Урок 18. Міні-проєкт

Портал погоди

[Презентація]

Мета уроку:

- Познайомити дітей із процесом взаємодії між фронтендом та бекендом, а також використанням API для отримання погоди.

На цьому уроці ми:

- Пригадаємо як працювати з JSON та REST
- Дізнаємось як розділити додаток на мікросервіс
- На практиці закріпимо попередні знання

Розробка має складатись з таких етапів:

Back-end:

- Створення застосунку на Flask
- Додавання можливості відправляти запити та ручне тестування
- Створення функції, яка створює правильний get-запит з JSON, який в подальшому буде приходити з фронтенда
- Створення функції, яка перетворює відповідь від API на JSON з потрібними даними
- Реалізація реєстрації
- Створення мікросервісу для роботи з емейлом

Front-end:

- Створення сторінок для реєстрації, авторизації та головної сторінки
- Додавання відправки запиту до бекенду коли користувач обирає параметри запиту
- Додавання обробки та відображення даних, які приходять у відповідь на головній сторінці
- Підключення стилів (використання Bootstrap)
- Загальне тестування застосунку, виявлення помилок та їх вирішення

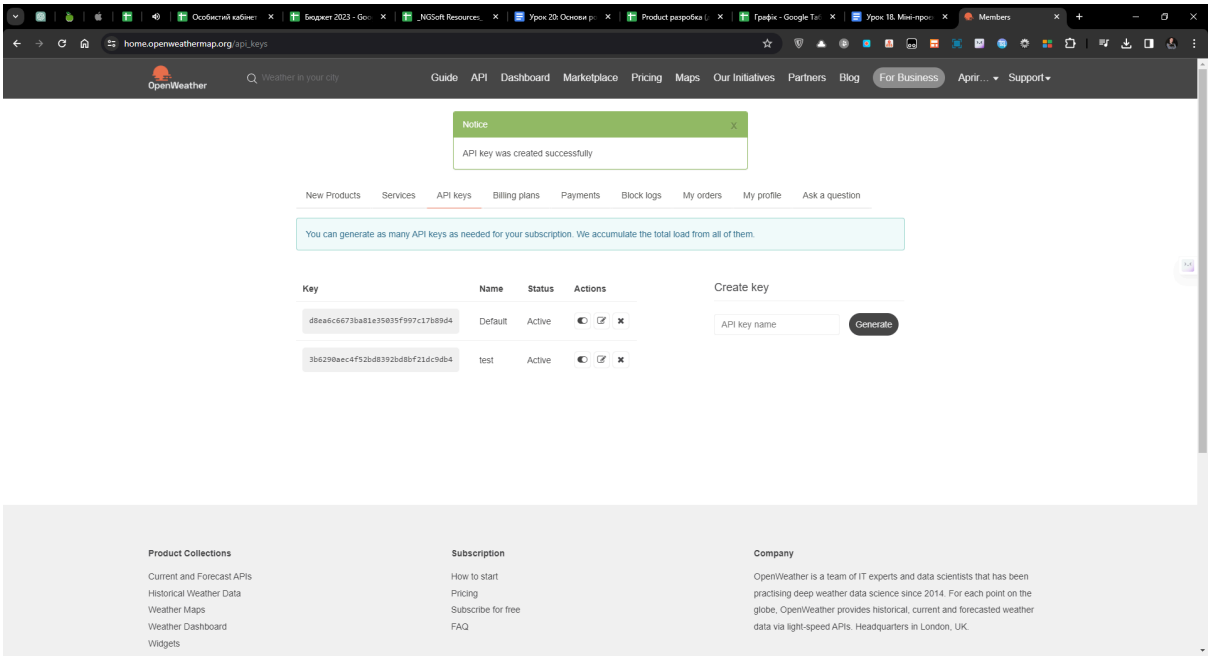
Щоб апка працювала треба токен для openweathermap та smtp сервер на укрнет пошті (для відправки емейлів) https://github.com/IvanLapchenko/weather-app

Послідовність дій:

pip install flask
pip install flask-cors
pip install requests
pip install flask-mail
pip install schedule

