|  |
| --- |
| Architecture and Design |
| MyNewMedia |
| Zachary Merrill, Stephanie Koesters, and John Mark De Los Reyes |

**Revision/Change Record**

|  |  |  |  |
| --- | --- | --- | --- |
| *Revision* | *Date* | *Revision/Change Description* | *Pages Affected* |
| 0 | 5 April 2013 | Outline Created | All |
| 1 | 6 April 2013 | Initial Draft | All |
| 2 | 18 April 2013 | Spell, Grammar Revisions | All |
|  |  |  |  |

1. **Introduction………………………………………………………………………………..5**

**1.1. Purpose….………………………………………………………………………………..5**

**2.2. Scope…………………………………………………………………….………………..5**

**2.3. References……………………………………………………………..………………..5**

1. **Overall Architecture………………………………………………………….……..5-7**

**2.1 Django…………………………………………………………………………….…..…..5**

**2.2 Justification……………………………………………………………..….………...5-6**

**2.3. Model…………………………………………………………………………..…….…..6**

**2.4. Database……………………………………………………………………..…………..6**

**2.5. View……………………………………………………………………………..…………7**

**2.6. Template………………………………………………………………………..………..7**

1. **Application Decomposition………………………………………………..……7-17**

**3.1. Channel…………………………………………………………………………..….…7-9**

**3.1.1. Model………………………………………………………………………....7**

**3.1.1.1. ChannelType…………………………………………………..7-8**

**3.1.1.2. Channel……………………….…………………………………..8**

**3.1.1.3. Link………………………………………………………….……..8**

**3.1.1.4. FeedItem…………………………………………………….…8-9**

**3.1.1.5. FeedTracker………………………………………………………9**

**3.1.2. View……………………………………………………….…………………..9**

**3.1.3. User Interface……………………………………………………….……..10**

**3.2. Subscription………………………………………………………………………..11-12**

**3.2.1 Model………………………………………………………………….………11**

**3.2.1.1. Subscription…………………………………………….………11**

**3.2.2 View………………………………………………………………….………..11**

**3.2.1. User Interface……………………………………………..…….……11-12**

**3.3. Profiles…………………………………………………………………………..….12-14**

**3.3.1. Model……………………………………………………………..…....12-13**

**3.3.1.1. UserProfile……………………………………………..….12-13**

**3.3.1.2. UserPreferences…………………………………….……….13**

**3.3.2. View………………………………………………………………..……….13**

**3.3.3. User Interface……………………………………………………….13-14**

**3.4. Core………………………………………………………………………………..14-15**

**3.4.1. Model………………………………………………………………..…….14**

**3.4.2. View………………………………………………………………….…….14**

**3.4.3. User Interface…………………………………………………..……….15**

**3.5. Tag……………………………………………………………………………………..16**

**3.5.1. Model……………………………………………………………………...16**

**3.5.1.1. Tag……………………………………………………………….16**

**3.5.2. View……………………………………………………………….……….16**

**3.5.3. User Interface……………………………………………..…..……….16**

**3.6. Search…………………………………………………………………….…..………17**

**3.6.1. Model……………………………………………………………..…..….16**

**3.6.2. View………………………………………………………………………..16**

**3.6.3. User Interface…………………………………………………..….16-17**

**3.7. Data Dependencies………………………………………………..……..……….18**

1. **Third Party Plugins……………………………………………………………..…18-19**
   1. **Django-hitcount………………………………………………………..………18**
   2. **Feedparser………………………………………………………………..……..18**
   3. **Unipath……………………………………………………………………..…….18**
   4. **Django-messages……………………………………………………….….…..19**
   5. **Django-taggit…………………………………………………………….……..19**
   6. **Django-registration…………………………………………………….……..19**
2. **Introduction**
   1. **Purpose**

This document is to present how the MyNewMedia works to our users. Also, this document contains information on what can be expected from the MyNewMedia, such as system functions and the Human Computer Interaction.

* 1. **Scope**

The scope of this document is to describe how MyNewMedia works. MyNewMedia gives podcasters the opportunity to combine their media in one website.

* 1. **References**

The following documents were used in the development of this System Requirements

Specification:

* Fall 2007 System Requirements Specification Example, Dr. Mansour Zand, April 2013
* Fall 2012 System Requirements Specification Example, Dr. Mansour Zand, April 2013
* IEEE Std 1233, 1998 Edition – Developing System Requirements Specifications

1. **Overall Architecture**
   1. **Django**

The MyNewMedia team used a free and open source web application framework called Django. Django is written in Python, which follows the MVC style architecture. Django's goal is to make it easy for people to make a website as fast and as easily navigated as possible. One of the reasons MyNewMedia team is using Django (rather than ASP.NET or something similar) is because of the limited time the team had to develop the website. Django also provides an automatic admin interface so developers don't have to write one for themselves. Moreover, Django also supports multi-language applications so developers can write code in JavaScript and other languages to make websites easy to use.

