Senior Design
ENG EC 463

Test Report

To:	Professor Pisano
From:	Arley Trujillo, Shivani Bhatia, Laura Salinas, Priya Kapadia, Steven Graham
Team:	14
Date:	02/16/18
Subject:	Second Deliverable Test Report

[··]vobot

1. Project Objective

1.1. The overall objective of this device is to help children on the autistic spectrum develop their articulation and communication skills.

2. Test Objective & Significance

2.1. Migration to CHiP Robot

2.1.1. The objective in switching to CHiP lies in a wider range of expressions that can be leveraged as reward behaviors for end users. In addition to more expressions and actions, CHiP looks and acts like a dog, giving it a more friendly appeal and a higher chance to stimulate children. The previous robot, MiP, also had a stability problem and could not stand without being in motion. It would often fall, possibly alarming the child in the middle of a session.

2.1.2. This deliverable is essential because it ties together the vocalization evaluation with the reward behavior that motivates the child to continue learning. In having a more stable, fun and excitable "instructor" the child will be more willing to participate in lessons and continue growing with the platform. In testing this component we made sure that we had control of the robots actions and that it behaved as programmed when a score is produced.

2.2. Database

2.2.1. The objective of the database is to store all elements of a child's progress and make that information accessible for parents and speech language pathologists. The test for this deliverable is to run through various push and pull commands to the database by using a RESTful API, and viewing the contents of the database afterwards. Specifically, once the API is run through terminal, we should be able to call programmed get/post routes in order to add, change, or delete elements from the database using Postman. Postman will then display the returned output of the database so that the results can be verified. With this implementation, the power of the cloud can be leveraged to maintain a small application size instead of being weighed down by local user data.

2.2.2. This deliverable is essential because it creates the storage for our product. Testing this deliverable provides a better understanding of what elements are vital to store in our database and a working model that can be modified to better fit the needs of our application upon integration. With this implementation, the power of the cloud can be leveraged to maintain a small application size instead of being weighed down by local user data.

2.3. Integrating Speechace API to Android Application

2.3.1. The objective of integrating Speechace with the application is to be able to run a complete loop of the system directly from the Android device. Speechace will break down the user vocalizations into phonemes and syllables, and provides a similarity score on each. Now, with its integration on the application we can move forwards and begin crafting leveling schemes for rewards based on user progression.

2.3.2. This deliverable was essential to test in order to demonstrate that a complete loop of our system is functional and running solely from the Android application. In addition, by integrating Speechace we were able to start designing the leveling schema for which reward behaviors are based on. By having this API fully integrate with the rest of the application a majority of the project components are taken care of; the rest of the design consists of polishing minor details.

2.4. Saving Audio Recordings

2.4.1. By having the ability to save audio recordings, our objective was to be able to then pass these files into Speechace for a return output of similarity score. The API requires a recording of the vocalization in specific formats before breaking down the word to generate a result of how accurate the vocalization is to the model word.

2.4.2. This deliverable was important for us because without it we would not be able to communicate to Speechace what our users were saying at the time of learning. Once we were able to save audio files locally we could then move on to attach those files to the API call for Speechace.

2.5. Reward Actions

2.5.1. The objective of reward actions are to encourage nonverbal autistic children to begin speaking in a safe and comforting environment. In order to allow for progression across word discovery, our ability to provide positive reinforcement is crucial to the motivation of our users.

2.5.2. This test is important because we want to make sure that the actions the robot is performing are stimulating enough for users and obvious enough that they signify positive reinforcement. By testing different actions we are able to see which ones provide the most stimulation and which would work negatively towards our users; i.e., scaring them or creating fear of the robot “instructor”.

2.6. 3D Printed Speaker Harness

2.6.1. The goal for this component is to create a sturdy, reliable harness for CHiP to wear which will hold the Bluetooth speaker. As children interact with the robot, it is important that the instructions to start practicing their vocalizations appear to be coming from the robot itself.

2.6.2. Unfortunately due to time constraints we were only able to print one of the parts for the harness at the time of testing so we presented a makeshift model of what the fully assembled parts would look like. The current designs include a “slip on” harness that hugs the belly of the robot with an enclosure for the speaker on top, as well as a design that comes together like two collars which slide through the enclosure holding the speaker. This test was significant in order to accurately size the printed models to the robot and make adjustments as we find incongruities.

