

Fiche PP 1 : Interface graphique sous Python

Kern Louis

BTS Services informatiques aux organisations Session 2020-2021

E6 – Parcours de professionnalisation
Coefficient

Description d'une situation professionnelle

Épreuve ponctuelle ~~Contrôle en cours de formation~~

~~Parcours SISR~~ Parcours SLAM

NOM et prénom du candidat : KERN Louis **N° candidat :**

Contexte de la situation professionnelle

Adhelios Software, entreprise d'éditions de logiciels pour entreprises basé à Strasbourg, cherche à expérimenter avec les interfaces utilisateurs sous Python, afin de potentiellement réutiliser ces recherches pour de futures applications.

Intitulé de la situation professionnelle

Développement d'une interface graphique en Python.

Période de réalisation : 2^{ème} semestre 2020 **Lieu :** Strasbourg

Modalité : ~~Equipe~~ Individuelle

Conditions de réalisation

L'interface doit être développée sous Python, et doit être capable d'interagir avec un serveur Rest distant, et doit pouvoir traiter des données envoyée en format json.

Productions associées

Le travail est entreposé sur un dépôt Git sur le serveur de l'entreprise
Interaction avec un serveur REST distant qui est managé par un membre du personnel d'Adhélios Software

Modalité d'accès aux productions :

Intranet de l'entreprise, et par copie de l'application sur appareil directement

Fiche PP 1 : Interface graphique sous Python

Kern Louis

Introduction :

Mon objectif premier était de déterminer quel environnement de développement et quelle librairie Python était le mieux adapté pour concevoir l'interface graphique. J'ai choisi Pycharm pour développer l'application, car c'est un IDE facile d'utilisation, qui comprend une prise en charge des dépôts GIT, peut installer facilement des librairies, et c'est aussi un IDE avec lequel je suis familiarisé.

Pour la librairie, j'ai utilisé la librairie Pyside2 de Qt, qui contient tous les modules nécessaires à la conception de l'interface graphique, y compris un logiciel de conception d'interfaces graphiques intégré à la librairie. De plus, Pyside2 s'installe facilement sous Windows et Linux.

Le projet n'est malheureusement pas fonctionnel étant donné que l'ajout de nouvelles fonctionnalités au fur et à mesure du stage a nécessité des changements perpétuels dans le code, et le résultat final ne fonctionne pas réellement par manque de temps pour le débogage. Mais nous pouvons proposer un aperçu du fonctionnement attendu du programme.

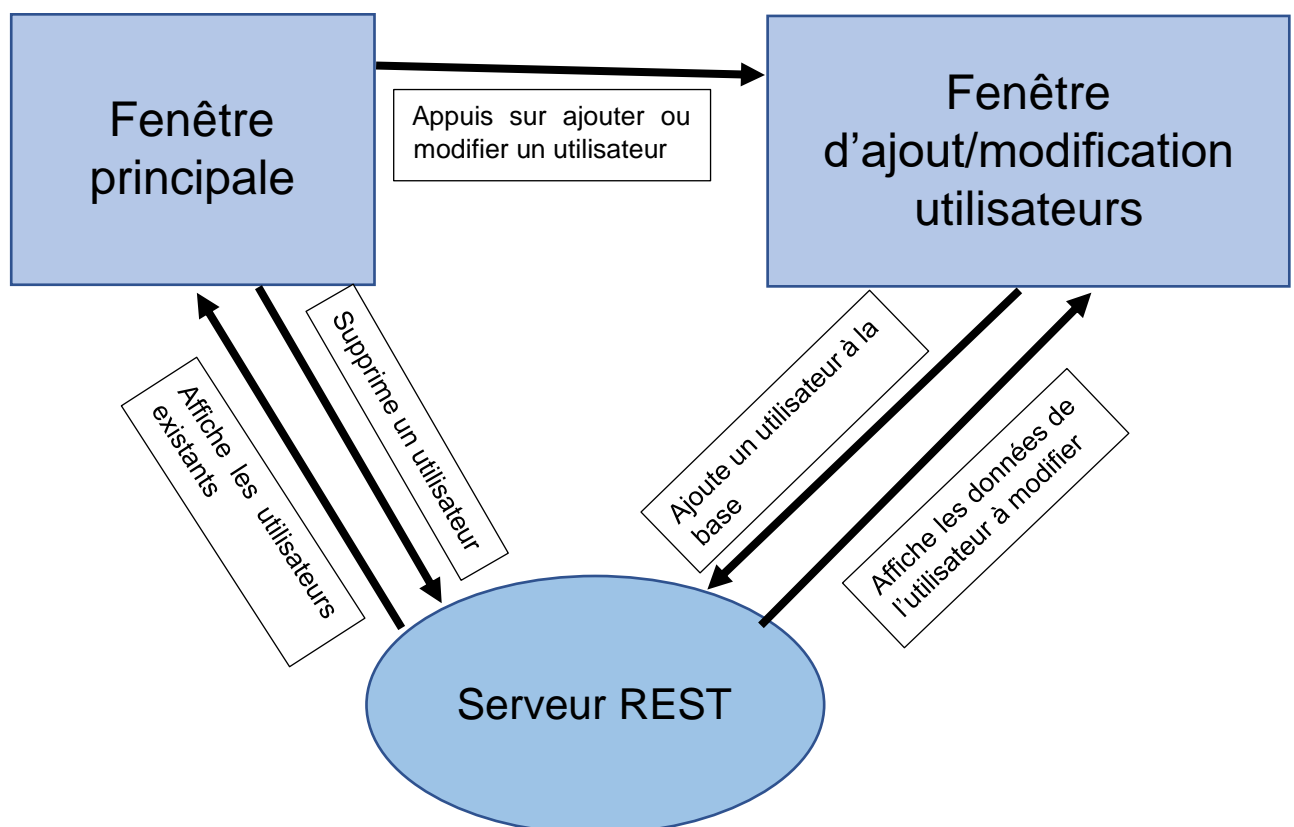
Condition de réalisation :

