Lucas NGUYEN
Tom CALVO
Dorian JOSSERAND

WhatsAPI

1. A quoi va servir l'API?

WhatsAPI est une API permettant de faciliter la réalisation d'une application de messagerie. L'API permettra notamment :

- ☐ Lister les utilisateurs de WhatsAPI.
- ☐ Ajouter un utilisateur de WhatsAPI.
- ☐ Afficher toutes les informations relatives à un utilisateur précis.
- ☐ Voir nos amis
- ☐ Ajouter un ami sur WhatsAPI
- ☐ Supprimer un ami
- ☐ Ajouter une bio
- ☐ Modifier sa bio

2. Schéma relationnel de la BDD Description de la structure de la base de données.

BDD :

- UTILISATEUR(id_utilisateur: INT, pseudo_utilisateur: TEXT,bio : TEXT)
- AMITIE(#util1:TEXT,#util2:TEXT)

3. Création de la BDD Ecriture du fichier .sql permettant d'initialiser la BDD.

```
DROP TABLE IF EXISTS UTILISATEUR;
DROP TABLE IF EXISTS AMITIE;

CREATE TABLE UTILISATEUR(
    id_utilisateur INTEGER PRIMARY KEY AUTOINCREMENT,
    pseudo_utilisateur varchar(20) UNIQUE,
    bio varchar(150)
);

CREATE TABLE AMITIE(
      util1 varchar(20),
      util2 varchar(20),
      CONSTRAINT AMITIE PRIMARY KEY (util1,util2),
      CONSTRAINT AMITIE FOREIGN KEY(util1) REFERENCES UTILISATEUR
(pseudo_utilisateur)
      CONSTRAINT AMITIE FOREIGN KEY(util2) REFERENCES UTILISATEUR
(pseudo_utilisateur)
);
```

4. Documentation de l'API Ecriture des spécifications des **nouvelles** fonctions de l'API

```
NAME
       WhatsApi

FONCTIONS
        ajouter bio()
        modifier_bio()

DESCRIPTION

    ajouter_bio(pseudo_utilisateur)
        Cette fonction ajoute une bio a un utilisateur donné pseudo_utilisateur.

        INPUT
            pseudo_utilisateur type STR

        OUTPUT

            Type Dict {
                status :
                data : []
                }

            status :
                0 -> None
                1 -> 'INPUT Not registered in UTILISATEUR'




    modifier bio(pseudo utilisateur)
        Cette fonction permet de modifier une bio déjà écrite de pseudo utilisateurs.

        INPUT
            pseudo_utilisateur type STR

        OUTPUT

            Type Dict {
                status :
                data : []
                }

            status :
                0 -> None
                1 -> 'INPUT Not registered in UTILISATEUR'
                2 -> 'INPUT There is no bio'


AUTHOR
       Written by Tom CALVO, Dorian JOSSERAND and Lucas NGUYEN.


COPYRIGHT
       Copyright (c) 2022 WhatsAPI
       This is free software: you are free to change and redistribute it.  There is NO WARRANTY, to the extent permitted by law.
```

