Report ISYS2120

Assignment 3

SQL Queries

Query (1): Login

```
SELECT *
        FROM mediaserver.UserAccount
        WHERE username = %s AND password = %s
```

Query (1 a): Get User Podcasts

```
SELECT SP.podcast_id, P.podcast_title, P.podcast_uri , P.podcast_last_updated
    FROM mediaserver.subscribed_Podcasts SP JOIN
            mediaserver.podcast P USING(podcast_id)
    WHERE SP.username =%s
```

Query (1 b): Get User Playlists

```
SELECT collection_id, collection_name, COUNT(MCC.media_id)
   From mediaserver.mediaCollection MC JOIN
            mediaserver.mediacollectioncontents MCC USING (collection_id)
   WHERE MC.username =%s
   GROUP BY collection_id
   ORDER BY collection_id
```

Query (1 c): Get User in Progress Items

```
SELECT MI.media_id,  UMC.play_count AS playcount, UMC.progress, UMC.lastviewed
, MI.storage_location
    FROM mediaserver.usermediaconsumption UMC JOIN
            mediaserver.mediaitem MI USING (media_id)
    WHERE username=%s AND progress < 100 AND progress > 0
```

Query (2 a,b,c): Get One song

```
Select s.song_id, s.song_title, string_agg(a.artist_name,',') artists,s.length
    From mediaserver.song s
    Join mediaserver.song_artists sa using (song_id)
    Join mediaserver.artist a
    On (sa.performing_artist_id = a.artist_id)
    WHERE S.song_id=%s
    Group by s.song_id, s.song_title, s.length
    Order by s.song_id;
```

Query (2 d): Get Metadata for One Song

```
SELECT s.song_id, mdt.md_type_name, md.md_value
    From mediaserver.song s
    Join mediaserver.mediaitemmetadata mimd
    On (s.song_id = mimd.media_id)
    join mediaserver.MetaData md using (md_id)
    join mediaserver.MetaDataType mdt using (md_type_id)
    WHERE S.song_id=%s;
```

Query (3 a,b c): Get all TV Shows

```
SELECT tvs.tvshow_id, tvs.tvshow_title, COUNT(tve.episode)
    FROM mediaserver.tvshow tvs JOIN
        mediaserver.tvepisode tve USING (tvshow_id)
    GROUP BY tvs.tvshow_id
    ORDER BY tvs.tvshow_id
```

Query (4 a): Get One TV Show

```
Select tvs.tvshow_id, tvs.tvshow_title
 From mediaserver.tvshow tvs
Where tvs.tvshow_id =%s;
```

Query (4 b): Get Metadata for TV Show

```
select tvsmd.tvshow_id, mdt.md_type_name, md.md_value
    from mediaserver.TVShowMetaData tvsmd
    join mediaserver.metadata md using (md_id)
    join mediaserver.MetaDataType mdt using (md_type_id)
    where tvsmd.tvshow_id = %s;
```

Query (4 c): Get All TV Show Episodes for One TV Show

```
Select media_id, tvshow_episode_title, season, episode, air_date
    From mediaserver.TVEpisode
    Where tvshow_id = %s
    Order by season, episode;
```

Query (5 a,b): Get One Album

```
SELECT a.album_id, a.album_title
    FROM mediaserver.album a
    WHERE album_id=%s
```

Query (5 c): Get All Songs for One Album

```
Select song_id, song_title, artist_name
    From mediaserver.Song JOIN mediaserver.Album_Songs USING (song_id) JOIN
mediaserver.Song_Artists USING (song_id) JOIN mediaserver.Artist on (performing_artist_id =
artist_id)
    Where album_id = %s
    Order By track_num;
```

Query (6): Get All Genres for One Album

```
Select distinct album_id, md.md_value as songgenres
    From mediaserver.Album_Songs a2s
    Join mediaserver.MediaItemMetaData mimd on (a2s.song_id = mimd.media_id)
    Join mediaserver.MetaData md Using (md_id)
    where md. md_type_id = 1 and album_id = %s
    order by songgenres
```

Query (7 a,b,c,d,e): Get One Podcast and Return All Metadata Associated With It

```
SELECT *
    FROM mediaserver.podcast P
    NATURAL JOIN mediaserver.podcastmetadata PM
    JOIN mediaserver.metadata M USING (md_id)
    NATURAL JOIN mediaserver.metadatatype MDT
    WHERE P.podcast_id=%s
```

Query (7 f): Get all Podcast Episodes for One Podcast

```
SELECT media_id, podcast_episode_title, podcast_episode_URI,
podcast_episode_published_date, podcast_episode_length
    FROM  mediaserver.podcast NATURAL JOIN mediaserver.podcastepisode
    WHERE podcast_id =%s
    ORDER BY podcast_episode_published_date;
```

Query (8 a,b,c,d,e,f): Get One Podcast Episode and Associated Metadata

```
SELECT media_id, podcast_episode_title, podcast_episode_uri,
podcast_episode_published_date, podcast_episode_length, md_value, md_type_name
    FROM mediaserver.podcast P NATURAL JOIN
        mediaserver.podcastepisode PE NATURAL JOIN
        mediaserver.podcastmetadata PMD NATURAL JOIN
        mediaserver.metadata MD NATURAL JOIN
        mediaserver.metadatatype
    WHERE PE.media_id =%s
```

Query (9): Add a New Song

```
SELECT
        mediaserver.addSong(
            %s,%s,%s,%s,%s,%s);
```

get_last_song() was created in *database.py* to return the latest *song_id*, so that when a user adds a new song, the app knows to redirect the webpage to the latest added song. This can be seen below:

```
select max(song_id) as song_id from mediaserver.song
```

Query (10): Find All Matching Movies

```
Select m.movie_id, m.movie_title, m.release_year
    From mediaserver.Movie m
    Where lower(m.movie_title) ~ lower(%s);
```

Added Functionality

New SQL Functions in '*database.py*':

```python
def find_matchingpodcasts(searchterm):
    """
    Get all the matching Albums in your media server
    """

    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()
    try:
        sql = """
        Select p.podcast_id, p.podcast_title
        From mediaserver.Podcast p
        Where lower(p.podcast_title) ~ lower(%s);
        """

        r = dictfetchall(cur,sql,(searchterm,))
        print("return val is:")
        print(r)
        cur.close()                     # Close the cursor
        conn.close()                    # Close the connection to the db
        return r
    except:
        # If there were any errors, return a NULL row printing an error to the debug
        print("Unexpected error getting All Podcasts:", sys.exc_info()[0])
        raise
    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return None
```

The above screenshot displays the code that allows the user to search for a podcast by providing a keyword to the search bar. As seen above, the query returns all podcasts with titles that contain the given keyword, with this query being case insensitive.

```python
def find_multimovies(searchterm, searchyear):
    """
    Get all the matching Movies in your media server
    """

    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()
    try:
        #########
        # TODO  #
        #########

        ##############################################################################
        # Fill in the SQL below with a query to get all information about movies    #
        # that match a given search term                                             #
        ##############################################################################
        sql = """
        Select m.movie_id, m.movie_title, m.release_year
        From mediaserver.Movie m
        Where (lower(m.movie_title) ~ lower(%s)) AND (m.release_year >= %s)
        order by m.release_year desc;
        """

        r = dictfetchall(cur,sql,(searchterm, searchyear))
        print("return val is:")
        print(r)
        cur.close()                     # Close the cursor
        conn.close()                    # Close the connection to the db
        return r
    except:
        # If there were any errors, return a NULL row printing an error to the debug
        print("Unexpected error getting All Movies:", sys.exc_info()[0])
        raise
    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return None
```

The above screenshot illustrates the query that outputs a list of movies within the database that corresponds to a title and release year provided by the user. Similar to the above feature, this query utilises both keywords to traverse through the database and return a result. Once again, this query is case insensitive, and returns corresponding movies despite case.

```python
##################################################
def find_matchingartists(searchterm):
    """
    Get all the matching Artists in your media server
    """

    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()
    try:
        sql = """
        Select a.artist_id, a.artist_name
        From mediaserver.Artist a
        Where lower(a.artist_name) ~ lower(%s);
        """

        r = dictfetchall(cur,sql,(searchterm,))
        print("return val is:")
        print(r)
        cur.close()                     # Close the cursor
        conn.close()                    # Close the connection to the db
        return r
    except:
        # If there were any errors, return a NULL row printing an error to the debug
        print("Unexpected error getting All Artists:", sys.exc_info()[0])
        raise
    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return None
```

