Sinan Şar
ID:150180734

BLG 317E – DATABASE SYSTEMS TERM PROJECT Implementation Report

A. Motivation and Requirements

Motivation Music is one of the biggest industries in our world. It is everywhere: in advertisements, there are musicians on the streets, most people are using streaming services. It is in everybody's lives; we are surrounded by music. One of the biggest issues in music industry, naturally, is finding a sound that you like for your creation. Everyone who listens to music at some point says "Wow, this song sounds great.", but what really goes into this statement? What makes a song great? In my mind there are two sides to this question: The composition and the sound. We are going to focus on the latter for this project. Musicians and producers are always trying to either recreate a sound that they have heard and were amused by, or they are trying to find a way to produce a specific sound/tone in their head. There are many elements that go into the mix for both of these processes: The choice of instrument, the amplification you use, settings for your amp, type of cabinet, choice of microphone, and the list goes on and on. Though, it sounds like a simple process when you think about it the first time, it's quite possible and in most cases unavoidable to run into issues. As a musician and someone who is interested in music production myself, I can relate to this struggle and would like to propose and offer to this problem. "FindYourTone" is not a database for a music library or an application to store and listen to music. It is designed as a solution for a problem that musicians face on every instance of recording or playing on a day to day basis. Users will be able to add and share detailed information of sounds they liked like the instruments used, amplification used, settings etc. They will also be able to upload small sound samples to display the sounds they have created. Achieving a musical sound that you like can be easier than ever with a database with some certain features. Let's go over those features and see which problems they help solve.

1. Provides a structured and moderated source One of the biggest issues regarding recreating a particular sound is lack of information. Sometimes you cannot find any information at all, sometimes you can find some buried deep within the reigns of internet. But can you really trust that? Let's say you are a fan of Led Zeppelin and want to create the guitar sound from Stairway to Heaven for your own musical work. And the only thing you can find is a forum entry from a random user from 2003 explaining gear that has been used for an album made in 1971. The information you find being flawless in these circumstances is not very high at all. Having a system closely monitored and well structured would help with this issue. Admins can find and delete defective information from the website. This way information can be a lot clearer and can be found in a single neatly designed platform, providing ease of use. Information being trustworthy or not, directly correlates with the second point as well.
2. Incorporates verified musicians to find the correct answers If we are talking about Smells Like Teen Spirit from Nirvana, who can provide better information than the band members themselves? Which guitar did Kurt Cobain play? Which pedals did they end up using? What brand was the drum kit? FindYourTone will provide a platform where amateur musicians get a chance to interact with and learn from their favorite artists. Admins will be able to add verified musicians as members to the website. Therefore, users will be able to view featured artist entries as well as entries from other users. When an artist makes an entry about their own material they will be distinguished from others, since this information will be certainly authentic. This will help us separate real answers from suggestions. Of course, entries from regular users are not disregarded when a verified artist provides an entry for a certain sound. Other users may have achieved similar result with a different set of equipment which may still

be valuable information. For example, a verified musician can spend enormous amounts of money on gear, but most amateur musicians don’t have such funding. Someone coming up with a different, more affordable road to produce a similar sound is hence still very beneficial.

3. Easier to find better sounds Users will be able to grade the sound suggestions they have tried and the average grade for each entry will be shared with all users. This way you can get an decide if an entry is good enough for you to try or should you try and find another one. This will provide a platform for the community to communicate with each other as well, which is always a plus.
4. Reduces cost & time spent on sourcing Once find what kind of gear you need, you have to purchase the gear if you don’t already own it and in most cases you won’t have all necessary parts on hand. Good gear in music is almost never cheap, professional grade equipment will cost in thousands of dollars range (up to hundreds of thousands for vintage and rare items). If you are starting with bad information and heading into the wrong direction, you may have to spend a lot more time, money and effort to reach result you want. This is an unnecessary waste and can be spared. This system will provide user with an environment, thanks to its set of features, where they can hit the result they want spot on.

B. Conceptual/Logical Database Design

I have found it sensible to merge the conceptual design and logical design parts for my project. Since I have delivered the ER design assignment with the tables already prepared. Therefore, these two sections will be addressed under one single section.

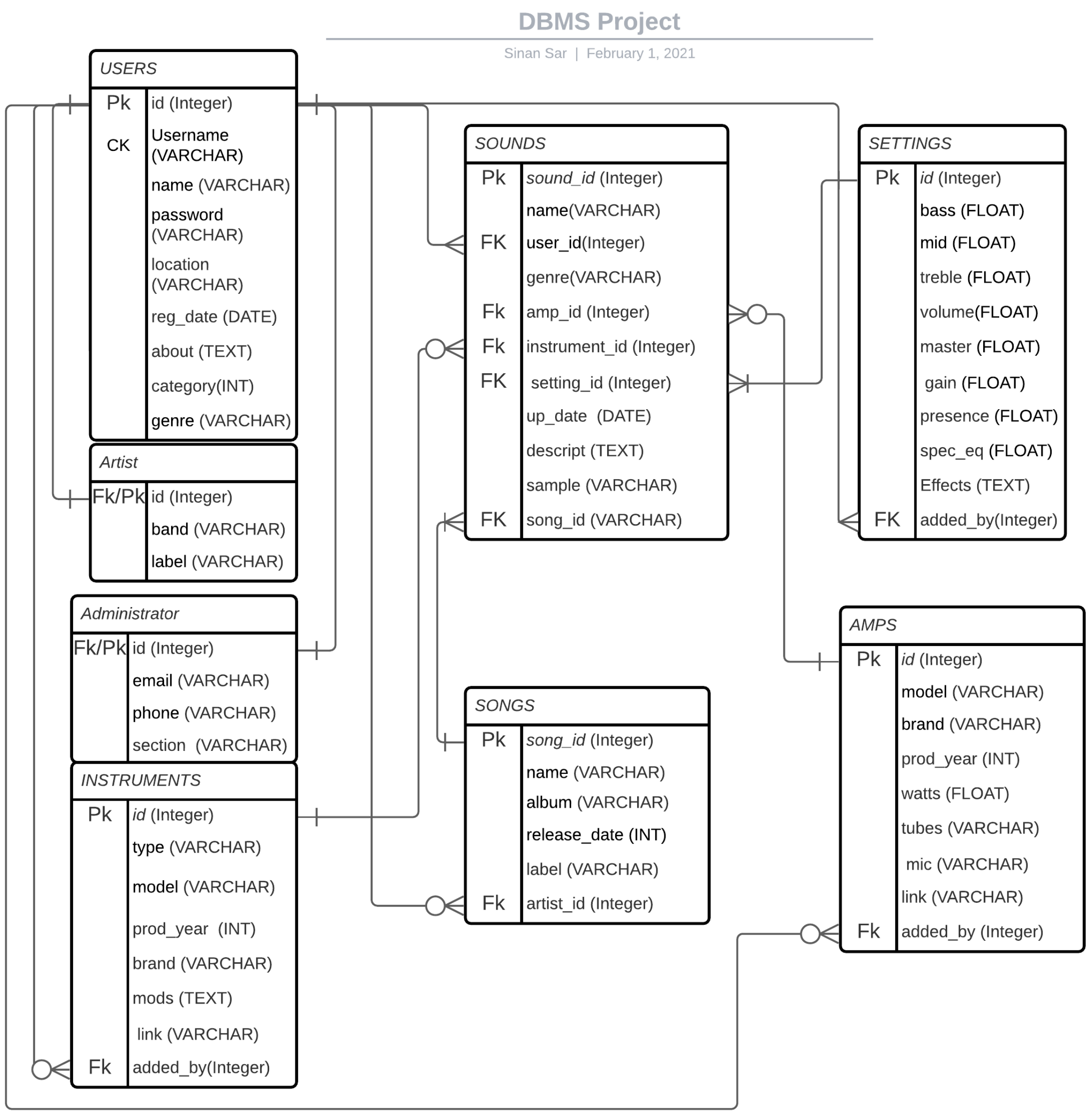
Data Requirements

- A sound is created by one user.
- A user can create zero or many sounds.
- A sound incorporates one instrument
- An instrument can be incorporated in zero or many sounds.
- An instrument is added by one user.
- A user can add many instruments.
- A user can add many amps.
- An amp is added by one user.
- A sound uses one amp. • An amp can be used in zero or many sounds.
- A song is created by one artist. (Artist is specified user, default users can’t create songs.)
- An artist can create many songs.
- A sound uses one setting.
- A setting can be used by many sounds.
- A song uses one or many sounds.
- A sound can be used in one song.
- An artist is a user.
- An admin is a user

As a revision to these requirements, multiple user types weren’t implemented. There is a single user type available on the system, which is the same as the “User” type from the proposal. For the purposes of this project single user type was sufficient in my opinion, hence the revision. The user actions are still the same as the proposed ones.

Sinan Şar
ID:150180734

ER/Logical Design Schema



This is the latest version of the tables used in the project. As mentioned above artist, administrator and songs tables haven't been implemented. But their CRUD statements have been written therefore they are still retained on the logical design diagram. There also have been slight modifications on the tables on the attributes since the initial proposition of the logical design. Those can also be noticed when the figure above is compared with the older version.

