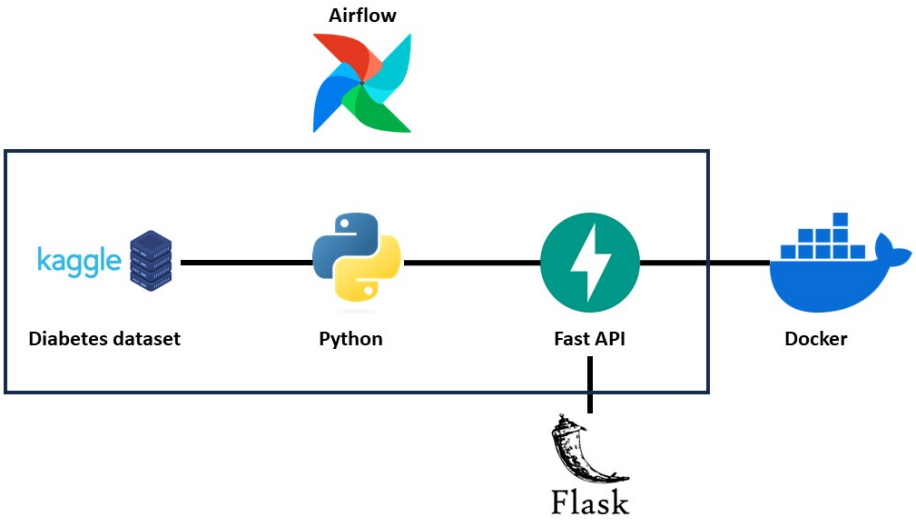
Université Chouaib Doukkali
Ecole Nationale des Sciences Appliquées d'El Jadida
Département Télécommunications, Réseaux et Informatique

Projet4 :

Mise en place d'un pipeline d'entrainement continu en utilisant Apache Airflow, FastAPI, Flask et docker

