**Implementation**

**Advanced Software Engineering – COMS W4156**

**Team Name:** PurplePoster

**Team Members:** Abhas Bodas (ab3599@columbia.edu)

Jared Pochtar (jrp2181@columbia.edu)

Maria Taku (mat2185@columbia.edu)

Suhan Canaran (sc3055@columbia.edu)

**Due Date:** November 19, 2012

**1. Requirements Change**

Our requested requirements changes were to:

1. Add an administrative interface through which users (that are administrators) can reset the password for other users or can ban other users.
2. Increase Requirements 4.1 (a user can create a profile with preferences and locations) and 4.2 (a user can create watch alert for movies, favorite actors/actresses, and locations) from medium to high priority and implement them.

Change 1:

Based on the Django web framework which we were using, change 1 was fairly simple to implement. Django comes with a built-in administrative interface that provides easy access to all the data in the database as well as tools for managing users that have registered with the site. This interface can be enabled by (1) registering the Admin Page url in the urls.py file and (2) adding "django.contrib.admin" to the INSTALLED\_APPS setting in settings.py. (Specifically, these steps allows access to managing the site's users. To access the data in the databases, the desired classes must be registered in an admin.py file).

This automatic Admin Site comes with prebuilt-in functions for resetting a user's password and for banning users (e.g., various levels of "banning" can be undertaken such as deleting the user, or switching the user to an inactive state, etc.). Thus, based on our decision during the "Requirements" stage to choose the Django framework, this feature could easily be implemented in less than 5 minutes. Our choices during the Design stage weren't relevant to the ease in which this could be implemented since—once again—despite our design, Django allowed this feature to be built quickly. Our choices during the Implementation stage were only relevant in that we chose to use the built-in Django admin site rather than, for example, building an Admin site from scratch (but this decision saved us a lot of time!)

Thus, in hindsight, we would not have done anything different during the requirements, design, or implementation stage in order to build this feature. Our choices not only allowed us to completely fulfill the new requirement and provide an attractive Admin Site interface, but they also allowed us to fulfill this requirement quickly and easily.

Change 2:

Change 2 required our team to set up user accounts with the ability to (1) register accounts, (2) log-in/log-out, (3) save preferences, and (4) receive PurplePoster alerts based on these saved preferences.

The Django web framework provides a pre-built user model. This model can hold fields such as username, password, email address, etc., and is automatically incorporated into the Admin Site mentioned above. We thus decided to use this user model since it could be suitably leveraged to complete change 2.

In order to store the "user preferences" (e.g., such as favorite actors/actresses, movies, purple posters, and locations), a class called "UserPreferences" was added to models.py which held this information. Each instance of this class is tied to a unique user of the user model; in other words, via this class each user can be affiliated with their own user preferences. Django uses the classes in models.py to automatically create database tables (e.g., via ORM), and thus automatically created the UserPreferences and User tables in our MySQL database.

We then simply needed to create webpages allowing a user to create an account, log-in/log-out, and create preferences. This also was easily done by editing our views.py and templates. Django had a few pre-built templates, such as the UserCreationForm for allowing new users to register accounts, which were suitable for our purposes so we reused them.

