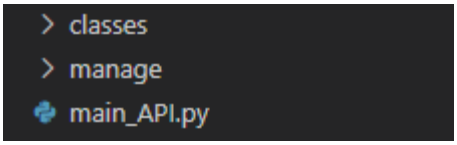
Documentation technique BackOffice

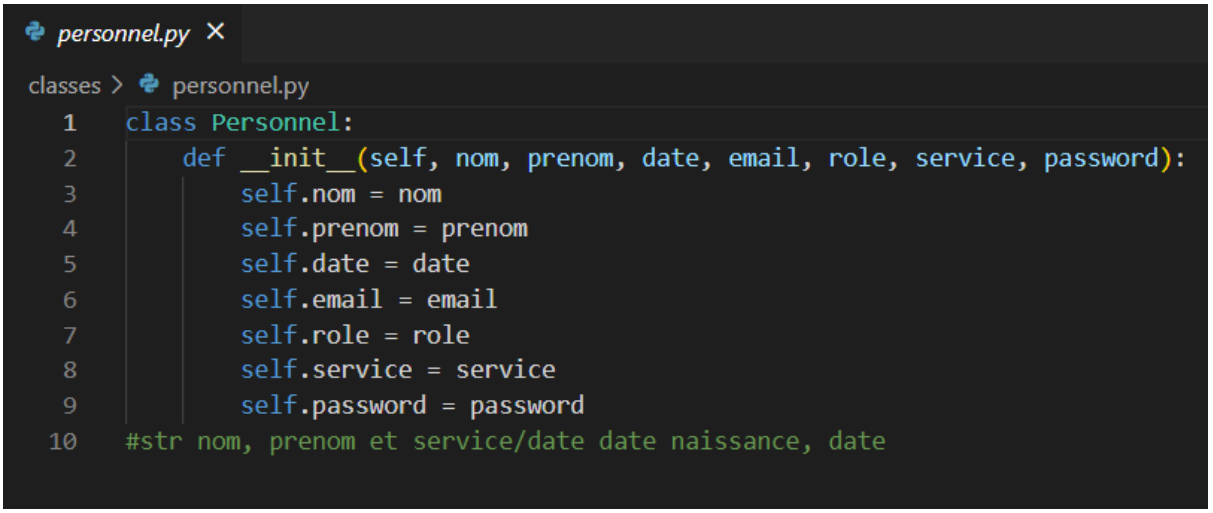
Sommaire :

a) Fonctionnement de l'API

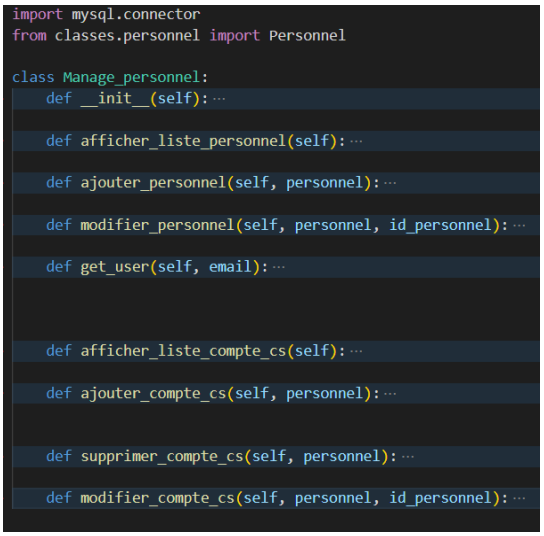
L'API a été développée sous le framework FLASK de python.
La structure du code source :



- Le répertoire "classes" contient les classes qui représentent les entités de la base de données. Cela permet de transiter un objet à la place de plusieurs paramètres.



- Le répertoire manage contient les classes qui ont pour fonction l'interaction avec la base de données. C'est donc dans ses fichiers que l'on a exécuter les requêtes SQL afin d'ajouter ou de récupérer les données voulues.



Le constructeur permet d'établir la connexion entre l'API et la base de données et de créer un curseur qui sera utilisé dans les méthodes.

