MyNewMedia Dynamic Content Channels

CSCI-4970-001 Capstone Project

Spring 2013

John De Los Reyes

Stephanie Koesters

Zachary Merrill

Table of Contents

[1 Introduction 4](#_Toc355362757)

[1.1 Motivation 4](#_Toc355362758)

[1.2 Functionalities 4](#_Toc355362759)

[1.2.1 Secure User Database 4](#_Toc355362760)

[1.2.2 Channel Functionality 4](#_Toc355362761)

[1.2.3 Auto-Subscribe Icons 5](#_Toc355362762)

[1.2.4 Channel Search 5](#_Toc355362763)

[1.2.5 Additional Functionality 5](#_Toc355362764)

[1.3 Implementation Strategy 5](#_Toc355362765)

[1.3.1 Server-Side 5](#_Toc355362766)

[1.3.2 Client-Side 6](#_Toc355362767)

[1.4 Technical Challenges 6](#_Toc355362768)

[1.4.1 Creating a database 6](#_Toc355362769)

[1.4.2 Using different media 6](#_Toc355362770)

[1.4.3 User Recommendations 6](#_Toc355362771)

[2 Functional and Nonfunctional Requirements 6](#_Toc355362772)

[2.1 Specific Requirements 6](#_Toc355362773)

[2.2 Functional requirements 6](#_Toc355362774)

[2.3 Non-functional Requirements 10](#_Toc355362775)

[2.4 Internal Data Requirements 12](#_Toc355362776)

[3 Architecture and Design 14](#_Toc355362777)

[3.1 Overall Architecture 14](#_Toc355362778)

[3.1.1 Django 14](#_Toc355362779)

[3.1.2 *Model* 15](#_Toc355362780)

[3.1.3 Database 15](#_Toc355362781)

[3.2 Application Decomposition 16](#_Toc355362782)

[3.2.1 Channel 16](#_Toc355362783)

[3.2.2 Subscription 20](#_Toc355362784)

[3.2.3 Profiles 22](#_Toc355362785)

[3.2.4 Core 23](#_Toc355362786)

[3.2.5 Tag 24](#_Toc355362787)

[3.2.6 Search 25](#_Toc355362788)

[3.3 Data Dependencies 27](#_Toc355362789)

[3.4 Third Party Plugins 27](#_Toc355362790)

[3.4.1. Django-hitcount 27](#_Toc355362791)

[3.4.2. Feedparser 27](#_Toc355362792)

[3.4.3. Unipath 27](#_Toc355362793)

[3.4.4. Django-messages 27](#_Toc355362794)

[3.4.5. Django-taggit 27](#_Toc355362795)

[3.4.6. Django-registration 28](#_Toc355362796)

[4 Implementation Issues 28](#_Toc355362797)

[5 Testing Plan 29](#_Toc355362798)

[5.1 Testing purpose 29](#_Toc355362804)

[5.2 Scope 29](#_Toc355362805)

[5.3 Technologies 29](#_Toc355362806)

[5.4 Overview 29](#_Toc355362807)

[5.2 Core Site Tests 30](#_Toc355362808)

[5.2.1 Selenium Tests 30](#_Toc355362809)

[5.2.2 3rd-Party Package Test 31](#_Toc355362810)

[5.3 Channel App Tests 31](#_Toc355362811)

[5.3.1 Model Tests 31](#_Toc355362812)

[5.3.2 View Function Tests 32](#_Toc355362813)

[5.5.4 Profile App Tests 32](#_Toc355362814)

[5.5.4.1 Model Tests 32](#_Toc355362815)

[5.4.2 View Function Tests 32](#_Toc355362816)

[5.5.5 Recommendation App Tests 32](#_Toc355362817)

[5.5.5.1 Model Tests 32](#_Toc355362818)

[5.6 Search App Tests 33](#_Toc355362819)

[5.6.1 Model Tests 33](#_Toc355362820)

[5.6.2 View Function Tests 33](#_Toc355362821)

[5.6.2.1 ‘searchresults’ function test 33](#_Toc355362822)

[5.7 Subscription App Tests 33](#_Toc355362823)

[5.7.1 Model Tests 33](#_Toc355362824)

[5.7.2 View Function Tests 33](#_Toc355362825)

[6 Lessons Learned 33](#_Toc355362826)

[6.1 John 33](#_Toc355362827)

[6.2 Stephanie 35](#_Toc355362828)

[6.3 Zach 35](#_Toc355362829)

# Introduction

## 1.1 Motivation

Today there are an innumerable amount of options for content creators to deliver their work to the public. The advent of free sites like Youtube and Soundcloud and the rise of grassroots podcasters have allowed people to break down the previously impenetrable barrier of entry that came with radio and television. Our media now comes in the form of streaming video and audio, embedded into pages or delivered via RSS feeds. But, as with any new media, there are some growing pains.

Average users, particularly from the later generation, are not used to the number of options available to them, which can quickly become overwhelming as they search for new content that suits their interests. Additionally, it is time consuming and frustrating to keep track of content updates when there could be dozens of content sources a user needs to watch.

The proposed site “MyNewMedia” will solve this issue. It allows content creators a single location to manage their content, while consumers will find their favorite sources organized into easy to use, subscribed channels that can notify them of any update, no matter what site it originated from. “MyNewMedia” will aggregate the content a user produces by embedding the audio, video, images or text hosted elsewhere onto the channel homepage. When a user subscribes to a channel, they will have access to all of this content in one location. Searching for content will also be made much easier. Each homepage will act as a gateway to any hosted content.

## 1.2 Functionalities

The following is a list of proposed functionalities for the “MyNewMedia” site.

### 1.2.1 Secure User Database

The most basic requirement is to allow users to create secure, persistent accounts. Initially, there will not be any need to hold sensitive information, but security will be a high priority in any case. Each user will have a list of subscriptions associated with their account, as well as a list of created channels. A user may also enter profile information, although this will not be required. The user must create an account and log in before any subscriptions or channels can be created. One goal is to integrate user accounts with Facebook, in order to make account creation as simple as possible.

### 1.2.2 Channel Functionality

The main draw of “MyNewMedia” will be the ability to create and maintain content channels as a way to streamline content to consumers. Each user will have the ability to create as many channels as they desire. Once a channel has been created, the user can name it as they wish and provide a description.

Content can be provided simply by adding a new link to a content list, which will automatically embed the content into the channel homepage, and notify any subscribed users of a change. The team is also investigating the ability to hook into an existing external account, such as Youtube, and dynamically add the links to the channel homepage when new external content is added. Each user will also have a dashboard that displays their subscribed channels and any updates that are available. Users will not be required to create an account to view a channel homepage, but an account is required to subscribe.

### 1.2.3 Auto-Subscribe Icons

In order to allow content creators to expand their audience as easily as possible, “MyNewMedia” will allow users to generate HTML image links that link back to their channel homepage and, when clicked, will add a subscription once a user has either logged in or created an account. These images can be added easily to any existing page, forums or feeds and will give users an easy way to grow their audience. A non-image link will also be available.

