PROJET 7
CRÉEZ GRANDPY BOT
LE PAPY-ROBOT 

OUDOT EMMANUEL

DÉVELOPPEUR D'APPLICATION - PYTHON

LE PROJET

- Le but du projet était de créer un robot qui vous répondrait comme un grand-père ! Si vous lui demandez l'adresse d'un lieu, il vous la donnera, mais agrémentée d'un long récit très intéressant.
- Proposer une interface sous la forme d’une page web.
- La réponse doit contenir le nom du lieu, l’adresse, sa position sur une carte, une description, un lien vers sa page Wikipédia, le tout assemblé dans une réponse qui semble humaine.

CONTRAINTES

- Une approche Test Driven Development (TDD).
- Des tests utilisant des mocks pour les API.
- Utilisation d'AJAX pour l'envoi des questions et l'affichage des réponses.
- Une interface responsive.
- Le code intégralement écrit en anglais.

TEST DRIVEN DEVELOPMENT

Pour concevoir cette application en TDD, j'ai utilisé le framework Pytest, il permet d'écrire facilement des tests unitaires, et permet de s'appuyer sur les mocks pour simuler les fonctions appelées à l'intérieur de la fonction testée. Au début du projet, j'ai réalisé une première liste de tests simples qui sera modifié par la suite :

- test if parser return simplified message
- test if google function return a location
- test if wikipedia function return a description
- test if grandpy answer contain all data and introductions

Puis à chaque nouvelle fonction je codais le test correspondant

INTERFACE WEB

Pour gérer l'application j'ai utilisé Flask, un framework d'application Web WSGI. L'application Grandpy étant plutôt simple, seules deux routes ont été nécessaires. Une pour le premier affichage de la page d'accueil, la seconde va utiliser la méthode POST pour récupérer le message puis générer et renvoyer le message de Grandpy.

Pour rendre le site dynamique, ne pas recharger la page à chaque message de l'utilisateur et pouvoir faire communiquer le frontend et le backend je me suis servi de JavaScript, jQuery et AJAX.

Pour la mise en forme et le rendu responsive, j'ai construit la page html en utilisant les classes de Bootstrap. Cela a aussi permis l'ajout d'éléments comme les boutons et border spinner.

LA RECHERCHE DES DONNÉES

1. Simplifier le message de l'utilisateur
2. Les API de Google Maps
3. L'API de Media Wiki

LA RECHERCHE DES DONNÉES

1. Vider la question des termes inutiles: **LE PARSER**

```python
class Parser:
    def __init__(self):
        pass

    def clean_the_message(self, message):...

    def replace_accents(self, message):
        """replace the accented letters with the unaccented letter."""
        for (char, accented_chars) in ACCENTS_DICT.items():
            for accented_char in accented_chars:
                message = message.replace(accented_char, char)
        return message

    def remove_special_characters(self, message):
        """remove all characters that are not simple letters and numbers."""
        message = re.sub("[^a-zA-Z0-9]", " ", message)
        return message
```

```python
ACCENTS_DICT = {
    'a': ['à', 'ã', 'á', 'â', 'ä'],
    'e': ['é', 'è', 'ê', 'ë'],
    'i': ['î', 'ï', 'í', 'ì'],
    'o': ['ô', 'ö', 'ó', 'ò', 'õ'],
    'u': ['ù', 'ü', 'û', 'ú'],
    'c': ['ç'],
    'n': ['ñ'],
}
```

La Classe Parser est composée de 5 méthodes :

1. replace_accents()

   Qui remplace les lettres accentuées par leur lettre non accentuée.

2. remove_special_characters()

   Qui ne conserve dans le message que les lettres de a à z et les chiffres de 0 à 9.

