1. Introduction

The image colour generator is a website built with the intent for helping artists in the theming of a certain art that requires the image. The aim of this website is to extracted the main Colores of an image in order to obtain a palate of colours. This website involves a special back-end development that allows to run an algorithm capable of classify the main colours and output its colours and RBG code.

For the code we use HTML, CSS and Python as the main processing langue. To communicate between languages, design the site and the algorithm of the generation for the colours we utilise Modules such as Flask, Bootstrap and Werkzeuge tools for the server and NumPy, PIL and Sklean cluster for the algorithm.

2. Site the explanation

For the frontend we use Bootstrap, that is an open-source development framework for web development and provides easy pre-built designs for various elements allowing a quick assemble of good looking already screen responsive website. As a framework bootstrap includes a pre-defined grid system that contains Hypertext Markup Language (HTML), cascading style sheets (CSS) and JavaScript creating the elements without spending time worrying about basic commands and functions.[1] To create the web application, we utilise flask web framework for the backend that provides tools and libraries for easy python application. Flask is a micro framework that contains no dependencies to external libraries which makes it light and little dependency two external libraries, it is also good for easy update and good for security bugs. This makes it perfect for our easy web application since it requires just a home page that as to communicate with python and display the output.[2]Finally for collection of the image we use the Werkzeuge that is a comprehensive WSGI web application library [3] That allow us to ask and receive the image from a user in the website. WSGI is Python standard specification that describes how a web server communicates with web applications, and how web applications can be chained together to process one request.

For the algorithm we utilise NumPy that is a well-known library for Python that allows processing off big multi-dimensional matrixes and we use it to convert the pixels of the image into arrays of RGB. Then use sklearn that is an open-source library for machine learning and includes algorithms for classification, regression and grouping of data including the k-mean methods that we will implement in order to classify the mains colours of the image. Finally, use the PIL library that is also an open-source library for python that is capable of image manipulations and creation in the code we use it to create the palates of colours extracted.

[1] https://bootstrap-flask.readthedocs.io/en/stable/

[2] https://flask.palletsprojects.com/en/2.2.x/

[3] https://werkzeug.palletsprojects.com/en/2.2.x/

https://peps.python.org/pep-3333/

3. Methods

3.1. Site Scheme

To start developing this website we started by importing flask and setting up all the parts that enable the running of the webpage. Then we create the html and using the help of bootstrap create an easy template with a section with a title, description and a button that scrolls to another division bellow where the main code will be placed. (fig 1)

```
{% extends 'bootstrap/base.html' %}
{% block styles %}


{{ super() }}
<link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}">
{% endblock %}

{% block title %} Image Color Generator {% endblock %}

{% block content %}
<div class="jumbotron text-center">
    <div class="container">
        <h1 class="display-4">🎨 Image Color Generator 💻</h1>
        <p class="lead">Want to work on an art but don't know what colors to use?</p>
        <hr class="my-4">
        <p>You've found the right place! Checkout my color extractor that will give you all the main colors in an
            image.</p>
        <a class="btn btn-warning btn-lg" href="#section1" role="button">Show Me!</a>
    </div>
</div>

<div class="container" id="section1">
```

3.2. Image upload

To create the part for the user to upload the image we created a HTML form that makes a request for an input of a file and calls as action a new function in python that contains a conditional statemented to the request. Here the file is stored in a variable where then its name is extracted using Werkzeuge library and saved to the OS path configured in the app of the images folder this path is then stored in the flask session. This results in a path to display the image in the browser that is returned as well with the html template and the image filename that is passed into function in another python file named color_extraction.

```
def uploadFile():
    if request.method == 'POST':  # request
        # Upload file flask
        uploaded_img = request.files['uploaded-file']  # o q recebeu para folder
        # Extracting uploaded data file name
        img_filename = secure_filename(uploaded_img.filename)  # werkzeug.utils
        # Upload file to database (defined uploaded folder in static path)
        uploaded_img.save(os.path.join(app.config['UPLOAD_FOLDER'], img_filename))  # complicaded upload ?
        # Storing uploaded file path in flask session
        session['uploaded_img_file_path'] = os.path.join(app.config['UPLOAD_FOLDER'], img_filename)  # import
        # session ?
        img_file_path = session.get('uploaded_img_file_path', None)  # Retrieving uploaded file path from session
        # EXTRATION
        # Load an image up to a vector, from a given path
        path = f'static/images/{img_filename}'
        vec = path_to_img_array(path)

```

3.3. Image Colour extraction

To make this algorithm we start by importing the important modules that will allows classify the RGB pixels. Then we developed 3 functions for manipulating the date. The first called path_to_img_array opens the image and return a larger matrix of the arrays with the RBG colour for each pixel. The other function then reshapes these pixels by dividing them into them into 3 Green, Red and Blue in a long chain of arrays that are feed into the KMeans clustering method that essentially finds all the "Target Points" for all of the pixel values that are grouped around a classifies accordingly.