Sinan Şar
ID:150180734

DDL Statements Used on the Project

DDL statements used to create the tables can be seen below.

```
CREATE TABLE IF NOT EXISTS USERS(
    Id serial,
    Name varchar NOT NULL,
    Username varchar NOT NULL UNIQUE,
    Password varchar NOT NULL,
    Location varchar,
    About text,
    genre varchar,
    Category int DEFAULT 0,
    Reg_date date DEFAULT CURRENT_DATE,
    PRIMARY KEY(Id));


    CREATE TABLE IF NOT EXISTS ARTIST(
    Id integer,
    band varchar,
    label varchar,
    FOREIGN KEY (Id) REFERENCES Users(Id) ON DELETE CASCADE);



    CREATE TABLE IF NOT EXISTS ADMINISTRATOR(
    Id integer,
    email varchar ,
    phone varchar,
    section varchar,
    FOREIGN KEY (Id) REFERENCES USERS(Id) ON DELETE CASCADE);


    CREATE TABLE IF NOT EXISTS INSTRUMENTS(
    Id serial,
    type varchar ,
    model varchar,
    prod_year int,
    mods text,
    link varchar,
    added_by integer,
    PRIMARY KEY(Id),
    FOREIGN KEY (added_by) REFERENCES USERS(Id) ON DELETE CASCADE);
```

Sinan Şar
ID:150180734

```
    CREATE TABLE IF NOT EXISTS AMPS(
    Id serial,
    model varchar ,
    brand varchar,
    prod_year int,
    watts float,
    tubes varchar,
    mic varchar,
    link varchar,
    added_by int,
    PRIMARY KEY(Id),
    FOREIGN KEY (added_by) REFERENCES USERS(Id) ON DELETE CASCADE);



    CREATE TABLE IF NOT EXISTS SONGS(
    song_Id serial,
    name varchar ,
    album varchar,
    release_date int,
    label varchar,
    artist_id integer,
    PRIMARY KEY(song_Id),
    FOREIGN KEY (artist_id) REFERENCES USERS(Id) ON DELETE CASCADE);

    CREATE TABLE IF NOT EXISTS SETTINGS(
    Id serial,
    bass float ,
    mid float ,
    treble float,
    volume float ,
    master float ,
    gain float ,
    presence float ,
    spec_eq float ,
    effects text,
    genre   varchar,
    added_by integer,
    PRIMARY KEY(Id),
    FOREIGN KEY (added_by) REFERENCES USERS(Id) ON DELETE CASCADE);


    CREATE TABLE IF NOT EXISTS SOUNDS(
    sound_Id serial,
    name varchar ,
    user_id integer,
    genre varchar,
    amp_id integer,
    instrument_id integer,
    setting_id integer,
    descript text,
    sample varchar,
    up_date date DEFAULT CURRENT_DATE,
    PRIMARY KEY(sound_Id),
    FOREIGN KEY (user_id) REFERENCES USERS(Id) ON DELETE CASCADE,
    FOREIGN KEY (instrument_id) REFERENCES INSTRUMENTS(Id) ON DELETE
CASCADE,
    FOREIGN KEY (amp_id) REFERENCES AMPS(Id) ON DELETE CASCADE,
    FOREIGN KEY (setting_id) REFERENCES SETTINGS(Id) ON DELETE CASCADE)
```

Sinan Şar
ID:150180734

C. Database Normalization

This part has been revised since the last time it was reported, since there have been some changes in the attributes for the tables. Changes related to this section are in Users and Administrator tables.

1. Functional Dependencies

A list of the functional dependencies can be seen below.

- Users table: All non-key attributes are dependent on "id" (primary key). Username is also candidate-key.
- Artist table: All non-key attributes are dependent on "id" (primary key).

- Administrator table: All non-key attributes are dependent on "id" (primary key).
- Instruments table: All non-key attributes are dependent on "id" (primary key).
- Sounds table: All non-key attributes are dependent on "sound_id" (primary key).
- Songs table: All non-key attributes are dependent on "song_id" (primary key).
- Settings table: All non-key attributes are dependent on "id" (primary key).
- Amps table: All non-key attributes are dependent on "id" (primary key).

2. Normalization

For this part, I have explained the requirements on each normal form and expressed why the tables on my design fit a normal form or not. Since there is only a single table having a different situation than the others, to save the readers time I explained the situation for all the tables at once. There wouldn't be much of a point in explaining the same thing over and over again for the 7 tables, since their situation is exactly the same. And then I explained the Users table exclusively, since it has 2 candidate keys while all other tables only have one and that needs to be discussed.

- For the first normal form, attribute values have to be atomic. This statement holds for all of the tables in my design, as it can be easily seen on the logical design diagram.
- For the second normal form, table must be in first normal form and it must not have any non-prime attribute that is functionally dependent on any proper subset of any candidate key of the relation.

Considering the functional dependencies shown on the first part of the report, all non-prime attributes on each table only depend on the primary keys. Hence, all non-prime attributes on each table are dependent on the whole of their respective candidate keys. Therefore, all tables are in second normal form. (If we consider the 2 keys present in Users table, the table still will be in 2nd normal form since non-prime attributes will still be dependent on the whole of each candidate key.)

- For the third normal form: The table has to be in second normal form, every non-prime attribute of R must be non-transitively dependent on every key of R.

All the tables in the design are already in 2nd normal form. On all tables, every non-prime attribute only depends on the primary key of the respective table. Hence all tables are in the third normal form. (If we consider the 2 keys in Users table, the table still will be in 3rd normal form since non-prime attributes will only be dependent on the keys for the table.)

For Boyce-Codd normal form, for every one of the tables dependencies $X \rightarrow Y$, at least one of the following conditions must hold:

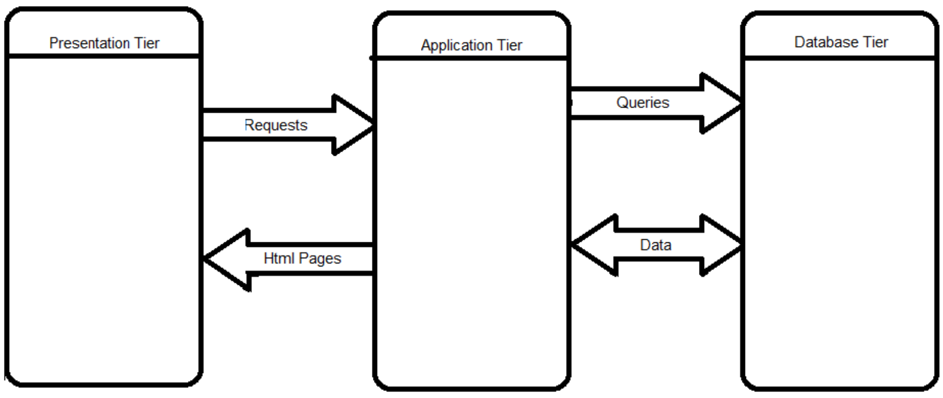
1. $X \rightarrow Y$ is a trivial functional dependency ($Y \subseteq X$),
2. X is a superkey for schema R.

Considering the functional dependencies on part 1, for every functional dependency left hand side will be a candidate key (which is a super key) on each table. Hence, we can say that all tables are in Boyce-Codd normal form. (If we consider the 2 keys in Users table, the table still will be in Boyce-Codd normal form since all attributes will still be dependent on keys for the table. Which satisfies Boyce-Codd normal form.)

D. Application Design and Implementation

1. Technical Manual

- We can think this application as a three-tier application. We have the presentation tier, application tier and the database tier. These parts will be called “Tier” if they are deployed and run on different servers and “Layer” if they are run on the same device. In my presentation for example, they were all run on the local device which would make them layers. But in a real-world application of such a system they would be tiers. Now these three tiers will be discussed in more detail.
  Firstly, the presentation tier can be shortly described as the user interface and the communication tier. This is the tier that users see and interact with. Its main purpose would be collecting and presenting data. Data can be collected via forums etc. from the user and then actions can be performed correspondingly on other tiers of the app. For this project this tier runs on a web browser and has been developed using html, CSS and JavaScript.
  Secondly, the application tier (the logic tier) is where data collected from presentation tier is processed. This tier can also manipulate the database tier and fetch/update/delete/add data. All communication between tiers go from this tier. Database and presentation tiers can’t communicate without going through application tier. In this project this tier is built using python.
  Lastly, the database tier is where data required for the app is stored and managed. In this project a PostgreSQL database is used for this purpose. (Different relational database management systems or NoSQL databases can also be used.)



