Languages used in this program are python, HTML\CSS with a little bit of JS.

This app is used to create a simple website where a user input an article and after using google's NLP technology, he receives a summarization of that article.

**summarize.py:**

server's side of the app can be demonstrated and explained here:

<https://www.youtube.com/watch?v=egDIqQIjDCI&t=325s&ab_channel=JamesBriggs>

(the difference is that we used BART model instead of the T5)

* summarize\_function:

The BART function receives a string, which we translate from a user file (dockx, pdf, html, txt). *open\_files* is a folder containing code to convert each one of those formats into a string:

\*clean\_html : uses BeautifulSoup to read an html file, clean it from unnecessary data and access the contents of the website

\*clean\_pdf : removes pictures, urls, etc...

\*clean\_text: performs an additional "cleaning" on a simple string, removing special characters, emails, phone numbers, turns text into lower-case...

\* there is also docx2txt which converts .docx files into text (included in summarize.py)

tokenizer = AutoTokenizer.from\_pretrained*(*"facebook/bart-lage-cnn/"*)*model = AutoModelForSeq2SeqLM.from\_pretrained*(*"facebook/bart-lage-cnn/"*)*

Those lines are responsible for loading the desired pretrained model from the "HuggingFace transformers" server. In my case I chose to download the model directly to the project's folder, those files are kinda heavy.

To obtain this functionality we need both the torch library(1GB+) and transformers.

Installing the torch library was my main obstacle in loading the app to the AWS server.

(In the pythonanywhere server I managed to install the torch library, however, I needed to manually upload the model's files to the server.

That piece of code above is firstly trying to load the model from a local file and if it can't find it, it will access HuggingFace website and download it from there.

In pythonAnyWhere, it kept failing and couldn't gain access to the website)

* summary\_length\_function:

there is an option for a user to determine how long he would like the summary to be.

this function simply calculates that number and it is later used as an argument in "model.generate" function (responsible for creating the summary)

* summarize\_from\_web:  we won't be using that in the actual site

This file is just an implementation of google's technology.

we created a function which we will later call from the "application.py" file.

**application.py**

I used the Flask microweb framework to launch this app and combine the frontend part.

this is one of the videos I watched as to how to load a flask app to the AWS server:

https://www.youtube.com/watch?v=iBeOvmt-tR0&ab\_channel=MathisVanEetvelde

Using Html\Css I created a basic front page for the app (index.html) which is located in the "templates" folder.

css and js files are located in the "static" folder.

from this page, using "file upload" functionality in the HTML page, we are receiving a file and verifying that this file is one of the four allowed types:

# verify type of file  
if file\_ext not in application.config*[*'UPLOAD\_EXTENSIONS'*]*:  
 return render\_template*(*"index.html", error="Only .docx, .txt, .pdf, .html allowed!"*)*

If the file is from one of those types we will save it to the local server (in my case to a folder on the pc):

application.config*[*'UPLOAD\_PATH'*]* = 'C:/Users/public'

path = os.path.join*(*application.config*[*'UPLOAD\_PATH'*]*, filename*)*f.save*(*path*)*

and then we will call the summarize function from summarize.py.

the page will then reload itself and present the "summarization" variable inside a text box.

Hopefully I didn't miss anything big.

Again, when loading the app to the AWS server, it failed when trying to install the torch library which was mentioned inside the requirements.txt file.