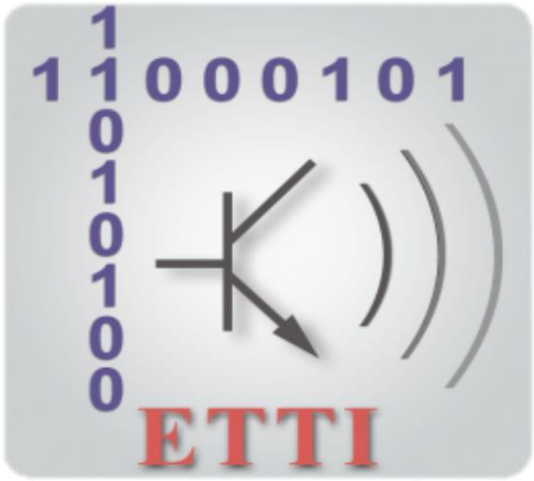
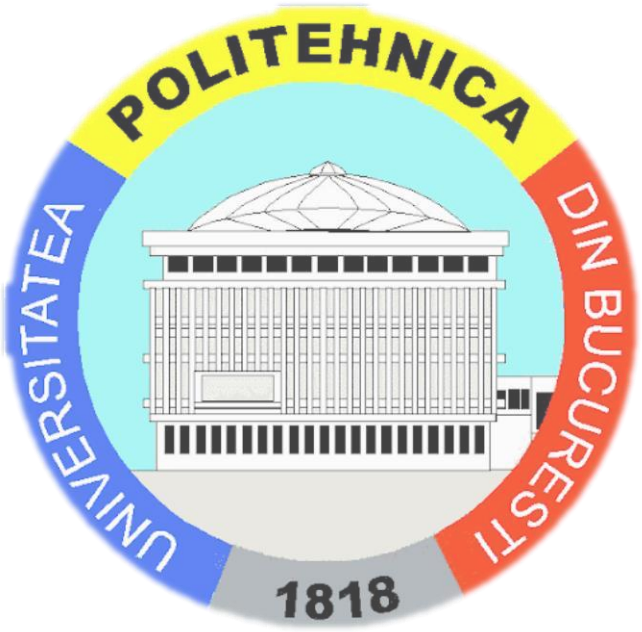
UNIVERSITATEA POLITEHNICĂ BUCUREȘTI

Facultatea de Electronică, Telecomunicații si Tehnologia Informației





Proiect 3

Aplicatie web pentru informatii si prognoza meteo

STUDENT: **ROȘU Sebastian**

GRUPA: **441Aa**

COORDONATOR: **Bogdan Cristian Florea**

București, 2022

CUPRINS:

1. TEHNOLOGII FOLOSITE

1.1 PYTHON[1]

Python este un limbaj de programare dinamic multi-paradigmă, creat în 1989 de programatorul olandez Guido van Rossum. Van Rossum este și în ziua de astăzi un lider al comunității de dezvoltatori de software care lucrează la perfecționarea limbajul **Python** și implementarea de bază a acestuia, CPython, scrisă în C. Python este un *limbaj multifuncțional* folosit de exemplu de către companii ca Google sau Yahoo! pentru programarea aplicațiilor web, însă există și o serie de aplicații științifice sau de divertisment programate parțial sau în întregime în Python.

Python pune accentul pe curățenia și simplitatea codului, iar sintaxa sa le permite dezvoltatorilor să exprime unele idei programatice într-o manieră mai clară și mai concisă decât în alte limbaje de programare ca C.

1.2 Visual Studio Code[2]

Visual Studio Code (VS Code) este un editor de cod sursă ușor, dar puternic, care rulează pe desktop și este disponibil pentru Windows, macOS și Linux. Acesta vine cu asistență încorporată pentru JavaScript, TypeScript și Node.js și are un ecosistem bogat de extensii pentru alte limbi (cum ar fi C+, C#, Java, Python, PHP și Go) și runtimes (cum ar fi .NET și Unity).
VS Code vă permite să vă extindeți capacitatea prin extensii. Extensiile de VS Code pot adăuga mai multe caracteristici experienței generale. Odată cu lansarea acestei caracteristici, acum puteți utiliza extensia VS Code pentru a lucra cu portalurile Power Apps.

