

MEDIA STREAMING WITH IBM CLOUD VIDEO STREAMING

Phase-1 Document Submission

Problem Definition:

Media streaming, facilitated by IBM Cloud Video Streaming, offers an efficient way to deliver audio and video content to a global audience. However, it presents several challenges that need to be clearly defined and addressed to ensure a seamless and satisfying streaming experience.

Media streaming with IBM Cloud Video Streaming faces a multitude of challenges that impact the quality, reliability, and accessibility of streamed content. These challenges include:

Latency and Buffering:

The presence of latency and buffering during media streaming can lead to viewer frustration and abandonment of content. The problem is exacerbated during live streaming events when real-time interaction is essential.

The root cause of the latency and buffering are network congestion, packet loss, software and app Issues (The main cause of software and app issues are bad design or Architecture). This can be avoided by taking simple steps

Scalability:

As media streaming services attract larger audiences, ensuring the scalability of the infrastructure to meet increasing demand while maintaining consistent performance becomes a significant challenge.

Security and Piracy:

Protecting copyrighted content from piracy and ensuring secure streaming experiences are paramount. Unauthorized access, content theft, and security breaches continue to pose serious threats.

User Experience Optimization:

Meeting and exceeding user expectations by incorporating features like personalized recommendations, interactive elements, and accessibility options is a constant pursuit, demanding innovation and user-centric design.

Content Delivery Efficiency:

Optimizing the delivery of media content, especially for viewers in diverse geographic locations, is a pressing issue. Content needs to be efficiently distributed to reduce delivery costs and minimize latency.

Goals:**Improve Viewer Experience:**

Enhance the overall viewer experience by minimizing latency, reducing buffering, and ensuring high-quality streaming, leading to increased viewer satisfaction and engagement.

Increase Scalability:

Enable seamless scaling of media streaming services to accommodate growing audiences, ensuring that the infrastructure can handle increased demand during peak times.

Optimize Content Delivery:

Streamline content delivery to minimize latency and maximize efficiency, particularly for global audiences, by leveraging IBM Cloud Video Streaming's robust content delivery network (CDN) capabilities.

Enhance Security:

Strengthen security measures to protect against unauthorized access, content piracy, and security breaches, thereby safeguarding copyrighted content and user data.

Simplify Customization and Integration:

Simplify the process of customizing media streaming services to align with branding and integration requirements, reducing deployment time and resource overhead.

Design Thinking:

Platform Definition: Virtual Cinema Platform

1. User Registration and Account Management:

User Registration:

Allow users to create accounts with personal information, including name, email, and password.

Profile Management:

Enable users to update and manage their profiles, including profile pictures and personal information.

Authentication:

Implement secure authentication mechanisms, such as email verification and password reset, to protect user accounts.

2. Content Management:

Video Upload:

Provide content creators with an easy-to-use interface to upload their films, specifying metadata like title, description, genre, and runtime.

Media Library:

Organize uploaded videos in a user-friendly media library for easy access and management.

Content Verification:

Implement a review process to ensure that uploaded content complies with platform guidelines and legal requirements.

3. On-Demand Streaming:

Video Playback:

Enable viewers to stream films on-demand with options for different video quality levels (e.g., SD, HD, 4K) based on their internet connection.

Streaming Servers:

Utilize robust and scalable streaming servers to ensure smooth playback, low latency, and adaptive bitrate streaming.

Search and Discovery:

Implement search and recommendation features to help users discover content based on genres, ratings, and user preferences.

Watchlist:

Allow users to create and manage watchlists for saving films they want to watch later.

User Interface Design: Virtual Cinema Platform

Designing an intuitive and user-friendly user interface (UI) for a virtual cinema platform is essential to provide viewers with a seamless and enjoyable experience.

1. Homepage:

Hero Banner:

Feature a visually appealing hero banner showcasing new releases and recommended films.

Browse Categories:

Display categories like "Top Picks," "Genres," and "New Releases" for easy content discovery.

Search Bar:

Include a prominent search bar at the top for quick access to specific films.

2. Navigation Menu:

Simple Navigation: Use a clean and straightforward navigation menu with easily recognizable icons or text labels.

Sections:

Organize sections for Home, Movies, TV Shows, Genres, Watchlist, and Account.

3. Content Listings:

Grid View:

Show films in a grid layout with clear thumbnail images, titles, and brief descriptions.

Sorting and Filtering:

Allow users to sort and filter films by genre, release date, ratings, and more.

Infinite Scrolling:

Implement infinite scrolling for effortless browsing through a large catalog of films.

4. Video Player:

Responsive Player:

Ensure the video player is responsive and adjusts to various screen sizes.

Playback Controls:

Include standard playback controls for play, pause, volume, and fullscreen.

Quality Selection:

Let users choose the video quality based on their internet connection.

5. User Profile:

Profile Picture:

Enable users to upload a profile picture.

Watchlist:

Show the user's watchlist and allow them to manage it.

Viewing History:

Display the user's viewing history and resume playback from where they left off.

Account Settings:

Provide options to update account information, change passwords, and manage notifications.

Video Upload

Enable users to upload movies and videos to the platform.

1. Upload Process:

Implement a step-by-step upload process for users. This can include the following stages:

Title and Description:

Have users provide essential information such as the title, description, genre, and language of the video.

Video Upload:

Allow users to select video files from their local storage for upload. Support various video formats and ensure there are clear size limits.

Thumbnail Upload:

Enable users to upload a custom thumbnail image or automatically generate one from the video.

Metadata:

Request additional metadata like cast and crew information, release date, and any relevant tags.

Rights and Licensing:

Have users specify the rights and licensing terms for their content.

Preview and Review:

Allow users to preview their uploaded content and make any necessary edits or corrections.

Submission:

Let users submit their content for review and approval.

Streaming Integration

The objective of this integration is to seamlessly integrate IBM Cloud Video Streaming services into our platform to deliver a high-quality video streaming experience to our users. This integration aims to provide smooth video playback, secure content management, adaptive streaming.

Design Goals:**Seamless User Experience:**

Design an intuitive and user-friendly interface for content upload, playback, and management that ensures a seamless viewing experience.

High-Quality Streaming:

Implement adaptive bitrate streaming to optimize video quality based on the viewer's internet connection speed, ensuring smooth playback.

Security and Access Control:

Integrate robust authentication and access control mechanisms to protect content and ensure authorized access.

CONCLUSION:

In conclusion, the first phase of our problem statement and design thinking process for the media streaming project with IBM Cloud Video Streaming has provided us with valuable insights and a solid foundation. Through thorough research, and ideation, we have identified key challenges and opportunities in the realm of media streaming. Our design thinking approach has allowed us to empathize with the needs of our users, define the problem statement clearly, and generate innovative ideas for potential solutions.

As we move forward into the next phases of this project, we will continue to build upon this strong start. We will refine our ideas, develop prototypes, and engage in iterative testing to ensure that our final solution meets the needs of our users and leverages the capabilities of IBM Cloud Video Streaming to the fullest extent. With a user-centered approach and a commitment to innovation, we are confident that our media streaming project will ultimately deliver a seamless and compelling streaming experience for all users.

MEDIA STREAMING

PROBLEM DEFINITION:-

The project involves creating a virtual cinema platform using IBM Cloud Video Streaming. The objective is to build a platform where users can upload and stream movies and videos on-demand. This project encompasses defining the virtual cinema platform, designing the user interface, integrating IBM Cloud Video Streaming services, enabling on-demand video playback, and ensuring a seamless and immersive cinematic experience.

INNOVATION:-

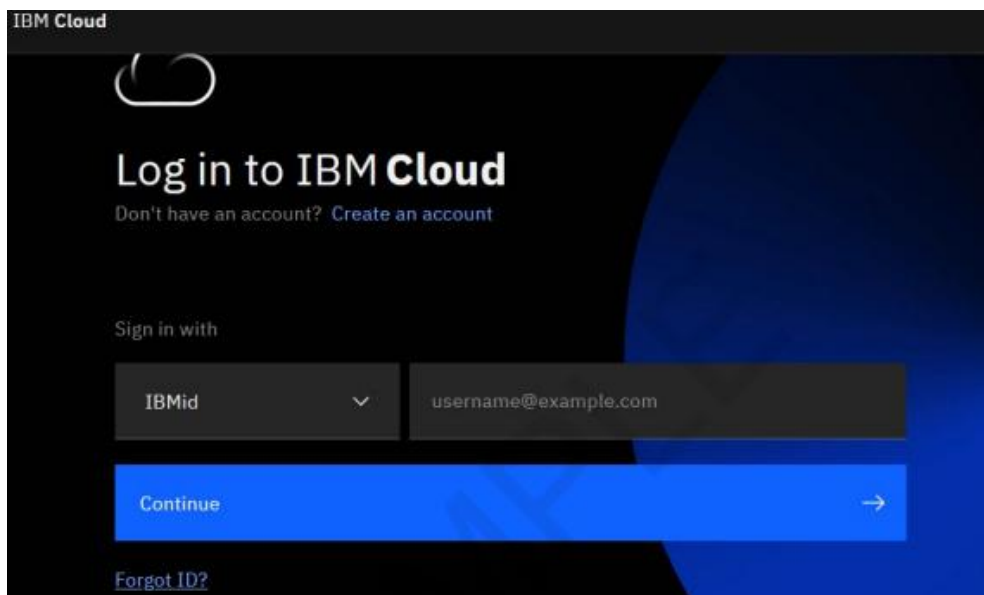
Considering incorporating features like user-generated playlists or real-time chat for a more engaging movie-watching experience.

CREATING DB2 AND COGNOS SERVICES ON IBM CLOUD ACCOUNT:-

After create a Lite account (Trail Free account) as a student. Login to IBM cloud.

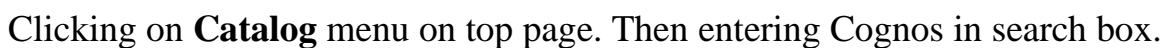
<https://cloud.ibm.com/login>

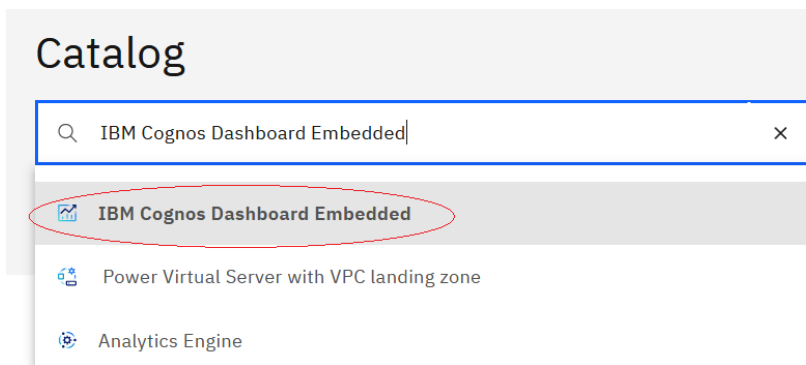
Entering our registered mail id.



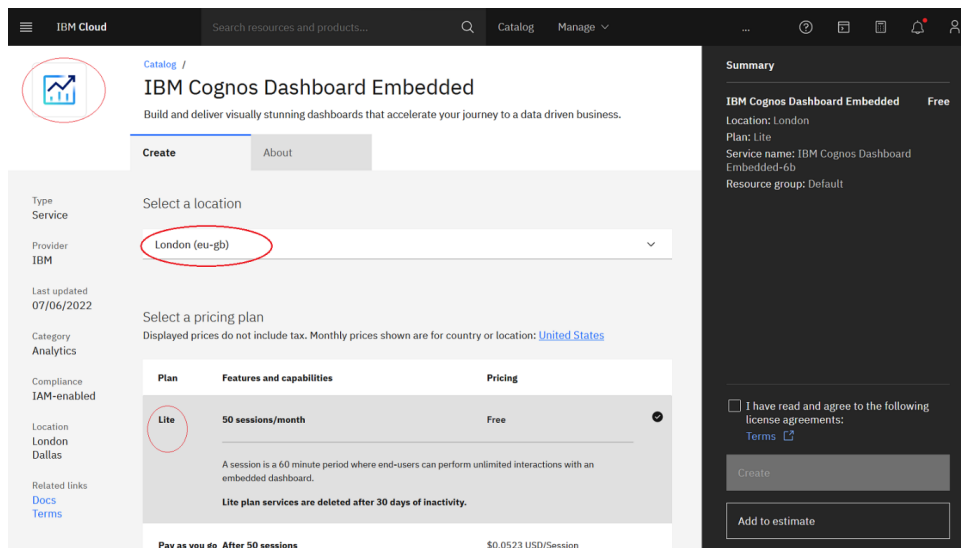
Then Entering Password

After successfully login, user will be redirected to account Dashboard page.