### 1.2.4 Channel Search

The ability to search channels and content is an essential function. Because “MyNewMedia” will be aggregating content on numerous sites, a powerful search will ensure that it remains the go to site to find any kind of content.

### 1.2.5 Additional Functionality

In addition to the essential functionality detailed above, the team has also discussed some more intelligent functionality that can be included. The first is a tag and recommendation system. The database will contain rich information about user activity, and this can be queried to provide recommended channels to a user based on what other users with similar subscriptions liked. A tag system that allows users to enter their own descriptions can provide similar functionality, as well as make searching for content easier.

Also discussed was a badge system. People respond well to rewards, especially online, so creating badges that can be awarded after a user performs a certain action can be great encouragement. Users will be rewarded for using the site in different ways, such as creating a channel, uploading content, or subscribing to a number of channels. It’s a fun way to get users acquainted with what “MyNewMedia” is capable of.

## 1.3 Implementation Strategy

Although the project is in the beginning planning stages, the team has a good idea of how to move forward. “MyNewMedia” will primarily be web-based, although it will be designed in a way to make the creation of a mobile app relatively simple. Because there is no need to host any content, a lightweight database can be used to hold user account and channel information. The team will be approaching the work by dividing it into two sections: the sever-side and the client-side.

### 1.3.1 Server-Side

The back-end will be built using the MySQL database, which is an excellent free database that is easy to set up and deploy. We plan on making liberal use of stored procedures to handle as much of the business logic as possible so the front-end clients have a light computational load.

### 1.3.2 Client-Side

Using new technologies like HTML5 in addition to the typical CSS should make the website design simple. The goal is to use as little code on the client-side as possible. PHP will be used as a scripting language. The team also has experience with MVC architecture tools such as Oracle ADF and ASP.Net if the need for a more robust toolset arises.

## 1.4 Technical Challenges

### 1.4.1 Creating a database

We will have to create a database that not only stores the information, but is easy to use and maintain in the future. We will also have to learn how to set up a non-local server so the site can actually be implemented upon our project's completion.

### 1.4.2 Using different media

The goal of our site is to unite different types of media in a useful manner, and our challenge is to not only unite the media in a viewer-friendly manner, but to create code for every type of media we wish to use.

### 1.4.3 User Recommendations

Once the basis of the site has been constructed and preferably populated with information, we would like to implement the previously mentioned recommendation system. This will demonstrate our programming ability to create a database that can use logic and think for itself.

# 2 Functional and Nonfunctional Requirements

## 2.1 Specific Requirements

The ultimate objective of our website is to provide an interface that efficiently combines a user's content from different media sites into one place.

## 2.2 Functional requirements

1. The system shall allow users to create accounts.

1.1. Accounts are required to subscribe to or create channels.

1.2. All accounts shall be protected by a password.

1.2.1. A user shall be able to change their account's password at any time.

1.2.2. A user shall use this password and a unique account name to log in.

**Name:** User\_Login

**Actors:** Initiated by User, carried out by System

**Entry Conditions:** User is logged out and within site

**Flow of Events:**

1. A User enters a User ID in the User ID textbox
2. User enters password in Password textbox
3. User clicks Login button
4. User Dashboard is displayed

**Exit Conditions:** The dashboard is displayed in the web browser

**Exceptions:**

1. The system cannot locate the user’s ID.
2. User's entered password and ID combination do not match

1.3. An account shall be capable of owning one or more content channels.

1.3.1. Each channel is created by the user that owns that channel.

1.4 An account shall be capable of subscribing to multiple channels

**Name:** Subscribe\_to\_Channel

**Actors:** Initiated by User, carried out by System

**Entry Conditions:** Logged-in user is on another user's channel page

**Flow of Events:**

1. User clicks "subscribe" button.
2. User's subscription is registered in database.

**Exit Conditions:** Newly subscribed channel content is displayed on user's dashboard.

**Exceptions:**

1. User is already subscribed.
2. User subscribes to incorrect channel.

1.4.1. A user shall be able to unsubscribe from subscriptions at any time.

2. A channel shall be a feed of user populated media content.

2.1. A channel page shall be automatically created for each channel.

2.1.2. Users shall create their own channels.

**Name:** Create\_Channel

**Actors:** Initiated by User, carried out by System

**Entry Conditions:** User is logged in, on Dashboard page, in Manage Channels tab

**Flow of Events:**

1. User clicks on ‘Add Channel’ in the menu
2. User enters channel information in form
3. User uploads picture to site
4. User clicks ‘Submit’
5. User is redirected to the Channel Home Page

**Exit Conditions:** The new Channel Home Page is displayed in the browser

**Exceptions:**

1. The channel form contains errors
2. The uploaded picture is too large
3. The uploaded picture is not a recognized format
4. The URL Extension is already in use

2.2. A user shall be able to add content by linking to the content's source

2.2.1. The system must be compatible with the client's requested sites and content.

2.2.1.1. Including but not limited to YouTube.

2.2.1.2. More media sites can be implemented when needed.

2.3. User content shall have tags indicating what subject matter the media is related to.

2.3.1. Tags shall be used in search functions and for recommendations.

.4 A channel shall be able to populate content itself.

2.4.1. Given an RSS feed link, MyNewMedia can retrieve media content by parsing the XML entries.

2.4.2. RSS feed code shall display as media content in the manner that was originally intended (videos display videos, music displays music playets, etc.)

3. A user's home page shall consist of the media content of subscribed channels.

3.1. All subscribed posts shall be listed vertically.

3.2. Media content from an external source shall have a link to the original source (such as YouTube).

3.3. Media content shall have some indication of the channel that owns it.

3.3.1. The channel's name and/or profile shall be displayed on the user's feed near the media content.

3.4. Media content shall be ordered showing the most recently uploaded content first, directing the user to what has most likely not been seen yet.

3.4.1. MyNewMedia shall log what the user has already seen and use that information to distinguish between new and old content to users.

4. The site shall be able to search for channels and media content.

**Name:** Search\_By\_Channel

**Actors:** Initiated by User, carried out by System

**Entry Conditions:** User is logged in.

**Flow of Events:**

1. A User clicks the Search text box
2. User enters in search criteria
3. User clicks "Search."
4. Channels matched by criteria are displayed on the page.

**Exit Conditions:** Channels the user has searched for are displayed.

**Exceptions:**

1. No results are found.
2. User enters no criteria before searching.

4.1. The basic search option shall be visible from each page.

4.2. The search function's criteria shall be able to consist of tags, user names, or channel names.

5. The site shall provide users with options to share their channels or media content.

5.1. HTML code to be posted on a user's side shall be generated for channels and media content.

6. The site shall be capable of recommending other channels to the user based on the user's preferred tags.

6.1. Preferred tags shall come from on the user's subscribed channels' tags.

6.2. Based on the most popular tags, MyNewMedia shall look for other channels that share the same tags and display those in the Recommendations section.

