Media streaming with IBM video streaming

Phase: 4 Development part-2

Continue building the platform by integrating video streaming services and enabling on-demand playback:

Dacast: Dacast is a professional streaming platform that offers live and on-demand video hosting services. It provides a range of features such as white-labeling, monetization, and analytics

Uscreen: Uscreen is a VOD platform that allows you to create your own branded video streaming service. It provides features such as custom branding, monetization, and marketing tools

Muvi: Muvi is a VOD platform that offers end-to-end video streaming solutions. It provides features such as multi-device support, monetization, and analytics

VPlayed: VPlayed is a cloud-based VOD platform that offers live and on-demand video hosting services. It provides features such as multi-device support, monetization, and analytics

Adding File Upload to Your Website

Do it Yourself (DIY) Approach

The DIY approach to adding file upload functionality to your website involves creating a

custom solution using frontend and backend technologies. Here's a breakdown of how this approach works:

Frontend (client-side):

- Design a user interface with an HTML form containing a file input field. This allows users to browse and select files from their local devices to upload.
- Optionally, use JavaScript for client-side validation, such as checking file types or sizes before the file is uploaded to the server.
- When the user submits the form, the selected file and any additional form data are sent to the server for processing.
- **Backend (server-side)**:
- Choose a server-side programming language (e.g., PHP, Node.js, Python, or Ruby) to handle the file upload process.
- Create a server-side script that receives the uploaded file and any additional form data from the client.
- Validate the received file on the server-side, checking for file types, sizes, and any other security measures necessary to ensure only valid files are accepted.

- Process the uploaded file as needed, which might include resizing images, generating thumbnails, or converting file formats.
- Save the file to a designated storage location on your server or another storage system, such as a database or cloud storage service.
- Store any additional form data (e.g., title, description, or tags) in a database or other data storage system, along with a reference to the uploaded file's location.
- Return a response to the client, indicating the success or failure of the file upload process. This might include displaying a confirmation message or providing a link to the uploaded file.

Project Setup

You generally create a Django project anywhere on your system and under your Django project you have one or more applications.

your **videouploaddisplay** in **djangovideouploaddisplay /djangovideouploaddisplay/settings.py** file under `INSTALLED_APPS` section as below:

```
INSTALLED_APPS = [
    ...

'videouploaddisplay.apps.VideouploaddisplayConfig']
```

The `videouploaddisplay.apps.VideouploaddisplayConfig` is formed as a dotted notation from **djangovideouploaddisplay/videouploaddisplay/apps.py**, where you will find the class name as `VideouploaddisplayConfig` that has name **videouploaddisplay**.
In your settings file
– djangovideouploaddisplay/djangovideouploaddisplay/settings.py,
define `MEDIA_ROOT` and `MEDIA_URL`, for example:

```
MEDIA_ROOT = os.path.join(BASE_DIR, '')

MEDIA_URL = '/'
```

In the above configuration I am not specifying any directory for storing the video but the video will be uploaded in the root directory of the project.

Template or View File

```
<!DOCTYPE html>
<html>
   <head>
      <title>Django - Video File Upload and Display</title>
   </head>
   <body>
            <div style="width: 500px; margin: auto;">
                  <fieldset name="Video File Upload and Display">
```

```
                {% if msg %} {% autoescape off %} {{
msg }} {% endautoescape %} {% endif %}
                <form method="post" action="/"
enctype="multipart/form-data">
                    {% csrf_token %}
                    <dl>
                        <p>
                            <label>Browse and
select a video file</label>  
                            <input type="file"
name="file" autocomplete="off" required>
                        </p>
                    </dl>
                    <p>
                        <input type="submit"
value="Upload and Display">
                    </p>
                </form>
            </fieldset>
{% if filename %}
                <div style="margin: 10px auto;">
```

```
                        <video autoplay="autoplay"
controls="controls" preload="preload">
                            <source src="{{ MEDIA_URL
}}/{{ filename }}" type="video/mp4"></source>
                        </video>
                    </div>
                {% endif %}
            </div>
    </body>
</html>
```

Form

The Django `Form` will map the fields to form on your template file. In the above template or view file I have kept only one field called `file` which is used to select a file. A view handling this form will receive data into `request.FILES`.

```
from django import forms


class UploadFileForm(forms.Form):
    file = forms.FileField()
```

Views

In this view I am going to show how to upload file into a server location. You will also see how file data is passed from request **to `Form`. Here also notice**

how I have used CSRF decorator to ensure the CSRF token.

```
from django.shortcuts import render

from .forms import UploadFileForm

from django.views.decorators.csrf import ensure_csrf_cookie


@ensure_csrf_cookie

def upload_display_video(request):

    if request.method == 'POST':

        form = UploadFileForm(request.POST, request.FILES)

        if form.is_valid():

            file = request.FILES['file']

            #print(file.name)

            handle_uploaded_file(file)

            return render(request, "upload-display-video.html", {'filename': file.name})

    else:

        form = UploadFileForm()

    return render(request, 'upload-display-video.html', {'form': form})
```

```
def handle_uploaded_file(f):
    with open(f.name, 'wb+') as destination:
        for chunk in f.chunks():
            destination.write(chunk)
```

URLs

Now you need to define the path or URL which will call the appropriate function (for this example, `upload_display_video()`) on your views.py file. You need to create a file urls.py under videouploaddisplay folder with the following code snippets.

```
from django.urls import path

from django.conf import settings


from . import views


urlpatterns = [

   path('', views.upload_display_video,
name='upload_display_video'),

]
```

Integrate IBM Video Streaming

Create an IBM Cloud Account: If you don't already have one, sign up for an IBM Cloud account.

Provision the IBM Video Streaming Service: Once you're logged in, provision the IBM Cloud Video streaming service from the IBM Cloud Catalog.

Configure Your Stream: Set up the necessary configurations for your video stream, including settings like bitrate, resolution, and encoding.

Generate API Keys: You'll need API keys to interact with IBM's streaming services programmatically. Generate these keys within your IBM Cloud account.

Choose a Video Player: Select a video player for playback. Popular options include HTML5 video players, JW Player, or Video.js.

Embed Video Player: Embed the chosen video player in your website or application. Configure it to use the API keys you generated earlier.

Upload Your Video Content: Upload your video content to IBM Cloud Video storage or use a content delivery network (CDN) for better performance.

Secure Your Content: If needed, implement access controls and encryption to protect your video content from unauthorized access.

Test Playback: Test your setup to ensure smooth and high-quality video playback. Pay attention to buffering, load times, and video quality.

Optimize for Mobile Devices: Ensure that your video playback is optimized for various devices, including mobile phones and tablets.

Monitor and Analyze: Set up monitoring and analytics to track video performance and user engagement. IBM Cloud Video may offer built-in analytics tools.

Scale as Needed: As your audience grows, scale your streaming infrastructure to handle increased demand. IBM Cloud offers scalability options to accommodate this.

Documentation and Support: Refer to IBM Cloud Video documentation and support resources for detailed guidance and troubleshooting.

Please note that the specific steps and services you need may vary depending on your requirements and the services offered by IBM Cloud Video at the time of your integration. Be sure to consult IBM's official documentation for the most up-to-date information.