TECHNICAL REPORT

Youtube Stats Technical Report

Our group and website name is Youtube Stats. Our members include Hassan Sheikh, Jordy Tello, Brennan Wilkins, Tristan Hsieh, and Chris Pulicken.

Problems

Website content problems:

One of the problems we ran into while creating the website was finding a third entity type. We already knew that we wanted to include videos and channels on the website, but we were unsure about what other entity type would have enough attribute statistics to include on the website. We originally came up with four ideas: Youtube topics, categories, comment threads, or playlists. We eventually decided to use playlists as our third entity type, since topics and categories didn't have any channels directly associated with them, and comment threads did not have a lot of associated attribute statistics. Playlists worked well as a third entity type because it had a single channel as its creator, and multiple videos associated with it. Another problem we ran into while creating this website was finding playlists that had the appropriate relationship with channels and videos. For example, we wanted there to be some overlap between all entity types. This problem was solved by deriving playlists from the channels we used and deriving videos from those playlists. A third problem we ran into was storing all the information we had. For example, making fifty-three html pages for the videos entity type would be tedious and hard to edit. In order to solve this problem, we filled a template with information pulled from a database, instead of manually typing the information into the static html pages. We also ran into an issue concerning the ability to scroll through each webpage. During the first launch of the website, the user had the ability to scroll on the about page, but not the instance pages. This problem was resolved when we found out that there was nothing in the CSS allowing the user to scroll. After the CSS was fixed, the user was given access to the part of each instance page containing the link to other instances, including the javascript search feature. We also had difficulty designing the splash page. We ended up using a youtube wallpaper image as a background image. However, when we first implemented this, the wallpaper did not occupy the entire screen on everyone's browser, due to the difference in resolutions. This problem was resolved when we made the CSS instruct the splash page to size the wallpaper according to the screen's resolution. After this change, there was no more unwanted empty space on the splash page.

Team coordination problem:

One more problem we encountered was the problem of keeping formatting consistent between every member of the group. Since it was likely that multiple people would end up writing code for this project, we had to ensure that there would be no sudden switch in coding style. In order to fix this problem, we collectively agreed that we would use four spaces as opposed to tabs. This especially helped our code when using Python, since it is relatively strict about indentation.

Another issue that challenged our team was balancing our obligations to complete our assignments in the middle of spring break. This resulted in logistical challenges where the majority of our work had to be done remotely, and members had to juggle being with their families while making sure to accomplish the workload this first milestone had. We also had trouble determining what the unique identifiers for each row in SQL would be. We narrowed our possibilities down to two suggestions. The first was to make each row the Youtube ID of the instance, and the second was to label each instance numerically. We eventually decided to use the IDs for each instance in order to make the process of linking the user to each Youtube page more direct.

Use Cases

This website can be used to access the statistics of Youtube videos, channels, and playlists, providing an efficient way to collect Youtube data. The statistics will include the date of publication or the date last edited as well as user statistics such as number of likes or number of subscribers. Each page will also have a link to its corresponding Youtube webpage. All the video pages will link to its creator's channel as well as any playlist it was on. The channel pages will link to the playlists and videos that were created by the channel. The playlist pages will link to the channel of the playlist's creator and any videos on each of their respectable playlists. This website's accessibility will be aided by a splash page to help users navigate the website, an about page that contains every group member's bio as well as Git Lab statistics and a list of sources, and model pages. The model pages will include sortable tables of every instance for each entity type and will make use of multiple pages. These features could be of use for advertisers to collect data on the best channels or videos to display their ads on, in order to decrease their cost-per-click and increase clicks-per-ad. If further scaled, this could also be used by Youtube as a tool to increase their ad revenue by attracting more advertisers. Each entity type will be sorted into three different HTML pages, each containing their respective models. These models will be in grid format, where the user can scroll through a series of pages. Each instance will be sortable by name and user-based statistics like number of views or number of likes. The link for each instance will lead the user to a template html page, where the corresponding information has been passed through. For example, if the user wants to see the twentieth video, the user will be lead to an HTML page with the nineteenth index of the video json file passed through.

Design

<u>Data source:</u>

The data for this site will be pulled from a subsection of Youtube. Our site will have fifty-three videos, fifty-three channels, and fifty-three playlists total, and each instance will display its corresponding info on Youtube. There will be a total one hundred fifty-nine instances which make up the subsection of Youtube that we're pulling from.