The above screenshot shows the implementation of the Artist search, which displays a list of Artists whose name corresponds to the keyword given by the user. Once again, the query uses the provided keyword to search through the database and returns a result.

```python
def find_matchingalbums(searchterm):
    """
    Get all the matching Albums in your media server
    """

    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()
    try:
        sql = """
        Select al.album_id, al.album_title
        From mediaserver.Album al
        Where lower(al.album_title) ~ lower(%s);
        """

        r = dictfetchall(cur,sql,(searchterm,))
        print("return val is:")
        print(r)
        cur.close()                     # Close the cursor
        conn.close()                    # Close the connection to the db
        return r
    except:
        # If there were any errors, return a NULL row printing an error to the debug
        print("Unexpected error getting All Albums:", sys.exc_info()[0])
        raise
    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return None
```

Screenshot shows the same functionality as the one above, however, searches through the database's albums and their titles.

```python
#####################################################
#   Update password
#####################################################
def update_password(new_password, username):
    """
    Updates a users password
    """
    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()
    try:
        # Try executing the SQL
        sql = """
        UPDATE mediaserver.UserAccount
        SET password=%s
        WHERE username=%s
        """
        cur.execute(sql,(new_password, username))
        r = cur
        print("return val is:")
        print(r)

        cur.close()                     # Close the cursor
        conn.commit()
        conn.close()                    # Close the connection to the db
        return r
    except:
        # If there were any errors, return a NULL row printing an error to the debug
        print("Unexpected error updating password:", sys.exc_info()[0])
        raise
    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return None
```

The above code segment shows the query used to change a password depending on the username of the user. As seen in the screenshot, the query accepts the username of the user and their new password and updates the database accordingly.

```python
#####################################################
#   Get user contact details
#####################################################
def get_user_contact_details(username):
    """
    Get user contact details
    """

    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()
    try:
        sql = """
        SELECT contact_type_value, contact_type_name
        FROM mediaserver.UserAccount NATURAL JOIN mediaserver.contactmethod
        NATURAL JOIN mediaserver.contacttype
        WHERE username=%s
        """

        r = dictfetchall(cur,sql,(username,))
        print("return val is:")
        print(r)
        cur.close()                     # Close the cursor
        conn.close()                    # Close the connection to the db
        return r
    except:
        # If there were any errors, return a NULL row printing an error to the debug
        print("Unexpected error getting contact details:", sys.exc_info()[0])
    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return None
```

The above query is used to get a specific user's contact details in order to allow for it to update, as well as to be displayed on the website's homepage. As seen above, the query uses the username of the user in order to obtain the corresponding contact information.

```python
#####################################################
#   Update email
#####################################################
def update_email(new_email, username):
    """
    Updates a users password
    """
    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()
    try:
        # Try executing the SQL
        sql = """
        UPDATE mediaserver.contactmethod
         SET contact_type_value =%s
        WHERE username = %s
          AND contact_type_id =
              (select contact_type_id
                from mediaserver.contacttype
                where contact_type_name = 'email')
        """
        cur.execute(sql,(new_email, username))
        r = cur
        print("return val is:")
        print(r)

        cur.close()                     # Close the cursor
        conn.commit()
        conn.close()                    # Close the connection to the db
        return r
    except:
        # If there were any errors, return a NULL row printing an error to the debug
        print("Unexpected error updating email:", sys.exc_info()[0])
        raise
    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
```

The above screenshot shows the SQL query which updates the user's email. Utilising the user's username, the query locates the user's contact information and changes their current email with the new one provided by the user.

```python
##################################################
#   Update phone no
##################################################
def update_phone(new_phone, username):
    """
    Updates a users phone no.
    """
    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()
    try:
        # Try executing the SQL
        sql = """
        UPDATE mediaserver.contactmethod
         SET contact_type_value =%s
        WHERE username = %s
          AND contact_type_id =
              (select contact_type_id
                from mediaserver.contacttype
                where contact_type_name = 'phone')
        """
        cur.execute(sql,(new_phone, username))
        r = cur
        print("return val is:")
        print(r)

        cur.close()                     # Close the cursor
        conn.commit()
        conn.close()                    # Close the connection to the db
        return r
    except:
        # If there were any errors, return a NULL row printing an error to the debug
        print("Unexpected error updating phone no.:", sys.exc_info()[0])
        raise
    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return None
```

Similar to the query above, however, updates the user's phone number to the new number provided.

New Routing Functions in *'routes.py'*:

```python
@app.route('/search/podcast', methods=['POST','GET'])
def search_podcasts():
    """
    Search all the PODCASTS in your media server
    """
    # Check if the user is logged in, if not: back to login.
    if('logged_in' not in session or not session['logged_in']):
        return redirect(url_for('login'))

    page['title'] = 'Podcast Search' # Add the title

    # Get a list of matching tv shows from the database
    podcasts = None
    if request.method == 'POST':
        # Set up some variables to manage the post returns

        # Once retrieved, do some data integrity checks on the data

        podcasts = database.find_matchingpodcasts(request.form['searchterm'])

        # Once verified, send the appropriate data to

    # Data integrity checks
    if podcasts == None or podcasts == []:
        podcasts = []
        page['bar'] = False
        flash("No matching albums found, please try again")
    else:
        page['bar'] = True
        flash('Found '+str(len(podcasts))+' results!')
        session['logged_in'] = True

    # NOTE :: YOU WILL NEED TO MODIFY THIS TO PASS THE APPROPRIATE VARIABLES
    return render_template('searchitems/search_podcasts.html',
                           session=session,
                           page=page,
                           user=user_details,
                           podcasts = podcasts)
```

```python
@app.route('/search/multi_movies', methods=['POST','GET'])
def multi_movies():
    """
    Search all the movies in your media server
    """
    # Check if the user is logged in, if not: back to login.
    if('logged_in' not in session or not session['logged_in']):
        return redirect(url_for('login'))

    #########
    # TODO  #
    #########

    ##############################################################################
    # Fill in the Function below with to do all data handling for searching for #
    # a movie                                                                    #
    ##############################################################################

    page['title'] = 'Movie Search' # Add the title

    # Get a list of matching tv shows from the database
    movies = None
    if request.method == 'POST':
        # Set up some variables to manage the post returns

        # Once retrieved, do some data integrity checks on the data

        movies = database.find_multimovies(request.form['searchterm'], request.form['searchyear'])

        # Once verified, send the appropriate data to

    # Data integrity checks
    if movies == None or movies == []:
        movies = []
        page['bar'] = False
        flash("No matching movies found, please try again")
    else:
        page['bar'] = True
        flash('Found '+str(len(movies))+' results!')
        session['logged_in'] = True

    # NOTE :: YOU WILL NEED TO MODIFY THIS TO PASS THE APPROPRIATE VARIABLES
    return render_template('searchitems/multi_movies.html',
                           session=session,
                           page=page,
                           user=user_details,
                           movies = movies)
```

```python
@app.route('/search/artist', methods=['POST','GET'])
def search_artists():
    """
    Search all the ARTISTS in your media server
    """
    # Check if the user is logged in, if not: back to login.
    if('logged_in' not in session or not session['logged_in']):
        return redirect(url_for('login'))

    page['title'] = 'Artist Search' # Add the title

    # Get a list of matching tv shows from the database
    artists = None
    if request.method == 'POST':
        # Set up some variables to manage the post returns

        # Once retrieved, do some data integrity checks on the data

        artists = database.find_matchingartists(request.form['searchterm'])

        # Once verified, send the appropriate data to

    # Data integrity checks
    if artists == None or artists == []:
        artists = []
        page['bar'] = False
        flash("No matching artists found, please try again")
    else:
        page['bar'] = True
        flash('Found '+str(len(artists))+' results!')
        session['logged_in'] = True

    # NOTE :: YOU WILL NEED TO MODIFY THIS TO PASS THE APPROPRIATE VARIABLES
    return render_template('searchitems/search_artists.html',
                           session=session,
                           page=page,
                           user=user_details,
                           artists = artists)
```

```python
@app.route('/search/album', methods=['POST','GET'])
def search_albums():
    """
    Search all the ALBUMS in your media server
    """
    # Check if the user is logged in, if not: back to login.
    if('logged_in' not in session or not session['logged_in']):
        return redirect(url_for('login'))

    page['title'] = 'Album Search' # Add the title

    # Get a list of matching tv shows from the database
    albums = None
    if request.method == 'POST':
        # Set up some variables to manage the post returns

        # Once retrieved, do some data integrity checks on the data

        albums = database.find_matchingalbums(request.form['searchterm'])

        # Once verified, send the appropriate data to

    # Data integrity checks
    if albums == None or albums == []:
        albums = []
        page['bar'] = False
        flash("No matching albums found, please try again")
    else:
        page['bar'] = True
        flash('Found '+str(len(albums))+' results!')
        session['logged_in'] = True

    # NOTE :: YOU WILL NEED TO MODIFY THIS TO PASS THE APPROPRIATE VARIABLES
    return render_template('searchitems/search_albums.html',
                           session=session,
                           page=page,
                           user=user_details,
                           albums = albums)
```

```python
@app.route('/singleitems/settings', methods=['POST','GET'])
def setting_functions():
    """
    Update user password and contact details
    """
    # Check if the user is logged in, if not: back to login.
    if('logged_in' not in session or not session['logged_in']):
        return redirect(url_for('login'))

    page['title'] = 'Settings' # Add the title

    new_password = None
    print("request form is:")
    newdict = {}
    print(request.form)
    if request.method == 'POST':

        if ('new_pass' not in request.form):
            newdict['new_pass'] = user_details['password']
        else:
            newdict['new_pass'] = request.form['new_pass']
            print("We have a value: ",newdict['new_pass'])

        new_password = database.update_password(newdict['new_pass'],user_details['username'])
        page['bar'] = True
        flash('Password Updated Successfully')
        return redirect(url_for('login'))
    else:
        # NOTE :: YOU WILL NEED TO MODIFY THIS TO PASS THE APPROPRIATE VARIABLES
        return render_template('singleitems/settings.html',
                               session=session,
                               page=page,
                               user=user_details)
```

```python
@app.route('/singleitems/settings', methods=['POST','GET'])
def setting_functions():
    """
    Update user password and contact details
    """
    # Check if the user is logged in, if not: back to login.
    if('logged_in' not in session or not session['logged_in']):
        return redirect(url_for('login'))

    page['title'] = 'Settings' # Add the title

    new_password = None
    print("request form is:")
    newdict = {}
    print(request.form)
    if request.method == 'POST':

        if (request.form['new_pass'] == ""):
            pass
        else:
            newdict['new_pass'] = request.form['new_pass']
            print("We have a value: ",newdict['new_pass'])
            new_password = database.update_password(newdict['new_pass'],user_details['username'])

        if (request.form['new_email'] == ""):
            pass
        else:
            newdict['new_email'] = request.form['new_email']
            print("We have a value: ",newdict['new_email'])
            new_email = database.update_email(newdict['new_email'],user_details['username'])

        if (request.form['new_phone'] == ""):
            pass
        else:
            newdict['new_phone'] = request.form['new_phone']
            print("We have a value: ",newdict['new_phone'])
            new_phone = database.update_phone(newdict['new_phone'],user_details['username'])


        page['bar'] = True
        flash('Updated Successfully')
        return redirect(url_for('login'))
    else:
        # NOTE :: YOU WILL NEED TO MODIFY THIS TO PASS THE APPROPRIATE VARIABLES
        return render_template('singleitems/settings.html',
                               session=session,
                               page=page,
                               user=user_details)
```

