Årsoppgave Documentation

Contents

Uke 9

- Started researching the Spotify API
- Watching some tutorials to understand the API and how it works

- My plan is to create a website that integrates the Spotify API, a login system, and databases using HTML, CSS, and JavaScript for the front end, and Python for the backend as the API. I will connect the databases and use Flask for the login systems.

Spotify API Code Example

- This week, my main goal is to test the different functions available in the Spotify API.
- Added a **get_token** function that connects to Spotify to obtain an access token, acting as a temporary pass to access Spotify's data.
- Added a **/get-artist-image** route that retrieves and displays an artist's image from Spotify based on the user's search.
- Added a **search_artist** function that displays the artist's name from Spotify when a user searches for an artist.

```python
# Function to fetch the token from Spotify
def get_token():
    auth_string = client_id + ":" + client_secret
    auth_bytes = auth_string.encode("utf-8")
    auth_base64 = str(base64.b64encode(auth_bytes), "utf-8")

    url = "https://accounts.spotify.com/api/token"
    headers = {
        "Authorization": "Basic " + auth_base64,
        "Content-Type": "application/x-www-form-urlencoded"
    }
    data = {"grant_type": "client_credentials"}
    result = post(url, headers=headers, data=data)
    json_result = json.loads(result.content)
    token = json_result["access_token"]
    return token

# Test the get_token function
# token = get_token()
```

```python

def search_artist(token, artist_name):
    url = "https://api.spotify.com/v1/search"
    headers = get_auth_header(token)
    query = f"?q={artist_name}&type=artist&limit=1"

    query_url = url + query
    result = get(query_url, headers=headers)
    json_result = json.loads(result.content)["artists"]["items"]

    if len(json_result) == 0:
        print("No artist found")
        return None

    return json_result[0]
```

```python
@app.route('/get-artist-image', methods=['GET'])
def get_artist_image():
    artist_name = request.args.get('artist', 'Dystinct')
    if not artist_name:
        return jsonify({'error': 'No artist name provided'}), 400

    # Fetch the Spotify access token
    access_token = get_token()
    url = f"https://api.spotify.com/v1/search?q={artist_name}&type=artist&limit=1"
    headers = get_auth_header(access_token)

    # Make the request to the Spotify API
    response = requests.get(url, headers=headers)
    data = response.json()

    # Check if artist data exists
    if data.get('artists', {}).get('items'):
        artist = data['artists']['items'][0]
        # Get the artist image URL
        artist_image = artist['images'][0]['url'] if artist['images'] else None

        # Pass the image URL and artist name to the template
        return render_template('get-image.html', image_url=artist_image, artist_name=artist_name)

    return jsonify({'error': 'Artist not found'}), 404
```