6.3. The Recommendations section shall not interfere with other site functionality and shall not create layout issues.

7. The site shall have an administrator interface.

7.1. The administrator shall have access to the database.

7.1.1. The administrator shall be able to delete or add accounts.

7.1.2. The administrator shall be allowed to add, edit, or delete tags.

7.1.3. The administrator shall be allowed to add, edit, or delete channels.

## 2.3 Non-functional Requirements

The website's functions shall be optimized for an audience ages 40 and up; many overwhelming functions in other forms of social media can be excluded for a simplified interface consisting of only what is required.

1. Creating an account will be simple.

1.1. Creating an account shall require minimal information.

1.2. Subscribing to a channel shall require one click.

1.3. Posting media content shall be concise.

1.3.1. Posting media content shall only require a URL link to the content's page.

2. Viewing media content shall be simple.

2.1. The center feed shall only consist of the user's desired media content, as opposed to combining recommendations and advertisements.

2.1.2. Should extra content be desired, it shall be placed above or to the sides of the center feed.

3. Information shall be secure.

3.1. The users shall have access to their own information.

3.1.1. Besides the initial user, only administrators shall have access to that user's information.

3.1.2. User information shall never be shared.

3.2. The site shall be coded in a manner that minds possible security breaches.

4. The site shall provide recommendations to the user

4.1 The site shall have the capability to analyze user behavior and create a list of recommendations to the user

4.1.1 Information gathered shall be the user’s subscription and the user’s watch history

4.1.2 The channels within user’s history and subscription list will be compared using an algorithm to other channels with the same tags that have high view or subscription counts and added to a list of recommendations to the user

4.2 The site shall have the capability to recommend a list of channels to a user based on the currently active channel

4.2.1 The channels within user’s history and subscription list will be compared using an algorithm to other channels with the same tags that have high view or subscription counts and added to a list of recommendations to the user

5. The project shall follow the minimal budget allotment indicated.

5.1. The team coding the websites will require no additional software finances. All software used shall either be free or in possession prior to the project.

5.1.1. The Django framework shall be used to create the database, as well as access the content from a coding level.

5.2. The final domain name shall require minimal funding to obtain.

6. The project shall require a database and a server.

6.1. The initial database shall be created using SQLite, and shall be implemented on a different system for the final project.

6.2. The data shall be accessed using Django and its built-in administrator interface.

6.3. The database shall be hosted on Amazon's web server.

7. The site shall be debugged extensively before deployment.

## 2.4 Internal Data Requirements

In summary, all tables much hold the data and relationships between users, channels, subscriptions, and statistics. Django automatically creates primary key IDs for each table; thus, all are identical in stature.

5.3.1. User

The user table holds data so that the user can log in.

|  |  |
| --- | --- |
| **Users Table** | |
| ID | Integer (PK) |
| Username | Char(30) |
| Password | Char(128) |

The Django framework automatically has an authorized user table, which is where an admin (superuser) property is set, and an id is also created whenever a new user is registered in the database.

5.3.2. User Profile

The table is a profile consisting of the user's personal information.

|  |  |
| --- | --- |
| **User Profile Table** | |
| ID | Integer (PK) |
| Owner\_id | Integer |
| Bio | Text |
| Location | Char(256) |
| Homepage | Char(200) |
| Birthday | Date |
| Occupation | Char(256) |
| Avatar | Char(100) |
| Page\_views | Integer |

The owner ID is linked to the ID of the User table. The bio, location, and occupation are custom text that the user enters. The homepage is a URL to the user's main site. The birthday is the user's date of birth. The avatar is linked to the user's uploaded image to represent himself/herself. The page views track how many times the profile has been viewed.

5.3.3. Channels

The channels table holds the data for each channel.

|  |  |
| --- | --- |
| **Channels Table** | |
| ID | Integer (PK) |
| Title | Char(256) |
| URL\_ext | Char(50) |
| Description | Text |
| Type\_id | Integer |
| Owner\_id | Integer |
| Language | Char(100) |
| Feed | Char(200) |
| Image | Char(100) |

The title, language, and description are custom text from the user. The URL extension (URL\_ext) is the ending of the user's URL for the channel page. The owner ID is automatically assigned as the user submits a channel creation form. The type ID is the channel type, selected from admin-defined categories. The Feed field is the link to an RSS feed used to populate the channel's content. The Image field is a .jpg image used to represent the channel.

5.3.4. Channel Type

The channel type table holds different categories a user can place channels in.

|  |  |
| --- | --- |
| **Channel Type Table** | |
| ID | Integer (PK) |
| Type | Char(256) |

The type is the name of the genre. The table is populated by the administrator(s).

5.3.5. Subscription

The subscription table is a relationship between a user and the user's subscribed channel.

|  |  |
| --- | --- |
| **Channel Type Table** | |
| ID | Integer (PK) |
| User\_id | Integer |
| Channel\_id | Integer |
| Unread | Integer |

The user and channel IDs are keys to their respective tables. This indicates the user in this record is subscribed to the channel in this record, and hasn't read the number of items in the "unread" field.

5.3.5. Tags

From an upper-level perspective, the tags must have a table filled with tag names and their corresponding IDs, as well as a table that associates those IDs to channels.

|  |  |
| --- | --- |
| **Tag Table** | |
| ID | Integer (PK) |
| Name | Char(100) |

The ID of the tag, and the displayed name of the tag.

|  |  |
| --- | --- |
| **TaggedItem Table** | |
| ID | Integer (PK) |
| Tagged\_ID | Integer |
| Object\_ID | Integer |

The ID is only a primary key. The Tagged\_ID is from the Tag table above, and the object\_ID is the object the tag is associated to. In some sense, the tags can be used for profiles or users, but MyNewMedia implements tags strictly to channels for simplicity.

5.3.6. Feed

The primary way of populating content is using a feed. Similar to tagging, this requires a feed table and a relationship to the feed from the channel.

|  |  |
| --- | --- |
| **FeedItem Table** | |
| ID | Integer (PK) |
| Title | Char(1000) |
| Channel\_id | Integer |
| Link | Char(200) |
| Itemcount | Integer |

The primary purpose is to have the link to the RSS feed.

|  |  |
| --- | --- |
| **FeedTracker Table** | |
| ID | Integer (PK) |
| Item\_id | Integer |
| User\_id | Integer |
| Channel\_id | Integer |

This table's purpose is to unite the FeedItem table with the other tables.