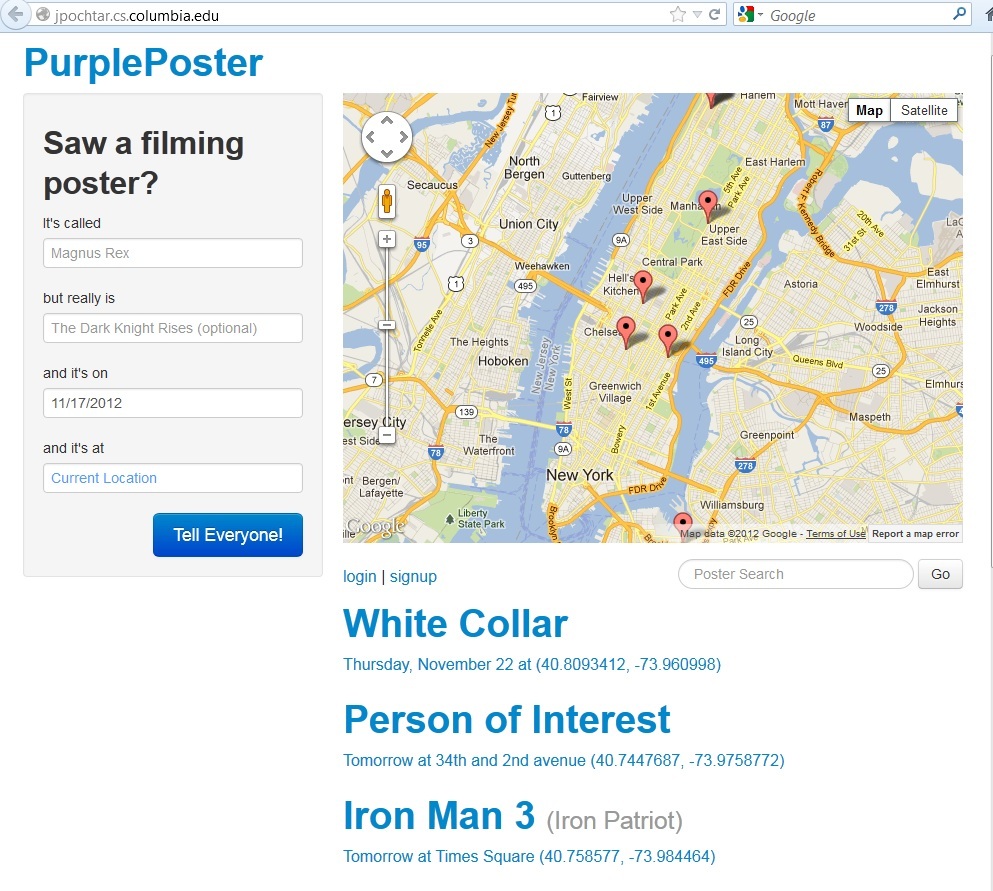
Lastly, the alerts system had to be created. This was done by writing a script that would run each day, determine which upcoming PurplePosters matched a user's preferences (e.g., that were filming the next day), and then emailing webpage links to these matching PurplePosters to the user.

Once again, our decision during the Requirements stage to use Django greatly increased the ease in which these new features could be implemented. We were able to re-use and modify a lot of Django's pre-built features such as user models and user-related webpage templates (e.g., such as the UserCreationForm). The alert system was the only system that really had to be built completely from scratch. Although Django does have a component for emailing users, this component was not robust enough for our purposes, so we chose to make our own system. Our decisions during the Design stage were sufficient since we had included user classes in our Class Model (even though this wasn't originally a high priority task), so our website was already well-designed for adding user functionality.

In hindsight, in this case, we would not have changed anything during the Requirements or Design stage in order to make this easier. During the Implementation stage, however, it may have been worthwhile to try to "unify" the user's account on the website a bit better. For example, the website is currently set up such that the user only enters a username and password upon registration. The registered-user must then go to a preferences page in order to enter preferences and an email address. It may have made more sense to "unify" these by allowing a user to register, enter an email address, and/or enter preferences all at once rather than doing it in a segmented, two-step processes. This two-step process resulted from our use of Django's UserCreationForm, which only allows users to register a username and password (and thus a second step was required to allow a user to enter an email address, etc.). Thus, if we had instead decided to make our own templates from scratch, we could have built a one-step registration process that allows a user to complete all fields at once.

However, in the end, the website was still fully functioned and in our opinion the two-step process is not much of a burden on the user (and perhaps the user even appreciates the fact that they can register quickly and don't need to waste time entering a lot of preference information if they don't want to). Additionally, re-using Django's UserCreationForm allowed us to get the site running quickly, which was very important since this was a last-minute change that we had to implement! Thus, in hindsight, even during the implementation stage we would not have done anything differently.

