heroku servers. /10/

### 3.7 SSH

Secure Shell or SSH stands for a cryptographic network protocol for a secure connection (Figure 5) between a client and a server inside an unsecured network, such as the Internet. The protocol operates in the client-server architecture and drives the connection setup process, using automatically generated public-private key pairs to validate the identity of the SSH server. After the setup phase, the SSH protocol uses secure symmetric encryption and hashing algorithms to ensure the confidentiality and uprightness of the data that transfers between the client and the server. /11/

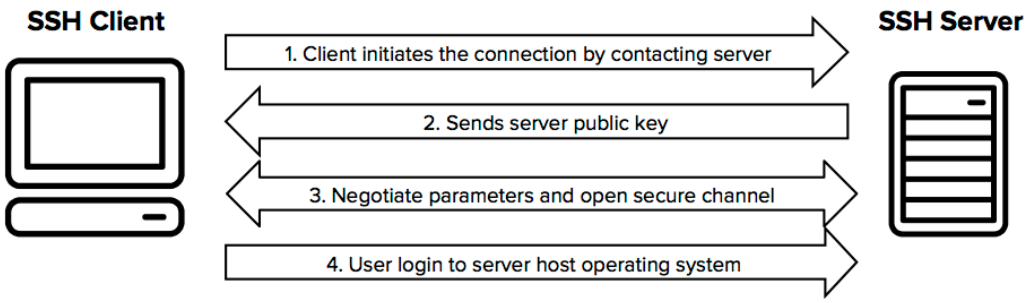


**Figure 5.** The structure of a practical SSH connection

An SSH client program establishes connections to an SSH daemon accepting remote connections. It works within the most modern operating systems: macOS, most distributions of Linux, OpenBSD, FreeBSD, NetBSD, Solaris and OpenVMS. There are various versions of freeware and open-source clients, such as PuTTY and OpenSSH. /12/

### 3.8 PuTTY

Developed by Simon Tatham, PuTTY is a versatile free and open-source terminal emulator and network file transfer application for the Windows platform. It supports several network protocols: SCP, SSH, Telnet and raw socket connection. The main feature of the product is the terminal window. PuTTY does not support session tabs directly, however many wrappers are available. /13/

# 4 APPLICATION DESIGN

## 4.1 Project Structure

The project consists of three main parts: RPi client, React client and server.