* 1. **Justification**

The MyNewMedia site has a very clearly defined architecture thanks to the Django framework. Django was developed as a slight modification to the standard Model-View-Controller (MVC) style architecture. The framework refers to the Controller layer as the ‘View’ and the View layer as the ‘Template;’ therefore, the name becomes the Model-View-Template (MVT) architecture. The purpose behind using the MVT architecture is to decouple all layers of the application and allow developers to work on different programming aspects – from the database to the front end design – at the same time. This enables teams to work rapidly and effectively on any application.

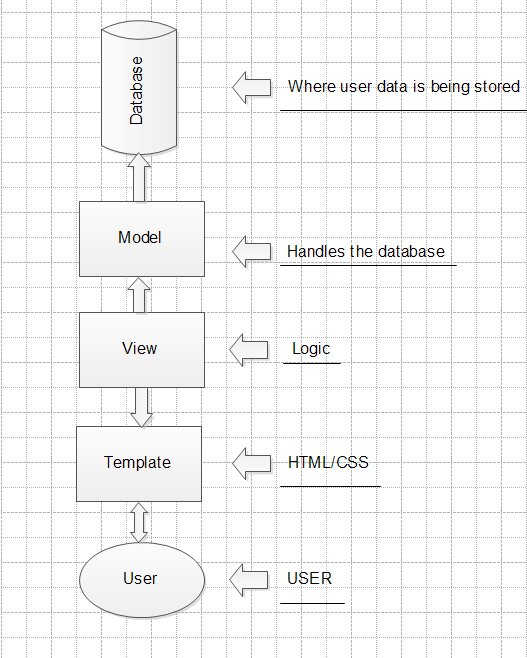
There are two keys to a successful Django web application, and any application that uses MVC or MVT architecture: code reusability and separation of concerns. The code for any Django project is organized into “applications” that can be bundled and reused anywhere in the application, or exported and used in different projects altogether. Because many web applications require very similar features, the ability to reuse code becomes essential. The separation of each of the layers not only increases the productivity of developers by allowing work to occur simultaneously, they also enable the portability of the code. By making sure that the template layer doesn’t rely on a certain database schema, a team could easily make changes to the database without affecting the other layers. It makes the application much more maintainable.

**2.2 Model**

The model layer essentially acts as a representation of the database and all related APIs. Within the model are defined entities (Python functions in Django’s case) that correspond to tables with in the database. Django will write the SQL calls needed to create the database tables so ensure they are identical to the model definition. In addition to building the schema, Django will provide an API that allows our application the hooks it needs to communicate with the other layers. Many MVC architectures use a passive model, where the view is responsible for updating the model. Ours will use the active version, where the model contains the update calls and the view simply accesses them.

* 1. **Database**

We are using SQLite to read and write from the database. Since the MyNewMedia team is building the website in their local computers, SQLite was a good choice because of its embedded database for local storage. The types of information stored in our database are user names, passwords, the channels they added, profile information, subscriptions, and so on. With that, information will also be stored differently for admins since they can get access to certain information.

**2.4 View** Figure 2.4A

In most MVC architectures, the view layer is what is shown to the user; the HTML or CSS. Django’s view layer is much more akin to the controller of the traditional MVC. Most of the logic is defined in the view layer. It acts as the middleware between the model and the template (as shown in Figure 2.4A). As the user manipulates the data presented in the view, the controller modifies the model. Then it notifies the template layer that the model has changed and updates the template with the requested data.

* 1. **Template**

The template is everything the user sees. This includes all HTML or CSS files, any images, any audio, text, and anything that can be queried. The Django framework refers to this as the template instead of the view because of how extensible it is. Using HTML tags, we can dynamically generate our pages based on the active view.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | Channel  3.1.1. | Subscription  3.2.1. | Profiles  3.3.1. | Core  3.4.1. | Tag  3.5.1. | N/A  3.6.1. |
| **View** | Channel  3.1.2. | Subscription  3.2.2. | Profiles  3.3.2. | Core  3.4.2. | N/A  3.5.2. | Search  3.6.2. |
| **Template** | Channel  3.1.3. | Subscription  3.2.3. | Profiles  3.3.3. | Core  3.4.3. | Tag  3.5.3. | Search  3.6.3. |

**3. Application Decomposition**

**3.1. Channel**

**3.1.1. Model**

The model layer has five different classes: ChannelType, Channel, Link, FeedItem, and FeedTracker.

**3.1.1.1. ChannelType**

The Channel Type class responsibility is to let admins organize channels by type of media, such as music, videos, pictures, etc. Each type of media exits on this database once, and more can be added or deleted by the administrators.

|  |  |
| --- | --- |
| **ChannelType Methods and Properties** | |
| type | It is an arbitrary string entered by the administrator where the types of media can be selected later. |

**3.1.1.2.Channel**

The Channel class allows a user to add channels to MyNewMedia.

|  |  |
| --- | --- |
| **Channel Methods and Properties** | |
| title | [Required field] Channel type stored as an arbitrary string. |
| url\_ext | [Required field] Arbitrary string that has to be unique for browser navigation. |
| owner | [Required field] Foreign key to the user who created the channel. |
| description | [Non-required field] Field for description of the channel. |
| type | [Required field] Foreign key to ChannelType table. |
| Language | [Non-required field] Language type stored as an arbitrary string. |
| Feed | [Non-required field] URL the channel will pull the RSS XML document from. |
| Image | [Non-required field] Picture stored and displayed with all instance of a channel. |
| Tags | [Non-required field] Set of labels stored describing the channel. |