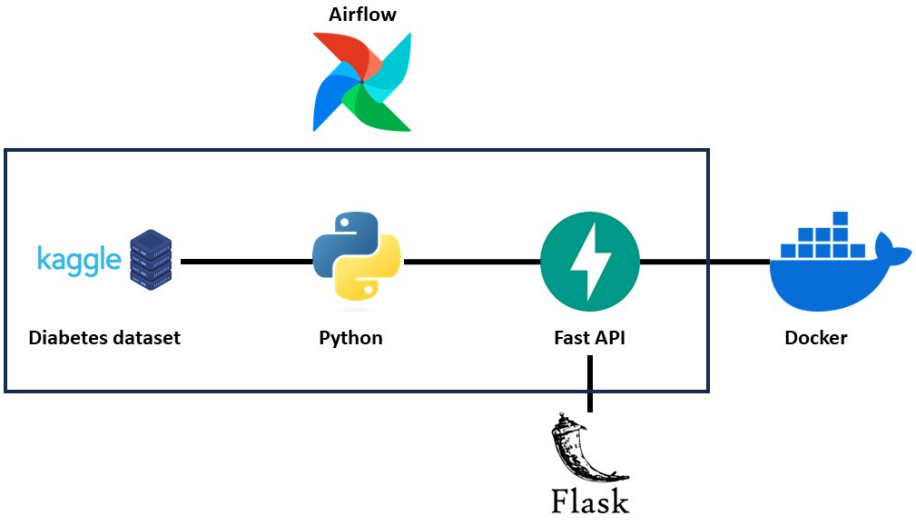


Professeur : F.KALLOUBI
Réalisé par : O.OUHAYOU

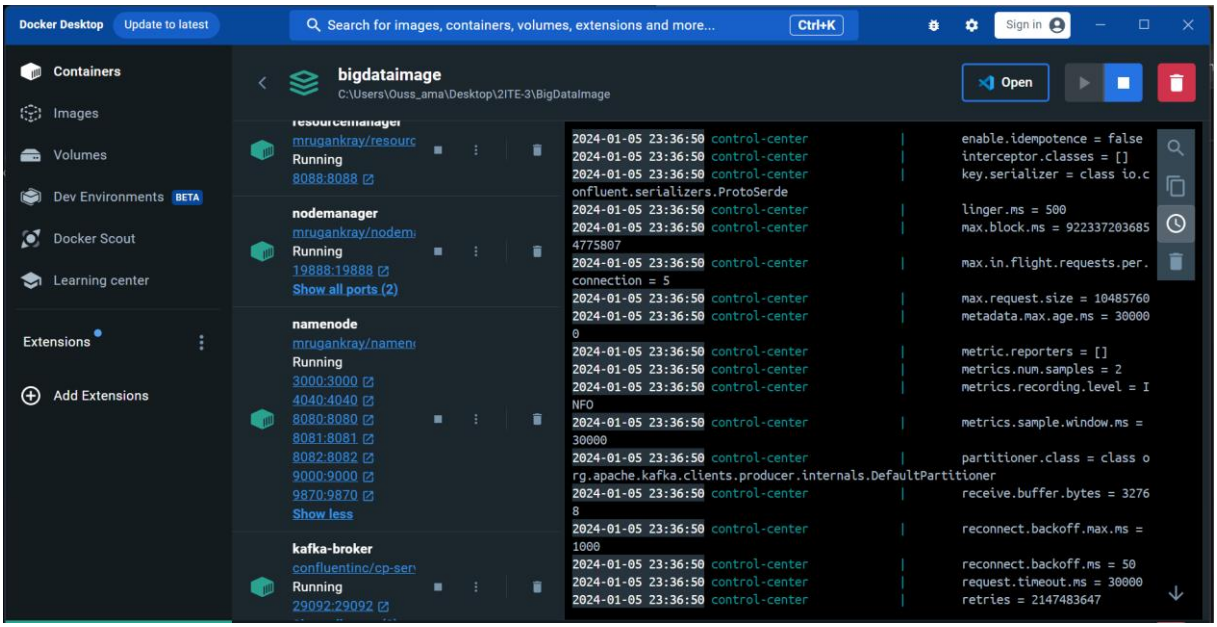
Table des matières

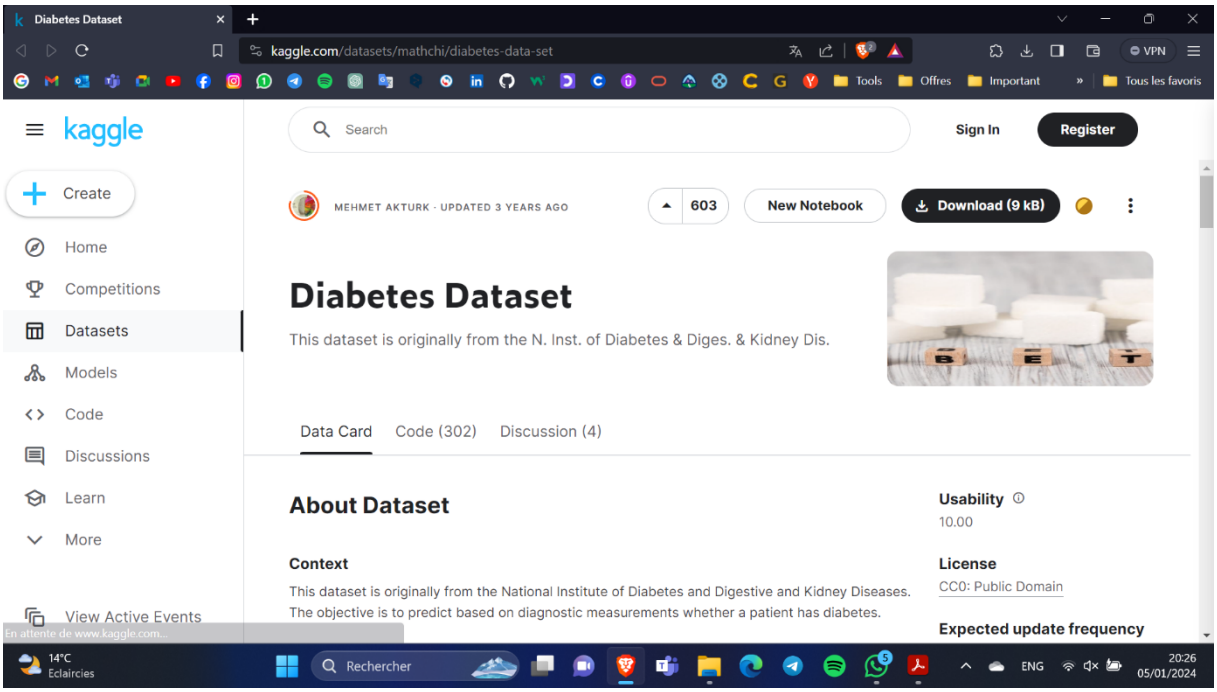
1) Objectif du projet :

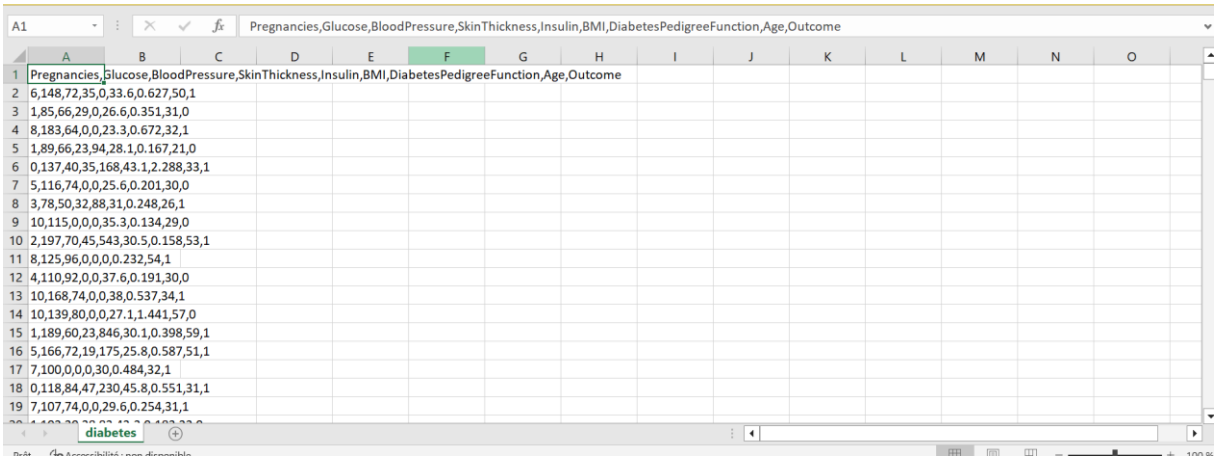
L'objectif de ce projet est d'automatiser l'ensemble du processus, de l'entraînement du modèle à la mise à disposition d'une API de prédiction, en passant par la création d'une application web pour l'interaction avec le modèle. L'utilisation d'outils tels que FastAPI, Apache Airflow, Flask, et Docker vise à rendre le processus efficace, automatisé et évolutif. Tous cela en opérant sur le dataset issue de kaggle :
https://www.kaggle.com/datasets/mathchi/diabetes-data-set



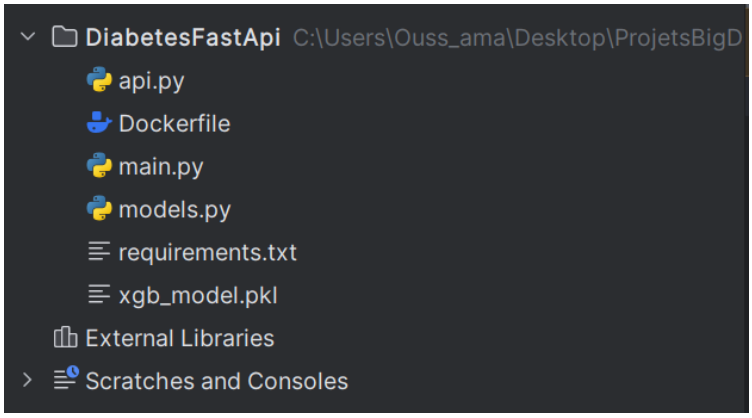
2) Installation et configuration :




