**Projet final 3LPIC**

Dureuil Corentin Mathia Esteban

Contexte:

Sup1nfo souhaite mettre en place un système de notation automatique des codes. L'objectif est de fournir un outil qui attribue automatiquement une note à un code soumis par un candidat pour un exercice donné.

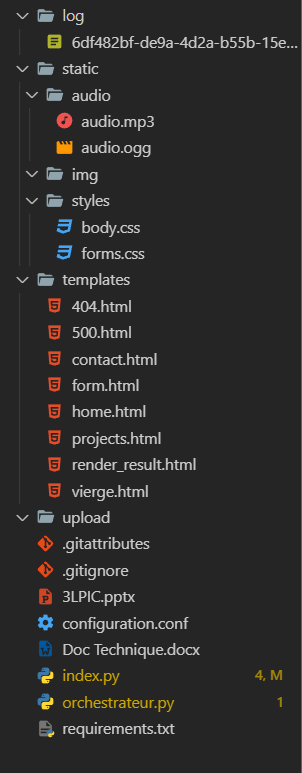
Infrastructure :

Sommaire :

1. Site Web
2. Systèmes de notations

# Site Web

Voici l’architecture du projet :



index.py :

import requests

import json

import os

import uuid

import shutil

import threading

import os

from threading import Thread

from orchestrateur import Orchestrateur

from time import sleep

from flask import Flask, request, render\_template

app = Flask(\_\_name\_\_, template\_folder='templates')

os.chdir(os.path.dirname(os.path.realpath(\_\_file\_\_)))

ALLOWED\_EXTENSIONS = ['py','c']

@app.errorhandler(404)

def page\_not\_found(e):

# note that we set the 404 status explicitly

return render\_template('404.html'), 404

@app.errorhandler(500)

def page\_not\_found(e):

# note that we set the 500 status explicitly

return render\_template('500.html'), 500

def returnJson(content):

status = content["status"]

del content['status']

response = app.response\_class(

response=json.dumps(content),

status=status,

mimetype='application/json'

)

return response

def allowed\_file(filename):

return '.' in filename and filename.split('.')[-1] in ALLOWED\_EXTENSIONS

@app.route('/render', methods =["GET", "POST"])

def render():

orchestrateur = Orchestrateur()

if request.method == "POST":

uuidMake = uuid.uuid1()

json\_receive = request.form.to\_dict()

json\_receive["uuidMake"] = str(uuidMake)

del json\_receive["Send"]

file = request.files['justification']

if file and allowed\_file(file.filename) and orchestrateur.if\_login(json\_receive):

extension = file.filename.split('.')[-1]

filename = f"{uuidMake}.{file.filename.split('.')[-1]}"

os.mkdir("upload/" + json\_receive["uuidMake"])

file.save("upload/" + json\_receive["uuidMake"] + "/" + filename)

result = True

Thread(target=orchestrateur.lunch\_correction,args=(json\_receive, extension)).start()

else:

result = False

return render\_template(f"render\_result.html",result = result)

return render\_template("form.html", exos = orchestrateur.get\_all\_exercice())

@app.route('/', methods =["GET"])

def home():

user = request.args.get('userid')

marks = None

if user:

orchestrateur = Orchestrateur()

marks = orchestrateur.all\_mark\_from\_user(user)

one = marks[0]["user"]

else:

orchestrateur = Orchestrateur()

marks = orchestrateur.best\_mark\_by\_exercice()

one = None

return render\_template("home.html", marks=marks, one=one)

@app.route("/projects", methods=["GET"])

def projects():

orchestrateur = Orchestrateur()

return render\_template("projects.html", projects\_data=orchestrateur.get\_all\_project())