- SQL queries used in the application and their meaning in plain English are listed below.
  i. *Query*:
  ```
  INSERT INTO users
  (name,
  username,
  password,
  location,
  about,
  genre)
  VALUES      ( %s, %s, %s, %s, %s, %s) ;
  ```
  Semantics: Inserts a new user into the users table with attributes provided

  ii. *Query*:
  ```
  SELECT *
  FROM users
  WHERE (username = %s);
  ```
  Semantics: Select a user from the database with a certain username

  iii. *Query:*
  ```
  UPDATE users SET
  name =%s ,
  username=%s,
  PASSWORD=%s,
  location=%s,
  about=%s,
  genre=%s
  WHERE (id = %s);
  ```
  Semantics: Update attributes of a user with a certain id

  iv. *Query:*
  ```
  DELETE FROM users
  WHERE (username = %s);
  ```
  Semantics: Delete a user with a certain username

  v. *Query:*
  ```
  INSERT INTO amps(
  model,
  brand,
  prod_year,
  watts,
  tubes,
  mic,
  link,
  added_by)
  VALUES ( %s, %s, %s, %s, %s ,%s, %s ,%s);
  ```
  Semantics: Insert a new amp into the amps table

vi. *Query:*
```
UPDATE amps SET
model =%s ,
brand=%s,
prod_year=%s,
watts=%s,
tubes=%s,
mic=%s,
link=%s,
added_by=%s
WHERE (id = %s);
```

Semantics: Update attributes of an amp with a certain id.

vii. *Query:*
```
SELECT *
FROM amps
WHERE (id = %s);
```

Semantics: Select an amp from the database with a certain id

viii. *Query:*
```
SELECT *
FROM amps
WHERE (added_by = %s);
```

Semantics: Select all amps from the database added by a certain user

ix. *Query:*
```
DELETE FROM amps
WHERE (id = %s);
```

Semantics: Delete an amp with a certain id from the database.

x. *Query:*
```
INSERT INTO instruments
(type, model,
prod_year,
 mods,
link,
added_by)
 values ( %s, %s, %s, %s, %s ,%s);
```

Semantics: Insert a new instrument into the instruments table.

xi. *Query:*
```
UPDATE instruments SET
type=%s ,
model=%s,
 prod_year=%s,
mods=%s,
 link=%s,
added_by=%s  where (id = %s);
```

Semantics: Update attributes of an instrument with a certain id.

xii. *Query:*
```
SELECT *
 FROM instruments
 WHERE (id = %s);
```

Semantics: Select an instrument from the database with a certain id.

Sinan Şar
ID:150180734

xiii. *Query*:
```
SELECT *
FROM instruments
WHERE (added_by = %s);
```
Semantics: Select all instrument from the database added by a certain user.

xiv. *Query:*
```
DELETE FROM instruments
WHERE (id = %s);
```
Semantics: Delete an instrument with a certain id from the database.

xv. *Query:*
```
INSERT INTO SETTINGS(bass, mid, treble, volume,
master,gain,presence,spec_eq,effects,genre,added_by)
values ( %s, %s, %s, %s, %s ,%s, %s ,%s,%s, %s ,%s);
```
Semantics: Insert a new setting into the settings table.

xvi. *Query:*
```
UPDATE SETTINGS SET
bass=%s,
mid=%s,
treble=%s,
volume=%s,
master=%s,
gain=%s,
presence=%s,
spec_eq=%s,
effects=%s,
genre=%s,
added_by=%s
WHERE (id=%s)
```
Semantics: Update attributes of a setting with a certain id.

xvii. *Query:*
```
DELETE FROM settings
WHERE (id = %s);
```
Semantics: Delete a setting with a certain id from the database.

xviii. *Query:*
```
SELECT *
FROM settings
WHERE (id = %s);
```
Semantics: Select a setting from the database with a certain id.

xix. *Query*:
```
SELECT *
FROM settings
WHERE (added_by = %s);
```
Semantics: Select all settings from the database added by a certain user.

xx. *Query*:
```
INSERT INTO sounds(
name,
user_id,
genre,
amp_id,
instrument_id,
setting_id,
descript,
sample)
values (%s, %s, %s, %s, %s ,%s,%s, %s );
```
Semantics: Insert a new sound into the settings table.

xxi. **Query:**
```
UPDATE SOUNDS SET
 name=%s,
 user_id=%s,
 genre=%s,
 amp_id=%s,
 instrument_id=%s,
 setting_id=%s,
 descript=%s,
 sample=%s,
 up_date=%s
 WHERE sound_Id=%s;
```
Semantics: Update attributes of a sound with a certain id.

xxii. *Query:*
```
SELECT *
FROM sounds
WHERE (id = %s);
```
Semantics: Select a sound from the database with a certain id.

xxiii. *Query*:
```
SELECT *
FROM sounds
WHERE (added_by = %s);
```
Semantics: Select all sounds from the database added by a certain user.

xxiv. *Query:*
```
SELECT *
FROM sounds
WHERE (name LIKE %s);
```
Semantics: Select all sounds from the database containing a certain string in their name.

xxv. *Query:*
```
SELECT *
FROM sounds;
```
Semantics: Select all sounds from the database.

xxvi. *Query:*
```
DELETE FROM sounds
WHERE (id = %s);
```
Semantics: Delete a sound with a certain id from the database.

xxvii. *Query* **`INSERT INTO SONGS( id,name,album,release_date,label,artist_id);`**
Semantics: Insert a new song into the songs table.

xxviii. *Query:* **UPDATE songs SET**
id=%s ,
name=%s,
album=%s,
`release_date` =%s,
`label` =%s,
`artist_id` =%s
where (id = %s);

Semantics: Update attributes of a song with a certain id.

xxix. *Query:*
```
SELECT *
 FROM songs
 WHERE (song_Id = song_id);
```

Semantics: Select a song from the database with a certain id.

xxx. *Query*:
```
SELECT *
FROM songs
WHERE (artist_id = user_id);
```

Semantics: Select all songs from the database added by a certain artist.

xxxi. *Query:*
```
DELETE FROM songs
WHERE (song_Id = song_id);
```

Semantics: Delete a song with a certain id from the database.

xxxii. *Query* **`INSERT INTO Artists(id,band, label);`**
Semantics: Insert a new artist into the artists table.

xxxiii. *Query:* **UPDATE Artists SET**
band=%s,
label=%s,
where (id = %s);

Semantics: Update attributes of an artist with a certain id.

xxxiv. *Query*:
```
SELECT *
FROM Artists
WHERE (id = %s);
```

Semantics: Select artist from the database with a certain id.

xxxv. *Query:*
```
DELETE FROM Artists
WHERE (id = %s);
```

Semantics: Delete an artist with a certain id from the database.

xxxvi. *Query* **`INSERT INTO Administrator (id,email. phone,section);`**

Semantics: Insert a new admin into the administrator table.

xxxvii. *Query:*

```
UPDATE Administrator SET
email=%s,
 phone=%s,
section=%s,
where (id = %s);
```

Semantics: Update attributes of an admin with a certain id.

xxxviii. *Query*:

```
SELECT *
FROM Administrator
WHERE (id = %s);
```

Semantics: Select admin from the database with a certain id.

xxxix. *Query:*

```
DELETE FROM Administrator
WHERE (id = %s);
```

Semantics: Delete an admin with a certain id from the database.

- In this project the application part has been written in Python3, front end has been implemented using Html, CSS (Bootstrap) and JavaScript. As DBMS, PostgreSQL has been used.

- Data has been added manually for the application for test purposes. A different data source hasn't been employed since it wasn't necessary for the purposes of this project.

- There is one error regarding the "edit sound" feature. As it was also pointed out in the demo session, even though the necessary query and backend infrastructure is set up there is an issue with editing sounds via a forum. There is an issue with the communication between the front-end and the backend I believe.

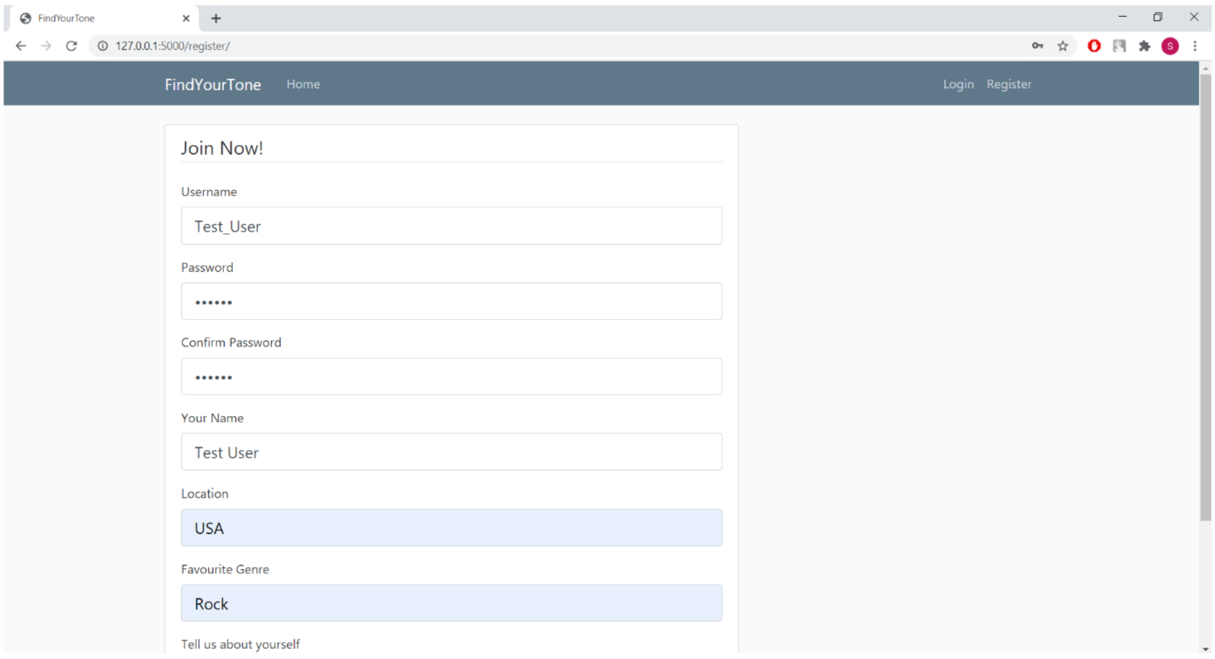
  Another difference in the implementation is the fact that user login is implemented with single privilege. In other words there are no admins, artists. The necessary tables and their SQL queries have been prepared (also added to the previous section) but the login method just hasn't been implemented.