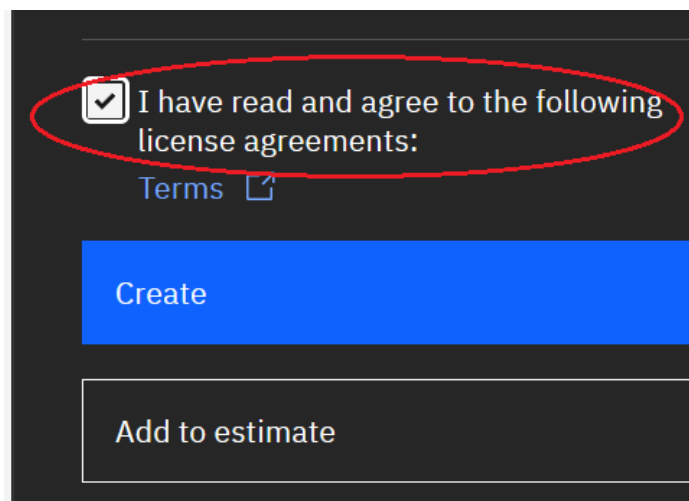
Click “IBM Cognos Dashboard Embedded”, it will redirect to Cognos page. Once you are on Cognos page.



Select a Location: **London**

Select a pricing plan: **Lite**.

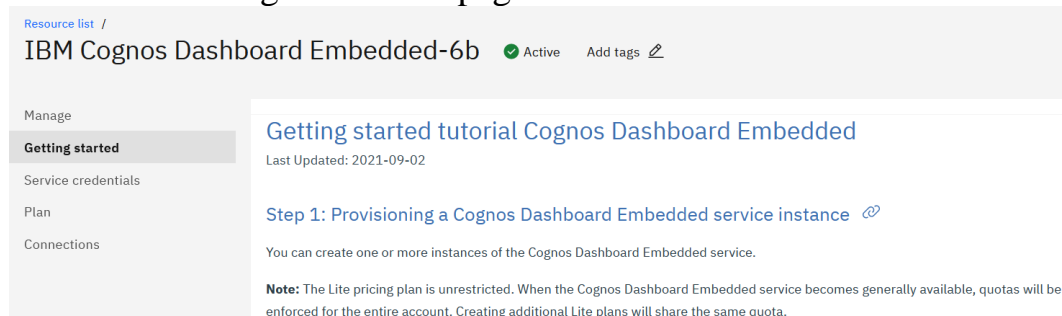
Select terms & condition (license agreements)



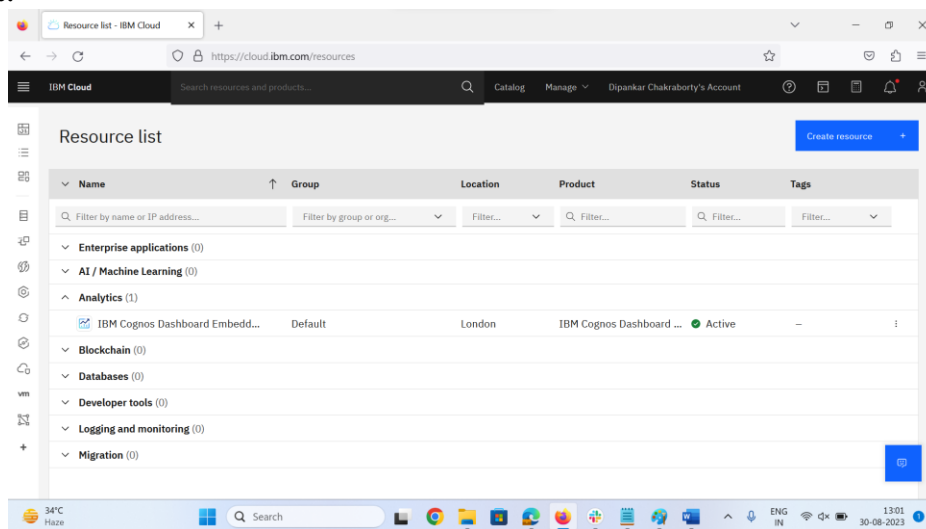
Then click on **Create** button.

After you click on create, it will take some time to create the service.

User will be redirected to Cognos tutorial page.

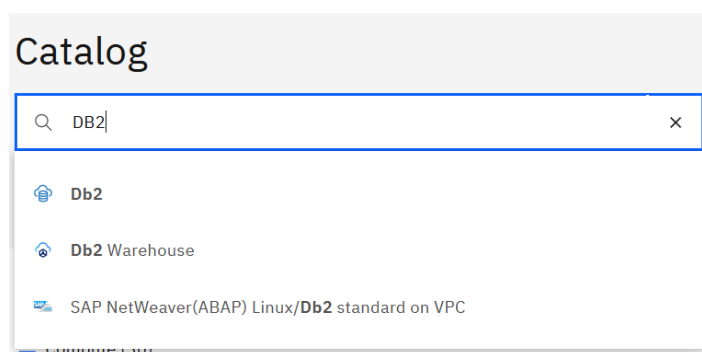


It takes few minutes to service created at cloud account. After few minutes you can check resource list.



ADDING DB2 SERVICES IN THE ACCOUNT:-

Now again search DB2 on catalog



Click on **Db2**, you will be redirected to DB2 service page.

The screenshot shows the IBM Cloud console for creating a Db2 instance. The 'Create' tab is active, and the 'Location' dropdown is set to 'London (eu-gb)'. The 'Plan' is set to 'Lite'. The 'Summary' panel on the right shows the configuration details. A checkbox for 'I have read and agree to the following license agreements:' is checked. The 'Create' button is highlighted in blue.

Select a Location: **London**

Select a pricing plan: **Lite**.

Select terms & condition (license agreements)

Then click on **Create** button.

After you click on create, it will take some time to create the service.

FEATURES

MULTIPLE ADMINS AND PERMISSIONS:-

Higher level plans can create a management structure inside their accounts. This can include granular permissions, such as setting someone up so they can manage published videos but prevent them from deleting or publishing content.

VIDEO REVIEW AND APPROVAL WORK FLOW:-

Setup custom roles that designate users as moderators of content, reviewing and approving with automated email notifications to content generators.

SIMULTANEOUS LIVE STREAMING:-

Creating multiple channels and conduct live streams simultaneously across them.

MULTILANGUAGE AUDIO TRACKS:-

Live stream video in multiple languages, appearing as language tracks during the live stream and for on-demand version.

4KVIDEO STREAMING:-

Creating 4K resolution live video streams or upload 4K video files.

SIMULATED LIVE:-

Simulate a live broadcast using previously recorded video. Choose one or multiple videos and schedule a start time from an experience that is presented as a live stream.

VIDEO EDITING / TRIMMING:-

On demand videos can be edited through the online dashboard. This can be used to remove unwanted parts at the start or end of video content. These edits are nondestructive and can be reverted at any time. Segments can also be saved as separate videos, as teasers or to break up long form content.

DYNAMIC MANUAL VIDEO PLAYLIST:-

Creating video playlists, a collection of on demand video content. These can be created manually or dynamically, evolving automatically with content based on assigning metadata criteria.

PLAYBACK SPEED CONTROL:-

Inside the player is an option to control the playback speed of on-demand videos. This allows viewers to slow videos down or do the opposite and speed up the rate of the video for playback.

CONTINUE WATCHING:-

Video on demand content has the playback position saved locally, and will start from the moment the viewer left off from if they return to the content within three months.

REGISTRATION GATE:-

Leverage a mandatory or optional registration gate, requiring users to enter details like their name, email address or phone number before accessing video content or attending an online event.

E-MAIL CONFIRMATION:-

When registration gate form is submitted, an email confirmation can be automatically sent to the registrant

CHAT MODULE:-

An embeddable text chat module can be shared on websites or via channel pages. Full moderation is available.

BREAKOUT CHAT ROOM:-

Spin off additional text chat rooms that live inside the same player experience, offering the ability for watch parties or separate topic discussion.

LIVE POLLING:-

Broadcasters can push a live poll inside the video player to audiences. Virtually real-time results are given.

CONCENTRATING INTO

1. Bandwidth Limitations:

Limited internet bandwidth can lead to buffering, lower video quality, and interruptions in streaming.

2. Latency:

Latency issues can cause delays in video playback and affect real-time interactions, such as live streaming or video conferencing.

3. Congestion:

Network congestion during peak usage times can result in slower streaming speeds and reduced video quality.

4. Codec Compatibility:

Incompatibility between codecs used by the streaming service and the viewer's device can lead to playback issues.

5. Device and Platform Fragmentation:

The wide variety of devices and platforms used for streaming (e.g., smartphones, smart TVs, gaming consoles) can make it challenging to ensure a consistent streaming experience across all devices.

6. Content Delivery:

Efficient content delivery is crucial. Content delivery networks (CDNs) help distribute content geographically to reduce latency and improve streaming quality.

7. Quality of Service (QoS):

QoS issues can impact streaming quality, especially in cases where network providers do not prioritize streaming traffic.

8. Buffering and Start-ups Delays:

Buffering delays can frustrate users, especially when content takes a long time to start playing.

9. Ad Insertion:

The insertion of ads during streaming can sometimes disrupt the viewing experience if not properly managed.

10. Content Piracy:

Protecting copyrighted content from piracy is an ongoing challenge for streaming providers.

11. Security Concerns:

Protecting user data and ensuring secure streaming experiences is vital, especially for paid streaming services.

12. Content Licensing:

Negotiating and maintaining licensing agreements with content providers can be complex and costly.

13. User Experience:

Providing a seamless and user-friendly interface for content discovery and playback is critical for retaining viewers.

14. Accessibility:

Ensuring that multimedia content is accessible to people with disabilities is a legal and ethical requirement in many regions.

15. Geographic Restrictions:

Some content may be subject to regional licensing restrictions, leading to limited availability in certain areas.

16. Live Streaming Challenges:

Live streaming involves real-time delivery, making it susceptible to issues like network instability and scalability challenges.

START UP PROGRAM FOR PROJECT USING VISUAL STUDIO CODE**room.html**

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
    <meta charset = 'utf = 8'>
```

```
    <meta http - equiv = 'x - UA - Compatible' content = 'IE = edge'>
```

```
    <title> Room </title>
```

```
    <meta name = " viewpoint' content = 'width = device - width, initial - scale = 1'>
```

```
    <link rel = 'stylesheet' type = 'text/css' media = 'screen href = 'styles/main.css'>
```

```
    <link rel = 'stylesheet' type = 'text/css' media = 'screen href = 'styles/room.css'>
```

```
</head>
```

```
<body>
```

```
    <header id = "nav">
```

```
        <h1> Cinimaful </h1>
```

```
    </header>
```

```
    <main id = "room__container">
```

```
        <section id = "participants__container">
```

```
            <div> id = "participants__heaader">
```

```
                <p>Room Members</p>
```

```
                <p>5</p>
```

```
            </div>
```

```
            <div class = "member__wrapper">
```

```
                <span class = "green_dot"></span>
```

```
                <p>Felix</p>
```

```
            </div>
```

```
<div class = "member__wrapper">
    <span class = "green_dot"></span>
    <p>Vishal</p>
</div>
```

```
<div class = "member__wrapper">
    <span class = "green_dot"></span>
    <p>Bhakkiyalakshmi</p>
</div>
```

```
<div class = "member__wrapper">
    <span class = "green_dot"></span>
    <p>Asmitha</p>
</div>
```

```
<div class = "member__wrapper">
    <span class = "green_dot"></span>
    <p>Iswarya</p>
</div>
```

```
</section>
```

```
<section id = "chat__container"></section>
```

```
<div class = "message_wrapper">
```

```
<p>Felix</p>
```

```
<p>class = "message">Stream And Host Your Video Reliably  
And Securely With Cinimaful.</p>
```

```
</div>
```

```
<div class = "message_wrapper">
```

```
  <p>Felix</p>
```

```
  <p>class = "message">Stream And Host Your Video Reliably  
  And Securely With Cinimaful.</p>
```