LA RECHERCHE DES DONNÉES

1. Vider la question des termes inutiles: **LE PARSER**

```
26 ›    def remove_special_characters(self, message):⋯

30

31      def remove_short_words(self, message):
32          """remove words smaller than 3 letters, keep numbers."""
33          words_list = message.split()
34          message = ''
35          for word in words_list:
36              if word.isdigit():
37                  message = message + ' ' + word
38              else:
39                  if len(word) > 2:
40                      message = message + ' ' + word
41          message = message[1:]
42          return message

43

44      def remove_stopwords(self, message):
45          """removes words that are in the stopwords list."""
46          words_list = message.split()
47          message = ''
48          for word in words_list:
49              if word not in STOPWORDS:
50                  message = message + ' ' + word
51          message = message[1:]
52          return message
```

3. remove_short_words()

   Qui supprime les mots de moins de 3 lettres.

4. remove_stopwords()

   Qui supprime du message tout les mots présent dans la liste de stop word.

5. clean_the_message()

   Qui utilise toutes les méthodes précédentes dans l'ordre pour retourner le message simplifié.

.

LA RECHERCHE DES DONNÉES

2. Les API de Google Maps

```
class GoogleGeocoding:
    def __init__(self):
        pass

    def find_location_in_message(self, message):
        """get from googlemaps geocoding API address and coordinates."""
        google_data = {'status': 'problem'}
        message = message.replace(" ", "+")
        params['address'] = message
        params['key'] = api_key
        try:
            server_response = requests.get(url=url, params=params)
            data = server_response.json()
        except requests.ConnectionError:
            pass
        if data["status"] == "OK":
            data = data["results"][0]
            google_data = {
                "address": data["formatted_address"],
                "lat": round(data["geometry"]["location"]["lat"], 6),
                "lng": round(data["geometry"]["location"]["lng"], 6),
            }
            if all(google_data.values()):
                google_data['status'] = 'ok'
        return google_data
```

La méthode find_location_in_message() de la classe GoogleGeocoding() interroge l'API :

Google geocoding.

Elle retourne :

- les coordonnées
- l'adresse du lieu

LA RECHERCHE DES DONNÉES

2. Les API de Google Maps

```
class GoogleMapsStatic:
    def __init__(self):
        pass

    def get_map_url_for(self, google_data):
        """get the url of a static map with google maps static api."""
        lat = str(google_data['lat'])
        lng = str(google_data['lng'])
        google_map_url = ''
        params["markers"] = lat + ',' + lng
        params['key'] = private_key
        try:
            server_response = requests.get(url=url, params=params)
        except requests.ConnectionError:
            pass
        if server_response.url:
            google_map_url = server_response.url
            google_map_url = google_map_url.replace(private_key, public_key)
        return google_map_url

```

La méthode get_map_url_for () de la calsse GoogleMapsStatic() interroge l'API:

Google Maps Static.

Elle retourne :

- l'URL de la carte situant l'endroit

LA RECHERCHE DES DONNÉES

3. L'API de Media Wiki

```python
class WikipediaGeosearch:
    def __init__(self):
        pass

    def get_data(self, google_geocoding_data):
        """get from wikipedia API name (title), page_id & url page."""
        lat = str(google_geocoding_data['lat'])
        lng = str(google_geocoding_data['lng'])
        wikipedia_data = {'status': 'problem'}
        params["gscoord"] = lat + '|' + lng
        try:
            server_response = requests.get(url=url, params=params)
            data = server_response.json()
        except requests.ConnectionError:
            pass
        if data['query']['geosearch']:
            data = data['query']['geosearch'][0]
            wikipedia_data = {
                'name': data['title'],
                'page_id': data['pageid'],
                'wikipedia_url': (page_id_url + str(data['pageid'])),
            }
            if all(wikipedia_data.values()):
                wikipedia_data['status'] = 'ok'
        return wikipedia_data
```

La méthode get_data () de la classe WikipediaGeosearch(), interroge l'API :

MediaWiki.

Elle retourne :

- le nom
- l'identifiant de la page Wikipédia
- l'URL de la page Wikipédia

LA RECHERCHE DES DONNÉES

3. L'API de Media Wiki

```
class WikipediaDescription:
    def __init__(self):
        pass

    def get_data(self, wikipedia_data):
        """get from wikipedia API short description."""
        description = ''
        page_id = wikipedia_data['page_id']
        params['pageids'] = page_id
        try:
            server_response = requests.get(url=url, params=params)
            data = server_response.json()
        except requests.ConnectionError:
            pass
        if data['query']['pages'][str(page_id)]['extract']:
            description = data['query']['pages'][str(page_id)]['extract']
        return description
```

La méthode get_data () de la classe WikipediaDescription() interroge l'API :

MediaWiki.

