Musician's Network – Final Report

COMP0034

Group 10		
Name	**Degree**	**Student Number**
Bryan Gunawan	BEng Mechanical Engineering	17080399
Weizheng Zhang	BEng Mechanical Engineering	16081188
Dan Thompson	MEng Mechanical Engineering	16012244
Dillon Lad	MEng Biochemical Engineering	17053282

Contents

1. Introduction

Musician's Network is a platform designed for artists and venues to find and network with each other. Allowing artists to find each other may lead to occurrences such as collaborations, formation of groups or recruitment of artists. Venues themselves can look for artists that they may find suitable for their businesses. Through Musician's Network, it is hoped that the platform can provide connectivity that artists and venues may find difficult without.

A video explaining the different aspects of our web application can be found at: https://www.youtube.com/watch?v=n2wFme-zcnQ.

Source code can be found in Github link here: https://github.com/danthp4/Musicians_network_2020

2. Requirement and deliverables

Interim report submitted contains a list of deliverables that was aimed to be delivered considering existing constraints and limitations in the course. This list consists of use cases that are selected from a more comprehensive break down in COMP0035. Selected use cases mentioned are as follows.

- US01 – As administrator/musician/venue operator, I want to create an account so that I can access the website
- US02 – As a musician/venue operator, I want to be able to post my audio/video to my profile so that I can showcase my talent
- US03 – As a musician/venue operator, I want to be able to type in a search bar to be able to find other musicians/venues/locations/groups/genres, so that I can precisely access what I am looking for
- US04 – As a musician, I want to create and update a profile so that I can show my information
- US05 – As a venue operator, I want to create and update a profile of my venue so that I can show its details
- US09 – As administrator, I want to be able to block users from website so that I can enforce website policy

Challenge that is realised amid web design is that because of the presence of multiple account types (administrator, musician and venue operator), use cases such as US01, US02 and US03 can essentially be broken down into multiple other use cases for each user type. For instance, in US01, musician and venue operator have different account relationship or attribute, hence are basically 2 different processes, just like how US04 and US05 are separated. Similarly, US03 search results should be able to display results based on selected search type (musicians, venues, location, *etc*), further breaking it into multiple use cases. Nevertheless, web app designed is able to fulfil all mentioned use cases, with only slight adjustments.

For US01, both musician and venue operator can create accounts in the web app as planned. But as for administrator, authority has to be granted manually by injecting Administrator table under database with relevant relationship (see ERD). Admins will have to first create a musician or venue account normally, then someone with direct access to database will manually a relationship in database to grant admin rights. This is so that not everyone can create an administrator account without being approved by someone authorised (whoever has access to database).

US02 is also achieved but limited to a certain extent. Musicians are able to show audio only in a form of playlist through Soundcloud embedding system. While venue accounts are allowed to upload one image and one video with flask-uploads and YouTube embedding respectively. Decision in using embeddings is purely practical as there are various well-known platforms that allow such use case and learning it from scratch is not worth investing the time.

With functionality of forming a group being eliminated, searching for a group is not possible. Apart from that, use case US03 is fully achieved.

US04 and US05 are also successfully implemented into the web app as users are able to create and also update their profile information accordingly. These profiles can be blocked by administrators, disallowing them from entering the site at all, making US09 another fully functioning use case.

Large amount of effort was placed into creating a platform that adapts to different types of user. In addition, there is an extra functionality such as rating other users. With small number of justified adjustments in a couple of use cases, web app designed is able to achieve every use case mentioned under interim report.

3. Application design

3.1. Application Design

Web Framework

Musician's Network inner mechanism operates by using Model-View-Controller (MVC) framework. And the design of MVC mechanism in Python is supported with several other third-party modules. These various modules work collectively to run MVC framework and handle many functions implemented in Musician's Network. Full list of features in Musician's network can be found in appendix below. Its core features, however, can be grouped as follows: register & login, homepage, profile page, edit profile, search.

Flask is the key web framework in which Musician's network operate in. Various other extensions are needed to complete the application, but Flask provides the main foundation. This means that aspects available in the app is directed under Flask's capabilities.

The Model under MVC is a representation of database used. In this case, SQLite is utilised as a local database storage system. SQLite is chosen due to its simplicity. Interaction between an SQLite file stored in app is done through SQLAlchemy and Flask-SQLAlchemy. SQLAlchemy is an object-relational mapper that allows SQL file to be accessed in Python. On the other hand, Flask-SQLAlchemy acts as an extension to Flask that connects it to SQLAlchemy's use. With these modules, database model in a form of SQLite file can be easily accessed, edited and updated from web app.

SQL model is widely exploited in this app. Register & login surely needs access to database as it has to add a new record, or query relevant user to authenticate. Distinction of users are done by enforcing unique username and email for every user. Other main features such as homepage, profile page, search and even edit profile also involves database query. Homepage displays users by inquiring existing profiles and iterating over them. Similar to this, search and profile route shows profile(s) by querying database based on selected term. So, it is clear that model plays an important role here storing information for the app, directed by Controller.

Renderings or display of inquired model is supplied with a combination of HTML and Jinja. Jinja is another common Python extension that allows web templating of HTML. Presence of Jinja provides adaptability that is very much needed in the designed app.

More about testing is discussed in Section 4, but unittest here will be used to automate tests in Musician's Network.

Forms

In a route where a form is needed, WTForms is added into the HTML & Jinja mix for more robust inputting. This is because functions such as validation and error flashing are easily applied with WTForms and form information can be conveniently supplied from a separate file. Here, Flask-WTForm extension helps in integrating Flask and WTForm together. In some cases, Flask-Uploads need to be implemented to allow images to be included. Such combination of Jinja and WTForms performs gracefully in route like *edit_profile*. This specific route requires adaptability in supplying input form to different account type; something that is not achieved simply when none of these extensions are present.

Login uses Flask-Login, letting current user to be easily queried from *current_user*. Additionally, Flas-Login makes implementing functions such as locking a route for logged in user only and logging out straightforwardly achieved.

Musician's Network designed as of now allows musician account to upload a playlist through Soundcloud embedding and allows venue account to upload one image along with one video through YouTube embedding. Early on in the development, venue was allowed to upload more than a single picture and video, hence the function *media_counter()* under *prof/routes.py* and the presence of a separate Media table. Moreover, it was planned for Media table to have a relationship with *profile_id* instead of just *venue_id* to allow musicians to upload media too. These ideas were omitted for the reason discussed in the next paragraph.

Many other varieties of implementation could have been chosen indeed: allowing musicians to upload image too, musicians to upload YouTube embedding, venues to upload multiple images, or embed multiple YouTube videos, *etc*. However, final decision made in Musician's Network is purely based on the interest of time constraint given. It is projected to be possible but would take too much time to set it up as it requires more validation checks and others.

Soundcloud is a widely known web for showcasing music, but its API has to be manually approved by developers hence embedding is used. It is safe to assume that most musicians have or is recommended to have a Soundcloud account anyway. As for video, it is realistically too time consuming to learn how to upload videos, while there are many external platforms available. YouTube is a named video platform and it is assumed that venues are happy to have an account there to post videos for publicity. Being so, Musician's Network takes advantage of this and promotes it further in its own platform by simply pasting their video link. Through Musician's Network, the possibility of allowing users to have freedom to upload media of their own choice is shown to be possible by having functionalities of uploading/embedding playlist, image and video, but not done in the interest of time.

Design Overview

Comprehensive list of capabilities in Musician's Network is listed in the appendix. An overview of those list shall be discussed here with some explanation regarding why it was necessary.

Identical to many other online web applications, register & login is the baseline function. Entering index of web app shows user description of what Musician's Network is. But the same index would show differently when logged in. It would show a list of registered musicians instead. If logged in user is interested in looking into venues, a pagination link that brings user into index with venue profiles can be clicked. Default view for index page when logged in however remain to be profiles of musician as it can be safely assumed that the main interest in Musician's Network is for musicians.

From index, many other routes can be accessed. Search-bar can be used by typing a term to be searched and its relative category, which would bring user to a similar page to index but only displaying relevant profiles. Profiles can be further entered too to look for more detailed information regarding the target user; this includes current user's own profile. This profile display is different for musicians and for venues. Information displayed in mentioned profile cards can be edited by filling out a form in edit profile which also adapts to user's account type. Adaptability is a desired aspect in the design of Musician's Network as it is ideal to keep formats with similar attributes under one route.

Use of Jinja can be realised also in Navbar. Navbar changes based on current user's login status. Logged out user is not provided with the option of logging out and logged user is cannot see an option to login; unless they manually enter the route address which will lead to flash messages. Again, this reflects versatility within Musician's Network.