New Templates in *‘templates/*.html’:*

```html
{% include 'top.html' %}
    <div class="content">
        <h1 class="title">Search Artists</h1>

        <!-- Query 10
            Search for Songs
            You need two sections
            First is a form to take in input
            Second is a table to display reults
         -->
         <form class="Search" method="POST" action="{{url_for('search_artists')}}">
             <input type="text" name="searchterm" placeholder="Artist" autofocus required>
             <button class="flat" type="submit">Search</button>
         </form>
             {% if artists | length > 0 %}
                 <!-- All Songs -->
                 <table class="styled">
                     <thead>
                         <tr>
                             <td>Artist ID</td>
                             <td>Artist Name</td>
                         </tr>
                     </thead>
                     <tbody>
                         {% for instance in artists %}
                             <!-- Each row is a link to each individual song page -->
                             <tr class="clickable-tr" data-href="{{ url_for('single_artist', artist_id=instance.artist_id)}}">
                                 <td style="text-align: center">{{ instance.artist_id }}</td>
                                 <td>{{instance.artist_name}}</td>
                             </tr>
                         {% endfor %}
                     </tbody>

                 </table>
             {% endif %}

    </div>
{% include 'bottom.html'%}
```

```html
{% include 'top.html' %}
    <div class="content">
        <h1 class="title">Search Albums</h1>

        <!-- Query 10
            Search for Albums
            You need two sections
            First is a form to take in input
            Second is a table to display reults
         -->
         <form class="Search" method="POST" action="{{url_for('search_albums')}}">
             <input type="text" name="searchterm" placeholder="Song" autofocus required>
             <button class="flat" type="submit">Search</button>
         </form>
             {% if albums | length > 0 %}
                 <!-- All Songs -->
                 <table class="styled">
                     <thead>
                         <tr>
                             <td>Album ID</td>
                             <td>Album Title</td>
                         </tr>
                     </thead>
                     <tbody>
                         {% for instance in albums %}
                             <!-- Each row is a link to each individual song page -->
                             <tr class="clickable-tr" data-href="{{ url_for('single_album', album_id=instance.album_id)}}">
                                 <td style="text-align: center">{{ instance.album_id }}</td>
                                 <td>{{instance.album_title}}</td>
                             </tr>
                         {% endfor %}
                     </tbody>

                 </table>
             {% endif %}

    </div>
{% include 'bottom.html'%}
```

```html
{% include 'top.html' %}
    <div class="content">
        <h1 class="title">Movie Search Multi</h1>

        <!-- Query 10
            Search for Movies
            You need two sections
            First is a form to take in input
            Second is a table to display reults
         -->
         <form class="Search" method="POST" action="{{url_for('multi_movies')}}">
             <input type="text" name="searchterm" placeholder="Movie Title" autofocus required>
             <select name="searchyear">
                 <option value="0" selected>Please select a year to search from...</option>
                 <option value="2018">2018</option>
                 <option value="2017">2017</option>
                 <option value="2016">2016</option>
                 <option value="2015">2015</option>
                 <option value="2014">2014</option>
                 <option value="2013">2013</option>
                 <option value="2012">2012</option>
                 <option value="2011">2011</option>
                 <option value="2010">2010</option>
                 <option value="2009">2009</option>
                 <option value="2008">2008</option>
                 <option value="2007">2007</option>
                 <option value="2006">2006</option>
                 <option value="2005">2005</option>
                 <option value="2004">2004</option>
                 <option value="2003">2003</option>
                 <option value="2002">2002</option>
                 <option value="2001">2001</option>
                 <option value="2000">2000</option>
                 <option value="1999">1999</option>
                 <option value="1998">1998</option>
                 <option value="1997">1997</option>
                 <option value="1996">1996</option>
                 <option value="1995">1995</option>
                 <option value="1994">1994</option>
                 <option value="1993">1993</option>
                 <option value="1992">1992</option>
                 <option value="1991">1991</option>
                 <option value="1990">1990</option>
                 <option value="1989">1989</option>
                 <option value="1988">1988</option>
                 <option value="1987">1987</option>
                 <option value="1986">1986</option>
                 <option value="1985">1985</option>
                 <option value="1984">1984</option>
                 <option value="1983">1983</option>
                 <option value="1982">1982</option>
                 <option value="1981">1981</option>
                 <option value="1980">1980</option>
                 <option value="1979">1979</option>
                 <option value="1978">1978</option>
                 <option value="1977">1977</option>
                 <option value="1976">1976</option>
                 <option value="1975">1975</option>
                 <option value="1974">1974</option>
                 <option value="1973">1973</option>
                 <option value="1972">1972</option>
```

```
                <option value="1935">1935</option>
                <option value="1934">1934</option>
                <option value="1933">1933</option>
                <option value="1932">1932</option>
                <option value="1931">1931</option>
                <option value="1930">1930</option>
                <option value="1929">1929</option>
                <option value="1928">1928</option>
                <option value="1927">1927</option>
                <option value="1926">1926</option>
                <option value="1925">1925</option>
                <option value="1924">1924</option>
                <option value="1923">1923</option>
                <option value="1922">1922</option>
                <option value="1921">1921</option>
                <option value="1920">1920</option>
                <option value="1919">1919</option>
                <option value="1918">1918</option>
                <option value="1917">1917</option>
                <option value="1916">1916</option>
                <option value="1915">1915</option>
                <option value="1914">1914</option>
                <option value="1913">1913</option>
                <option value="1912">1912</option>
                <option value="1911">1911</option>
                <option value="1910">1910</option>
                <option value="1909">1909</option>
                <option value="1908">1908</option>
                <option value="1907">1907</option>
                <option value="1906">1906</option>
                <option value="1905">1905</option>
            </select>
            <button class="flat" type="submit">Search</button>
        </form>
            {% if movies | length > 0 %}
                <!-- All Movies -->
                <table class="styled">
                    <thead>
                        <tr>
                            <td>Movie ID</td>
                            <td>Movie Title</td>
                            <td>Release Year</td>
                        </tr>
                    </thead>
                    <tbody>
                        {% for instance in movies %}
                            <!-- Each row is a link to each individual movie page -->
                            <tr class="clickable-tr" data-href="{{ url_for('single_movie', movie_id=instance.movie_id)}}">
                                <td style="text-align: center">{{ instance.movie_id }}</td>
                                <td>{{instance.movie_title}}</td>
                                <td>{{instance.release_year}}
                            </tr>
                        {% endfor %}
                    </tbody>

                </table>
            {% endif %}

    </div>
{% include 'bottom.html'%}
```

```html
{% include 'top.html' %}
<div class="content">
    <div class="container details">

        <h2 class="title">Settings</h2>

        <form class="Add" method="POST" action="{{url_for('setting_functions')}}">
                <h3> Change Password: </h3>
                New Password: <input type="text" name="new_pass" placeholder="Input New Password">
                <br/>
                <br/>
                <h3> Update Contact Details: </h3>
                Email: <input type="text" name="new_email" placeholder="Input New Email">
                Phone: <input type="text" name="new_phone" placeholder="Input New Phone No.">
                <br/>
                <br/>
                <button class="flat" type="submit">Update</button>
            </form>

    </div>
</div>
{% include 'bottom.html'%}
```