# 3 Architecture and Design

## 3.1 Overall Architecture

### 3.1.1 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.

#### 3.1.1.1 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.

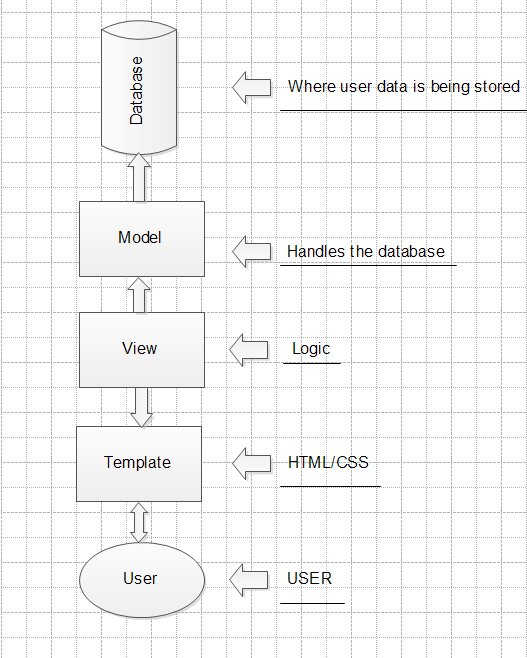
There are two keys to a successful Django web application, and any application that uses MVC or MVT architecture: code reusability and separation of concerns. The code for any Django project is organized into “applications” that can be bundled and reused anywhere in the application, or exported and used 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.

### 3.1.2 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.

### 3.1.3 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.

3.1.4 ViewFigure 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.

#### 3.1.5 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.

## 3.2 Application Decomposition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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. |

### 3.2.1 Channel

#### 3.2.1.1 Model

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

##### 3.2.1.1.1 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. |

##### 3.2.1.1.2 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. |

##### 3.2.1.1.3 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. |

##### 3.2.1.1.4 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. |

##### 3.2.1.1.5 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. |

#### 3.2.1.2 View

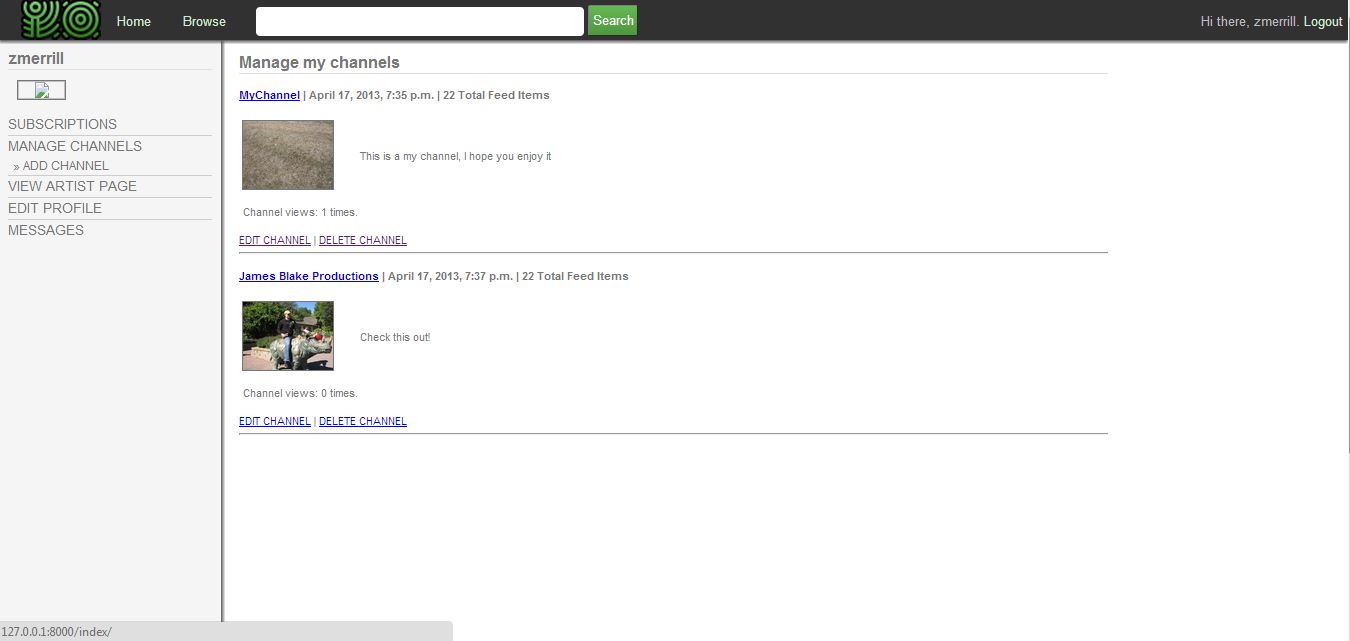
This handles the web response of what is being displayed on the user’s channel page.

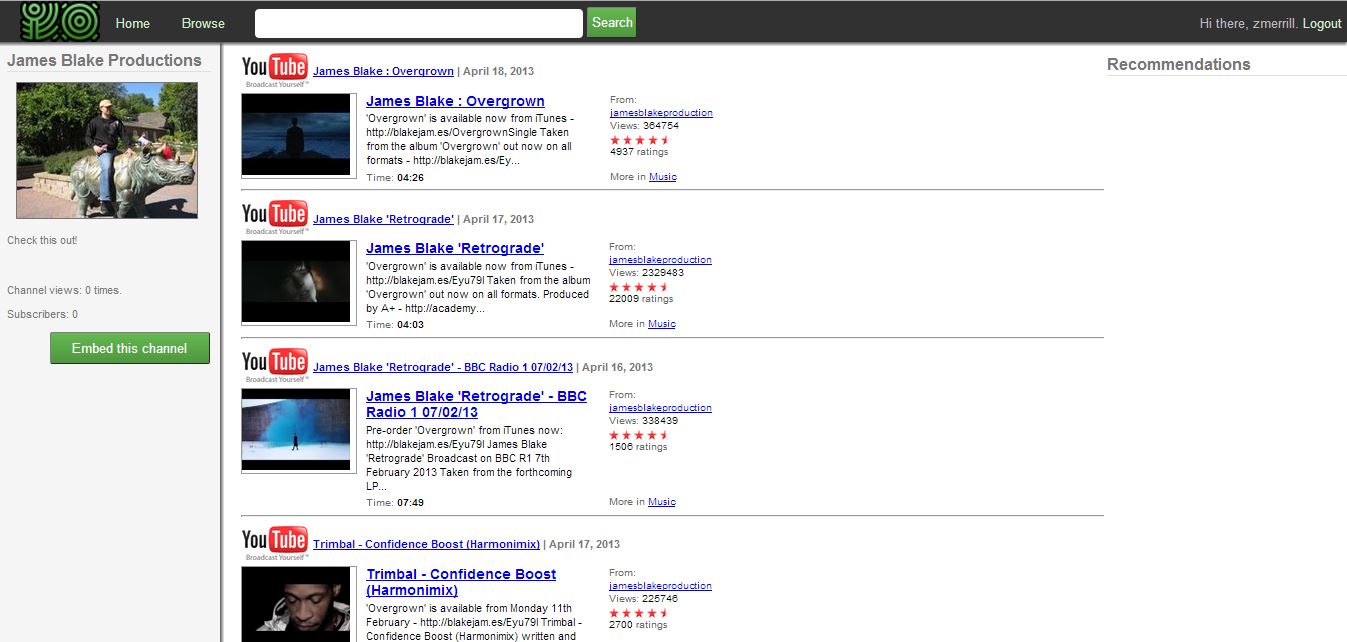
|  |  |
| --- | --- |
| **View Methods and Properties** | |
| Channelhome() | Displays all content and information about a channel on the user’s page. |
| MarkAsRead() | Adds the passed feed item to a user’s history and returns a response only by default. Since it’s called with Ajax, nothing will be 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. |

#### 3.2.1.3 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

### 3.2.2 Subscription

#### 3.2.2.1 Model

This model layer has one class called Subscription.