If current user is authorised as an administrator, they are able to block another user, but not another admin. This is done by clicking 'block user' under target's profile card, disallowing them to enter Musician's Network. Rating, however, is open for any user but limited to one recorded rate only to prevent users from rating another user indefinitely. Presence of rating attempts to quantitatively measure user's quality, perhaps a necessary quick-review system for such social media type web app. Sidebar offers sorted or filtered view of these rated users.

Limitations

Obviously, current Musician's Network has a large room for improvements, especially compared to other competitors on the internet. More use cases could have been included with more time, completing more use cases compared to select few essential use cases focused in this project. But placing those use cases aside and limiting perspective to current Musician's Network design, some improvements could still have been made with more time and below is a list of realised potential improvements:

- Current *main.index* route design splits off into 2 possibilities: showing musicians or showing profiles. Making a more adaptable rendering template and database query may produce cleaner route by uniting these possibilities.
- Display of Genres in each profile cards uses nested for loop iteration as Profile to Genre has many-to-many relationship. More efficient system may be implemented to improve performance of relating genres to each profile, especially with escalating registered users.
- Pagination can be implemented to split up multiple profiles view to different pages, improving performance especially with many users.

- Select-field for cities and genres in *prof.edit_profile* are not comprehensive. This may be improved by utilising APIs.
- As mentioned before, currently musicians can only embed a playlist, and venues can only upload a single image and video link. Allowing more uploads may be beneficial for users.
- When there is an empty non-mandatory record, such as empty location, genre or description under Profile table, profile card should adapt nicely to hide those empty fields.
- Letting venue account to upload more than one media the current database design allows it but prevented by controller for slightly simpler code implementations.
- Limit image upload size. VARCHAR is chosen as data type to allow this limitation to be made but not implemented in Musician's Network.
- Email could have been displayed in profile cards, but it was argued that it is part of a personal information and user may wish to choose to hide or show it. Such feature could have been made but it was decided that if users would want to show email this can be done in description.
- Manually registering administrator into database for authority may be improved by figuring out a robust and fool-proof way of registering admins through the web itself.
- This may be another use case in itself but having a messaging functionality may significantly improve Musician's Network capabilities in networking. As of for now, communication is limited to user's willingness on exposing their email or contact address.

While above are potential improvements, below are potential issues thay may be found in Musician's Network:

- It is realised that user is unable to perform search from *prof.profile* route due to clash between variable route and argument request. This is prevented by directing user to search from *main.index* but could be resolved better with more time.
- Birthdate formatting can be inconsistent with HTML input conversion
- No check is being done for rating blocked users. This is not expected to happen in the first place because blocked accounts are not displayed, hence no button for them to be rated. But user who may know how to enter rating route and supply argument may able to find a way around this.

3.2. Database Design

Design of database has changed slightly with originally proposed ERD from COMP0035. Most of the changes are elimination of unnecessary tables, taking into account reduction in use cases. Other changes include alterations in table content and relationships to accommodate design that was overlooked during high-level design. Figure below shows currently used ERD.

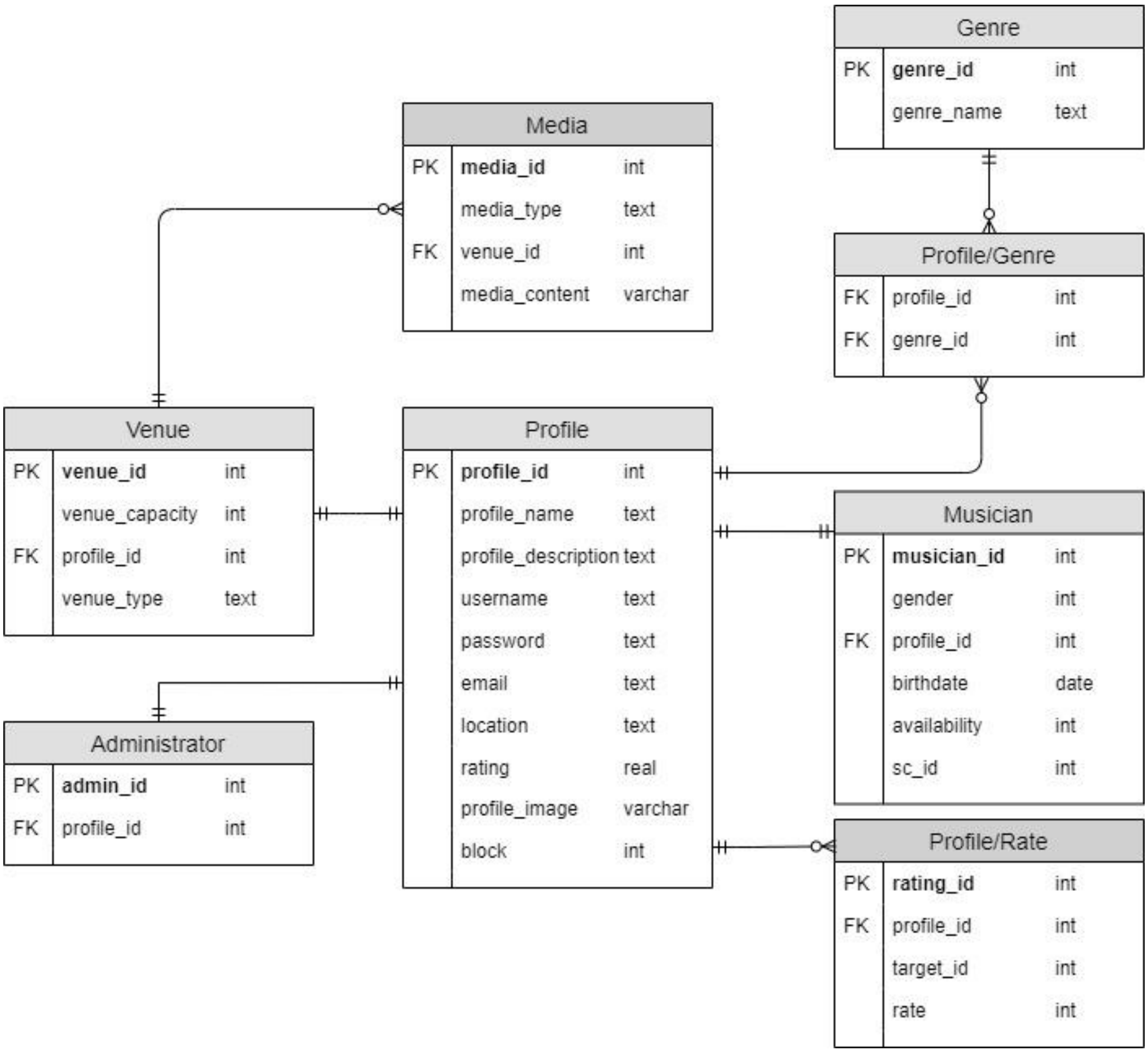


Figure 1 - Final ERD for Musician's Network

Design

Key table that is widely accessed here is the Profile table. Since both musician and venue account has overlapping attributes that they share such as username and password, it makes sense to have a table that represent that principal information. This Profile table is then extended into their respective types to store special data for that respective account.

Media table here is related to Venue table only as the current design only allows venue account to upload image onto their profile card. If only one image and one YouTube embedding is allowed in venue’s profile card, Media table and Venue table may be technically possible to be combined into a table only. However, as discussed in previous section, having a separate Media table opens the possibility of letting venue account upload more than one picture with one-to-many relationship although this is not currently implemented.

Profile/Genre table exist to help relate Profile & Genre's many-to-many relationship. Therefore, in routes like *main.index* or *main.search* where multiple profiles have to be shown, nested for loop in Jinja template is required to make every correct relation shows in every profile card.

Rating system is desired in Musician's Network. This works by taking an average of rates that a user has received from other user(s). To fairly run such system between user, a table is needed to record whom have rated whom. Otherwise, unrecorded rating between user will let users to indefinitely rate another user, undermining the averaging system. Additionally, it is impossible to make an average without having a form of record on how many rates have been made to a specific user. Thus, the Rate table is created. Rate table stores record of which user has rated which user and by how many. When a request is needed on what is the current rating of a specific user, this table is then queried to find out total number of ratings received from other registered accounts. Table is updated when new ratings are made.

Administrator table simply stores *profile_id*s of those who are authorised as admins. This is the table to be injected or manually written when an account is wanted to be registered as an admin.