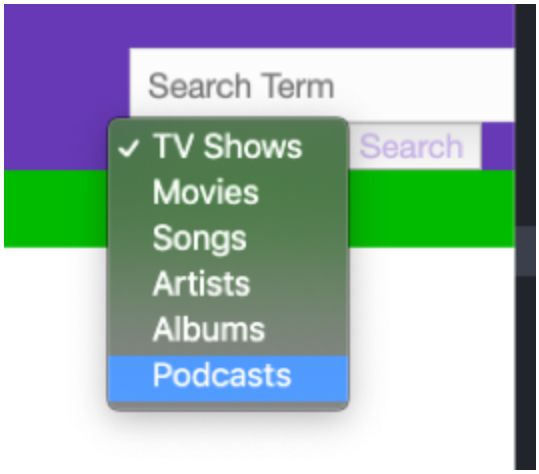
```html
{% if session.logged_in %}
    <li><a href="{{ url_for('setting_functions') }}">Settings</a></li>
    <li><a href="{{ url_for('logout') }}">Logout</a></li>
    <li></li>
{% endif %}
```

Modifications:

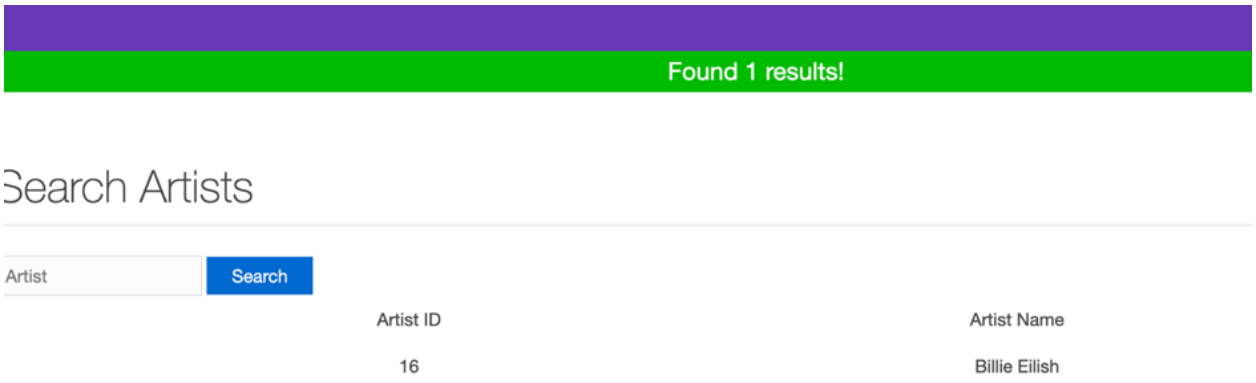
The added functionality that was created included the addition of search options for the categories TV Shows, Artists, Albums, Podcasts, as well as a multi-variable search for Movies. This feature allows a user to filter movies by a specific release date. Additionally, another feature was added which allows the user to change their password and their contact details.

Additional Search Features:

The TV Show, Artist, Album, and Podcast search all work similarly to the search page provided, in which the user selects option from the dropdown menu and enter a keyword. Utilising this keyword, the SQL query will then extract data from the database, causing the HTML to output the result.

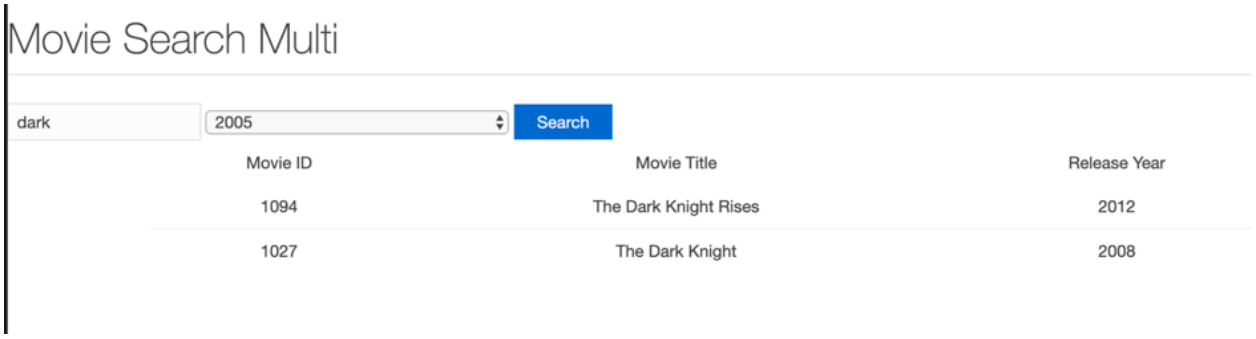


Once an option from the dropdown box has been selected, the user is taken to its corresponding page, in which you can click on a Podcast, Album, or TV Show included in the search result and view further information.

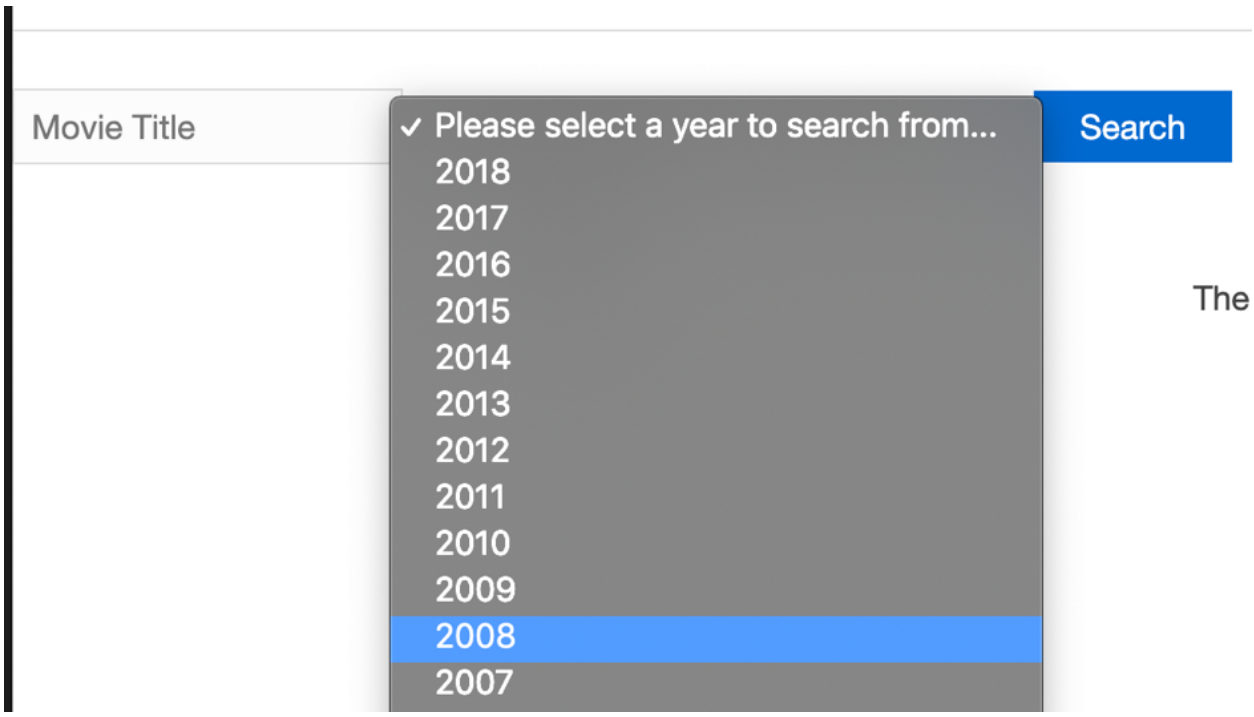


Above image displays the output given when searching for the keyword "Billie".