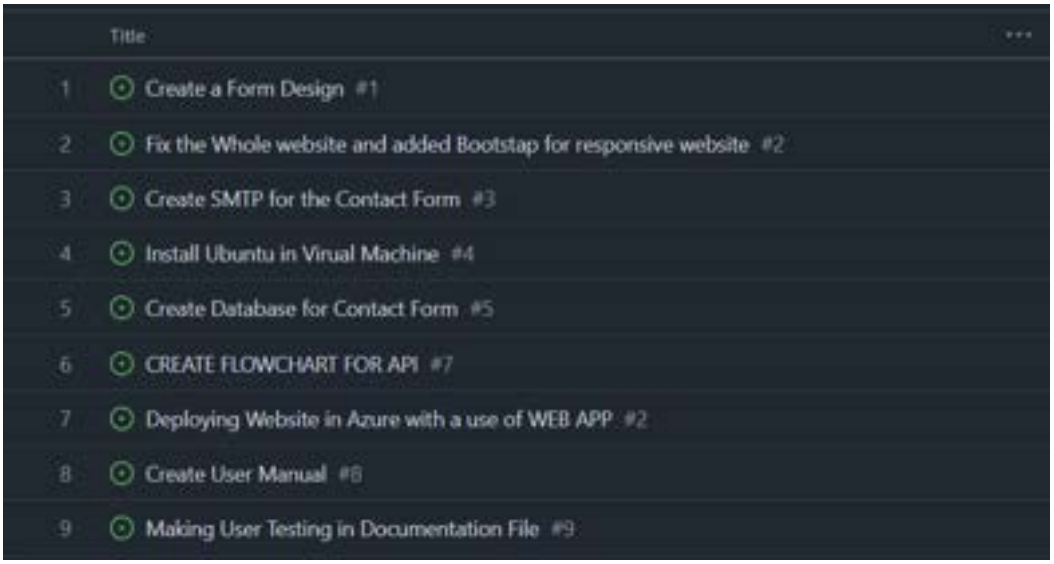
```swipe.js
const cards = document.querySelectorAll('.music-card');
let currentCardIndex = 0;
let isSwiping = false;

const card = cards[currentCardIndex];

// Start dragging
card.addEventListener('mousedown', (e) => {
    isSwiping = true;
    card.classList.add('moving');
});

// On mouse move, update the card's position
document.addEventListener('mousemove', (e) => {
    if (!isSwiping) return;

    const diff = e.clientX - card.getBoundingClientRect().left;
    card.style.transform = `translateX(${diff}px)`;  // Moves the card

    if (diff > 100) {
        card.classList.add('right'); // Swipe Right
    } else if (diff < -100) {
        card.classList.add('left'); // Swipe Left
    }
});
```

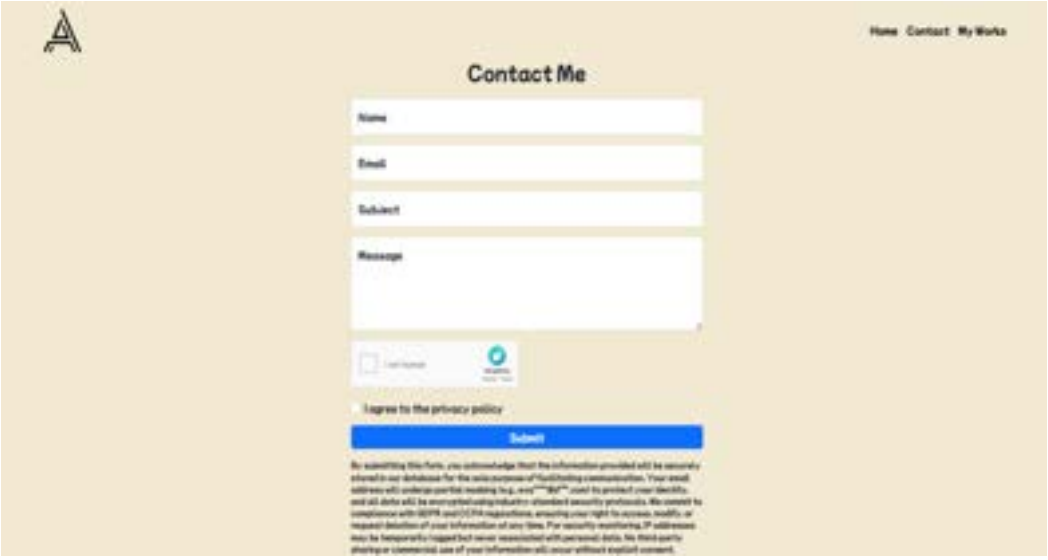
Conclusion

- After working with the Spotify API, I realized that there were parts of the code I didn't fully understand, especially on JavaScript. My initial plan was to create a Tinder-style music app using Spotify, where users could like or dislike songs to generate a playlist. However, I discovered that Spotify doesn't allow full song playback, which is a key feature for my project. Because of this limitation, I decided to shift my focus on working my portfolio instead.

