



Indian Institute of Technology, Indore

CS 257 DBIS Project

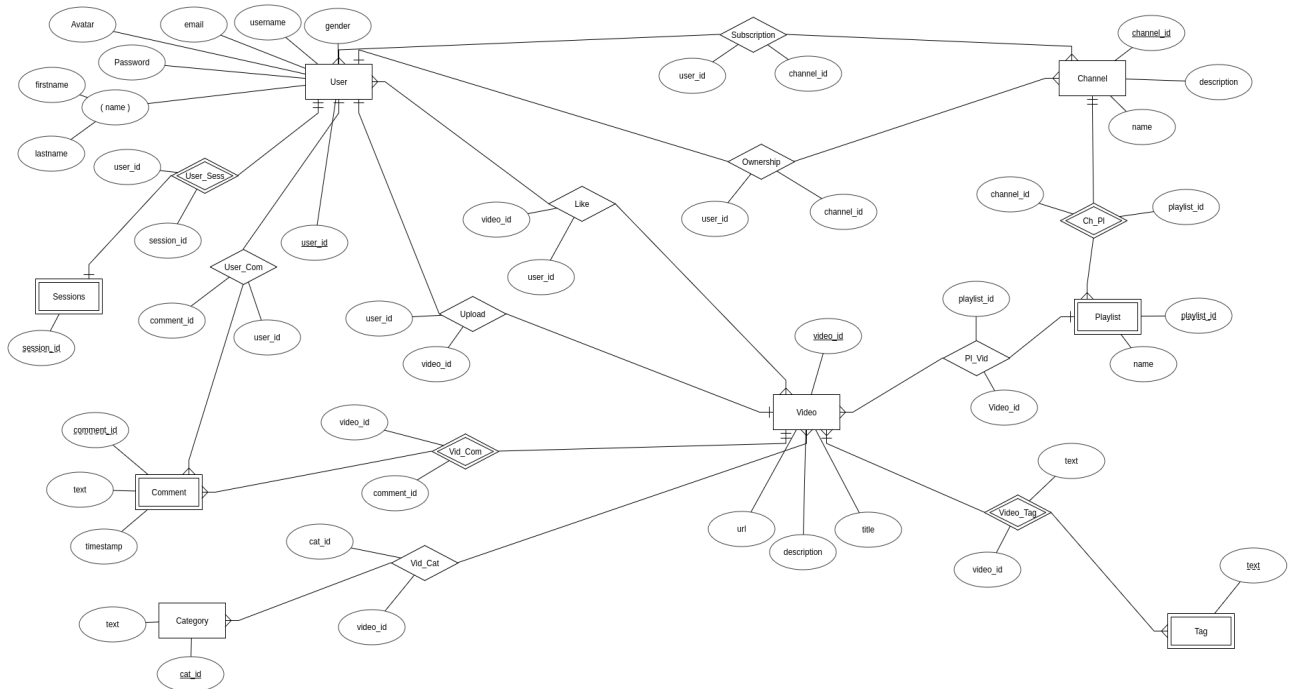
Digital Video Library Management System

**Submitted by- Niranjan Joshi (160001026) and Kanishkar J
(160001028)**

AIM OF THE PROJECT:

The aim of the project is to provide online digital management system where multiple users can upload, access videos, create channels, playlists and like or comment upon the videos. The project can be used for educational or entertainment purpose.

ER DIAGRAM:



ENTITY SET:

- Video
- Channel
- Category
- User
- Tag: Weak entity set Existentially Dependent on Video
- Comments: Weak entity set existentially Dependent on Video
- Playlist: Weak entity set Existentially Dependent on Channel

ENTITY RELATIONSHIPS:

- Subscription (User-Channel)
- Ownership (User-Channel)
- Upload (User_Video)
- Like (User_Video)
- Video_Comment
- Belongs (Video_Channel)
- Video_Category
- Video_Tag
- Playlist_Video
- Playlist_Channel

FUNCTIONAL DEPENDENCY CONSTRAINTS:

User:

$(\text{user_id}) \rightarrow (\text{name}, \text{email}, \text{password})$

Channel:

$(\text{channel_id}) \rightarrow (\text{name}, \text{description}, \text{User.user_id}, \text{User.name}, \text{User.email}, \text{User.password})$

$(\text{channel_id}) \rightarrow (\text{name}, \text{description})$

Video:

$(\text{video_id}) \rightarrow (\text{title}, \text{url}, \text{description})$

$(\text{url}) \rightarrow (\text{title}, \text{description})$

Category:

$(\text{cat_id}) \rightarrow (\text{name})$

Tag:

$(\text{video_id}, \text{text}) \rightarrow (\text{text})$

Comment:

$(\text{video_id}, \text{Comment_id}) \rightarrow (\text{text}, \text{timestamp})$

Playlist:

$(\text{channel_id}, \text{playlist_id}) \rightarrow (\text{name})$

$(\text{video_id}, \text{playlist_id}) \rightarrow (\text{name})$

TABLES FROM ER DIAGRAM AND FUNCTIONAL DEPENDENCIES:

User(user_id, username, email, password)

Profile(user_id, firstname, lastname, gender, avatar)
foreign key : user_id references User.user_id

Channel(channel_id, name, description, user_id)
foreign key : user_id references User.user_id

Video(video_id, url, title, description, user_id)
foreign key : user_id references User.user_id

Tag(text, video_id)
foreign key : video_id references Video.video_id

Category(cat_id, text)

Playlist(playlist_id, name, channel_id)

foreign key : channel_id references Channel.channel_id

Comment(comment_id, text, timestamp, video_id, user_id)

foreign key : user_id references User.user_id

video_id references Video.video_id

Subscription (user_id, channel_id)

foreign key : user_id references User.user_id

channel_id references Channel.channel_id

Like(video_id, user_id)

foreign key : user_id references User.user_id

video_id references Video. video_id

Pl_Vid(video_id, playlist_id)

foreign key : video_id references Video. video_id

playlist_id references Playlist.playlist_id

Vid_Cat(cat_id, video_id)

foreign key : video_id references Video.video_id

cat_id references Category.cat_id

Sessions(user_id, session_id)

foreign key : user_id references User.user_id

SOFTWARE AND HARDWARE REQUIREMENTS:

Hardware requirements for hosting website:

- At least 2 GB RAM
- 1.5 GHz Processor
- Internet connection

Software requirements:

- Linux system
- python3
- pip
- mysql client
- apache server
- Software for database administration (Mysql workbench)

Instructions on linux:

- Install LAMP server on the system.
- Create a database with `utf8_unicode_ci` character encoding.
- Create user with username `django` and password `django`.
- If you want to configure the server and connection details open `Revels/.env` and edit the value for the variables.
- Open the terminal, then navigate into `Revels` directory in the terminal now run the following command to create the databases :
- `python3 create.py`
- Run the queries in `triggers.sql` to create the triggers.
- Run the queries in `data.sql` to create the sample database.
- Open the terminal, then navigate into `Revels` directory in the terminal now run the following command to install all dependencies :
- `sudo pip install -r req.txt`
- In the terminal now run the following command to initiate the server :
- `python3 manage.py runserver 0.0.0.0:8000`
- Open the browser and navigate to `http://localhost:8000`

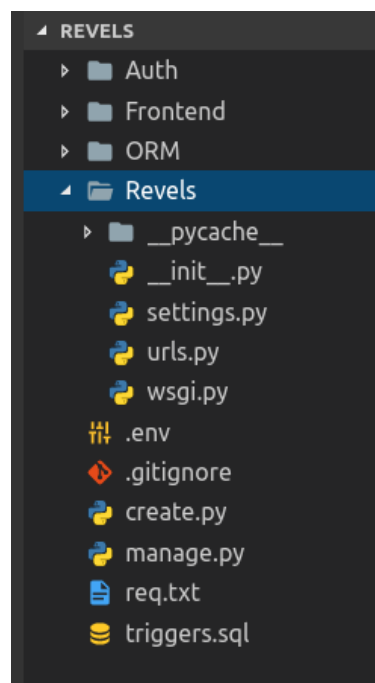
FILE STRUCTURE OF THE PROJECT/ BLOCKS:

The project is implemented using django web framework of python.

1. Overall structure of the project:

The project is named `Revels`. The main directory contains all the apps- `Auth`, `Frontend` and `ORM` along with git files for version control, the `manage.py` file which is used to run the project, `req.txt` containing requirements, `create.py` containing create table queries, `triggers.sql` for setting up triggers.