The movie-year search is a separate HTML page with a link at the top of the index, with a dropdown menu with a range of years to search from.



The above image shows the results after searching for Movies released between 2005-2019.



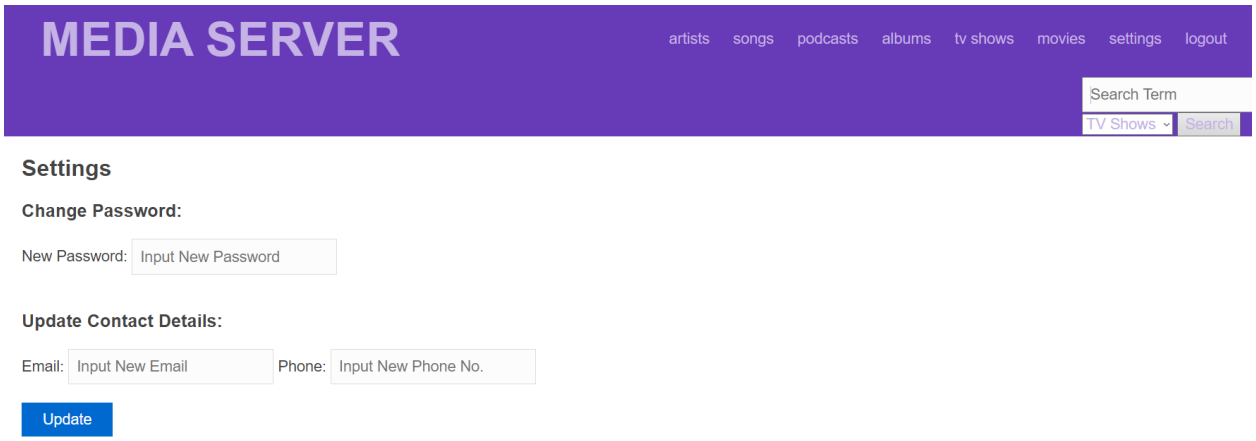
The above image displays the dropbox and the options to search for the Movie release year.

Change User Contact Information and Password Feature:

As an additional feature to the website, the user contact information can be altered by the user if they have successfully logged into the website. This information is displayed when the user first logs into their account.

Contact details displayed on the homepage after user logs in

Additionally, a new option was also added to the navigation located at the top of the webpage named *"settings"*. The screenshot below displays the page the user is taken to once this category is selected:



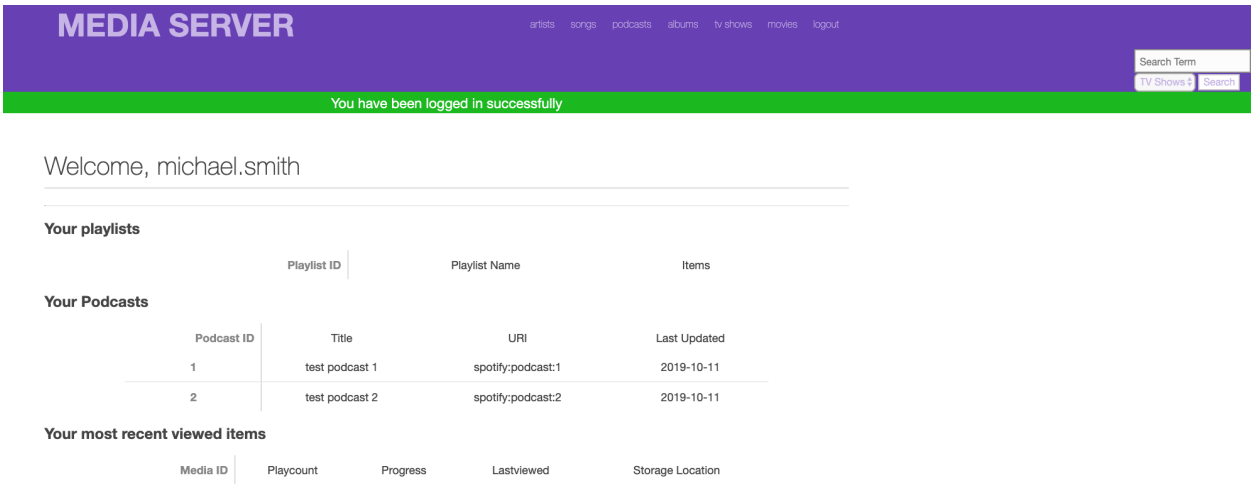
Settings Page

As seen above, text boxes are provided to the user, prompting them to enter new contact information or password. Once the button "Update" is selected, the given information is used to update the user's data in the database.

Screenshots:

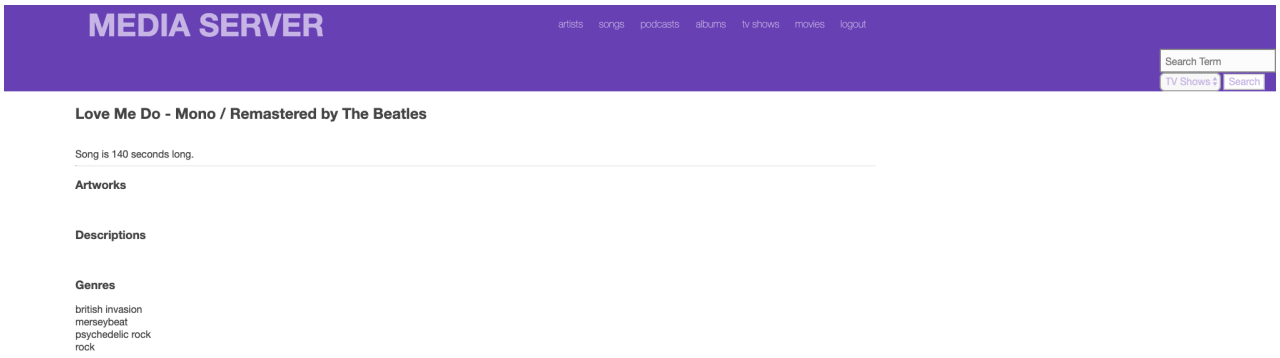
A user logs into the system by providing their username and password. Once logged in, they reach the landing page which displays:

- User subscribed Podcasts
- User Playlists
- User current in-progress items



When a user clicks on a song, they should see all the song information, including:

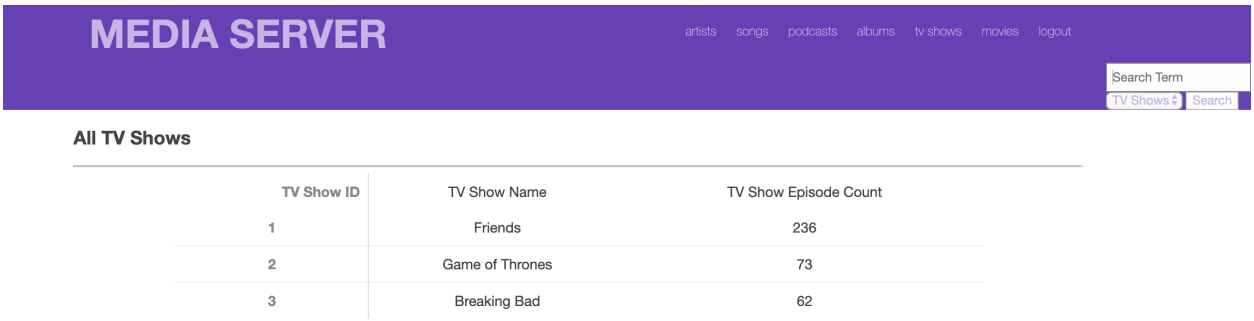
- Song Name
- Song Artists
- Song length
- Song Metadata such as Artwork, description, Genres