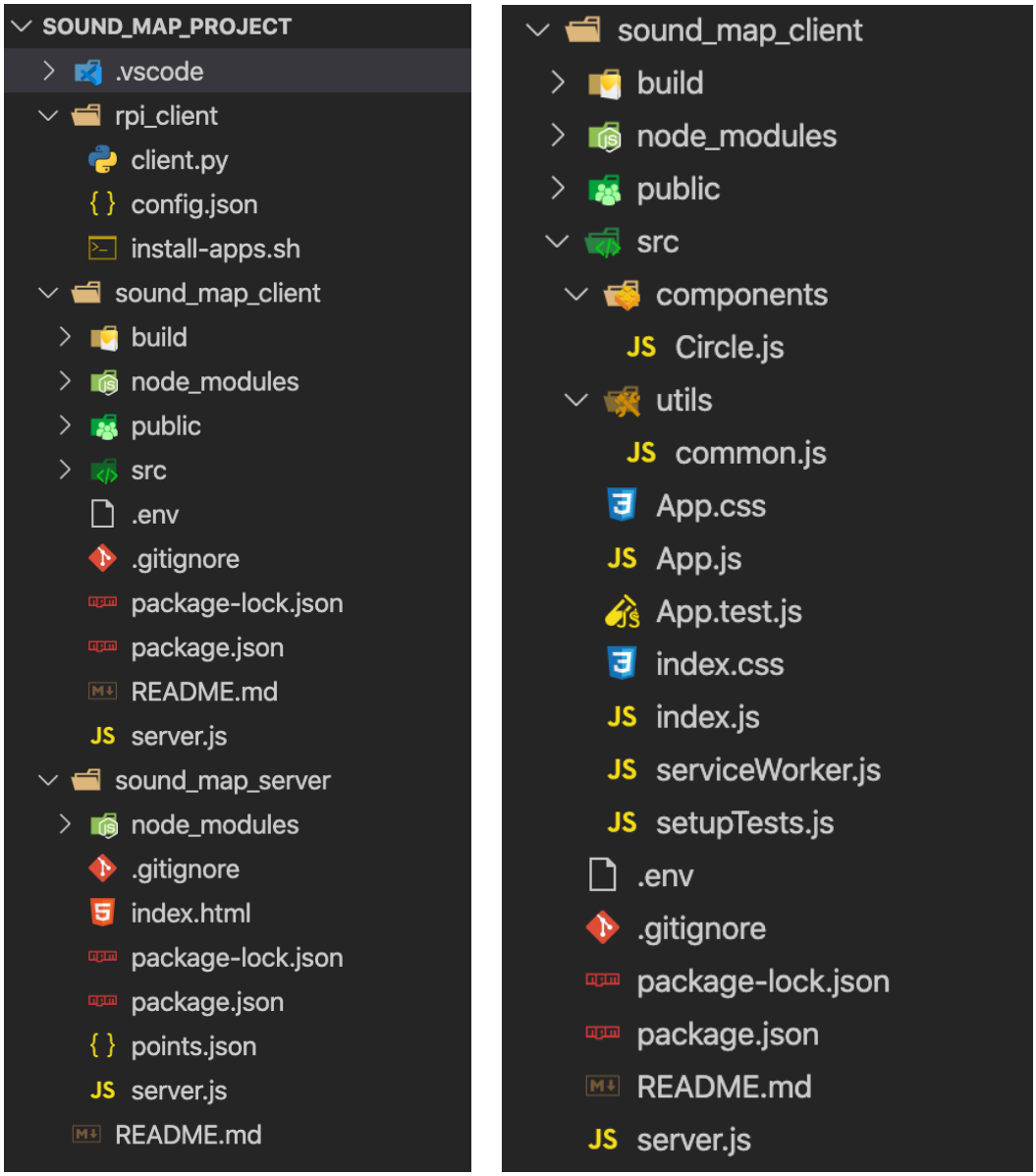


**Figure 6**. Project structure

The “rpi_client” folder contains “client.py”, which includes all the necessary code to run the client application. The object with node properties locates in “config.json” file.

Each node has a unique “node_id” property, which has to be changed while establishing a new node and additionally written on the RPi case. Moreover, there is an “install-apps.sh” file to install Python 3 and necessary modules to the RPi by running a “sh” command on it.

The “sound_map_client” folder (Figure 6 right side) includes all the code for the React client, the “scr” folder inclusively, which carries the source code written during the development process of the React client. This folder holds several files as well as two sub-folders: “components” and “utils”. The file “App.js” is the entry point to the program and serves as the top node of the component tree. Inside the “components” folder, there is a “Circle.js” file that is responsible for rendering volume levels of RPi clients. The utility functions are located inside the “common.js” file, which is present under the “utils” folder. Besides, “sound_map_client” folder contains “.env” file with variables to establish the connection with the server.

The folder "sound_map_server" contains "server.js", "points.json" and other files related to source control and package management. The "server.js" file includes the code that runs when the server starts. The "points.json" file stores an array with nodes' positions and volume data.

## 4.2 Data Management

### 4.2.1 RPi Client to Server Communication

The communication between RPi client application and server implements by using the WebSocket protocol. Firstly, the client connects to the server using the information from the configuration file. The RPi client receives data from a microphone, which connects to an external audio card. The audio card connects to the USB-port embedded into the RPi. Then this data is then used to calculate average and maximum volume levels within a specific time. A JSON object is then generated from the processed data and sent to the server.

### 4.2.2 React Client to Server Communication

The WebSocket protocol is required to establish the communication between the React client application and the server too.

The application establishes a connection with the server and waits for the data dispatched from it. The server broadcasts the data received from the RPi clients to the React clients to visualize it. Later the JSON file received from the server is parsed to a JavaScript object and used to display data on the screen.