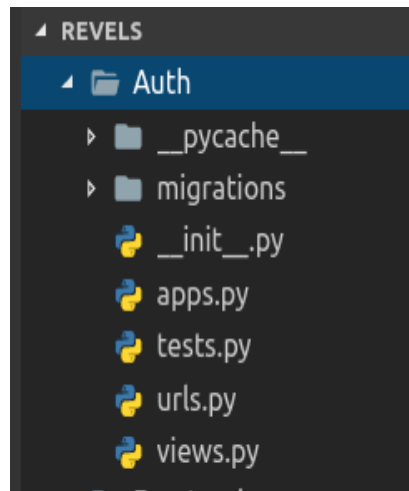
The subdirectory `Revels` consists of `settings.py` file where database and admin are configured. The `urls.py` contains urlpatterns which django checks upon getting request from the user.



2. Inside Auth:

The Auth is an app that handles user authentication. The main django urls.py searches for urls related to authentication in Auth/urls.py.

The views.py file consists of functions which handle the requests from users and render appropriate HTTP response. The apps.py file tells django that 'Auth' is an app.



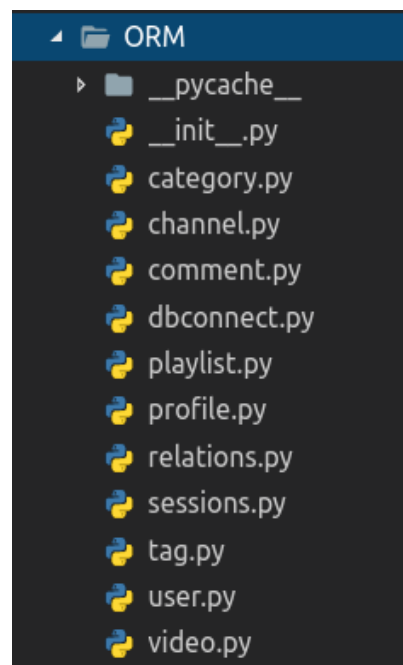
3. Inside ORM:

ORM stands for Object Relational Mapping.

The ORM contains python files of all tables. These files contain create table and other sql queries for these table. The sql queries are wrapped inside python functions. The purpose of this is to increase readability of the code.

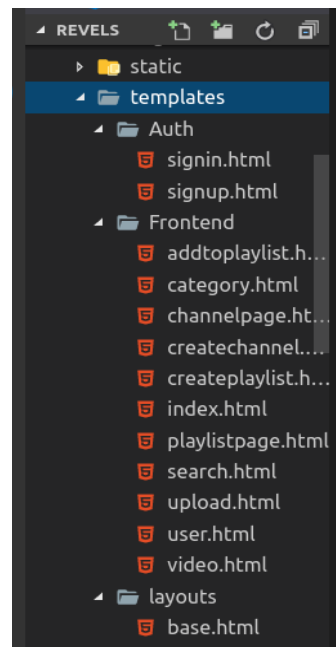
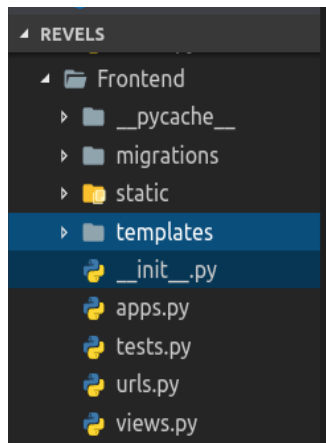
These functions can be called as normal python functions from views.py.

The important file here is dbconnect.py. In this file functions along with SQL Exception handling are present which can query or modify the database. The **connection** is opened and the **cursor** is used to execute queries. The relations.py file contains queries related to all relation sets.



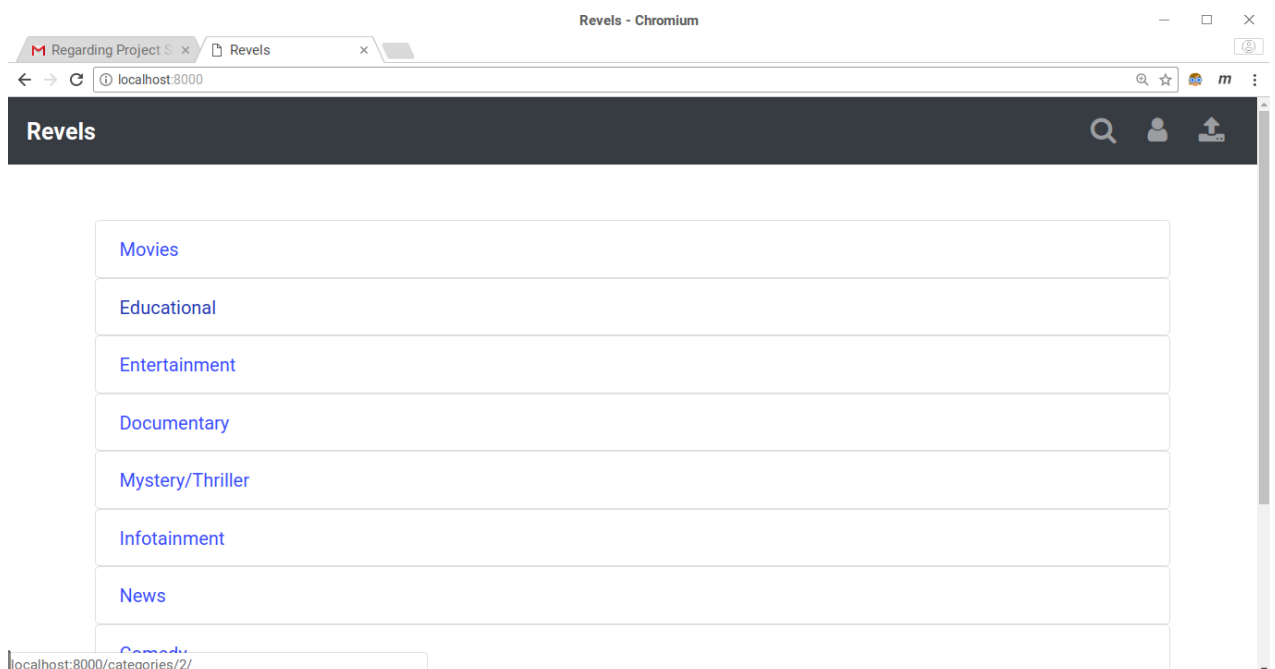
4. Inside Frontend:

The Frontend app handles all GUI of the website except authentication part. The `urls.py` in the Frontend is checked when django receives request for url related to Frontend. The `views.py` file contain functions to render templates. The templates are html files with output the response from server. The templates contain dynamic parts like for loops, if-else conditions. Each page is rendered from `views.py`, content of the template is provided through python dictionary.



FEATURES OF DIGITAL VIDEO LIBRARY MANAGEMENT SYSTEM (REVELS PORTAL):

1. Index (/):



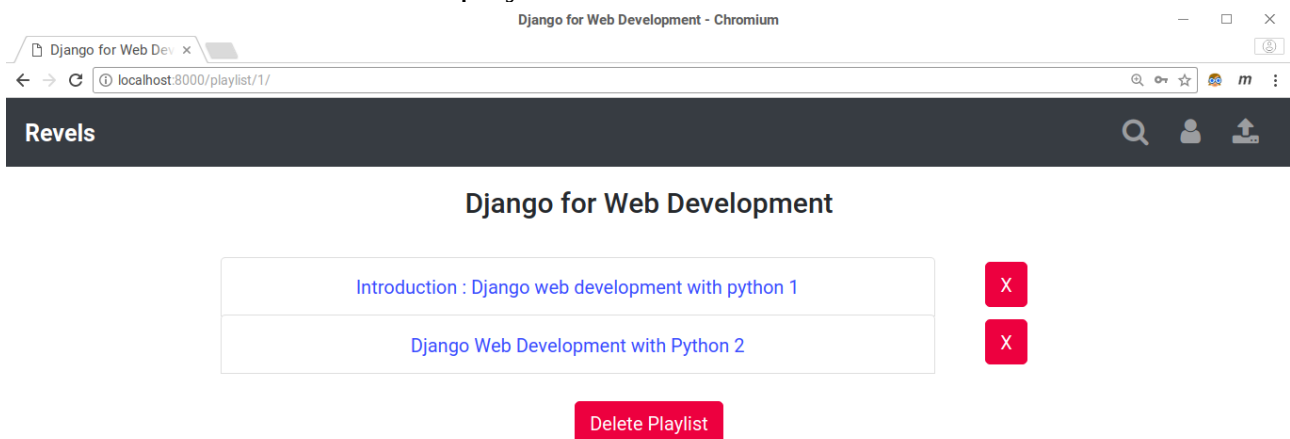
The index page is the home page of the website at url 'localhost:8000/'. It displays all the categories, user signin / profile page and upload video button. The template for index is

```
1 {% extends 'layouts/base.html' %} {% block title %} Revels {% endblock title %} {% block body %}
2 <div class="container" style=" margin: auto; margin-top: 3em;margin-bottom: 3em;">
3     <ul class="list-group">
4         {% for item in category %}
5             <li>
6                 <a class="list-group-item" href="{% url 'catvideo' item.cat_id %}">{{ item.text }}</a>
7             </li>
8         {% endfor %}
9     </ul>
10 </div>
11
12 {% endblock body %}
```

The view function index renders this template.