When a user clicks on the ‘TV shows’ navigation item, they should see a list of all tv shows information, including:

- TV Show ID

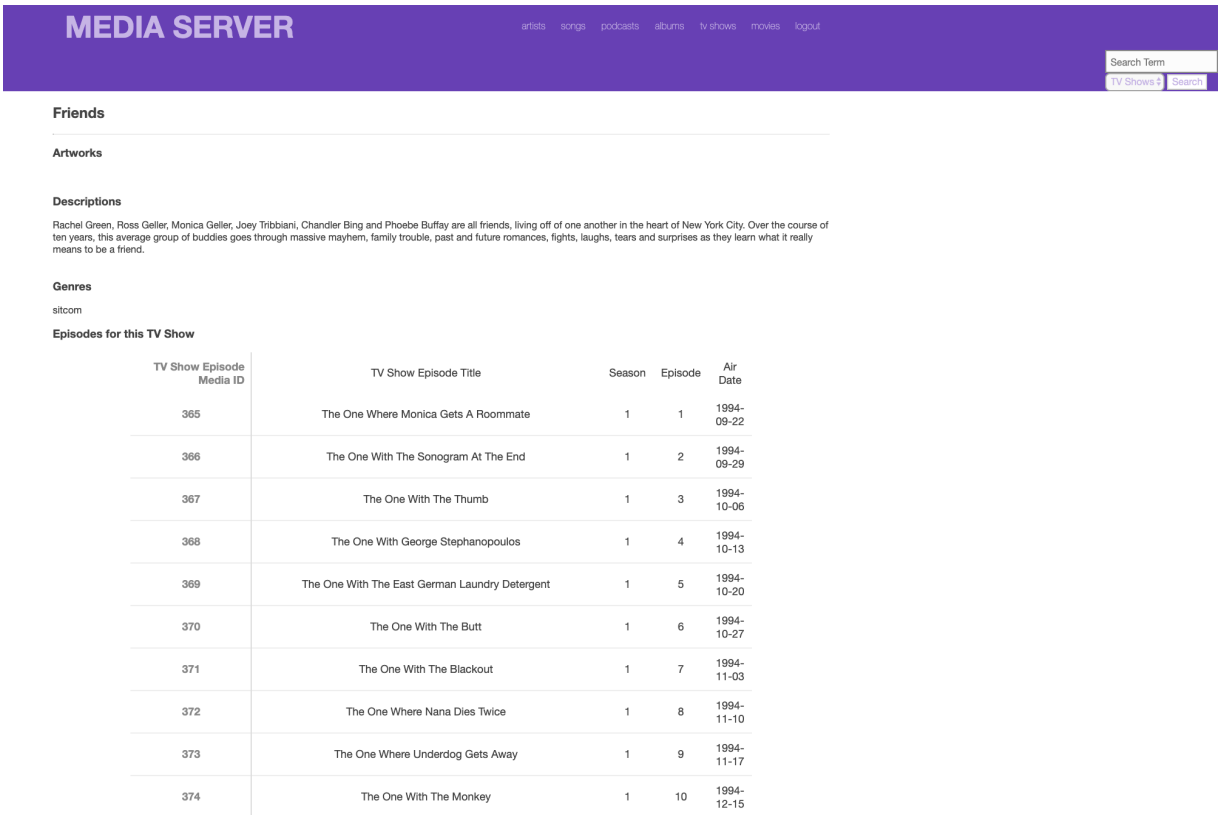
- TV Show Name
- Total number of TV show episodes for this TV show



TV Show ID	TV Show Name	TV Show Episode Count
1	Friends	236
2	Game of Thrones	73
3	Breaking Bad	62

When a user clicks on a single TV Show, they should see a list of relevant information, including:

- TV Show Name
- TV Show Metadata such as Artworks, Descriptions, Genres
- A list of every episode for this tv show ordered by Season and then Episode including:
    - TV Show Episode ID
    - TV Show Episode Title
    - Season
    - Episode
    - AirDate



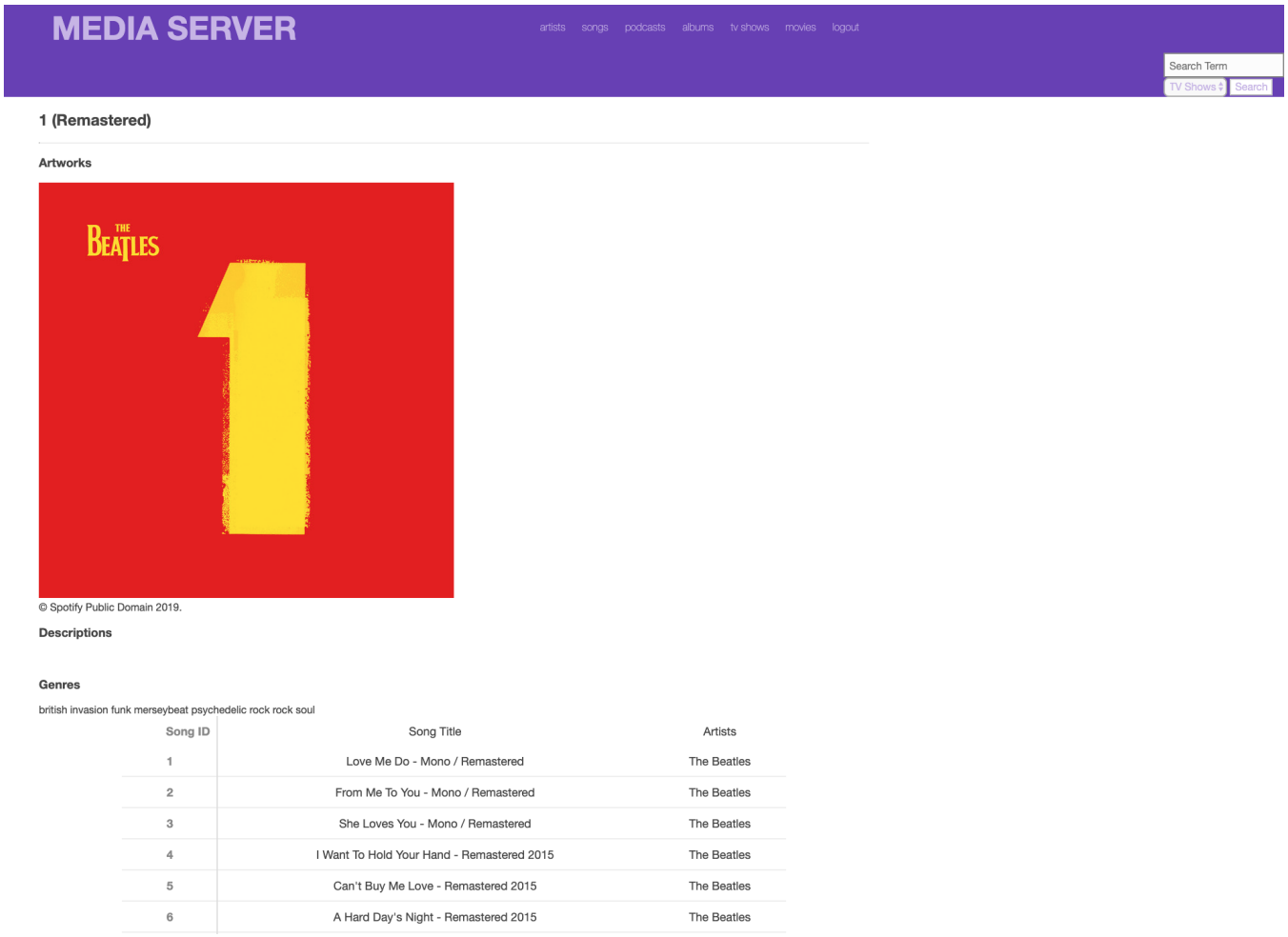
TV Show Episode Media ID	TV Show Episode Title	Season	Episode	Air Date
365	The One Where Monica Gets A Roommate	1	1	1994-09-22
366	The One With The Sonogram At The End	1	2	1994-09-29
367	The One With The Thumb	1	3	1994-10-06
368	The One With George Stephanopoulos	1	4	1994-10-13
369	The One With The East German Laundry Detergent	1	5	1994-10-20
370	The One With The Butt	1	6	1994-10-27
371	The One With The Blackout	1	7	1994-11-03
372	The One Where Nana Dies Twice	1	8	1994-11-10
373	The One Where Underdog Gets Away	1	9	1994-11-17
374	The One With The Monkey	1	10	1994-12-15