**2. Demo Screenshots**

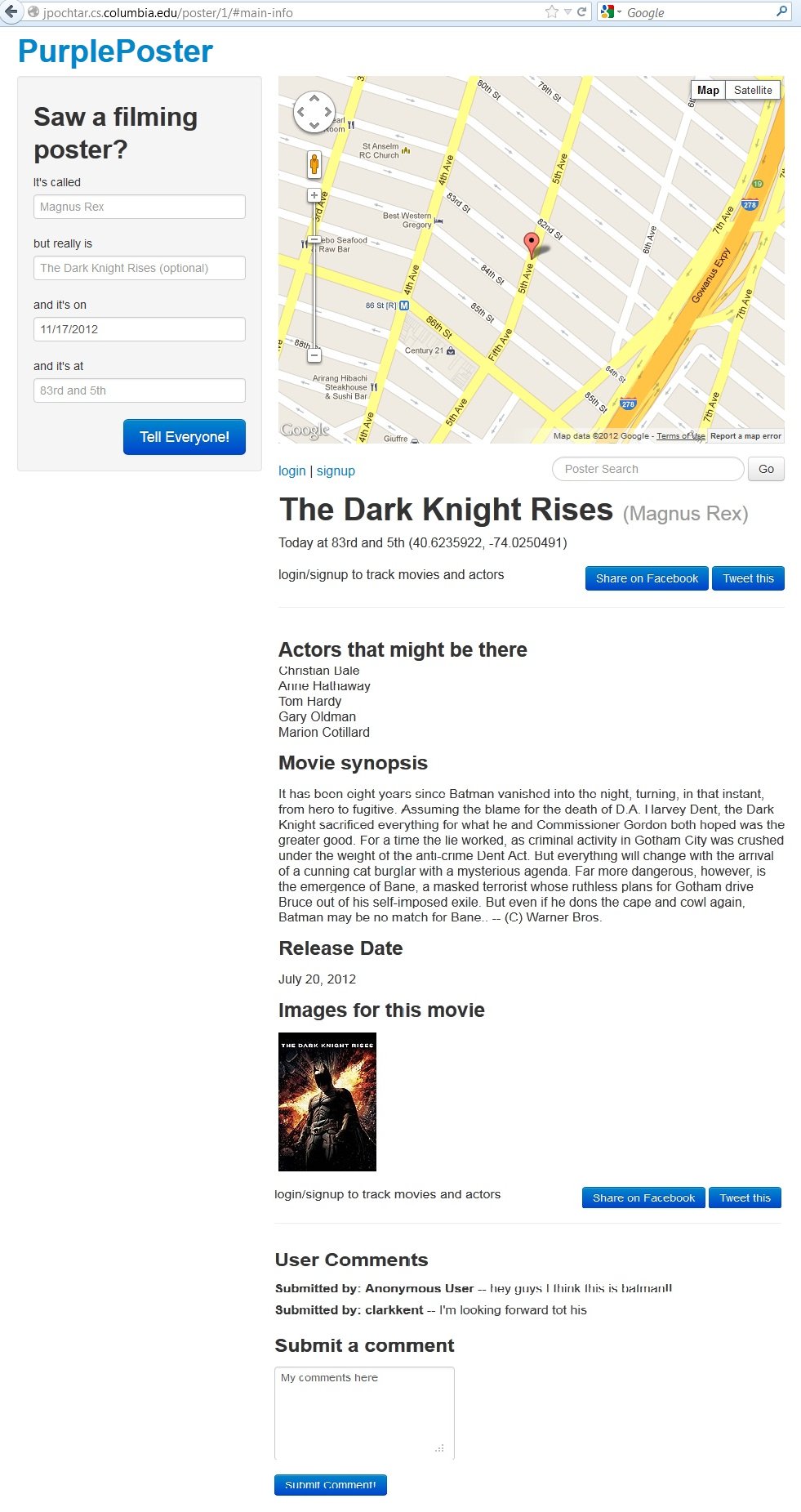


*Screenshot 1: Homepage*

*Screenshot 1* shows the PurplePoster homepage. From this page, a user can enter filming posters which they have just seen (left side of the page), see a map view of all upcoming filming locations (top right), and see a list-view of all upcoming filming locations (bottom right). As shown by the URL, a server was setup/deployed to host the website, and this website currently runs live.