Les méthodes exécutent les requêtes SQL et récupèrent le résultat pour le changer dans le format voulu

```
def afficher_liste_compte_cs(self):
    # methode pour afficher tous les comptes
    instructionBDD = "SELECT personnelsoignant.id_personnel, nom, prenom, date_naissance, adresse_mail, num_role, num_service,
    self.curseurBDD.execute(instructionBDD)
    resultat = self.curseurBDD.fetchall()
    personnels_soignants = []
    for personnel_soignant in resultat:
        personnels_soignants.append({"IdCompte":personnel_soignant[0], "Nom":personnel_soignant[1], "Prenom":personnel_soignant
    retour = {"ListComptes": personnels_soignants}
    return retour
```

- Le fichier main_API contient les routes et l'application FLASK qui permet de lancer L'API

Le back office utilise 8 routes :

- /api/cs/service : GET
- /api/cs/lit : GET
- /api/cs/vaccin : GET
- /api/cs/role : GET
- /api/cs/compte : GET / POST / DELETE / PUT

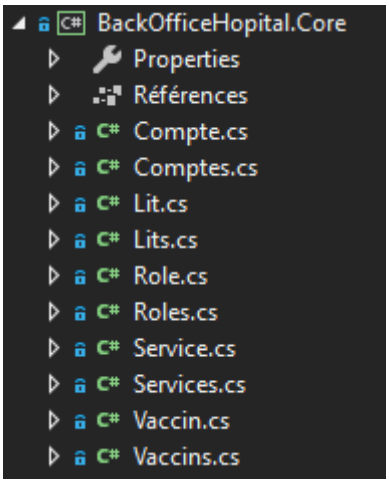
```
#partie c#
@main_API.route('/api/cs/service', methods={'GET'})
def listeServiceCS():···

@main_API.route('/api/cs/lit', methods={'GET'})
def listeLitCS():···

@main_API.route('/api/cs/vaccin', methods={'GET'})
def listeVaccinCS():···


@main_API.route('/api/cs/role', methods={'GET'})
def listeRole():···

@main_API.route('/api/cs/compte', methods={'GET'})
def listeCompte():···

@main_API.route('/api/cs/compte', methods={'POST'})
def inscriptionCS():···

@main_API.route('/api/cs/compte/<id>', methods={'DELETE'})
def suppressionCompte(id):···


@main_API.route('/api/cs/compte', methods={'PUT'})
def modificationCompte():···
```

Les routes utilisent les "manages", ce qui leur permettent de faire appel à la bonne fonction et récupèrent le résultat de la méthode manage choisie. Il peut l'envoyer en réponse de la requête en le convertissant au format JSON.

b) BackOffice

1) Les classes

Le langage C# est un langage typé. Donc pour la manipulation d'objets, il est nécessaire de créer des classes.



La création d'une classe va entraîner la création d'une autre : une classe qui comporte les attributs et les données de l'objet et une autre permettra d'avoir une liste.

```
6 références
public class Role
{
    3 références
    public int IdRole { get; }
    4 références
    public string NomRole { get; }


    0 références
    public Role(int idRole, string nomRole)
    {
        this.IdRole = idRole;
        this.NomRole = nomRole;
    }

    0 références
    public override string ToString()
    {
        return $"{NomRole}";
    }
}
```

```
public class Roles
{
    2 références
    public List<Role> ListRoles { get; set; }
}
```

La classe de liste est nécessaire pour les conversions JSON.

2) Librairie conversion JSON

La conversion du JSON vers un objet ou inversement se fait par la librairie Newtonsoft.

Documentation officiel de Newtonsoft : https://www.newtonsoft.com/json

Conversion d'un JSON vers une liste de vaccins :

```
Vaccins vaccins = JsonConvert.DeserializeObject<Vaccins>(data);
this.lbxStockVaccins.DataSource = vaccins.ListVaccins;
```

Conversion d'un objet vers le JSON :

```
var json:string = JsonConvert.SerializeObject(compte);
```

3) Requête GET

Nous faisons un appel à l'API pour récupérer le stock des vaccins avec la méthode GET. Idem pour récupérer les informations sur l'occupation des lits et pour récupérer les comptes habilités.

```
//Appel à l'API Stock Vaccins
string url = "http://127.0.0.1:5000/api/cs/vaccin";
HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
request.Method = "GET";

WebResponse webResponse = request.GetResponse();

var webStream = webResponse.GetResponseStream();
string data = new StreamReader(webStream).ReadToEnd();

Vaccins vaccins = JsonConvert.DeserializeObject<Vaccins>(data);
this.lbxStockVaccins.DataSource = vaccins.ListVaccins;
```