Limitations

One limitation that has been mentioned clearly is the inability for venue accounts to upload more than a single image or YouTube embed. Musician's Network would be more interactive if more time was dedicated to allowing users, both musicians and venues, to post a media of their own choice.

Current design also depends on external websites, namely Soundcloud and YouTube. There are surely benefits in relying in popular sites as such, but this can be argued that external dependency is something that should be avoided if possible. Using BLOB type and storing media in the database itself may be better when multiple media types (image, audio and video) wanted to be handled internally within Musician's Network.

During edit profile, a user can enter location for their profile. But selections offered for this is very limited as it is unrealistic to manually write every single city. Using API is an option to improve this or having another table that stores list of cities and perhaps their countries too for thorough selections.

Although loosely linked to database limitation, administrator only being able to be authorised by developers who have access to database may not be suitable. It is not discovered yet on how this could be resolved but perhaps a better design of database may allow more robust registration of admins.

3.3. User Interface Design

With the focus of the project being placed on the MVC as a whole, HTML view is not created from scratch. A bootstrap template is taken from https://startbootstrap.com/previews/sb-admin-2/. With that said, it is therefore very unlikely to be able to find exact design as originally planned from sketches in wireframe. As a result, template is selected with consideration of being as close as possible to proposed design around profile cards. Main element in sketched design is to make use of profile cards that can be entered further into an expanded version of itself, with navbar and sidebar offering options to user as what can be done. This can be seen from sketches below.

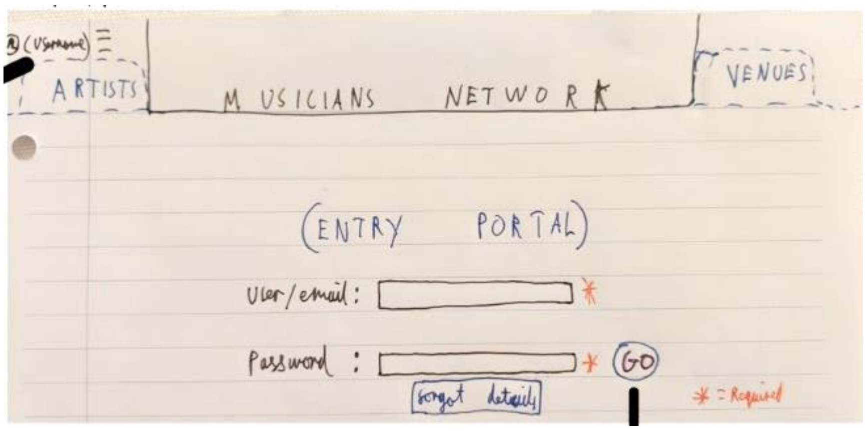
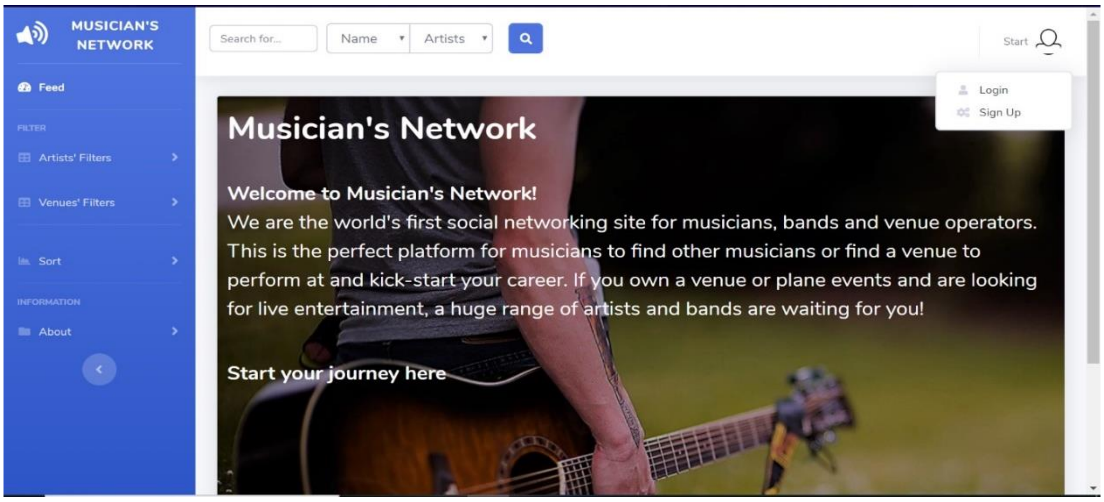


Figure 2 - Original wireframe of landing page



Musician's Network

Welcome to Musician's Network!
We are the world's first social networking site for musicians, bands and venue operators. This is the perfect platform for musicians to find other musicians or find a venue to perform at and kick-start your career. If you own a venue or plane events and are looking for live entertainment, a huge range of artists and bands are waiting for you!

Start your journey here

Figure 3 - User Interface for landing page with 'Start' button expanded

Figures 2 and 3 compare final design of landing page and the original wireframe. The two differ significantly. Conveniently, template selected offers a separate register & login page. This allows index before logging in to be designed into an about page, briefly explaining what Musician's Network does. Since it is not expected for new users to be immediately familiar with the app, such explanatory pages are found to be useful.

To enter the app, register or login page can be found by clicking 'Start' dropdown button in the corner of the index. Then, a login or register page will be shown as shown below. It can be observed that they look more similar to original wireframe's login section, which was drawn as a landing page. Image by the side is added for the interest of aesthetics. 'Log in with Google' and 'Log in with Facebook' buttons do not work as of now but were kept as the use of such third-party websites for user accounts could benefit the app if more time could be dedicated to connecting to them.