2. Displaying Playlist (/playlist/(no.)/):

The playlist is viewed differently for different users. If a user owns the channel and hence the playlist then he gets the option for deletion of playlist and deletion of videos from playlist.



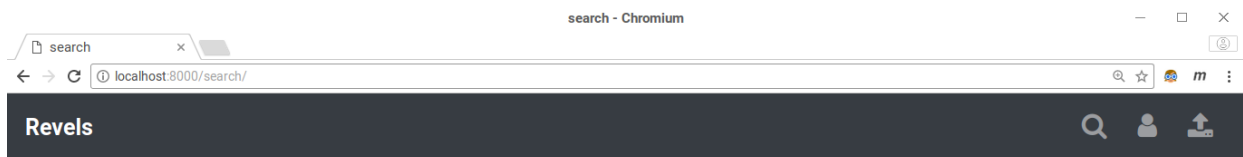
If the user is not signed in or if he's not the owner of the playlist then he cannot deletion the video. Addition of the video to the playlist is done from video page itself.



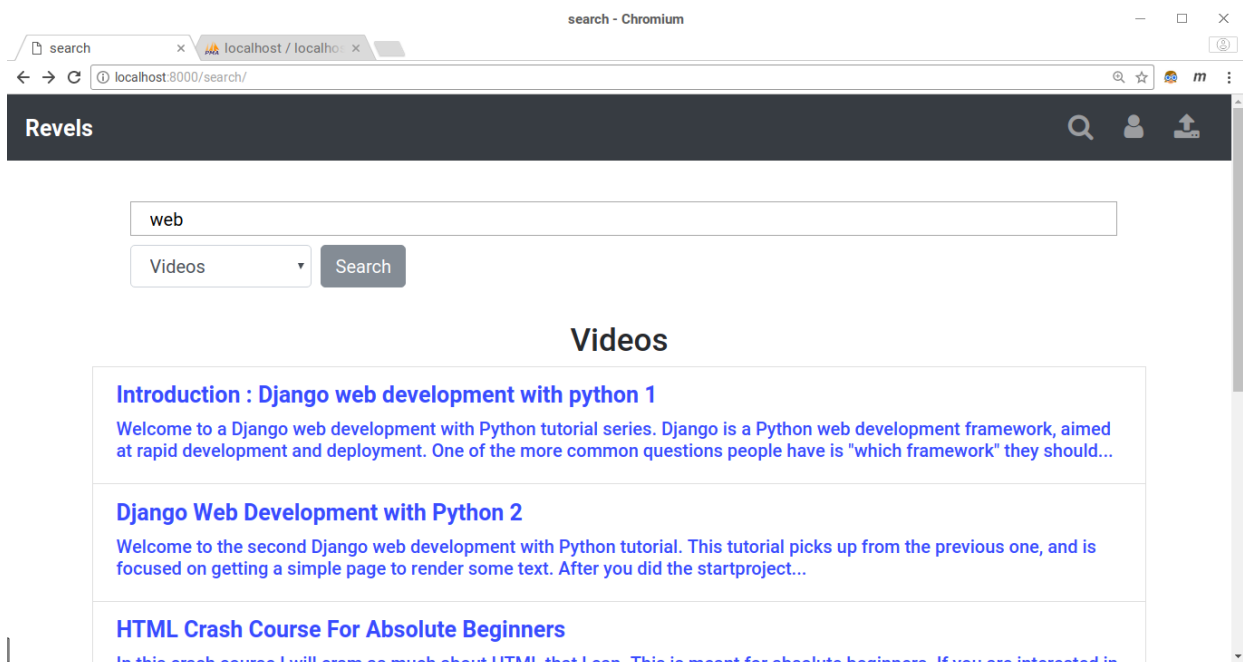
```
playlistpage.html x
1  {% extends 'layouts/base.html' %} {% block title %} {{ pl.0.name }} {% endblock title %} {% block body %}
2  <div class="col col-lg-8 text-center" style="margin: auto; margin-top: 1em;">
3      <h4>{{ pl.0.name }}</h4>
4      <br> {{ msg }}
5      <ol class="list-group">
6          {% for item in vidlist %}
7              <li class="row">
8                  <a href="{% url 'viewVideo' item.video_id %}" class="list-group-item col col-lg-10">
9                      {{ item.title }}</a> {% if showdel == True %}
10                     <form action="{% url 'removeVidPl' plid=pl.0.playlist_id vid=item.video_id %}" class="col col-lg-2" method="post" sty
11                         {% csrf_token %}
12                         <input type="submit" value="X" class="btn btn-danger">
13                     </form>
14                 {% endif %}
15             </li>
16         {% endfor %}
17     </ol>
18     <br> {% if showdel == True %}
19     <form action="{% url 'deletePlaylist' pl.0.playlist_id %}" method="post">
20         {% csrf_token %}
21         <input type="submit" value="Delete Playlist" class="btn btn-danger">
22     </form>
23     {% endif %}
24 </div>
25 {% endblock body %}
```

3. Search page (/search/):

The search page helps to search a particular video, channel, playlist, other users, etc. The video is searched based on title, description, tags, etc. The results are displayed on the same page. Clicking on the video redirects to the video page of that video.



After the search...



