Livrable projet 7 (PDF)

Projet codé en python

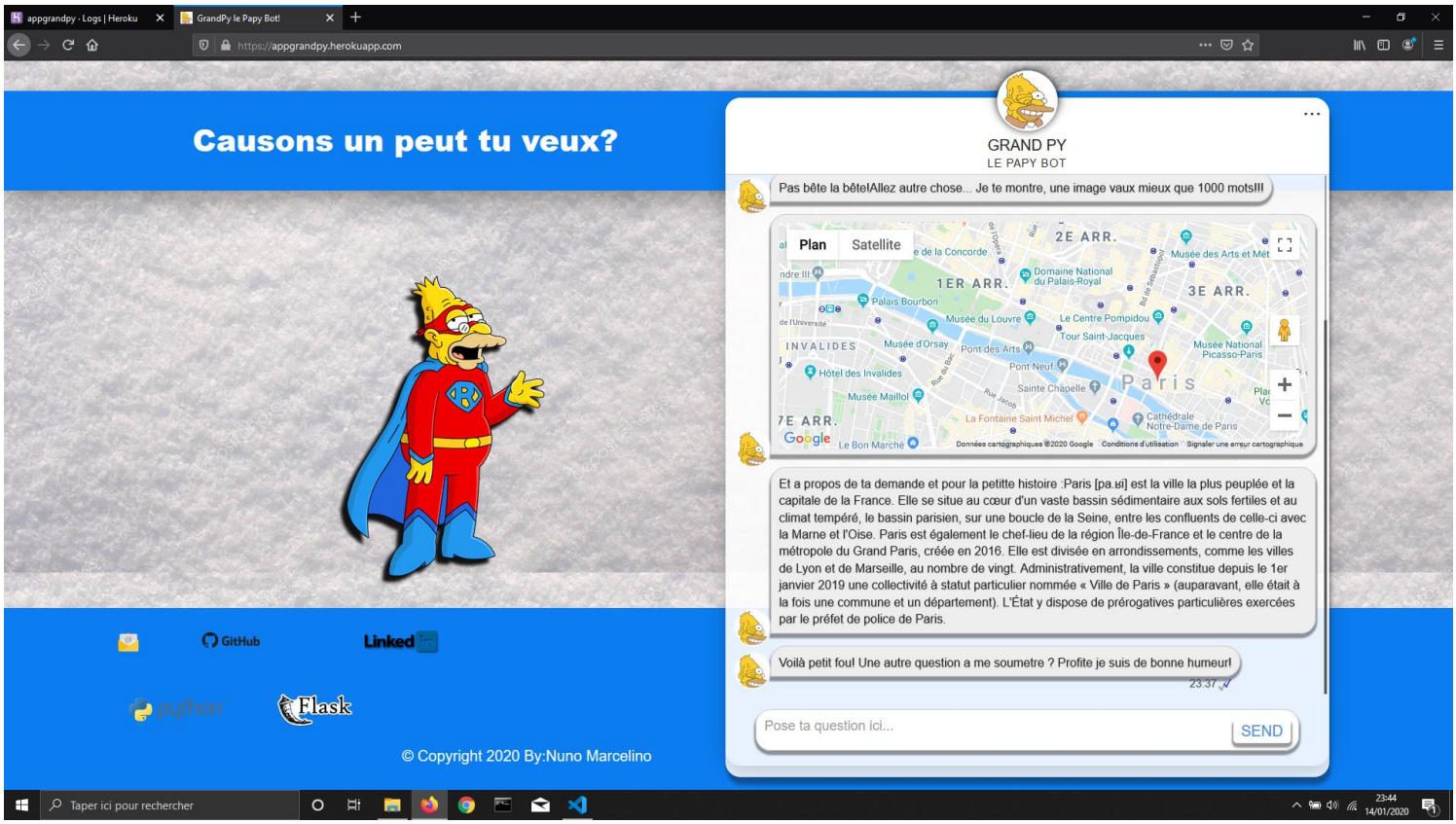
en utf8 et tournant en environnement virtuel et avec la méthode de TDD (Test Driven Devlopement).

https://github.com/NunoMars/GrandPy

Nuno Ricardo Silva Marcelino

https://trello.com/b/FtaZfSXV/grandpyapp

Vois-ci la page du site en production sur Heroku (Frontend):



Avant d'en arriver là, j'ai donc mis en place tout le Backend, avec l'utilisation de Flask et les tests avec Pytest.

Le programme se lance par le module « run.py » et est divisée un modules, ayant chaque un une fonction bien définie.

Le dossier « tests » contiens tous les tests, un test par module du programme.

Le dossier « GrandPyApp » contiens l'application avec les modules du « parseur » les « requettes » et un module « process » qui vas instancier et gérer le tout, afin de ne pas alourdir le module « views » :

```python
from flask import Flask, render_template, request, jsonify
from .process import grandPyWork


app = Flask(__name__)
app.config.from_object('config')
g_maps_key = app.config['API_PASS_FRONT']


@app.route('/')
@app.route('/index')
def home():
    return render_template(
            "index.html",
            GOOGLE_API_KEY_MAPS=g_maps_key
        )


@app.route('/process', methods=['POST'])
def process():
    if request.method == 'POST':
        input_value = request.form['messageInput']
        result = grandPyWork(input_value, app)
        return jsonify(result)


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

« Views » vas donc gérer l'application utilisant Flask, en renvoyant les « routes ».

« index.html » sera donc la vue principal.

« process » sera la vue qui servira à faire les liaisons Back et Front par des requêtes « POST ».

Une part importante ce sont les tests et donc pour chaque fonction du programme, j'ai donc mis en place une batterie de tests. Un module de tests pour chaque module du programme.