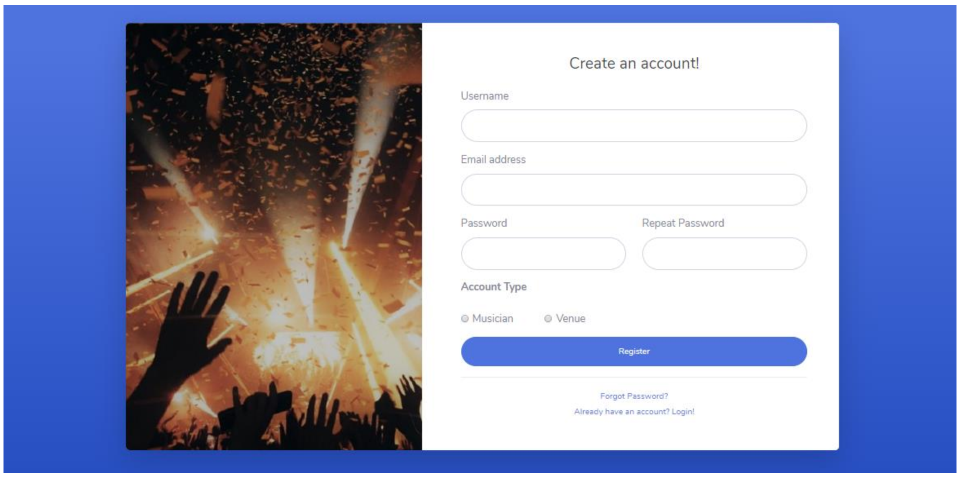


Figure 4 - Register page

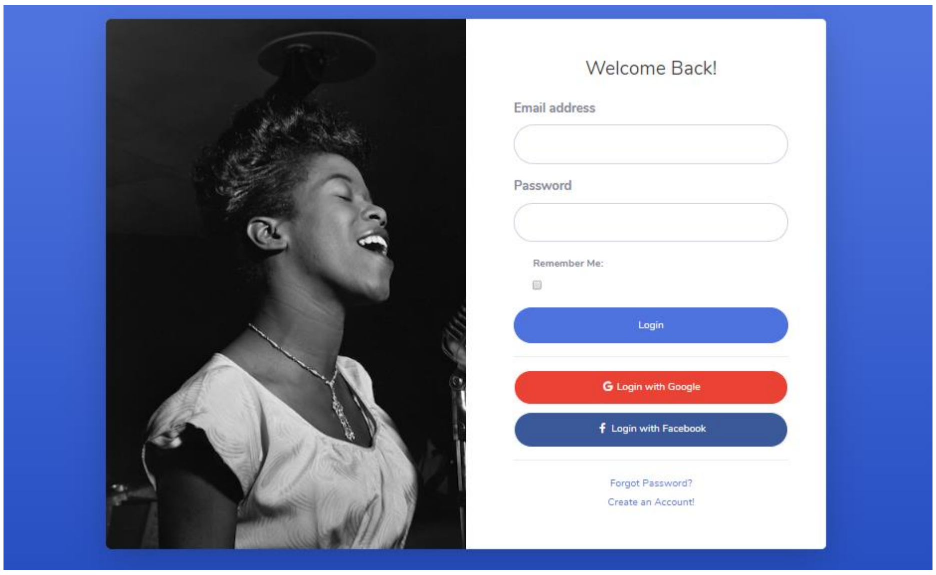


Figure 5 - Login page

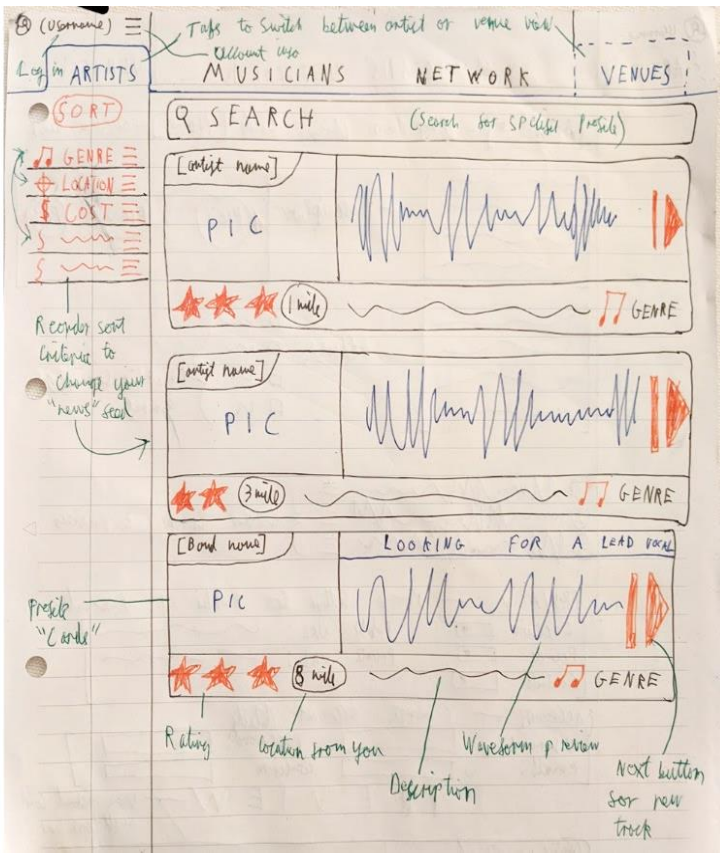


Figure 6 - Original sketch of homepage as user logs in

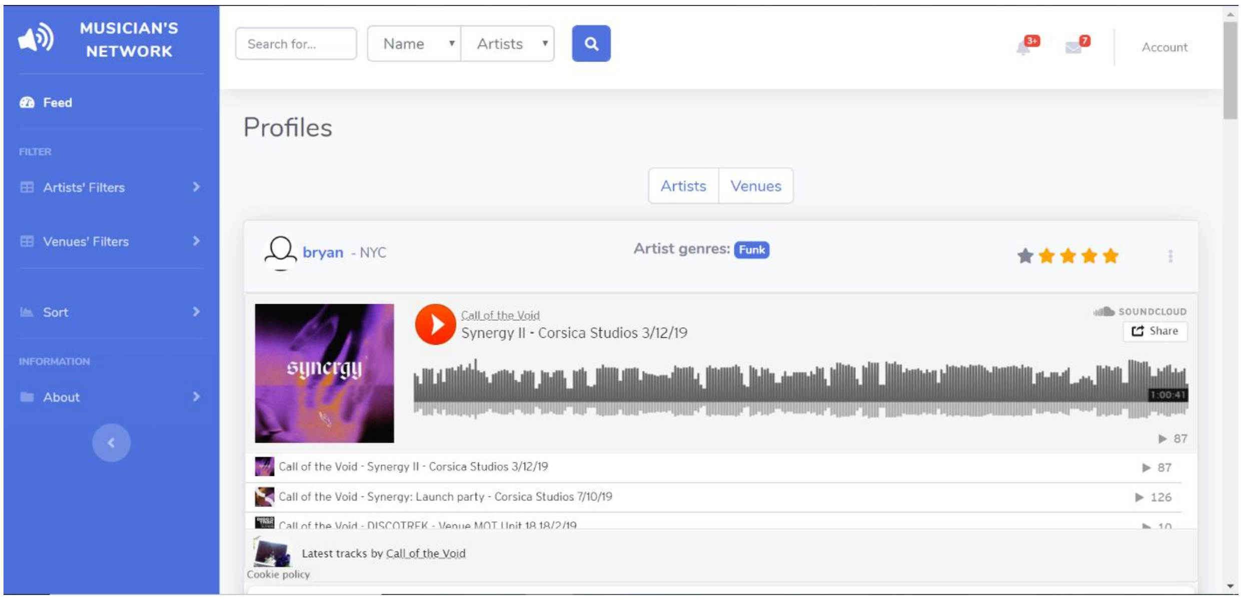


Figure 7 - Final User Interface of homepage

Figure 5 shows homepage as it was originally sketched. Final homepage looks similar to initially designed wireframe with profile cards system with only minor deviances in label locations. Profile cards in the main content of the homepage are surrounded by Sidebar and Navbar, each containing different functions listed in the appendix such as filtering artists based on their rating. Individual profile card seen here displays information under Profile table only. Profile image is added into the final design for better identification of users, giving more familiarity for musicians or venue owners when browsing for other users. More information, which expands query into information from Musician or Venue table depending on their account type, regarding that user can be found in their profile when entered.

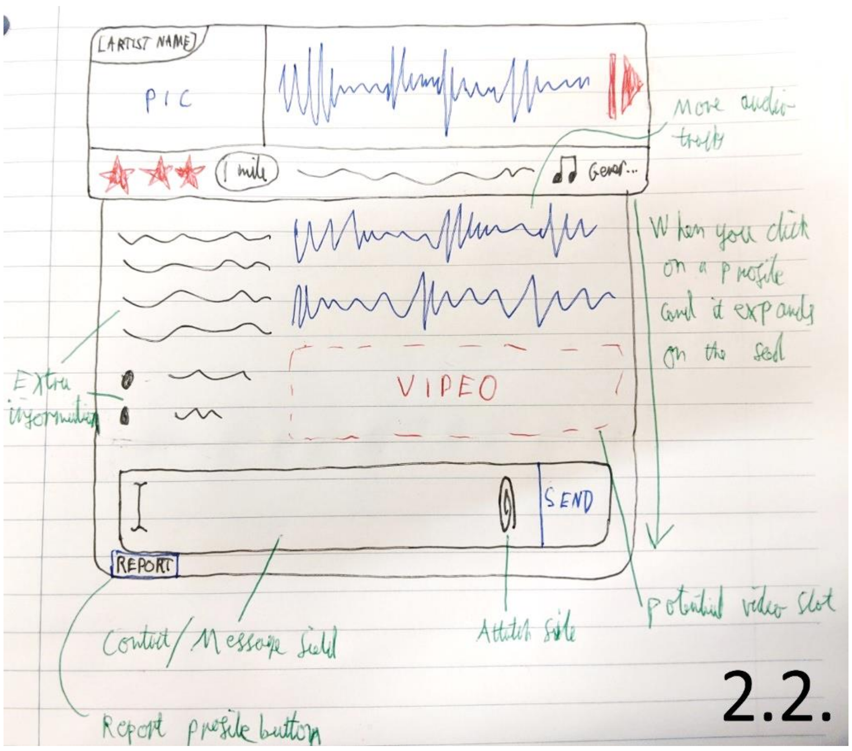
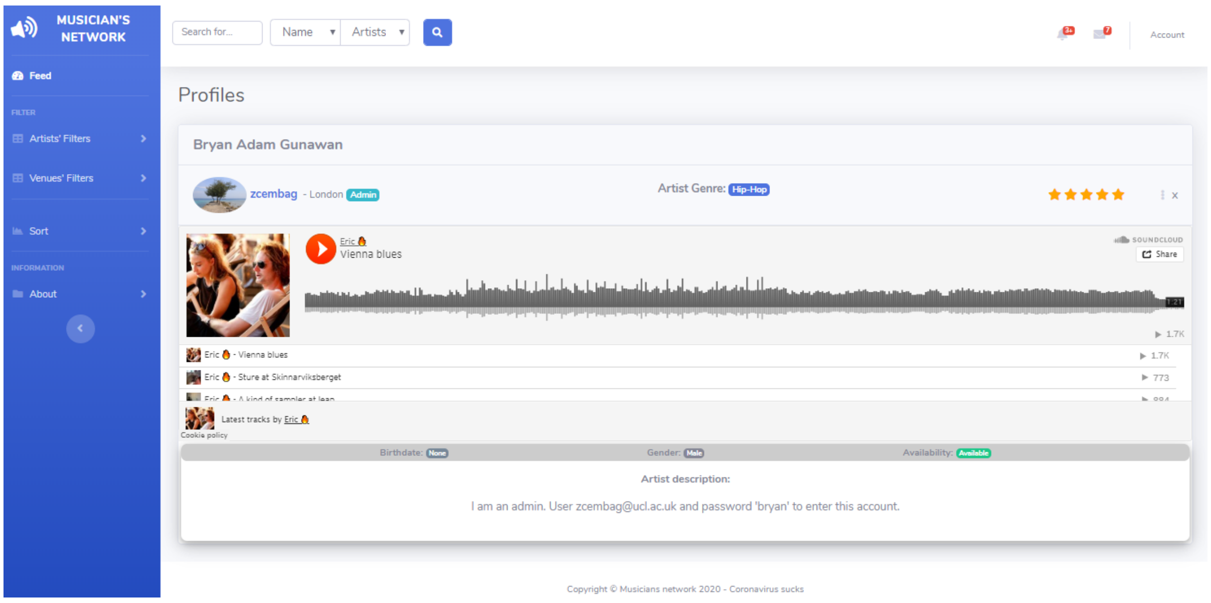