When a user clicks on a single Album, they should see a list of relevant information, including:

- Album Name

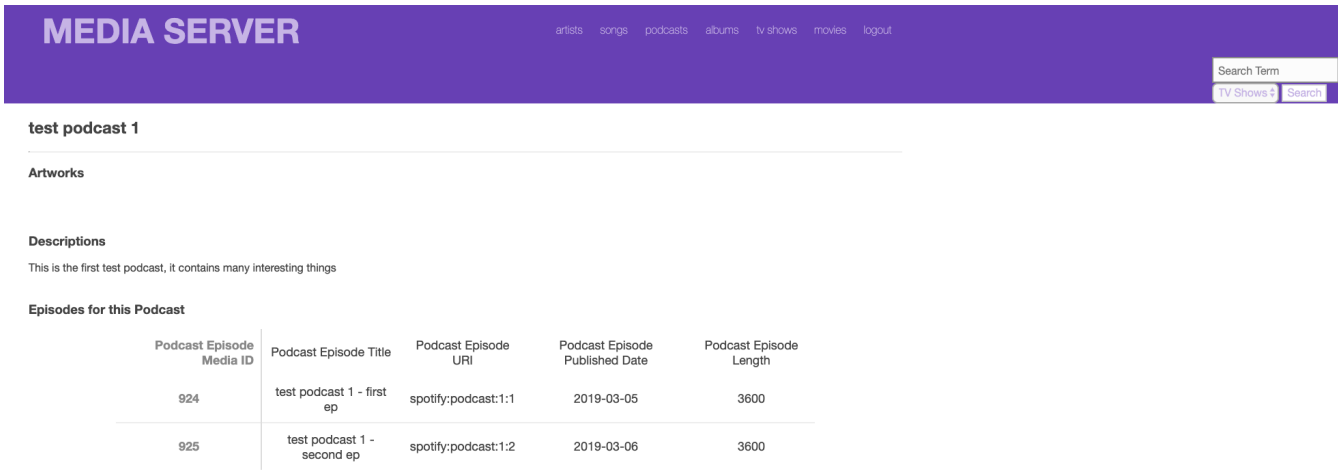
- Album Metadata such as Artworks, Descriptions
- A list of all songs in an album ordered by track number including:
    - Song ID
    - Song Name
    - Song Artist(s)

Users should also be able to see all genres for an Album in part (5). The Genres for an album are composed of all the Genres for it's songs.



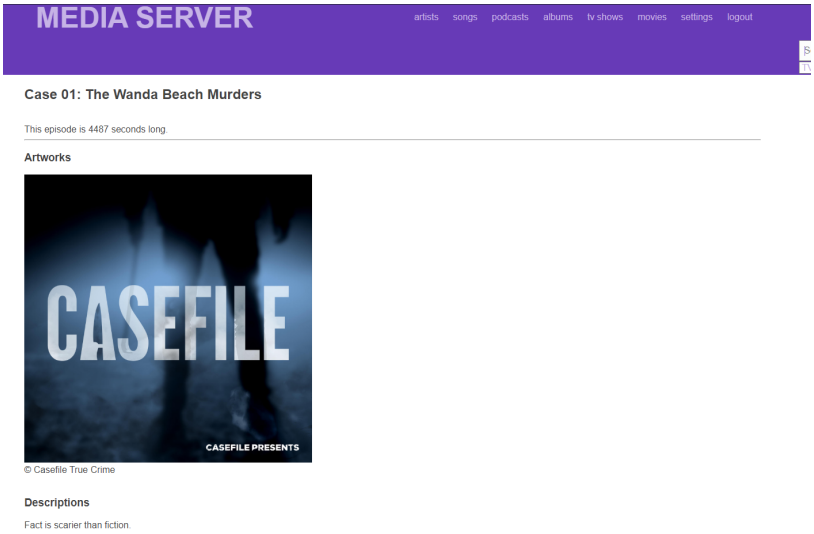
When a user clicks on a single podcast from the 'podcasts' list (or their subscribed podcasts), they should see a list of relevant information including:

- Podcast ID
- Podcast Name
- Podcast URI
- Last Updated
- Podcast Metadata such as Artworks, Descriptions, Genres
- A list of all podcast episodes in this podcast ordered by descending publication date including:
    - Podcast Episode ID
    - Podcast Episode Title
    - Podcast Episode URI
    - Podcast Episode Date Published
    - Podcast Episode Length



When A user clicks on a single podcast episode from the list in part 7, they should see all relevant podcast episode information including:

- Podcast Episode ID
- Podcast Episode Title
- Podcast Episode URI
- Podcast Episode Date Published
- Podcast Episode Length
- Podcast Episode Metadata such as Artworks, Descriptions, Genres



Write the SQL to ensure the proper insert of a new Song, including valid artist checks and appropriate MetaData inserts.

Figure 1.1:

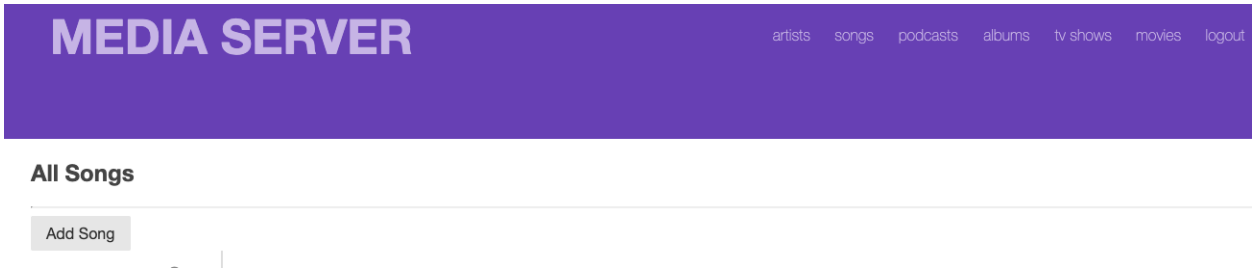


Figure 1.1 shows the navigator to traverse to the "Add Song" page.

Figure 1.2:

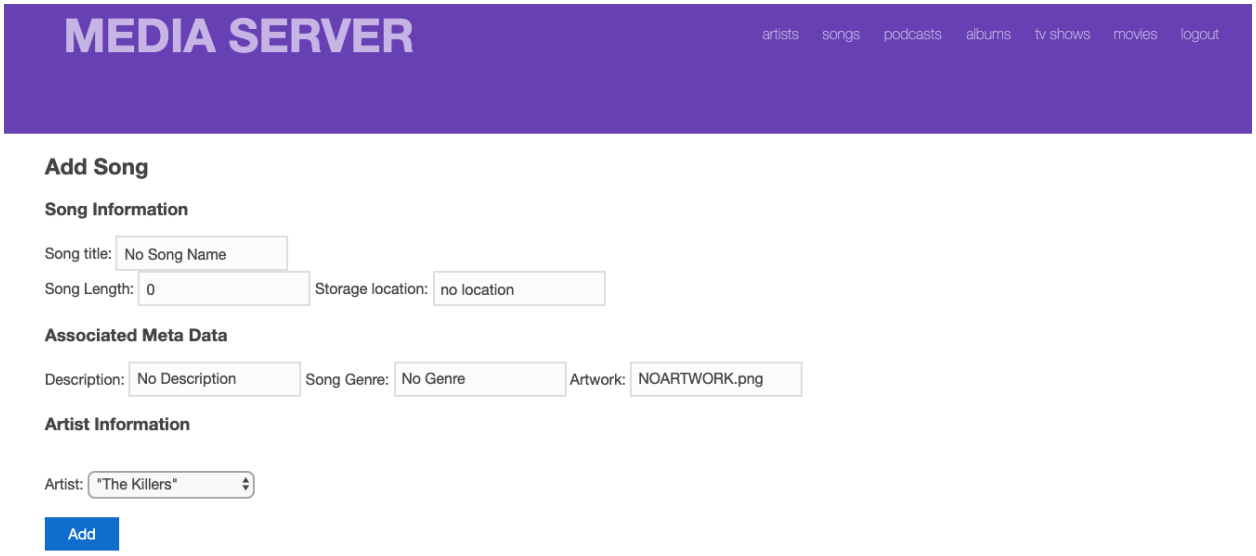


Figure 1.2 shows the display given to the user, prompting them to add information about the song to be added.