Sinan Şar
ID:150180734

2. User Manual

In this section all the features of the application will be described. A screenshot corresponding to each feature can also be found in this part of the report.

- A new user can register to the application using the “Register” button on the landing page and filling a form with necessary user credentials.



Join Now!

Username
Test_User

Password
••••••

Confirm Password
••••••

Your Name
Test User

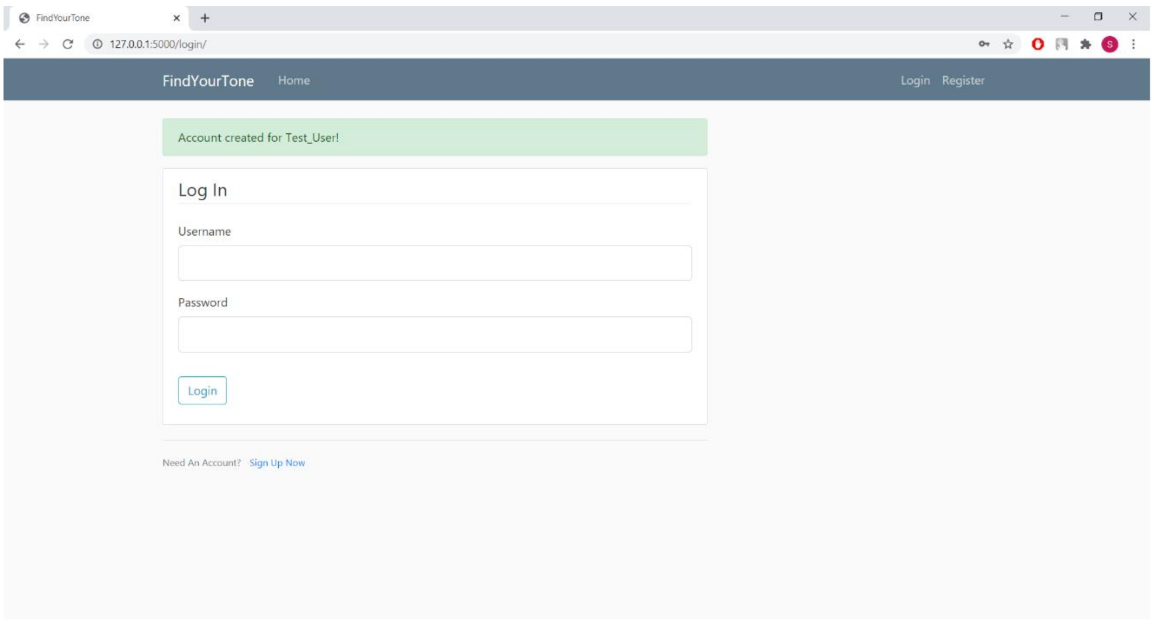
Location
USA

Favourite Genre
Rock

Tell us about yourself

Figure 1 Register Page

- A successful registration is indicated with a success message as user is redirected to login page. An unsuccessful registration attempt would create an error message and redirect user back to the registration page.



Account created for Test_User!

Log In

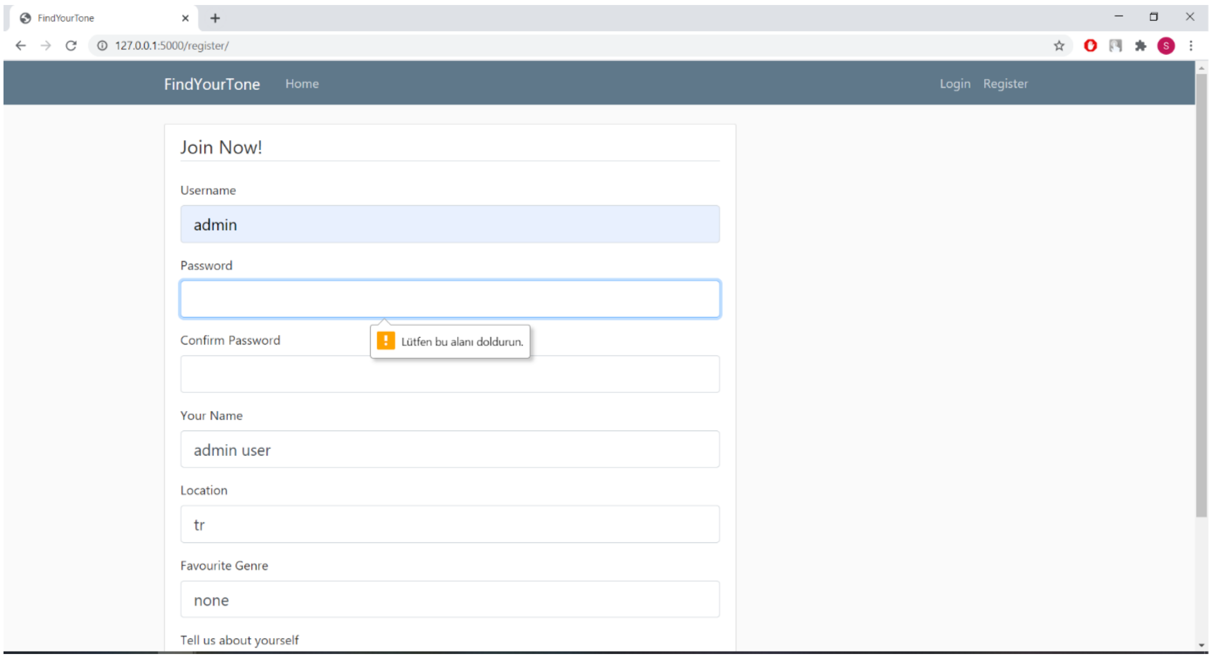
Username

Password

Login

Need An Account? Sign Up Now

Figure 2 Successful Registration



Join Now!

Username

admin

Password

Lütfen bu alanı doldurun.

Confirm Password

Your Name

admin user

Location

tr

Favourite Genre

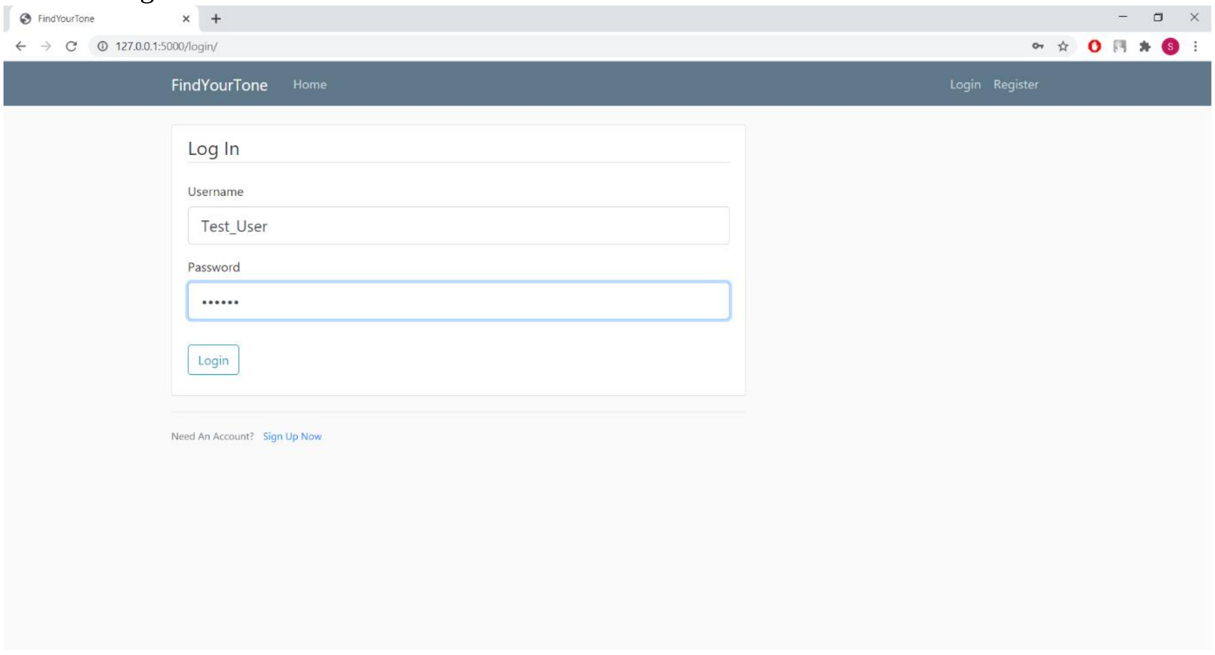
none

Tell us about yourself

Figure 3 Error message after registration attempt

These types of error messages are available for any user action on the website such as adding, deleting or editing amps, sounds etc. But for other actions these messages won't be shown in the screenshots since they are extremely similar to one other and there is not a valid reason to flood the report with them. Showing one should be enough.

- Users can login to the application after being registered. When login is successful they are redirected to the homepage, if authentication fails they are redirected to the login page with an error message.