<u>DB Models:</u>

The databases that store data of each instance will be a set of lists containing dictionaries. For example, the list for the videos' data will be fifty-three indices long. The dictionary within each index will contain the video's info. For example, the value for "Date published" at the fourth index will contain a string with the date published for the third video. For the links that lead to other instances, the value will be another list with each index being a linked instance. For example, the value for "Channel links" in the fourth index will have a list containing channel names. If a video doesn't have any associated instances, the value will just be the string "(none)". Every value will be a string, in order to make formatting much easier. There will be another database that stores every name of each instance and their corresponding html page's name. This database will be used to generate links for the associated instance names. For example, if a video has an associated channel in the form of a string, that channel will be accessible through the html page's name found in this database. The databases will allow us to generate pages for one hundred fifty-nine instances using three template html pages, one for each entity type. The pages will be called through python with the associated instance number, and they will draw from the json files that contain the information.

Search Capabilities:

We implemented a second search function that allowed the user to search the website using less strict rules. For example, our Javascript search feature only brings the user to a different webpage if the user types in the instance name character for character. Using the second search function, the user can type in a string, and the search feature will bring the user to a separate page with a list of every name that contains that string. This could be helpful when the user wants to find an instance with a name they can not entirely remember. The strings will also be case insensitive to increase search flexibility. In order to apply the search function, we will allow the user to type in a string and choose an entity type from a field and a drop-down menu. The field and the drop-down menu themselves will be implemented using HTML, and the string as well as the choice the user has selected will be received with JavaScript code. Then the user's string and the user's selection will be passed through main.py, which will then query the respective databases for the string. The resulting table will be pulled from the database, and the user will have the search results shown to them without having to reload the web page, all through the power of the mighty AJAX, (Asynchronous JavaScript and XML). The implementation details are the following:

After the user confirms their search parameters, a JavaScript function will make an asynchronous request to a python file via an XMLHTTP object. The object will pass along the information using a POST request to a python function. The function will take the query and fashion it inside a SQL statement using a case sensitive version of the 'like' keyword to get as many results that loosely meet the query as possible. The script will thus return a list of the objects using SQLAlchemy's Object Relational Mapping abilities. From there, we can pass the information relating to each of the objects back to the page the user is on, making sure to pass the ID, name, and link to the object as necessary for the results as requested in the requirements for milestone 5. We then intend on removing all elements of the page that aren't the nav bar in order to display search results in a clear way. We will accomplish this through our knowledge of the DOM (document object model) which allows us to manipulate the structure of html documents using JavaScript.

This will bring the total number of search features to 2, the one we first made, as well as the one mentioned above. The one discussed above will also be able to filter results by the entity type requested.

Tools

Front-end:
For the front-end portion, we created a basic template html file for videos, channels, and playlists. For example, each html file for a video contained a name, the date published, number of views, number of comments, number of likes, and number of dislikes. Each html page also included links to the other two entity types. Plain HTML was used to determine which order the text was in, which text would serve as a heading, and where the links would be located. We also implemented CSS to change the aesthetic of all the web pages. For example, CSS was used to change the color of the links, the background color, and the text color. We also used CSS to change the formatting and style of the text as well as the way the links respond to users clicking them. Python was also a way of gathering data. We pulled information from an SQL database to collect data to put into our html pages. This was to minimize the number of HTML pages necessary to build the website. For example, fifty-three video pages could all be generated using the same video template page. In order to access the database, we created classes for each entity type and populated the databases with fifty-three of these classes. This allowed us to retrieve data on Youtube videos, channels, and playlists in the form of an array. Then, we took this data, and displayed it on the html page. Most of the HTML, CSS, and Javascript for the web-pages were implemented using Bootstrap. In order to do this, we inserted a Bootstrap CSS URL into a link tag at the beginning of our HTML files. This allowed us to use design features of Bootstrap. We also downloaded the specific bootstrap files onto the server that would be hosting the site, instead of using a Content Distribution Network. For example, the text and navbar designs were called from Bootstrap CSS. We also implemented Javascript as a search feature on our website. On each webpage, we included a field and a button where the user can type the name of a desired instance and click the button. This provides the user with a shortcut to each instance, which they can use instead of following links or looking through other grids on entity type HTML pages. The Javascript takes the user's inputted name and searches for it in the database. It then takes the index where the name shows up and allows the user to access the webpage with that instance's information on it. If the user inputs a name that doesn't exist, or the user inputs nothing, then the user will receive an alert from the web page.