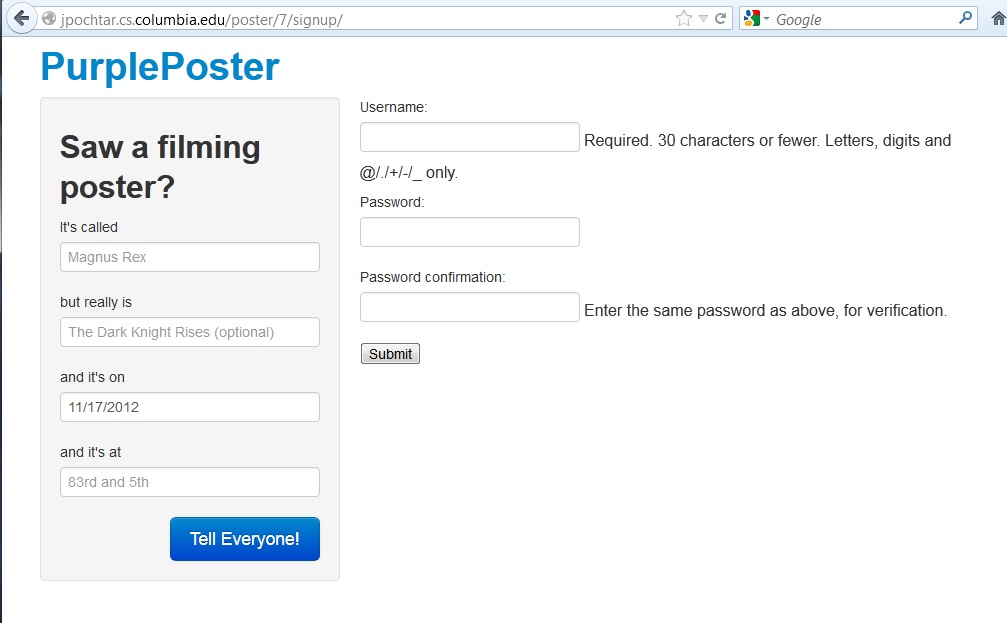
When entering a filming poster, a user can enter the Alias (e.g., a possible fake name printed on the poster so passing pedestrians don't know what is actually filming), an optional real movie name, the date of the filming, and the location. Since this website was optimized for mobile use, the date defaults to "today" (e.g., since filming usually occurs on the same day you discover a poster) and the location defaults to your current location (automatically determined by your cell phone/computer if you choose to share your location). The movie name is used by the website to query RottenTomatoes.com and pull up additional information about the film. If the user does not enter a real movie name, then the website will query the alias in its attempt to locate the movie on RottenTomatoes.com.

Note that from this homepage a user may also login, register a new user account, or search the purple posters. Searching the purple posters can allow a user to not only locate future filming locations, but also past filming locations (e.g., in case the user wants to visit famous sites where movies were once filmed).



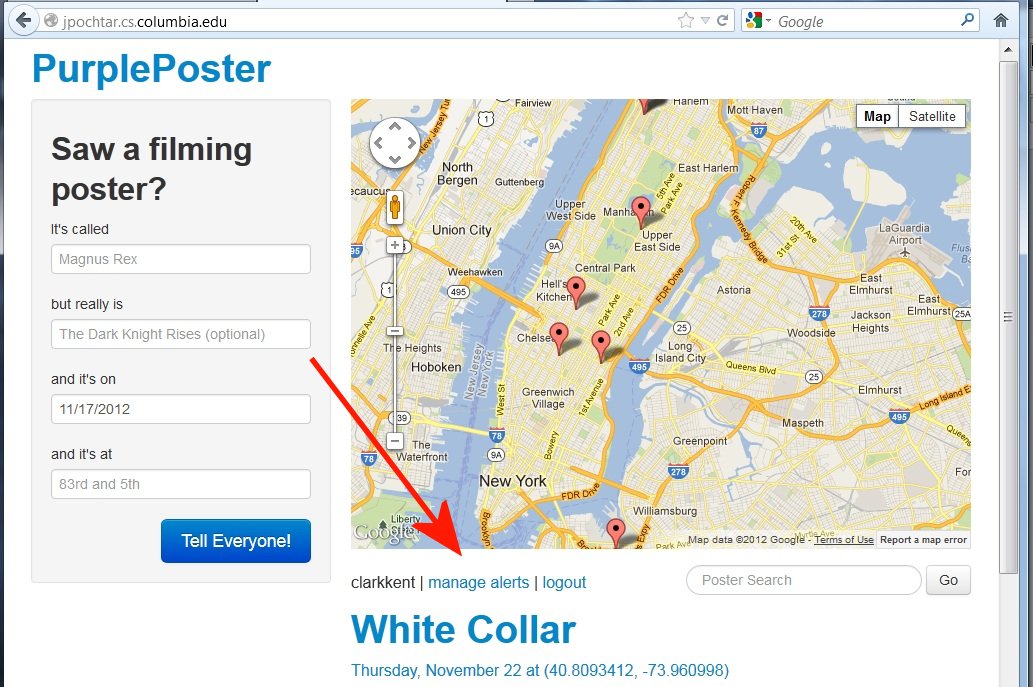
*Screenshot 2: A Purple Poster's individual page*

*Screenshot 2* shows a purple poster's individual page. For example, a page like this can be reached by choosing a purple poster from the listings of posters on the homepage. This page not only shows information which the user entered—such as the alias, movie name, date, and location—but it also shows any related movie information which was pulled from RottenTomatoes.com. For example, in the screenshot depicted, information such as actors, a movie synopsis, release date, and images have all been pulled and displayed from RottenTomatoes.com. Lastly, this page also allows a user to add comments about the purple poster. If the user was logged in while posting the comment, the username will be displayed beside the comment. However, if the user was not logged in, the credit for the comment is simply given to an "Anonymous User."