Figure 8 - Initial wireframe of expanded/full profile card

As seen in Figure 8, it was initially thought that full details of profile can be shown by expanding compact profile cards in homepage. However, it does not deliver a feeling of entering someone's profile information. Therefore, in the final design, full profile is provided separately in a variable route and shows information as such:



Profiles

Bryan Adam Gunawan

zcembag - London Admin

Artist Genre: Hip-Hop

Eric
Vienna blues

▶ 1.7K

Eric - Vienna blues ▶ 1.7K

Eric - Sture at Skinnarviksberget ▶ 773

Latest tracks by Eric

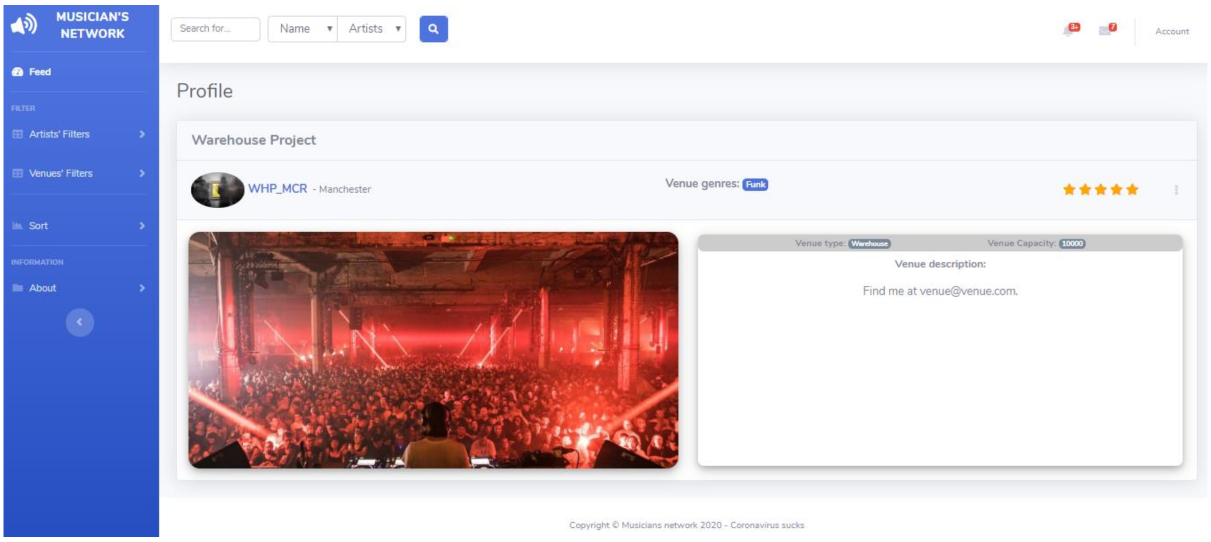
Birthdate: None Gender: Male Availability: Available

Artist description:

I am an admin. User zcembag@ucl.ac.uk and password 'bryan' to enter this account.

Copyright © Musicians network 2020 - Coronavirus sucks

Figure 9 - Example of full profile card for musician account



Profile

Warehouse Project

WHP_MCR - Manchester

Venue genres: Funk

Venue type: Warehouse

Venue Capacity: 10000

Venue description:

Find me at venue@venue.com.

Copyright © Musicians network 2020 - Coronavirus sucks

Figure 10 - Example of full profile card for venue account

Differences in full profile can be found in terms of content. Reporting is replaced by blocking function that is only accessible by admins for simplicity as reporting requires delivery of message to admins. It was also thought that a musician would be able to upload video, but this was discarded as discussed in the interest of time. Apart from that, the main difference in profile would be the content placement and style because of the need to adapt to two different account types: musician and venues.

Comparing edit profile, core aspects in terms of forms remain similar with only main difference being absence of preview as this would involve real time update of page as form is being filled in. Account edit form that allows user to change personal information (username, email and password) is separated instead of combined to produce a less cluttered form. Furthermore, as there are changes in database content, input forms also change accordingly. That aside, edit profile form remains similar with changes in form placement.

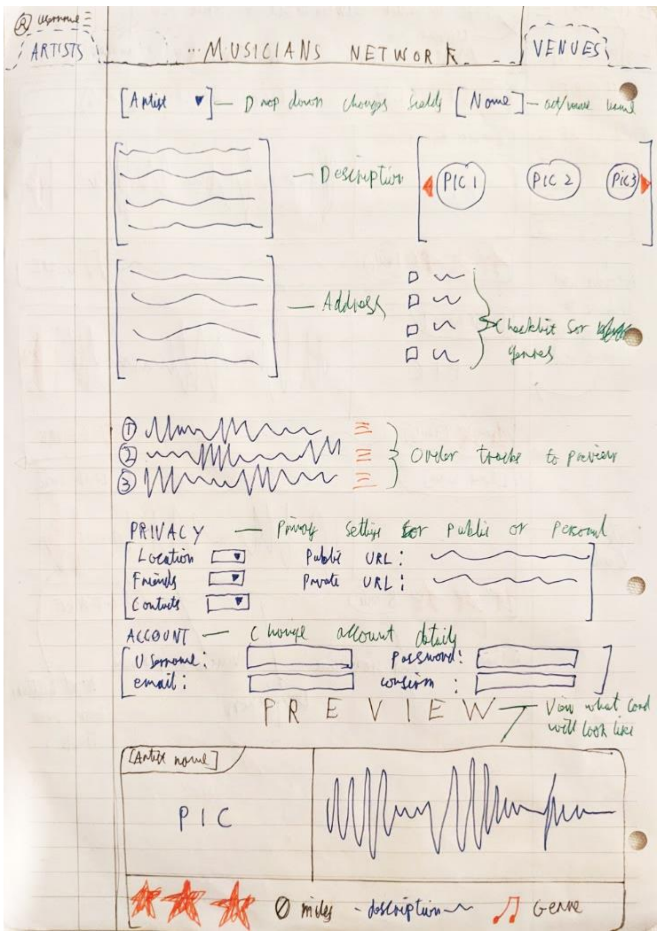
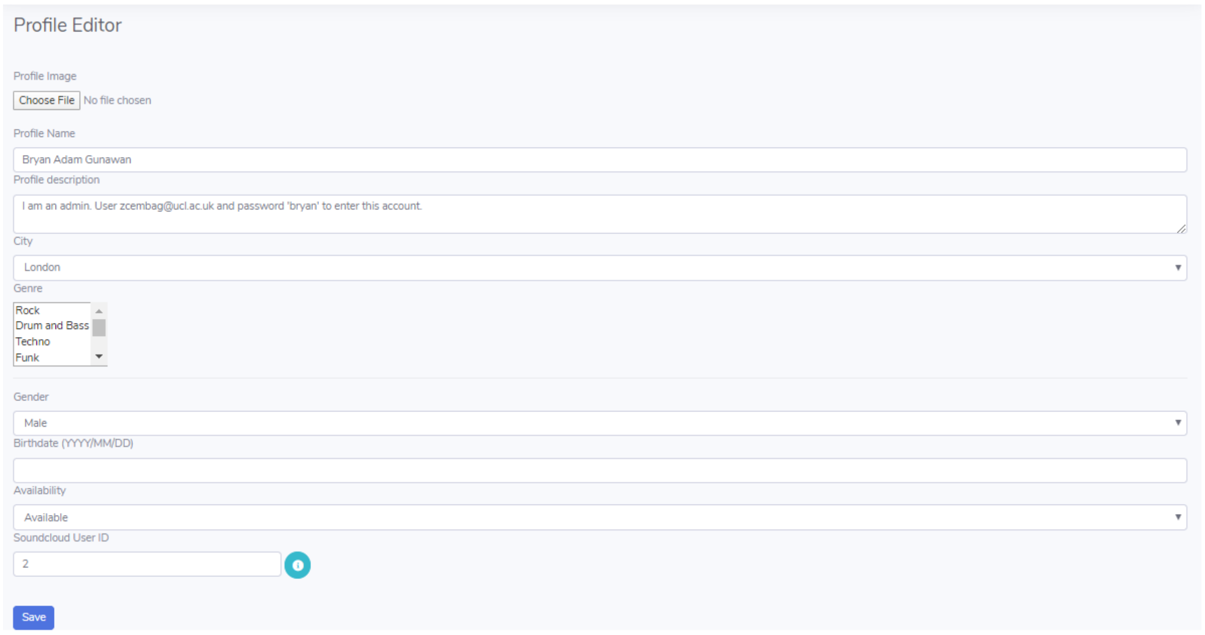