@app.route("/contact", methods=["GET"])

def contact():

return render\_template("contact.html")

if \_\_name\_\_=='\_\_main\_\_':

app.run(debug = True)

orchestrateur.py :

import json

import pymysql

import logging

import configparser

import hashlib

import os

import shutil

from difflib import SequenceMatcher

class Orchestrateur():

def \_\_init\_\_(self) -> None:

self.config = self.get\_config()

logging.basicConfig(filename=f"log/6df482bf-de9a-4d2a-b55b-15ec92ea60c0.log",

format='%(asctime)s %(message)s',

filemode='a',

level=logging.INFO)

self.logger = logging.getLogger()

db = self.config["database"]

connection = pymysql.connect(host = db["host"], user = db["user"], db = db["db"] , password = db["pwd"], autocommit = 1)

self.cur = connection.cursor()

def get\_config(self) -> configparser.ConfigParser:

config = configparser.ConfigParser()

config.read("configuration.conf")

return config

def convertResultFetchallToListOfDict(self, data, keys):

return [{keys[column] : row[column] if "date" not in keys[column] else row[column].timestamp() for column in range(len(row))} for row in data]

def get\_all\_exercice(self):

result = {}

self.cur.execute("SELECT e.titre, e.id, l.langage FROM exercice e JOIN langage l ON e.langage = l.id")

all\_exercice\_brut = self.convertResultFetchallToListOfDict(self.cur.fetchall(), ["titre", "value", "langage"])

for exercice in all\_exercice\_brut:

if exercice["langage"] not in result:

result[exercice["langage"]] = []

result[exercice["langage"]].append({"titre" : exercice["titre"], "value" : exercice["value"]})

return result

def all\_mark\_from\_user(self, user\_id : int):

self.cur.execute("SELECT e.titre, r.mark, u.name, l.langage FROM render r JOIN user u ON u.id = r.user JOIN exercice e ON e.id = r.exercice JOIN langage l ON l.ID = e.langage where u.UUID = %s order by e.titre asc, r.mark desc", user\_id)

return self.convertResultFetchallToListOfDict(self.cur.fetchall(), ["exo", "mark", "user", "langage"])

def best\_mark\_by\_exercice(self):

self.cur.execute("SELECT e.titre, max(r.mark), u.name, l.langage FROM render r JOIN user u ON u.id = r.user JOIN exercice e ON e.id = r.exercice JOIN langage l ON l.ID = e.langage GROUP BY e.id")

return self.convertResultFetchallToListOfDict(self.cur.fetchall(), ["exo", "mark", "user", "langage"])

def get\_all\_project(self):

self.cur.execute("SELECT e.titre, e.description, e.attent, l.langage FROM exercice e JOIN langage l ON e.langage = l.id")

return self.convertResultFetchallToListOfDict(self.cur.fetchall(), ["titre", "description", "attent", "langage"])

def if\_login(self, dict):

return self.cur.execute("SELECT \* from user where pwd = %s and UUID = %s", (hashlib.md5(dict["pwd"].encode()).hexdigest(), dict["uuid"]))

def get\_user(self, pwd, uuid):

self.cur.execute("SELECT id from user where pwd = %s and UUID = %s", (hashlib.md5(pwd.encode()).hexdigest(), uuid))

return self.cur.fetchall()[0][0]

def get\_exercice(self, id\_exo):

self.cur.execute("SELECT attent, correction from exercice where id=%s", (id\_exo))

return self.convertResultFetchallToListOfDict(self.cur.fetchall(), ["attent", "correction"])[0]

def lunch\_correction(self, json, extension):

self.logger.info(f'lancement de notation pour {json} avec un fichier de type {extension}')

mark = None

if extension == "py":