У нас буде використовуватись мікросерверна архітектура, коли бекенд і сервер підписки працюють на різних портах і в різних теках незалежно один від одного

Отримуємо wheather api key
Можете показати дітям як його створити на сторінці
https://home.openweathermap.org/api_keys
або дати наш: `d8ea6c6673ba81e35035f997c17b89d4`



потім створюємо новий файл secret.py, куди кладемо наш ключ

```
openweather_api_key = "d8ea6c6673ba81e35035f997c17b89d4"
```

створюємо новий файл app.py, куди імпортуємо все, що нам буде потрібно

```
from flask import Flask, request, jsonify
import requests
from secret import openweather_api_key
from flask_cors import CORS
```

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

```
app.run(debug=True, port=5000)
```

Напишемо перший роут, який потім буде використовувати фронтенд

```
@app.route('/get_weather')
def get_weather():
    location = request.args.get('location')
    if location:
        url =
f'http://api.openweathermap.org/data/2.5/weather?q={location}&appi
d={openweather_api_key}&units=metric'
        response = requests.get(url)
        data = response.json()
        return jsonify(data)
    else:
        return jsonify(error='Location not provided'), 400
```

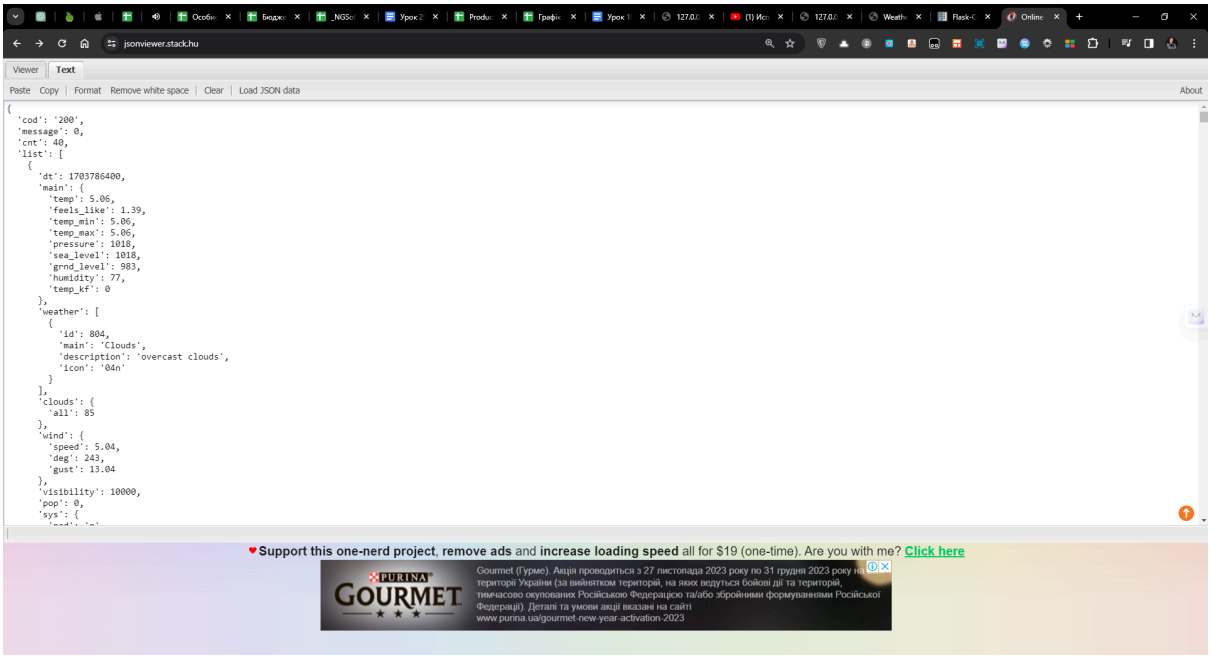
Перервірити ми його зараз можемо так:
http://127.0.0.1:5000/get_weather?location=lviv