Environnement de développement

Logiciels : Pycharms, Qt Designer, Notepad++

Langages : Python

Organisation de l'interface :

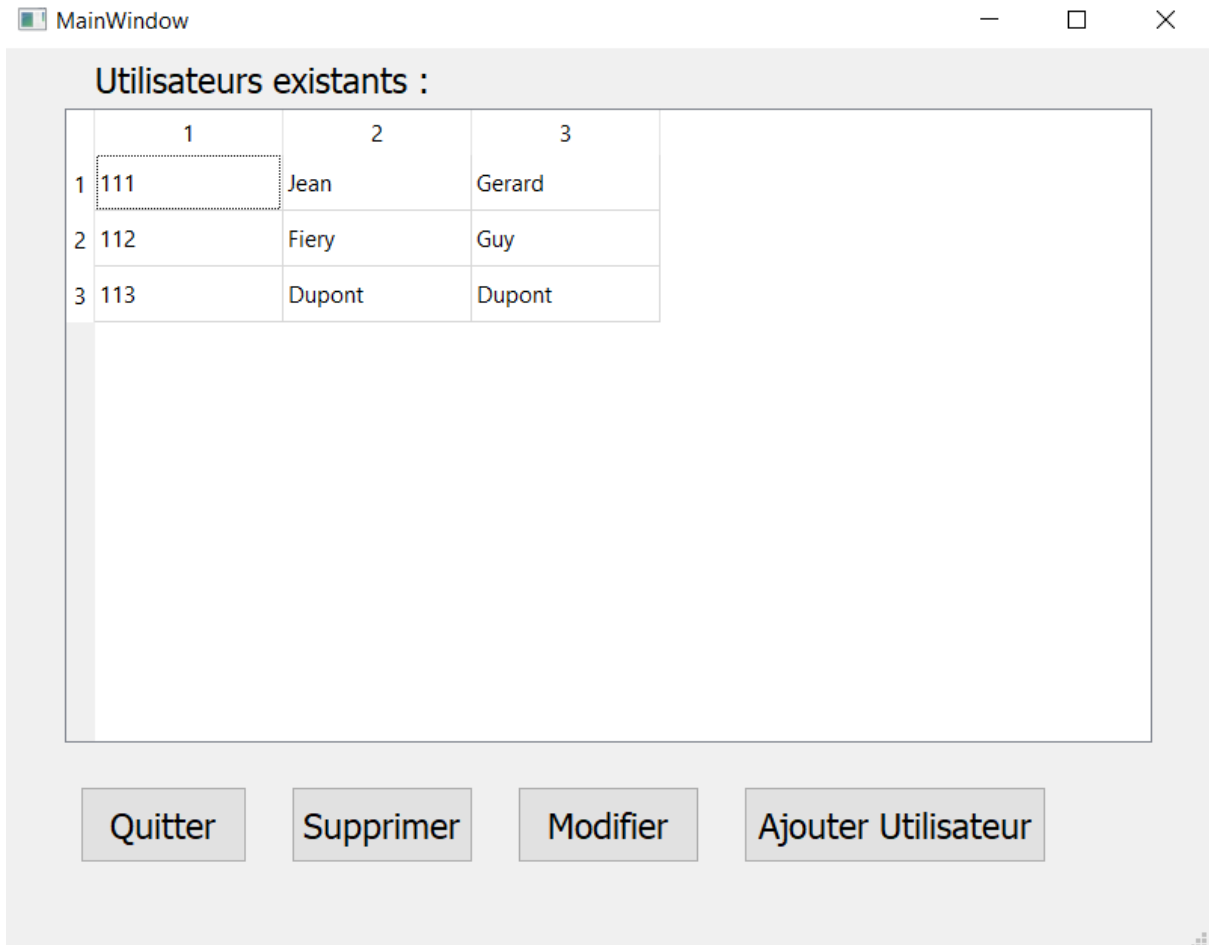


Fiche PP 1 : Interface graphique sous Python

Kern Louis

Réalisation :

Tout d'abords, au lancement de l'application, cette fenêtre s'affiche :



Cette fenêtre à été conçu sur le designer, puis il génère une page en format xml :
(Ceci n'est qu'un morceau du code, a titre d'exemple)

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
  <class>MainWindow</class>
  <widget class="QMainWindow" name="MainWindow">
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>800</width>
        <height>600</height>
      </rect>
    </property>
    <property name="windowTitle">
      <string>MainWindow</string>
    </property>
    <widget class="QWidget" name="centralwidget">
      <widget class="QTableView" name="tableViewUser">
        <property name="geometry">
          <rect>
            <x>40</x>
            <y>40</y>
            <width>721</width>
            <height>421</height>
          </rect>
        </property>
      </widget>
      <widget class="QLabel" name="lblPersExist">
        <property name="geometry">
```

Fiche PP 1 : Interface graphique sous Python

Kern Louis

Code que l'on va ensuite convertir en format Python avec le logiciel fournis par la librairie, pyside2-uic.exe.

Nous obtenons un code comme ceci :

(ici aussi, code écourté pour illustrer)

```
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        if not MainWindow.setObjectName():
            MainWindow.setObjectName(u"MainWindow")
        MainWindow.resize(800, 600)
        self.centralwidget = QWidget(MainWindow)
        self.centralwidget.setObjectName(u"centralwidget")
        self.tableViewUser = QTableView(self.centralwidget)
        self.tableViewUser.setObjectName(u"tableViewUser")
        self.tableViewUser.setGeometry(QRect(40, 40, 721, 421))
        self.lblPersExist = QLabel(self.centralwidget)
        self.lblPersExist.setObjectName(u"lblPersExist")