3. Equipment and Setup

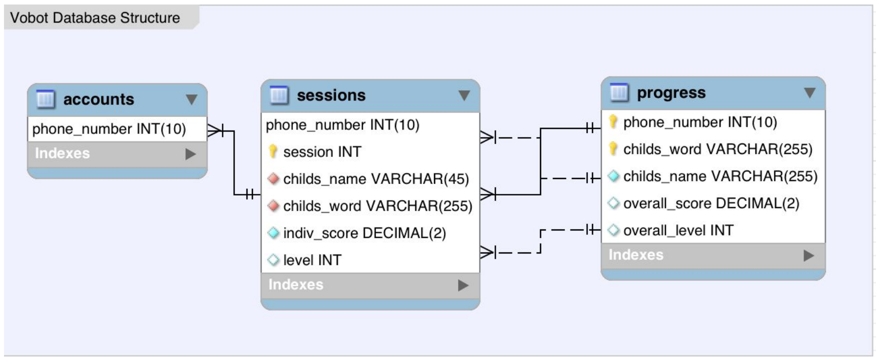
3.1. Migration to CHiP Robot

3.1.1.

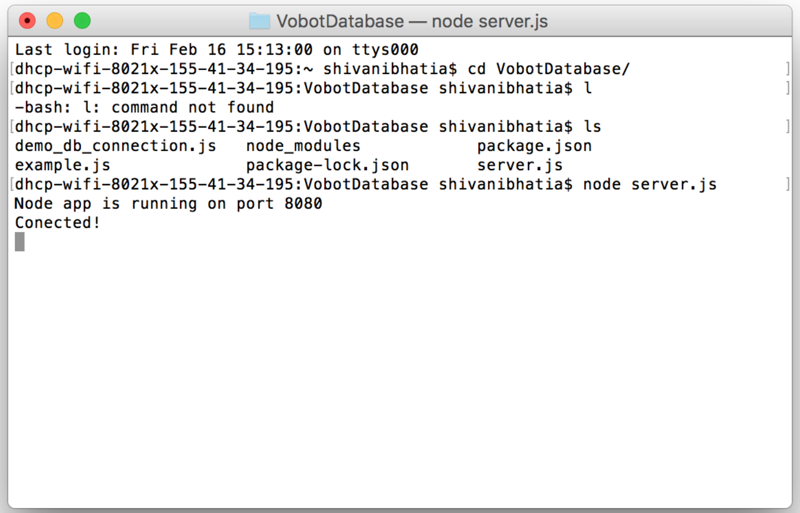
3.2. Database

3.2.1. In order to test the database, begin by downloading the server.js file from https://github.com/priyakapadia/Vobot/tree/shivani to the directory where you want to test the database. Modify the credentials of your database in the server.js file. You will need to have Node.js installed on your machine. Create a package.json file in your directory by running npm init on terminal.

3.2.2. Create a MySQL database on your computer following the diagram below:

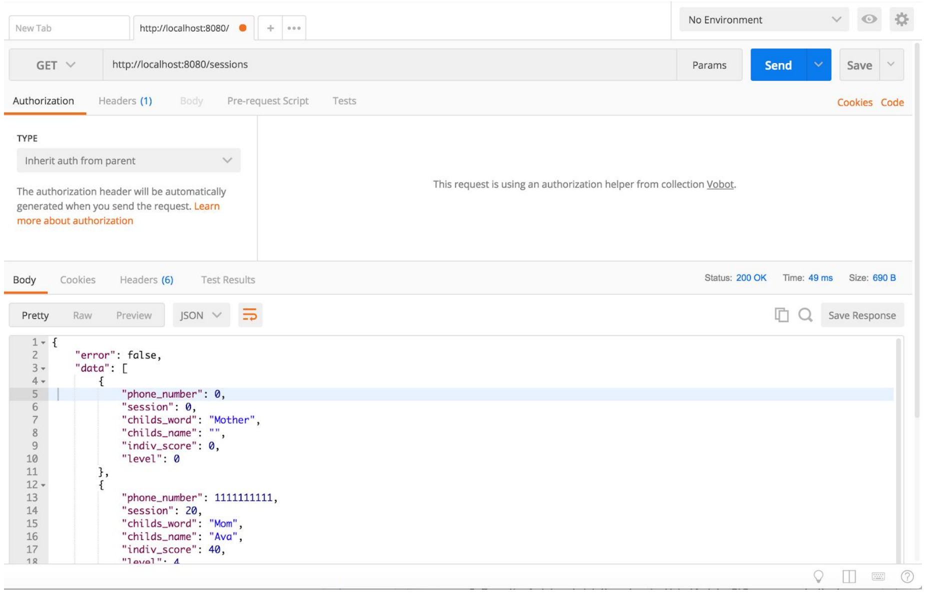


3.2.3. Once the database is created and the server.js file has been modified, run the command 'node server.js' on terminal. This will launch a localhost server and establish a connection to the database.



```
Last login: Fri Feb 16 15:13:00 on ttys000
dhcp-wifi-8021x-155-41-34-195:~ shivanibhatia$ cd VobotDatabase/
dhcp-wifi-8021x-155-41-34-195:VobotDatabase shivanibhatia$ l
-bash: l: command not found
dhcp-wifi-8021x-155-41-34-195:VobotDatabase shivanibhatia$ ls
demo_db_connection.js   node_modules            package.json
example.js              package-lock.json       server.js
dhcp-wifi-8021x-155-41-34-195:VobotDatabase shivanibhatia$ node server.js
Node app is running on port 8080
Conected!
```