SQL queries for the search....

```
def search(req) :
    if req.method=='GET':
        return render(req,'Frontend/search.html')
    elif req.method=='POST':
        qr = req.POST['query'].strip()
        cat = req.POST['cat'].strip()

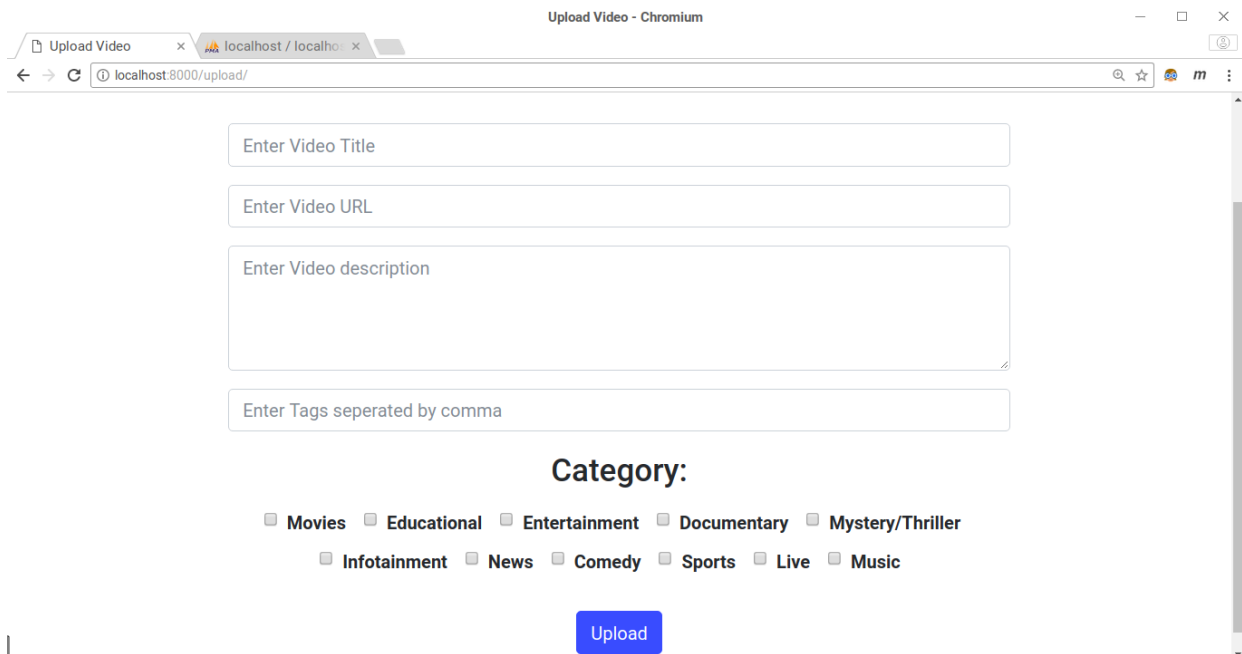
        if cat == "Channels" :
            res = con.query("""
            SELECT * FROM Channel WHERE Channel.name LIKE %s OR Channel.description LIKE %s;
            """, ("%"+qr+"%", ("%"+qr+"%"))
            for i in range(len(res)) :
                res[i]['description'] = trimVidDesc(res[i]['description'])

        if cat == "Categories" :
            res = con.query("""
            SELECT * FROM Category WHERE Category.text LIKE %s;
            """, ("%"+qr+"%"))

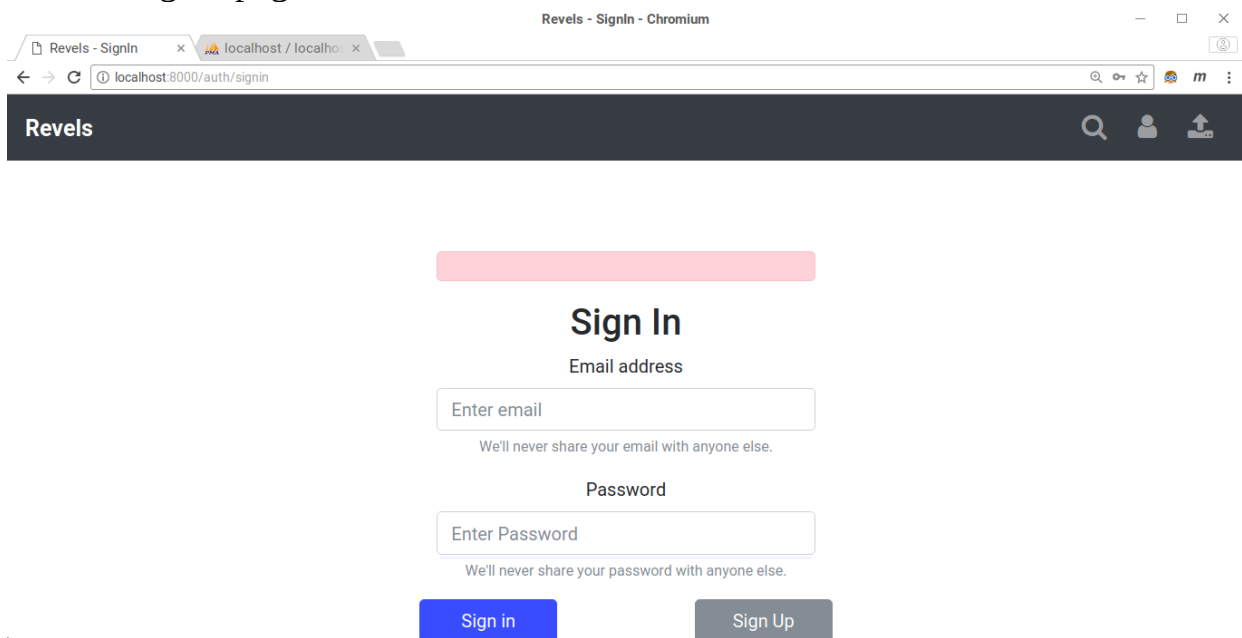
        if cat == "Playlists" :
            res = con.query("""
            SELECT * FROM Playlist WHERE Playlist.name LIKE %s;
            """, ("%"+qr+"%"))

        if cat == "Videos" :
            res = con.query("""
            SELECT DISTINCT Video.* FROM Video LEFT JOIN Tag
            ON Video.video_id=Tag.video_id WHERE
```

4. Upload video (/upload/):

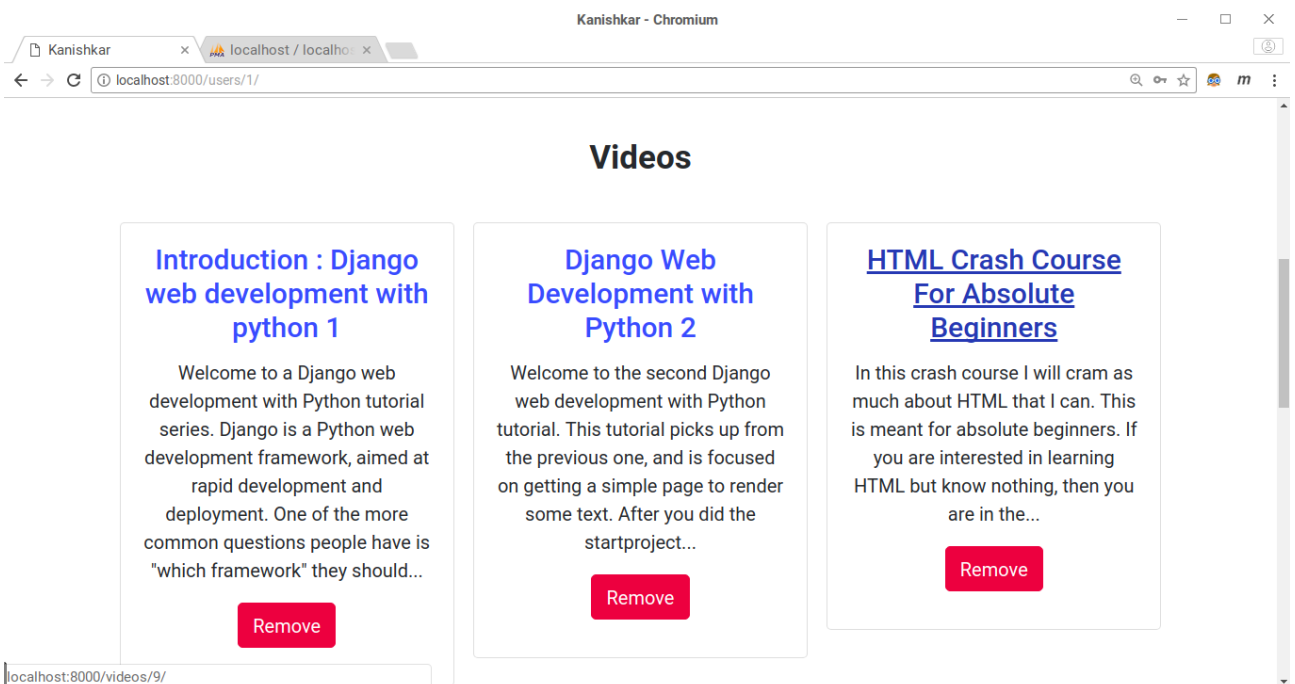
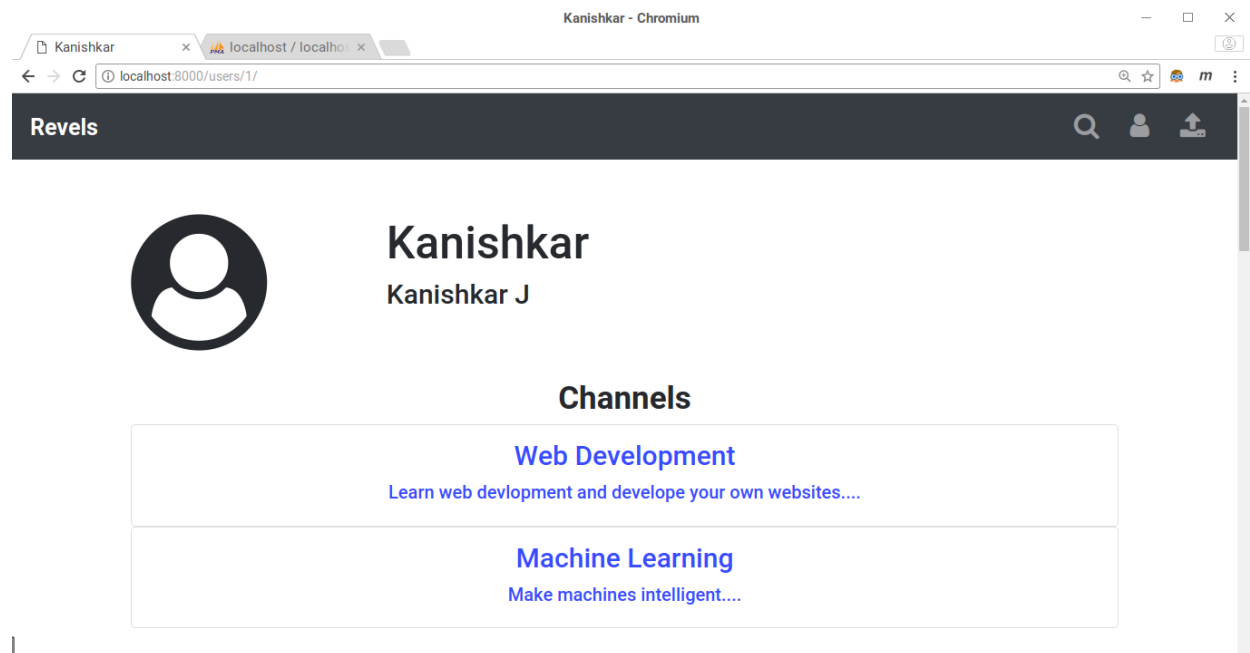