5. Écriture des tests (assert ou testmod) qui vérifient les spécifications. Il peut être utile de créer un fichier test.sql pour remplir la BDD avec des valeurs dédiées aux tests.

```python
if __name__ == "__main__":
    DB_FILE = 'BDDtest.db'
    execution_SQL("BDD.sql")
    execution_SQL("BDDtest2.sql")

    #teste de list_util() avec une bdd vide
    execution_SQL('BDD.sql')
    assert list_util() == {"status" : 0, "data" : []}
    #teste de list_util() avec une bdd non vide
    execution_SQL('BDDtest2.sql')
    assert list_util() == {"status" : 0, "data" : [{"id_utilisateur" : 1, "pseudo_utilisateur" : 'Tom.clv'},{"id_utilisateur": 2, "pseudo_utilisateur":
'Dorian.jsr'},{"id_utilisateur": 3, "pseudo_utilisateur": 'Nyn.luk'}]}

    #teste de ajouter_util()
    assert ajouter_util('Guilbert') == {"status" : 0, "data" : []}
    assert ajouter_util(1880) == {"status" : 1, "data" : ['INPUT type not STR']}
    assert ajouter_util(' ') == {"status" : 2, "data" : ['INPUT Length not between 5, 20']}
    assert ajouter_util('Le.Boulanger.Qui.Fait.Du.Pain ') == {"status" : 2, "data" : ['INPUT Length not between 5, 20']}
    assert ajouter_util('Tom.clv') == {"status" : 3, "data" : ['INPUT already in database']}
    #teste de ajouter_util() (état de la bdd après les testes)
    assert list_util() == {"status" : 0, "data" : [{"id_utilisateur" : 1, "pseudo_utilisateur" : 'Tom.clv'},{"id_utilisateur": 2, "pseudo_utilisateur":
'Dorian.jsr'},{"id_utilisateur" : 3, "pseudo_utilisateur" : 'Nyn.luk'},{"id_utilisateur" : 4, "pseudo_utilisateur" : 'Guilbert'}]}

    #teste de info_util()
    assert info_util('Dorian.jsr') == {"status" : 0, "data": [{"id_utilisateur": 2, "pseudo_utilisateur": 'Dorian.jsr', 'bio': ""}]}
    assert info_util(1880) == {"status" : 1, "data" : ['INPUT type not STR']}
    assert info_util('Le.Boulanger.Qui.Fait.Du.Pain') == {"status" : 2, "data" : ['INPUT Length not between 5, 20']}
    assert info_util(' ') == {"status" : 2, "data" : ['INPUT Length not between 5, 20']}
    assert info_util('util.inexistant') == {"status" : 3, "data": ['INPUT Not in database']}

    #Teste de voir_amis()
    execution_SQL("BDDtest3.sql")
    assert voir_amis("Tom.clv") == {'status' : 0, 'data' : ['Dorian.jsr']}
    assert voir_amis('Dorian.jsr') == {'status' : 0, 'data' : ["Tom.clv"]}
    assert voir_amis(5) == {'status' : 1, 'data' : ['INPUT Type not STR']}
    assert voir_amis("123456789101112131415161718192O") == {'status' : 2, 'data' : ['INPUT Lenght not between 5, 20']}
    assert voir_amis("123") == {'status' : 2, 'data' : ['INPUT Lenght not between 5, 20']}
    assert voir_amis("Josseline") == {'status' : 3, 'data' : ['INPUT Not Registered in UTILISATEUR']}

    #test de ajouter_amis()
    assert ajouter_amis('Tom.clv','Nyn.luk') == {"status" : 0, "data": []}
    assert ajouter_amis('Dorian.jsr',1880) == {"status" : 1, "data" : ['INPUT type not STR']}
    assert ajouter_amis(1880,'Dorian.jsr') == {"status" : 1, "data" : ['INPUT type not STR']}
    assert ajouter_amis('Tom.clv','Le.Boulanger.Qui.Fait.Du.Pain') == {"status" : 2, "data" : ['INPUT Length not between 5, 20']}
    assert ajouter_amis('Le.Boulanger.Qui.Fait.Du.Pain','Tom.clv') == {"status" : 2, "data" : ['INPUT Length not between 5, 20']}
    assert ajouter_amis('Nyn.luk',' ') == {"status" : 2, "data" : ['INPUT Length not between 5, 20']}
    assert ajouter_amis(' ','Nyn.luk') == {"status" : 2, "data" : ['INPUT Length not between 5, 20']}
    assert ajouter_amis('Tom.clv','Xx_Michelle_xX') == {"status" : 3, "data": ['INPUT Not register in UTILISATEUR']}
    assert ajouter_amis('Xx_Michelle_xX','Tom.clv') == {"status" : 3, "data": ['INPUT Not register in UTILISATEUR']}
    assert ajouter_amis('Tom.clv','Dorian.jsr') == {"status" : 4, "data": ['INPUT already register as friend']}
    assert ajouter_amis('Tom.clv','Tom.clv') == {"status" : 5, "data": ['INPUT are the same']}
    #Vérification de l'éxécution
    assert voir_amis('Tom.clv') == {"status" : 0, "data": ['Dorian.jsr', 'Nyn.luk']}

    #Teste de supprimer_amis()
    assert supprimer_amis('Tom.clv','Dorian.jsr') == {"status" : 0, "data": []}
    assert supprimer_amis('Dorian.jsr',1880) == {"status" : 1, "data" : ['INPUT type not STR']}
    assert supprimer_amis(1880,'Dorian.jsr') == {"status" : 1, "data" : ['INPUT type not STR']}
    assert supprimer_amis('Tom.clv','Le.Boulanger.Qui.Fait.Du.Pain') == {"status" : 2, "data" : ['INPUT Length not between 5, 20']}
    assert supprimer_amis('Le.Boulanger.Qui.Fait.Du.Pain','Tom.clv') == {"status" : 2, "data" : ['INPUT Length not between 5, 20']}
    assert supprimer_amis('Nyn.luk',' ') == {"status" : 2, "data" : ['INPUT Length not between 5, 20']}
    assert supprimer_amis(' ','Nyn.luk') == {"status" : 2, "data" : ['INPUT Length not between 5, 20']}
    assert supprimer_amis('Dorian.jsr','Nyn.luk') == {"status" : 3, "data" : ['INPUT Not in friendlist']}
    #Vérification de l'éxécution
    assert voir_amis('Tom.clv') == {"status" : 0, "data": ['Nyn.luk']}

    #test de ajouter_bio
    assert ajouter_bio('Tom.clv', 'Ego similis valde conditus litterarum stews, opera honestatis, ubi quaedam flax sensus reponit buxom salutem classical eras.') ==
{"status" : 0, "data": []}
    assert ajouter_bio("Josseline", "j'aime la cuisine") == {'status' : 1, 'data' : ['INPUT Not Registered in UTILISATEUR']}
    assert ajouter_bio('Tom.clv', 'Ego similis valde conditus litterarum stews, opera honestatis, ubi quaedam flax sensus reponit buxom salutem classical eras. Ego aetas
mea. ubi quaedam.') == {'status': 2, 'data': ['INPUT Length superior at 150']}
    #Vérification de l'éxécution
    assert info_util('Tom.clv') == (({'status': 0, 'data': [{'id_utilisateur': 1, 'pseudo_utilisateur': 'Tom.clv', 'bio': 'Ego similis valde conditus litterarum stews,
opera honestatis, ubi quaedam flax sensus reponit buxom salutem classical eras.'}]}))

    #test modifier_bio
    assert modifier_bio('Tom.clv', 'voici une super bio') == {'status' : 0, 'data' : []}
    assert modifier_bio('Jean_claude', 'voici une bio de test') == {'status' : 1, 'data' : ['INPUT Not Registered in UTILISATEUR']}
    assert modifier_bio('Nyn.luk', 'voici une bio de test') == {'status' : 2, 'data' : ['INPUT There is no bio']}
    assert modifier_bio('Dorian.jsr', 'voici une bio de test') == {'status' :2, 'data' : ['INPUT There is no bio']}
    #Vérification de l'éxécution
    assert info_util('Tom.clv') == (({'status': 0, 'data': [{'id_utilisateur': 1, 'pseudo_utilisateur': 'Tom.clv', 'bio': 'voici une super bio'}]}))

    os.remove('BDDtest.db')
    DB_FILE = 'BDD.db'
```