```python
key = MAPS_API_KEY


def test_call_google_maps(monkeypatch):
    with open("tests/gmaps_data.json") as g_maps_data:
        results_test = json.load(g_maps_data)

    class MockResponse:
        def read(self):
            result_strings = json.dumps(results_test)
            result_bytes = result_strings.encode()
            return result_bytes

    def mock_g_maps(url):
        return MockResponse()

    monkeypatch.setattr(
      "GrandPyApp.interface_requests.call_google_maps_positionnement",
      mock_g_maps
      )

    place_id_test = results_test["results"][0]["place_id"]
    location_test = results_test["results"][0]["geometry"]["location"]
    adress_test = results_test["results"][0]["formatted_address"]

    assert call_google_maps_positionnement(
      key,
      "openclassrooms")[0] == place_id_test
    assert call_google_maps_positionnement(
      key,
      "openclassrooms")[1] == location_test
    assert call_google_maps_positionnement(
      key,
      "openclassrooms")[2] == adress_test
    assert call_google_maps_positionnement(
      key,
      "") == "Desolé je n'ai pas pu t'aider mon petit, peut-tu " + \
      "refaire ta demande autrement stp? Tu sais avec mon age..."
```

Sur « test_first_input_parseur » je test le parseur, notamment les réponses qui vont être envoyées vers les requêtes sue les API, ainsi que au cas où on rentre une « chaine vide » il retourne bien une « chaine vide »

Sur « test_requests » j'utilise donc une réponse de l'API, ou j'ai controlé l'envoi et enregistré le retour, ensuite, je vais utiliser « monkeypatch » afin de simuler la réponse, et ainsi la comparer les résultat avec les reponses obtenues avec une requete en direct.

Ensuite j'ai consulté pas mal de pages avec des caractéristiques similaires à celle que je voulais mettre en place (ici sur forme de dialogue type messagerie).

J'ai donc codé une page HTML qui va héberger l'interface de messagerie. Situé sur le dossier « templates » avec des liens avec la feuille de style.css et script.js situées dans le dossier « static ».

Dans le fichier « Js », j'ai donc la logique de construction des réponses qui seront envoyées à la vue « index.html » et récupérées du back end via AJAX.

```
function getMessageGrandPy(msg) {
  $.ajax({
    data : {messageInput : msg},
    type : 'POST',
    url : '/process',
    dataType: "json",
    success: function(data) {
      setTimeout(function() {
        mapGrandPyMessages(data.messages, data.position, data.tag);
      }, 2000);
    },
  });
};

var id_tags = Array();
function mapGrandPyMessages(messages, position, tag) {
  if(id_tags.includes(tag)){
    var message_ups = "Petit coquin, a faire des blagues a PaPy.., Cherche plus haut dans la conve
    grandPyMessage(message_ups);
  } else{
    id_tags.push(tag);
    grandPyMessage(messages[0]);
    grandPyMessage(messages[1]);
    grandPyMessage(messages[2]);
    $('<div class="message loading new"><figure class="avatar"><img src="../static/images/papy.gif
    updateScrollbar();
      setTimeout(function() {
      $('.message.loading').remove();
      $('<div class="message new"><figure class="avatar"><img src="../static/images/papy.gif" /></
      var elmt = document.getElementById('showMap_'+tag);
      var mq = window.matchMedia("screen and (min-width: 1024px");
      if (mq.matches) {
        elmt.style.height= "300px";
        elmt.style.width= "700px";
      } else {
        elmt.style.height= "200px";
```

Dans le fichier Js. J'ai donc des fonctions que me « servent » à gérer l'interface.

Je construis donc les messages que j'envoi, j'ajoute une fonction HTML escape afin de prévenir toute injection de code Html dans les inputs.

L'application web est type responsive, optimisé smartphones.

```
@media only screen and (max-device-width: 667px), screen and (max-width: 320px) {
  .avenue-messenger {
    margin: 0px;
    z-index: 2147483001 !important;
    width: 100% !important;
    height: 100% !important;
    max-height: none !important;
    top: 0 !important;
    left: 0 !important;
    right: 0 !important;
    bottom: 0 !important;
    border-radius: 0 !important;
    background: #fff;
  }
  .messages .message, .message-box .message-input {
    font-size: 12px;
  }

  body {
    background: none;
  }
  .footer {
    display: none;
  }

  .superPapy {
    display: none;
  }

  h3 {
    display: none;
  }
  .message-box .message-input {
    height: 18px;
  }
```

J'ai mis en place en « css » et sur « Js », (pour les besoins de l'affichage de la carte) des instructions afin de réduite la boite de dialogues et enlever les items, moins importants.