**

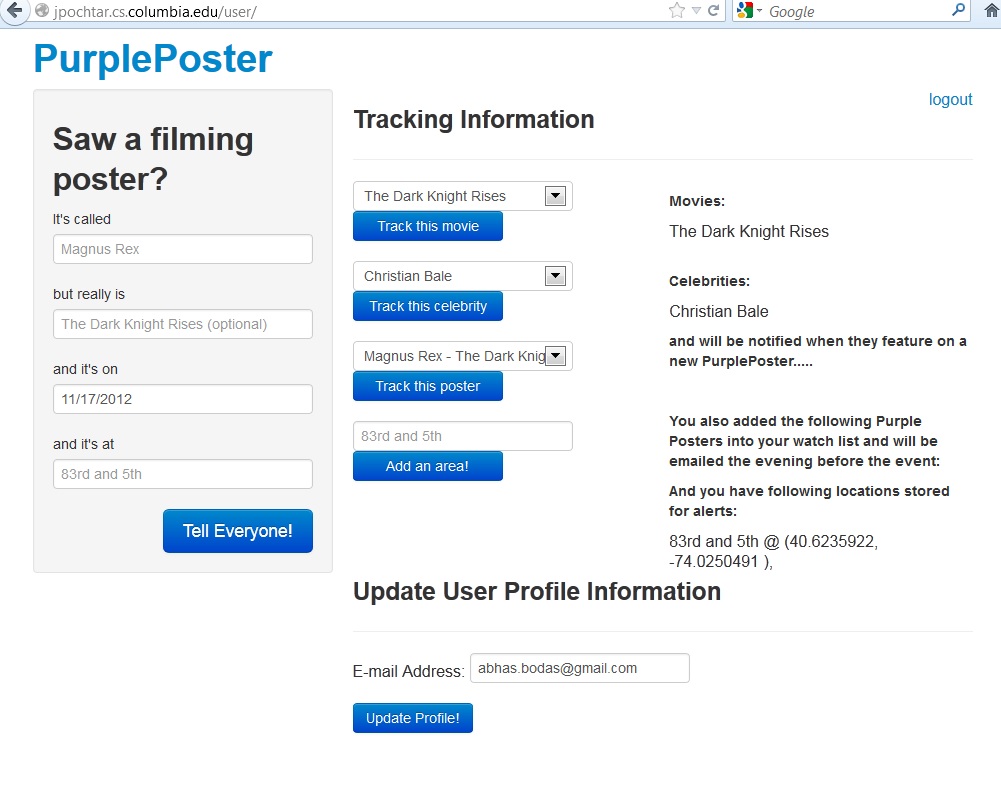
*Screenshot 3: User Registration Page*

*Screenshot 3* shows the new user registration page. For example, this page can be reached by clicking on any of the "signup" links on the website. As illustrated by this screenshot, a user is only required to enter a username and password to create an account, and thus can quickly and easily create their new account.

**

*Screenshot 4: Manage Alerts is available to a logged-in user*

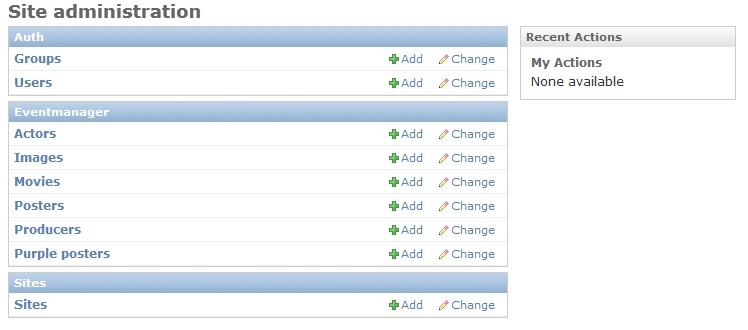
Once a user has logged into PurplePosters, they will now have the option to create and modify user preferences. For example, *Screenshot 4* shows that the homepage and other relevant pages are modified to include the username and a "Manage Alerts" link once the user has logged-in.