Diabetes Dataset
kaggle.com/datasets/mathchi/diabetes-data-set
kaggle
Search
Sign In
Register
Create
Home
Competitions
Datasets
Models
Code
Discussions
Learn
More
View Active Events
MEHMET AKTURK · UPDATED 3 YEARS AGO
603
New Notebook
Download (9 kB)
Diabetes Dataset
This dataset is originally from the N. Inst. of Diabetes & Diges. & Kidney Dis.
Data Card
Code (302)
Discussion (4)
About Dataset
Context
This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases.
The objective is to predict based on diagnostic measurements whether a patient has diabetes.
Usability
10.00
License
CC0: Public Domain
Expected update frequency
14°C
Eclaircies
Rechercher
ENG
20:26
05/01/2024


A1
Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,DiabetesPedigreeFunction,Age,Outcome
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1
5,116,74,0,0,25.6,0.201,30,0
3,78,50,32,88,31,0.248,26,1
10,115,0,0,0,35.3,0.134,29,0
2,197,70,45,543,30.5,0.158,53,1
8,125,96,0,0,0,0.232,54,1
4,110,92,0,0,37.6,0.191,30,0
10,168,74,0,0,38,0.537,34,1
10,139,80,0,0,27.1,1.441,57,0
1,189,60,23,846,30.1,0.398,59,1
5,166,72,19,175,25.8,0.587,51,1
7,100,0,0,0,30,0.484,32,1
0,118,84,47,230,45.8,0.551,31,1
7,107,74,0,0,29.6,0.254,31,1
diabetes

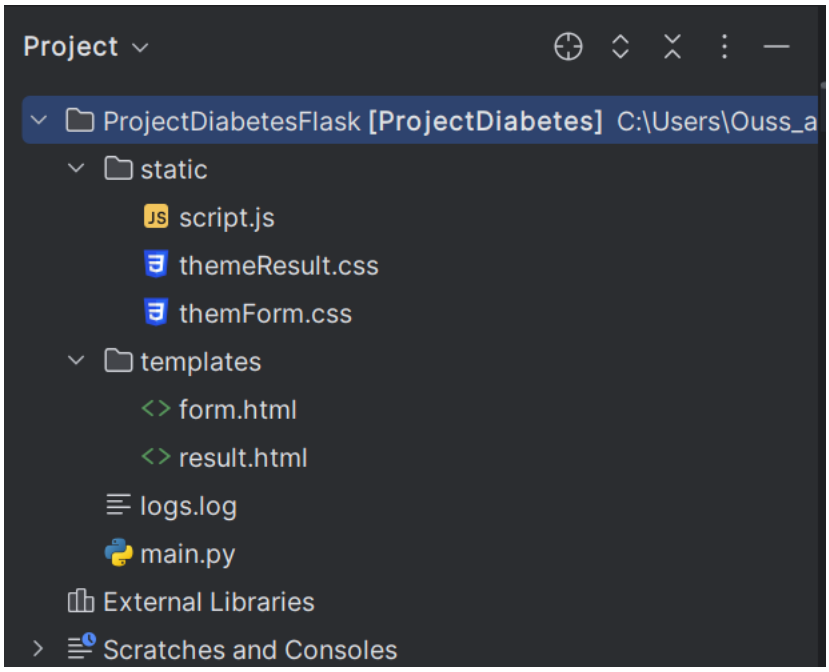
3) Créations du modèle sérialisé et de l'API




```python
import pickle
import uvicorn
from fastapi import FastAPI
import numpy as np  # Add this import

from models import Diabetes

app = FastAPI()
import joblib

# ...

# Chargement du modèle
model_path = "xgb_model.pkl"  # Mettez le chemin correct s'il est différent
model = joblib.load(open(model_path, "rb"))

@app.get("/")
def great():
    return {"message": "bonjour"}

1 usage (1 dynamic)
@app.post("/predict")
def predict(req: Diabetes):
    preg = req.Pregnancies
```

```python
@app.post("/predict")
def predict(req: Diabetes):
    preg = req.Pregnancies
    glucose = req.Glucose
    bp = req.BloodPressure
    skinthickness = req.SkinThickness
    insulin = req.Insulin
    bmi = req.BMI
    dpf = req.DPF
    age = req.Age
    features = np.array([preg, glucose, bp, skinthickness, insulin, bmi, dpf, age]).reshape(1, -1)  # Con
    prediction = model.predict(features)[0]
    probab = model.predict_proba(features)

    if prediction == 1:
        return {"ans": f"You have been tested positive with {probab[0][1]} probability"}
    else:
        return {"ans": f"You have been tested negative with {probab[0][0]} probability"}

if __name__ == "__main__":
    uvicorn.run(app)
```

4) Création de l’interface avec Flask



main.py

```python
from flask import Flask, request, render_template, jsonify
import requests  # Ajouter cette importation

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('form.html')  # Assurez-vous d'avoir un fichier 'form.html'

@app.route('/predict', methods=['POST'])
def predict():
    # Récupérer les valeurs du formulaire
    preg = float(request.form['Pregnancies'])
    glucose = float(request.form['Glucose'])
    bp = float(request.form['BloodPressure'])
    skinthickness = float(request.form['SkinThickness'])
    insulin = float(request.form['Insulin'])
    bmi = float(request.form['BMI'])
    dpf = float(request.form['DPF'])
    age = float(request.form['Age'])

    # Construire les données à envoyer à l'API FastAPI
    data = {
        "Pregnancies": preg,
```

```python
    # Construire les données à envoyer à l'API FastAPI
    data = {
        "Pregnancies": preg,
        "Glucose": glucose,
        "BloodPressure": bp,
        "SkinThickness": skinthickness,
        "Insulin": insulin,
        "BMI": bmi,
        "DPF": dpf,
        "Age": age
    }

    # Envoyer la requête POST à l'API FastAPI
    api_url = "http://localhost:80/predict"  # Assurez-vous que le port est correct
    response = requests.post(api_url, json=data)
    result = response.json()

    # Afficher le résultat
    return render_template('result.html', prediction_text=result['ans'])
if __name__ == "__main__":
    app.run(debug=True)
```

main.py form.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" type="text/css" href="/static/themForm.css">
    <title>Diabetes Prediction Form</title>
</head>
<body>
    <div class="container">
        <h1>Diabetes Prediction Form</h1>