**3.1.1.3. Link**

This provides a link model to users of the channels that have been made.

|  |  |
| --- | --- |
| **Link Methods and Properties** | |
| Title | Title of the link |
| channel | Foreign key to the channel the link belongs to. |
| description | Description of media link. |
| url | Clickable link pointing to the media. |

**3.1.1.4. FeedItem**

This provides individual record of an RSS feed. It contains all items from an RSS field and uses to create a user history.

|  |  |
| --- | --- |
| **FeedItem Methods and Properties** | |
| title | Title of the RSS feed item. |
| channel | Foreign key to channel the item belongs to. |
| Link | Links the user to the RSS. |
| Itemcount | The number of RSS feed(s) within the channel. Note, not unique across channels. |

**3.1.1.5. FeedTracker**

This tracks the number of RSS feeds in the user watch history.

|  |  |
| --- | --- |
| **FeedItem Methods and Properties** | |
| Item | Foreign key to the feed item that had been viewed. |
| User | Foreign key to the user who viewed the RSS feed channel. |
| Channel | Foreign key to the channel the item belongs to. |

**3.1.2 View**

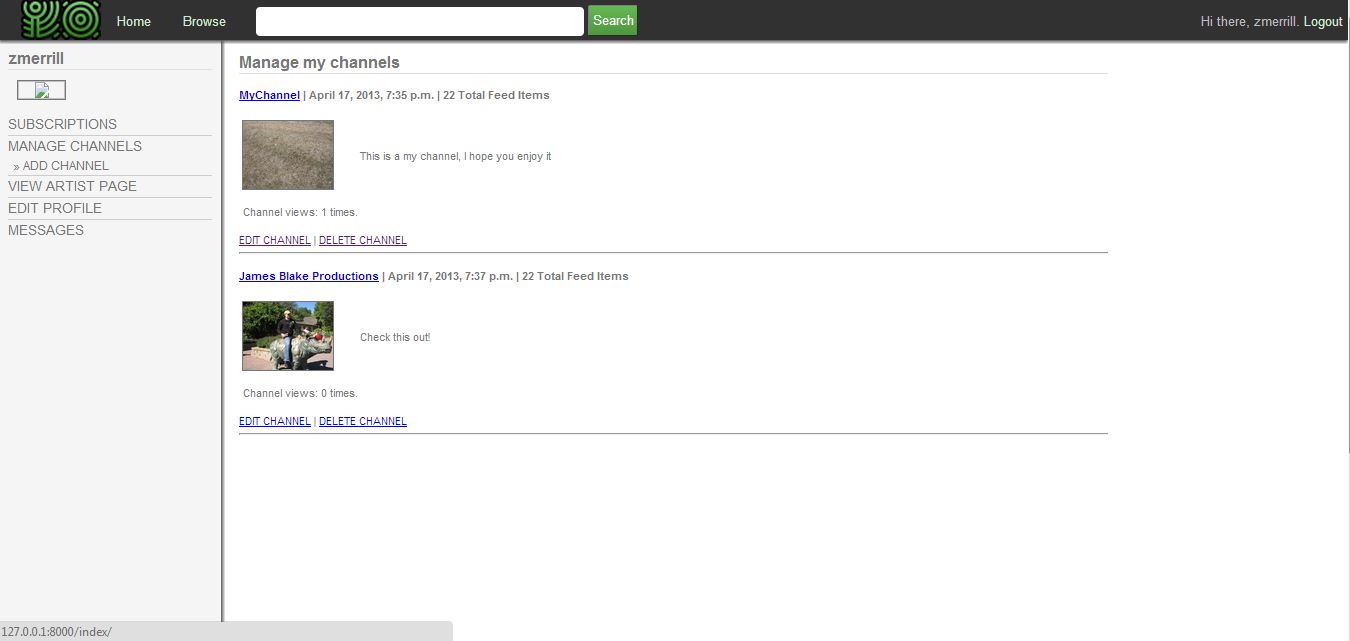
This handles the web response of what is being displayed on the user’s channel page.