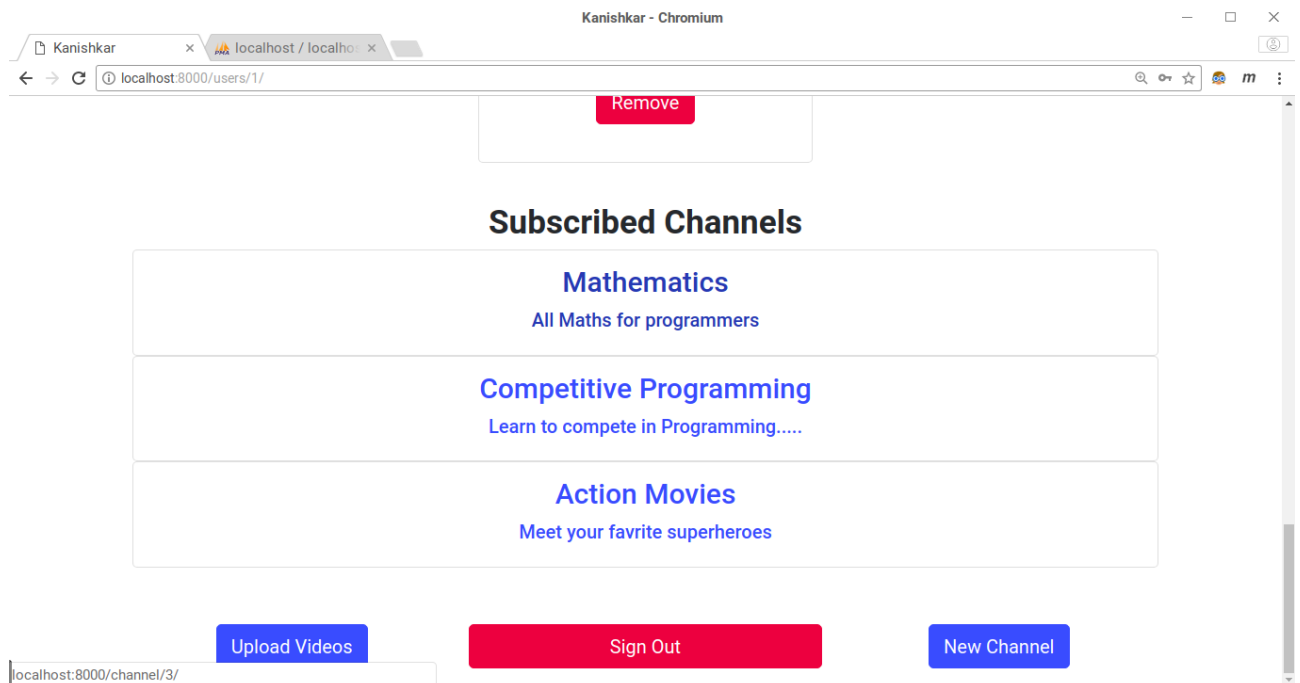
The video can only be uploaded if user is signed in else the page redirects to the signin page.



5. User Profile (/users/(no.)/):

The user page displays all the information about the user which includes username, name, channels, uploaded videos, subscribed channels, etc.



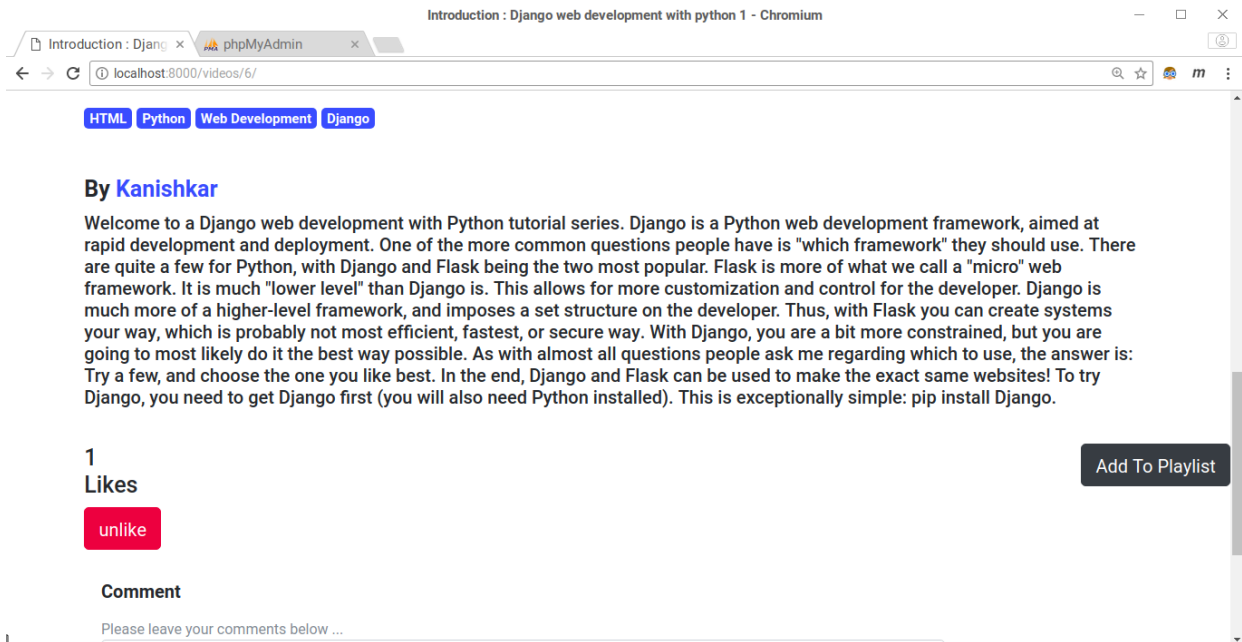


If the user goes to his own user page the he can get features for adding new channel, uploading videos, remove his uploaded videos, etc.

6. Video page (/videos/(no.)):

The video page is the main video display page. Anyone can like, comment, add video to their playlist, navigate through tags by clicking on them, etc.





4. Sign In(/auth/signin) :

The users can authenticate into the application from this page. Once the user is authenticated a cookie will be sent to the client system. This cookie aids further authentication check. The page also contains