```
</div>
```

```
<div class = "message_wrapper">
```

```
  <p>Felix</p>
```

```
  <p>class = "message">Stream And Host Your Video Reliably  
  And Securely With Cinimaful.</p>
```

```
</div>
```

```
</main>
```

```
</body>
```

```
</html>
```

main.css

```
@import url
```

```
('https://fonts.googleapis.com/css2?family=Poppins;500;600;700;800&display=swap');
```

```
{
```

```
  font-family: 'Poppins', sans-serif;
```

```
}
```

```
body
```

```
{
```

```
  background-color: #1a1a1a;
```

```
  font: 14px;
```

```
  padding: 0;
```

```
margin: 0;
box-sizing: border-box;
}

#nav
{
    background-color: #1a1a1a;
    height: 50px;
    border-bottom: 1px solid #000;
}

.green__dot
{
    height: 10px;
    width: 10px;
    background-color: #zaca3e;
    border-radius: 50%;
}
```

room.css

```
#room__container
{
    display: grid;
    grid-template-columns: 200px 800px;
}

#participants__container
```

```
{  
    border - right: #797a79;  
}  
  
.message__wrapper  
{  
    background-color: #363739;  
    border-radius: 10px;  
    padding: 10px;  
    margin top: 10px;  
}
```

----- END OF THE DOCUMENT -----

MEDIA STREAMING WITH IBM CLOUD STREAMING

PHASE – 3

DEVELOPMENT PHASE – 1

Date	23 October 2023
Team ID	5566
Team Name	Proj 227254 Team 2
Project Name	Media streaming with IBM Cloud streaming

1.INTRODUCION:

Streaming files like audio, video and others are stored on a server somewhere on the world wide web(WWW). When a user request file ,It gets transmitted over the web as sequential packets of data that are streamed instantly . Since streaming data is broken down into data packets ,its transmission is similar to that of other types of data sent over the internet.

An audio or video player hosted by the browser accepts the flow of data packets from the streaming service's remote server and interprets them as video or audio, then plays the media for the user.

STREAMING REQUIREMENTS:

Streaming usually requires a reliable, high-speed internet connection because the media files must be retrieved from a remote location and then delivered to a user's local system with minimal lag or latency (delay). A slow connection decreases the speed at which the content is delivered, affecting the user's streaming experience.

INSTALL PYTHON IN OS:

1. Download PyCharm:

Visit the JetBrains website (<https://www.jetbrains.com/pycharm/download/>) and download the community (free) or professional version of PyCharm, depending on your needs. Make sure to download the Windows version.

2. Run the Installer:

Locate the downloaded installer (usually an .exe file) and double-click it to run the installation.

3. Choose Installation Type:

During the installation, you'll be prompted to select the installation type. You can choose the default settings or customize them according to your preferences. You can also select the option to create associations for .py files, which allows you to open Python scripts with PyCharm by default.

4. Select the Destination Folder:

Choose the folder where you want to install PyCharm, or leave it at the default location.

5. Create Shortcuts:

You can choose whether you want to create desktop shortcuts or add PyCharm to the Start menu.

6. Complete the Installation:

Click the "Install" button to start the installation process. PyCharm will be installed on your system.

7. Launch PyCharm:

Once the installation is complete, you can launch PyCharm by checking the "Run PyCharm" option in the installer or by finding it in your Start menu or desktop shortcuts.

INSTALLING PACKAGES :

1.PACKAGE NAME: Flask

USE : to use flask framework in python

COMMAND TO INSTALL: pip install flask

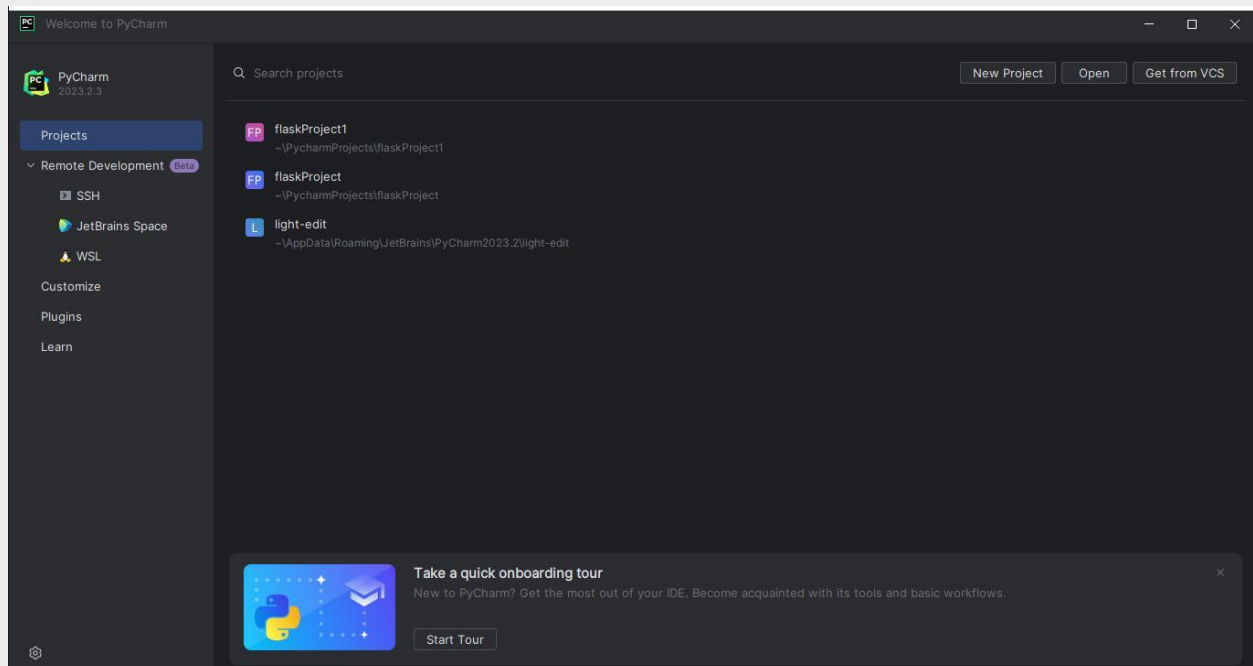
```
Command Prompt
Collecting flask
  Downloading flask-3.0.0-py3-none-any.whl (99 kB)
----- 99.7/99.7 kB 5.6 MB/s eta 0:00:00
Collecting Werkzeug>=3.0.0
  Downloading werkzeug-3.0.0-py3-none-any.whl (226 kB)
----- 226.6/226.6 kB 13.5 MB/s eta 0:00:00
Collecting Jinja2>=3.1.2
  Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)
----- 133.1/133.1 kB ? eta 0:00:00
Collecting itsdangerous>=2.1.2
  Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting click>=8.1.3
  Downloading click-8.1.7-py3-none-any.whl (97 kB)
----- 97.9/97.9 kB 5.5 MB/s eta 0:00:00
Collecting blinker>=1.6.2
  Downloading blinker-1.6.3-py3-none-any.whl (13 kB)
Collecting colorama
  Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.1.3-cp311-cp311-win_amd64.whl (17 kB)
Installing collected packages: MarkupSafe, itsdangerous, colorama, blinker, Werkzeug, Jinja2, click, flask
Successfully installed Jinja2-3.1.2 MarkupSafe-2.1.3 Werkzeug-3.0.0 blinker-1.6.3 click-8.1.7 colorama-0.4.6 flask-3.0.0 itsdangerous-2.1.2

[notice] A new release of pip available: 22.3 -> 23.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\Elcot>
```

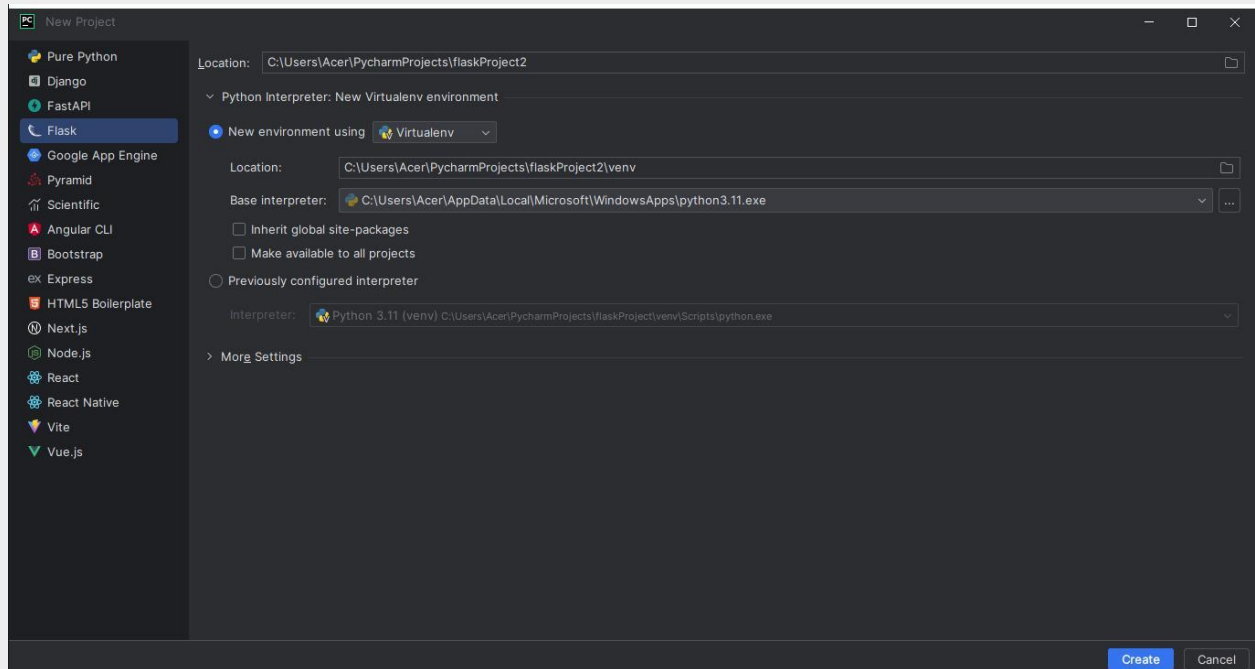
GETTING STARTED WITH PYCHARM:

1. To create a new project click on new project:

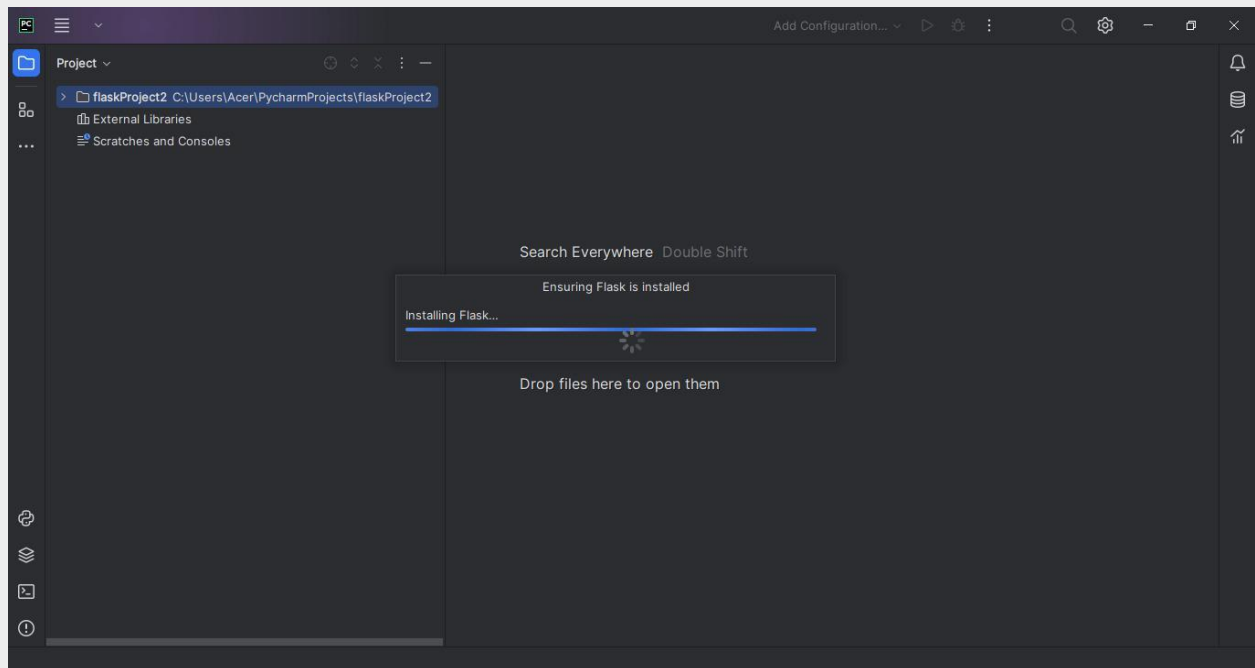


2. Select flask to create a project in flask program
 - Select your environment as you need.

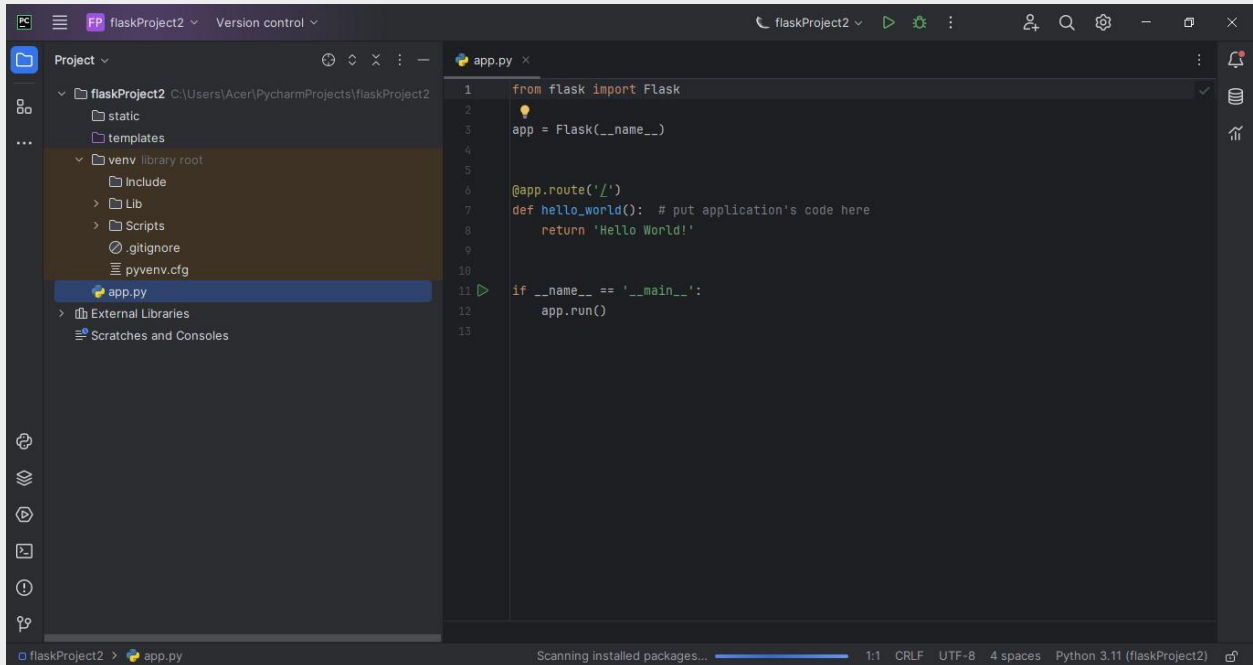
- Select the required path you need the projects to be stored.



3. Pycharm will automatically installs required packages for flask automatically.



4. Required folders will be installed automatically. Then we can ready to code successfully.



STEPS TO CREATE LOGIN AND REGISTRATION PAGE:

1. Code this to render a html code in flask