3.2.4. Open Postman and test different routes by typing http://localhost:8080/ and changing the extension in the address bar. To retrieve information from the database, use the 'get' command. To push information to the database, use the 'post' command.



http://localhost:8080/sessions

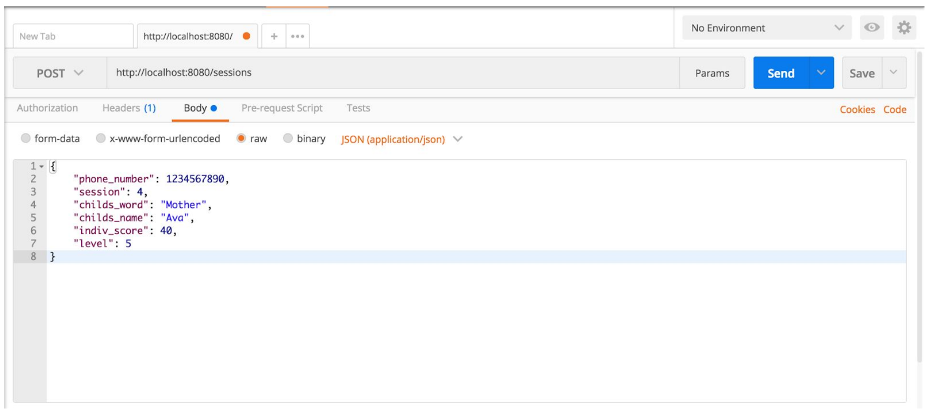
The authorization header will be automatically generated when you send the request. Learn more about authorization

This request is using an authorization helper from collection Vobot.

Status: 200 OK Time: 49 ms Size: 690 B

```
{
    "error": false,
    "data": [
        {
            "phone_number": 0,
            "session": 0,
            "childs_word": "Mother",
            "childs_name": "",
            "indiv_score": 0,
            "level": 0
        },
        {
            "phone_number": 1111111111,
            "session": 20,
            "childs_word": "Mom",
            "childs_name": "Ava",
            "indiv_score": 40,
            "level": 4
```

3.2.5. If you use the post command, type the parameters of the database you want to push to, and the information you want to push, into the body, as shown below:



```
{
    "phone_number": 1234567890,
    "session": 4,
    "childs_word": "Mother",
    "childs_name": "Ava",
    "indiv_score": 40,
    "level": 5
}
```

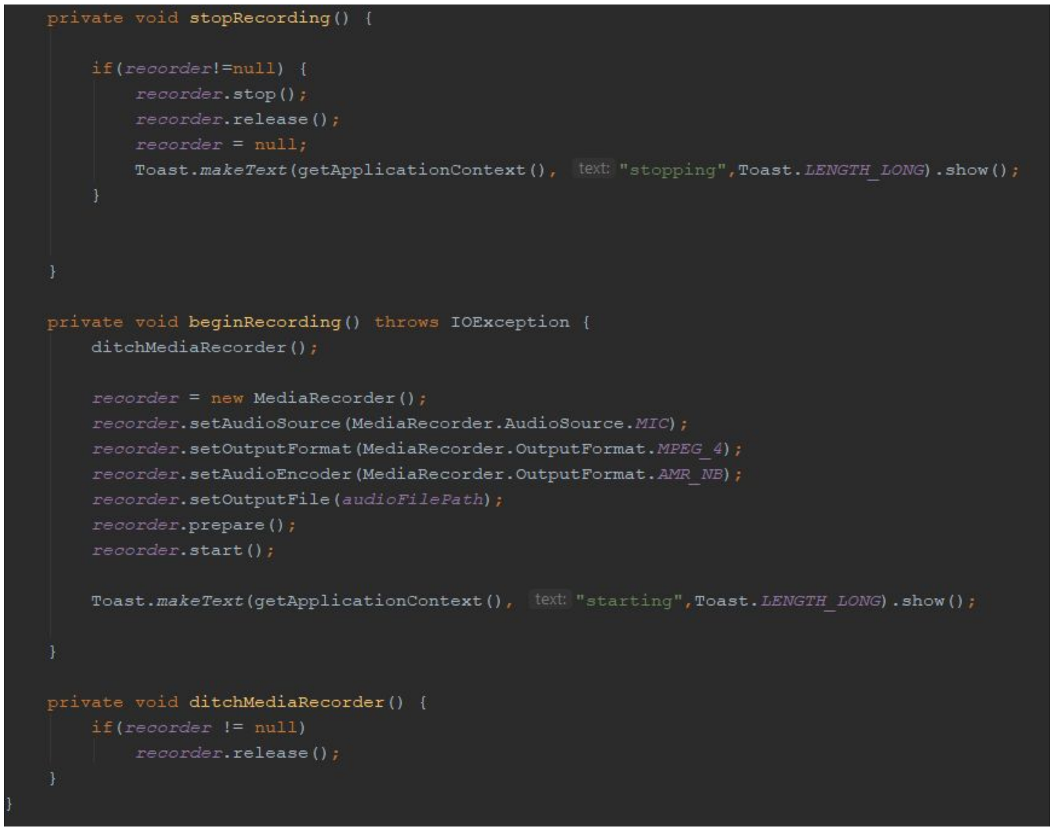
3.3. Integrating SpeechAce API to Android Application