##### 3.2.2.1.1 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. |

#### 3.2.2.2 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. |

#### 3.2.2.3 User Interface

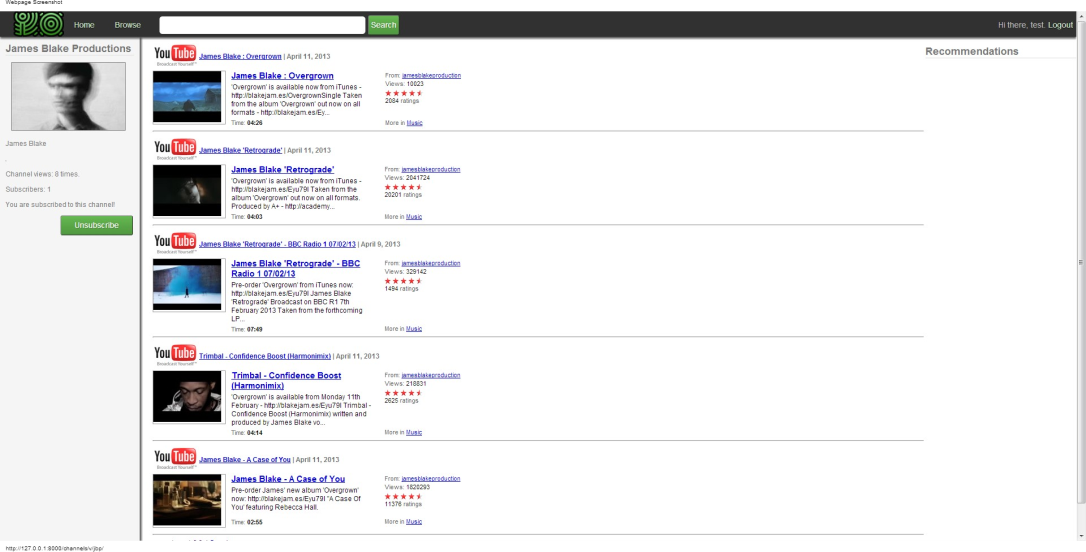
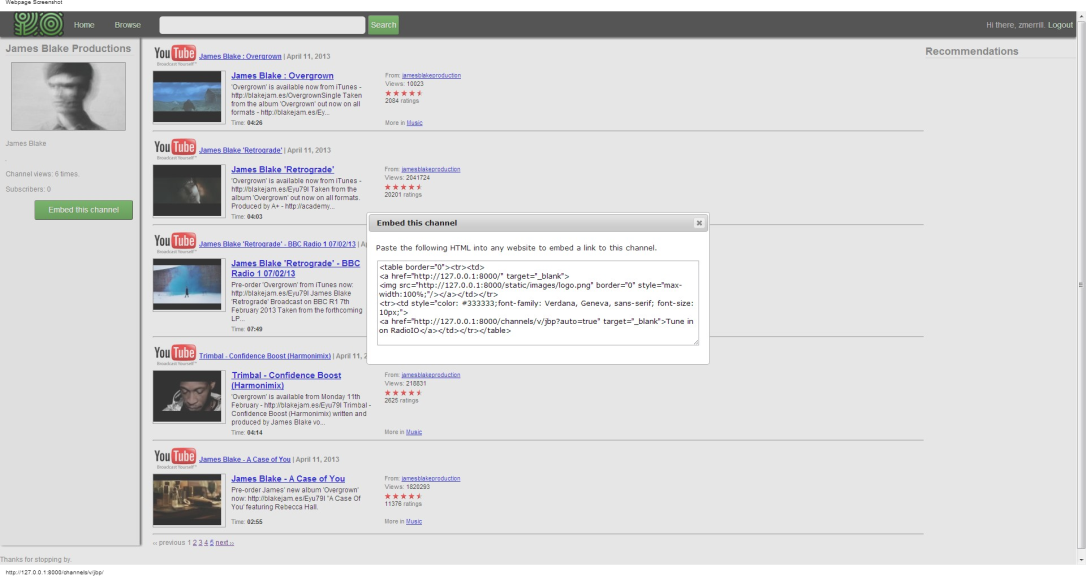
This interface shows a green button so users can subscribe and unsubscribe to other user’s channel (Figure 3.2.1A). Depending whether the user is subscribed or unsubscribed to another user’s channel, it will either choose the “subscribe” or the “unsubscribe” method in the view model once the user clicks 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