        <form action="/predict" method="post">
            <!-- Ajoutez des champs pour chaque caractéristique -->
            Pregnancies: <input type="text" name="Pregnancies"><br>
            Glucose: <input type="text" name="Glucose"><br>
            Blood Pressure: <input type="text" name="BloodPressure"><br>
            Skin Thickness: <input type="text" name="SkinThickness"><br>
            Insulin: <input type="text" name="Insulin"><br>
            BMI: <input type="text" name="BMI"><br>
            DPF: <input type="text" name="DPF"><br>
            Age: <input type="text" name="Age"><br>
            <input type="submit" value="Predict">
        </form>
```

main.py form.html result.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
     <link rel="stylesheet" type="text/css" href="/static/themeResult.css">
    <title>Diabetes Prediction Result</title>
</head>
<body>
   <div class="container">
        <h2>Prediction Result:</h2>
        <p class="result">{{ prediction_text }}</p>

        <!-- Bouton de retour -->
        <a href="/" class="back-btn">Retour à l'accueil</a>
    </div>
</body>
</html>
```

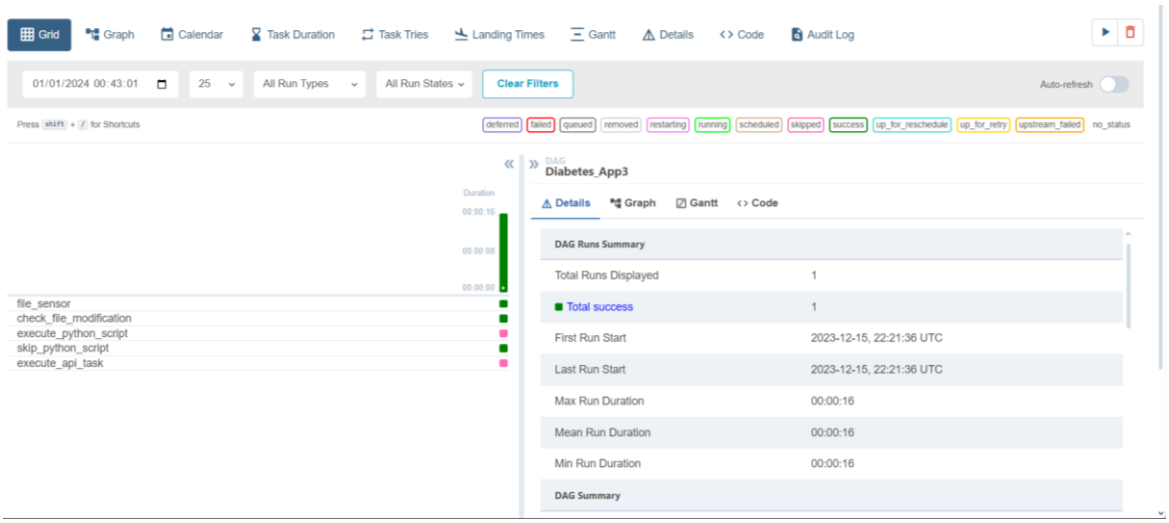
5) Mise en place avec Apache Airflow

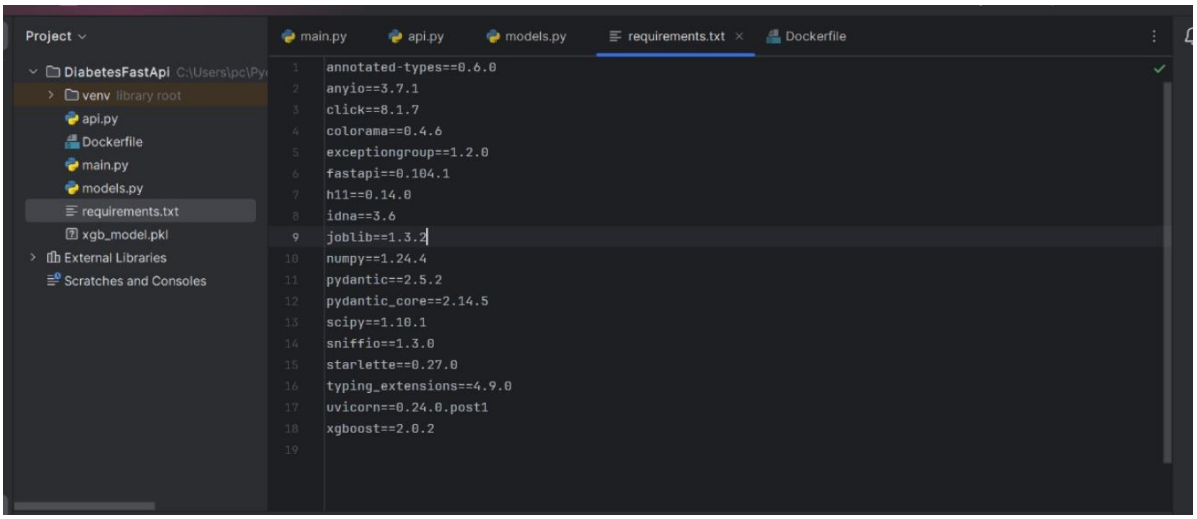
D: > continuous-training-pipeline-airflow > projet diabetes > airflow > dags > dag2.py > ...