Uke 13

- Began planning using GitHub Projects.
- My objective is to combine two APIs, **Abstract** and **hCaptcha**, to create a contact form. Abstract will validate whether the user's email address is valid, and hCaptcha will prevent spam. Additionally, once the user is validated, the form will send an email to both the user and myself using **SMTP**.
- I started developing the front end of the website using Bootstrap for responsiveness.





SMTP Code Example

- Added an **SMTP** function that automatically sends an email via Gmail to both me and the user when the contact form is filled out and submitted.
- The **os.gatenv** function is linked to **.env** where all the important information is stored.
- My goal for next week is to create a submit function that generates a receipt for both the user who sends the email and for myself. This information will then be saved to the database using **MariaDB.**

```python
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import os
from dotenv import load_dotenv

load_dotenv()
def send_email(from_email, subject, body):
    if not all([from_email, subject, body]):
        raise ValueError("Email parameters cannot be None or empty")

    smtp_server = os.getenv('SMTPSERVER')
    smtp_port = int(os.getenv('SMTPPORT'))
    smtp_username = os.getenv('SMTPUSERNAME')
    smtp_password = os.getenv('SMTPPASSWORD')
    smtp_email = os.getenv('SMTPEMAIL')

    message = MIMEMultipart()
    message["From"] = smtp_email
    message["To"] = from_email
    message["Subject"] = subject
    message["Reply-To"] = from_email

    body_part = MIMEText(body.strip(), "plain", "utf-8")
    message.attach(body_part)

    try:
        with smtplib.SMTP(smtp_server, smtp_port) as server:
            server.starttls()
            server.login(smtp_username, smtp_password)
            server.send_message(message)
            return True
    except Exception as e:
        print(f"Failed to send email: {str(e)}")
        raise
```

Conclusion

- After switching projects, I feel that the code is easier for me to understand, and it aligns better with my goal of focusing more on Python rather than JavaScript. Creating the GitHub plan also made me feel more in control of the project. I received advice from my friends and teachers, which was very helpful.

Uke 14

- This week, my focus was to create the submit function in my Python backend. However, I realized it would be more effective to first focus on integrating the

two APIs **hCaptcha** and abstract for email validation. The contact form should first go through hCaptcha and validation when the user clicks the submit button, so it makes sense to implement these checks before developing the submit function.

- Started linking the two APIs in my **.env** to secure the keys and manage to create a function to verify if both API works.

Abstract & Hcaptcha Code Example

Abstract Verification Code

```python
def verify_email_address(email):
    api_key = os.getenv("ABSTRACTAPIKEY")
    url = f"https://emailvalidation.abstractapi.com/v1/?api_key={api_key}&email={email}"

    allowed_free_domains = ["gmail.com", "yahoo.com", "outlook.com", "hotmail.com"]

    try:
        response = requests.get(url)
        data = response.json()


        print(f"AbstractAPI response for {email}: {data}")


        quality_score = float(data.get("quality_score", 0))
        is_valid_format = data.get("is_valid_format", {}).get("value", False)
        deliverability = data.get("deliverability", "UNDELIVERABLE")
        domain = email.split('@')[-1] if '@' in email else None


        print(f"Quality Score: {quality_score}, Valid Format: {is_valid_format}, "
              f"Deliverability: {deliverability}, Domain: {domain}")

        # Primary validation checks
        if not is_valid_format:
            print("Failed: Invalid email format")
            return False

        if quality_score < 0.60:  # Acceptable quality score
            print("Failed: Low quality score")
            return False

        if deliverability != "DELIVERABLE":
            print("Failed: Not deliverable")
            return False

        # Check for allowed free domains
        if domain not in allowed_free_domains:
            print("Passed: Allowed free domain")
            return False

        # Accept any email that passed the above validations
        print("Passed: Valid email")
        return True

    except Exception as e:
        print(f"Email validation error: {str(e)}")
        return False
```