#os.system('python upload' + "/" + json["uuidMake"] + "/" + json["uuidMake"] +".py")

os.system("./python.sh")

elif extension == "c":

os.system("./c.sh")

else:

return False

witness\_exercice = self.get\_exercice(json["nb\_exo"])

try:

with open("upload" + "/" + json["uuidMake"] + "/result.txt", "r") as f:

attent\_render = "".join(f.readlines())

except Exception:

mark = 0

attent\_render = ""

try:

with open("upload" + "/" + json["uuidMake"] + "/" + json["uuidMake"] + "." + extension, "r") as f:

file\_render = "".join(f.readlines())

except Exception:

mark = 0

file\_render = ""

if not mark:

coeff\_result = SequenceMatcher(None, witness\_exercice["attent"], attent\_render).ratio()

coeff\_file = SequenceMatcher(None, witness\_exercice["correction"], file\_render).ratio()

mark = 50\*coeff\_result + 50\*coeff\_file

self.logger.info('5')

self.cur.execute("INSERT INTO `render`(`user`, `exercice`, `mark`) VALUES (%s, %s, %s);", (self.get\_user(json["pwd"], json["uuid"]), json["nb\_exo"], mark))

shutil.rmtree("upload" + "/" + json["uuidMake"])

return True

if \_\_name\_\_ == "\_\_main\_\_":

o = Orchestrateur()

print(o.all\_mark\_from\_user(111111))

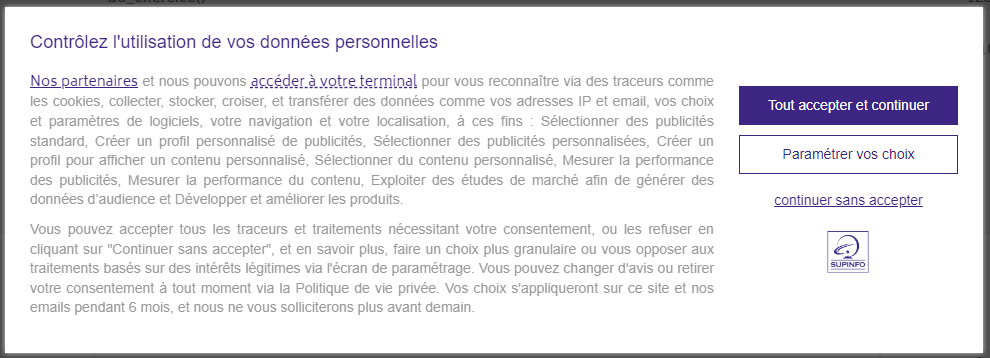
print(o.best\_mark\_by\_exercice())