## 4.3 Component Structure

### 4.3.1 App.js

The “App.js” component (Figure 7**Error! Reference source not found.**) is responsible for rendering most of the user interface as in Figure 2 such as the map of Technobothnia, “Edit” button, “Import” button, “Export” button and the drop-down list of nodes. Buttons are used to trigger user operations and data communications. It also renders a child component “Circle.js” that discussed in more detail below.

```
290   return (
291     <div className="App">
292       <header className="App-header">
293         <h2> Sound Map of Technobothnia </h2>
294         <button onClick={handleExportJson} className="exportButton">
295           Export
296         </button>
297         <button onClick={handleStartImport} className="importButton">
298           Import
299         </button>
300         <input
301           type="file"
302           id="fileInput"
303           style={{ display: "none" }}
304           onChange={handleImportJson}
305         />
306         {isEditing ? (
307           <button onClick={stopEditPosition} className="editingButton">
308             Stop editing
309           </button>
310         ) : (
311           <button onClick={editPosition} className="editingButton">
312             Edit position
313           </button>
314         )}
315         {showRpiList ? (
316           <select onChange={handleAddVolumePoint}>
317             <option key="select id" value="">
318               Select node
319             </option>
320             {staticEditData
321               .sort(
322                 (first, second) =>
323                   Number(first.node_id) - Number(second.node_id)
324               )
325               .map((d) => (
326                 <option key={d.node_id} value={d.id}>
327                   RPi {d.node_id}: {d.id}
328                 </option>
329               ))}
330           </select>
331         ) : null}
332       </header>
333       <div className="Content" ref={divRef}>
334         <img src="tb.png" className="tbImage" />
335         {volumePoints.map((point) => {
336           const { id, x, y, volX, volY, volDb, averageVolDb, disabled } = point;
```

**Figure 7.** App.js code example

Moreover, the “App.js” component contains the logic for the connection with the server (Figure 8).

```
337
338           return (
339             <Circle
340               onMouseDown={(event) => {
341                 handleDragStart(id, event);
342               }}
343               onMouseUp={() => {}}
344               disabled={disabled}
345               sizeX={volX * 0.5}
346               sizeY={volY * 0.5}
347               posX={x}
348               posY={y}
349               id={id}
350               volDb={volDb}
351               averageVolDb={averageVolDb}
352               isEditing={isEditing}
353               onDelete={
354                 isEditing
355                   ? (event) => {
356                       handleDelete(id);
357                     }
358                   : undefined
359               }
360               key={id}
361             />
362           );
363         })}
364       </div>
365     </div>
366   );
367 }
368
369 export default App;
370
```

**Figure 8.** App.js code example

### 4.3.2 Circle.js

The “Circle.js” component (Figure 9) is responsible for rendering the visualization of the volume levels in the form of blue circles. This component accepts “props” from the parent component and utilizes them to display a circle of the correct size on the user’s screen.

```
53    return (
54      <div
55        onMouseDown={props.onMouseDown}
56        onMouseUp={props.onMouseUp}
57        onMouseMove={props.onMoveCircle}
58        id={id}
59        className={`CircleRoot ${disabled && "Disabled"}`}
60        style={{
61          width: `${sizeX}%`,
62          height: `${sizeY}%`,
63          left: `${newPosX}%`,
64          top: `${newPosY}%`,
65        }}
66      >
67        <div
68          onClick={props.onDelete}
69          className="circle-delete"
70          style={{
71            left: `100%`,
72            top: `0%`,
73            display: isEditing ? undefined : "none",
74          }}
75        >
76          <p>&#10006;</p>
77        </div>
78        <div
79          className="pulsating-circle-ave"
80          style={{
81            animation: isEditing ? "none" : undefined,
82          }}
83        >
84          <div
85            className="pulsating-circle-max"
86            style={{
87              animation: isEditing ? "none" : undefined,
88            }}
89          />
90        </div>
91        {!disabled && (
92          <div className="circle-volume">
93            <p>{volDb.toFixed(1)} dB</p>
94            <p>{averageVolDb?.toFixed(1)} dB</p>
95          </div>
96        )}
97        <div className="circle-id">
98          <p>{id}</p>
99        </div>
100     </div>
101   );
102 }
103
```

**Figure 9.** Circle.js code example

# 5 IMPLEMENTATION

## 5.1 Setup

In order to start developing the application, it the use of several tools is required. As an IDE, the Visual Studio Code is lightweight and provides the possibility to install necessary plugins for the developing client application and server. It completely suits the needs of application implementation.

A user client application was written with the help of React JavaScript library because it is suitable for fast development pace. Additionally, the "npm" package manager has been used to install libraries for the project. It requires the WebSocket library additionally to establish a connection with the server.