Теперь зробимо роут для погоди на декілька днів

```
@app.route('/get_weather_forecast')
def get_weather_forecast():
    location = request.args.get('location')
    if location:
        url =
f'http://api.openweathermap.org/data/2.5/forecast?q={location}&app
id={openweather_api_key}&units=metric'
        response = requests.get(url)
        data = response.json()

        # get weather info for nearest few days
        forecast = []
        print(data)
        for entry in data['list']:
            extracted_data = extract_weather_data(entry)
            forecast.append(extracted_data)
        return jsonify(forecast)
    else:
        return jsonify(error='Location not provided'), 400
```

як бачимо нам треба написати сервіс який буде парсити нашу погоду,
`extract_weather_data`
тому що погода нам приходить у вигляді листу із декількох днів, нам потрібно його розпарсити,
щоб подивитись як він виглядає треба скористатися сайтом
https://jsonviewer.stack.hu/
і вставити туди те,що нам виводить print



Цей парсер ми зробимо у файлі service.py

```
def extract_weather_data(weather_data):
    main = weather_data['main']

    extracted_data = {
        'temp': main['temp'],
        'feels_like': main['feels_like'],
        'temp_min': main['temp_min'],
        'temp_max': main['temp_max'],
        'weather_main': weather_data['weather'][0]['main'],
        'weather_description':
weather_data['weather'][0]['description'],
        'wind_speed': weather_data['wind']['speed'],
        'clouds': weather_data['clouds']['all'],
        'dt_txt': weather_data['dt_txt']
    }

    return extracted_data
```

у app.py

```
from service import extract_weather_data
```

Тепер переходимо до фронтенду, створюємо теку FRONTEND і у ній index.html та script.js

index.html

```
<!DOCTYPE html>
```

```
<html>
<head>
   <title>Weather App</title>
   <meta charset="UTF-8">
   <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.
min.css">
</head>
<body>
   <div class="container mt-5">
       <h1 class="text-center">Weather App</h1>

       <div class="row">

           <!-- display form for city to get weather for today -->
           <div class="col">
               <h2 class="my-4">Today's Weather</h2>
               <form id="weather-form" class="my-4">
                   <div class="form-group">
                       <label for="location">Enter Location:</label>
                       <input type="text" class="form-control"
id="location" required>
                   </div>
                   <button type="submit" class="btn btn-primary">Get
Today's Weather</button>
               </form>
           </div>

           <!-- display form for city to get weather for a few days
-->
           <div class="col">
               <h2 class="my-4">10-Day Forecast</h2>
               <form id="forecast-form" class="my-4">
                   <div class="form-group">
                       <label for="forecast-location">Enter
Location:</label>
                       <input type="text" class="form-control"
id="forecast-location" required>
                   </div>
                   <button type="submit" class="btn btn-primary">Get
16-Day Forecast</button>
               </form>
           </div>

       </div>

       <div id="current-weather" class="my-4"></div>
       <div id="forecast"></div>
```

```
    </div>


    <script
src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
    <script src="script.js"></script>
</body>
</html>
```

script.js

Як поступати з JavaScript кодом вирішуйте самі, якщо часу багато - пишіть разом з дітьми і коментуйте, якщо мало - просто скиньте їм код і прокоментуйте його коротенько
https://github.com/IvanLapchenko/weather-app/blob/main/frontend/script.js