Figure 11 - Original wireframe for edit profile page



Profile Editor

Profile Image

Choose File No file chosen

Profile Name

Bryan Adam Gunawan

Profile description

I am an admin. User zcembag@ucl.ac.uk and password 'bryan' to enter this account.

City

London

Genre

Rock
Drum and Bass
Techno
Funk

Gender

Male

Birthdate (YYYY/MM/DD)

Availability

Available

Soundcloud User ID

2

Save

Figure 12 - Main section for final design of edit profile page

As for searching/sorting/filtering offered in navbar or sidebar, results are rendered under similar template to homepage, showing multiple profile cards in a page. Hence, no comparison is very much necessary here.

4. Testing and code quality

4.1. Code Quality

Prior to release, code is double checked to follow PEP8 Python standard. This is done by activating linting feature onto text editor used. The linting tool indicates mismatch between written code and PEP8's conventional standard, notifying user that format changes need to be made when necessary. This includes enforcement of 80 characters per line maximum, double space between functions, space after comma and others. In result, final Python code submitted follows PEP8 standard.

For HTML, indentation matters less in regard to function, so it is there to give clarity. Indentations hence were placed to easily distinguish opening and closing HTML aspects.

4.2. Error Handlers

Error catchers for unavailable page (Error 404) and internal server error (Error 500) is defined within Musician's Network. It leads to 404.html or 500.html renderings which are adapted from web template taken online. In the occurrence of these error happening, user is redirected to these HTML instead of default browser error output, allowing user to get back into other routes easily.

There are many cases in which input validation error can happen in Musician's Network forms. From authentication of registering/changing existing username/email, invalid login information, required data validation, maximum character limit error to file type validation error, all of these are caught by flashing message to user indicating why inputs are invalid.

Other than validation error, there are other handlers that manage different unwanted scenarios. When a search term is not found during searching, for example, flash message will be displayed saying no results were found. Or when a non-admin is trying to unrightfully block another user, that user will be redirected to main index without blocking anyone seeing a flash message of unauthorised use. Users are also unable to rate themselves to manipulate the rating system, user will also be flashed for this. Same goes for blocking. In summary, scenarios that were identified to be unwanted in Musician's Network is managed in a way that user is notified on what they are allowed or not allowed to do. More complete list about possible features can be found in appendix.

4.3. Testing

Approach

Requirements testing and unit testing are used to verify that web system developed runs as expected, fulfilling user stories defined at the beginning, and expose issues that may exist. Since Musician's Network system is relatively simple in terms of its requirements, most tests can be covered by just using unit tests. For other unreachable aspects by unit tests, manual requirements testing is applied as a supplementary tool to confirm user stories like US02 and US09.

Unit test is chosen for three reasons. First, Extreme Programming (XP) was decided to be the basis of development and unit test is an integral part of an XP process, making the programming development more Agile. Therefore, after part of a new function or use case has been released, tests can be run and issues are immediately captured. In this way, unit tests provide safer and more Agile working environment. Secondly, execution of unit tests is quick and simple as unit tests commonly have straightforward scope. Although a single test only covers a limited range of the whole project, tester can run tests separately in a short period of time. In other words, test can be applied specifically to a portion of the code that was modified. Finally, unit testing saves time and effort. The earlier issues are found, the easier it is solved. If tests were to be done in later stage, it would be more difficult and slower to find which changes cause rise to such errors.

Musician's Network run under Python so integrated Unittest framework was used to create unit tests for designed flask application.

Test setup

As mentioned, tests should be defined according to user requirements (see Section 2). Tables below show that two different testing methods were used to validate those requirements, and what acceptance criterion should be used for each verification.

Test cases

User Story ID	Value Statement
US01	As administrator/musician/venue operator, I want to create an account so that I can access the website
Acceptance Criterion	*Criterion 1*: GIVEN the new user has not signed up WHEN the new user goes to register page, and input correct information THEN check they have been signed up and enter the profile pages. *Criterion 2:* GIVEN a Flask application WHEN the new user goes to register page, and input incorrect information THEN ensure they cannot be signed up and rejection messages are displayed.
Test Approach	Automated Unittest
US02	As a musician/venue operator, I want to be able to post my audio/video to my profile so that I can showcase my talent
Acceptance Criterion	*Criterion 1*: Given that the user has signed up and has SoundCloud ID. When the user uploads an audio/video to SouldCloud and add the SoundCloud ID to profile Then the audio/video can be viewed in the user's profile
Test Approach	Manual requirements test
US03	As a musician/venue operator, I want to be able to type in a search bar to be able to find other musicians/venues/locations/groups/genres, so that I can precisely access what I am looking for
Acceptance Criterion	*Criterion 1*: GIVEN that the user has signed up

	WHEN the user type existing inform in the search bar and choose the search term and category THEN the responses show the correct search results *Criterion 2*: GIVEN that the user has not signed up WHEN the user tries to manually go to search page THEN ensure the page cannot be GET and rejection messages are displayed. *Criterion 3*: GIVEN that the user has not signed up WHEN the user type non-existent inform in the search bar and choose the search term and category THEN the response shows message of no results.
Test Approach	Automated Unittest
US04	As a musician, I want to create and update a profile so that I can show my information
Acceptance Criterion	*Criterion 1*: GIVEN that the musician has signed up WHEN the user input new information in the profile form THEN the musician's profile has updated with new input. *Criterion 2*: GIVEN that the musician has not signed up WHEN the user tries to manually go to edit profile page THEN ensure the page cannot be GET and rejection messages are displayed.
Test Approach	Automated Unittest
US05	As a venue operator, I want to create and update a profile of my venue so that I can show its details
Acceptance Criterion	*Criterion 1*: GIVEN that the venue operator has signed up WHEN the venue operator input new information in the profile form THEN the venue's profile has updated with new input. *Criterion 2*: GIVEN that the musician has not signed up WHEN the user tries to manually go to edit profile page THEN ensure the page cannot be GET and rejection messages are displayed.
Test Approach	Automated Unittest

US09	As administrator, I want to be able to block users from website so that I can enforce website policy
Acceptance Criterion	*Criterion 1*: Given that a user's account has not been block When the administrator clicks block in the user profile card Then the user cannot sign in, and rejection message shows in sign in page *Criterion 2*: Given that a user's account has been block When the administrator clicks unblock in the user profile card Then the user can sign in and enter the profile page *Criterion 3*: Given that an administrator's account has not been block When another administrator clicks block in the user profile card Then the user cannot be blocked, and the operator receives the rejection message.
Test Approach	Manual requirements test

Manual Test Results

Automated tests done by Unittest have all passed. Manual requirements testing results are recorded by Test Case spreadsheet below.