Finally, we call one last function for the algorithm to take these RGB Targets and plot out in some simple boxes to show the colour palette it found.

```python
def path_to_img_array(path):
    '''
    Load image into numpy array
    '''
    img = Image.open(path)
    vec = np.array(img)
    return vec


def pick_colors(vec, numColors):
    '''
    Do k-means clustering over ``vec`` to return ``numColors``
    '''
    vec = vec.reshape(-1, 3)
    model = KMeans(n_clusters=numColors).fit(vec)
    return model.cluster_centers_


def show_key_colors(colorList):
    '''
    Make a long rectangle, composed of the colors
    detailed in colorList, a list of (R, G, B) tuples
    '''
    n = len(colorList)

    im = Image.new('RGBA', (100 * n, 100))
    draw = ImageDraw.Draw(im)

    save_colors = []
    for idx, color in enumerate(colorList):
        color = tuple([int(x) for x in color])
        save_colors.append(color)
        draw.rectangle([(100 * idx, 0), (100 * (idx + 1), 100 * (idx + 1))], fill=tuple(color))

    im.save("static/images/savethis.png")
    return save_colors
```

To end the code, we added first to the form a required text input in order gives the user the option of choosing the number of colours want to extract from the image. Then called all these 3 functions by passing this colour value and save the returned RGB list created by the show_key_colors. We create know a path for the image of the colours palete and return all the items into the html. Were know all was left was to hide the items for images that appear if no image was output, however this is very easily done using flask conditional system.

```html
<div class="d-flex justify-content-center">

    {% if hide %}
    <h5></h5>
    {% else %}

    <div class="imagecenter" style="padding-top: 25px;">
        <img src="{{ user_image }}" class="rounded mx-auto d-block" alt="Image" width="500" height="400">
    </div>
    <img src="{{ user_image_palete }}" alt="colors">
    {% endif %}

    <div class="d-flex justify-content-center" style="padding: 25px;">
        <p><b> RGB Colors: </b></p>

        {% if hide %}
        <h5></h5>
        {% else %}

        <h5> {{ RGB }} </h5>
        {% endif %}


    </div>

</div>
</div>
```

Appendix A Python:

```python
import os
from flask import Flask, render_template, request, session
from flask_bootstrap import Bootstrap
from werkzeug.utils import secure_filename
from color_extration import path_to_img_array, pick_colors, show_key_colors


ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}


app = Flask(__name__, template_folder='templates', static_folder='static')
Bootstrap(app)


app.secret_key = 'fuckideletehis'


@app.route("/")
def home():


    hide = True
    return render_template("index.html", hide=hide)


@app.route('/', methods=("POST", "GET"))  # methods
def uploadFile():
    if request.method == 'POST':  # request
        # Upload file flask
        uploaded_img = request.files['uploaded-file']  # o q recebeu para folder
        # Extracting uploaded data file name
        img_filename = secure_filename(uploaded_img.filename)  # werkzeug.utils
        # Upload file to database (defined uploaded folder in static path)
        uploaded_img.save(os.path.join(app.config['UPLOAD_FOLDER'], img_filename))  #
complicaded upload ?
        # Storing uploaded file path in flask session
        session['uploaded_img_file_path'] = os.path.join(app.config['UPLOAD_FOLDER'],
img_filename)  # import
        # session ?
        img_file_path = session.get('uploaded_img_file_path', None)  # Retrieving uploaded
file path from session
        # EXTRATION
        # Load an image up to a vector, from a given path
        path = f'static/images/{img_filename}'
        vec = path_to_img_array(path)


        # Unrolls our image and runs K-Means clustering to find "Target Points" all of the
pixel values are grouped around
        color_value = request.form.get("nrcolors")
        print(color_value)


        colors = pick_colors(vec, int(color_value))
        rgb = str(show_key_colors(colors)).replace("[", "").replace("]", "")
        palete_filename = os.path.join(app.config['UPLOAD_FOLDER'], 'savethis.png')  #
Display in html
        print(rgb)


        return render_template('index.html', user_image=img_file_path,
user_image_palete=palete_filename, RGB=rgb)
```

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



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

Appendix B HTML:

```
{% extends 'bootstrap/base.html' %}
{% block styles %}

{{ super() }}
<link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}">
{% endblock %}

{% block title %} Image Color Generator {% endblock %}

{% block content %}
<div class="jumbotron text-center">
    <div class="container">
        <h1 class="display-4">🎨 Image Color Generator 💻</h1>
        <p class="lead">Want to work on an art but don't know what colors to use?</p>
        <hr class="my-4">
        <p>You've found the right place! Checkout my color extractor that will give you all
the main colors in an
            image.</p>
        <a class="btn btn-warning btn-lg" href="#section1" role="button">Show Me!</a>
    </div>
</div>

<div class="container" id="section1">
    <h1>Image Extractor</h1>
    <h3> Please upload your image</h3>
    <div class="row" id="boxes">
        <form method="POST" enctype="multipart/form-data" action="/">

            <div class="col-xs-6" id="leftbox">
                <input type="file" id="myFile" name="uploaded-file" accept=".jpg"
class="btn btn-primary" required>
            </div>

            <div class="col-xs-6">
                <p><b> Configurations: </b></p>
                <label>Number of colors : <input type="text" name="nrcolors" required />
</label>
            </div>

            <div class="col-xs-6" id="rightbox">
                <input type="submit" value="Submit" class="btn btn-warning">
            </div>

        </form>
    </div>

    <div class="d-flex justify-content-center">
```

```
        {% if hide %}
        <h5></h5>
        {% else %}

        <div class="imagecenter" style="padding-top: 25px;">
            <img src="{{ user_image }}" class="rounded mx-auto d-block" alt="Image"
width="500" height="400">
        </div>
        <img src="{{ user_image_palete }}" alt="colors">
        {% endif %}

        <div class="d-flex justify-content-center" style="padding: 25px;">
            <p><b> RGB Colors: </b></p>

            {% if hide %}
            <h5></h5>
            {% else %}

            <h5> {{ RGB }} </h5>
            {% endif %}



        </div>

    </div>
</div>
{% endblock %}
```