4) Requête POST et PUT

L'objet de la classe Compte est créé à partir des informations du formulaire.
Il est ensuite converti en JSON.
Il faut spécifier la taille de l'objet envoyé avec la requête (request.ContentLength)

code source pour POST :

```
string url = "http://127.0.0.1:5000/api/cs/compte";
var request = WebRequest.Create(url);
request.Method = "POST";

var json :string = JsonConvert.SerializeObject(compte);
byte[] byteArray = Encoding.UTF8.GetBytes(json);

request.ContentType = "application/x-www-form-urlencoded";
request.ContentLength = byteArray.Length;

var reqStream = request.GetRequestStream();
reqStream.Write(byteArray, offset: 0, count: byteArray.Length);

var response = request.GetResponse();
Console.WriteLine(((HttpWebResponse)response).StatusDescription);

var respStream = response.GetResponseStream();

var reader = new StreamReader(respStream);
string data = reader.ReadToEnd();
Console.WriteLine(data);

int id = JsonConvert.DeserializeObject<int>(data);
```

Code source pour PUT :

```
string url = "http://127.0.0.1:5000/api/cs/compte";
var request = WebRequest.Create(url);
request.Method = "PUT";


var json :string = JsonConvert.SerializeObject(compte);
byte[] byteArray = Encoding.UTF8.GetBytes(json);

request.ContentType = "application/x-www-form-urlencoded";
request.ContentLength = byteArray.Length;

var reqStream = request.GetRequestStream();
reqStream.Write(byteArray, offset: 0, count: byteArray.Length);

var response = request.GetResponse();
Console.WriteLine(((HttpWebResponse)response).StatusDescription);

var respStream = response.GetResponseStream();

var reader = new StreamReader(respStream);
string data = reader.ReadToEnd();
```