1.3 Flask[3]

Flask este un micro framework web scris în Python. Este clasificat ca microframe, deoarece nu necesită anumite instrumente sau biblioteci. Nu are un strat de abstractizare a bazei de date, validarea formularelor sau alte componente în care bibliotecile terțe preexistente oferă funcții comune.

1.4 HTML[4]

HyperText Markup Language (HTML) este un limbaj de marcare utilizat pentru crearea paginilor web ce pot fi afișate într-un browser (sau navigator).

Scopul HTML este mai degrabă prezentarea informațiilor – paragrafe, fonturi, tabele ș.a.m.d. – decât descrierea semanticii documentului. În cadrul dezvoltării web de tip front-end, HTML este utilizat împreună cu CSS și JavaScript.

1.5 Bootstrap 5[5]

Bootstrap este un framework CSS gratuit și opened-source, direcționat către dezvoltarea web front-end receptivă și mobilă. Conține șabloane de design bazate pe CSS și JavaScript pentru tipografie, formulare, butoane, navigare și alte componente de interfață.

2. LIBRARII UTILIZATE

Pentru dezvoltarea aplicatiei web s-au folosit urmatoarele librarii:

- *Requests*
- *Flask*
- *Datetime*
- *Math*

3. EXTENSII PENTRU FLASK UTILIZATE

- *flask-login* – folosit pentru managementul userilor
- *flask-sqlalchemy* – folosit pentru maparea modelelor la baza de date
- *flask-bcrypt* – support pentru hashuirea parolelor
- *flask-wtf* – folosit pentru crearea form-urilor
- *wtforms* – librarie ce ofera validare pentru fieldurile form-urilor

.

4. DESCRIEREA APLICATIEI

Aplicatia web permite utilizatorului sa obtina informatii despre prognoza meteo conform locatiei dorite, apeland API-ul Open Weather Map.

Userul poate opta pentru obtinerea datelor meteo curente, dar si pentru o prognoza pe intreaga saptamana atata timp cat detine un cont si este logat. Informatiile meteo constau in temperature curenta, temperatura resimtita, descrierea starii curente a vremii, viteza vantului, umiditate si presiune atmosferica.

5. STRUCTURA APLICATIEI SI FUNCTIONALITATI

- __init__.py – In cadrul acestui script sunt initializate cele mai importante obiecte care vor ajuta la construirea siteului
- models.py – Aici este definita clasa User care va contine fieldurile necesare pentru a stoca informatiile utilizatorilor si pe baza a carui instanta se va crea tabelul corespunzator in baza de date folosind SQLAlchemy

- forms.py – Crearea formurilor de Login, Register, User Settings si Get Weather folosind FlaskForm si validatorii oferiti de wtforms

- get_data.py – In interiorul acestui script sunt definite cele doua functii pentru apelarea API-ului si anume:

```
def get_data(city, country, unit):
```

pentru apelarea endpointului ce ofera informatii meteo curente si

```
def get_data_7days(city, country, unit):
```

pentru apelarea endpointului ce ofera informatii pentru prognoza de 7 zile. Raspunsul este stocat in variabila response prin efectuarea unui request in care este oferit URL-ul, headerul ce contie API key-ul si querystring-ul ce contine parametrii pe baza carora se doreste sa se obtina informatii, in cazul de fata orasul, tara si sistemul de unitati de masura.

Din response sunt extrase doar informatiile necesare care vor fi stocate intr-un dictionar, urmand ca acesta sa fie returnat.

```python
def get_data(city, country, unit):
    url = "https://community-open-weather-map.p.rapidapi.com/weather"

    querystring = {"q":f"{city},{country}","units":f"{unit}"}


    headers = {
        'x-rapidapi-host': "community-open-weather-map.p.rapidapi.com",
        'x-rapidapi-key': "b9ac356800mshf0da663a66d341bp12b8e4jsn145828c1f17e"
        }

    response = requests.request("GET", url, headers=headers, params=querystring).json()


    weather_data = {'description': response['weather'][0]['description'],
                    'icon': response['weather'][0]['icon'],
                    'temp': math.floor(response['main']['temp']),
                    'temp_feels_like':math.floor(response['main']['feels_like']),
                    'temp_min':math.floor(response['main']['temp_min']),
                    'temp_max':math.floor(response['main']['temp_max']),
                    'pressure': response['main']['pressure'],
                    'humidity': response['main']['humidity'],
                    'wind_speed': response['wind']['speed'],
                    'country': response['sys']['country'],
                    'city': response['name'],
                    'code': response['cod'],
                    'date': date.fromtimestamp(response['dt']).strftime("%m/%d/%Y")
                    }



    return weather_data
```