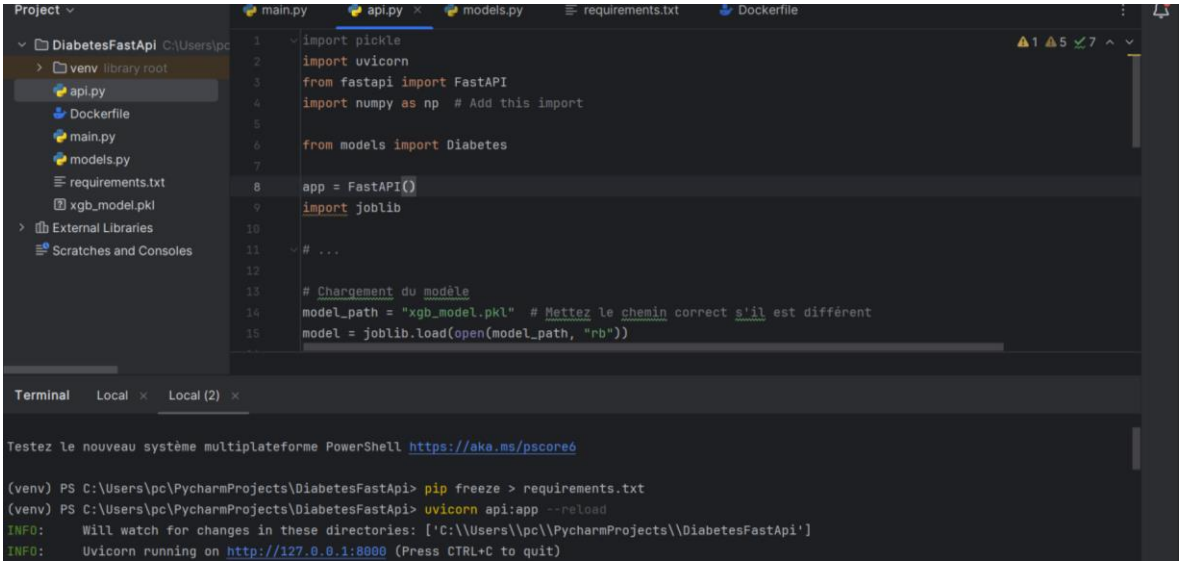
```python
from datetime import datetime, timedelta
from airflow.utils.dates import days_ago
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.sensors.filesystem import FileSensor
import requests
from pytz import timezone
from airflow.operators.bash import BashOperator
from airflow.operators.python import BranchPythonOperator
import os
from datetime import datetime
from subprocess import run

# Set the timezone to Casablanca
casablanca_tz = timezone('Africa/Casablanca')

default_args = {
    'owner': 'oussama',
    'depends_on_past': False,
    'start_date': datetime.now(casablanca_tz),
    'retries': 0,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(
    'Diabetes_App3',
    default_args=default_args,
    description='An Airflow DAG to automize the pipeline',
    schedule_interval='0 */1 * * *',  # Run every 5 minutes
    max_active_runs=1,  # Ensure only one run at a time
    catchup=False,  # Do not run backfill for the intervals between start_date and the current date
)


def check_file_modification(**kwargs):
    file_path = '/opt/airflow/dags/scripts/diabetes.csv'
    modified_time = os.path.getmtime(file_path)
    last_modified = datetime.utcfromtimestamp(modified_time)
    five_minutes_ago = datetime.now() - timedelta(minutes=5)
```

```
    return 'execute_python_script' if last_modified >= five_minutes_ago else 'skip_python_script'

def execute_api_task(**kwargs):
    # Assuming your FastAPI application script is named api_script.py
    api_script_path = 'opt.airflow.dags.scripts.APITask'
    uvicorn_command = f"uvicorn {api_script_path}:app --host 0.0.0.0 --port 8000 --reload > /opt/airflow/dags/uvicorn.log 2>&1"
    # Run the uvicorn command
    run(uvicorn_command, shell=True)

# Define the path to the file to be monitored
file_path = '/opt/airflow/dags/scripts/diabetes.csv'  # Replace with the actual path

# Create a FileSensor task to check for changes
file_sensor_task = FileSensor(
    task_id='file_sensor',
    filepath=file_path,
    poke_interval=60,  # Check every 60 seconds
    timeout=600,  # Timeout after 600 seconds (10 minutes)
    mode='poke',
    soft_fail=True,
    dag=dag,
)


check_file_task = BranchPythonOperator(
    task_id='check_file_modification',
    python_callable=check_file_modification,
    provide_context=True,
    dag=dag,
)

python_script_path = '/opt/airflow/dags/scripts/Final_diabetes.py'
python_task = BashOperator(
    task_id='execute_python_script',
    bash_command=f'python {python_script_path}',
    dag=dag,
)

skip_python_task = BashOperator(
```

```
    task_id='skip_python_script',
    bash_command='echo "No modification in the last 5 minutes. Skipping script execution"',
    dag=dag,
)

api_task = PythonOperator(
    task_id='execute_api_task',
    python_callable=execute_api_task,
    provide_context=True,
    dag=dag,
)


file_sensor_task >> check_file_task >> api_task
check_file_task >> [python_task , skip_python_task]
```