Back-end:
For the back-end portion, we referred to the splash.html file with Python code. This was done in a virtual environment, which set up a local host. Within the python file, we created a Flask object, and then under a function called "index," we used the Jinja Template Engine to render a template of a pre-existing html file called "splash.html". We then used Flask to publish the html

files to the local host, allowing them to be viewed in a browser. Flask will also determine the information that is displayed on each page. For example, if the user wants to access the thirtieth playlist, Flask will lead the user to a template playlist file and pass the twenty-ninth index of the playlist json file onto the web page.

Embedding-media services:
For embedding-media services, we used collected image and embedded video links from Youtube and stored them in JSON files. When the user accesses an instance, the page, in which they are lead to, will locate this JSON file through Flask and Python. Then, the embedded video link or the channel's profile url is returned and displayed on the webpage. The video and playlist pages contain the videos, and the channel playlist contains the channel's profile picture. Each video will be playable in browser, and the playlists will contain a list of videos the user can choose from, and play in a similar fashion to the embedded videos.

Hosting

In order to set up Google Cloud Project, we had to download the Google Cloud software development kit. After downloading the kit, we went to the Google Cloud Project console and created a project with our website's name. We then established a billing account in order to successfully complete the project's creation. After that, we inserted a yaml file into our repository to handle the configuration of the machine that would be used to host our site, obviously choosing to use a weaker hardware machine in order to save on credit since we believed our site would not be attracting much web traffic. After the yaml file was created and validated, the final step was to make sure we had a production-ready version of our site created. We fixed any final pesky errors. Then, via the Google Cloud free ephemeral VM provided, we decided to deploy our project with a simple command, 'gcloud app deploy'. After some time, the site was ready and accessible via the web.

Other

Namecheap:
Earlier on, we had established a way of reaching the live production version of our site by deploying it on Google Cloud Platform. However, this was still not an acceptable state since we had no control over what our domain was going to be, and typing out our domain wasn't very user-friendly. We were fortunate enough to be able to solve this problem by being able to select a domain from Namecheap for free.

We thus got to the business of choosing a specific domain for our site, when we settled on the idea of using the site name 'youtubestats.me', owing to the fact that our site was basically all about documenting youtube statistics on videos, channels, and users.

Getting the domain to point to our site involved a series of steps. First, we had to add the domain under Google's AppEngine domain settings. Afterwards, Google required us to create a DNS record to the domain of our site that proved we owned the site. Then we needed to add the text file to the DNS configuration of our domain.

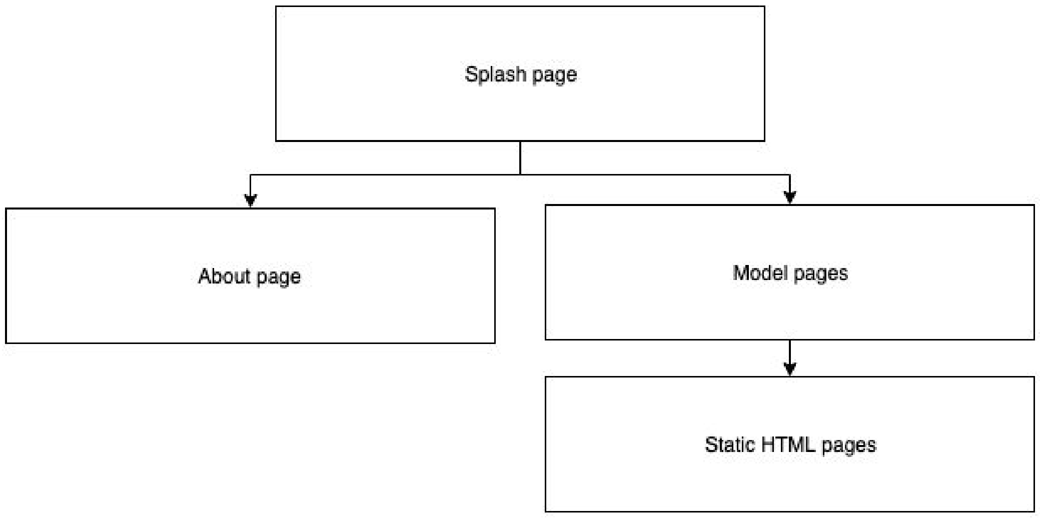
To complete the above, we then logged into the credentials we had for the user that owned the rights to 'youtubestats.me' on Namecheap. We transferred the text file over that proved our ownership of the site.