- routes.py – Aici sunt definite routele aplicatiei web dupa cum urmeaza:

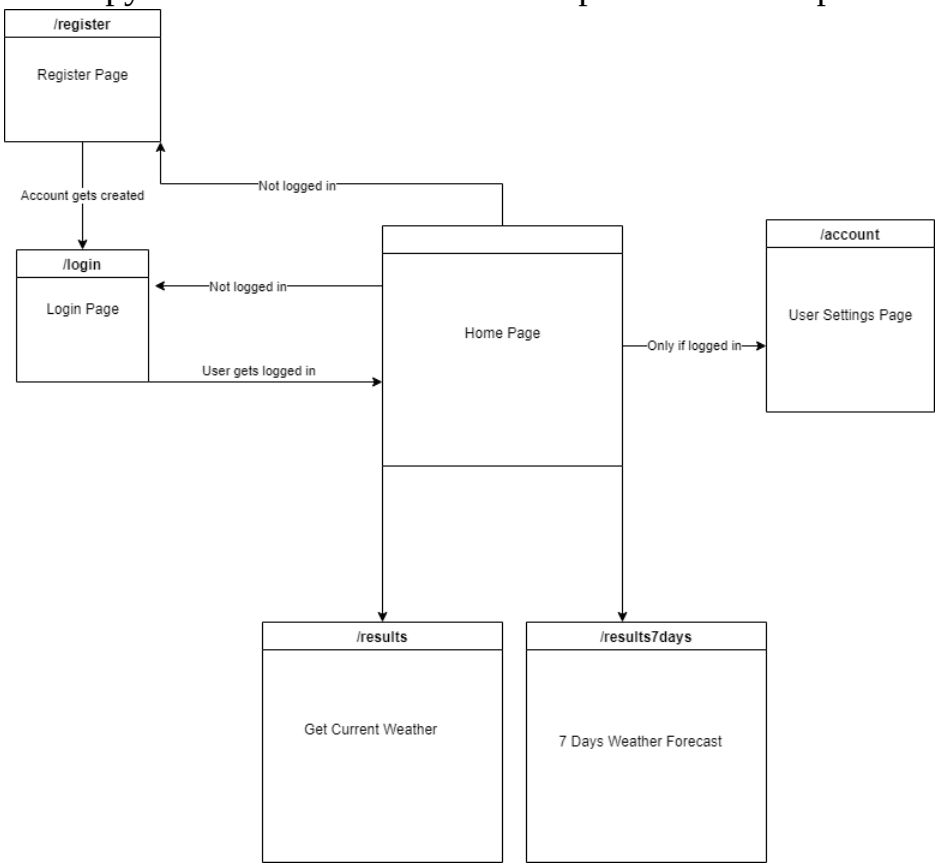


Figure 1[6]

➢ @app.route(‘/’) – in aceasta ruta este definit form-ul pentru preluarea datelor si anume orasul si tara. Daca user este logat, acesta va putea solicita informatii atat pentru vremea curenta cat si pentru prognoza pe o saptamana fiind redirectionat catre /results respectiv results7days.
Un user nelogat are access doar la vremea curenta.

```
@app.route('/', methods=['GET', 'POST'])
def home():

    form = GetWeatherForm()

    if current_user.is_authenticated:
        unit = current_user.measurement_unit
    else:
        unit = 'metric'

    if form.validate_on_submit():
        if form.get_current_weather.data:
            return redirect(url_for('results',city=form.city.data, country=form.country.data, unit=unit))
        elif form.get_weather_7days.data:
            return redirect(url_for('results_7_days',city=form.city.data, country=form.country.data, unit=unit))

    elif request.method == 'GET' and current_user.is_authenticated:
        form.city.data = current_user.city
        form.country.data = current_user.country
        return render_template("home.html", form=form)
    else:
        return render_template("home.html", form=form)
```