Log In

Username

Test_User

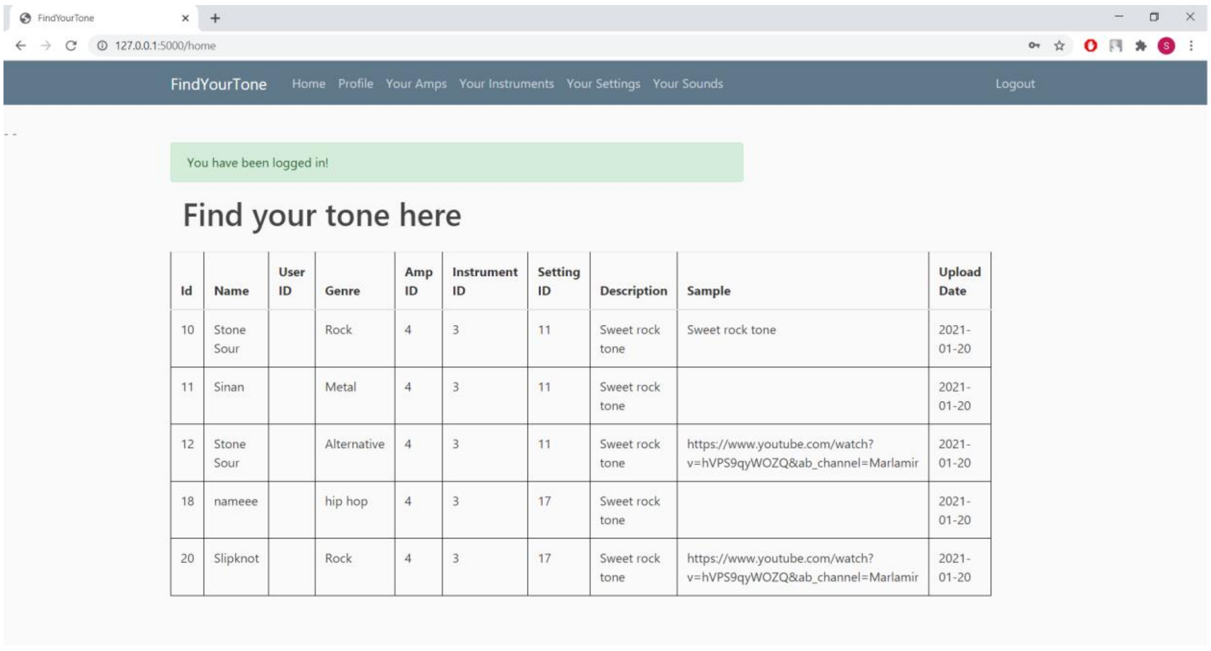
Password

••••••

Need An Account? Sign Up Now

Figure 4 Login page

- Users are redirected to homepage after login. It’s also noticeable that once user logs in, navbar changes and new options appear. Also, instead of register or login, logout option appears. Users can browse sounds from all users in the homepage.



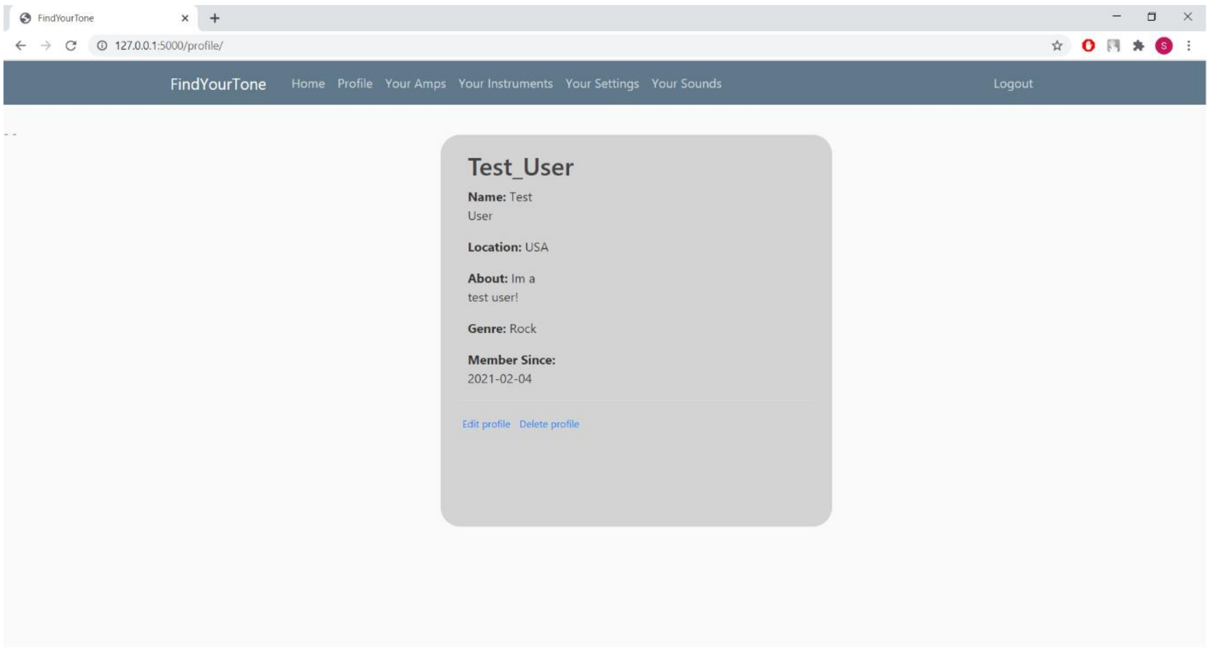
You have been logged in!

Find your tone here

Id	Name	User ID	Genre	Amp ID	Instrument ID	Setting ID	Description	Sample	Upload Date
10	Stone Sour		Rock	4	3	11	Sweet rock tone	Sweet rock tone	2021-01-20
11	Sinan		Metal	4	3	11	Sweet rock tone		2021-01-20
12	Stone Sour		Alternative	4	3	11	Sweet rock tone	https://www.youtube.com/watch?v=hVPS9qyWOZQ&ab_channel=Marlamir	2021-01-20
18	nameee		hip hop	4	3	17	Sweet rock tone		2021-01-20
20	Slipknot		Rock	4	3	17	Sweet rock tone	https://www.youtube.com/watch?v=hVPS9qyWOZQ&ab_channel=Marlamir	2021-01-20

Figure 5 Homepage after login

- Users can go to Profile tab to view their profile page. There they can edit or delete their profiles as they would like.



Test_User

Name: Test User

Location: USA

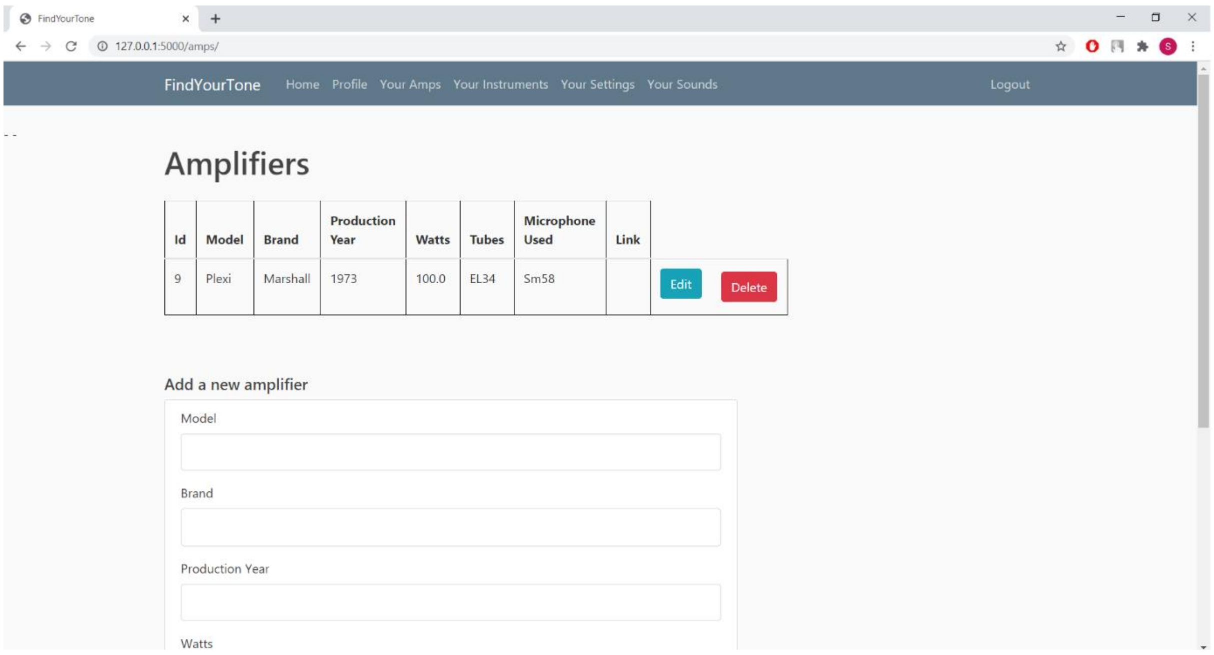
About: Im a test user!

Genre: Rock

Member Since: 2021-02-04

Figure 6 Profile tab for a user

- Users can go to Your Amps tab to add, browse, delete or edit amplifiers belonging to them.