**

*Screenshot 5: The Manage Alerts Page*

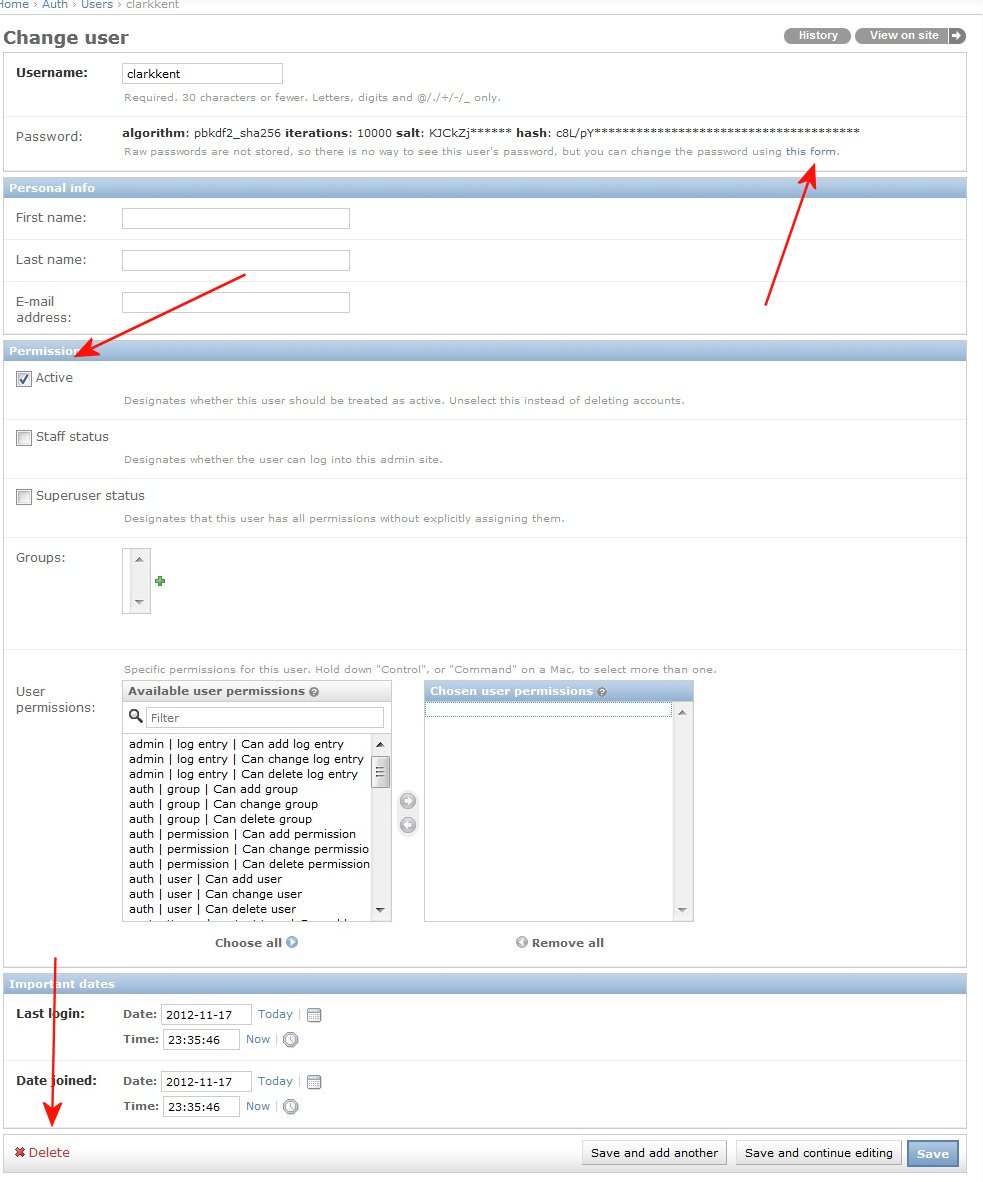
*Screenshot 5* shows the Manage Alerts webpage. From this page a user can register an email address and indicate preferences for purple poster alerts. For example, as shown in *Screenshot 5,* the user has indicated they like the movie "The Dark Knight Rises," the actor "Christian Bale," and the location "83rd and 5th"

The purple posters website has been programmed with a script that runs once a day, locates purple posters in the near future that match a user's preferences, and then emails the user about these matching purple posters. This allows a user to easily keep track of filming locations which match their interests.