script.js

```
document.addEventListener('DOMContentLoaded', function() {
    const weatherForm = document.getElementById('weather-form');
    const forecastForm = document.getElementById('forecast-form');
    const currentWeatherDiv =
document.getElementById('current-weather');
    const forecastDiv = document.getElementById('forecast');
    const host = 'http://127.0.0.1:5000'

    weatherForm.addEventListener('submit', function(event) {
        event.preventDefault();
        const location = document.getElementById('location').value;

        fetch(`${host}/get_weather?location=${location}`)
            .then(response => response.json())
            .then(data => {
            const weatherInfo = `
                <div class="card">
                    <div class="card-header">
                        <h3>${data.name}, ${data.sys.country}</h3>
                    </div>
                    <div class="card-body">
                        <p class="card-text">Temperature:
${data.main.temp}°C</p>
                        <p class="card-text">Feels Like:
${data.main.feels_like}°C</p>
                        <p class="card-text">Humidity:
${data.main.humidity}%</p>
                        <p class="card-text">Visibility:
${data.visibility} meters</p>
```

```
                        <p class="card-text">Wind Speed:
${data.wind.speed} m/s</p>
                        <p class="card-text">Wind Direction:
${data.wind.deg}°</p>
                        <p class="card-text">Cloudiness:
${data.clouds.all}%</p>
                        <p class="card-text">Weather:
${data.weather[0].description}</p>
                    </div>
                </div>
            `;
                currentWeatherDiv.innerHTML = weatherInfo;
            })
            .catch(error => console.log(error));
    });

    forecastForm.addEventListener('submit', function(event) {
        event.preventDefault();
        const forecastLocation =
document.getElementById('forecast-location').value;


fetch(`${host}/get_weather_forecast?location=${forecastLocation}`)
                    .then(response => response.json())
                    .then(data => {
                        const groupedForecast =
groupForecastByDate(data);
                        displayForecast(groupedForecast);
                forecastDiv.innerHTML = forecastInfo;
            })
            .catch(error => console.log(error));
    });



function groupForecastByDate(forecastData) {
    const grouped = {};
    forecastData.forEach(entry => {
        const date = entry.dt_txt.split(' ')[0];
        if (!grouped[date]) {
            grouped[date] = [];
        }
        grouped[date].push(entry);
    });
    return grouped;
}

function displayForecast(groupedForecast) {
```

```
    let forecastInfo = '';
    for (const date in groupedForecast) {
        forecastInfo += `
            <div class="card mb-3">
                <div class="card-header">
                    <h5>${date}</h5>
                </div>
                <div class="card-body">
                    <div class="row">
        `;
        groupedForecast[date].forEach(entry => {
            forecastInfo += `
                <div class="col-md-3 border p-3 m-0">
                    <p>${entry.dt_txt.split(' ')[1]}</p>
                    <p>${entry.temp}°C</p>
                    <p>${entry.weather_description}</p>
                    <p>Wind: ${entry.wind_speed} m/s</p>
                </div>
            `;
        });
        forecastInfo += `
                    </div>
                </div>
            </div>
        `;
    }
    forecastDiv.innerHTML = forecastInfo;
}

});
```

Пояснення по коду :

DOMContentLoaded Event Listener:

```
document.addEventListener('DOMContentLoaded', function() {
```

Отримання елементів форми та інших DOM-елементів:

```
const weatherForm = document.getElementById('weather-form');
const forecastForm = document.getElementById('forecast-form');
const currentWeatherDiv = document.getElementById('current-weather');
const forecastDiv = document.getElementById('forecast');
const host = 'http://127.0.0.1:5000';
```

Тут визначаються різні елементи DOM, такі як форми (`weatherForm` та `forecastForm`), елементи відображення погоди (`currentWeatherDiv` та `forecastDiv`), і `host` - адреса сервера, до якого будуть відправлятися запити.

EventListener для форми отримання погоди:

```
weatherForm.addEventListener('submit', function(event) {
```

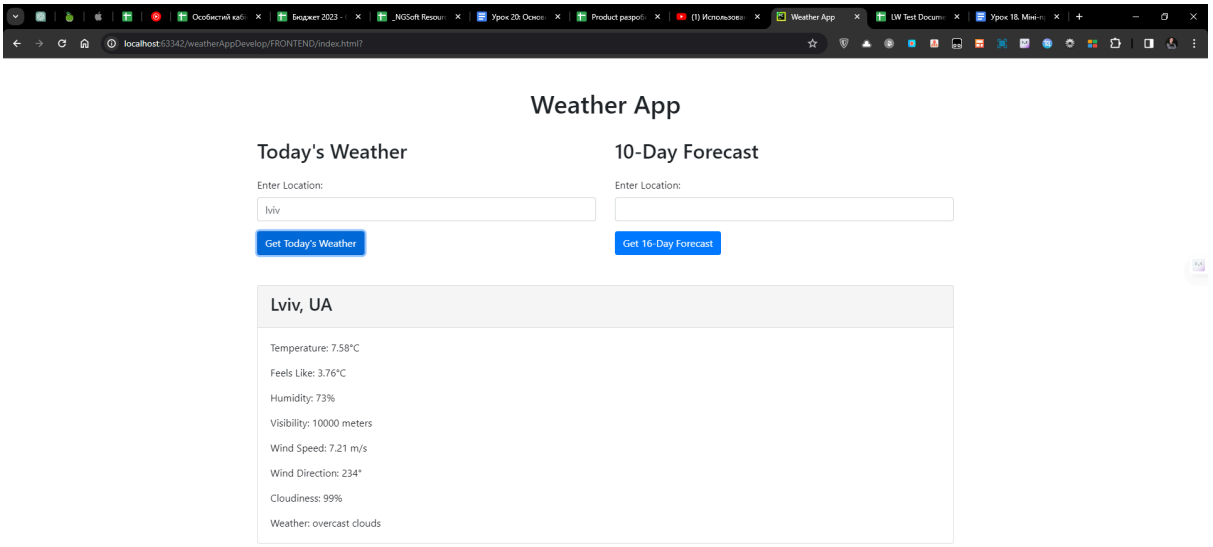
Цей блок коду слухає подію подачі форми для отримання погоди.