FindYourTone
127.0.0.1:5000/amps/
FindYourTone Home Profile Your Amps Your Instruments Your Settings Your Sounds Logout

Amplifiers

Id	Model	Brand	Production Year	Watts	Tubes	Microphone Used	Link		
9	Plexi	Marshall	1973	100.0	EL34	Sm58		Edit	Delete

Add a new amplifier

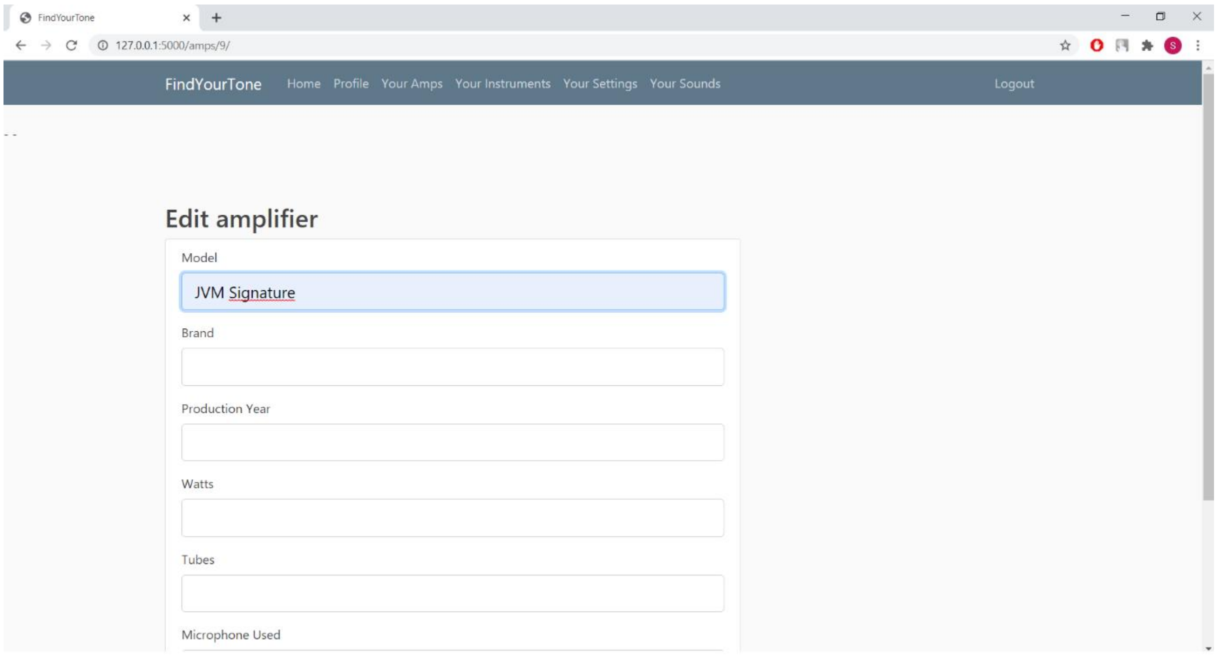
Model

Brand

Production Year

Watts

Figure 7 Amplifier page


FindYourTone
127.0.0.1:5000/amps/9/
FindYourTone Home Profile Your Amps Your Instruments Your Settings Your Sounds Logout

Edit amplifier

Model: JVM Signature

Brand

Production Year

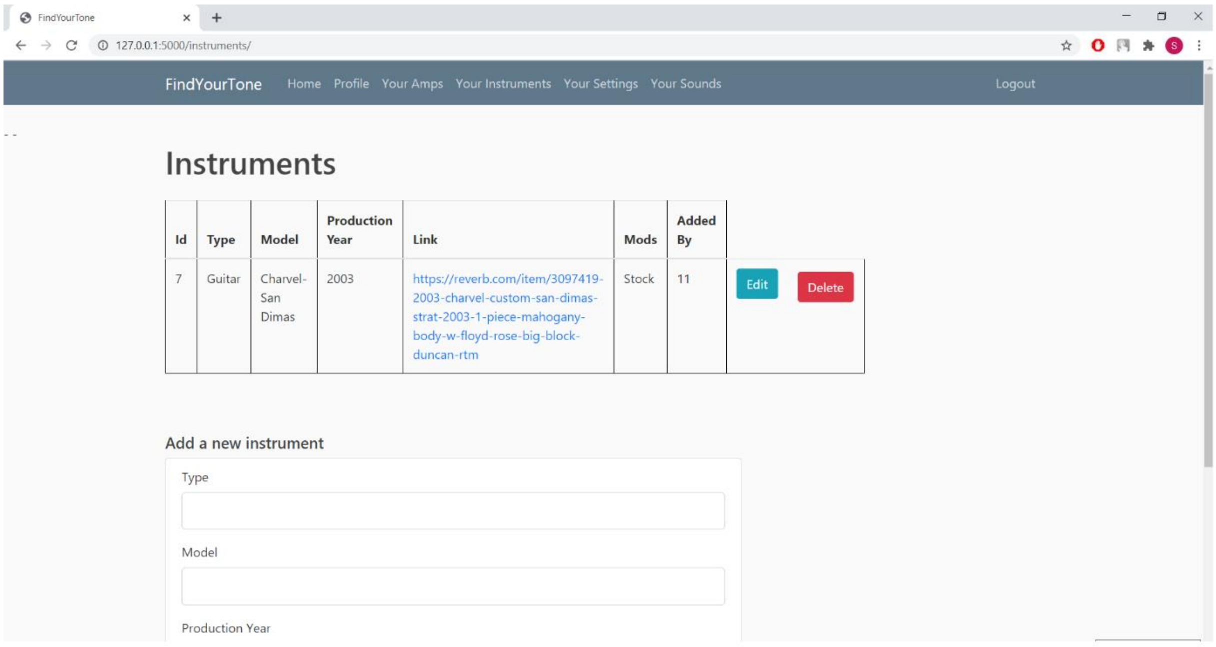
Watts

Tubes

Microphone Used

Figure 8 Edit Amplifier

- Users can go to Your Instruments tab to add, browse, delete or edit instruments belonging to them.



Instruments

Id	Type	Model	Production Year	Link	Mods	Added By		
7	Guitar	Charvel-San Dimas	2003	https://reverb.com/item/3097419-2003-charvel-custom-san-dimas-strat-2003-1-piece-mahogany-body-w-floyd-rose-big-block-duncan-rtm	Stock	11	Edit	Delete