Elle retourne :

- la description du lieu

LA RÉPONSE DE GRANDPY

1. Des message en cas de problème.
2. Un description plus courte si besoin.
3. Des messages d'introduction aléatoires.
4. Regrouper toutes les données.

LA RÉPONSE DE GRANDPY

1. Des message en cas de problème.

```python
class Answer:
    def __init__(self):
        pass

    def stops_because(self, problem):
        """return an answer to each problem during the search for the answer
        data."""
        answers_list = globals()[ANSWER_TO_PROBLEM[problem]]
        answer = {
            'status': 'problem',
            'grandpyMessage': random.choice(answers_list),
        }
        return answer
```

```python
ANSWER_TO_PROBLEM = {
    'blank_new_user_message': 'ANWS_BLANK',
    'nonallowed_characters': 'ANWS_NONALLOWED',
    'cant_find_location': 'ANWS_CANT_FIND',
}
ANWS_BLANK = [
    "Parle plus fort, je ne t'entends pas.",
    "Essaye avec des mots, je ne lis pas dans les pensées.",
    "Il ne manque pas quelque chose ?",
    "Attend, je mets de lunettes mais je ne vois rien.",
]
ANWS_CANT_FIND = [
    "Ça me dit quelque chose mais pas moyen de me souvenir",
    "Tu es certain que cet endroit existe ? Jamais entendu parler.",
]
ANWS_NONALLOWED = [
    "Petit chenapan ! Tu ne devrais pas utiliser ces caractères !",
    "Allé, on va dire que tu n'as pas fait exprès d'utiliser ces caractères…",
]
```

Si l'utilisateur envoi un message vide, si son message contient des caractère susceptible de provoquer un fonctionnement non attendus ou si la recherche de données n'est pas complète, GrandPy va répondre en choisissant aléatoirement dans les listes de réponse prévues pour chaque problème.

LA RÉPONSE DE GRANDPY

2. Un description plus courte si besoin

```
class Answer:
    def __init__(self):
        pass

    def stops_because(self, problem):

    def builds_with_data(self, google_data, wikipedia_data):
        """
        build grandpy answer with location data.
        """
        description = wikipedia_data['description']
        description_first_part = description
        description_collapsible_part = ''
        """if the description is too long, split the message"""
        if len(description) > INTRO_MAX_SIZE:
            description_first_part = description[: INTRO_MAX_SIZE - 1]
            description_collapsible_part = description[INTRO_MAX_SIZE - 1:]
        """includes all parts of the answer"""
        answer = {
```

Les descriptions « courtes » de Wikipédia sont parfois très longues et prennent une place importante sur la réponse de GrandPy et cela nuit à la lisibilité surtout sur les petits écrans.

Dans ce cas le message est tronqué et un bouton permet d'afficher ou masquer la partie supplémentaire.

LA RÉPONSE DE GRANDPY

3. Des messages d'introduction aléatoires.

```
        answer = {
            'intro_name': random.choice(INTRO_NAME),
            'name': wikipedia_data['name'],
            'intro_address': random.choice(INTRO_ADDRESS),
            'address': google_data['address'],
            'intro_map': random.choice(INTRO_MAP),
            'map_url': google_data['map_url'],
            'intro_description': random.choice(INTRO_DESCRIPTION),
            'description_first_part': description_first_part,
            'description_collapsible_part': description_collapsible_part,
            'wikipedia_url': wikipedia_data['wikipedia_url'],
        }
        answer = {'status': 'ok', 'grandpyMessage': answer}
        return answer
```

```
INTRO_NAME = [
    "<br>OK, Tu veux des infos sur cet endroit:<br>",
    "<br>Je vois, je connais bien cet endroit :<br>",
]
INTRO_ADDRESS = [
    "<br>Alors je te donne déjà l'adresse :<br>",
    "<br>Cela se situe à l'adresse suivante :<br>",
]
INTRO_MAP = [
    "<br>Si tu ne situes pas, regarde la carte :<br>",
    "<br>Je peux te montrer sur une carte :<br>",
    "<br>Jette un œil sur la carte pour te repérer:<br>",
]
INTRO_DESCRIPTION = [
    "<br>J'avais discuté avec un riverain avant la visite, sais-tu que ",
    "<br>Quand j'y étais le guide m'avait expliqué que ",
    "<br>C'est un super endroit à l'école on m'avais dit que ",
    "<br>Mamie y est allée elle m'a raconté que ",
]
```