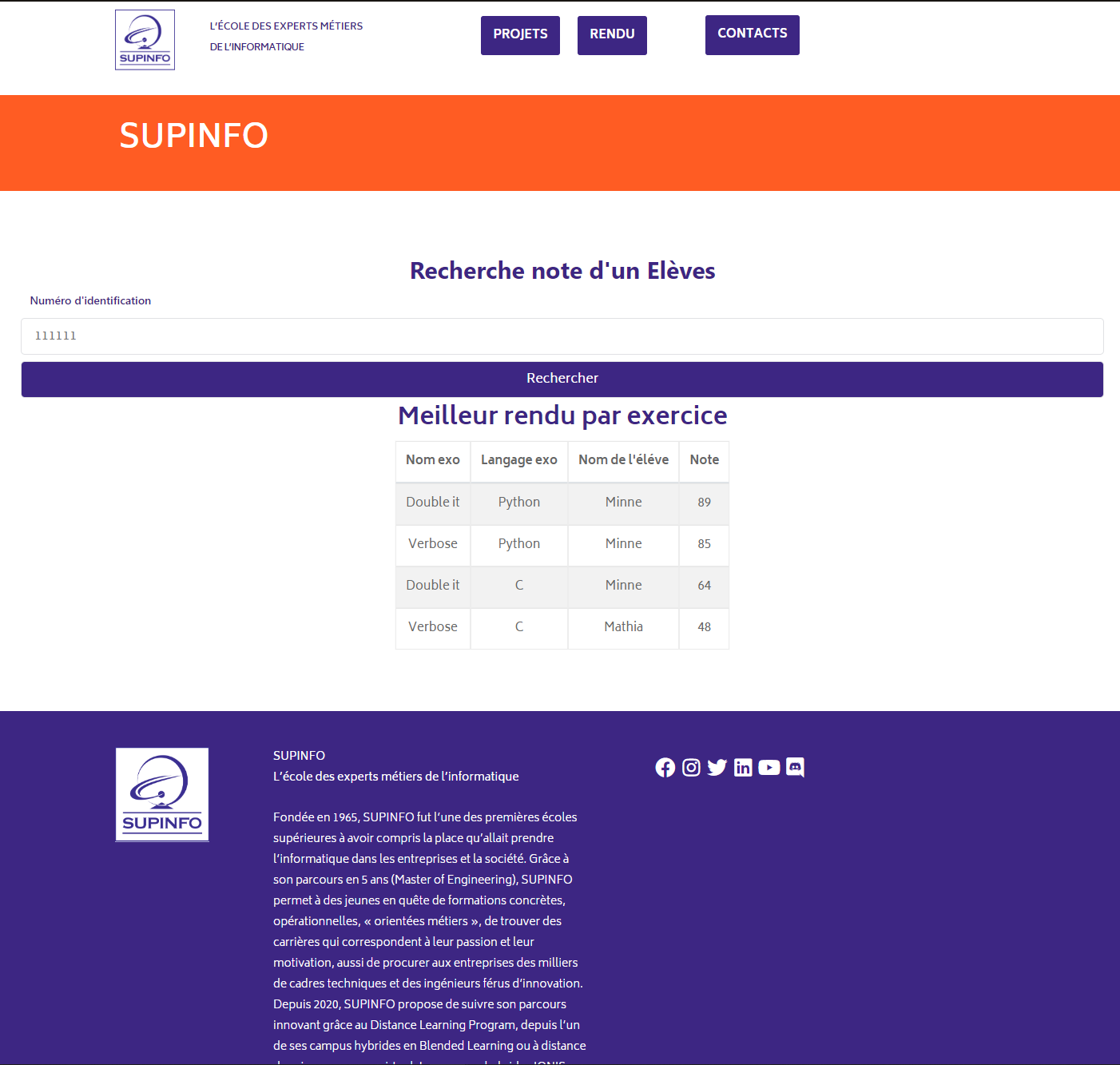
print(o.get\_all\_exercice())

print(o.get\_all\_project())