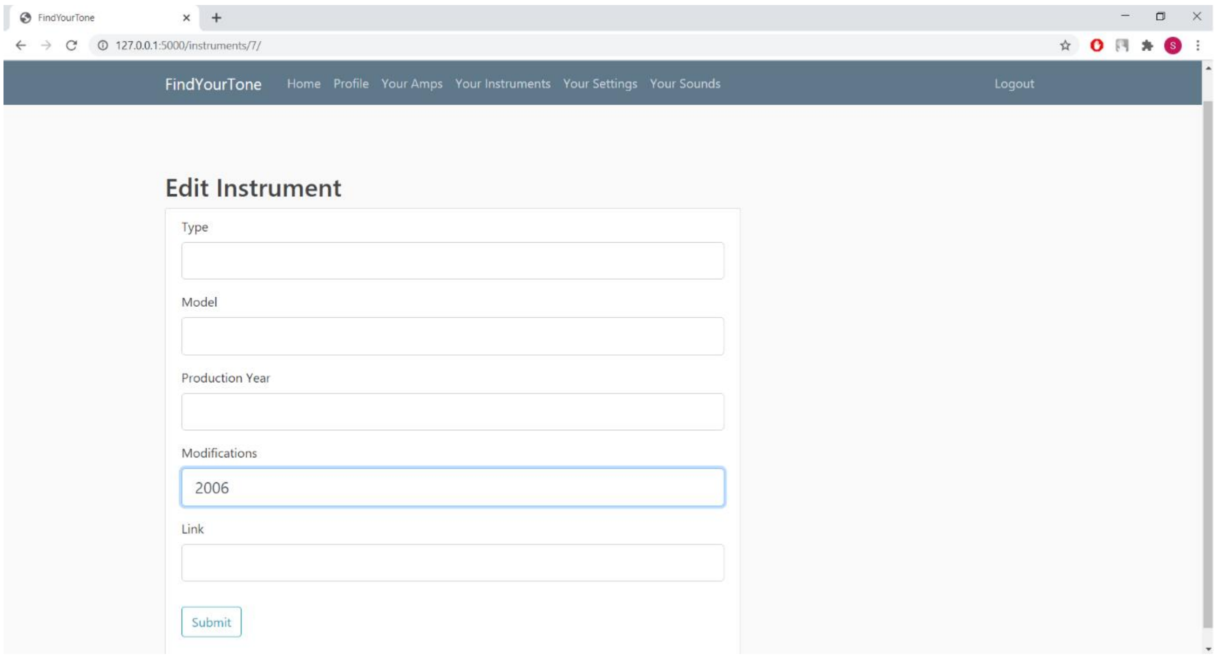
Add a new instrument

Type

Model

Production Year

Figure 9 Instrument Page



Edit Instrument

Type

Model

Production Year

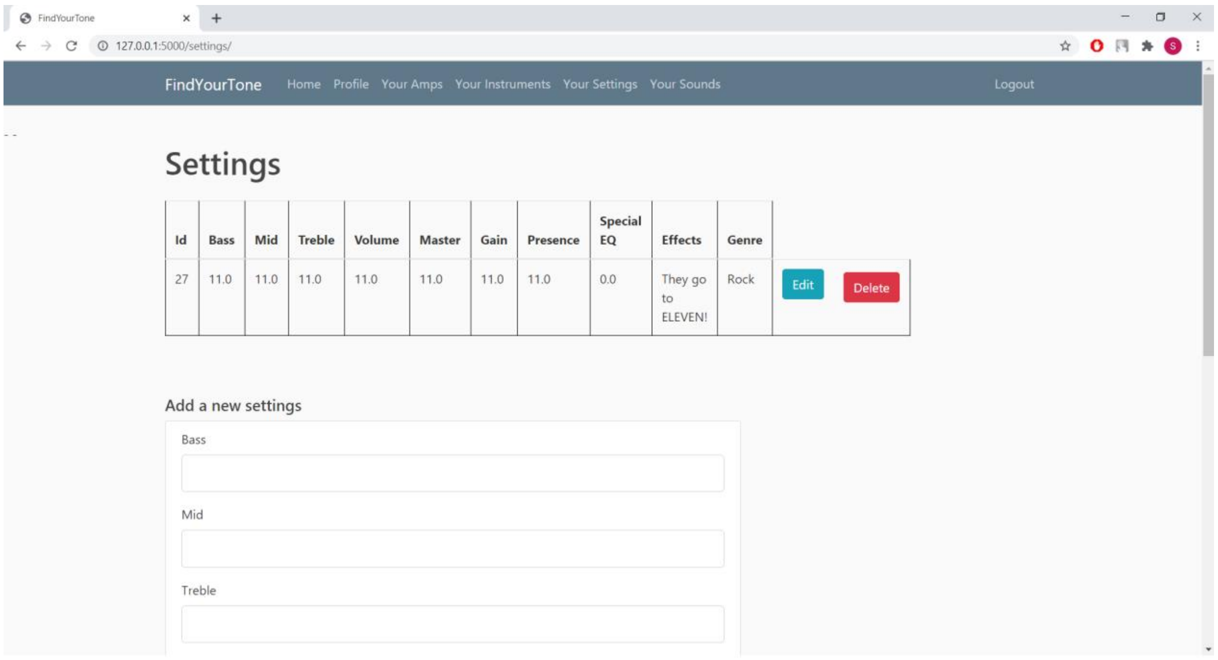
Modifications

2006

Link

Figure 10 Edit Instruments

- Users can go to Your Settings tab to add, browse, delete or edit settings belonging to them.



Settings

Id	Bass	Mid	Treble	Volume	Master	Gain	Presence	Special EQ	Effects	Genre		
27	11.0	11.0	11.0	11.0	11.0	11.0	11.0	0.0	They go to ELEVEN!	Rock	Edit	Delete

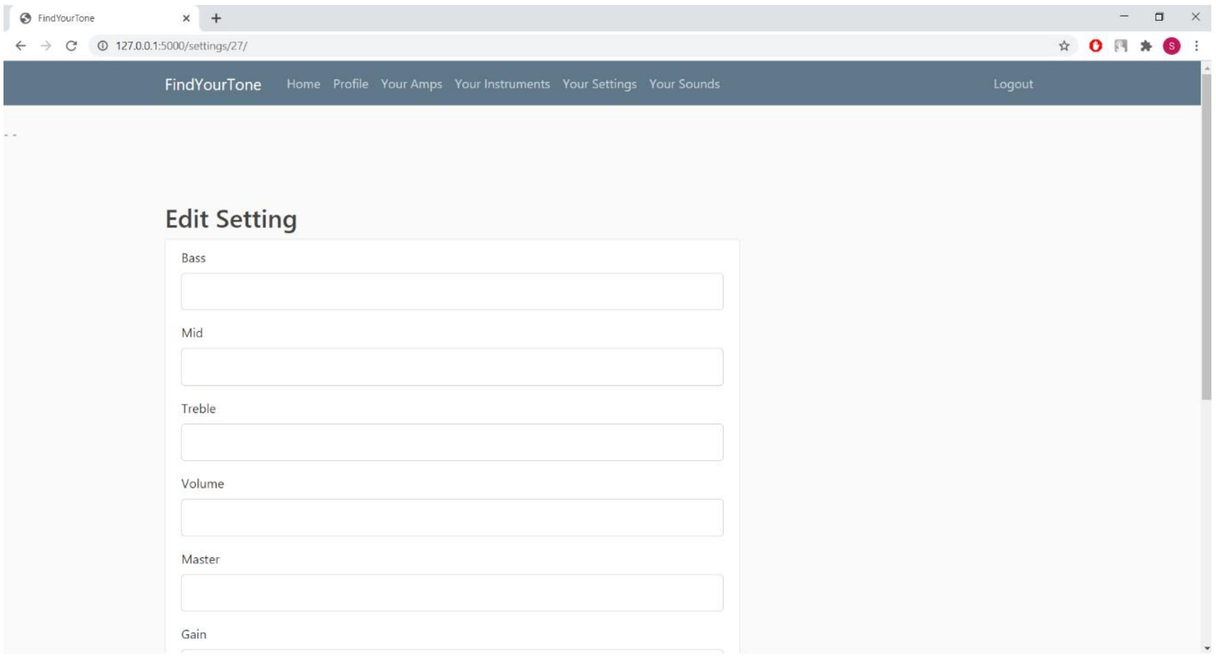
Add a new settings

Bass

Mid

Treble

Figure 11 Settings Page



Edit Setting

Bass

Mid

Treble

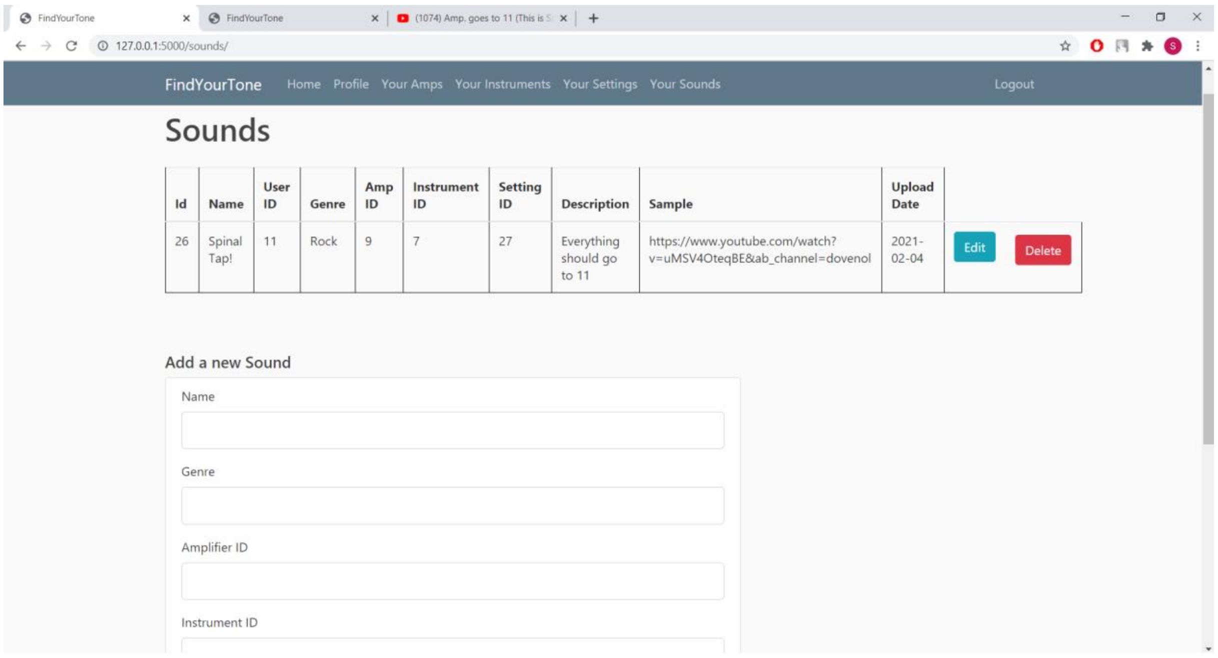
Volume

Master

Gain

Figure 12 Edit Settings

- Users can go to Your Sounds tab to add, browse, delete or edit Sounds belonging to them.