Chaque donnée contenue dans la réponse de GrandPy est précédée d'une introduction choisie aléatoirement dans des listes de réponse.

LA RÉPONSE DE GRANDPY

4. Regrouper toutes les données.

```python
class Answer:
    def __init__(self):
        pass

    def stops_because(self, problem):

    def builds_with_data(self, google_data, wikipedia_data):
        """
        build grandpy answer with location data.
        """
        description = wikipedia_data['description']
        description_first_part = description
        description_collapsible_part = ''
        """if the description is too long, split the message"""
        if len(description) > INTRO_MAX_SIZE:
        """includes all parts of the answer"""
        answer = {
            'intro_name': random.choice(INTRO_NAME),
            'name': wikipedia_data['name'],
            'intro_address': random.choice(INTRO_ADDRESS),
            'address': google_data['address'],
            'intro_map': random.choice(INTRO_MAP),
            'map_url': google_data['map_url'],
            'intro_description': random.choice(INTRO_DESCRIPTION),
            'description_first_part': description_first_part,
            'description_collapsible_part': description_collapsible_part,
            'wikipedia_url': wikipedia_data['wikipedia_url'],
        }
        answer = {'status': 'ok', 'grandpyMessage': answer}
        return answer
```

Toutes les données sont enfin regroupées dans un dictionnaire et retourné pour l'affichage.

LES TESTS

1. Test simple.
2. Test avec Mock.
3. Couverture des tests.

LES TESTS

1. Test simple

```
from grandpy.data_search.parser import Parser

"""
tests the simple functions of the parser
"""
def test_if_replace_accents_function_work():
    assert Parser.replace_accents(
                Parser, 'àãáâéèêëîïùüûôö') == ('aaaaeeeeiiuuuoo')
```

Un exemple des tests simples utilisé pour une méthode de la classe Parser(),

Ici le test serra réussit si la chaine de caractères: àãáâéèêëîïùüûôö

donnée en paramètre à la méthode: replace_accents()

retourne bien la chaine de caractères: aaaaeeeeiiuuuoo

LES TESTS

2. Test avec Mock

```
from grandpy.data_search.wikipedia_description import WikipediaDescription
from .constants import ANSW_REQUEST_GET_WIKIPEDIA_DESCRIPTION

def test_if_get_data_function_work(monkeypatch):
    wikipedia_data = {'page_id': 999999}

    class MockRequestsGet:
        def __init__(self, url, params):
            self.status_code = 200
        def json(self):
            return ANSW_REQUEST_GET_WIKIPEDIA_DESCRIPTION

    monkeypatch.setattr('requests.get', MockRequestsGet)
    assert WikipediaDescription.get_data(
        WikipediaDescription, wikipedia_data) == ('test description')

```

Sur ce test la méthode *get_data()* de la classe *WikipediaGeosearch()* utlise *requests.get*.

Donc quand nous testons *get_data()* le résultat est soumis à la réponse de *requests.get* ce qui n'est plus un test unitaire. Il faut donc utiliser un Mock pour imiter le comportement de *requests.get* et bien tester que la méthode *get_data()*.

C'est le rôle de la classe *MockRequestsGet()* qui remplace *requests.get* dans le monkeypatch.