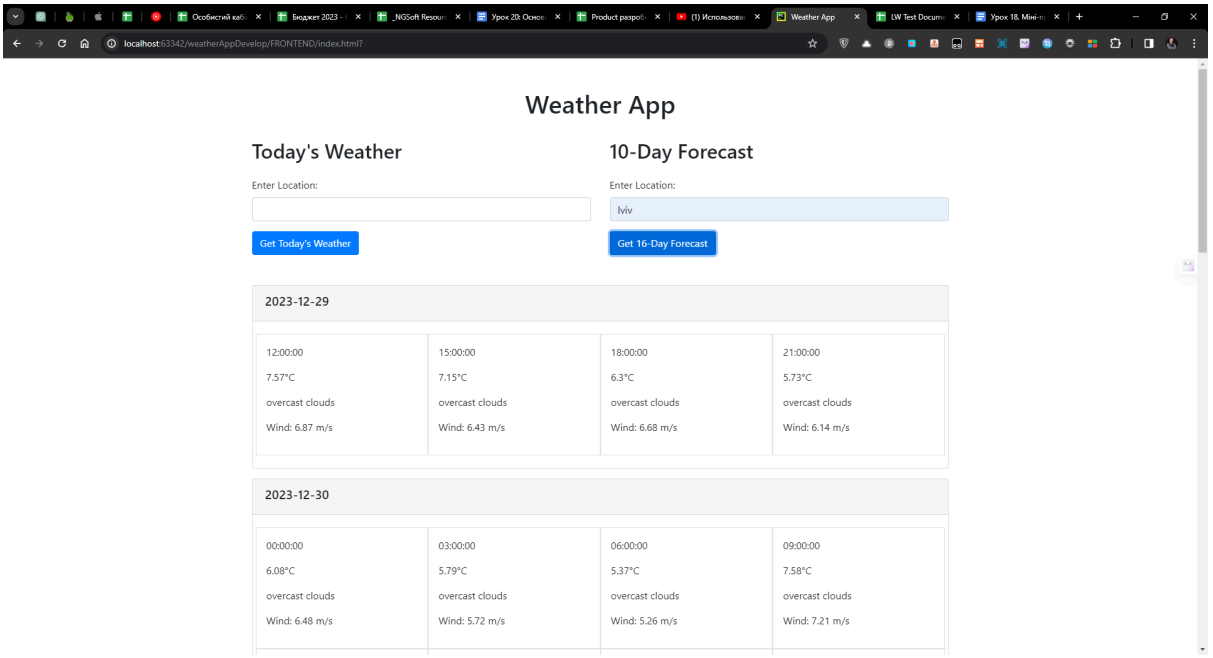
```
event.preventDefault();
```
Запобігає перезавантаженню сторінки після відправлення форми.
```
const location = document.getElementById('location').value;
```
Отримує значення, введене користувачем в поле вводу місця.
```
fetch(`${host}/get_weather?location=${location}`)
    .then(response => response.json())
    .then(data => {
        // ... виводить інформацію про погоду в блок currentWeatherDiv
    })
    .catch(error => console.log(error));
```
Виконує асинхронний запит до сервера для отримання прогнозу погоди за введеним місцем. Викликає функції для групування та відображення прогнозу.