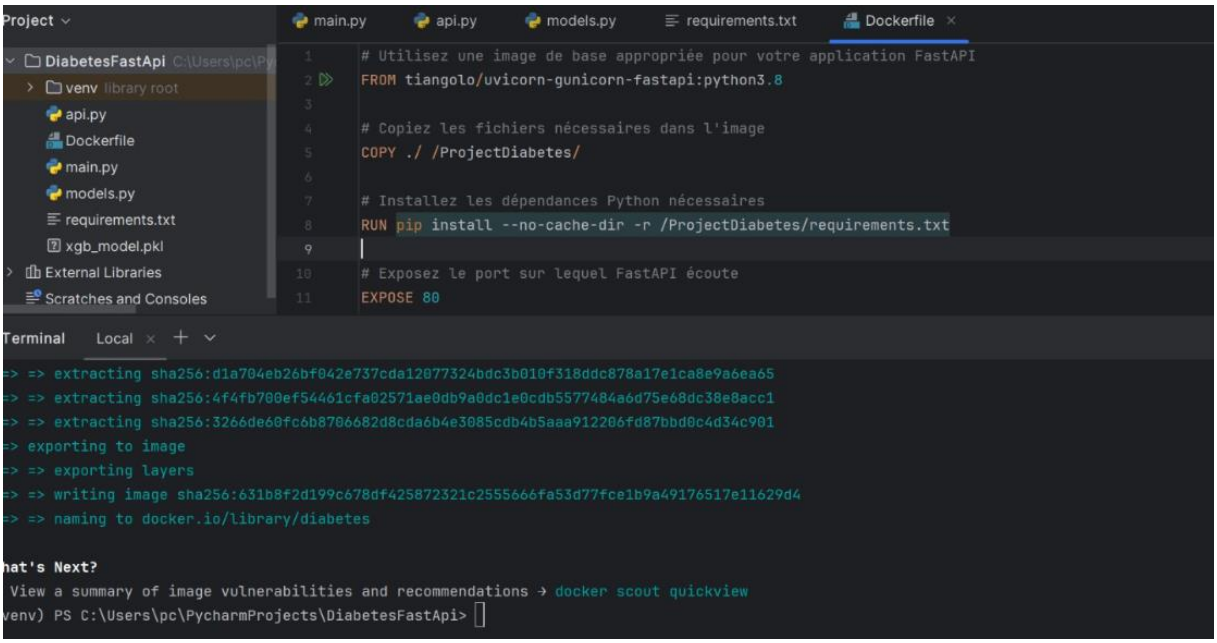


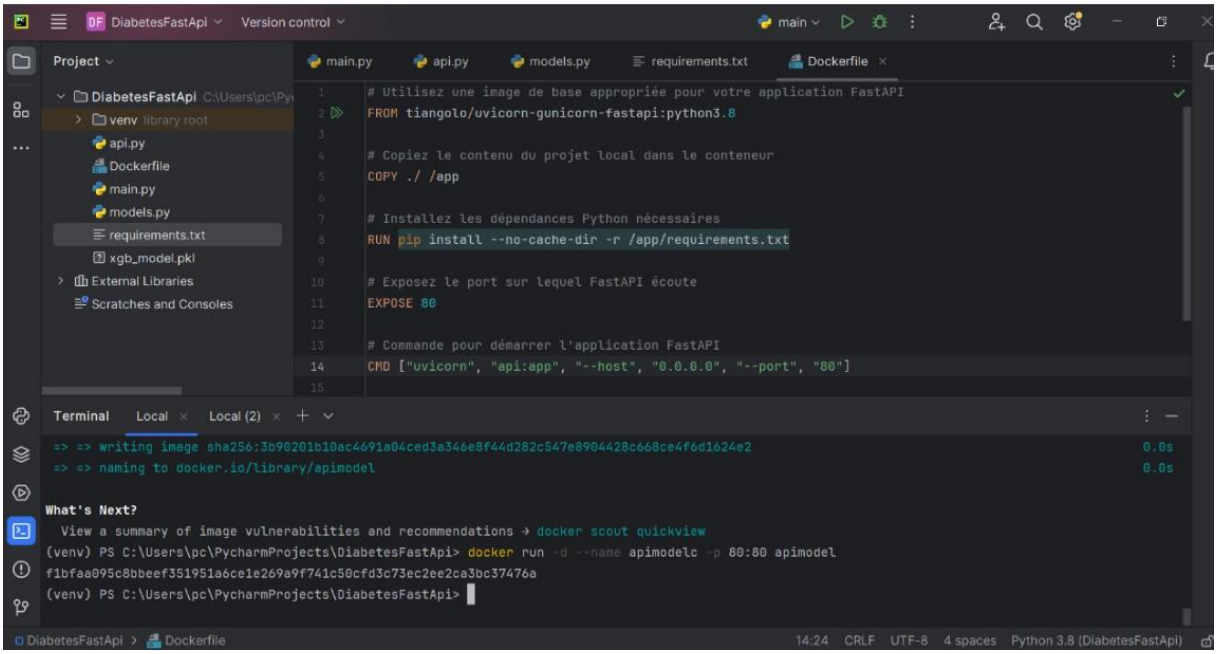

Project
DiabetesFastApi C:\Users\pc\Py
venv library root
api.py
Dockerfile
main.py
models.py
requirements.txt
xgb_model.pkl
External Libraries
Scratches and Consoles
main.py
api.py
models.py
requirements.txt
Dockerfile
annotated-types==0.6.0
anyio==3.7.1
click==8.1.7
colorama==0.4.6
exceptiongroup==1.2.0
fastapi==0.104.1
h11==0.14.0
idna==3.6
joblib==1.3.2
numpy==1.24.4
pydantic==2.5.2
pydantic_core==2.14.5
scipy==1.10.1
sniffio==1.3.0
starlette==0.27.0
typing_extensions==4.9.0
uvicorn==0.24.0.post1
xgboost==2.0.2