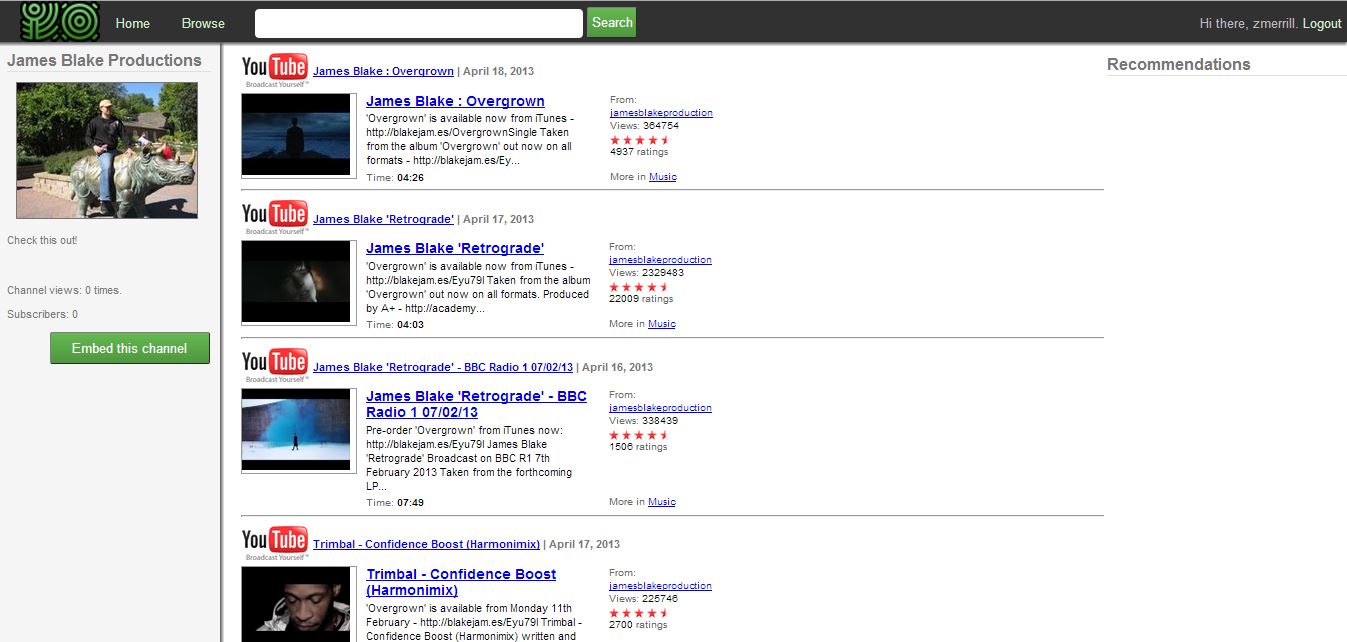
|  |  |
| --- | --- |
| **View Methods and Properties** | |
| Channelhome() | Displays all content and information about a channel on the user’s page. |
| MarkAsRead() | Adds the passed feed item to a user’s history and returns a response only by default. Since it’s called with Ajax, nothing will be rendered. |
| Pull\_Feed() | Uses the feedparser package to pull an RSS feed XML for use. |
| Browsechannel() | Creates a dictionary organized by type for use in the browse page. The user must populate by adding channels in order for this method to work. |
| ajaxFeedPull() | Allows feed to be pulled from where it came from and gets updated. |
| Artisthome() | Returns information about the user as well as all channels that belong to the user. |
| Add() | This allows users to add a channel if there are no errors in the user’s inputs. |
| Edit() | This allows users to edit each channel that has already been created in the past. |
| Addlink() | This allows users to add a new link to the channel home page. |

**3.1.2 User Interface**

This interface shows the user's channel page (Figure 3.1.2A). If the user has added any channels, it will use the Channelhome function to display all content and information about a channel. If the channel has been viewed, it will use the MarkAsRead function to display how many views the channel has had. If the user decides to make changes to a channel, the add function or the edit function will be called to let the user do so. If the user clicks on the channel’s title, the pull\_feed function will be called to output the media (Figure 3.1.2B).

Figure 3.1.2A



 Figure 3.1.2B

**3.2. Subscription**

**3.2.1. Model**

This model layer has one class called Subscription.

**3.2.1.1. Subscription**

The Subscription class allows users to subscribe and unsubscribe from other channel.

|  |  |
| --- | --- |
| **Subscription Methods and Properties** | |
| User | Foreign key to the authenticated user. |
| Channel | Foreign key relationship to the channel. |
| Unread | Number of unread posts. |

**3.2.2 View**

This handles the web response so users can subscribe and unsubscribe from another user's channel(s).

|  |  |
| --- | --- |
| **View Methods and Properties** | |
| Subscribe() | Adds a subscription under the logged in user for the passed extension. |
| Unsubscribe() | If the user is subscribed to another user’s channel, this will remove a subscription. |