The implementation of the RPi client application required using the Python language. The implementation of the solution took less time compared to other programming languages. As in the React client, it utilizes the WebSocket library for the communication between client and server. To record sound samples, the "sounddevice" library is necessary to use.

According to the theoretical background, it was important to use the sample rate of 44100 Hz and 1 channel sound. The array of samples has been recorded every one second. Every measured value has been converted to decibels.

The server application uses the "express" library, which allows hosting the server on a local machine. Moreover, the "fs" library allowed us to read and write files with the node ' positions to the hard drive. The WebSocket library is also used here to communicate with clients.

## 5.2 Deployment

Using the Heroku cloud platform to deploy the system enabled running and operating the application in the cloud. Both the front-end and server got received unique domains for the remote access. It also uses the git system for project management, which makes the deployment process easy to follow.

Heroku held configuration, orchestration, load balancing, failovers, logging and security. The useful feature called “Heroku Runtime” run the front-end part of the application and the server inside the containers, where it manages their runtime environment.

### 5.3 Field Testing

The field-testing took place at the Technobothnia laboratory (Figure 10). All the materials, including RPis, Ethernet cables, microphones and charging cables, were provided by the thesis supervisor on behalf of the university. University-owned desktops have been working as the power source for RPis. According to the project scope, all nodes were placed on a reasonable distance from each other and marked with the associated node order number. After turning the system on, the verification of every node occurred for the proper Internet and microphone connection. All the measurements have been compared to the correct possible data, taking into account the usual volume level in the laboratory.

There were several rounds of testing, which lead to amendments and improvements to the project. Thus, the UI was completed with the mode when one or more nodes do not respond to the server.


**Figure 10.** Testing in the laboratory

# 6 CONCLUSION

The system proved to be reliable during the testing process. A feature to show the status of node connection in real-time was added to it. The supervisor has checked the formula of maximum and average sound level used in the application. Then it was corrected to provide an efficient result to the end-user. Moreover, the application works correctly on mobile devices that provide easy access to the data even remotely.

The installation process is accomplished by executing a few steps, which provides good user experience during the setup. The user-friendly interface helps to focus more on data and adjust the location of nodes more precisely. Besides, the system is reachable remotely, and there is no need to check every node physically.

The system is easy to maintain indoors since it requires just the Internet connection and power source. On the other hand, in case there is an enormous number of nodes needed, they all should be transported and maintained at the corresponding place. It makes the user experience difficult with regard to time.

The Sound Map of Technobothnia satisfies all the requirements of the development work. The main challenges in the scope of this project were the full-stack experience with back-end development. There are possibilities to improve the application by enhancing the design and adding more features; however, it would require more time to implement.

# 7 REFERENCES

/1/ Noise pollution. Wikipedia. Accessed 23.04.2020.
https://en.wikipedia.org/wiki/Noise_pollution
/2/ Data and statistics. World Health Organization. Accessed 23.04.2020.
https://www.euro.who.int/en/health-topics/environment-and-health/noise/data-and-statistics
/3/ Slabbekoorn, H. 2019. Noise pollution. Current Biology 29. R942–R995. Elsevier Ltd
/4/ What is Raspberry Pi. The Raspberry Pi Foundation. Accessed 24.04.2020.
https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/
/5/ WebSocket. MDN Web Docs. Accessed 27.11.2020.
https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
/6/ React JS. Wikipedia. Accessed 16.05.2020.
https://en.wikipedia.org/wiki/React_(web_framework)
/7/ Python. Wikipedia. Accessed 16.05.2020.
https://en.wikipedia.org/wiki/Python_(programming_language)
/8/ Visual Studio Code. Microsoft Software. Accessed 30.05.2020.
https://code.visualstudio.com/docs
/9/ Visual Studio Code. Wikipedia. Accessed 30.05.2020.
https://en.wikipedia.org/wiki/Visual_Studio_Code
/10/ Heroku Cloud Platform. Heroku. Accessed 30.05.2020.
https://www.heroku.com/about
/11/ SSH Protocol. SSH Communication Security. Accessed 30.05.2020.
https://www.ssh.com/ssh/protocol/
/12/ SSH Secure Shell. Wikipedia. Accessed 30.05.2020.
https://en.wikipedia.org/wiki/Secure_Shell
/13/ PuTTY SSH Client. SSH Communication Security. Accessed 30.05.2020.
https://www.ssh.com/ssh/putty/