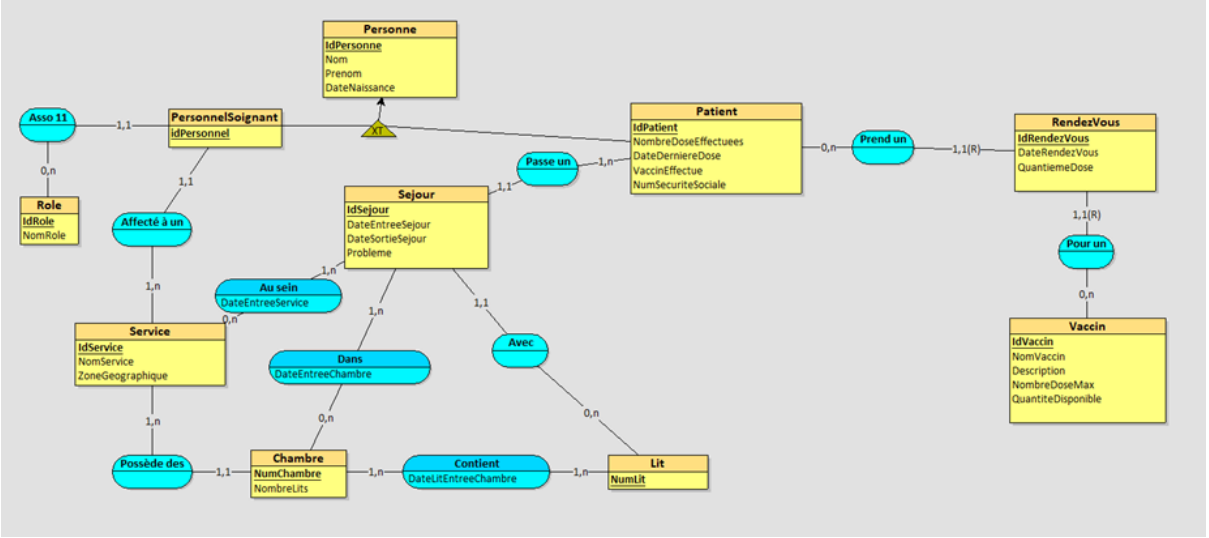
5) Requête DELETE

L'identifiant du compte est dans l'url de la requête

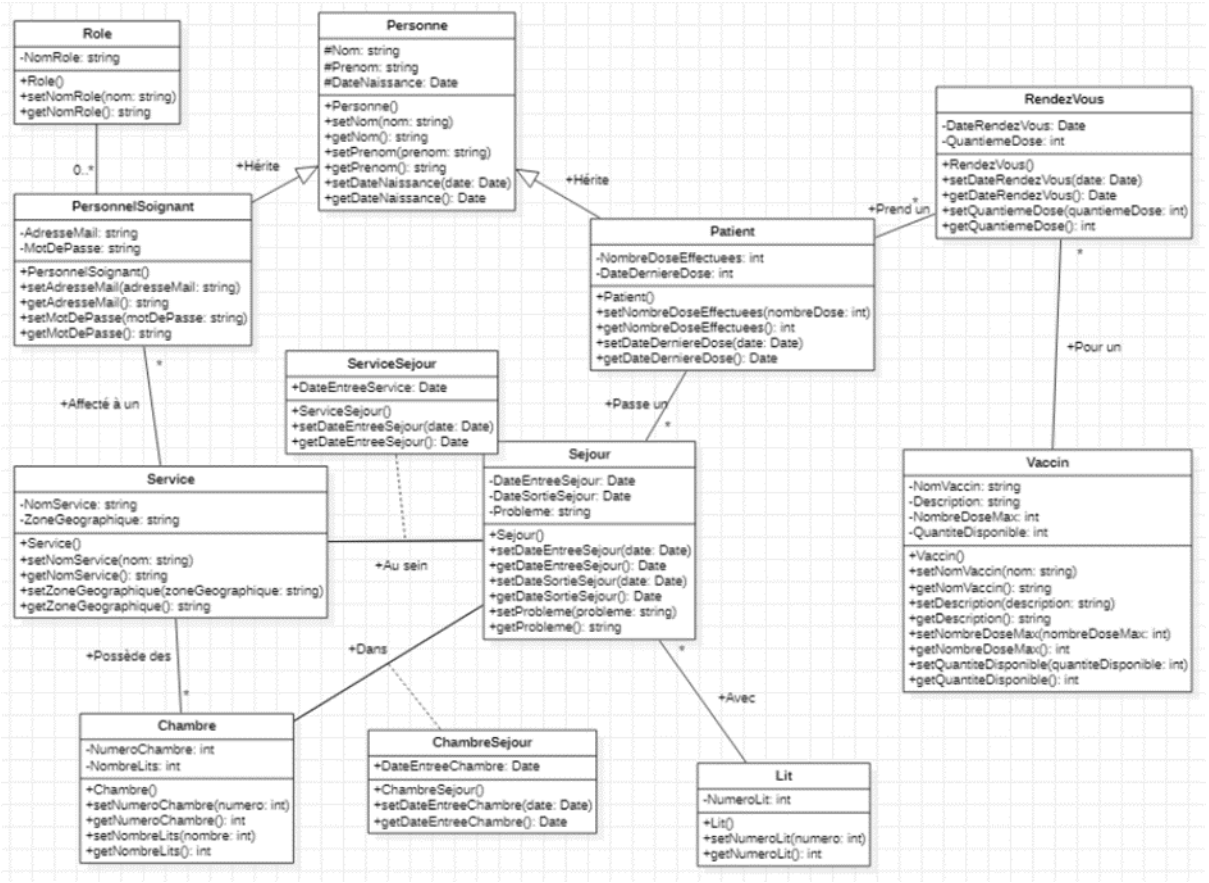
```
string sURL = $"http://127.0.0.1:5000/api/cs/compte/{compte.IdCompte}";
WebRequest request = WebRequest.Create(sURL);
request.Method = "DELETE";

HttpWebResponse response = (HttpWebResponse)request.GetResponse();
this.comptes.ListComptes.Remove(this.compte);
this.Close();
```

c) Base de données

1) Modèle conceptuel de données



2) Diagramme des classes



d) IHM et enchaînement des écrans

Voir la documentation utilisateur