5. Sign Up Page(/auth/signup)

Sign Up

firstname

surname

username

email@gmail.com

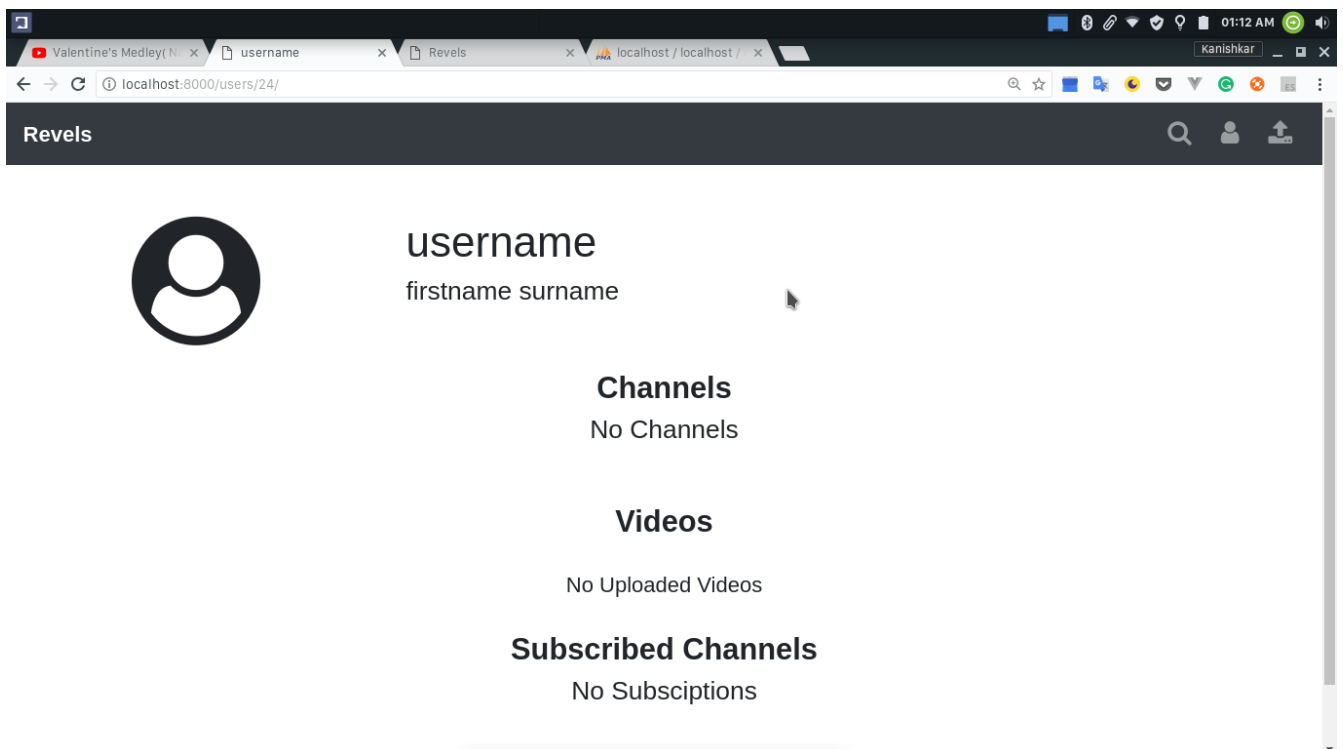
.....

Gender

M

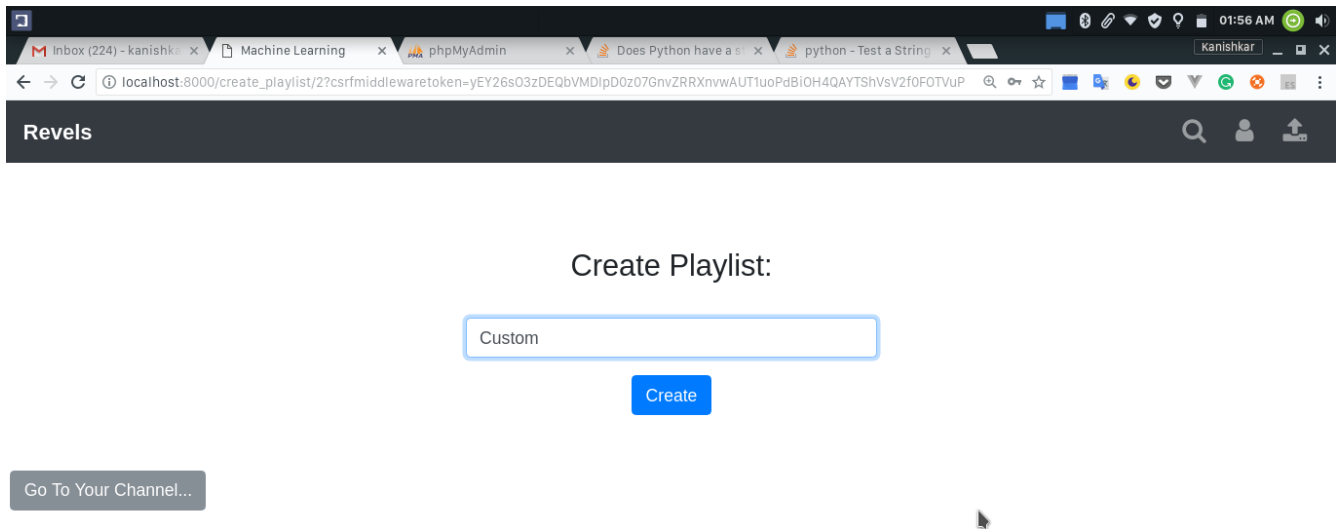
Sign Up

Users can create accounts from this page. When a account is created from this portal entry is made into the user table also the profile table. Then the user is redirected to the sign in page for security reasons.

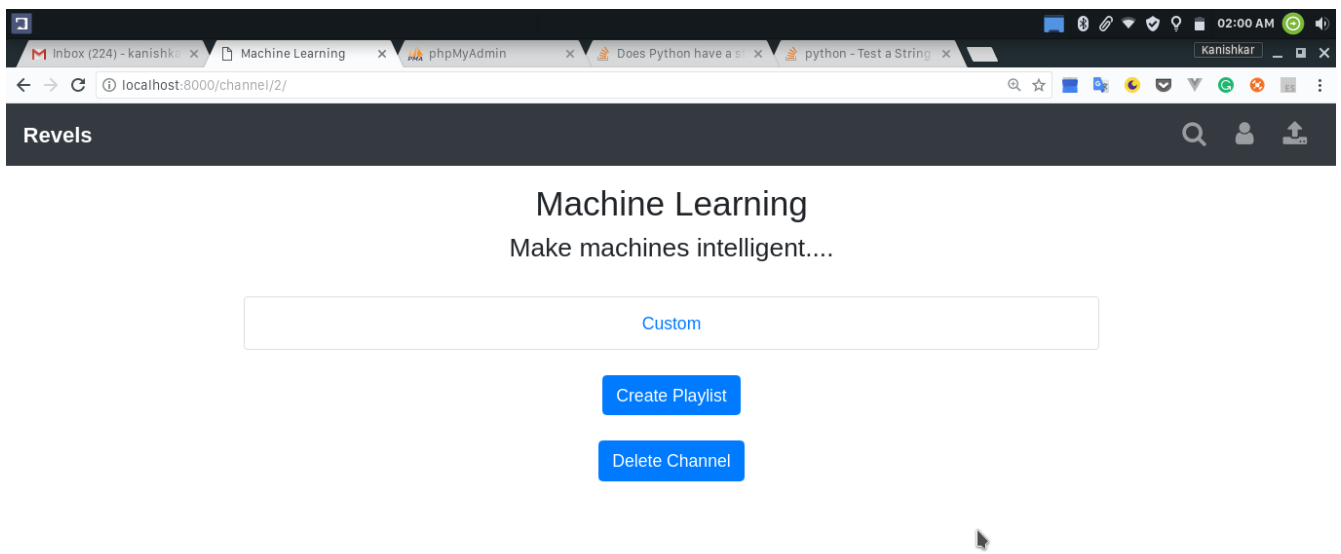


The above screenshot is after sign in.

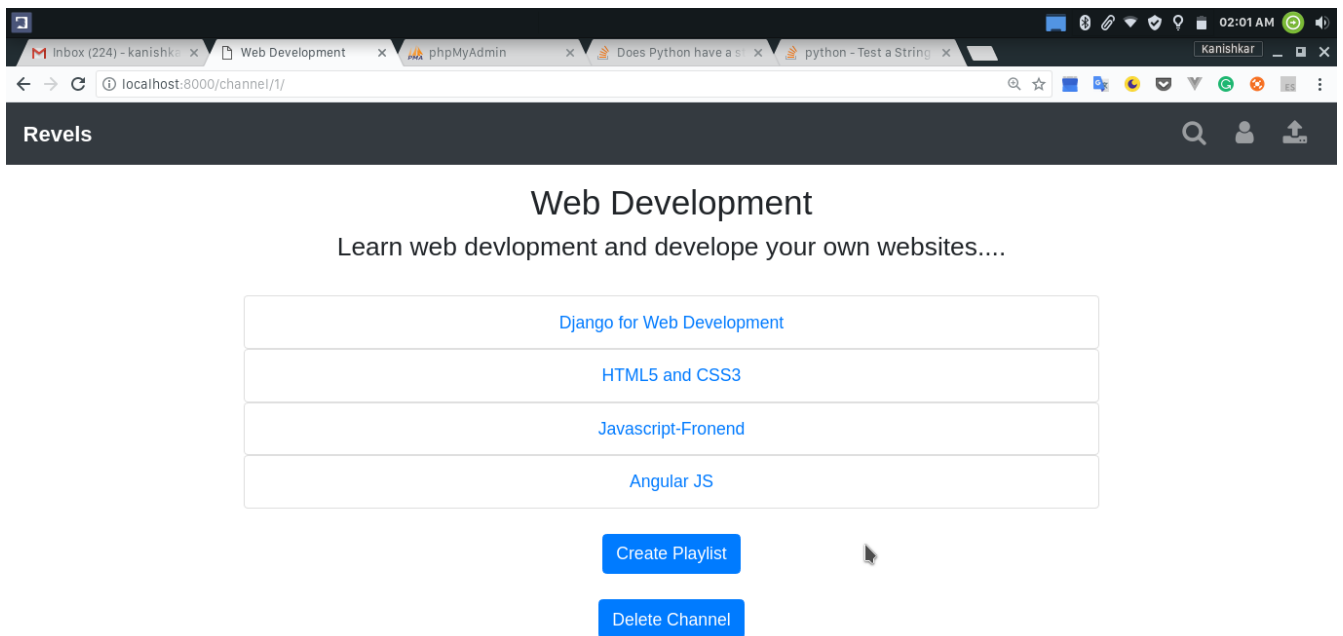
6. Create Playlist (/create_playlist/channel_id) :