Hcaptcha Verification Code

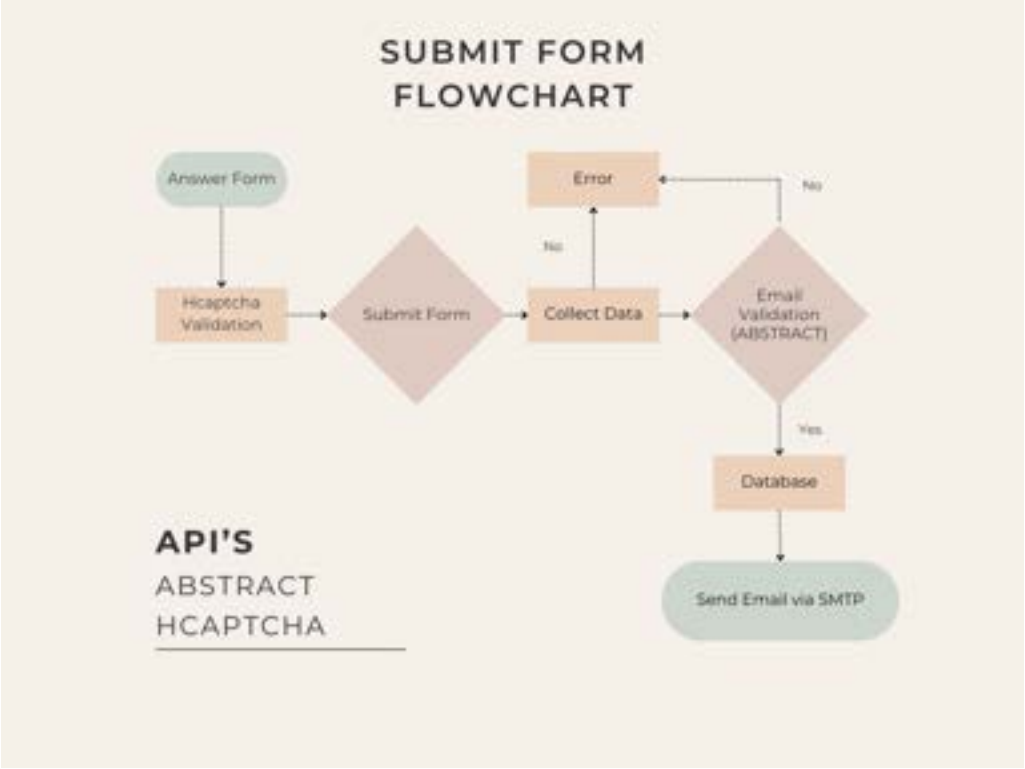
```python
# Verify CAPTCHA response
def verify_captcha(token):
    url = 'https://hcaptcha.com/siteverify'
    data = {'secret': HCAPTCHA_SECRET_KEY, 'response': token}
    response = requests.post(url, data=data).json()
    # print(f"CAPTCHA verification response: {response}")
    return response
```

Conclusion

- This week was challenging because it was my first time integrating an API into my project. Fortunately, I found some helpful videos that guided me through establishing the connection.
- My plan for next week is to create the submit function. After the user completes verification using hCaptcha and Abstract, the function will check if the email domain is valid, such as Gmail, Hotmail, Yahoo, or Outlook. Additionally, Abstract will assess whether the email is a dummy based on its score and format before approving it.

Uke 15

- My plan for this week is to finally create the submit function
- Create Database via MariaDB and connect it to my python script
- Create a Flowchart on how my Whole website works



Submit Form Function Code Example

Submit Function with the database connection

Added Tokenize Email

- This Code will make the database table look something like “User123” instead of the user real email. The reason for that is to make sure the user security and privacy.