We afterwards switched back to GCP and verified we had completed the above steps. We then confirmed the domain we wanted our project to use ('youtubestats.me'). Google finally required us to add some resource records to our domain's registrar (Namecheap).

We were advised to wait up to 24 hours for the proper changes to take place in order for our domain to finally point to our project, but were pleasantly surprised when we checked for ourselves and it worked near-instantly. Our project was finally accessible from the domain we had chosen.

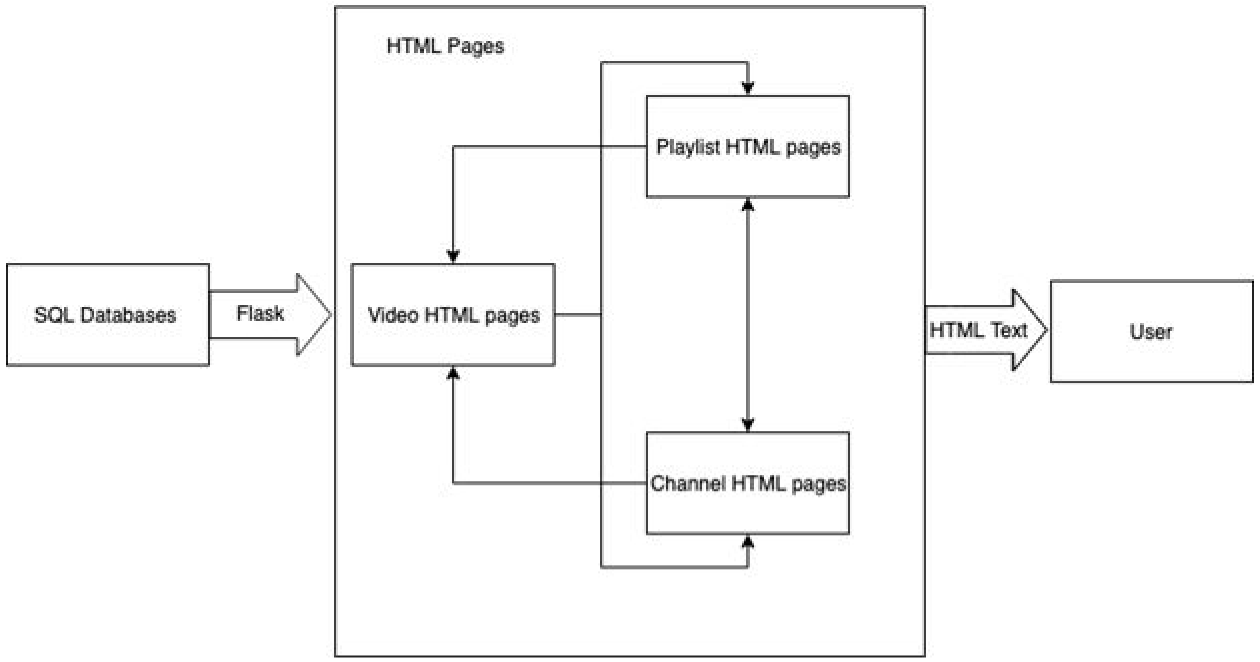
Diagrams

<u>Site layout diagram:</u>



This diagram displays how the site will be laid out. The user will be greeted by a splash page, which will allow them to navigate to the about page or the model pages. From the model pages, the user can access the static HTML pages. The about page will contain the user bios and photos, and the static HTML pages will contain Youtube statistics.

User case diagram:



This diagram shows how the site can be used. The user can navigate through the html pages for different entity types via links. Each html page will draw data from Youtube using an API, and the data will be converted into text on an HTML page for the user to read directly.