```
from flask import Flask,render_template
```

```
app = Flask(__name__)
```

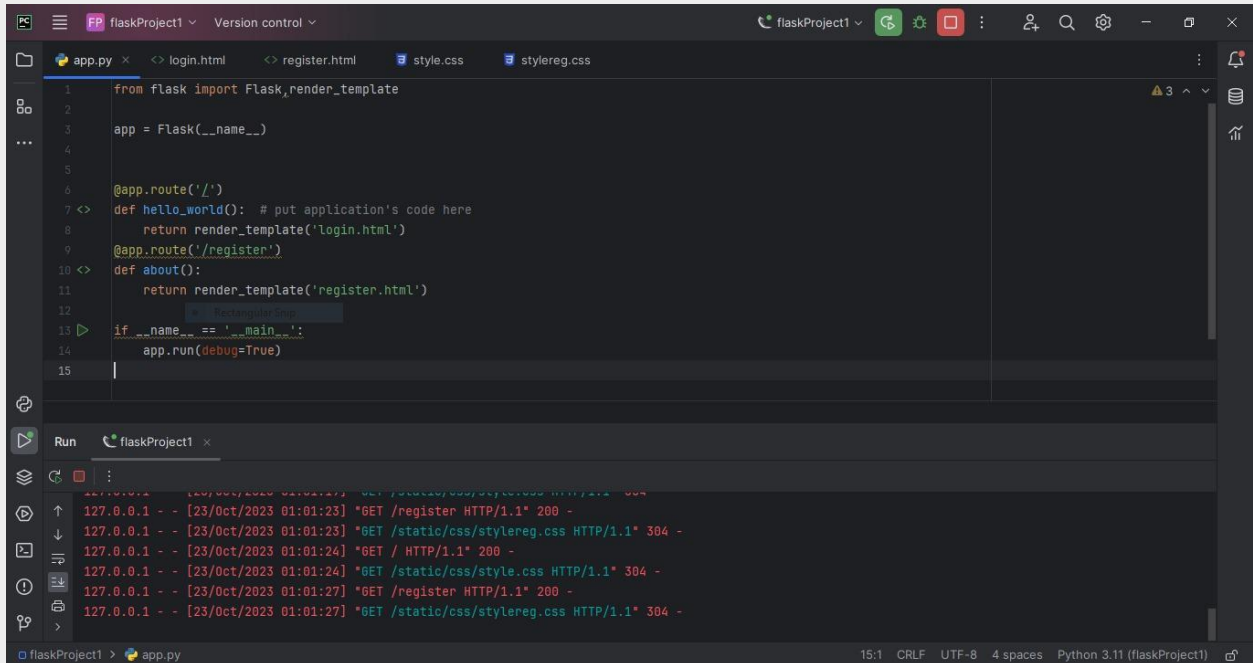
```
@app.route('/')
```

```
def hello_world(): # put application's code here
    return render_template('login.html')
```

```
@app.route('/register')
```

```
def about():
    return render_template('register.html')
```

```
if __name__ == '__main__':
    app.run(debug=True)
```



```
1 from flask import Flask, render_template
2
3 app = Flask(__name__)
4
5
6 @app.route('/')
7 def hello_world(): # put application's code here
8     return render_template('login.html')
9
10 @app.route('/register')
11 def about():
12     return render_template('register.html')
13
14 if __name__ == '__main__':
15     app.run(debug=True)
```

Run flaskProject1

```
127.0.0.1 - - [23/Oct/2023 01:01:23] "GET /register HTTP/1.1" 200 -
127.0.0.1 - - [23/Oct/2023 01:01:23] "GET /static/css/stylereg.css HTTP/1.1" 304 -
127.0.0.1 - - [23/Oct/2023 01:01:24] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [23/Oct/2023 01:01:24] "GET /static/css/style.css HTTP/1.1" 304 -
127.0.0.1 - - [23/Oct/2023 01:01:27] "GET /register HTTP/1.1" 200 -
127.0.0.1 - - [23/Oct/2023 01:01:27] "GET /static/css/stylereg.css HTTP/1.1" 304 -
```

2. Code for login page html:

<!DOCTYPE html>

<html>

<head>

<title>Slide Navbar</title>

<link rel="stylesheet" type="text/css" href="slide navbar style.css">

<link

href="https://fonts.googleapis.com/css2?family=Jost:wght@500&display=swap" rel="stylesheet">

</head>

<body>

<div class="main">

```
<input type="checkbox" id="chk" aria-hidden="true">
```

```
<div class="signup">
```

```
<form>
```

```
<label for="chk" aria-hidden="true">Sign  
up</label>
```

```
<input type="text" name="txt" placeholder="User  
name" required="">
```

```
<input type="email" name="email"  
placeholder="Email" required="">
```

```
<input type="password" name="pswd"  
placeholder="Password" required="">
```

```
<button>Sign up</button>
```

```
</form>
```

```
</div>
```

```
<div class="login">
```

```
<form>
```

```
<label for="chk" aria-hidden="true">Login</label>
```

```
<input type="email" name="email"  
placeholder="Email" required="">
```

```
<input type="password" name="pswd"  
placeholder="Password" required="">
```

```
<button>Login</button>
```

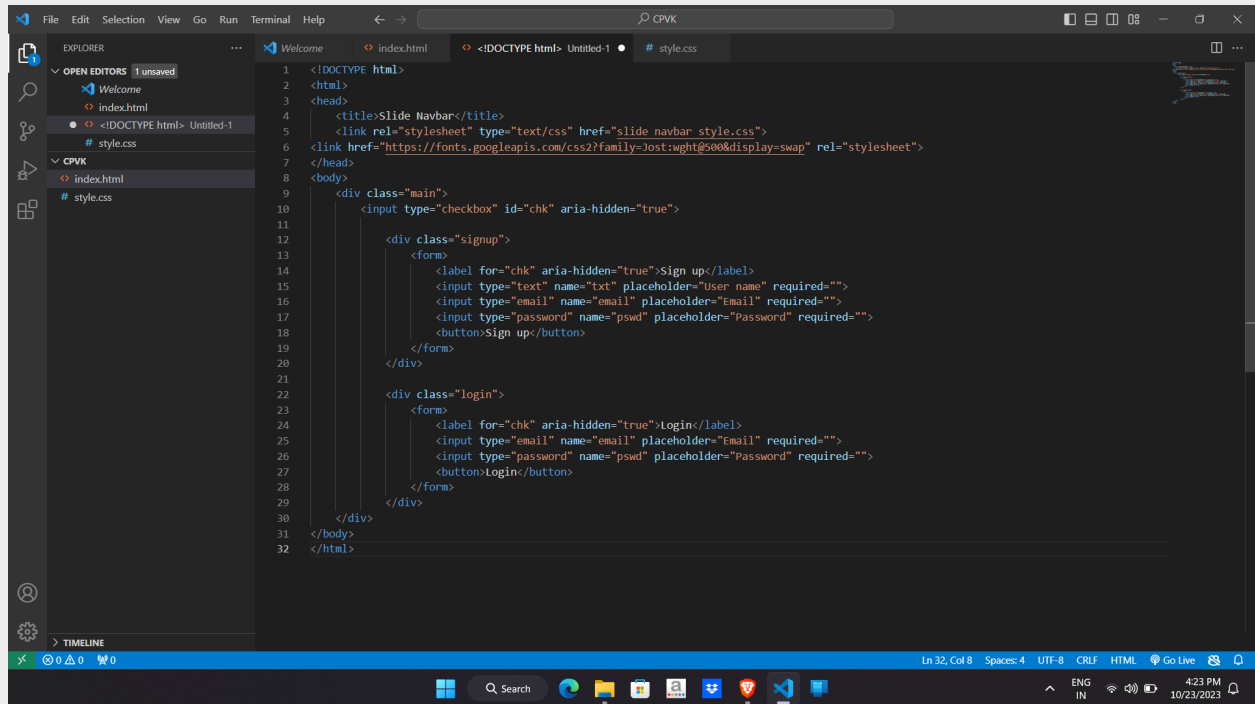
```
</form>
```

</div>

</div>

</body>

</html>



3. SYNCHRONIZING A CSS FILE WITH LOGIN HTML FOR DESIGNING PURPOSE:

body{

margin: 0;

padding: 0;

```
display: flex;

justify-content: center;

align-items: center;

min-height: 100vh;

font-family: 'Jost', sans-serif;

background: linear-gradient(to bottom, #0f0c29, #302b63, #24243e);
}

.main{

width: 350px;

height: 500px;

background: red;

overflow: hidden;

background:
url("https://doc-08-2c-docs.googleusercontent.com/docs/securesc/68c90smigl
ihng9534mvqmq1946dmis5/fo0picsp1nhiucmc0l25s29respgpr4j/16315242750
00/03522360960922298374/03522360960922298374/1Sx0jhdpEpnNIydS4rnN
4kHSJtU1EyWka?e=view&authuser=0&nonce=gcrocepgbb17m&user=0352236
0960922298374&hash=tfhgbs86ka6divo3llbvp93mg4csvb38") no-repeat
center/ cover;

border-radius: 10px;

box-shadow: 5px 20px 50px #000;
}

#chk{

display: none;
}
```



```
.signup{  
    position: relative;  
    width:100%;  
    height: 100%;  
}  
label{  
    color: #fff;  
    font-size: 2.3em;  
    justify-content: center;  
    display: flex;  
    margin: 60px;  
    font-weight: bold;  
    cursor: pointer;  
    transition: .5s ease-in-out;  
}  
input{  
    width: 60%;  
    height: 20px;  
    background: #e0dede;  
    justify-content: center;  
    display: flex;  
    margin: 20px auto;  
    padding: 10px;
```

