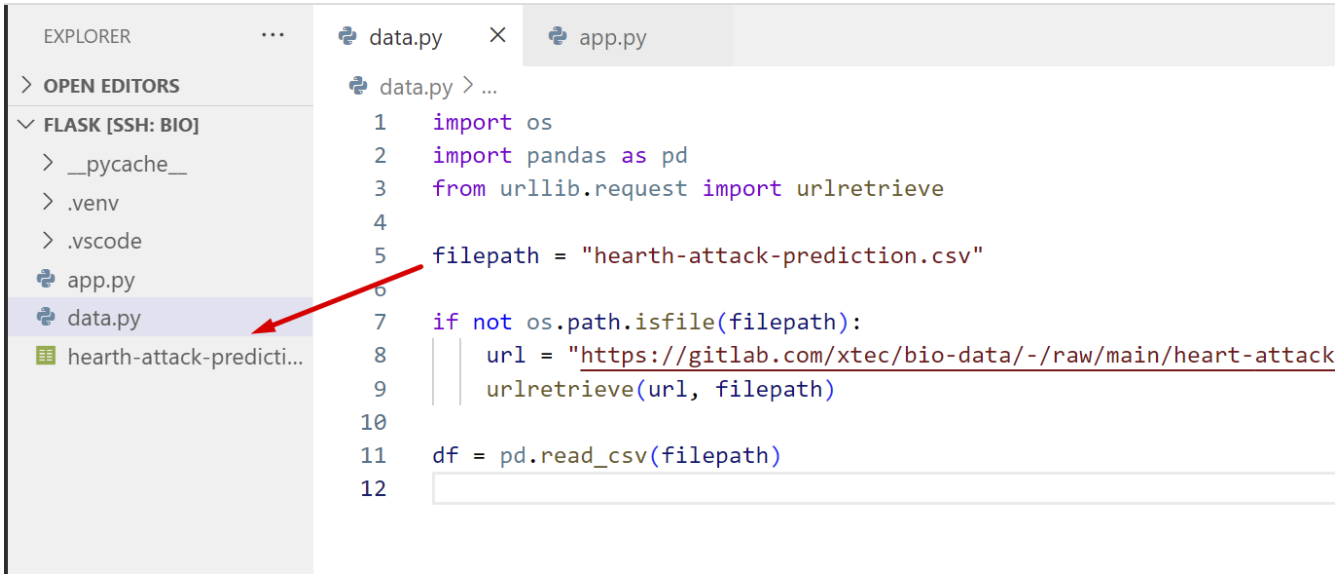
Exemple: Heart Attack

https://gitlab.com/xtec/bio-data/-/raw/main/heart-attack-prediction.csv

Creem un fitxer `data.py` per les dades:



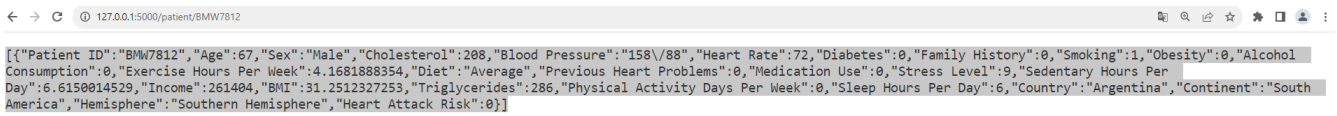
A continuació creem un fitxer `app.py` per poder realitza les operacions REST amb les dades dels pacients:

```python
import data
from flask import Flask, request, Response
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

df = data.df


class PersonResource(Resource):

    def get(self, id):
        data = df[df['Patient ID'] == id]
        if data.empty:
            return {"error": "patient id not found", "id": id}, 404

        return Response(
            data.to_json(orient="records"),
            mimetype='application/json')


api.add_resource(PersonResource, "/patient/<id>")

```

Podem consultar les dades d'un pacient:

```
(.venv) box@bio:~/flask$ http -b 127.0.0.1:5000/patient/CZE1114
[
    {
        "Age": 21,
        "Alcohol Consumption": 1,
        "BMI": 27.194973352,
        "Blood Pressure": "165/93",
        "Cholesterol": 389,
        "Continent": "North America",
        "Country": "Canada",
        "Diabetes": 1,
        "Diet": "Unhealthy",
        "Exercise Hours Per Week": 1.8132416179,
        "Family History": 1,
```

També des del navegador:



[{"Patient ID":"BMW7812","Age":67,"Sex":"Male","Cholesterol":208,"Blood Pressure":"158\/88","Heart Rate":72,"Diabetes":0,"Family History":0,"Smoking":1,"Obesity":0,"Alcohol Consumption":0,"Exercise Hours Per Week":4.1681888354,"Diet":"Average","Previous Heart Problems":0,"Medication Use":0,"Stress Level":9,"Sedentary Hours Per Day":6.6150014529,"Income":261404,"BMI":31.2512327253,"Triglycerides":286,"Physical Activity Days Per

Week":0,"Sleep Hours Per Day":6,"Country":"Argentina","Continent":"South America","Hemisphere":"Southern Hemisphere","Heart Attack Risk":0}]

Si el pacient no existeix retorna un error:

```
(.venv) box@bio:~/flask$ http -b 127.0.0.1:5000/patient/CZE1
{
    "error": "patient id not found",
    "id": "CZE1"
}
```

Exercici:

Implementa les operacions que falten: `PUT`, `POST` i `DELETE`

Consultes

També podem implementar un endpoint perquè els usuaris puguin buscar pacients en funció d'un camp determinat:

```
26  @app.route("/query/<field>/<value>")
27  def query(field, value):
28      data = df[df[field] == value]
29      if data.empty:
30          return {"error": "no data found", "field": field, "value": value}, 404
31
32      data = data.to_json(orient="records")
33
34      return Response(data, mimetype='application/json')
35
36
37  if __name__ == '__main__':
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 3 bash - flask

```
(.venv) box@bio:~/flask$ http -b 127.0.0.1:5000/query/Country/Canada
```

Es un implementació bàsica que funciona, i que permet al client tenir dades que pot processar i filtrar com vulgui:

```
(.venv) box@bio:~/flask$ http -b 127.0.0.1:5000/query/Country/Canada | jq
'.[] | {age: .Age , smoking: .Smoking }' | head -n 10
{
  "age": 21,
  "smoking": 1
}
{
  "age": 84,
  "smoking": 1
}
{
  "age": 90,
(.venv) box@bio:~/flask$
```

Però que passa si fa una consulta amb `Age` ?

```
(.venv) box@bio:~/flask$ http -b 127.0.0.1:5000/query/Age/30
{
    "error": "no data found",
    "field": "Age",
    "value": "30"
}

(.venv) box@bio:~/flask$
```

Els paràmetres d'entrada de la funció `query` per defecte són `string` si no especifiquem altre cosa:

```
26  @app.route("/query/<field>/<value>")
27  def query(field, value):
28
29      if field == "Age":
30          value = int(value)
31
32      data = df[df[field] == value]
33      if data.empty:
34          return {"error": "no data found", "field": field, "value": value}, 404
35
36      data = data.to_json(orient="records")
37
```

hem de convertir l'edat en un int

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 3

```
(.venv) box@bio:~/flask$ http -b 127.0.0.1:5000/query/Age/30
```