        self.lblPersExist.setGeometry(QRect(60, 10, 241, 21))
        font = QFont()
        font.setPointSize(14)
        self.lblPersExist.setFont(font)
        self.BtnQuit = QPushButton(self.centralwidget)
        self.BtnQuit.setObjectName(u"BtnQuit")
        self.BtnQuit.setGeometry(QRect(50, 490, 111, 51))
        self.BtnQuit.setFont(font)
        self.BtnDelete = QPushButton(self.centralwidget)
        self.BtnDelete.setObjectName(u"BtnDelete")
        self.BtnDelete.setGeometry(QRect(190, 490, 121, 51))
        self.BtnDelete.setFont(font)
        self.BtnModif = QPushButton(self.centralwidget)
        self.BtnModif.setObjectName(u"BtnModif")
        self.BtnModif.setGeometry(QRect(340, 490, 121, 51))
        self.BtnModif.setFont(font)
        self.BtnAddUser = QPushButton(self.centralwidget)
        self.BtnAddUser.setObjectName(u"BtnAddUser")
        self.BtnAddUser.setGeometry(QRect(490, 490, 201, 51))
```

La fenêtre se connecte au serveur Rest et récupère les données des utilisateurs sur le serveur sous format json, tous ceci est géré par la méthode getPeronne, qui utilise la méthode preparDataToPost (qui dans le cas de getPersonne, envoie une requête de récupération de données) afin de formater la requête qui va demander le code, le nom et le prénom de l'utilisateur et de convertir cette requête au format json avec la librairie json Pickles.

Ensuite, la requête est envoyée vers le serveur Rest (son adresse url est masquée pour des raisons de sécurité). On récupère les données en format json qu'on convertis en dictionnaire python et on affiche les utilisateurs dans le tableau de la première fenêtre (voir exemple ci-dessus).

Il est important de noter qu'en effet le code tel quel ne permet pas encore d'instancier parfaitement de manière dynamique les personnes dans le tableau, les utilisateurs affichés sur la capture d'écran, ont été créé en dur dans le code, par manque de temps, la méthode getPersonne ne permet pas dans son état actuel d'accomplir à 100% sa fonction.

Fiche PP 1 : Interface graphique sous Python

Kern Louis

Méthode getPersonne :