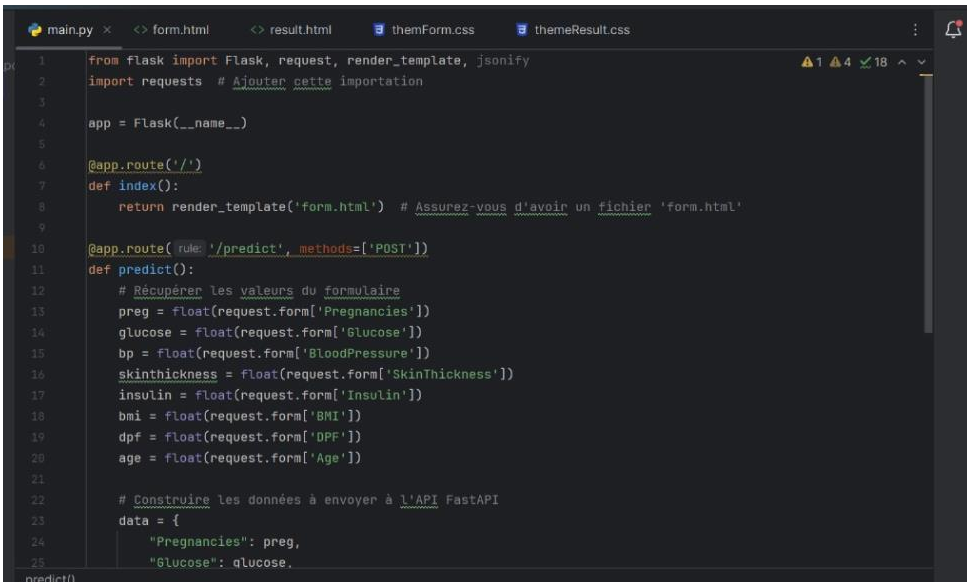

Project
DiabetesFastApi C:\Users\pc
venv library root
api.py
Dockerfile
main.py
models.py
requirements.txt
xgb_model.pkl
External Libraries
Scratches and Consoles
main.py
api.py
models.py
requirements.txt
Dockerfile
import pickle
import uvicorn
from fastapi import FastAPI
import numpy as np # Add this import
from models import Diabetes
app = FastAPI()
import joblib
# ...
# Chargement du modèle
model_path = "xgb_model.pkl" # Mettez le chemin correct s'il est différent
model = joblib.load(open(model_path, "rb"))
Terminal
Local
Local (2)
Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6
(venv) PS C:\Users\pc\PycharmProjects\DiabetesFastApi> pip freeze > requirements.txt
(venv) PS C:\Users\pc\PycharmProjects\DiabetesFastApi> uvicorn api:app --reload
INFO: Will watch for changes in these directories: ['C:\\Users\\pc\\PycharmProjects\\DiabetesFastApi']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)


'http://localhost/predict' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
"Pregnancies": 6,
"Glucose": 2,
"BloodPressure": 3,
"SkinThickness": 12,
"Insulin": 20,
"BMI": 0,
"DPF": 0,
"Age": 50
}'
Request URL
http://localhost/predict
Server response
Code
Details
200
Response body
{
"ans": "You have been tested negative with 0.686874270439148 probability"
}
Download
Response headers
content-length: 74
content-type: application/json
date: Tue,12 Dec 2023 08:56:26 GMT
server: uvicorn

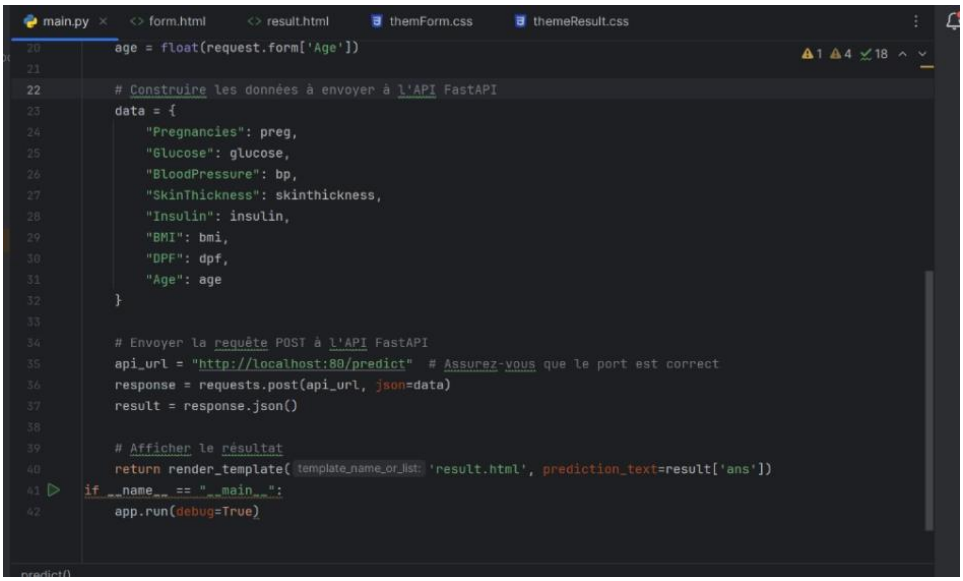
```
[+] Building 511.8s (7/8)                                                                                    docker:default
 => => extracting sha256:6f440e4468f7b98709fbd28336e242dba9b6ac5e248410e5d4ee69820ce884f4                        0.0s
 => => extracting sha256:ee788b96bd548c5af66dc2e482c8fd2592c567ce4395394bb5d7ba53368675cc                        0.0s
 => => extracting sha256:a0ad0fe3761e840b9d11ff92cb77fa209b1424dd45290f387ae9df0236682b3e                        0.0s
 => => extracting sha256:82f2ec9490f9d9c79e34d2bd7e5157db3702fa98c2efb9e2bbaf7fdd358415bb                        0.0s
 => => extracting sha256:93966f533967f820a282647b4ea7418d2958befaa138f6f724421ecf1b12b0f3                        0.0s
 => => extracting sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1                        0.0s
 => => extracting sha256:a9777f4f3e5f62a25bc962eb3b990ab23cf25b5ae8cb05e78505955f0c42716c                        0.0s
 => => extracting sha256:5766f1b810e3da9875fbe95a54421215089d7e79239f4f39aa2d237a720f6c30                        1.1s
 => => extracting sha256:0b1e404eb6ac14d56a20aa73046c71ac292d168aeea8419ff2a146fd7f90a550                        0.0s
 => [2/3] COPY ./ /app                                                                                          8.5s
 => [3/3] RUN pip install --no-cache-dir -r /app/requirements.txt                                             336.8s
 => => # ) (4.9.0)
 => => # Collecting uvicorn==0.24.0.post1
 => => #   Downloading uvicorn-0.24.0.post1-py3-none-any.whl (59 kB)
 => => #      ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 59.7/59.7 kB 625.6 kB/s eta 0:00:00
 => => # Collecting xgboost==2.0.2
 => => #   Downloading xgboost-2.0.2-py3-none-manylinux2014_x86_64.whl (297.1 MB)
```