```
border: none;

outline: none;

border-radius: 5px;
}

button{

width: 60%;

height: 40px;

margin: 10px auto;

justify-content: center;

display: block;

color: #fff;

background: #573b8a;

font-size: 1em;

font-weight: bold;

margin-top: 20px;

outline: none;

border: none;

border-radius: 5px;

transition: .2s ease-in;

cursor: pointer;

}

button:hover{

background: #6d44b8;
```

```
}
```

```
.login{
```

```
    height: 460px;
```

```
    background: #eee;
```

```
    border-radius: 60% / 10%;
```

```
    transform: translateY(-180px);
```

```
    transition: .8s ease-in-out;
```

```
}
```

```
.login label{
```

```
    color: #573b8a;
```

```
    transform: scale(.6);
```

```
}
```

```
#chk:checked ~ .login{
```

```
    transform: translateY(-500px);
```

```
}
```

```
#chk:checked ~ .login label{
```

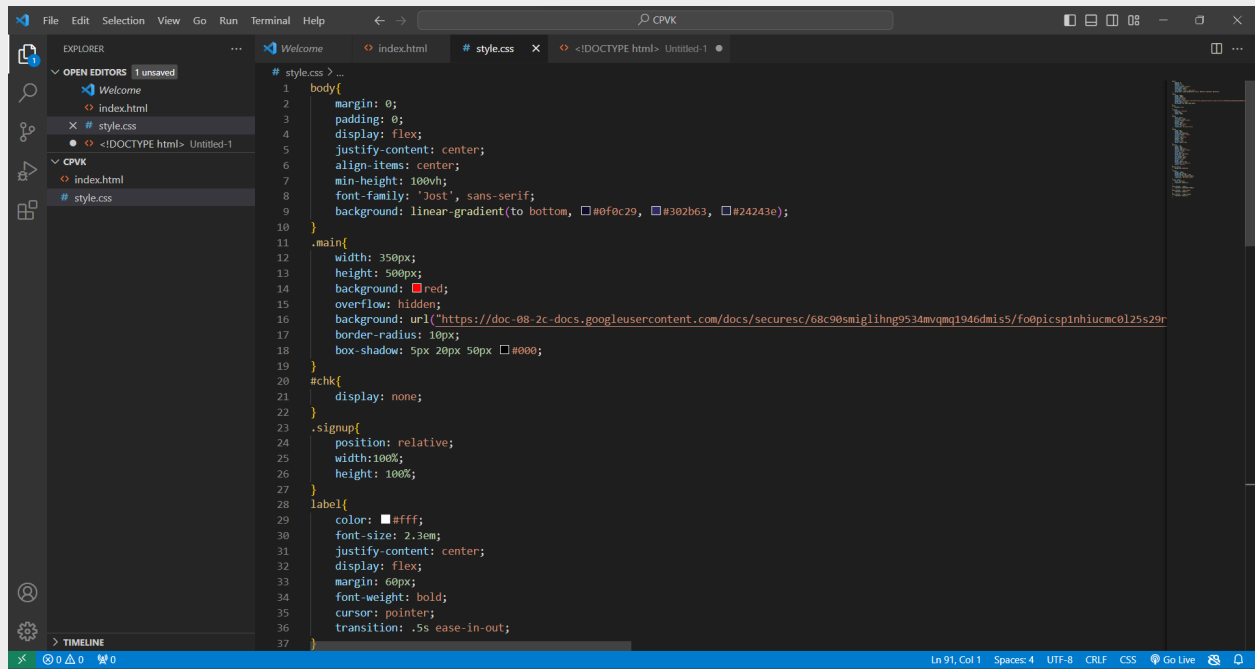
```
    transform: scale(1);
```

```
}
```

```
#chk:checked ~ .signup label{
```

```
    transform: scale(.6);
```

```
}
```



4. CREATING A SQL TO STORE THE DATA'S IN THE DATABASE:

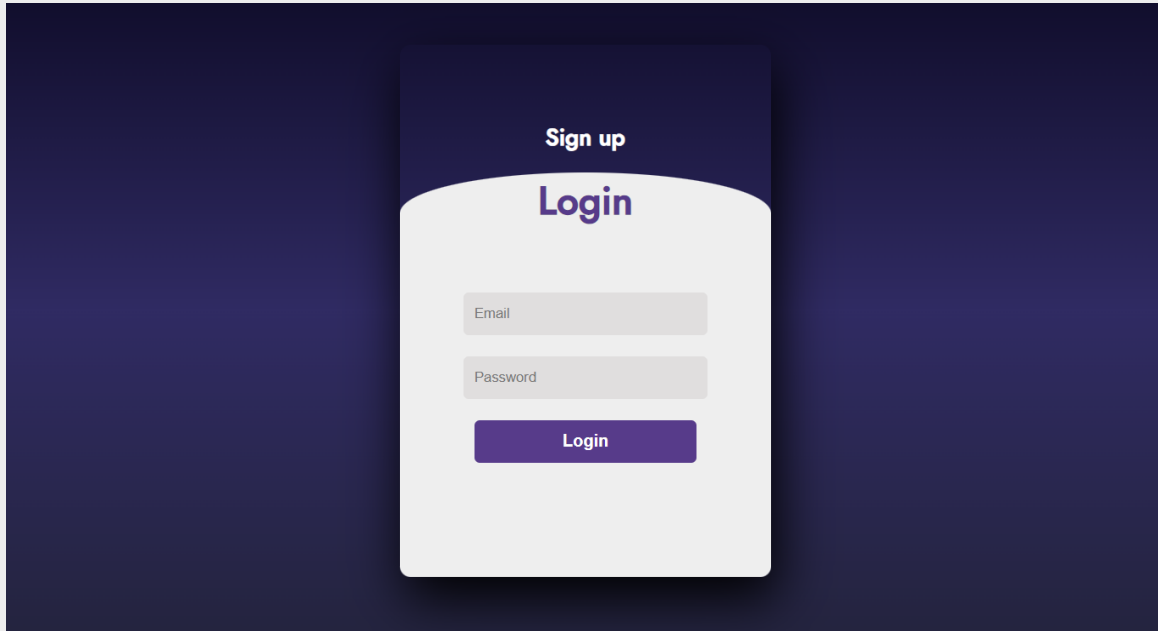
CREATE DATABASE IF NOT EXISTS UserDatabase;

USE UserDatabase;

CREATE TABLE IF NOT EXISTS Users (
 UserID INT AUTO_INCREMENT PRIMARY KEY,
 Username VARCHAR(50) NOT NULL,
 MobileNo VARCHAR(15) NOT NULL,
 EmailID VARCHAR(100) NOT NULL,
 Password VARCHAR(255) NOT NULL

);

5. FINAL OUTPUT FOR LOGIN PAGE:



The image displays a login page with a dark blue background. A white, rounded rectangular card is centered on the page. At the top of the card, the text "Sign up" is written in white, and below it, the word "Login" is written in a larger, bold, dark blue font. Below the text, there are two input fields: "Email" and "Password", both with light gray borders. Below the input fields is a dark blue button with the word "Login" written in white. The card has a subtle drop shadow against the background.

6. FINAL OUTPUT FOR REGISTRATION PAGE:

The image displays a registration page with a dark blue background. A central, lighter blue rounded rectangle contains the registration form. At the top of this rectangle is the title "Sign up" in white. Below the title are three input fields, each with a light gray border and a light gray placeholder text: "User name", "Email", and "Password". Below these fields is a purple button with the text "Sign up" in white. At the bottom of the central rectangle is a white rounded rectangle containing the text "Login" in purple.

Sign up

User name

Email

Password

Sign up

Login

MEDIA STREAMING WITH IBM CLOUD STREAMING

PHASE – 4

DEVELOPMENT PHASE – II

THESE CODES ARE CONTINUATION OF DEVELOPMENT PHASE -1

STEPS TO BE IMPLEMENTED

1. CODE TO RENDER HTML IN FLASK:

```
from flask import Flask,render_template, request, redirect, url_for,  
send_from_directory
```

```
import os
```

```
app = Flask(__name__)
```

```
app.config['UPLOAD_FOLDER'] = 'upload'
```

```
app.static_folder = 'static'
```

```
@app.route('/')
```

```
def hello_world(): # put application's code here
```

```
    return render_template('login.html')
```

```
@app.route('/register')
```

```
def about():
```

```
    return render_template('register.html')
```

```
@app.route('/video')
def upload_form():
    return render_template('upload.html')

@app.route('/upload', methods=['POST'])
def upload_video():
    if 'video' not in request.files:
        return redirect(request.url)

    video = request.files['video']

    if video.filename == '':
        return redirect(request.url)

    if video:
        video.save(os.path.join(app.config['UPLOAD_FOLDER'],
video.filename))
        return redirect(url_for('view_video', filename=video.filename))

@app.route('/upload/<filename>')
def view_video(filename):
    return render_template('view.html', filename=filename)

#@app.route('/upload/<filename>')
#def send_video(filename):
```



```
# return send_from_directory(app.config['UPLOAD_FOLDER'], filename)*/
```

```
@app.route('/upload/<filename>')
```

```
def send_video(filename):
```

```
    return send_from_directory(app.config['UPLOAD_FOLDER'], filename,  
                               mimetype='video/mp4')
```

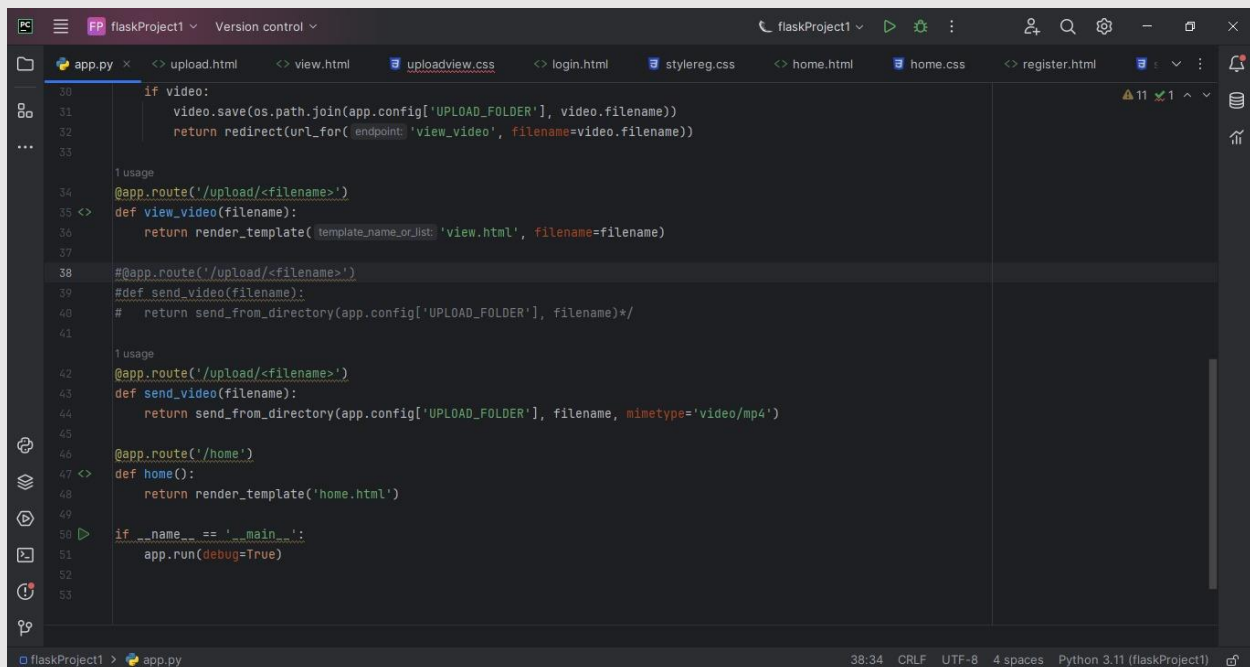
```
@app.route('/home')
```