3.3.1. Send request to speechace by sending a file and returning a string json response using OKHttp.

```
private String sendRequestToSpeechAce(File file){
    RequestBody requestFile = RequestBody.create(MediaType.parse("audio/mpeg3"), file);
    //RequestBody requestFile = RequestBody.create(MediaType.parse("audio/x-wav"), file);
    MultipartBody.Part fileBody = MultipartBody.Part.createFormData("user_audio_file", "apple", requestFile);
    MultipartBody body = new MultipartBody.Builder().setType(MultipartBody.FORM)
            .addPart(fileBody)
            .addFormDataPart("text", "apple")
            .addFormDataPart("user_id", "1234")
            .build();

    Request request = new Request.Builder()
                                 .url(SPEECH_ACE_URL)
                                 .post(body)
                                 .build();
    try {
        OkHttpClient client = new OkHttpClient();
        Response response = client.newCall(request).execute();

        return response.body().string();
    }
    catch (IOException e) {

        return "Error";
    }
}
```

3.4. Saving Audio Recordings

3.4.1.



```
    private void stopRecording() {

        if(recorder!=null) {
            recorder.stop();
            recorder.release();
            recorder = null;
            Toast.makeText(getApplicationContext(),  text: "stopping",Toast.LENGTH_LONG).show();
        }


    }

    private void beginRecording() throws IOException {
        ditchMediaRecorder();

        recorder = new MediaRecorder();
        recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
        recorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
        recorder.setAudioEncoder(MediaRecorder.OutputFormat.AMR_NB);
        recorder.setOutputFile(audioFilePath);
        recorder.prepare();
        recorder.start();

        Toast.makeText(getApplicationContext(),  text: "starting",Toast.LENGTH_LONG).show();

    }

    private void ditchMediaRecorder() {
        if(recorder != null)
            recorder.release();
    }
}
```

3.5. Reward Action Behaviors

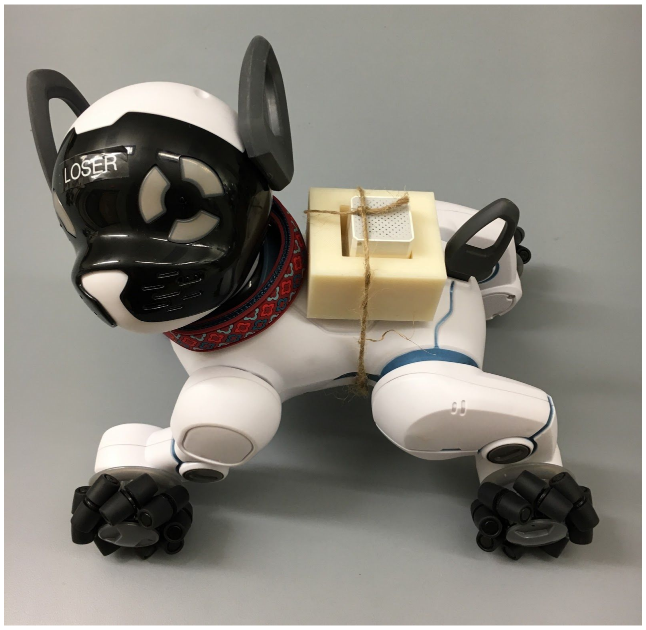
3.5.1. The code for the reward action behaviors will consist of a series of if-else statements that follow specific standards as shown below. This has not yet been successfully integrated into our application due to some problems with scaling the similarity score from Speechace.

3.5.1.1. Level 1: CHiP will dance for each vocalization, and will progress to Level 2 alongside rewarding the child after 5 vocalizations.