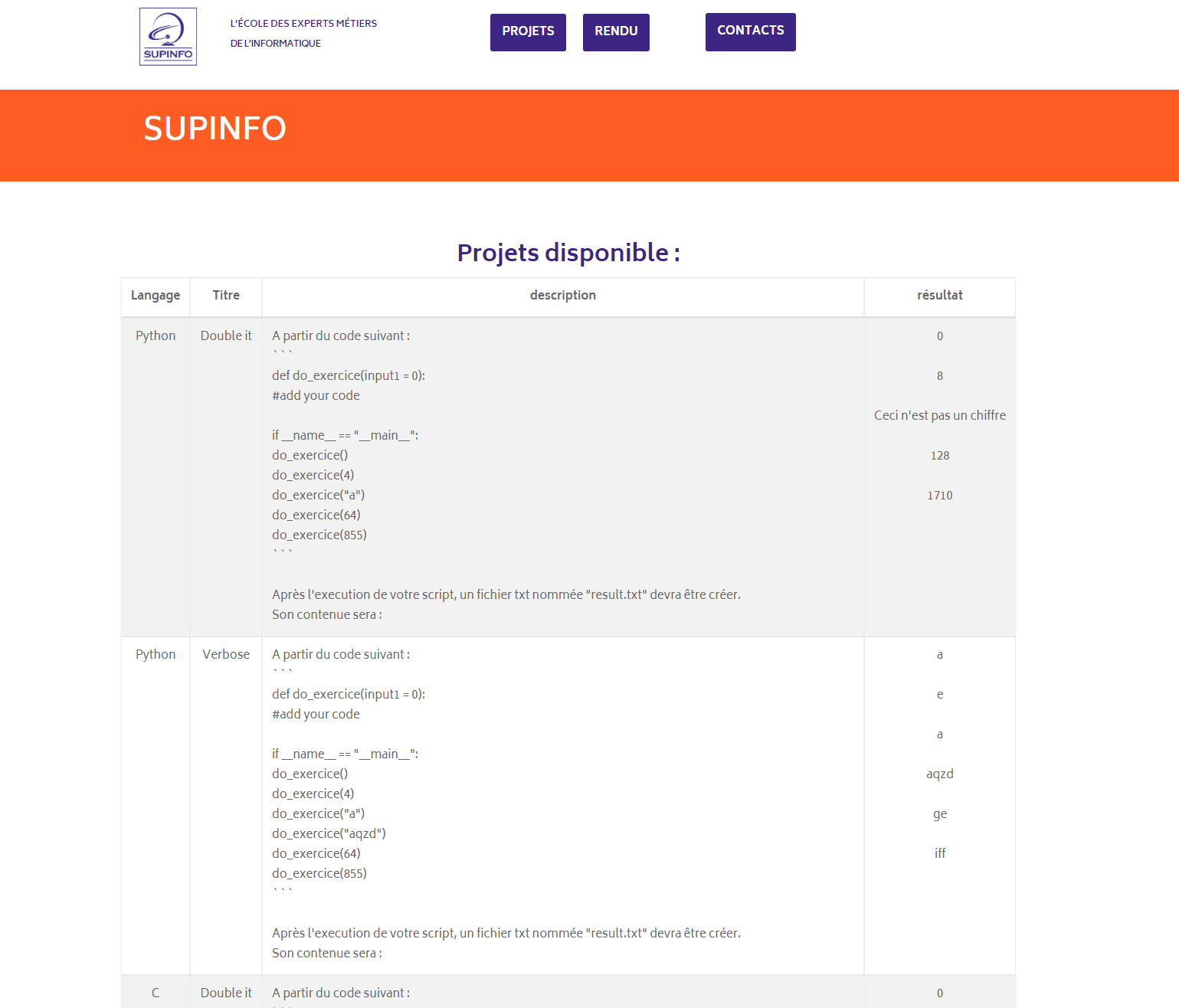
Pour des raisons évidentes de place nous n’allons pas documenter nos fichiers ressources dans les dossiers ./statique et ./templates.

Cependant voici quelques exemples du rendu visuel du site, de leur “responsivité” ainsi que leur utilité.