**3.2.1. User Interface**

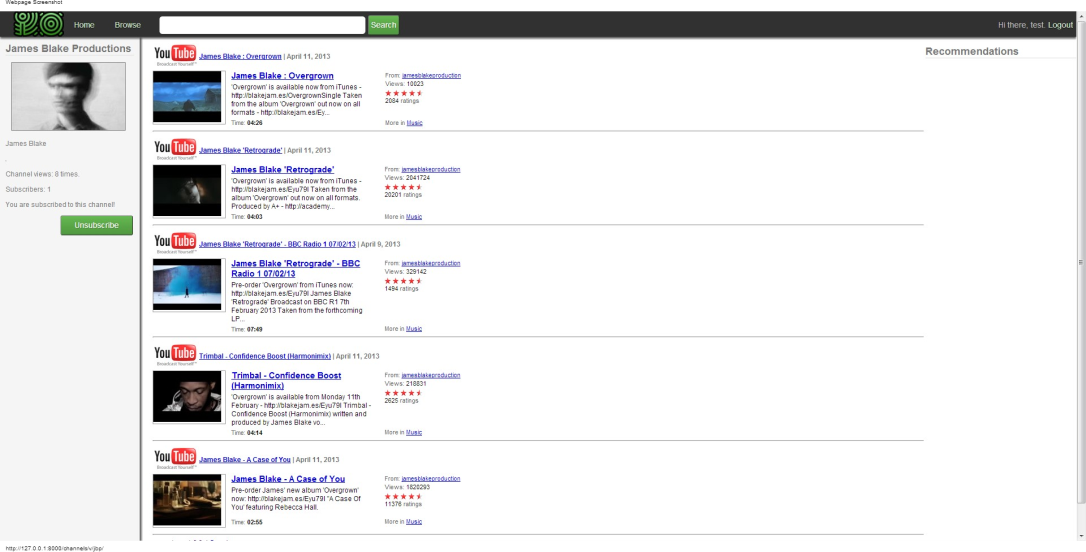
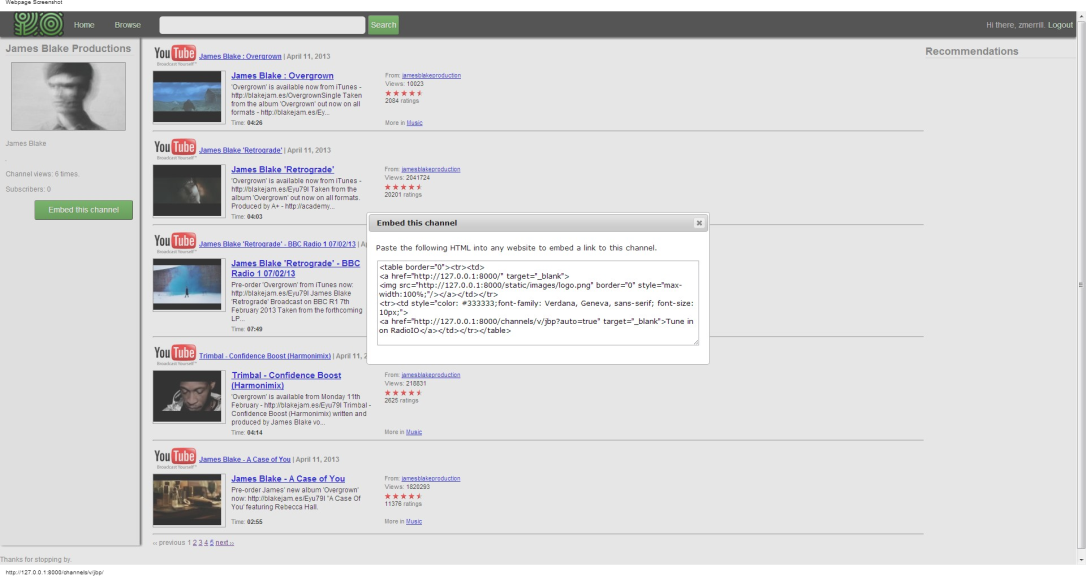
This interface shows a green button so users can subscribe and unsubscribe to other user’s channel (Figure 3.2.1A). Depending whether the user is subscribed or unsubscribed to another user’s channel, it will either choose the “subscribe” or the “unsubscribe” method in the view model once the user clicks on the “subscribe” or “unsubscribe” button. If the user of the channel decides that he/she wants to embed his channel, he/she can do so by clicking the “Embed this channel” button and copying the link. If a different user decides to click on the embedded link, the user will automatically go to the owner of the original link and have the “subscribe” function to automatically subscribe the user (Figure 3.2.1B).

Figure 3.2.1A

Figure 3.2.1B

**3.3. Profiles**

**3.3.1. Model**

The Profiles’ model layer has two classes: UserProfile and UserPreferences.

**3.3.1.1. UserProfile**

This gives users the ability to personalize a user profile.

|  |  |
| --- | --- |
| **UserProfile Methods and Properties** | |
| Owner | [Required field] Foreign key to the user that is unique who created the channel. |
| Bios | [Non-required field] Where users are able to add a description of themselves. |
| Location | [Non-required field] Where users are able to add where they live. |
| Homepage | [Non-required field] A link so other users can access to the user’s homepage. |
| Birthday | [Non-required field] Where users can add their birthday. |
| Occupation | [Non-required field] Where users can add their occupation. |
| Name | [Non-required field] Where users can add their real name. |
| Avatar | [Non-required field] Where users can add a profile picture. |
| Page\_views | Number of page views the user gets. |

**3.3.1.2. UserPreferences**

Gives users the ability to add their preferences of what kind of media they like.

|  |  |
| --- | --- |
| **UserPreferences Methods and Properties** | |
| Profile | [Required field] Foreign key to the UserProfile class. |
| Type | [Non-required field] Foreign key to the ChannelType class of the type of media. |
| Links\_per\_page | The number of links with a maximum of five per webpage. |