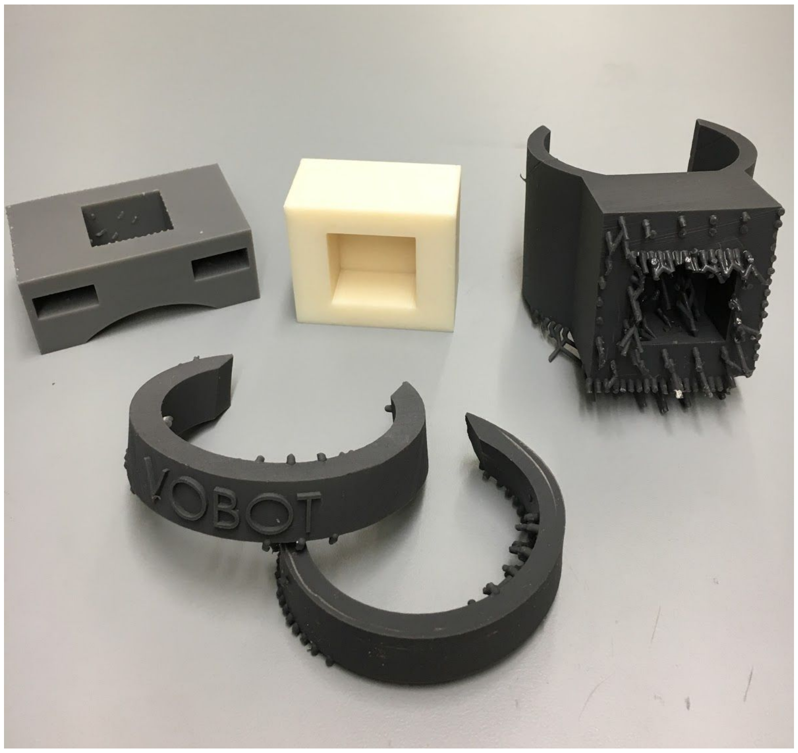
3.5.1.2. Level 2: CHiP will dance for more improved vocalizations (quality score > 10). If after 5 attempts, the child's average quality score is greater than 10, he/she will progress to the next level and is rewarded; however, if the quality score is less than 10, the child returns to Level 1.

3.5.1.3. As the levels increase in number, the threshold quality score to advance increases as well. If the quality score > 90 on an attempt, CHiP will reward with a dance.

3.6. 3D Printed Speaker Harness



3.6.1. Harness makeshift model at the time of demo



3.6.2. Additional 3D models printed and pending adjustments

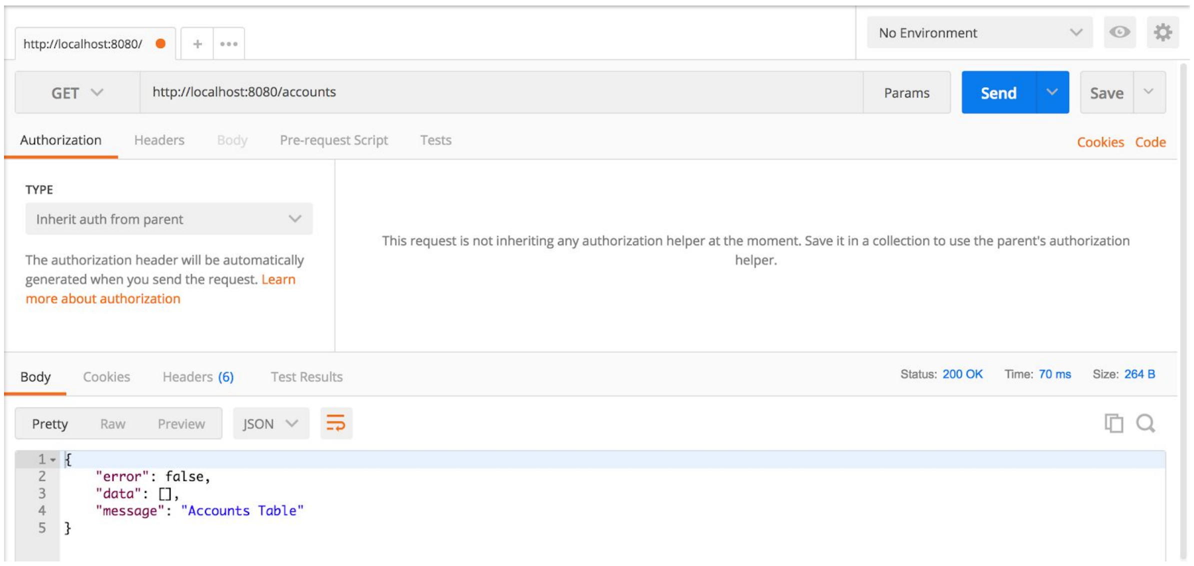
4. Measurements and Data

4.1. Migration to CHiP Robot

4.1.1. The CHiP model is fully functional and integrated with the Android application. The app can connect to any CHiP robot using bluetooth with the click of a button, control its actions, and run through a cycle of speech recognition with rewards according to specified progress levels.