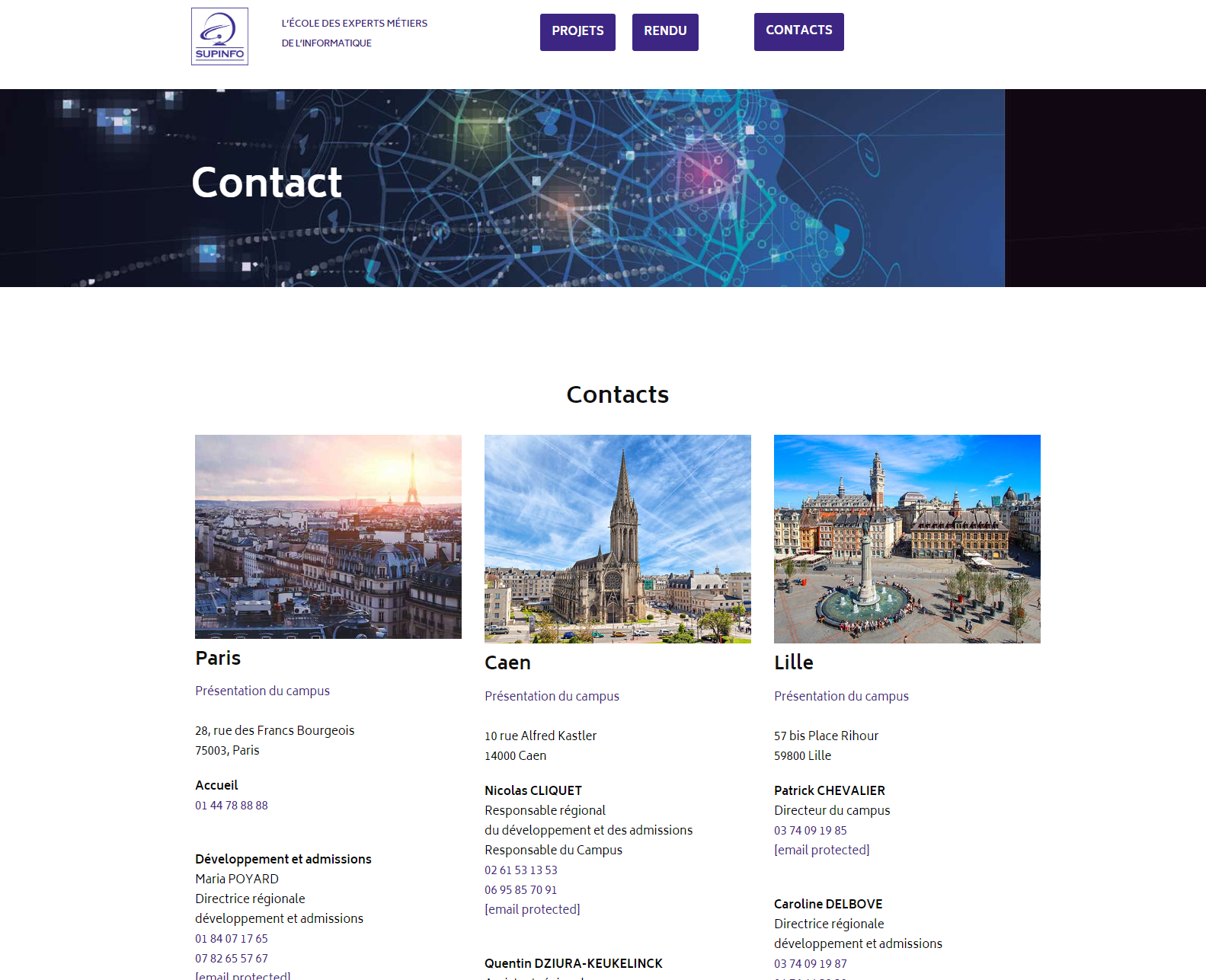
Dans un soucis de respect de la RGPD nous avons mis en place un système pour la gestion de la RGPD et demandons l’avis à nos utilisateurs : 

Page d'accueil : 