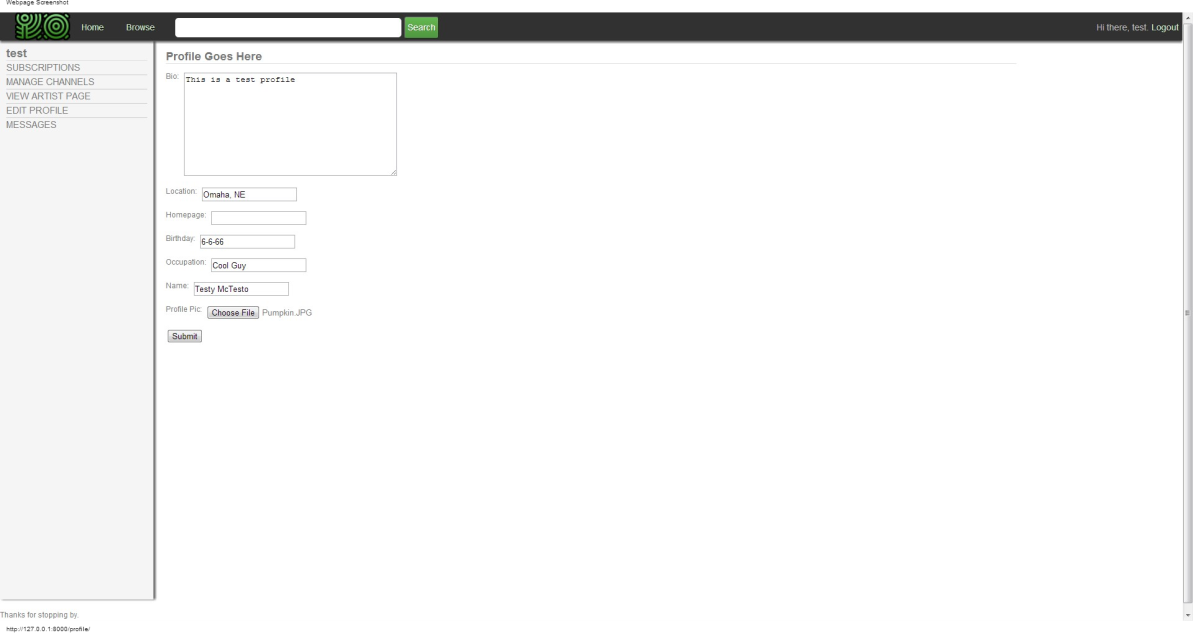
**3.3.2. View**

This handles the web request and web response so users can make a user profile about themselves.

|  |  |
| --- | --- |
| **View Methods and Properties** | |
| Profile() | If a user wants to create a profile about themselves, it redirects the user to the profile editor section of the dashboard. There the user can create and edit a profile about themselves. |

**3.3.3. User Interface**

Once the profile function is called, this interface will show up. This screen shows how a user can edit their profile. (Figure 3.3.3A)

(Figure 3.3.3A)

**3.4. Core**

**3.4.1. Model**

The Core model layer has one class called TimeStampedModel.

|  |  |
| --- | --- |
| **User Preferences Methods and Properties** | |
| TimeStampedModel | Base class that all model classes inherit from. It ensures that all models have a created and modified date fields for tracking. |

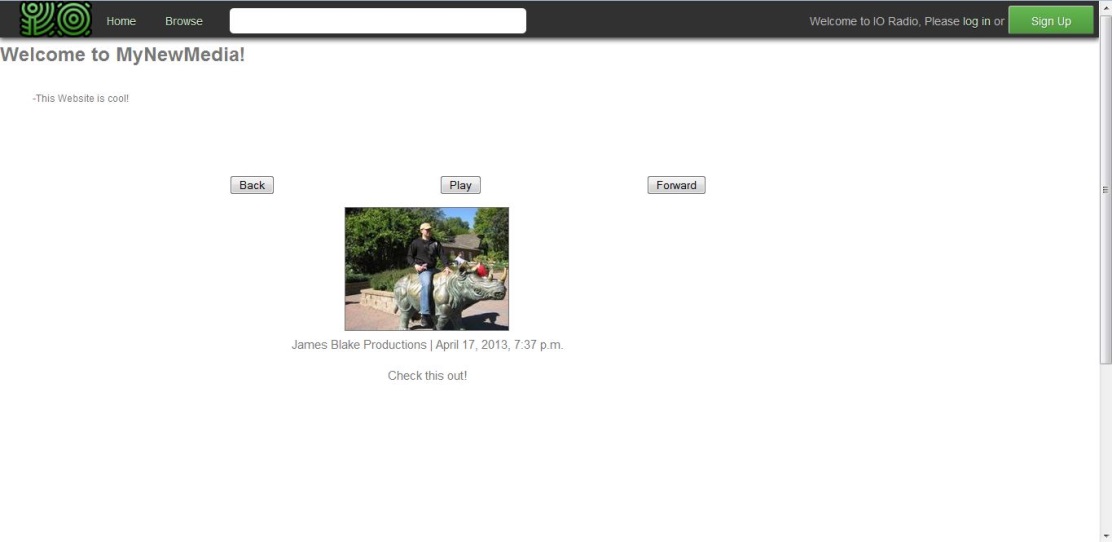
**3.4.2. View**

This handles the web request and web response so users can make user profiles about themselves.