Ці дві функції (`groupForecastByDate` і `displayForecast`) працюють разом для обробки та відображення прогнозу погоди, отриманого від сервера.
`groupForecastByDate` функція:
Ця функція приймає масив з прогнозом погоди (`forecastData`) і групує його за датою. Кожен запис містить `dt_txt`, в якому зазначається дата і час прогнозу. Функція використовує цю інформацію, щоб розділити прогноз за датами. Вона створює об'єкт `grouped`, де ключ - це дата, а значення - масив записів прогнозу для цієї дати.

`displayForecast` функція:
Ця функція приймає згрупований прогноз `groupedForecast` і створює HTML-код для відображення прогнозу погоди. Вона використовує цикл `for...in`, щоб перебрати кожен запис у `groupedForecast`. Для кожної дати вона створює HTML- код з вкладеними картками (`<div class="card">`) для відображення окремих днів прогнозу. Кожна картка містить інформацію про час, температуру, опис погоди та швидкість вітру для кожного запису прогнозу в цей день.

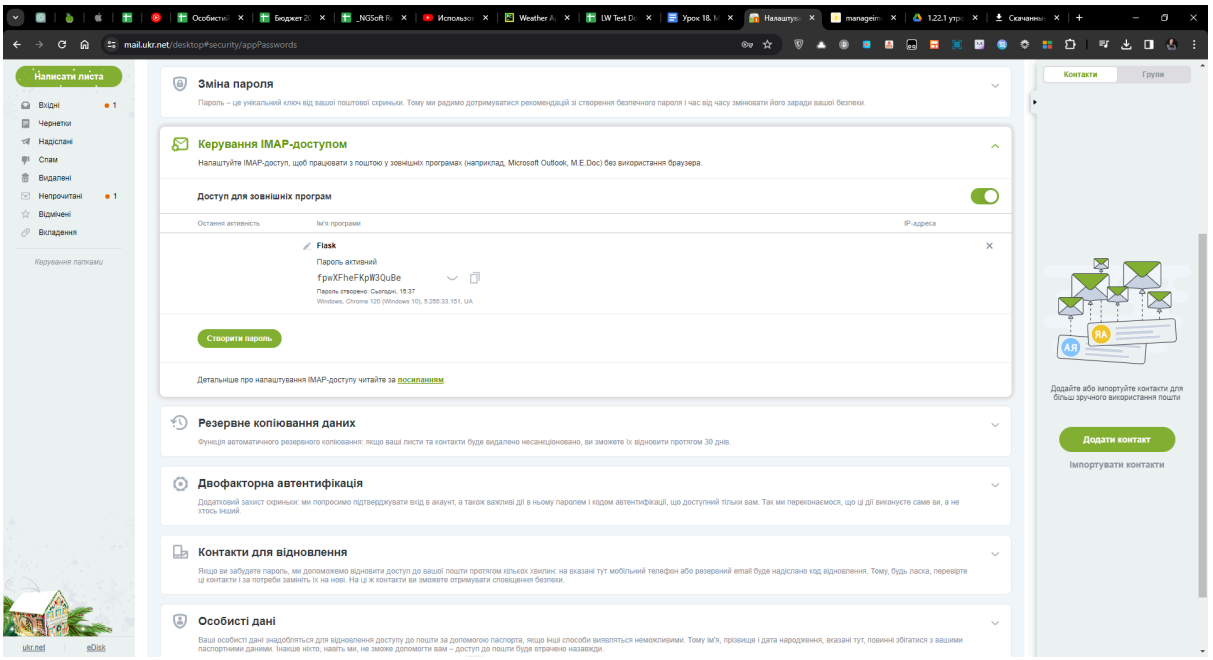
Перевіримо як усе працює





Тепер починаємо писати сервер підписки з підключенням до бази даних, це ми робимо у новому файлі mail.py
Але для того, щоб наш smtp сервер працював потрібно спочатку зареєструватися на https://www.ukr.net/
Краще створіть свою, але якщо ліньки - то ось моя:
sounddummies@ukr.net
Genius87
після цього, необхідно відкрити доступ на сторінці:
https://mail.ukr.net/desktop#security/appPasswords