```
dataToPost = '{"client":{"info":{"type":"Python","nom":"PyAnnuaireClient","version":"1.0.0"},"application":{"type":"Python","nom":"PyAnnuaire"},"'
print(dataToPost)
entree = '{"code" : "' + codePersonne + '"}'
dataToPost = Data.prepareDataToPost("READ", entree)
print(dataToPost)

res = requests.post(url, data=dataToPost, verify="u2testca.pem", auth=("adhpvpython", "ACB831"))
print(res.status_code)
print(res.text)
# Conversion de la chaîne de caractères JSON en dictionnaire Python :
resultDict = json.loads(res.text)
# Récupération du dictionnaire pour l'objet JSON "erreur" et affichage du code de l'erreur :
dictErreur = resultDict["erreur"]
print(" dictErreur['code'] : " + str(dictErreur["code"]))
# Récupération du dictionnaire pour l'objet JSON "sortie" et ajout de "py/object" au dictionnaire :
dictSortie = resultDict["sortie"]
dictSortie["py/object"] = "Personne.Personne"
# Conversion du dictionnaire en objet Personne :
resultPers = jsonpickle.unpickler.Unpickler().restore(dictSortie)
if dictErreur["code"] != 0:
    msgBox = QMessageBox()
    msgBox.setIcon(QMessageBox.Information)
    msgBox.setText("Une erreur est survenue, code d'erreur : " + str(dictErreur["code"])) + ": " + dictErreur["message"])
    msgBox.setWindowTitle("Erreur")
    msgBox.setStandardButtons(QMessageBox.Ok)

    returnValue = msgBox.exec()
    if returnValue == QMessageBox.Ok:
        print('OK clicked')
```

Méthode prepareDataToPost :

```
def prepareDataToPost(action, entree):
    dataToPost = '{"client": {"info": {"type": "Python", "nom": "PyAnnuaireClient", "version": "1.0.0"}, "application": {"type": "Python", "nom": "PyAnnuaire"}, '
    dataToPost += action
    dataToPost += '}', "entree": '
    dataToPost += entree
    dataToPost += '}'
    return dataToPost
```

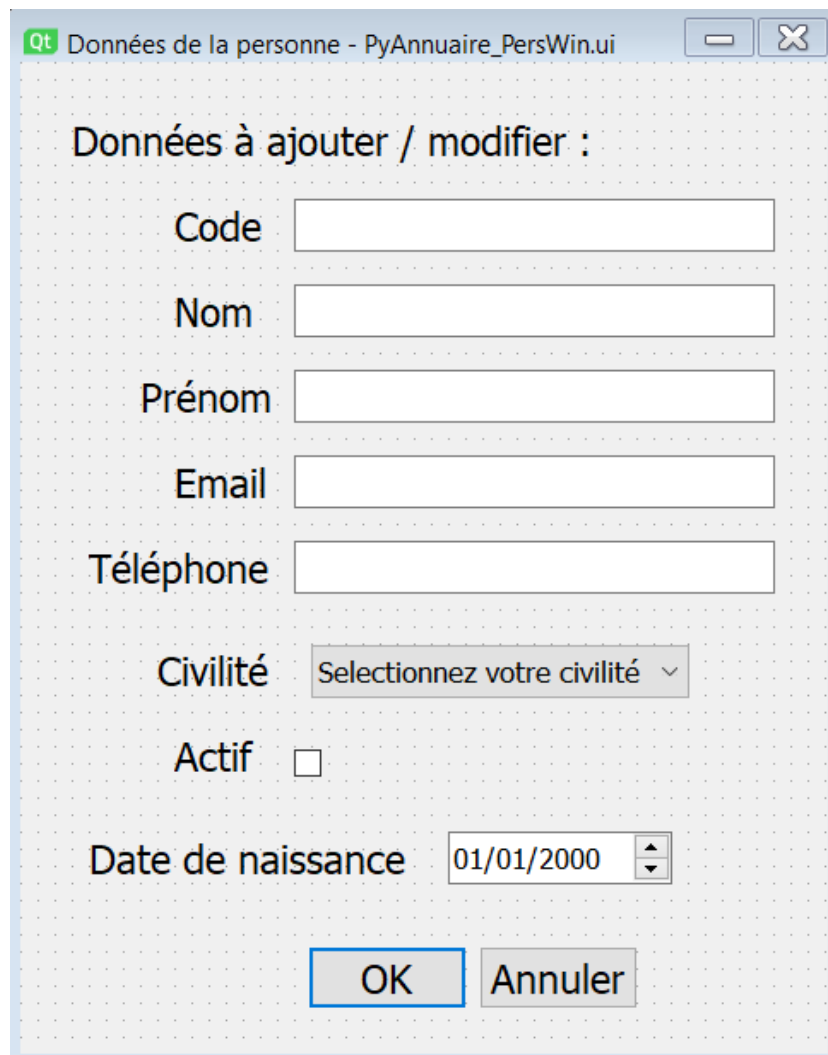
Le bouton Supprimer, va afficher une boîte de dialogue qui va demander à confirmer la suppression (par manque de temps, le fait de cliquer sur ok ne produit rien) :