4.2. Database

4.2.1. We are able to successfully push and pull information to the database using a RESTful API. Starting with empty tables in a MySQL database, the system is able to populate it with data and return appropriate errors if incorrect parameters are entered. The system is also able to retrieve data from the database and return it in a json format. The data we are collecting is the information that is pushed and pulled to and from the database. As shown below, a post command to the RESTful API (Figure 2) populates the initially empty database (Figure 1) with one row of information (Figure 3), corresponding to the data in the body of the post command.



http://localhost:8080/accounts

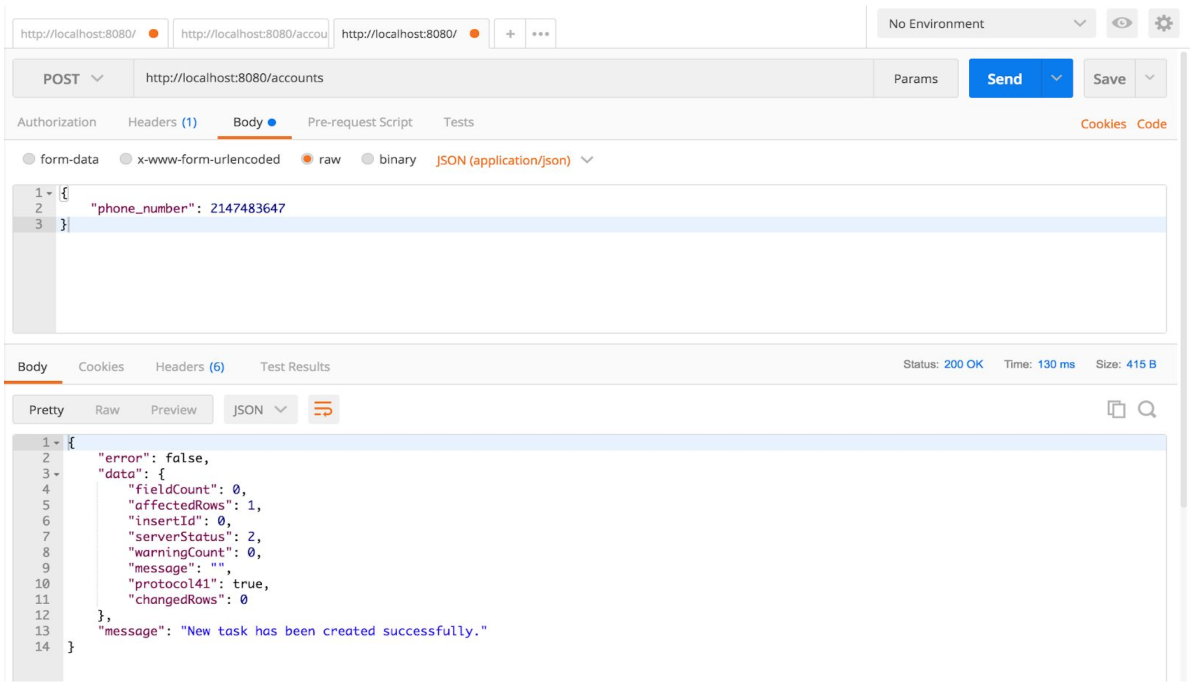
The authorization header will be automatically generated when you send the request. Learn more about authorization

This request is not inheriting any authorization helper at the moment. Save it in a collection to use the parent's authorization helper.

Status: 200 OK Time: 70 ms Size: 264 B

```
{
    "error": false,
    "data": [],
    "message": "Accounts Table"
}
```