BDD.sql

```sql
DROP TABLE IF EXISTS AMITIE;
DROP TABLE IF EXISTS UTILISATEUR;


CREATE TABLE UTILISATEUR(
    id_utilisateur INTEGER PRIMARY KEY AUTOINCREMENT,
    pseudo_utilisateur varchar(20) UNIQUE,
    bio varchar(150)
);

CREATE TABLE AMITIE(
        util1 varchar(20),
        util2 varchar(20),
        CONSTRAINT AMITIE PRIMARY KEY (util1,util2),
        CONSTRAINT AMITIE FOREIGN KEY(util1) REFERENCES UTILISATEUR (pseudo_utilisateur)
        CONSTRAINT AMITIE FOREIGN KEY(util2) REFERENCES UTILISATEUR (pseudo_utilisateur)
);
```

BDDtest1.sql :

```sql
DELETE from AMITIE;
DELETE from UTILISATEUR;
```

BDDtest2.sql :

```sql
INSERT INTO UTILISATEUR
(id_utilisateur, pseudo_utilisateur, bio)
VALUES
(1,'Tom.clv',""),
(2,'Dorian.jsr',""),
(3,'Nyn.luk',"")
```

BDDtest3.sql :

```sql
INSERT INTO AMITIE
(util1 , util2)
VALUES
('Dorian.jsr','Tom.clv'),
('Tom.clv','Dorian.jsr');
```

6. Codage de l'API Codage des fonctions que vous avez spécifiées

- [ ] ajouter bio() Tom Terminé
- [ ] modifier_bio() Lucas Terminé
- [ ] () Dorian Terminé

7. Passage des tests Vérification que les tests passent

☐ Terminé

8. Intégration Rassemblement du travail des membres du groupe dans le projet principal.

☐ Terminé