```
def home():
```

```
    return render_template('home.html')
```

```
if __name__ == '__main__':
```

```
    app.run(debug=True)
```

A screenshot of a code editor window titled 'flaskProject1'. The editor shows a Python file 'app.py' with the following code:

```
30 if video:  
31     video.save(os.path.join(app.config['UPLOAD_FOLDER'], video.filename))  
32     return redirect(url_for('view_video', filename=video.filename))  
33  
34 1 usage  
35 @app.route('/upload/<filename>')  
36 def view_video(filename):  
37     return render_template(template_name_or_list='view.html', filename=filename)  
38  
39 # @app.route('/upload/<filename>')  
40 # def send_video(filename):  
41 #     return send_from_directory(app.config['UPLOAD_FOLDER'], filename)*/  
42  
43 1 usage  
44 @app.route('/upload/<filename>')  
45 def send_video(filename):  
46     return send_from_directory(app.config['UPLOAD_FOLDER'], filename, mimetype='video/mp4')  
47  
48 @app.route('/home')  
49 def home():  
50     return render_template('home.html')  
51  
52 if __name__ == '__main__':  
53     app.run(debug=True)
```

 The editor interface includes a sidebar with file explorer, a top toolbar with various icons, and a bottom status bar showing '38:34 CRLF UTF-8 4 spaces Python 3.11 (flaskProject1)'.

2. HTML CODE TO CREATE A HOMEPAGE:

```
<!DOCTYPE html>

<html lang="en" dir="ltr">

  <head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-
scale=1.0">

    <title> Website Layout | CodingLab</title>

    <!link rel="stylesheet" href="css/home.css">

    <link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename='css/home.css') }}">

  </head>

<body>

  <nav>

    <div class="menu">

      <div class="logo">

        <a href="#">IBM Media Streaming</a>

      </div>

      <ul>

        <li><a href="#">Home</a></li>

        <li><a href="#">About</a></li>

        <li><a href="/video">upload</a></li>

        <li><a href="#">profile</a></li>
```

```
<li><a href="#">Feedback</a></li>

</ul>

</div>

</nav>

<div class="img"></div>

<div class="center">

  <div class="title">WELCOME TO IBM MEDIA STREAMING</div>

  <div class="sub_title">Watch ,Hear ,Enjoy...</div>

  <div class="btns">

    <button>watch videos</button>

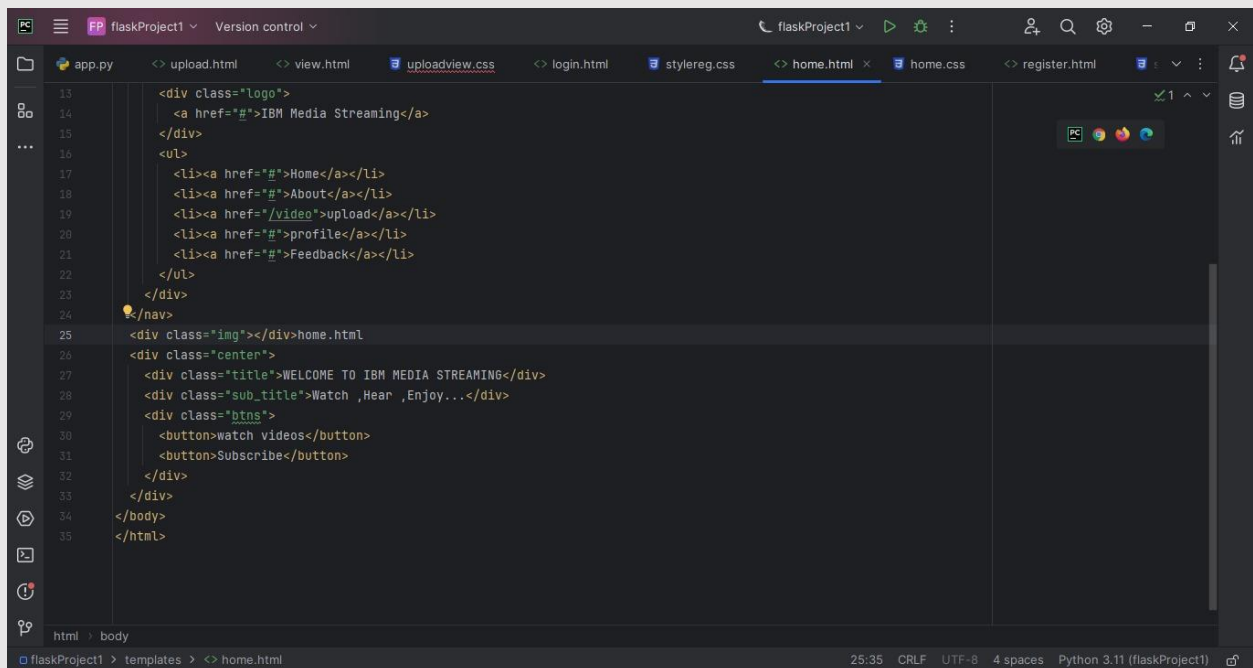
    <button>Subscribe</button>

  </div>

</div>

</body>

</html>
```



```
13 <div class="logo">
14   <a href="#">IBM Media Streaming</a>
15 </div>
16 <ul>
17   <li><a href="#">Home</a></li>
18   <li><a href="#">About</a></li>
19   <li><a href="/video">upload</a></li>
20   <li><a href="#">profile</a></li>
21   <li><a href="#">Feedback</a></li>
22 </ul>
23 </div>
24 </nav>
25 <div class="img"></div>home.html
26 <div class="center">
27   <div class="title">WELCOME TO IBM MEDIA STREAMING</div>
28   <div class="sub_title">Watch ,Hear ,Enjoy...</div>
29   <div class="btns">
30     <button>watch videos</button>
31     <button>Subscribe</button>
32   </div>
33 </div>
34 </body>
35 </html>
```

3. CSS CODE TO CREATE A HOMEPAGE:

```
@import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@200;300;400;
500;600;700&display=swap');
*{
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: 'Poppins',sans-serif;
}
::selection{
  color: #000;
  background: #fff;
}
nav{
  position: fixed;
  background: #1b1b1b;
  width: 100%;
  padding: 10px 0;
  z-index: 12;
}
nav .menu{
  max-width: 1250px;
  margin: auto;
  display: flex;
  align-items: center;
  justify-content: space-between;
  padding: 0 20px;
}
.menu .logo a{
  text-decoration: none;
  color: #fff;
  font-size: 35px;
  font-weight: 600;
```

```
}  
.menu ul{  
  display: inline-flex;  
}  
.menu ul li{  
  list-style: none;  
  margin-left: 7px;  
}  
.menu ul li:first-child{  
  margin-left: 0px;  
}  
.menu ul li a{  
  text-decoration: none;  
  color: #fff;  
  font-size: 18px;  
  font-weight: 500;  
  padding: 8px 15px;  
  border-radius: 5px;  
  transition: all 0.3s ease;  
}  
.menu ul li a:hover{  
  background: #fff;  
  color: black;  
}  
.img{  
  /*background: url('static/images.jpg') no-repeat; bv  
  width: 100%;  
  height: 100vh;  
  background-size: cover;  
  background-position: center;  
  position: relative;*/  
  background-color: rgba(0, 255, 102, 0.82); /* Green background color */  
  width: 100%;  
  height: 100vh;  
  position: relative;
```

```
}  
.img::before{  
  content: "";  
  position: absolute;  
  height: 100%;  
  width: 100%;  
  background: rgba(0, 0, 0, 0.4);  
}  
.center{  
  position: absolute;  
  top: 52%;  
  left: 50%;  
  transform: translate(-50%, -50%);  
  width: 100%;  
  padding: 0 20px;  
  text-align: center;  
}  
.center .title{  
  color: #fff;  
  font-size: 55px;  
  font-weight: 600;  
}  
.center .sub_title{  
  color: #fff;  
  font-size: 52px;  
  font-weight: 600;  
}  
.center .btns{  
  margin-top: 20px;  
}  
.center .btns button{  
  height: 55px;  
  width: 170px;  
  border-radius: 5px;  
  border: none;
```

```

margin: 0 10px;
border: 2px solid white;
font-size: 20px;
font-weight: 500;
padding: 0 10px;
cursor: pointer;
outline: none;
transition: all 0.3s ease;
}

.center .btns button:first-child{
  color: #fff;
  background: none;
}

.btns button:first-child:hover{
  background: white;
  color: black;
}

.center .btns button:last-child{
  background: white;
  color: black;
}

```

```

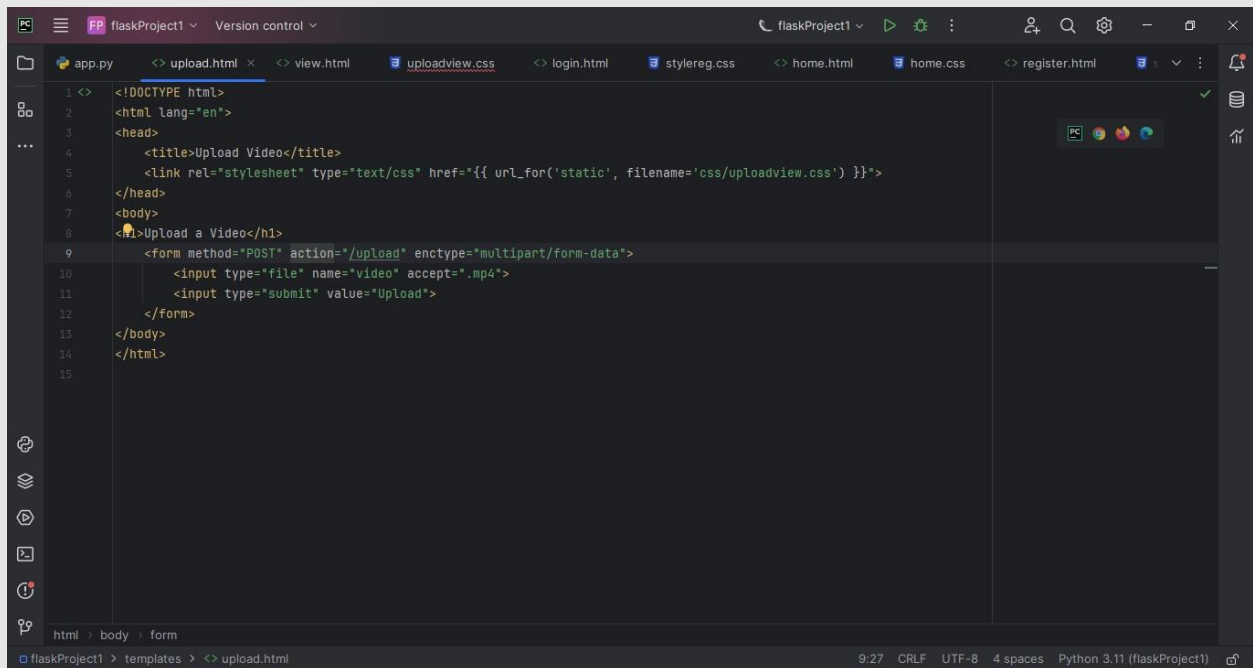
48 padding: 8px 15px;
49 border-radius: 5px;
50 transition: all 0.3s ease;
51 }
52 .menu ul li a:hover{
53   background: #fff;
54   color: black;
55 }
56 .img{
57   /*background: url('static/images.jpg') no-repeat; b
58   width: 100%;
59   height: 100vh;
60   background-size: cover;
61   background-position: center;
62   position: relative;*/
63   background-color: rgba(0, 255, 102, 0.82); /* Green background color */
64   width: 100%;
65   height: 100vh;
66   position: relative;
67 }
68 .img::before{
69   content: '';
70   position: absolute;
71   height: 100%;
72   width: 100%;
73   background: rgba(0, 0, 0, 0.4);
74 }

```

flaskProject1 > static > css > home.css 57:55 CRLF UTF-8 2 spaces* Python 3.11 (flaskProject1)

4. HTML CODE TO UPLOAD A DATA IN STREAMING SITE:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Upload Video</title>
  <link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename='css/uploadview.css') }}">
</head>
<body>
<h1>Upload a Video</h1>
  <form method="POST" action="/upload" enctype="multipart/form-
data">
    <input type="file" name="video" accept=".mp4">
    <input type="submit" value="Upload">
  </form>
</body>
</html>
```