|  |  |
| --- | --- |
| **View Methods and Properties** | |
| GetImage() | Finds image from dynamic media folder. |
| Index() | This is the main index view for the website. If the user is logged in, it will redirect the user to the splash page. Otherwise, it will redirect the user to the subscription manager of the dashboard specific to the logged in user. |
| Mychannels() | Redirects to the channel manager section of the dashboard. It returns most of the same information as the subscription page above. |
| Mymessages() | Redirects to the messaging section of the dashboard. |

**3.4.3. User Interface**

It will call the index() function so the user will either go to the main index view for the website or it will redirect the user to the splash page if the user is not logged in. (Figure 3.4.3A) Another function will be called, too, such as the getImage() to output the user’s picture. The user is also able to interact with this webpage by clicking on any of the three buttons: play, stop, back, and forward. If a user clicks on any of the buttons, the Javascript function that belongs with the button will be called and the user will either go the next featured channel and etc. We got the source code from Karthik Viswanathan posted on lateralcode.com.

**** (Figure 3.4.3A)

**3.5. Tag**

**3.5.1. Model**

The Tag model layer has one class called Tag.

**3.5.1.1. Tag**

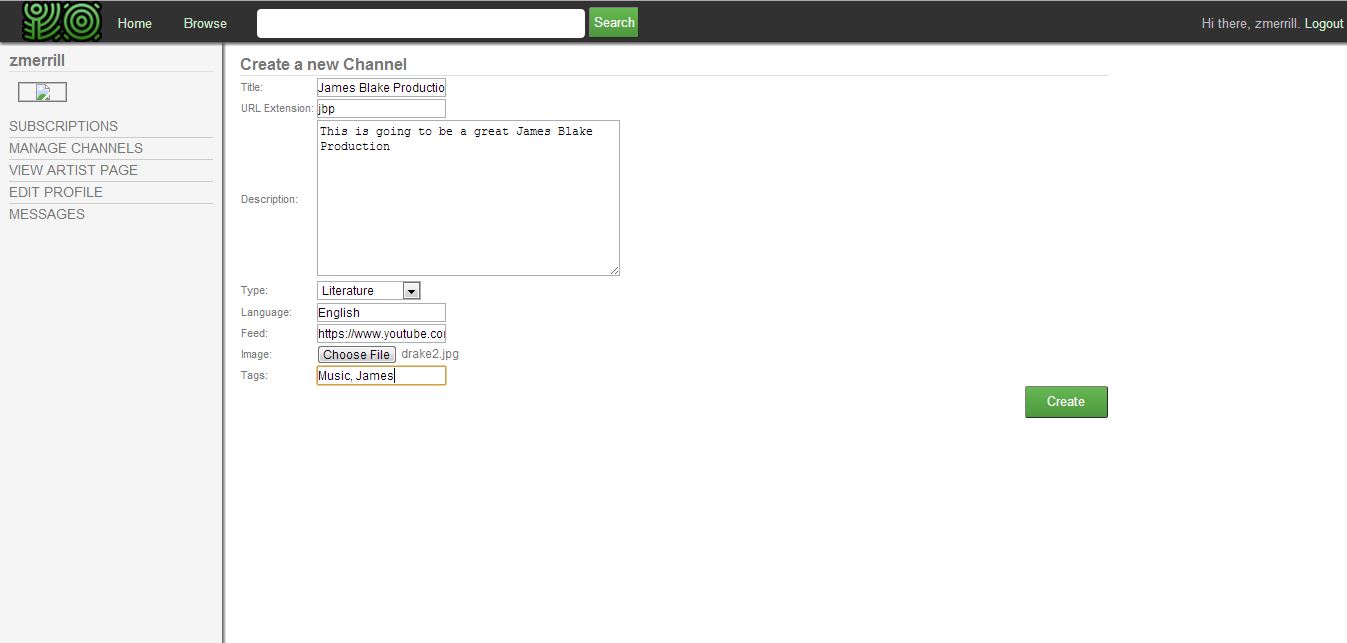
This gives the user the ability to tag what kind of media they are going to post.

|  |  |
| --- | --- |
| **ChannelType Methods and Properties** | |
| tagName | Where the user is able to tag a channel by type(s). |

**3.5.2. View**

There is no view layer required for the tag.

**3.5.3. User Interface**

****This interface shows how the tagName class is being used when a user is tagging his/her added channel (Figure 3.5.3A). Once a user submits his/her tags, it will be saved to the database.

(Figure 3.5.3A)

**3.6. Search**

**3.6.1. Model**

There is no model required for the search engine.