➢ @app.route(‘/results’) – in aceasta ruta este apelat API-ul prin functia `get_data(request.args.get('city'),request.args.get('country'),request.args.get('unit'))` folosind request.args.get pentru a query stringurile rezultate din completarea form-ului de pe pagina de home. Daca userul este logat, sistemul pentru unitatea de masura cu care se va face call-ul catre API va fi cel ales de catre acesta (by default este “metric”), altfel acesta va fi metric. Atat ariabila returnata de get_data cat si unit vor fi introduse in fisierul html care urmeaza sa fie rendered, urmand ca aceastea sa fie folosite in interiorul lui cu ajutorul templating engine-ului Jinja2.

```
app.route('/results',methods=['GET'])
ef results():
    try:
        if current_user.is_authenticated:
            weather = get_data(request.args.get('city'),request.args.get('country'),request.args.get('unit')
            return render_template('result.html', weather=weather, unit=request.args.get('unit'))
        else:
            weather = get_data(request.args.get('city'),request.args.get('country'),'metric')
            return render_template('result.html', weather=weather, unit='metric')

    except KeyError:
        flash('The city or country you entered is not valid !', 'danger')
        return redirect(url_for('home'))
    except e:
        return e
```

- @app.route('/results7days’)
  Similar cu /results, numai ca va fi folosita functia ce returneza informatii despre prognoza pe o saptamana. Aceasta ruta poate fi accesata doar daca utilizatorul este logat.

- @app.route('/login’)
  Este creat form-ul de Login. Userul este nevoit sa introduca emailul si parola contului.Se va face un query in baza de date pe baza emailului introdus dupa care se va compara parola introdusa cu ce hashuita din baza de data cu ajutorul bcrypt.check_password_hash. In caz ca datele sunt corecte, userul este redirectionat pe pagina de home, altfel acesta va primi un mesaj de atentionare si va fi rugat sa introduca iar datele.

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email=form.email.data).first()
        if user and bcrypt.check_password_hash(user.password, form.password.data):
            login_user(user, remember=form.remember.data)
            next_page = request.args.get('next')
            return redirect(next_page) if next_page else redirect(url_for('home'))
        else:
            flash('Login unsuccessful. Please check email and password !', 'danger')

    return render_template('login.html', title='Login', form=form)
```

- @app.route('/login’)
  Este creat form-ul de Register. Dupa ce userul introduce toate datele iar acestea sunt valide, parola este hashuita si introdusa in baza de date odata cu celelalte date prin intermediul variabilei user care este o instanta a modelului User.

```python
@app.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form = RegistrationForm()
    if form.validate_on_submit():
        hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        user = User(email=form.email.data, password=hashed_password, first_name=form.first_name.data, last_name=form.last_name.d
        db.session.add(user)
        db.session.commit()
        flash(f'Account created! You can now log in.', 'success')
        return redirect(url_for('login'))
    return render_template('register.html', title='Register', form=form)
```

- @app.route('/account’)
  Este creat form-ul pentru UserSettings. Cand userul aceseaza pagina, fieldurile din form vor fi populate in prealabil cu datele acestuia existente in baza de date. Acesta poate modifica datele existente si

poate alege sistemul de unitati de masura dorit printr-un dropdown. Userul poate accesa aceasta ruta doar daca este logat.

```python
@app.route('/account', methods=['GET', 'POST'])
@login_required
def account():
    form = UserSettingsForm()
    if form.validate_on_submit():
        current_user.email = form.email.data
        current_user.first_name = form.first_name.data
        current_user.last_name = form.last_name.data
        current_user.city  = form.city.data
        current_user.country = form.country.data
        current_user.measurement_unit = form.unit.data
        db.session.commit()
        flash('Your account has been updated!', 'success')
        return redirect(url_for('account'))
    elif request.method == 'GET':
        form.email.data = current_user.email
        form.first_name.data = current_user.first_name
        form.last_name.data = current_user.last_name
        form.city.data = current_user.city
        form.country.data = current_user.country
        form.unit.data = current_user.measurement_unit

    return render_template('account.html', title='Account', form=form)
```

6. BIBLIOGRAFIE

[1]https://ro.wikipedia.org/wiki/Python
[2]https://docs.microsoft.com/ro-ro/powerapps/maker/portals/vs-code-extension
[3]https://en.wikipedia.org/wiki/Flask_(web_framework)
[4]https://ro.wikipedia.org/wiki/HyperText_Markup_Language
[5]https://ro.wikipedia.org/wiki/Bootstrap
[6]https://app.diagrams.net/
[7][https://rapidapi.com/community/api/open-weather-map