```
def deleteUser(self):
    msgBox = QMessageBox()
    msgBox.setIcon(QMessageBox.Information)
    msgBox.setText("Êtes-vous sûr de vouloir supprimer l'utilisateur sélectionné ?")
    msgBox.setWindowTitle("Confirmation de suppression")
    msgBox.setStandardButtons(QMessageBox.Ok | QMessageBox.Cancel)
    returnValue = msgBox.exec()
    if returnValue == QMessageBox.Ok:
        print('OK clicked')
```

Fiche PP 1 : Interface graphique sous Python

Kern Louis

On peut aussi choisir d'ajouter ou modifier un utilisateur avec l'utilisation de leurs boutons correspondant, appuyer sur un de ces boutons affiche la fenêtre suivante :



Conçu aussi par le designer, les différents éléments qui compose cette fenêtre correspond aux paramètres de l'objet Personne :

```
class Personne(object):
    def __init__(self):
        self.code=""
        self.nom = ""
        self.prenom = ""
        self.active = False
        self.email = ""
        self.telephone = ""
        self.civilite = ""
        self.date = ""
```

Fiche PP 1 : Interface graphique sous Python

Kern Louis

A noter que pour la date Qt offre une méthode permettant de convertir la date en chaîne de caractère au format jj/mm/aaaa :

```
date_final=QDate.fromString(pers.date, "dd/MM/yyyy")
```

Une fois les modifications apportées, on n'a plus qu'à cliquer sur le bouton « OK », et la méthode updatePersonne se charge du reste, code de la méthode que l'on peut retrouver ici :

```
def updatePersonne(pers):
    # Conversion de l'objet Personne en chaîne de caractères JSON
    jsonStr = jsonpickle.encode(pers)
    print(" jsonStr : " + jsonStr)

    entree = jsonStr
    dataToPost = Data.prepareDataToPost("UPDATE", entree)
    print(dataToPost)

    res = requests.post(url, data=dataToPost, verify="u2testca.pem", auth=("adhpvthon", "ACB831"))
    resultDict = json.loads(res.text)
    # Récupération du dictionnaire pour l'objet JSON "erreur" et affichage du code de l'erreur :
    dictErreur = resultDict["erreur"]
    if dictErreur["code"] != 0:
        msgBox = QMessageBox()
        msgBox.setIcon(QMessageBox.Information)
        msgBox.setText("Une erreur est survenue, code d'erreur : " + str(dictErreur["code"]) + ": " + dictErreur["message"])
        msgBox.setWindowTitle("Erreur")
        msgBox.setStandardButtons(QMessageBox.Ok)

        returnValue = msgBox.exec()
        if returnValue == QMessageBox.Ok:
            print('OK clicked')
    print(" dictErreur['code'] : " + str(dictErreur["code"]))
    print(res.status_code)
    print(res.text)
```

(l'url à été aussi masqué ici).

Même procédé que pour getPersonne mais à l'inverse, on convertis d'abords l'objet personne en dictionnaire python puis en format json, pour ensuite l'envoyer sur le serveur.

Quant à modifier, même procédé qu'ajout d'utilisateur sauf que cette fois on fait un mix de getPersonne et updatePersonne, on récupère ses données, on les modifie, et les renvoie.

Faute de temps, nous n'avons pas pu créer le code correspondant

Conclusion :

Malgré un manque de temps qui nous a empêché de finaliser le projet, nous avons pu acquérir des compétences en matière de recherches de solutions adaptés pour répondre à un besoin précis et de les utiliser adéquatement, ainsi que de pouvoir travailler plus en détail sur le développement d'interfaces graphiques, mais aussi l'interaction avec un serveur Rest distant, nécessitant un formatage de données au format json.