### 3.2.3 Profiles

#### 3.2.3.1 Model

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

##### 3.2.3.1.1 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. |

##### 3.2.3.1.1 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. |

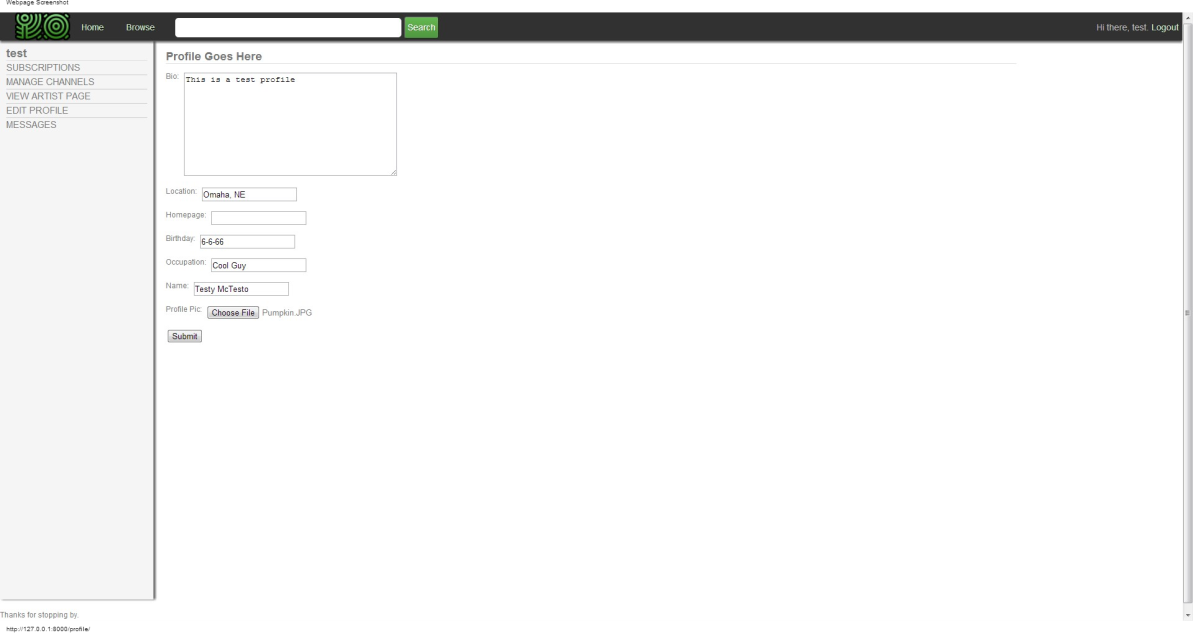
#### 3.2.3.2 View

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. |

#### 3.2.3.3 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)

### 3.2.4 Core

#### 3.2.4.1 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. |

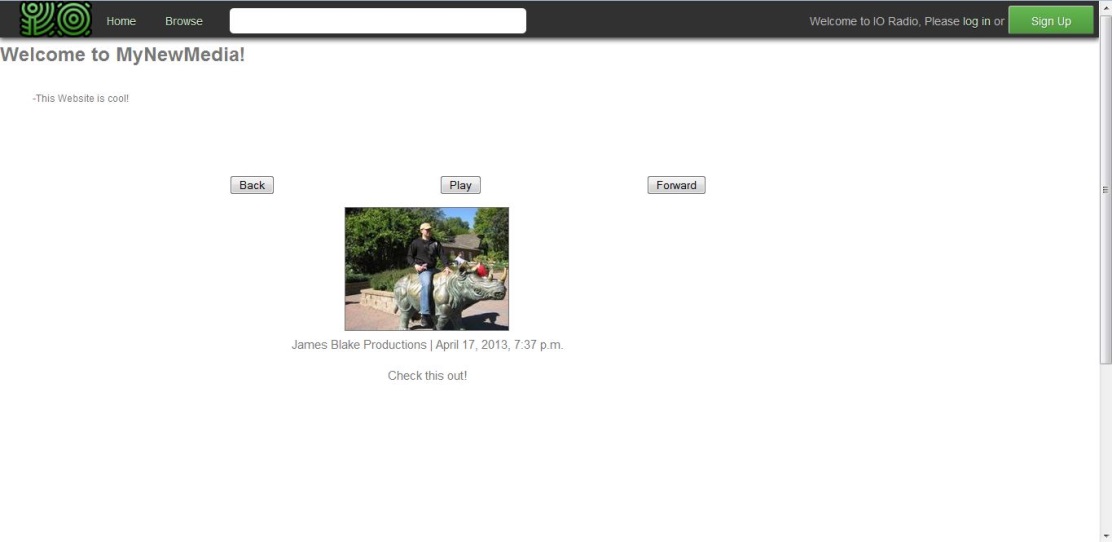
#### 3.2.4.2 View

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. |

#### 3.2.4.3 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)

### 3.2.5 Tag

#### 3.2.5.1 Model

The Tag model layer has one class called Tag.

##### 3.2.5.1.1 Tag

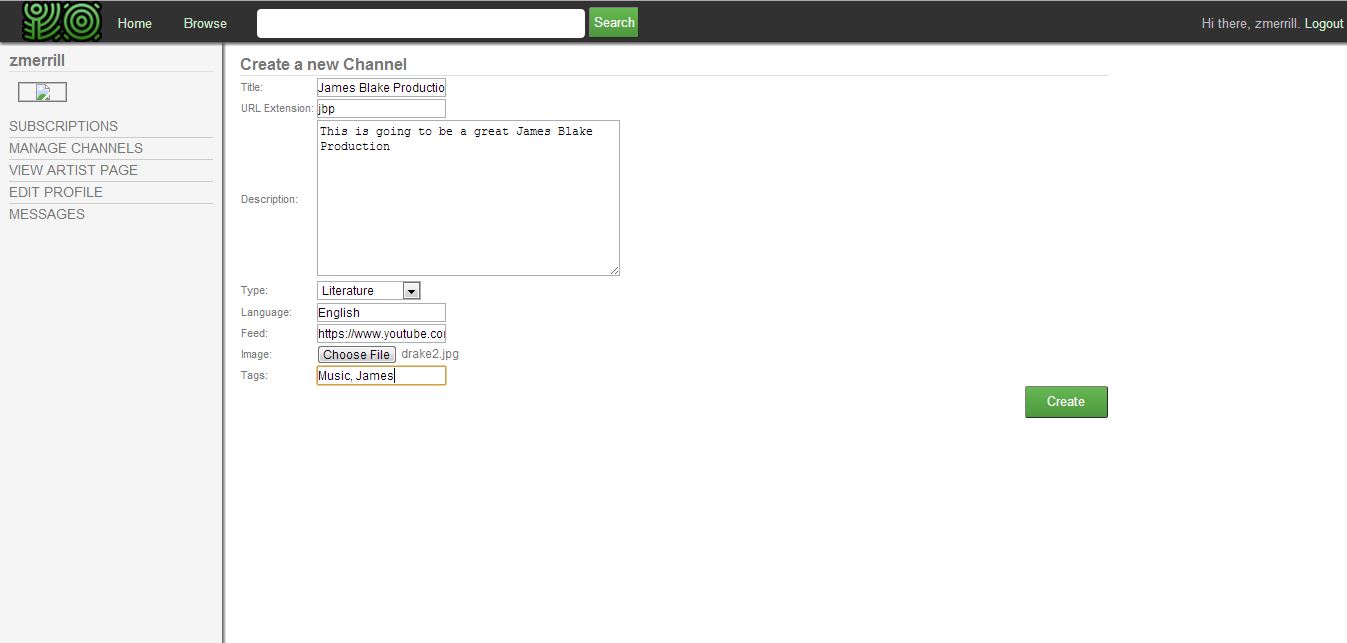
This gives the user the ability to tag what kind of media they are going to post.

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

#### 3.2.5.2 View

There is no view layer required for the tag.

#### 3.25.2 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)

### 3.2.6 Search

#### 3.2.6.1 Model

There is no model required for the search engine.