Пароль:
fpwXFheFKpW3QuBe



А тепер сам код
Почнемо з бд - database.py

```
import sqlite3

conn = sqlite3.connect('subscriptions.db',
check_same_thread=False)
cursor = conn.cursor()


def create_subscription(email):
    cursor.execute(f'INSERT INTO subscriptions (email)
VALUES("{email}")')
    conn.commit()


def get_all_emails():
    cursor.execute('SELECT email FROM subscriptions')
    emails = [row[0] for row in cursor.fetchall()]
    return emails


def create_database():
```

```
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS subscriptions (
            id INTEGER PRIMARY KEY,
            email TEXT NOT NULL
        )
    ''')

    conn.commit()
```

check_same_thread=False - це аргумент, який можна використовувати при створенні об'єкта бази даних SQLite в Python, коли ви працюєте з багатопотоковим додатком. Він вказує SQLite не перевіряти, чи знаходяться всі SQL-операції в одному й тому ж потоці.

У багатьох випадках SQLite не призначений для багатопотокового використання і може викидати винятки, якщо ви спробуєте використовувати його в багатьох потоках одночасно. Але, якщо ви впевнені, що ваші SQL-операції виконуються в одному потоці (наприклад, якщо ви використовуєте SQLite в однопотоковому веб -застосунку), встановлення check_same_thread=False може уникнути винятків.

Один раз викличемо метод для створення бд
`create_database()`
і після запуску файлу і створення бд видалимо його виклик

у файл secret додаємо email, pasword від укрнет
```
openweather_api_key = "d8ea6c6673ba81e35035f997c17b89d4"
```

тепер код mail.py
```
from flask import Flask, jsonify, request
from secret import email, password
from flask_mail import Mail, Message
from database import create_subscription, get_all_emails
import schedule
import time
import requests
import json


app = Flask(__name__)
```

```
app.config['MAIL_SERVER'] = 'smtp.ukr.net'
app.config['MAIL_PORT'] = 465
app.config['MAIL_USERNAME'] = email
app.config['MAIL_PASSWORD'] = password
app.config['MAIL_DEFAULT_SENDER'] = email
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True

mail = Mail(app)
app.app_context().push()


def create_email_with_data_from_api():
    response =
requests.get('http://localhost:5000/get_weather?location=Sumy'
)
    raw_content = response.content

    json_string = raw_content.decode('utf-8')
    weather_data = json.loads(json_string)

    email_text = f"Weather Update for
{weather_data['name']}:\n"
    email_text += f"Temperature:
{weather_data['main']['temp']}°C\n"
    email_text += f"Feels Like:
{weather_data['main']['feels_like']}°C\n"
    email_text += f"Pressure:
{weather_data['main']['pressure']} hPa\n"
    email_text += f"Wind Speed: {weather_data['wind']['speed']}
m/s\n"
    email_text += f"Cloudiness:
{weather_data['clouds']['all']}%\n"
    email_text +=
f"{weather_data['weather'][0]['description']}\n"
    email_text += f"Visibility: {weather_data['visibility']}
meters\n"
    return email_text


def send_email():
    msg = Message('Event reminder',
recipients=get_all_emails())
    msg.body = 'Your daily weather'
```

```
    msg.html = create_email_with_data_from_api()
    mail.send(msg)
    print('Email sent')



@app.route('/create_subscription', methods=['POST'])
def subscription():
    email = request.form.get('email')
    if email:
        create_subscription(email)
        return jsonify(message='Subscription added
successfully')
    else:
        return jsonify(error='Email not provided'), 400


schedule.every().day.at("10:00").do(send_email)
app.run(port=3000)

while True:
    schedule.run_pending()
    time.sleep(60*60)
```