Test Case Field	Description
Test case ID	TS-US-01
Test by	Bryan Gunawan
Date of test	27/03/2020
Test Title/Name	Test post audio/video
Pre-condition	1. User must sign in an unblocked account 2. User must be able to access SoundCloud. 3. User must have a SoundCloud ID
Test Steps	1. Upload an vedio/audio to SoundCloud ID. 2. Go to edit profile and add SoundCould ID to the profile. 3. Confirm edition.
Test Data	Username: zcembag Password: bryan
Expected Results	Success upload the media can be viewed in profile
Post-Condition	Uploaded media can be viewed in profile
Actual result	Pass

Notes/Comments	
Requirements	US02

Test Case Field	Description
Test case ID	TS-US-02
Test by	Weizheng Zhang
Date of test	28/03/2020
Test Title/Name	Block/Unblock user
Pre-condition	1. User must sign in an administrator account 2. At least another musician or venue account is existing.
Test Steps	1. Search the user should be blocked/unblocked 2. Click block/unblock at the user's profile card
Test Data	Username: vizon Password: vizon
Expected Results	Success block/unblock an account
Post-Condition	The account holder cannot sign in
Actual result	Pass
Notes/Comments	Account cannot be log out if it has been signed in
Requirements	US09

Remainder requirement test results can be found in the appendix along with other tests for the web app, including route tests and database tests.

Limitations

Manual testing is something that is undesired in web development. However, it was still used here because of Unittest's limitations. With more time, a combination of other testing framework such as Selenium should be implemented to automate as much tests as possible.

Based on carried out tests, no problem was identified. This means that basic requirements have been successfully implemented. However, it is not to say that the app is perfect. There may still be many issues with current testing plan. In terms of project development or management, some codes were made before the automated tests. Additionally, feedback from test results cannot completely reflect adjustments to be made on the project.

Performance and security testing have not been rigorously applied to Musician's Network to expose problems that may arise in more practical scenarios. As a form of social media platform, perhaps functionality testing that gauge app's usability is still insufficient. Furthermore, performance testing and early release would be ideal for a proper production. But this is out of the current project's scope. Further development stage may need to focus on versatility on the web itself.

5. Conclusion

Musician’s Network submitted is able to achieve all previously mentioned use cases to be delivered, with only minor variances to lightly tune down complexity. This small change in requirement limits user’s freedom in terms of what a user can do with their own profile. At the moment, the platform is able to perform as a baseline application where users can network. Further progress may be dedicated to putting more features into the app, widening use cases for Musician’s Network. Functionality such as messaging, allowing musicians to also upload videos, upload multiple media into profile cards and others may be more desirable for use. But placing functionality aside, there are still other parts that may be helped with more time.

Automated testing at this current stage does not cover every necessary point. Some manual requirement testing was required. Further improving automated testing will help advance project development.

Nonetheless, Musician’s Network designed functions to its purpose and ticked every point originally laid out. With a platform ready to use, further developments can be built on more easily on top of currently established system.

6. References

Bootstrap web template: https://startbootstrap.com/previews/sb-admin-2/

Third party packages:

- werkzeug
- pytest
- attrs==19.3.0
- Click==7.1.1
- devconfig==0.4.8
- Flask==1.1.1
- Flask-Login==0.5.0
- Flask-SQLAlchemy==2.4.1
- Flask-WTF==0.14.3
- itsdangerous==1.1.0
- Jinja2==2.11.1
- MarkupSafe==1.1.1
- python-json-logger==0.1.11
- PyYAML==5.3.1
- SQLAlchemy==1.3.15
- WTForms==2.2.1
- decorator==4.4.2
- Flask-Session==0.3.1
- infinity==1.4
- intervals==0.8.1
- selenium==3.141.0

- six==1.14.0
- SQLAlchemy-Utils==0.36.3
- urllib3==1.25.8
- validators==0.14.2
- WTForms-Alchemy==0.16.9
- WTForms-Components==0.10.4

7. Appendix

Contribution

NAME	CONTRIBUTION
BRYAN GUNAWAN	25%
DAN THOMPSON	25%
WEIZHENG ZHANG	25%
DILLON LAD	25%

Full list of features

Baseline functions:

- Register @/register
  - All input fields are required: *username*, *email*, *password*, *account type* (used to create musician/venues record)
    - Flash message showing required data when no entry
    - Limit of 20 characters for username using Length() (actually I haven't tested this)
    - Email validated using Email()
    - Password is hashed
  - Registering creates a Profile record with entered *username*, *email*, *password* and set block = 0; indicating created account is not blocked from musicians' network. Other attributes in Profile table are left to None for now.
    - If account type = musicians, then create Musician record with *profile_id* linked to Profile. This will indicate linked Profile is a Musician, not Venue.
    - If account type = venue, then create Venue record with profile_id linked to Profile.
  - After successfully registering, user is directed into main.index showing musicians.
  - *Username* and *email* entry have to be unique. Validated by a function under auth/forms.py
    - Flash message is shown when not satisfied
  - Logged in user trying to enter route @/register is redirected to main.index

- Login @/login
  - All input fields are required: *email*, *password*
    - Flash message when no entry
  - Successful registration is redirected into main.index showing musicians profile
  - Failed login if no user is registered with entered email OR password does not match user with entered email
    - Flash message 'Invalid email/password combination'

        - Redirected back to login page
    - Failed login if user is found, but its block = 1; meaning they are blocked from web
        - Flash message 'You are blocked from Musicians' Network'
        - Redirected to about page (main.index)
    - Signing up and logging in page is not accessible if user is authenticated. Logout is needed to access this route. Flash message will indicate this functionality.
    - Logged in user trying to enter route @/login is redirected to main.index
    - Remember me remembers user within one minute

- Logout @/logout
    - Logs out user and redirects user to main.index, rendering about page
    - Logout can be accessed on top right corner only when user is logged in as navbar adapts to login status.
        - Pop up will be shown for logout confirmation
- Error catchers
    - Renders 400.html or 500.html for their respective error if found.

Web Features:

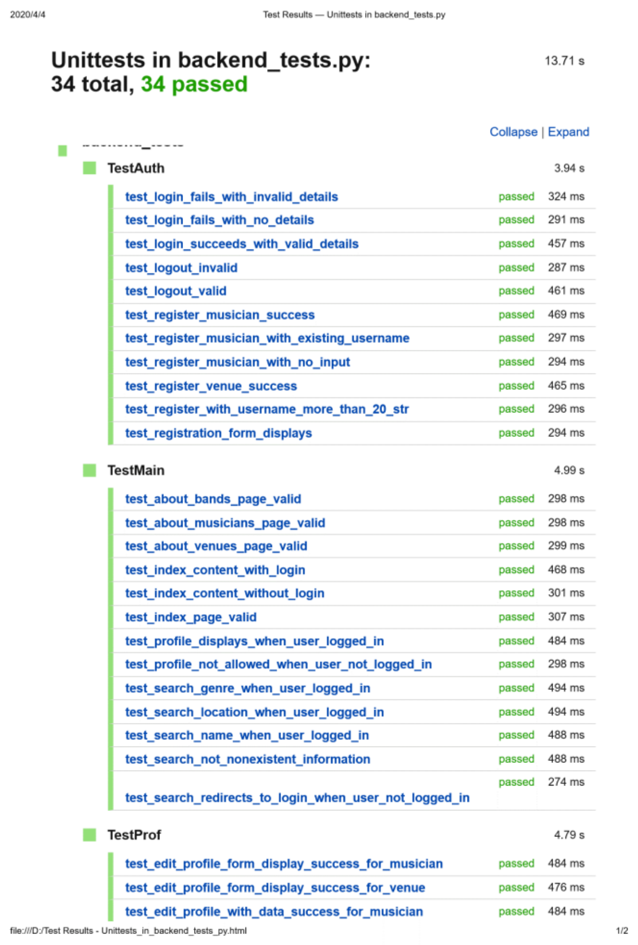
- Main @/main
    - Logged out user gets about page rendered in index.html, instead of Musician Network's homepage
    - Main route checks logged user whether they are an admin.
        - Admin gets to see every profile in database, including blocked profiles
        - Non-admin only gets served unblocked profiles
    - Main route takes in a parameter/argument which corresponds to account type separated by a pagination link
        - If *argument* = *musicians* or *no input* (which means default), then route renders musician profiles to be shown/displayed
        - If *argument* = *venues*, then route renders homepage to be filled with venues profile cards
        - Admin functionality on showing unblocked users only for non-admins work for both *musicians* or *venues* homepage
        - Both renders have different attributes needed and therefore is separated into 2 *if functions* and *return*
    - Since Profile table and Genre is a many-to-many relationship, Profile/Genre intermediated table is needed
        - This is Profile/Genre relationship is queried and iterated in html rendering with jinja to check genre for each profile listed in homepage
    - Administrator status for each user is queried to give badge for admins in rendering
- Search @/search
    - Search route takes in 3 arguments: *search_term, category* and *search_type*. These will be used to query relevant data.
        - *Search_term* is simply the term wanted to be searched
        - *Category* assigns the *search_term* into intended search category by user: username, location or genre.
        - Results shown depends on selected *search_type.* If *Artists* is selected, *musician* profiles are shown, otherwise *venues* result are shown.