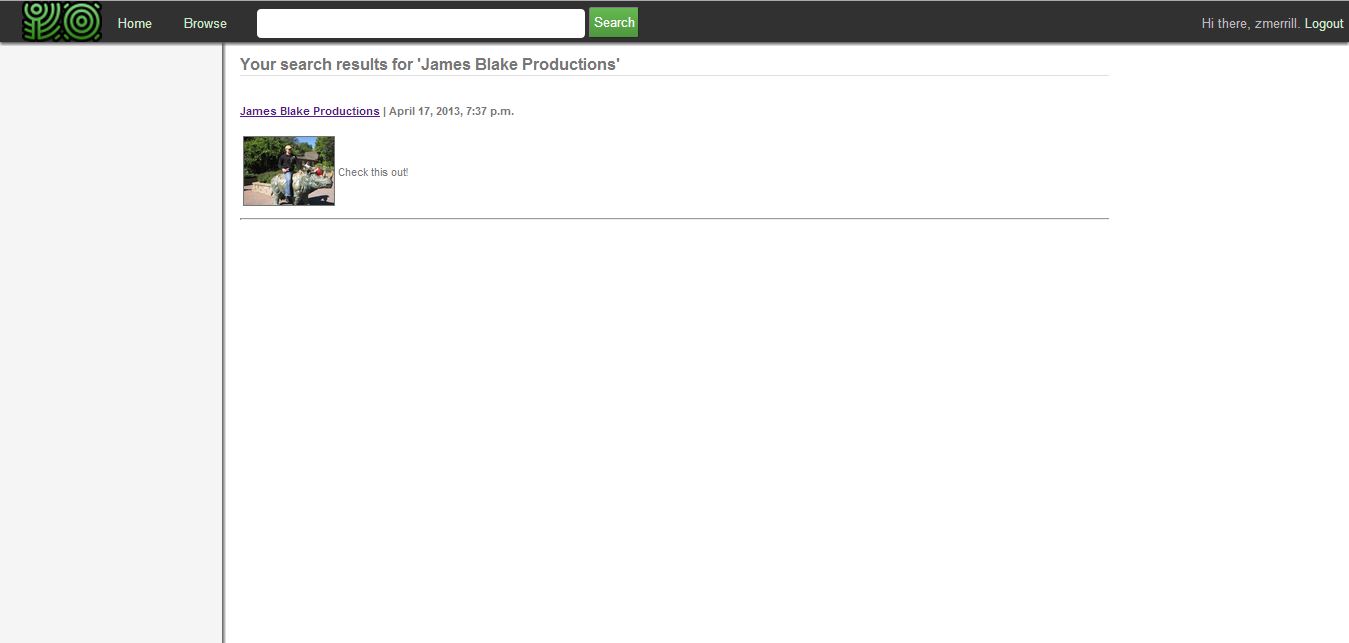
#### 3.2.6.2. View

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. |

#### 3.2.6.3 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

## entity3.3 Data Dependencies

## 3.4 Third Party Plugins

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

### 3.4.1. Django-hitcount

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

### 3.4.2. Feedparser

This gives users the ability to parse syndicated feeds.

### 3.4.3. Unipath

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

### 3.4.4. Django-messages

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

### 3.4.5. Django-taggit

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

### 3.4.6. Django-registration

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

# 4 Implementation Issues

We used GitHub to download and upload files of code and documentations to our local computer and to the website. GitHub was able to upload files nicely to the website, but when it came to downloading files from GitHub, it resulted in a lot of issues because sometimes it didn't fully download everything from the file. With that, it resulted in a lot of frustration because it took a while to know which file(s) in the folder were missing. There were two cloning issues that resulted in a lot of frustration to the MyNewMedia team. The first problem was one of our group members didn't get a couple of files after he cloned it to his local PC. Another problem was that cloning the folder of files to our local PC didn’t always work, it then forced the group members to download the zip file. Even if you downloaded the zip file, there is a chance that a couple of files will be missing.

We had to download multiple third party plugins and there were several issues with coding when we installed them to make them work with our website. Since we had to use our command prompt to install them in our local computers, sometimes it didn’t install them properly, and sometimes it wouldn’t even install them at all. We were not able to get the full functionality of the website unless we downloaded all the needed Django applications. It took more time for team members to get started coding because we had to communicate with the person who added the Django application to see how he/she did it. There were a lot of tutorials for these applications, but that didn’t help much, and sometimes it actually made it worse because we installed something that we shouldn’t have. For example, one of the team members was trying to install a Django application called Django-taggit, and the problem was that the tutorial instructed to install a folder that he didn't need. This resulted in multiple errors to the code, but once he deleted the folders that he didn't need, the errors went away. Even though our group had trouble with installing Django applications, there were some packages that were outdated which resulted in us not being able to use them.

Setting up Django was very difficult because it was free and there was a lot to install, such as Python, an Apache server, and so on. Since there was a lot to do, it took the team a long time before they could start writing code to make the website work. Similar to installing Django applications, some applications didn't install in our local PC's nicely, others couldn't install to our local PC's at all. This resulted in a lot of time wasted because all the work we did before the other installations couldn't be put into use.

Lastly, reading other people's code took a lot of time because each person in our group thinks differently. Since we only meet twice a week, we had to email each other using our Gmail or by Facebook to understand specific lines of code or to get an idea on how to code something up in our website. With that being said, since most of the group members knew HTML and CSS, those languages were not a big issue. It was the backend part of the code that gave most of the group members trouble because programming in Python was new to everyone.

# 5 Testing Plan



## Testing purpose

As a potentially high-traffic internet destination the MyNewMedia site will need to be subjected to a rigorous testing plan. The site will need to be completely bug free before it can be deployed to a general audience. All tests shall be designed with regression testing and automation in mind, so that future developments can be tested against old requirements with ease.

## Scope

There are three main types of tests the MyNewMedia site will use, each corresponding to the Model-View-Template architecture that the Django framework enforces. The model tests will create database objects and query for them, to ensure the database both accepts and returns good data and rejects data that does not conform to the constraints. View level tests will make sure the underlying site functions are behaving as expected. Template level tests will be browser based scripts that ensure HTML and CSS components are being rendered correctly.

## Technologies

All model and view level tests will be programmed as Python unit tests that can be run from the server directly. Template layer tests will be scripted using the Selenium IDE – a plug in for the Firefox web browser. Selenium allows developers to mimic browser interaction in code. These tests can be exported to Python and packaged with the other unit tests, but require a Firefox browser to be installed with the Selenium IDE to be run.

## Overview

The following is an outline of the tests to be performed:

* Core Site Tests – includes Selenium tests and tests of 3rd-party packages
* Channel App Tests
* Profile App Tests
* Recommendation App Tests
* Search App Tests
* Subscription App Tests

## 5.2 Core Site Tests

### 5.2.1 Selenium Tests

The following tests shall be scripted using the Selenium IDE and shall require a Firefox web browser to run. Each test references a specific web page behavior in the site.

#### 5.2.1.1 Login Test

The login test verifies the login behavior by logging into the site with a test user that shall be created during the test set up.

#### 5.2.1.2 User Home Redirect Test

The user home redirect test verifies that the test user will be redirected to the User Home page after logging in.

#### 5.2.1.3 Search Test

The script will enter a search for a unique test channel that shall be created during test set up. The search will be performed by looking up a user, channel title, channel description and channel tag. A successful test will return the correct channel in all cases.

#### 5.2.1.4 Subscription Test

This test will subscribe the test user to the test channel. A successful test will update the web page components to reflect the new data.

#### 5.2.1.5 Channel Manager Test

The script will create a channel using the channel form then verify the newly created channel appears on the test user’s Channel Manager page. The new channel will then be deleted. This script will test both the channel form and the Channel Manager functionality.

#### 5.2.1.6 Channel Home Test

The browser will be navigated to the test channel page using a specified URL. This test will verify the dynamic channel URLs are working correctly.

#### 5.2.1.7 RSS Feed Pull Test

On the test channel page, the script will verify the correct feed has been pulled, and the correct elements are present. A test RSS feed XML document has been created for this test.

#### 5.2.1.8 Logout Test

The script will verify the logout link on the upper right menu works correctly.