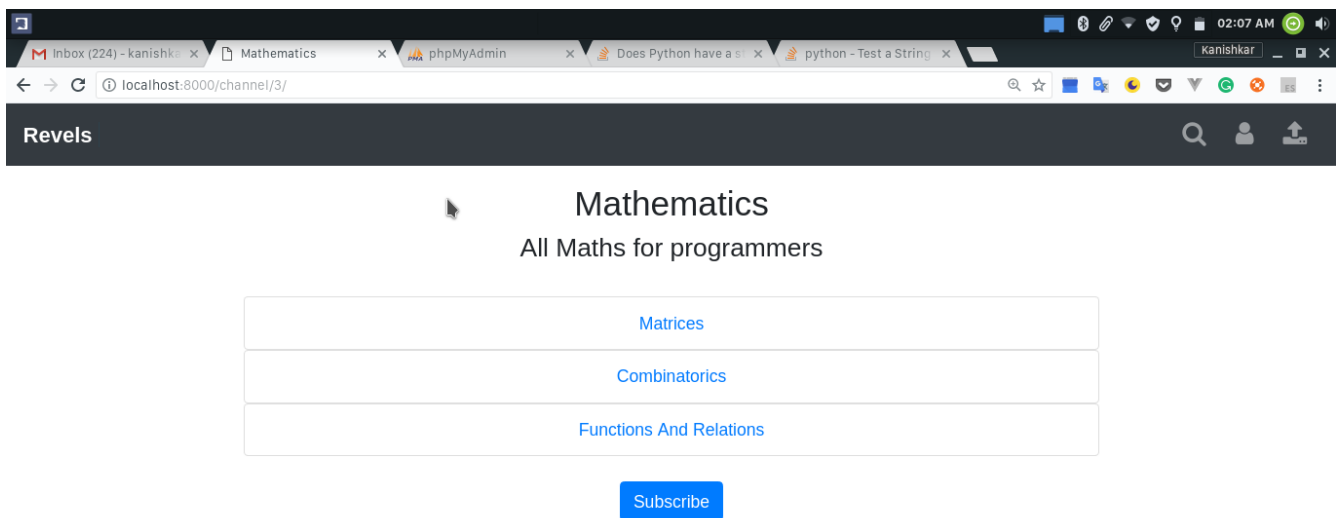
The owner of the channel can create a playlist in a channel from this page. If someone user tries to access this page they are served a 500 server error page.



7. Channel page (/channels/channel_id) :

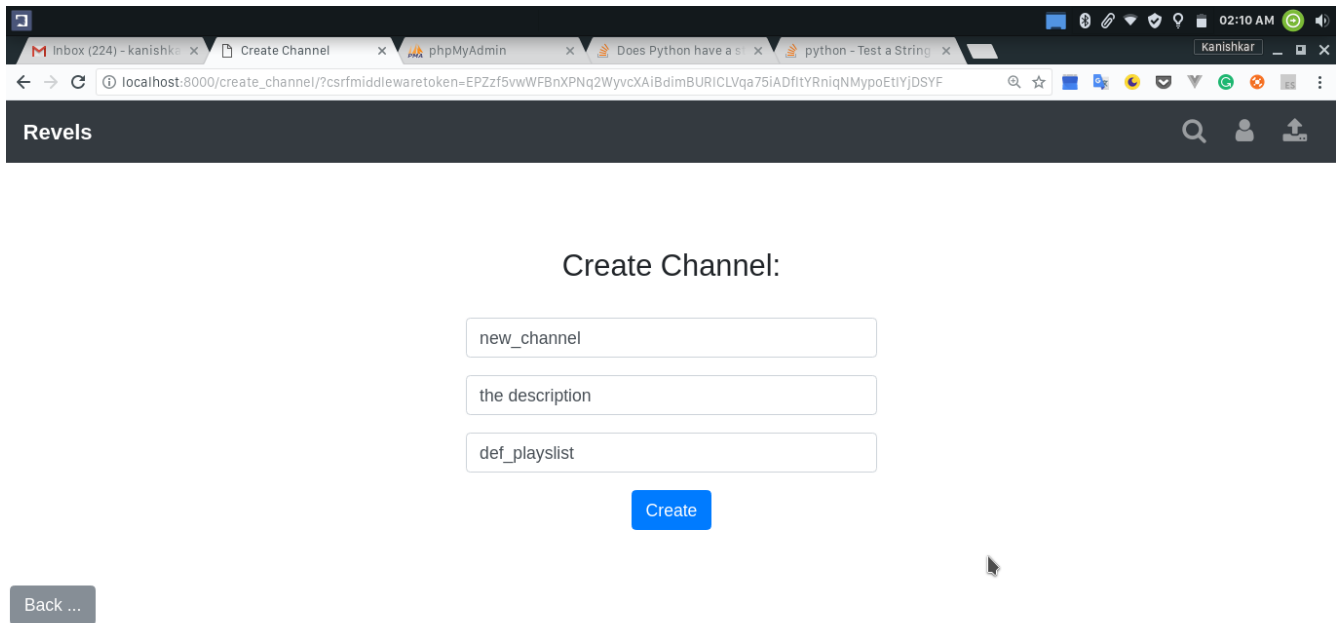


The above screenshot is from a channel the user owns, while the one below is of a channel someone else owns.

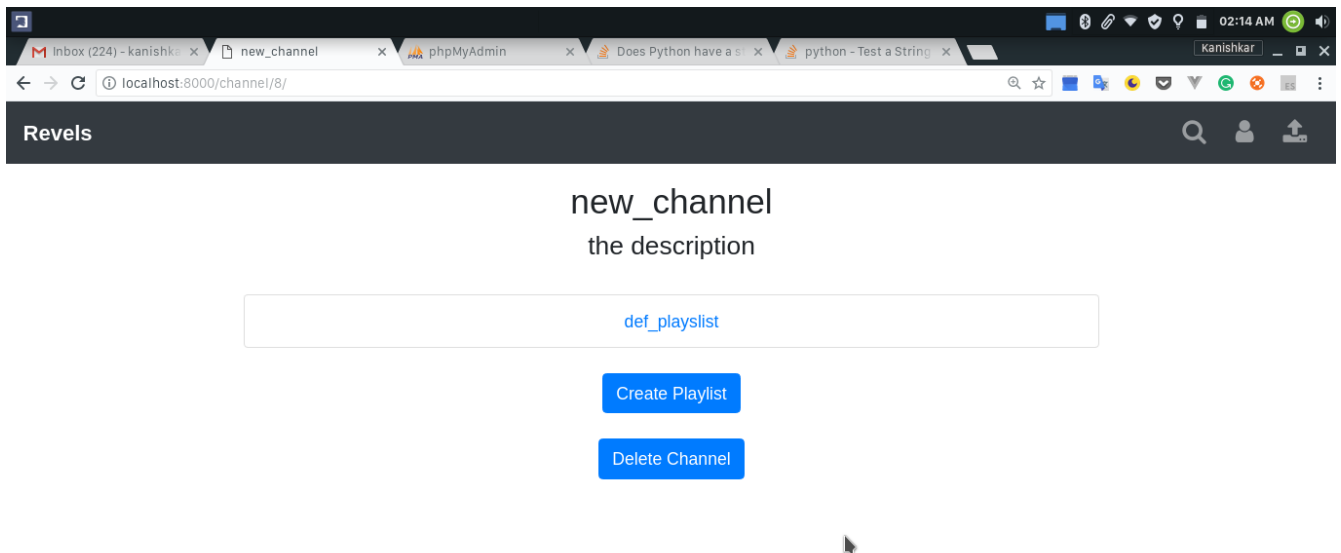


This page displays channel details, with the list of playlists. If the user accessing this page is the owner of the channel then create playlist and delete channel page buttons are enabled. Otherwise the buttons won't be displayed, instead there will be a subscribe button.