    - Similar to main route, for admins, every user is shown despite them being blocked. This works on all different kinds of *search_type* or *category*.
    - Only accessible when logged in
        - Flash message from @login_required if attempted from logout
- Profile cards: an html used to represent a profile
    - Musicians
        - Information under Profile table is shown in each card (username, location, rating, genres, profile description and embedded Soundcloud playlist)
        - More comprehensive information about musicians can be found under Musician table and will be shown when someone enters user's profile in @profile/<username> route
    - Venues
        - Information under Profile table is shown in each card. For venues, users are allowed to upload a single picture describing their venue, with a YouTube embed pop up when clicked.
        - More detailed information about venues are under Venues table linked to Profile and is shown when profile is entered
    - Genres are generated by iterating over relationships from Profile/Genre table and matching *profile_id*
    - Admin badges are granted besides their username for registered admin users
    - Block badges are also given to indicate their status for admins
- Block @/block
    - Block route takes in a parameter, which is *username. Username* is then used to query user to be blocked
        - Current user's authority is first checked. If user is an admin, then proceed to block specified user.
        - If user is already blocked, the option offers to unblock target user instead
        - If current user is not an admin, they are redirected to main.index showing musicians with a flash message.
    - Block can be accessed by clicking on the triple dot button in each profile card corner
- Rate @/rate
    - Rate route takes in 2 parameters: *username* and *rate_value.*
    - User is not allowed to rate their own profile
        - Flash message 'you are unable to rate yourself'
        - Redirect to main.index
    - *Username* parameter is used to query target user to be rated and query previous rate records.
        - Since a user is only allowed to rate a target user by one value only, Profile/Rate table is needed to record who rated who and by how much
        - If current_user has never rated target_user, then new record under Profile/Rate is made recording rating value given
        - Otherwise, previous rating record between them is updated with new rating
        - This update is reflected in target_user's rating by updating their account under Profile table with a new average out of all recorded rating for *target_user* in Profile/Rate table (done through iteration)
    - Rate can be accessed by any user using triple dot button in each profile cards. A pop up will appear showing number of stars user wants to give target user.

    - Since blocked user is not shown in the first place, thus they are unable to be rated through link
- Administrator: account that is granted power to block certain users
    - Administrator is created by manually injecting/assigning administrative authority into the Administrator table in database. This is done by altering or adding Administrator table. Manual creation ensures that admins are only assigned by people who can access database.
    - Admins will have to first create an account, musicians or venues. Then when granted admin, a badge will appear by the username and main.index shows every profiles.
    - Sidebar also shows filtering option to see blocked users only (musicians or venues)
- Blocked @/blocked: gives out list of users like main.index, but only blocked users
    - Only admins are able to access this route, so users are first check for authority
        - Unauthorised user is flashed 'You are not authorised…'
    - Blocked route takes in *account* parameter
        - If account = musicians, then query blocked musicians only
        - If account = venues, query blocked venue profiles only
    - if blocked route is addressed manually without argument, flash message 'invalid request' to main.index
- Ratings @/ratings: filter users by their rating
    - Ratings route takes in a parameter *account* and *star*
        - If account is musicians, then output gives list of musicians with specified rating
        - Otherwise, output gives a list of venue profiles with specified rating
    - Profile ratings are filtered by specified argument *star*
        - since ratings are calculated by average, value is not integer, so range needs to be specified for query. Range is calculated with *rate_range_calculator()*
        - if star = 1, then rating route gives out profiles with 1 star rating only
        - and so on for other stars…
        - if star = 0, this is interpreted as all user, so route will give out every profile SORTED in descending order from highest rate (5 stars)
    - if ratings route is addressed manually without argument, flash message 'invalid request' to main.index
- Profile @/profile/<username>: variable route that displays a user's full profile
    - Target user is queried based in variable input <username>, so manual entry is also accepted
        - If user with such username is not found, flash message
        - If user.block=1, then user is blocked, flash message 'user is blocked, return to main.index
    - User with specified username can correspond to either musician or venue
        - This is checked by querying both Musician table and Venue table for user.profile_id. Whichever existed is taken
        - Musician profile uses a special html template, and also venue has its own
    - Profile route can be accessed through triple dot button in every profile card's corner
- Edit Profile @/edit_profile: where user can complete their account
    - Profile name and genre are required field.
        - If profile image is attached, replaces profile image with a new one (includes deleting old file), or create new image if there was previously none
    - Form input adapts to current user's account type

            - If user is musician, form input displays form for Musician table
            - Otherwise form input is for Venue table
                - Checks venue image during update, if there was one new image replaces old image (include deleting old file), otherwise an image is created
                - Checks youtube image during update, also limited to one
            - If not identified, flash message
        - Previously recorded information is set as default input
            - Recorded information also adapts to account type accordingly to fill in their special forms
        - Database allows venue to have more than one image, but controller does not allow this
            - This is why *media_counter()* function is present, it was written to count number of media stored by a user to check it against a defined limit – in this case one.
- Settings @/settings: change personal information
    - All fields are required: username, email, password.
        - User cannot update username with existing username
        - User cannot update email with existing email
        - Validation error flash message otherwise
- Sidebar filters
    - Filter by number of stars can be done through side bar
        - There are options for musician results and also venue results, both adapts to selected *account_type* to be filtered through ratings route.
    - For admins, seeing blocked users option can be found under side bar too
        - Results also adapt to selected *account_type* of blocked users.
        - No flash message for empty results
    - Rating sort can be used to query every profile based on their rating in descending order
        - Results adapt to selected *account_type* in sidebar

Test Results



2020/4/4 Test Results — Unittests in backend_tests.py

Unittests in backend_tests.py:
34 total, 34 passed 13.71 s

Collapse | Expand

[illegible]

TestAuth 3.94 s

test_login_fails_with_invalid_details	passed	324 ms
test_login_fails_with_no_details	passed	291 ms
test_login_succeeds_with_valid_details	passed	457 ms
test_logout_invalid	passed	287 ms
test_logout_valid	passed	461 ms
test_register_musician_success	passed	469 ms
test_register_musician_with_existing_username	passed	297 ms
test_register_musician_with_no_input	passed	294 ms
test_register_venue_success	passed	465 ms
test_register_with_username_more_than_20_str	passed	296 ms
test_registration_form_displays	passed	294 ms

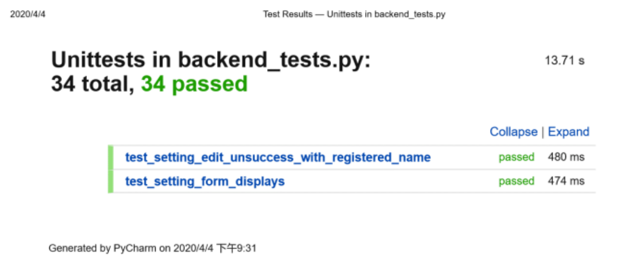
TestMain 4.99 s

test_about_bands_page_valid	passed	298 ms
test_about_musicians_page_valid	passed	298 ms
test_about_venues_page_valid	passed	299 ms
test_index_content_with_login	passed	468 ms
test_index_content_without_login	passed	301 ms
test_index_page_valid	passed	307 ms
test_profile_displays_when_user_logged_in	passed	484 ms
test_profile_not_allowed_when_user_not_logged_in	passed	298 ms
test_search_genre_when_user_logged_in	passed	494 ms
test_search_location_when_user_logged_in	passed	494 ms
test_search_name_when_user_logged_in	passed	488 ms
test_search_not_nonexistent_information	passed	488 ms
test_search_redirects_to_login_when_user_not_logged_in	passed	274 ms

TestProf 4.79 s

test_edit_profile_form_display_success_for_musician	passed	484 ms
test_edit_profile_form_display_success_for_venue	passed	476 ms
test_edit_profile_with_data_success_for_musician	passed	484 ms

file:///D:/Test Results - Unittests_in_backend_tests_py.html 1/2



2020/4/4 Test Results — Unittests in backend_tests.py

Unittests in backend_tests.py:
34 total, 34 passed 13.71 s

Collapse | Expand

test_setting_edit_unsuccess_with_registered_name	passed	480 ms
test_setting_form_displays	passed	474 ms

Generated by PyCharm on 2020/4/4 下午9:31