5. HTML TO STREAM A DATA IN STREAMING SITE:

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .highlight-video {
      border: 2px solid rgba(0, 255, 183, 0.61); /* Add a red border around
the video */
      animation-name: none
    }
  </style>

  <link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename='css/styles.css') }}">
</head>
<body>
  <!-- Background Container -->
  <div class="background-container">
    <!-- Stars -->
    <div class="stars"></div>

    <!-- Twinkling Stars -->
    <div class="twinkling"></div>

    <!-- Clouds -->
    <div class="clouds"></div>

    <!-- Your Video Element -->

  </div>

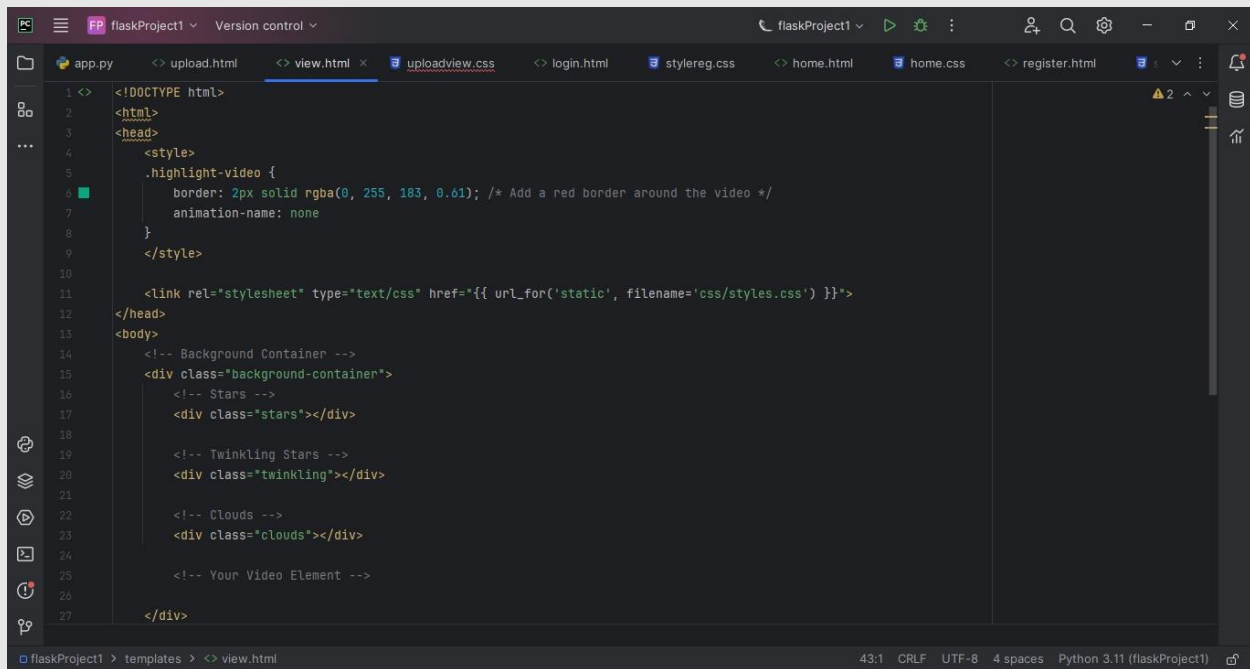
  <div class="video-container">
    <video class="highlight-video" width="640" height="480" controls >
```

```

        <source src="{{ url_for('send_video', filename=filename) }}"
type="video/mp4">
    </video>
</div>
<!-- Your regular page content goes here -->
<div class="page-content">
    <!-- More content, other sections, etc. -->
</div>

</body>
</html>

```



6. CSS CODE FOR BOTH UPLOAD AND STREAMING A VIDEO:

```

@keyframes move-background {
  from {
    -webkit-transform: translate3d(0px, 0px, 0px);
  }
  to {
    -webkit-transform: translate3d(1000px, 0px, 0px);
  }
}

```

```
}
```

```
@-webkit-keyframes move-background {  
  from {  
    -webkit-transform: translate3d(0px, 0px, 0px);  
  }  
  to {  
    -webkit-transform: translate3d(1000px, 0px, 0px);  
  }  
}
```

```
@-moz-keyframes move-background {  
  from {  
    -webkit-transform: translate3d(0px, 0px, 0px);  
  }  
  to {  
    -webkit-transform: translate3d(1000px, 0px, 0px);  
  }  
}
```

```
.background-container {  
  position: fixed;  
  top: 0;  
  left: 0;  
  bottom: 0;  
  right: 0;  
}
```

```
.stars {  
  background: black url(https://s3-us-west-2.amazonaws.com/s.cdpn.io/1231630/stars.png) repeat;  
  position: absolute;  
  top: 0;  
  bottom: 0;  
  left: 0;
```

```
right: 0;
display: block;
z-index: 0;
}
```

```
.twinkling {
width: 10000px;
height: 100%;
background: transparent url("https://s3-us-west-
2.amazonaws.com/s.cdpn.io/1231630/twinkling.png") repeat;
background-size: 1000px 1000px;
position: absolute;
right: 0;
top: 0;
bottom: 0;
z-index: 2;
}
```

```
-moz-animation: move-background 70s linear infinite;
-ms-animation: move-background 70s linear infinite;
-o-animation: move-background 70s linear infinite;
-webkit-animation: move-background 70s linear infinite;
animation: move-background 70s linear infinite;
}
```

```
.clouds {
width: 10000px;
height: 100%;
background: transparent url("https://s3-us-west-
2.amazonaws.com/s.cdpn.io/1231630/clouds_repeat.png") repeat;
background-size: 1000px 1000px;
position: absolute;
right: 0;
top: 0;
bottom: 0;
z-index: 3;
}
```

```

-moz-animation: move-background 150s linear infinite;
-ms-animation: move-background 150s linear infinite;
-o-animation: move-background 150s linear infinite;
-webkit-animation: move-background 150s linear infinite;
animation: move-background 150s linear infinite;
}

.video-container {
    position: relative;
}

/* CSS rule for highlighting the video */
.highlight-video {
    border: 2px solid red;
}

video {
    height: 70vh;
    width: 70vh;
    position: absolute;
    z-index: 3;
    right: 20px;
}

```

The screenshot shows a code editor with a dark theme. The active file is `uploadview.css`. The code defines a keyframe animation `move-background` and a `.video-container` class. The animation moves a background image from the left to the right over 150 seconds. The video container is positioned relative, and the video element is absolutely positioned on the right side of the container.