8. Create Channel (/create_channel/):

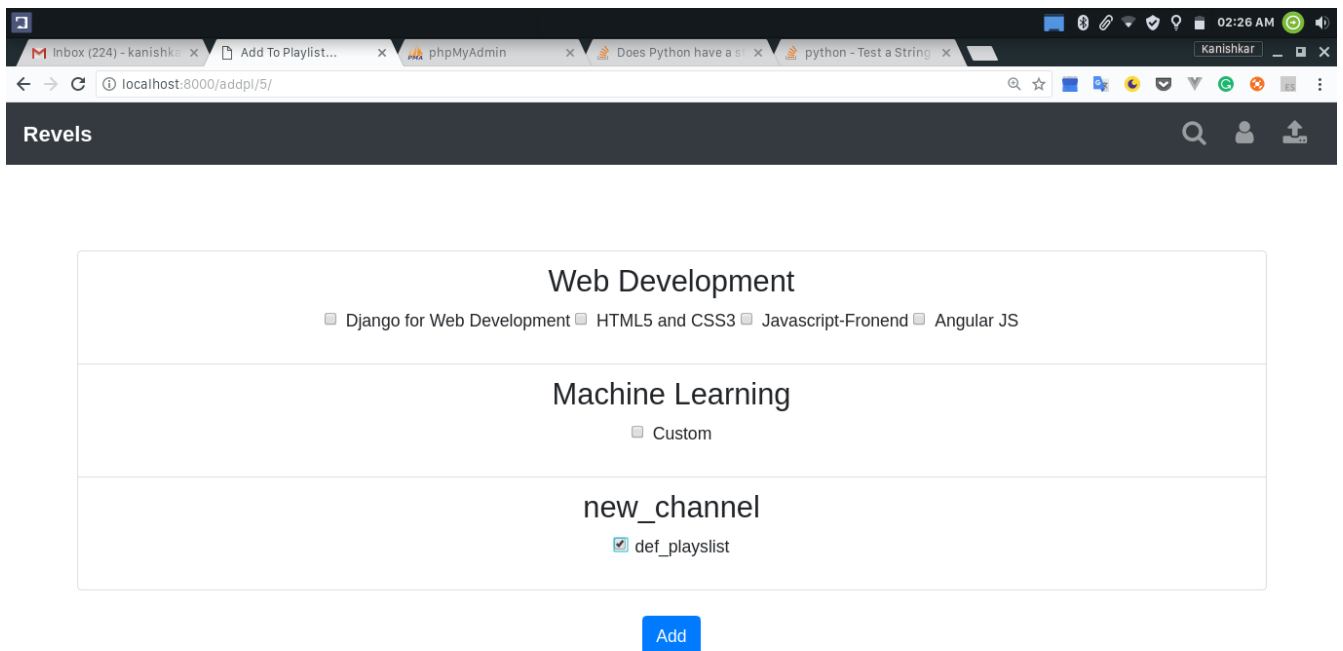


The above screenshot is while creation. The below one has been taken after the creation.

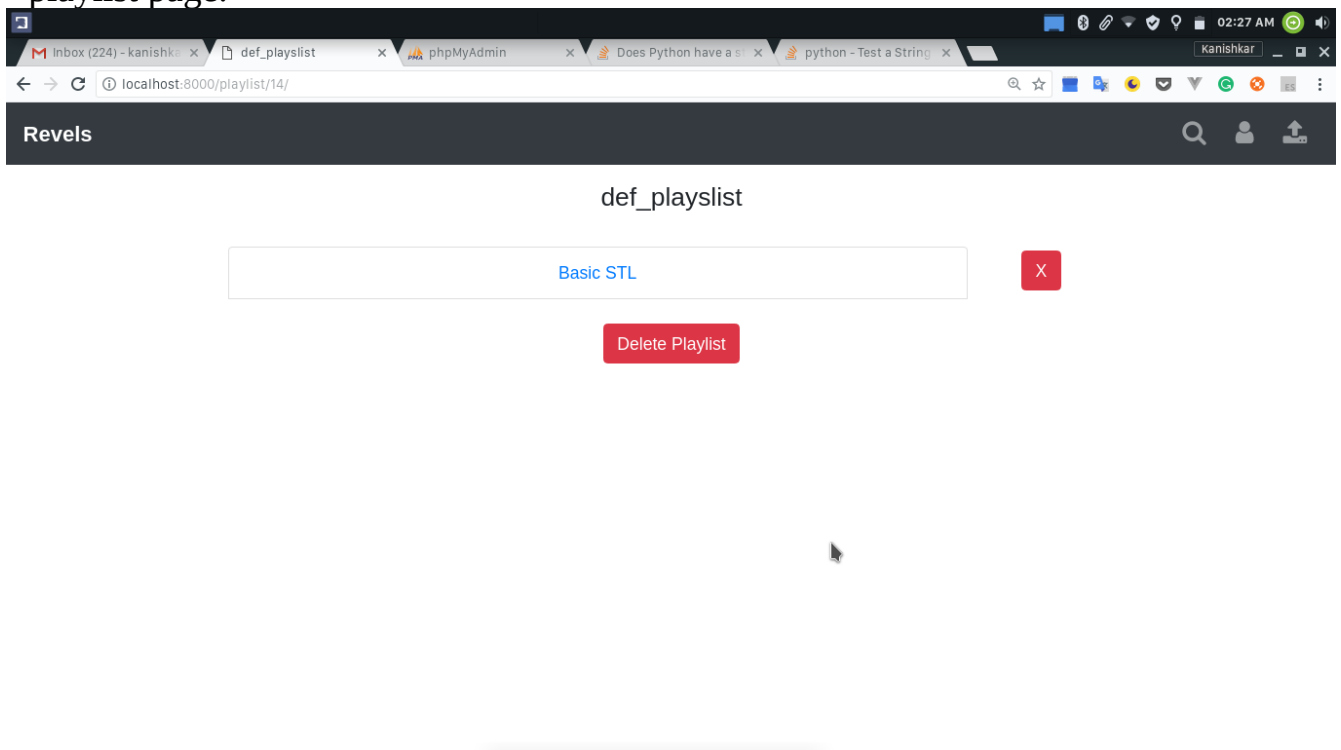


One can create new channels from this page. After the channel is created a default playlist is also created.

9. Add To Playlist(/addpl/video_id/)



The above page is the add to playlist page. The one below is a screenshot from playlist page.



Add to playlist page displays all the channels and the playlists they enclose. Then the user gets to choose the playlists they wants to add the current video to.

TRIGGER ANALYSIS

1. CheckProfileInsert

The screenshot shows a web browser window with the URL `localhost:8000/auth/signup`. The page has a dark header with the text "Revels" and a search icon. The main content area is white and contains a "Sign Up" form. At the top of the form, there is a red error message: "(1644, 'Please enter a Firstname')". Below this, the form fields are: "Enter First Name", "Enter Last Name", "user3", "user3@gmail.com", "Enter Password", and a "Gender" dropdown menu with "M" selected.

This trigger is called whenever an insertion operation is performed on the profiles table. This trigger first checks if the user has given a firstname and lastname or not. If the first name is not entered the trigger will cancel the insertion and pop up a error and delete the user details from the user table with the same user_id.

2. CheckUserInsert

The screenshot shows a web browser window with the URL `localhost:8000/auth/signup`. The page has a dark header with the text "Revels" and a search icon. The main content area is white and contains a "Sign Up" form. At the top of the form, there is a red error message: "(1644, 'Please enter a valid email address')". Below this, the form fields are: "user13", "user13", "user13", "user13", "Enter Password", and a "Gender" dropdown menu with "M" selected.

This trigger is executed whenever an insertion operation takes place on the user table. The trigger checks if the email address provided by the user is a valid email address. It performs the following check : ‘%_@_%._%’ .