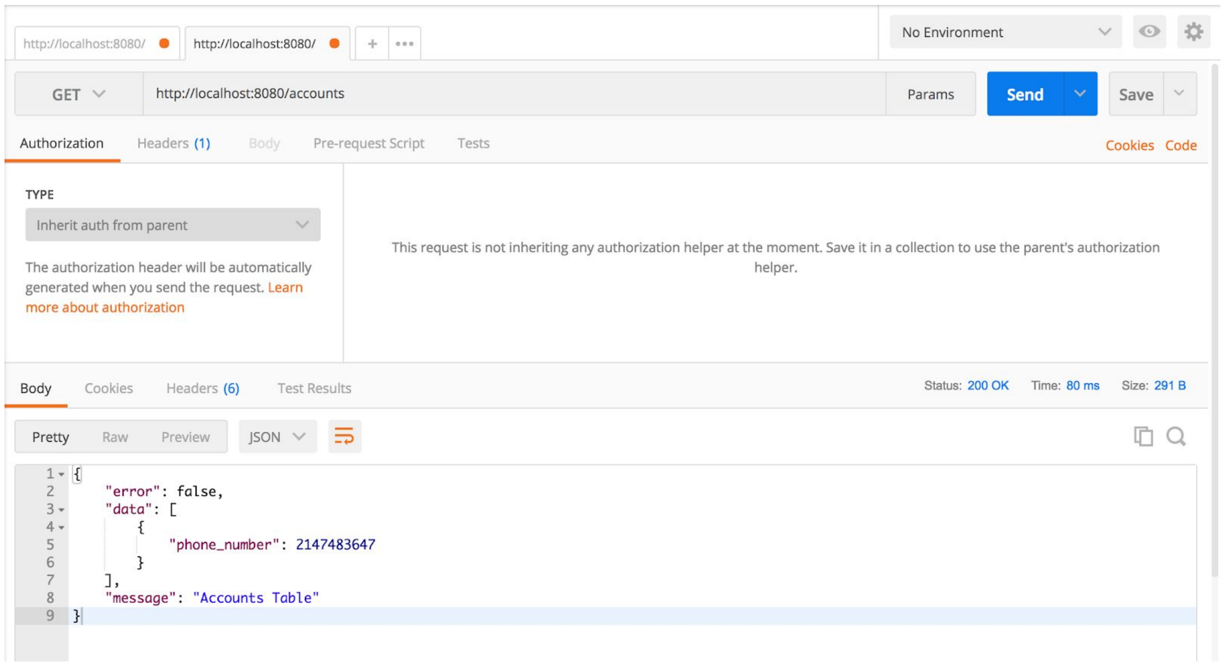
Figure One: Initially empty database shown using ‘GET’ command



```
{
    "phone_number": 2147483647
}
```

```
{
    "error": false,
    "data": {
        "fieldCount": 0,
        "affectedRows": 1,
        "insertId": 0,
        "serverStatus": 2,
        "warningCount": 0,
        "message": "",
        "protocol41": true,
        "changedRows": 0
    },
    "message": "New task has been created successfully."
}
```

Figure Two: Populating the database with one phone number shown using 'POST' command



The authorization header will be automatically generated when you send the request. Learn more about authorization

This request is not inheriting any authorization helper at the moment. Save it in a collection to use the parent's authorization helper.

```
{
    "error": false,
    "data": [
        {
            "phone_number": 2147483647
        }
    ],
    "message": "Accounts Table"
}
```

Figure Three: Now populated database with one entry, shown using 'GET' command

4.3. Integrating SpeechAce API to Android Application

4.3.1. Once the user finishes speaking, his/her audio is saved and formatted, and the API call which is made prints the similarity score on the appropriate screen in the application. The integration of SpeechAce into the Android application measures the similarity score between the user's audio recording to the model word. The comparison will allow us not only to give feedback on speech progress, but also serve as the foundation for the module which rewards users based on their progress.

4.4. Saving Audio Recordings

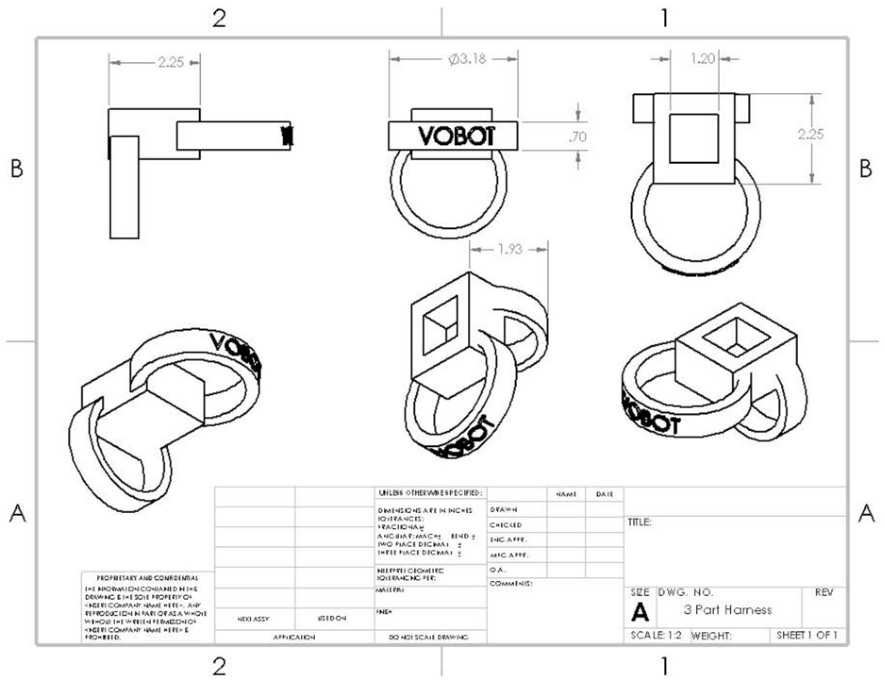
4.4.1. We are able to successfully record and save audio recordings on the phone's SD card. If this is the first session, a new file is created called "audiorecorded.mp4". When a new session begins, the previous recording is overwritten so that the app doesn't take up too much storage on the phone.