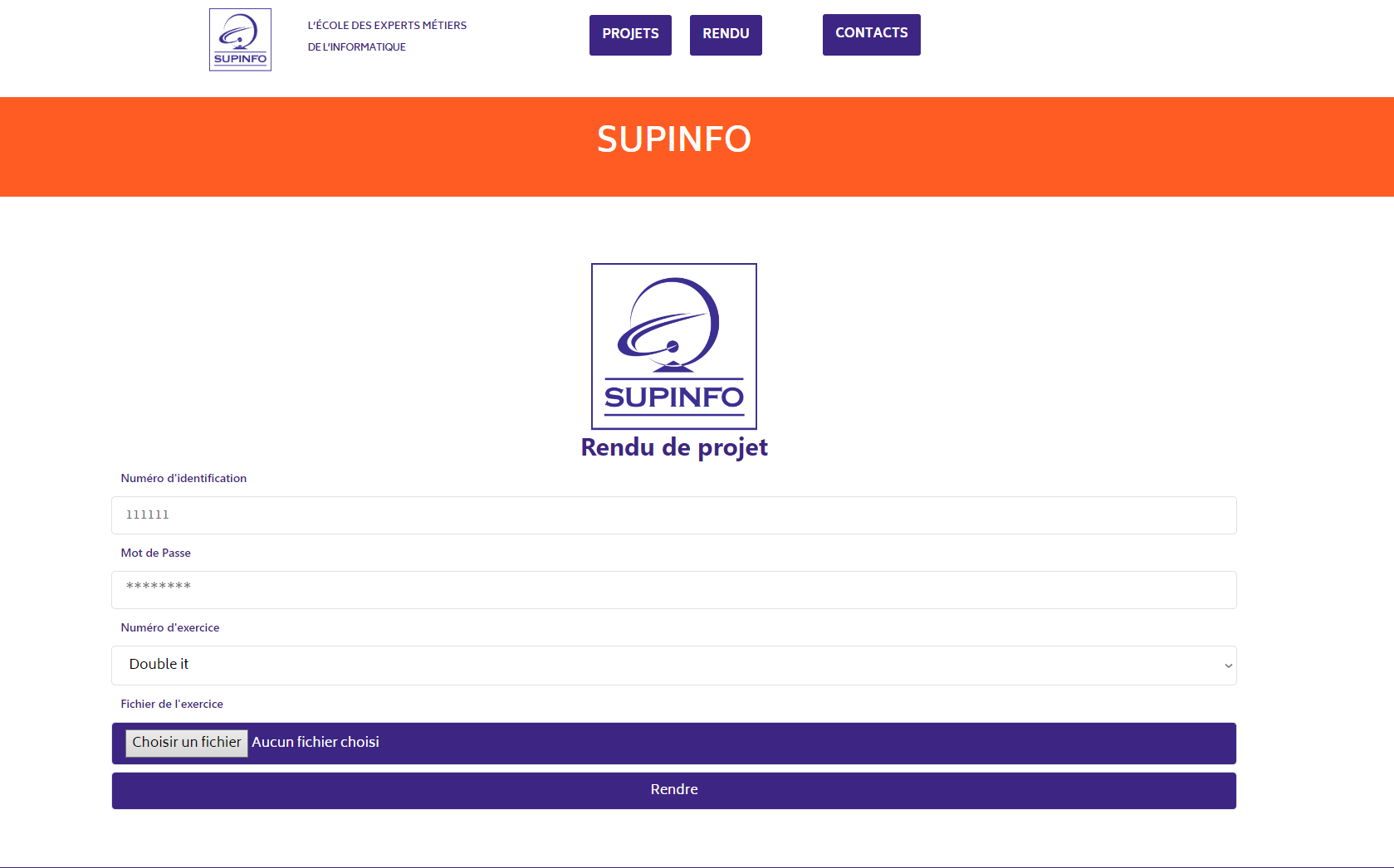
Cette page permet à l’utilisateur de voir les meilleures notes par exercice rendu ainsi que via l’espace de recherche d’obtenir l’ensemble des notes d’un numéro d’identification.

page des exercices : 

On peut parcourir dans cette page l’ensemble exercices disponibles avec les critères de réussite de l’exercice

page de contact de l’école : 

page de rendu des exercices:

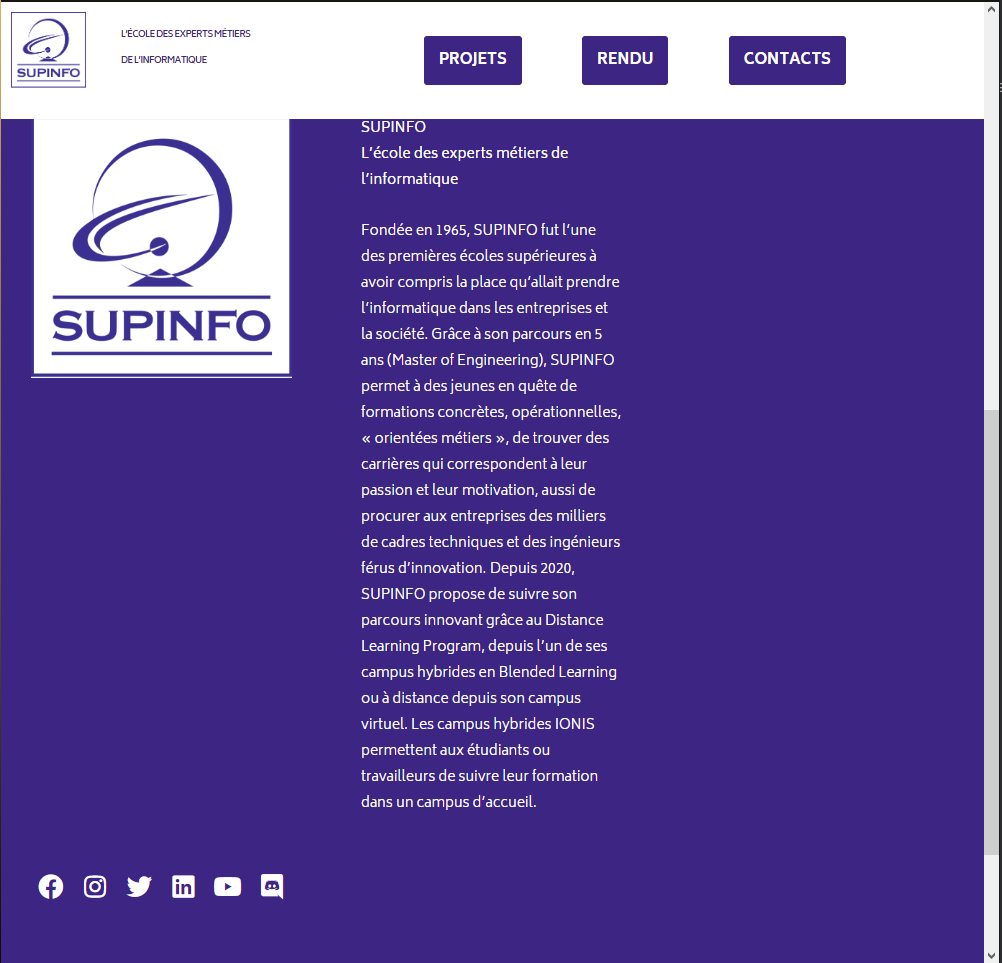


Les étudiants peuvent ainsi rendre leur exercice en s’identifiant, sélectionnant leur exercice et en uploadant leur fichier d’exercice. Tout ceci sur un fond musical : Arstotzkan Anthem en référence satirique de “Paper Paper please”.

Suite à cela ils ont une page leur informant du bon rendu ou non de leur exercice :



Evidemment les pages sont “responsive “ et peuvent s’adapter à plusieurs formes d’écrans. Voici un exemple sur la partie basse de la page au niveau du footer



# 

# Systèmes de notations

Nous avons pris le parti pris de juger les élèves essentiellement sur le résultat final. En effet leur notation est basé à 50% sur le résultat final du fichier txt produit à la fin de leur exercice. L’autre partie de la note est basée sur la ressemblance entre le fichier de script rendu par l'élève et la correction fournis par le professeur chargé de l’exercice. Ce qui fait qu’un élève peut passer l'exercice en fournissant un script qui correspond à 100% aux attentes et avoir 50% des points. Les points supérieurs à la moyenne valorisent donc les élèves studieux de bien faire leur algorithme.

Le système pour comparer nos 2 éléments se base sur n'importe quel type, tant que les éléments de la séquence sont hachables. L'algorithme de base est antérieur, et un peu plus sophistiqué, à un algorithme publié à la fin des années 1980 par Ratcliff et Obershelp sous le nom hyperbolique "gestalt pattern matching". L'idée de base est de trouver la plus longue sous-séquence contiguë qui ne contient pas d'éléments "inutiles".(R-O ne traite pas des éléments inutiles). La même idée est ensuite appliquée récursivement aux morceaux des séquences situées de part et d’autre de la sous-séquence correspondante. Cela ne donne pas des séquences d'édition minimales mais tend à produire des correspondances qui "semblent correctes" pour les gens.