LES TESTS

3. Couverture des tests

```
tests/test_answer.py::test_if_stops_because_function_work PASSED                    [  9%]
tests/test_answer.py::test_if_builds_with_data_function_work PASSED                 [ 18%]
tests/test_google_geocoding.py::test_if_find_location_in_message_function_work PASSED [ 27%]
tests/test_google_maps_static.py::test_if_get_map_url_for_function_work PASSED      [ 36%]
tests/test_parser.py::test_if_replace_accents_function_work PASSED                  [ 45%]
tests/test_parser.py::test_if_remove_special_characters_function_work PASSED        [ 54%]
tests/test_parser.py::test_if_remove_short_words_function_work PASSED               [ 63%]
tests/test_parser.py::test_if_remove_stopwords_function_work PASSED                 [ 72%]
tests/test_parser.py::test_if_clean_the_message_function_work PASSED                [ 81%]
tests/test_wikipedia_description.py::test_if_get_data_function_work PASSED          [ 90%]
tests/test_wikipedia_geosearch.py::test_if_get_data_function_work PASSED            [100%]

================================== 11 passed in 0.37s ==================================

(P7venv) C:\Users\manuo\Google Drive\Openclassrooms\P7\main>coverage report -m
Name                                             Stmts   Miss  Cover   Missing
------------------------------------------------------------------------------
grandpy\__init__.py                                  0      0   100%
grandpy\data_search\__init__.py                      0      0   100%
grandpy\data_search\api_config\settings.py          17      7    59%   13-23
grandpy\data_search\constants.py                    10      0   100%
grandpy\data_search\google_geocoding.py              23      3    87%   10, 21-22
grandpy\data_search\google_maps_static.py            21      3    86%   11, 22-23
grandpy\data_search\parser.py                        38      2    95%   8, 37
grandpy\data_search\wikipedia_description.py         18      3    83%   9, 19-20
grandpy\data_search\wikipedia_geosearch.py           23      3    87%   10, 21-22
grandpy\grandpy_answers\__init__.py                  0      0   100%
grandpy\grandpy_answers\answer.py                   19      3    84%   10, 31-32
grandpy\grandpy_answers\constants.py                 9      0   100%
tests\__init__.py                                    0      0   100%
tests\constants.py                                   5      0   100%
tests\test_answer.py                                11      0   100%
tests\test_google_geocoding.py                      11      0   100%
tests\test_google_maps_static.py                     9      0   100%
tests\test_parser.py                                23      0   100%
tests\test_wikipedia_description.py                 11      0   100%
tests\test_wikipedia_geosearch.py                   11      0   100%
------------------------------------------------------------------------------
TOTAL                                              259     24    91%
```

Tout les tests réalisés sont au vert ☺

Et le taux de couverture des test est de 91%

En analysant les parties non testées, il s'agit principalement du fichier de configuration, de l'initialisation des classes et la gestion des exceptions (try/except).

DIFFICULTÉS ET AXES D'AMÉLIORATIONS :

Flask n'a pas posé de problème particulier sur ce projet et les tests ne se sont compliqués qu'au moment des mock de la classe request.get, j'ai mis du temp à comprendre qu'il fallait une classe pour mocker une autre classe.

La partie requêtes avec les API à été plus compliquée, celles de googles sont plutôt bien documentés avec beaucoup d'exemple mais celles de media-wiki beaucoup moins et il a été difficile de trouver les paramètres à appliquer pour obtenir les éléments voulut, j'ai dû faire beaucoup de recherches hors documentation pour trouver.

Bootstrap est un très bel outil très complet et facile à prendre en main, j'ai réussi à n'utiliser aucun css personnel, tout était disponible avec bootstrap, et son approche « mobile first » on permis de rendre l'interface responsive plutôt facilement.

JavaScript, jQuery et ajax sont peu utilisé sur ce projet mais c'est l'envoi en post via Ajax qui m'a posé des problèmes. Je n'avais pas saisie qu'il fallait une route flask dédiée à cet envoi.

Pour les améliorations, je pense que le visuel est sommaire et pourrait être amélioré et le système de recherche par les coordonnées pose parfois des soucis de cohérence entre le lieu trouvé sur google et le lieu trouvé sur Wikipédia, il doit y avoir un moyen diffèrent de rechercher.