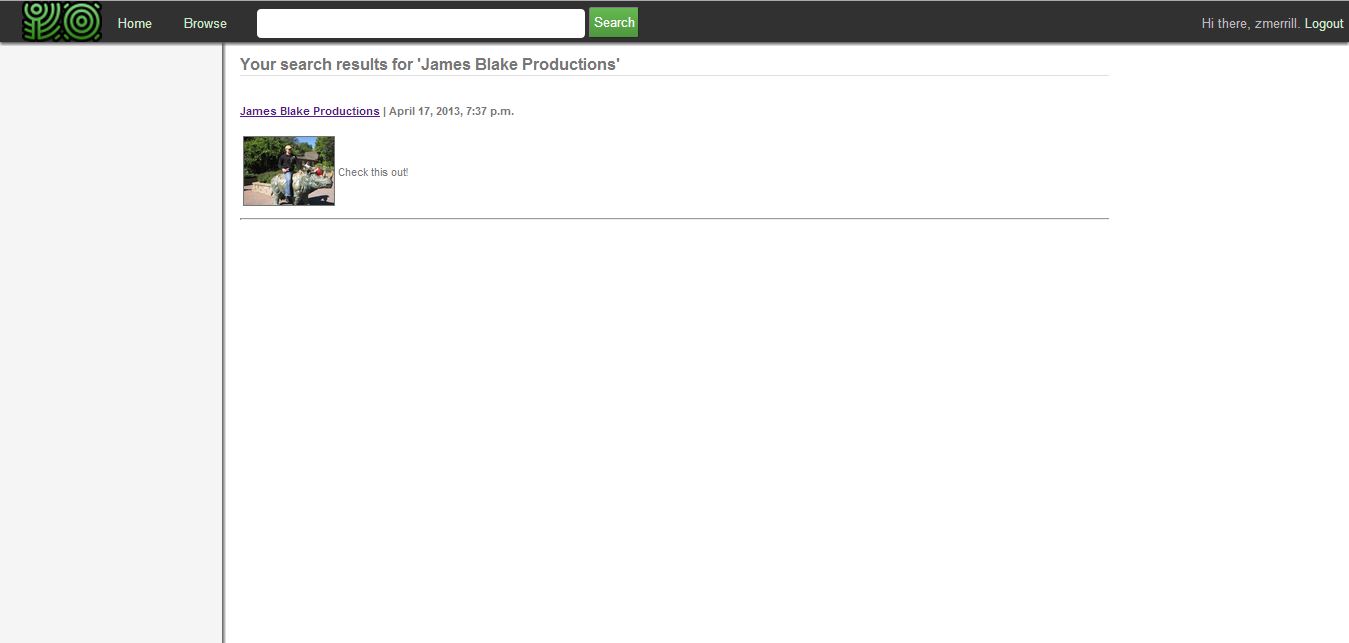
**3.6.2. View**

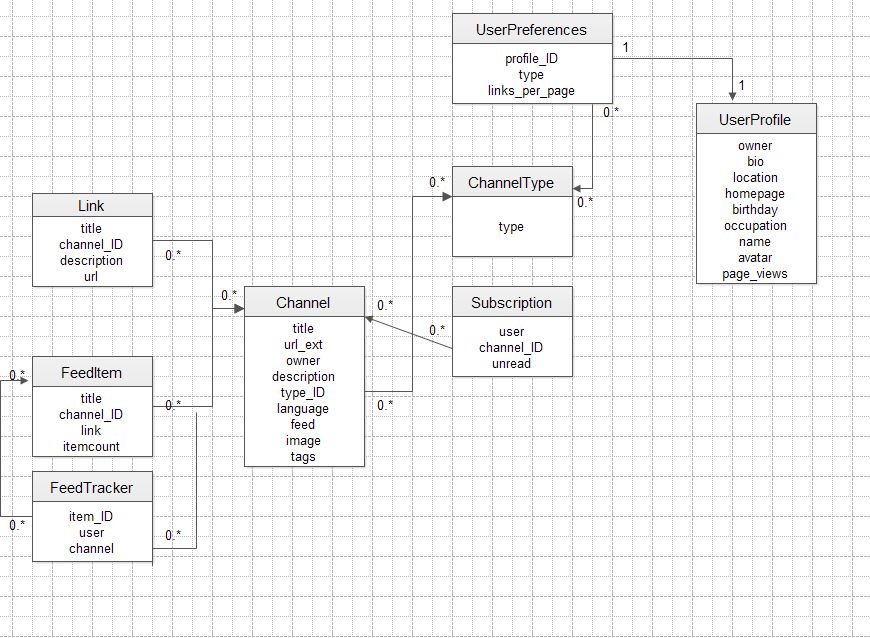
This handles the web request and web response so the search engine can find what the user is searching for.

|  |  |
| --- | --- |
| **Results Methods and Properties** | |
| Results | Searches all channels for matches in either channels or users. |

**3.6.3. User Interface**

This interface shows how a user searched for a specific title of a channel (Figure 3.6.3A). Once the user hits enter to find what he/she is looking for, the results() function will search for what the user is searching for.

Figure 3.6.3A

**3.7. Data Dependencies**

**4. Third Party Plugins**

The MyNewMedia uses third-party plugins to save time from creating these applications.

**4.1. Django-hitcount**

This allows users to know how many people have subscribed to their channel, how many people viewed their channel, and etc.

**4.2. Feedparser**

This gives users the ability to parse syndicated feeds.

**4.3. Unipath**

Unipath helps the MyNewMedia team to interact with the front end to the directory function.

**4.4. Django-messages**

This gives users the ability to send private messages back and forth to another user.

**4.5. Django-taggit**

This gives users and admins the ability to tag a channel media of their own choosing.

**4.6. Django-registration**

Gives the ability to the user to register and have their account activated in the website.