Додамо код у наш фронтенд index.html

```
<h4 class="text-center">Subscribe to Weather Updates</h4>
<form id="subscription-form"
action="http://localhost:3000/create_subscription"
method="post">
    <div class="form-group text-center" style="max-width: 30%;
margin: auto;">
        <label for="email" class="" >Email:</label>
        <input type="email" class="form-control"  id="email"
name="email" required>
        <button type="submit" class="btn btn-primary mt-2"
>Subscribe</button>
    </div>
</form>
```

після того, як натискаємо” підписатися”, викликається роут
`create_subscription`
в цьому методі зберігаємо емейл у базі данних
і потім відправляємого його кожного дня у 10:00

```
schedule.every().day.at("10:00").do(send_email)
```

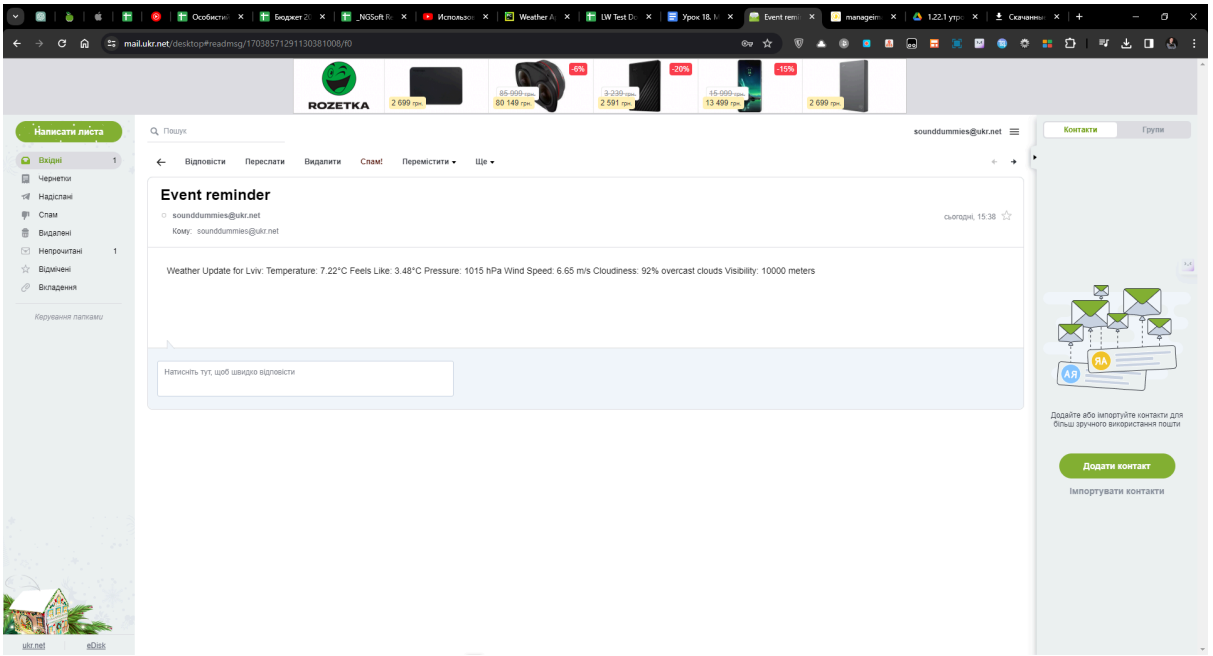
while True:
Безкінечний цикл для постійної перевірки запланованих подій та забезпечення роботи веб-сервера.

schedule.run_pending()
Перевіряє та виконує заплановані події, які повинні відбутися на даний момент. Якщо, наприклад, поточний час дорівнює 10:00, то викликається функція send_email.

time.sleep(60*60)
Затримка в одну годину перед наступною ітерацією циклу. Це робиться для того, щоб цикл не витрачав занадто багато ресурсів CPU, оскільки події плануються щодня о 10:00, і зазвичай немає потреби перевіряти їх кожну секунду.

Щоб перевірити і не чекати до завтра, викличемо метод send_email()

якщо все ок, то на пошту прийде лист



Це все

Як варіант самостійної роботи - запропонуйте учням самостійно створити кул портал на основі цього api
Використовується https://spoonacular.com/food-api.