*Screenshot 6: Admin Site*

*Screenshot 6* shows the PurplePoster Admin Site which allows administrators to manage the data and users of the website. For example, via this webpage an administrator can add, edit, or delete any of purple poster-related data in the website, such as movies, actors, purple posters, etc. Via this Admin Site, administrators can also manage the users of the website. For example *Screenshot 7* shows the User Modification Admin Page. From this page, various functions such as resetting a user's password or banning a user can be performed (e.g., various levels of "banning" can be undertaken such as deleting the user, or switching the user to an inactive state, etc.).

**

*Screenshot 7: User Modification Admin Page*

**3. Traceability Matrix**

Please refer to the Traceability Matrix in Appendix A.

Paragraph/Offset Columns: Note that for the paragraph and offset values, the SCORE competition only says that these refer to "document where this requirement is located" and does not give a more exact description of which document they are referring to. Thus, for now we have filled in this information with the paragraphs/offsets from the Requirements document we previously submitted in class. There is therefore no value for the "Admin Site" requirement, since this requirement was added later on after the Requirements document was submitted, and thus does not currently appear in the Requirements.

These paragraphs/offsets will be updated later on, if a different document is eventually requested from us to describe our requirements.

Design Columns: All of the marks in the Design columns are based on the diagrams submitted in our “Design” assignment. Due to changes during implementation and the customer’s last minute changes (as described in Section 1 of this report), the submitted diagrams are currently not up-to-date. We have edited the diagrams to accurately portray our current design in our own records. However, since those are not the ones which were submitted and which the TA’s etc. have access to, we had the Traceability Matrix instead reflect the submitted records – this means the Design columns may be slightly inaccurate at times.

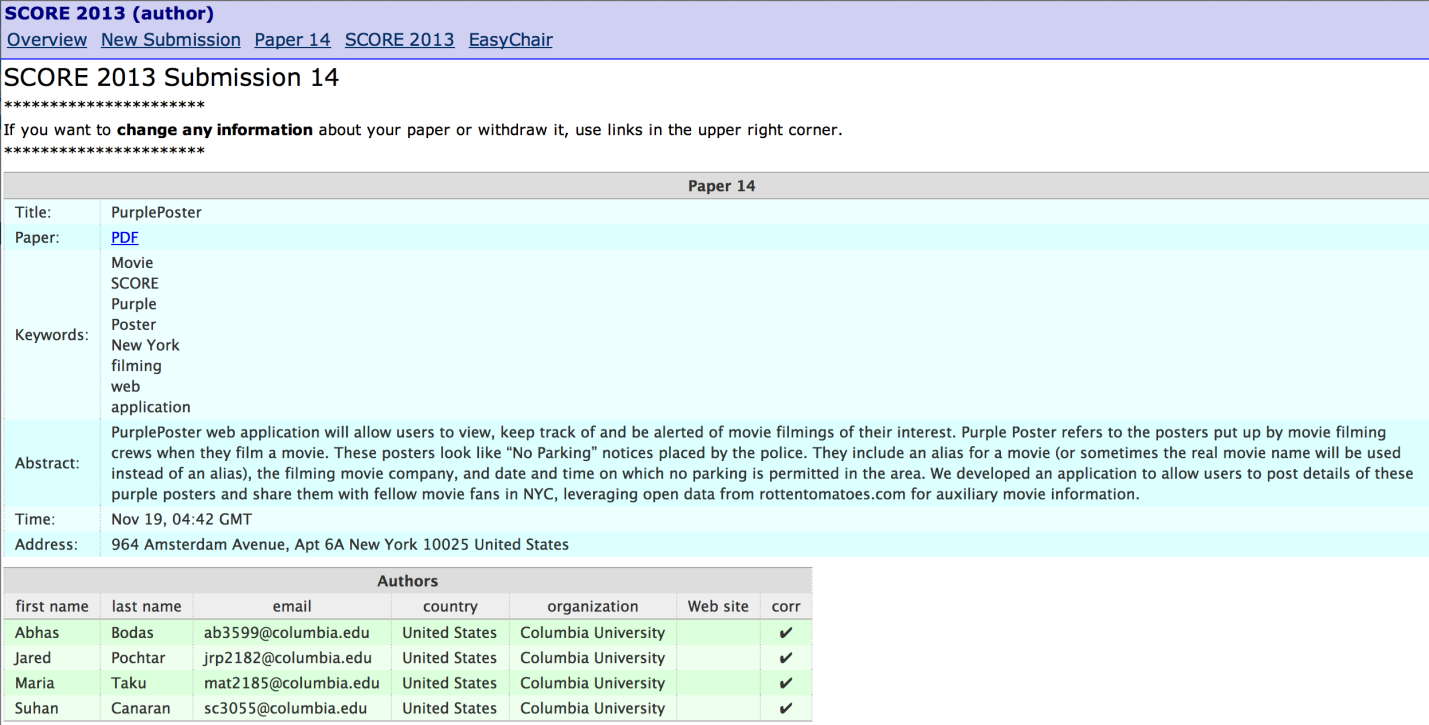
If/when we are asked to submit new, updated diagrams, then we will also correct the Traceability Matrix.