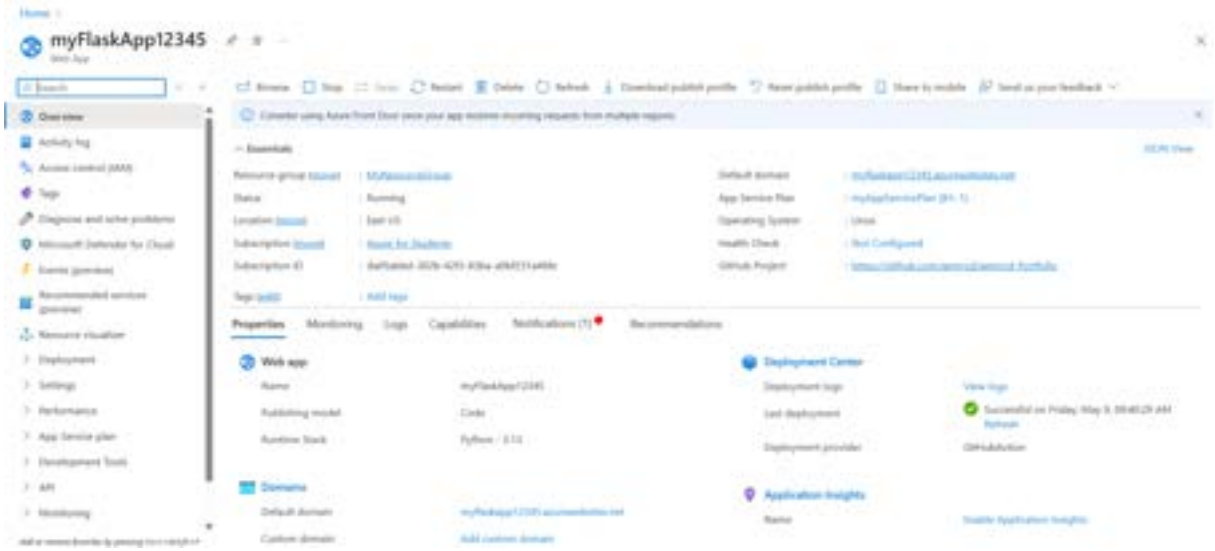
```python
def tokenize_email(email):

    user_id = secrets.randbelow(1000000)


    domain = email.split('@')[-1]


    email_token = f"User_{user_id}"

    return email_token, domain
```