```

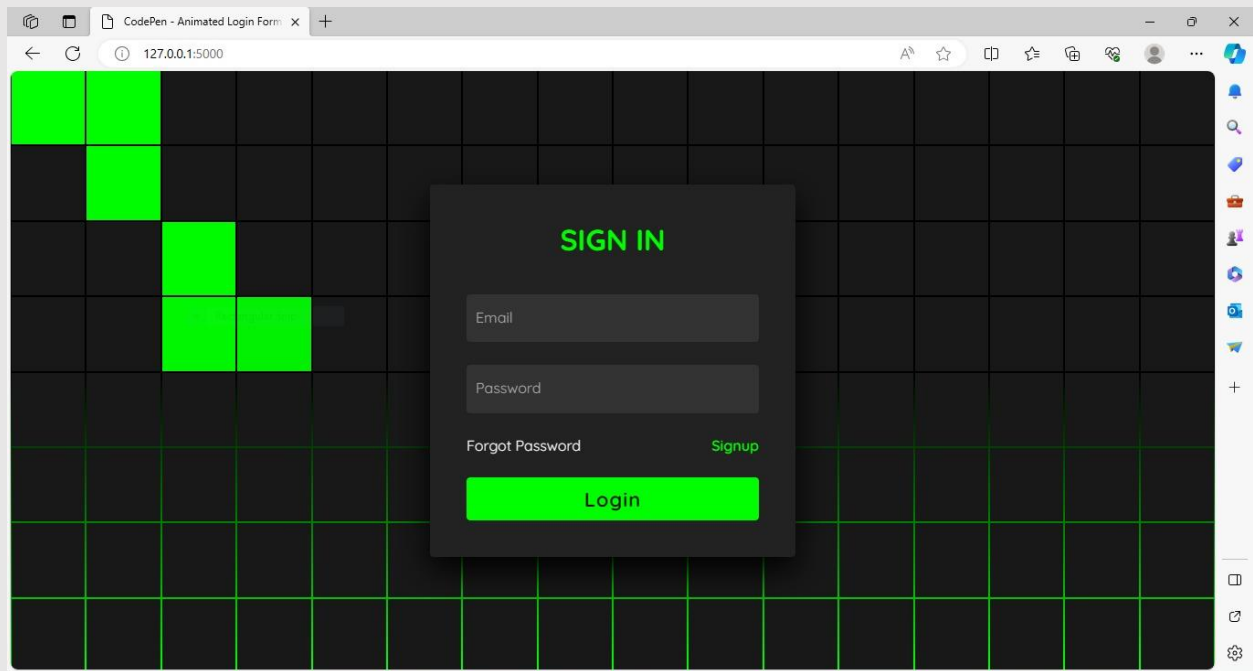
1  /* styles.css */
2  @keyframes move-background {
3      from {
4          -webkit-transform: translate3d(0px, 0px, 0px);
5      }
6      to {
7          -webkit-transform: translate3d(1000px, 0px, 0px);
8      }
9  }
10
11 @-webkit-keyframes move-background {
12     from {
13         -webkit-transform: translate3d(0px, 0px, 0px);
14     }
15     to {
16         -webkit-transform: translate3d(1000px, 0px, 0px);
17     }
18 }
19
20 @-moz-keyframes move-background {
21     from {
22         -webkit-transform: translate3d(0px, 0px, 0px);
23     }
24     to {
25         -webkit-transform: translate3d(1000px, 0px, 0px);
26     }
27 }
28
29 .video-container

```

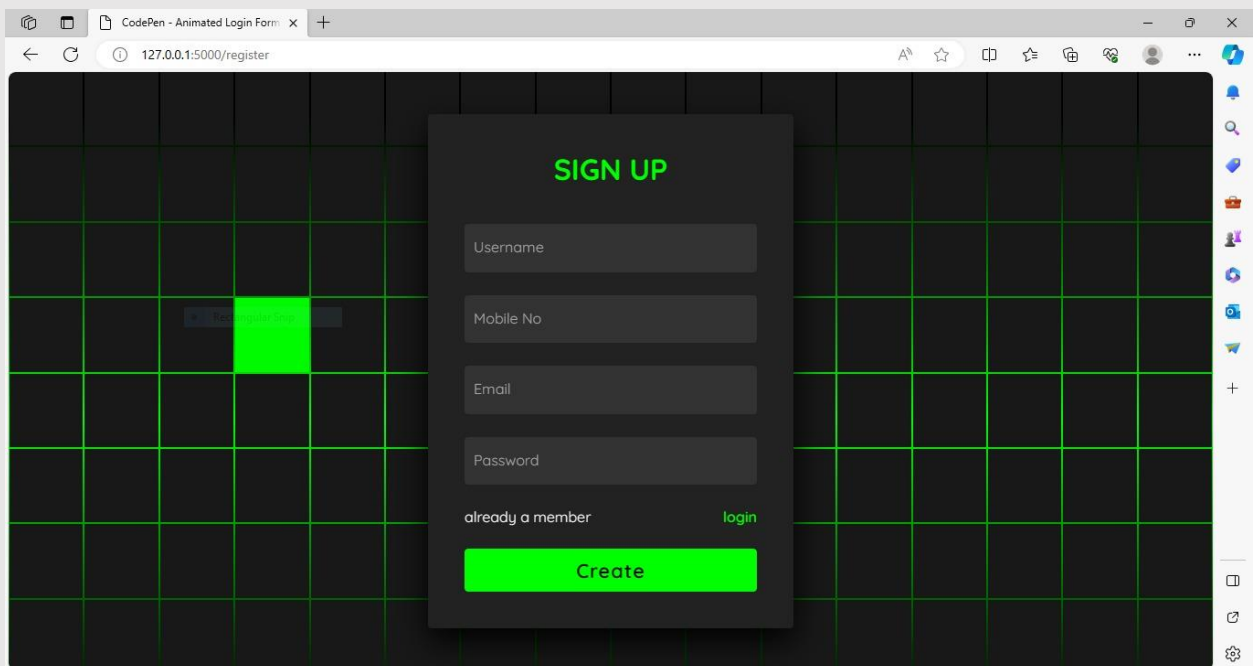
The editor's status bar at the bottom indicates the file path `flaskProject1 > static > css > uploadview.css`, the character count `84:19 (1 char)`, the line ending `CRLF`, the encoding `UTF-8`, the indentation `2 spaces*`, and the Python version `Python 3.11 (flaskProject1)`.

FINAL OUTPUT FOR ALL PROGRAMS:

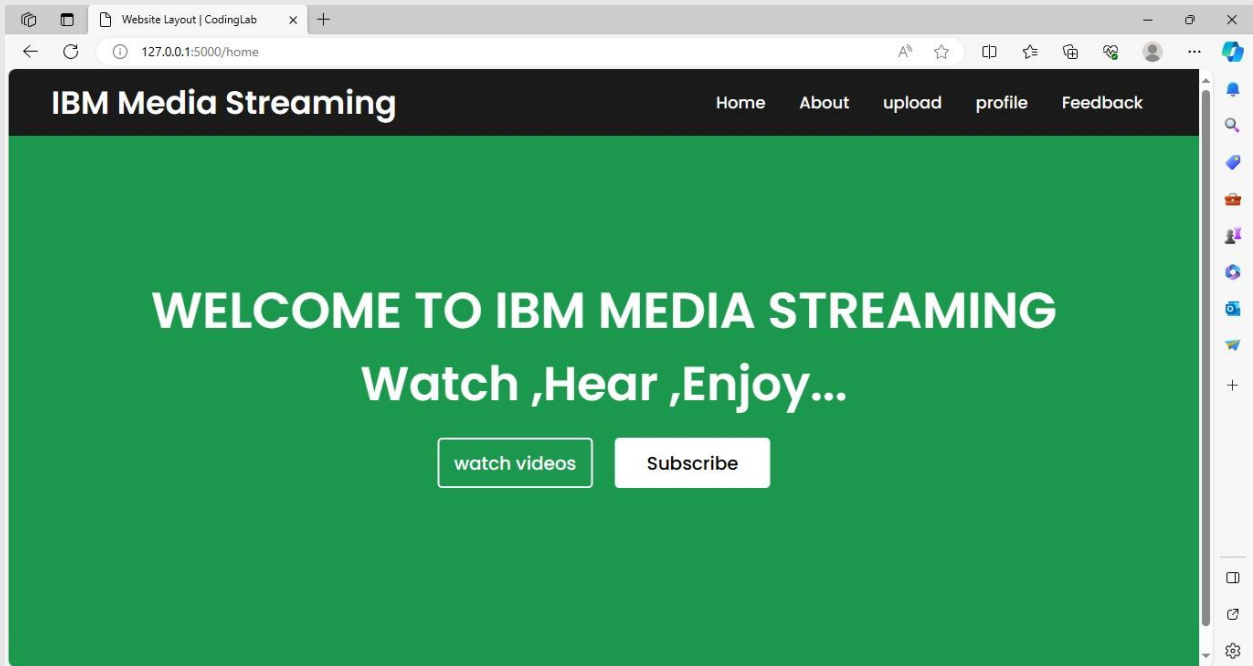
1. LOGIN PAGE TO ENTER INTO THE HOME PAGE:



2. SIGN UP PAGE TO CREATE AN ACCOUNT TO BE LOGGED IN:

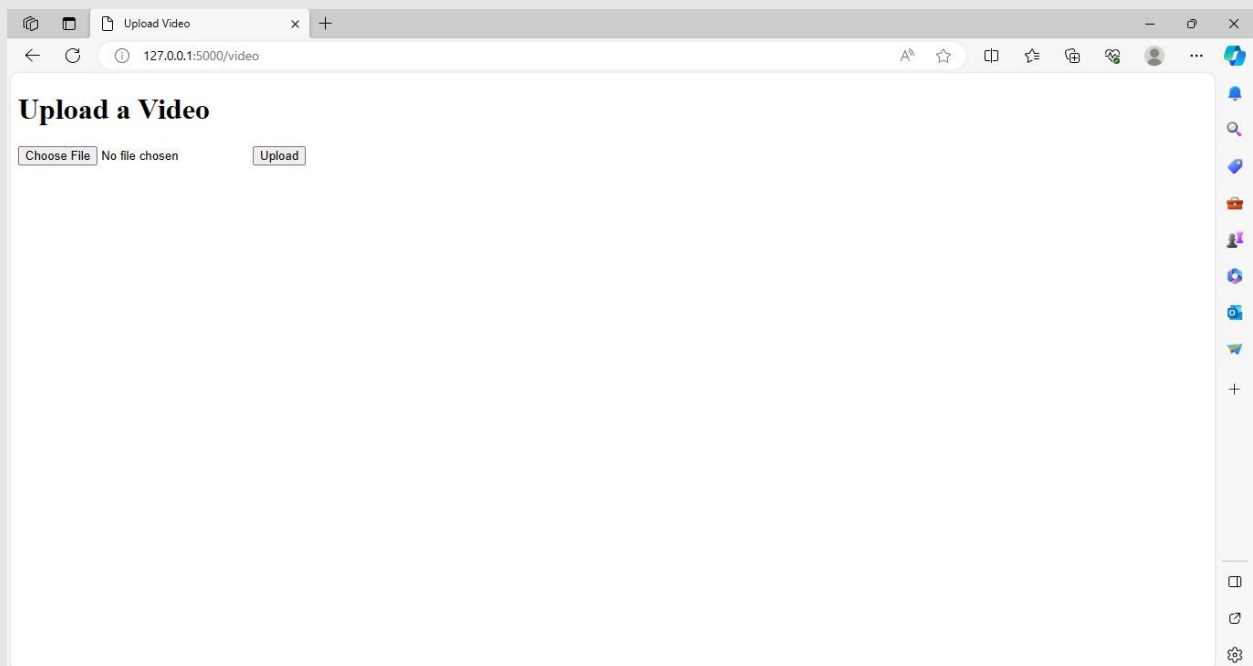


3. HOME PAGE INTERFACE FOR MEDIA STREAMING:

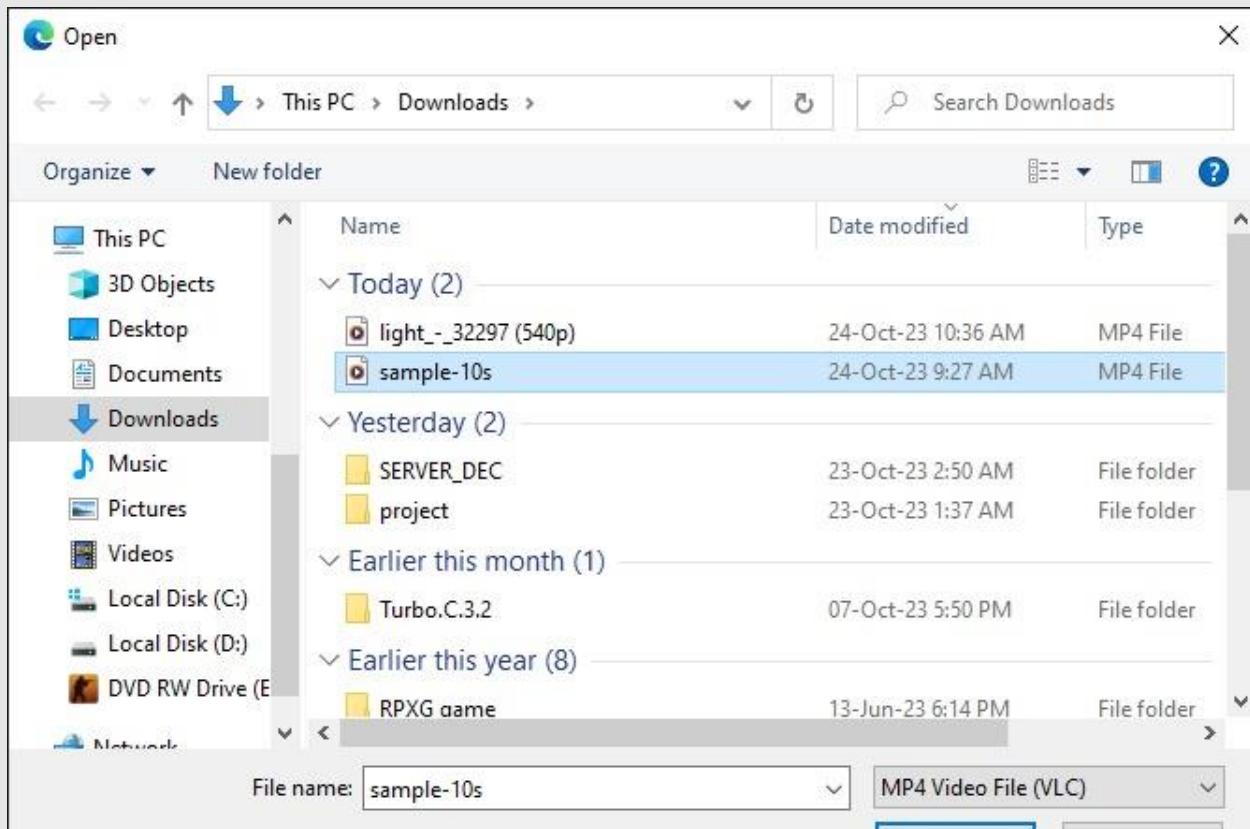


4. STEPS TO UPLOAD A FILE:

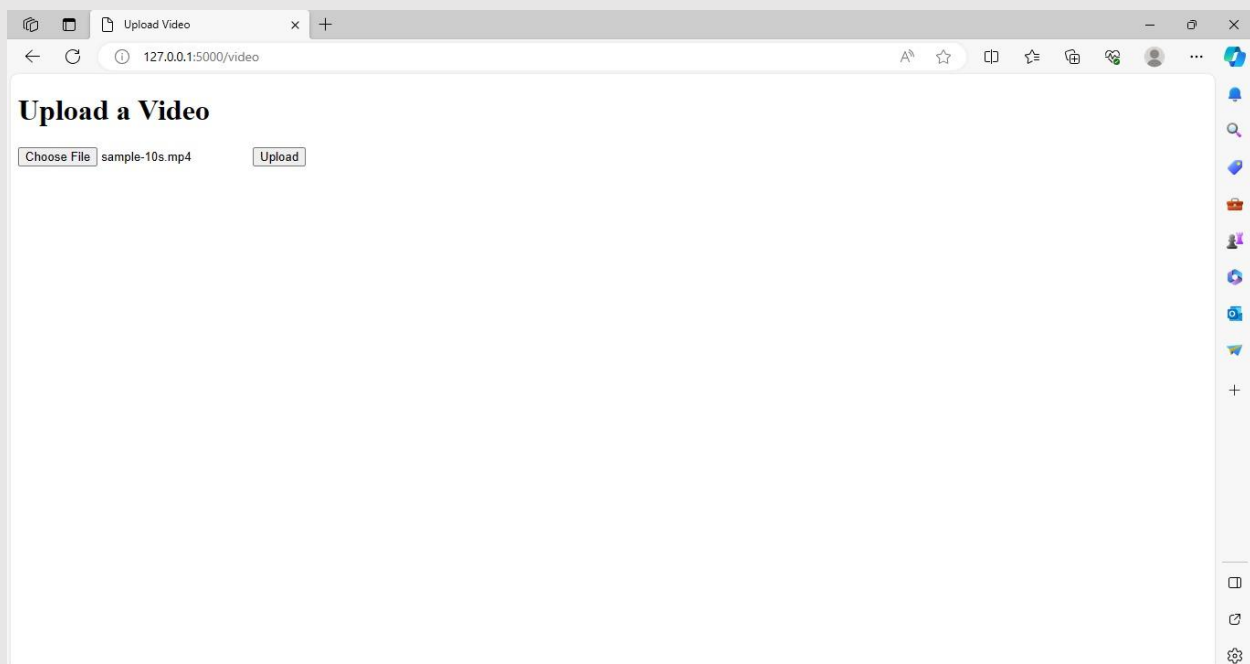
4.1) FIRST CLICK CHOOSE FILE CHECK BOX:



4.2) CHOOSE THE FILE AND CLICK OPEN TO CONTINUE:



4.3) CLICK THE UPLOAD CHECK BOX TO UPLOAD A VIDEO TO BE STREAMED:



5. STREAMING MEDIA INTERFACE:

