Big Binary Brand - TrackSpot

Project Overview

Our project vision remains very similar to our original goal and our second submission. We want our website to be a destination for which user and critic reviews can be gathered together for albums and songs.. However, we have changed some of the internal model structure to accommodate user authentication. We removed our own custom user and critic models and simply added a profile model to store all of the additional information for users. Then, we used Django authentication groups to separate users into normal users and critics.

Team Members

John Sweeney, Daniel Macdonald, Dongwon Park, Tommy Do, Aric Huang, Cameron Scigliano

GitHub Repository

https://github.com/jswny/trackspot

Design Overview

Our application uses Django's default authentication system to provide user and group models. We have implemented a login form, and a button to logout for a given user. We have also implemented forms for users and critics to create new accounts. If the user is a critic, they will be shown the "organization" field when signing up for their account. Once a new account has been created and the user is signed in, they can post reviews to songs and albums. To do this, we have created a review form which allows users and critics to rate the item and describe their review for it as well. In addition, users can view their profile and edit any of their profile fields such as their avatar, location, and bio.

Problems and Successes

For this project, we communicated and met up in person more frequently in order to prevent some of the problems that we encountered last project. This worked quite well, as we needed a greater level of teamwork for this project since most of our assignments were dependent on other team members' contributions. So, we tried to layer our development as much as possible so that we could all work on things concurrently, which worked quite well. We also began development earlier so that we had plenty of time to coordinate our work and fix any bugs or hard parts of the application that we encountered. Also, since most of us had spent a good amount of time figuring out Git from the last project, we were much more successful in using Git this time, and didn't encounter many problems with merging or branching.

However, we still had a few problems coming up how some of our authentication features would work. It was hard to figure out how everything would work without first having someone experiment with groups and authentication so that we would know what would work best. For instance, how to determine if a user is signing up as a critic and then from there provide the "organization" field for them to fill in. In general, we found it a little cumbersome working with the different groups (types of users) in our application and making sure that all of the authentication functionality was implemented correctly for each kind of user.

Team Choice

For our team choice, we chose to implement a search option, sorting through albums, songs, artists, and users. After we input a search, we will be redirected to a search results page displaying the matches.

Write-Up: John Sweeney

Approximate contribution: 20%

For my part of the project, I migrated our existing user models to the Django authentication user models. Then, I created the normal user and critic groups in the Django admin site and assigned some test users to those groups so that we could demonstrate our functionality. I then removed the old critic model and replaced our old user model with a profile model, which I linked to the user model, as a way of inputting extra information to users. Then, I documented my changed in the readme and also documented how to query and work with the new system so that other members could more easily understand how the new model structure worked.

Finally, I added the current user's profile link to the navbar, moved all of our existing reviews to the new test users, and fixed any errors that came from the new version of our data model. After that, I worked on helping others with various Git difficulties and any other things that came up when trying to implement the rest of the authentication and form functionality. Then, I worked on the team write up and also my own individual write up.

Write-Up: Cameron Scigliano

Approximate contribution: 13%

The main part of this part of the project that I worked on was the login/logout functionality. First, I created a registration section where i would include the login form and any other user authentication that was necessary. For the login form, I included a way for the user to type in their username and password, and also functionality for if they entered wrong information. Authentication also required non-admin users to be denied from trying to enter an admin section. The user would then get redirected back to the home page. In the base template on the toolbar, along with adding the login/logout buttons, i made it so that only one of the login/logout buttons was displayed depending on if the user was logged in or not. Once the user was logged in, they can navigate to their profile page, or log back out. Similar to in the homework, i included a logout template so when trying to log out, the user would not be directed to the admin login/logout page. I ran into some issues while doing this, but eventually figured it out by creating a new templates/registration directory in a different folder than the rest of our templates were in. Instead they could just simply log back in on the base website. I also included some basic forms and templates for password reset/management. Finally and obviously, I worked on my individual part of the write up.

Write-Up: Daniel Macdonald

Approximate contribution: 14%

At the beginning of the project, we wrote down each piece that needed to be implemented and divided up the work amongst us. Joe took on a couple of extra tasks to help the rest of us after we decided to use Django's user implementation. For this project, I was responsible for creating a form that allowed the user to edit their profile. I started by creating one view. Initially, I was going to follow the implementation that our Django locallibrary homeworks created. However, I decided to change to a model-based view and that I would make one for each group(trackspotters & critics). After making the views, I made the appropriate paths in our URLs file. Then I made a basic form template and the form itself. I ran into a few problems while testing where the incorrect profile was being edited after form submission. While Tommy was helping me resolve this issue, we discovered another issue where I didn't redirect the view to the correct page after the form submission. We managed to resolve both problems with a little help from Joe. I would say I contributed as much as everyone else, except maybe Joe. He always goes above and beyond.

Write-Up: Dongwon Park

Approximate contribution: 13%

For this project, I was responsible to hook pages up to forms, pages including Albums, Songs, Artists and etc if needed. Before working on it, I planned to make my own form classes to implement it. After realizing that Django provides the generic editing view class models, I adjusted the way to use those built-in View classes. So, I made create/update/delete classes for albums, songs and artists. Then, I added url patterns in urls.py so that a user can modify objects using Django form through the url on a browser.

For this time, the whole amount of work didn't take much time because I just used the built-in classes, but it took a while and it was worth to decide which way I should follow, to determine how to implement it efficiently. My decision was to use the Django classes rather than to create custom classes which needs a lot of works and which tends to bring more complexity, to me and also to other team members.

Write-Up: Tommy Do

Approximate contribution: 13%

For this project, I was responsible for handling the authentication/permissions that allow users to only be able to edit their own profile page. I was also responsible for displaying an additional field called "Organization" if the user was a Critic rather than a regular user. I accomplished these two tasks by dynamically rendering the views for the User Profile page based on the authenticated user that is logged in. If a user is authenticated and they are viewing their own profile page, then an "edit" button would be displayed next to their profile. I then linked this edit button with a form that will allow the user to update their profile. To check if an authenticated user is viewing their own profile page, I would compare the id of the authenticated user with the user id associated with the profile page they are viewing. And if the user is a Critic, I would display an additional field in the profile called "Organization".

I had also helped in fixing some issues that occurred when replacing our old User and Critic model with Django's built in User model. An issue was that parts of the profile page were not rendering properly and that was because the variables in the template for User Profile page needed to be updated due to a change in our models.

Write-Up: Aric Huang

Approximate contribution: 12%

For this project, I worked on the review forms. This included making the review links from songs, albums, and user, as well as making the review forms while associating them with their respective user, song, and album. I took it upon myself to try to figure it out, and even with the help of my groupmates, I was unable to complete all the functionality I wanted ontime.

The first steps that I took was to create the forms and attributes associated with them in a forms.py document. After this, I created the generic html templates for deleting and creating the reviews. Thirdly, I attached the url patterns accordingly as well as added links to the respective pages where I thought there would be interaction to reviews. Afterwards, with the help of the tutorial, created the views functions to attempt to pass along attributes to the forms page. This is where I ran into problems.

One major issue I ran into was associating id values, such as song or user, to their reviews. I wanted to not have the user manually input these values, and within the time frame, I was unable to implement this without it crashing. I also had issues with get/post errors where objects were calling gets.