Code/Implementation Columns: For these columns, the SCORE instructions ask that “elements of the source code (e.g., classes, methods).” However, the SCORE instructions did not give clear instructions on how to choose which classes/methods etc., to include. This was especially a little tricky in our Django implementation, since it includes a lot of templates (html), settings files, definitions, and other files/code that do not directly correspond to traditional classes or methods. Thus, when choosing the columns for the code/implantation, we made judgment calls in splitting up our files into the most important/valuable definitions, templates, views, models, etc., that captured our application, and then translated these choices into our columns.

Non-Implemented Features: A couple of the features that originally appeared in our design etc., were trivial priority and never ended up being implemented. Thus, in these cases, you may find some columns without a corresponding “1” in them. For example, one of our diagrams from the Design stage was “Sequence Diagram 1: Submitting an Image of a Purple Poster.” However, image submission was trivial priority feature and thus has not yet been implemented. So this column of our Design section of the Traceability Matrix only has zeros.

**4. SCORE Competition Registration**

The confirmation email for our SCORE Competition registration is below**:**

****

**5. Component Services**

A wide variety of component services were used in creating the PurplePosters webpage. Some of these services are detailed below.

ORM – in order to manage our database, we used Django's built-in ORM functionality. Specifically, you can instruct Django to automatically translate desired classes into SQL tables. During this translation, the class becomes a table, the attributes become the table's columns, and each instance of the class becomes a row within the table. You can also, for example, specify relationships such as "many-to-many" between classes, and Django will also automatically create the table(s) necessary to fulfill this relationship.

This service component greatly helped the project since it allowed us to concentrate on the pure design and the logic of the website itself, without needing to worry about the underlying structure of the database housing the site's data.

Authentication - Another component service which we used was Django's built-in user authentication component service. Authentication support is bundled as a Django application in "django.contrib.auth." This application provides services such as hashed passwords (Django doesn't store raw passwords), methods for validating user/password combinations, and even prebuilt login page templates. Once again, this greatly helped our website since it allowed us to quickly and easily deploy standard functions commonly re-used by webpages, while giving us more time to concentrate on the unique characteristics of the PurplePoster's page itself.

Authorization – Django also provides an authorizations component services for easily checking whether a user is authorized to view certain pages. In order to require a particular page/view to require a user login, the keyword "@login\_required" needs to be placed before that view's definition. Django then automatically checks that a user is logged-in before executing the page. This @login\_required keyword was thus placed before functions such as updating user profiles and updating preferences, etc. This component service was not only valuable in that it saved us time, but it also provided a additional layer of security to our website.

User Registration – Django's user related component services were also used in setting up the user registration pages. For example, Django provides user models that includes user data (e.g., username, password, email, first name, last name, etc.) and various user-related functions (e.g., get\_username, is\_authenticated, set\_password, check\_password, etc.). Django also provides pre-made user registration forms that could be easily utilized to setup our own PurplePoster user registration pages.

Admin Site­ – One of Django's most useful component services is its Administrative Site. As described in more detail in Section 1, this Administrative Site is very easy to setup and can be fully calibrated within a few minutes. Via this site, administrators can easily view all of the information in the website's databases, as well as add, modify, or delete any of the data.

The Administrative website also provides an easy interface for managing the users. For example, you can add users, reset user passwords, delete users, mark users as inactive, and grant users various permissions. This was incredibly helpful since it allowed us to quickly fulfill one of our last-minute requirement changes.

**6. Controversies**

None – everyone on the team agrees 100% with the design.

**Appendix A: Traceability Matrix**