#### 5.2.1.9 Splash Page Redirect Test

The script will verify the browser is redirected to the splash page on logout.

### 5.2.2 3rd-Party Package Test

The following tests are designed to verify the 3rd-party Django packages used in the site are working correctly.

#### 5.2.2.1 Messages

This will test the Messages database tables for the Django-Messages packages. A message will be ‘sent’ from one user to another and verified that it ends up in another user’s inbox. The message will then be deleted from the senders outbox and the receiver’s inbox.

#### 5.2.2.2 Tags

This test will verify that tags are correctly assigned to channels and duplicate tags are not created.

#### 5.2.2.3 Registration

This test will create a test user and ensure the information is added to the database correctly and all information is properly encrypted.

## 5.3 Channel App Tests

### 5.3.1 Model Tests

#### 5.3.1.1 Channel

The Channel test will verify that added channels are correctly entered into the channel database tables and entries with incorrect data are rejected without saving. It will also verify that the channel pictures are correctly saved in the Media folder where all dynamic content is saved.

#### 5.3.1.2 Link

The Link test will verify that added manual links are correctly entered into the channel database tables and entries with incorrect data are rejected without saving.

#### 5.3.1.3 Feed Item

The Feed Item test will perform a feed pull and insert the returned items in the database. The test will verify that all feed items returned are entered correctly.

#### 5.3.1.4 Feed Tracker

The Feed Tracker test will verify that data is entered correctly into the database and duplicate entries are not allowed.

### 5.3.2 View Function Tests

Note that the forms used to add channels, edit channels, delete channels and add links will be tested in the browser using Selenium.

#### 5.3.2.1 ‘channelhome’ function test

This test will verify the return the correct channel in the resulting dictionary object when passed a test URL extension.

#### 5.3.2.2 ‘markasread’ function test

Verifies that the correct feed tracker object is inserted into the database when this function is passed a test channel id and feed item id.

#### 5.3.2.3 ‘pullfeed’ function test

Verifies the returned list contains the expected feed items when this function is passed a test feed URL and number of posts to limit the return list.

#### 5.3.2.4 ‘browsechannels’ function test

Verifies the returned dictionary contains all channel types.

#### 5.3.2.5 ‘artisthome’ function test

Verifies the returned dictionary contains all channel objects belonging to the passed user.

## 5.5.4 Profile App Tests

### 5.5.4.1 Model Tests

#### 5.4.1.1 Profile

The profile tests will verify all profile information is entered correctly. It will also verify that the profile pictures are correctly saved in the Media folder where all dynamic content is saved.

#### 5.4.1.2 Preferences

The preferences test will verify that user preferences are correctly saved in the database.

## 5.4.2 View Function Tests

#### 5.4.2.1 ‘profilequery’ function test

Verifies that the returned profile will always belong to the user currently logged in.

## 5.5.5 Recommendation App Tests

### 5.5.5.1 Model Tests

The Recommendation App does not use the model layer.

## 5.6 Search App Tests

### 5.6.1 Model Tests

The Search App does not use the model layer.

### 5.6.2 View Function Tests

### 5.6.2.1 ‘searchresults’ function test

Verifies that when passed a specific search term, the function will return three separate sets of objects in the dictionary. They are the search results based on channel title or description, based on all existing users and based on all existing tags.

## 5.7 Subscription App Tests

### 5.7.1 Model Tests

#### 5.7.1.1 Subscription

The subscription tests will verify that a subscription object created will be entered into the database correctly. It will also verify that the deletion of a subscription object will remove that object from all related tables.

### 5.7.2 View Function Tests

#### 5.7.2.1 ‘subscribe’ function test

Verifies the current user has a subscription created for the passed channel.

#### 5.7.2.2 ‘unsubscribe’ function test

Verifies the current user has an existing subscription, then verifies the function removed that subscription.

# 6 Lessons Learned

## 6.1 John

I’ve learned a lot in the project when it comes to learning how to code in Pyhton, how a database works, the Django framework, and eclipse. Before the project, I had never programmed in Python, used the Django framework, or used eclipse as a text editor. I did learn how to work with a database at my work, but not as much as what I have learned doing the project. So programming in Python was a bit strange for me because of the syntax, but after a while, I started to get used to it. When I started on the project, the only framework that I used was the .NET framework, so learning how Django worked was interesting because it made it easy for people to develop a website fast. Since I never used Eclipse as a text editor before, it was an interesting experience because it would tell me if I had an error in a code and etc., but I didn’t like eclipse’s interface and the lack of background themes.

The classes that helped me while I was working on the project was Introduction to Web Development (CIST 1300) and Programming on the Internet (CSCI 2850) because it helped me out a lot when it came to me understanding more about HTML, CSS, and CGI programming. Even though I never thought I would ever use those classes again, those two classes helped me get a good start on the project and on how to make a decent looking website.

If I could do it all over again, and since we only had three months to finish up the project, I would have learned more about Python because Django is a Python web framework. I would have also looked into other frameworks to use because it took the team time to find a framework that is free and provides fast development. Since looking online didn’t help me much when it came to programming in Python, I would have loved to find a good book to help me program in Python because I find it easier reading code from a book than reading it from a computer. Also, programming tutorial books usually have comments on what is happening in the code, and when I was looking online on how to code specific things, about 99% of the code that I looked at didn’t have any comments in them. I would also have learned more about how to interact with a database because I lacked in knowledge about it.

Since I’m not going to be working on the website after this class, I did enjoy the project because I learned more about web development and a new computer language called Python. If I have the time this summer, I hope I can learn more about Python because programming in Python was a fun experience. Not only was it a fun experience, I learned a new programming language that I can probably use in the future. I’m also hoping that I can learn more about databases after this class because using a database and interacting with it was a bit of a challenge for me while I was programming on the website. With that being said, since I had more experience working with a team, more knowledge about Python, and more knowledge about databases, I hope what I learned can be taken to my next internship this summer and/or this fall.

## 6.2 Stephanie

I think the project would've been more feasible with a concrete plan from the beginning and quicker milestones to present to the client. Most of the time vanished when we tried to use things we never previously learned (such as Python, XML parsing, and the Django applications with only fractions of documentation). I've learned through my actual job that Python can be useful, and delving into it slightly with this project was a good experience, although I wish I learned more than the bare minimum required to work with Django objects.

It seems that unless Django is used in other places, I didn't learn much syntax-wise, admittedly because I didn't touch many of the elements of code besides the tags and recommendations. I did, however, like using Eclipse for the first time, and used it for other school projects. If anything, perhaps I learned ways to keep track of monitoring a project's progress, and I see better ways to improve my priorities in future projects.

## 6.3 Zach

For me, the biggest lesson learned from this project would be to spend more time considering the scope of the project before the actual coding begins. Our clients expectations required much more time than we had in the semester, and we had to cut back on what was asked, simply because there was no way to fit it into the project. The underlying algorithm was not terribly difficult, but because none of the team had experience with web development, I feel we spent much more time trying to figure out how to implement it than was necessary.