FindYourTone Home Profile Your Amps Your Instruments Your Settings Your Sounds Logout

Sounds

Id	Name	User ID	Genre	Amp ID	Instrument ID	Setting ID	Description	Sample	Upload Date		
26	Spinal Tap!	11	Rock	9	7	27	Everything should go to 11	https://www.youtube.com/watch?v=uMSV4OteqBE&ab_channel=dovenol	2021-02-04	Edit	Delete

Add a new Sound

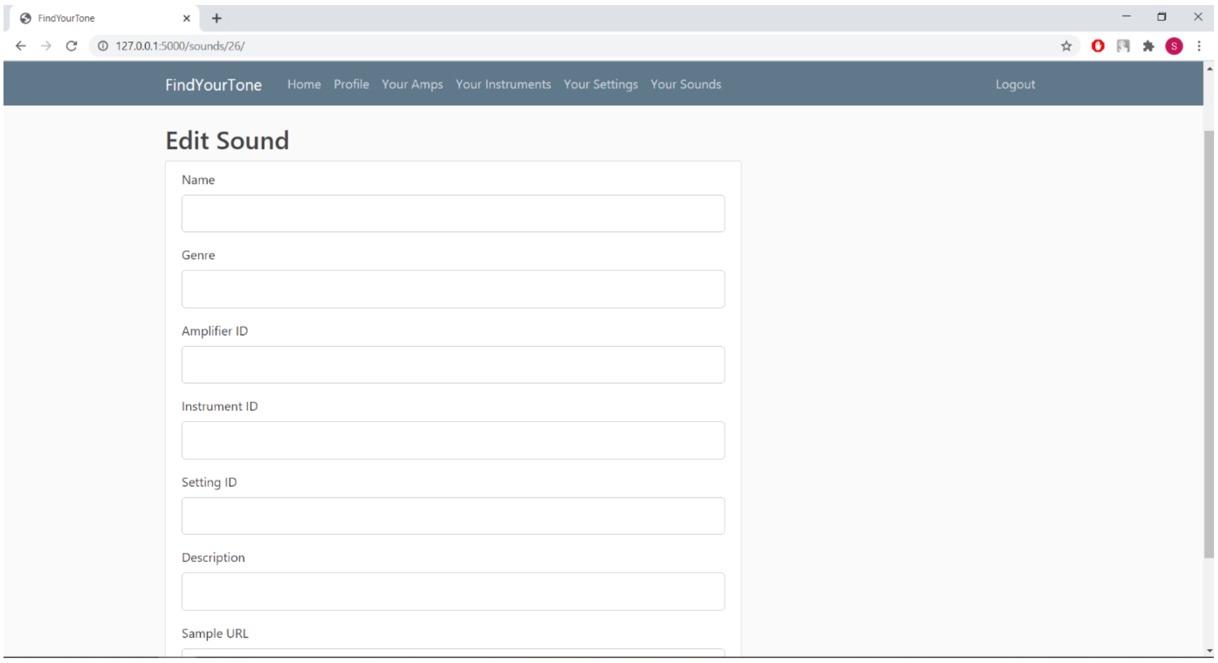
Name

Genre

Amplifier ID

Instrument ID

Figure 13 Sounds Page



FindYourTone Home Profile Your Amps Your Instruments Your Settings Your Sounds Logout

Edit Sound

Name

Genre

Amplifier ID

Instrument ID

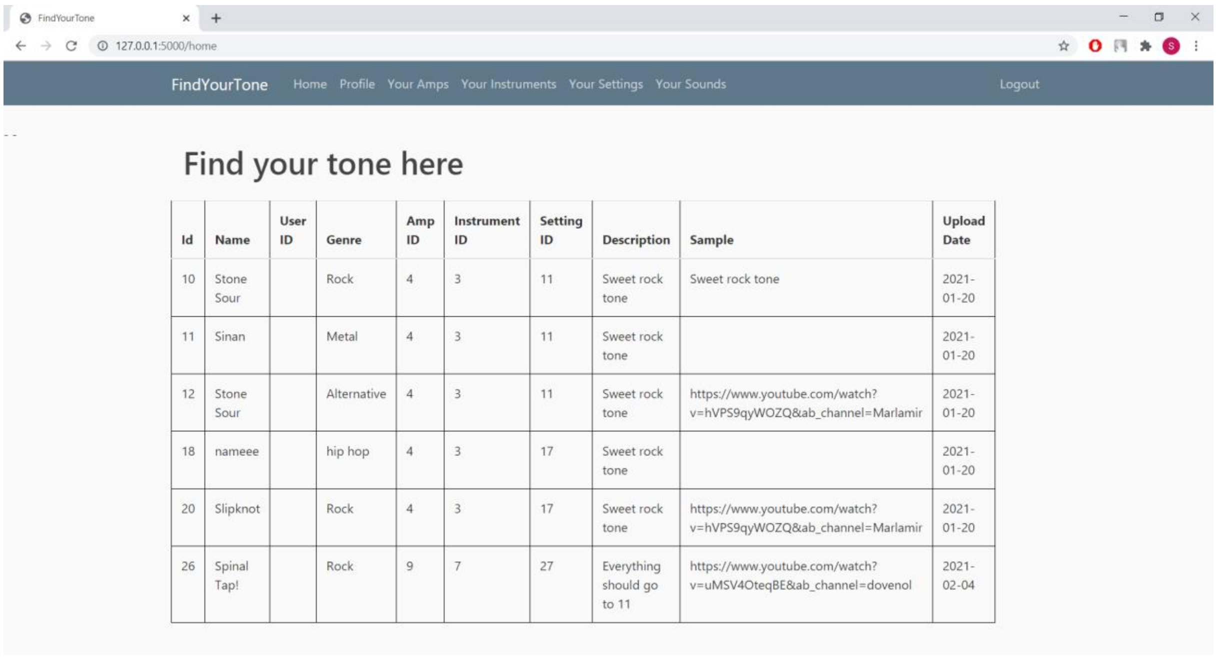
Setting ID

Description

Sample URL

Figure 14 Edit Sounds

- Homepage after test user added an amp, instrument, setting and added a sound using them. It is different from the homepage screenshot above(Figure 5).

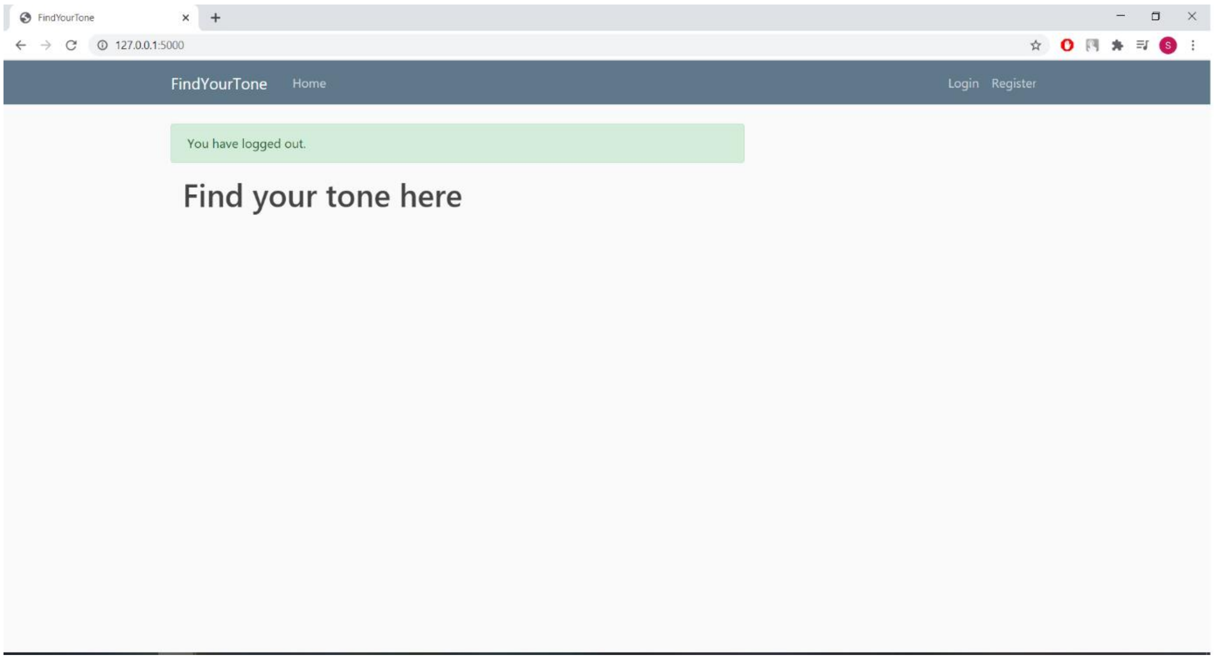


Find your tone here

Id	Name	User ID	Genre	Amp ID	Instrument ID	Setting ID	Description	Sample	Upload Date
10	Stone Sour		Rock	4	3	11	Sweet rock tone	Sweet rock tone	2021-01-20
11	Sinan		Metal	4	3	11	Sweet rock tone		2021-01-20
12	Stone Sour		Alternative	4	3	11	Sweet rock tone	https://www.youtube.com/watch?v=hVPS9qyWOZQ&ab_channel=Marlamir	2021-01-20
18	nameee		hip hop	4	3	17	Sweet rock tone		2021-01-20
20	Slipknot		Rock	4	3	17	Sweet rock tone	https://www.youtube.com/watch?v=hVPS9qyWOZQ&ab_channel=Marlamir	2021-01-20
26	Spinal Tap!		Rock	9	7	27	Everything should go to 11	https://www.youtube.com/watch?v=uMSV4OteqBE&ab_channel=dovenol	2021-02-04

Figure 15 Homepage after additions to db

- In Figure 16, an instance of a Logout can be seen. User is redirected to landing page



You have logged out.

Find your tone here

Figure 16 Application after user logged out

3. Installation Manual

Firstly, there are some requirements that have to be installed. Requirements to run this application are:

- werkzeug
- Flask==1.1.2
- Flask-Bootstrap==3.3.7.1
- Flask-Login==0.5.0
- Flask-WTF==0.14.3
- psycopg2==2.8.6
- passlib==1.7.4
- WTForms-Components==0.10.4
- FlaskSocket-IO==5.0.1

There were some issues with the deployment to Heroku hence the project demo has been done on localhost. But for deployment Gunicorn can be used. While deploying new credentials for the database used should be written in the necessary parts of the program.

Source code is available on Github: https://github.com/sinansar123/DBMSPublic