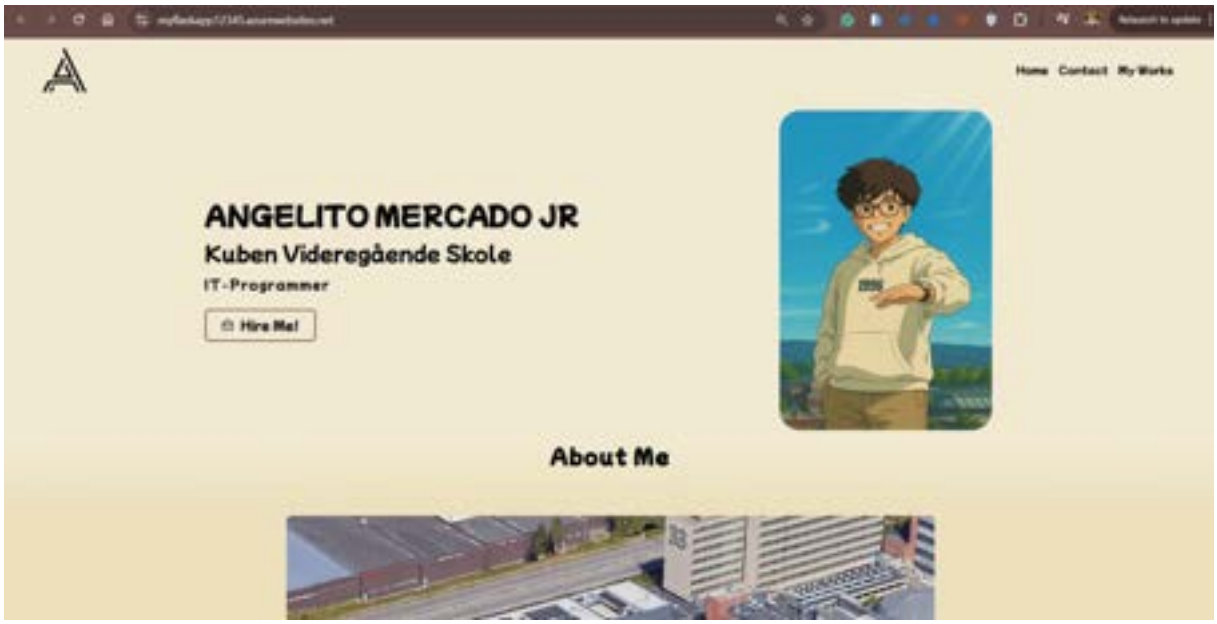
Conclusion

- This week was more challenging because I had to fix not only the code as well as if the database worked or not. In Addition to that I added a Tokenize function for the my table database so I can
- Next week i will try to deploy my website by using Azure and a Web App as a Resource Group.

Uke 19

- This week my plan is to deploy my Project into Azure by connecting my GitHub repository to it.





Conclusion

- The process was difficult but luckily I got help from Alf, the only problem that I got is managing my .env because I forgot to put all the secret information in Environment Variable inside of my web app.

Uke 20

- Plan for this week is to double all my works from the start to make sure I got all of my Projects ready for the incoming discussion.
- Fix the readme for User-Manual

README

My Portfolio Website

This is my personal portfolio site with a contact form. The form includes spam protection via hCaptcha and email validation via Abstract API.

Table of Contents

Clone and Setup Project

To use the contact form functionality, you'll need to set up Virtual Environment and the API keys in your `.env` file.

Follow these simple steps to clone the repository to your local machine:

1. **Create a folder** where you want to store the cloned files:
   - You can create a folder anywhere on your computer (example, in your "Documents" or on your desktop).
2. **Open the Command Prompt**:
   - Press `Windows Key + R` to open the "Run" menu.
   - Type `cmd` and press **Enter**. This will open the Command Prompt.
3. **Clone the Repository**
   - In the Command Prompt, use the `git clone` command to clone the repository into the folder you created:

```
git clone https://github.com/aemrcd/aemrcd.github.io
```

   - Installing libraries to python by typing the following commands. This will download all the libraries to use the website.

```
pip install -r requirements.txt
```

User-Testing

User Experience Feedback - Jasan

1. What do you think about the contact form?
    - The contact form is good designed, functions work fine but, the authentication “I am human” is too difficult.
2. Was the website responsive on both desktop and mobile?
    - Yes
3. What do you like most about the website?
    - Contact information and the color palette.
4. What did you dislike or find frustrating?
    - Azure is slow on deployment.

User Experience Feedback - Frendon

1. What do you think about the contact form?
    - I really like it, it has a simple and minimalistic design. Only it has the necessary points. Very straightforward.
2. Was the website responsive on both desktop and mobile?
    - Everything seems to be good and responsive but the picture slide in home screen, might need a little bit space in mobile version.
3. What do you like most about the website?
    - I like the simple design, the colors compliment each other.
4. What do you dislike or find frustrating?
    - Only the home picture slide I mentioned earlier.

User Experience Feedback - Knut

1. What do you think about the contact form?
   - Good, responsive but confusing captcha
2. Was the website responsive on both desktop and mobile?
   - Good , works on phone and desktop.
3. What do you like most about the website?
   - I like the project slide.
4. What do you dislike or find frustrating?
   - No audio on video and burger menu looks trippy.

Reference

- https://www.hcaptcha.com/
- https://www.abstractapi.com/
- http://chatgpt.com/
- https://portal.azure.com/