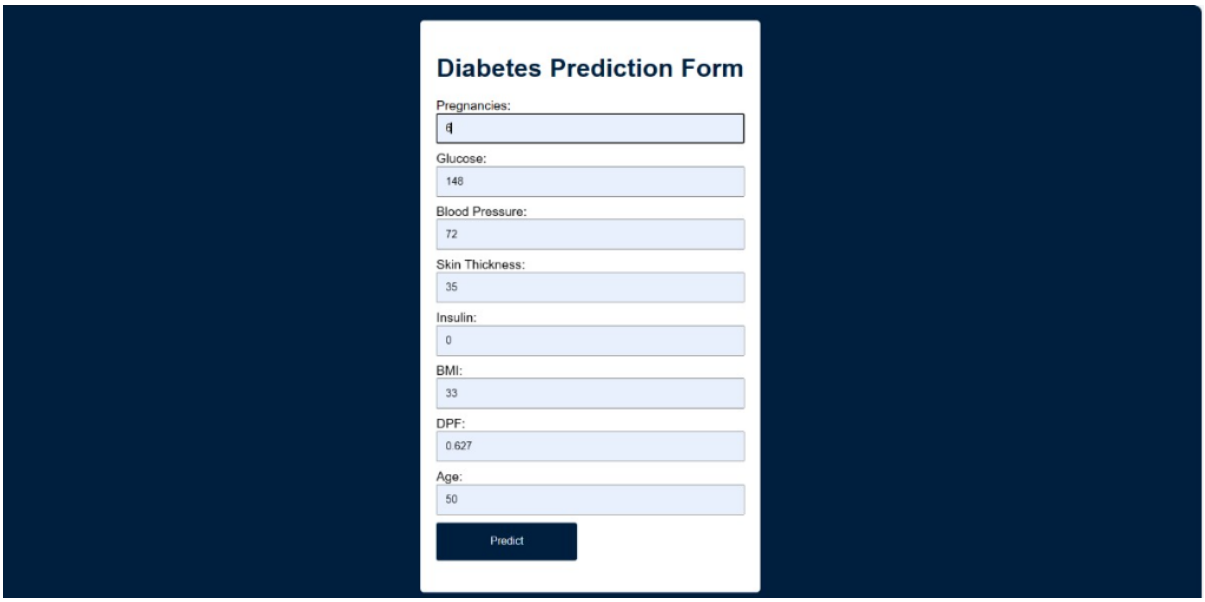


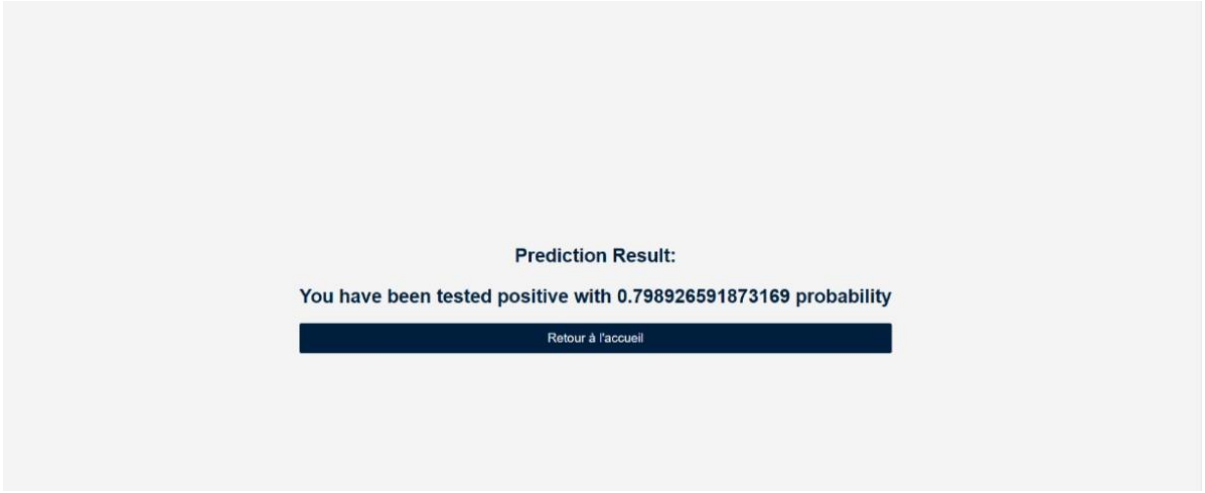
```python
from flask import Flask, request, render_template, jsonify
import requests  # Ajouter cette importation

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('form.html')  # Assurez-vous d'avoir un fichier 'form.html'

@app.route( rule: '/predict', methods=['POST'])
def predict():
    # Récupérer les valeurs du formulaire
    preg = float(request.form['Pregnancies'])
    glucose = float(request.form['Glucose'])
    bp = float(request.form['BloodPressure'])
    skinthickness = float(request.form['SkinThickness'])
    insulin = float(request.form['Insulin'])
    bmi = float(request.form['BMI'])
    dpf = float(request.form['DPF'])
    age = float(request.form['Age'])

    # Construire les données à envoyer à l'API FastAPI
    data = {
        "Pregnancies": preg,
        "Glucose": glucose,
```



```python
    age = float(request.form['Age'])

    # Construire les données à envoyer à l'API FastAPI
    data = {
        "Pregnancies": preg,
        "Glucose": glucose,
        "BloodPressure": bp,
        "SkinThickness": skinthickness,
        "Insulin": insulin,
        "BMI": bmi,
        "DPF": dpf,
        "Age": age
    }

    # Envoyer la requête POST à l'API FastAPI
    api_url = "http://localhost:80/predict"  # Assurez-vous que le port est correct
    response = requests.post(api_url, json=data)
    result = response.json()

    # Afficher le résultat
    return render_template( template_name_or_list: 'result.html', prediction_text=result['ans'])
if __name__ == "__main__":
    app.run(debug=True)
```

```
Run   main
  * Serving Flask app 'main'
  * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 119-139-627
```


Diabetes Prediction Form
Pregnancies:
6
Glucose:
148
Blood Pressure:
72
Skin Thickness:
35
Insulin:
0
BMI:
33
DPF:
0.627
Age:
50
Predict


Prediction Result:
You have been tested positive with 0.798926591873169 probability
Retour à l'accueil