4.5. Reward Action Behaviors

4.5.1. CHiP is currently able to give positive reward feedback by performing songs. Initially upon first attempt vocalizations, CHiP will always return a reward for any utterance made, and then upon the remaining four attempts, CHiP only returns a reward behavior if 90% accuracy is met. The data used to determine the behavior is the similarity score, and the measurement made is the reward displayed after the similarity score has been achieved.

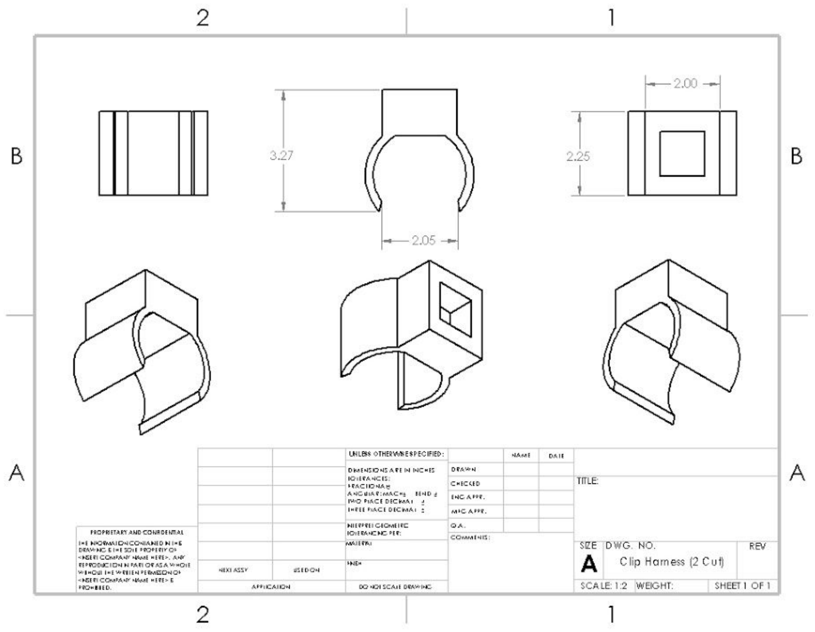
4.6. **3D Printed Speaker Harness**

4.6.1. Assembled drawing of three parts with measurements



4.6.2.

Clip harness drawing with measurements



5. Conclusions

5.1. Migration to CHiP Robot

5.1.1. We have successfully been able to migrate to the CHiP Robot. We can control CHiP's actions through our application.

5.2. Database

5.2.1. At its current stage, the RESTful API is able to push and pull information to/from the all tables in the database. Trying to pull conditional queries from the database throw errors, which is most likely a syntax problem that needs to be researched and resolved. The RESTful API and database have been used successfully in testing, but in order to integrate both of these into the application, the stated issue must be resolved. Moving forward, we will attempt to integrate the database and RESTful API with the Android application.

5.3. Integrating SpeechAce API to Android Application

5.3.1. Integrating the SpeechAce API with our application has been a success, yet some concerns like latency and accuracy linger.

5.3.2. We have our work cut out for us in the quality scoring cleanup as we combat noise and hardware limitations like microphone resolution.

5.4. Saving Audio Recordings

5.4.1. Our current system of storing audio files locally on the application suffices for testing purposes but is not viable long-term.

5.4.2. In order to make our application leaner, we intend to make use of another service the SpeechAce API offers - generating reference audio for a given text or word. This service resolves the memory-intensive system in place by replacing recorded and saved audio files into programmatically streamed audio files.

5.5. Reward Action Behaviors

5.5.1. The rewards as well as the levelling system are not complete, however they show promising potential.

5.5.2. Moving forward we will consult therapists in order to get their expert opinion regarding the most effective reward actions to offer as well as fleshing out the levelling system entirely as we tailor individual levels to have a meaningful relationship with the quality score threshold at each level.

5.6. 3D Printed Speaker Harness

5.6.1. The Bluetooth speaker model at time of demo was too big and the speaker would move around within the opening where it sits. Seeing as that was the first prototype model, we were able to understand how dimensions worked for both CHiP and the speaker and can make adjustments moving forward.

5.6.2. The additional models printed after demo date also had some shortcomings due to sizing. The clip harness was large around the back of the robot and would not let the speaker sit flush with the curved back. Additional measurements and prototypes will be printed to accommodate for these size discrepancies.