Figure 1.3:

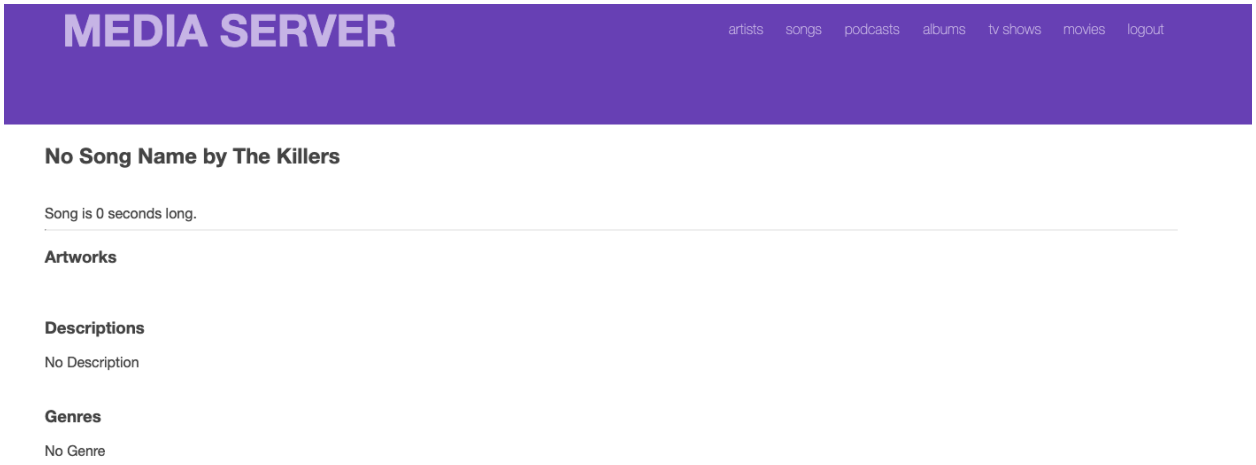
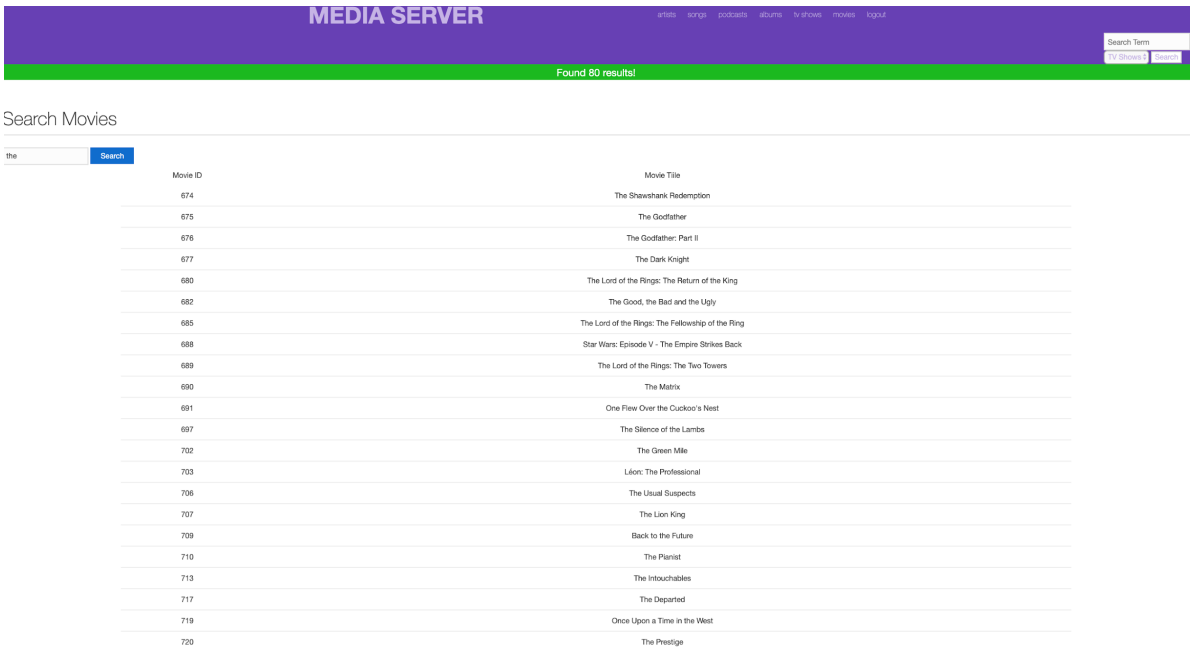


Figure 1.3 displays the output given once a song has been successfully added.

Write SQL for getting all relevant details for searching through all movies by title.

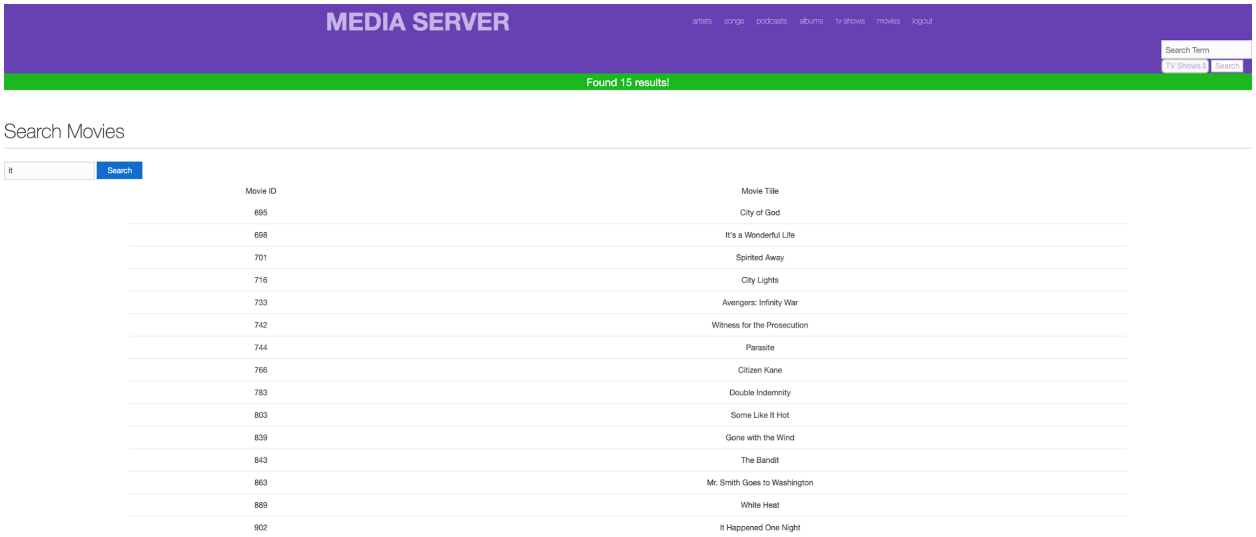
Figure 2.1:



Movie ID	Movie Title
674	The Shawshank Redemption
675	The Godfather
676	The Godfather: Part II
677	The Dark Knight
680	The Lord of the Rings: The Return of the King
682	The Good, the Bad and the Ugly
685	The Lord of the Rings: The Fellowship of the Ring
688	Star Wars: Episode V - The Empire Strikes Back
689	The Lord of the Rings: The Two Towers
690	The Matrix
691	One Flew Over the Cuckoo's Nest
697	The Silence of the Lambs
702	The Green Mile
703	Léon: The Professional
706	The Usual Suspects
707	The Lion King
709	Back to the Future
710	The Pianist
713	The Intouchables
717	The Departed
719	Once Upon a Time in the West
720	The Prestige

Figure 2.1 shows the output given when the keyword “the” was entered into the Movies search bar

Figure 2.2:



Movie ID	Movie Title
695	City of God
698	It's a Wonderful Life
701	Spirited Away
716	City Lights
733	Avengers: Infinity War
742	Witness for the Prosecution
744	Parasite
766	Citizen Kane
783	Double Indemnity
803	Some Like It Hot
839	Gone with the Wind
843	The Bandit
863	Mr. Smith Goes to Washington
889	White Heat
902	It Happened One Night

Figure 2.2 shows *the output given when the keyword "it" was entered into the Movies search bar*