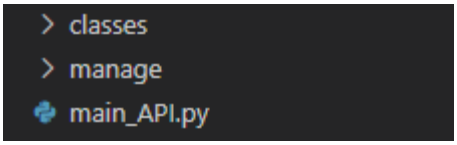
Documentation technique Gestion Vaccins

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.

```python
class Rendez_vous:
    def __init__(self, nom, prenom, date_naissance, num_secu, date_vaccination, nom_vaccin, dose):
        self.nom = nom
        self.prenom = prenom
        self.date_naissance = date_naissance
        self.num_secu = num_secu
        self.date_vaccination = date_vaccination
        self.nom_vaccin = nom_vaccin
        self.dose = dose
```

```python
class Vaccin:
    def __init__(self, id_vaccin, nom_vaccin, description, nombre_dose_max, quantitee_disponible):
        self.id_vaccin = id_vaccin
        self.nom_vaccin = nom_vaccin
        self.description = description
        self.nombre_dose_max = nombre_dose_max
        self.quantitee_disponible = quantitee_disponible
```

- 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.

```python
class Manage_rendez_vous:
    def __init__(self):
        self.conn = mysql.connector.connect(host="localhost",user="root",password="", database="gestion_Patients")
        # connexion à la base de donnée
        self.curseurBDD = self.conn.cursor()

    def ajouter_rendez_vous(self, rendez_vous: Rendez_vous): ···

    def afficher_liste_rendez_vous(self): ···
```

```python
class Manage_vaccin:
    def __init__(self):
        self.conn = mysql.connector.connect(host="localhost",user="root",password="", database="gestion_Patients")
        # connexion à la base de donnée
        self.curseurBDD = self.conn.cursor()

    def afficher_liste_vaccin(self):···
```

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.

```python
def afficher_liste_vaccin(self):
    # methode pour afficher tous les service

    instructionBDD = "SELECT id_vaccin, nom_vaccin, description, nombre_dose_max, quantitee_disponible FROM vaccin"
    self.curseurBDD.execute(instructionBDD)
    resultat = self.curseurBDD.fetchall()
    print(resultat)
    retour = []
    for vaccin in resultat:
        retour.append({"id_vaccin":vaccin[0], "nom_vaccin": vaccin[1], "description": vaccin[2], "nombre_dose_max": vaccin[3], "quantitee_dispon
    return retour
```

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

L'application mobile utilise 3 routes :

- /api/vaccin : GET
- /api/rendez_vous : GET et POST

```python
@main_API.route('/api/vaccin', methods={'GET'})
def listeVaccin():
    try:
        BaseDD = Manage_vaccin()
        dictionnaire_vaccin = BaseDD.afficher_liste_vaccin()
        return jsonify(dictionnaire_vaccin)
    except:
        abort(500)



@main_API.route('/api/rendez_vous', methods={'GET'})
def listeRendez_vous():
    try:
        BaseDD = Manage_rendez_vous()
        dictionnaire_rendez_vous = BaseDD.afficher_liste_rendez_vous()
        return jsonify(dictionnaire_rendez_vous)
    except:
        abort(500)

@main_API.route('/api/rendez_vous', methods={'POST'})
def ajout_rendez_vous():
    message = request.get_json(force=True)
    BaseDD = Manage_rendez_vous()
    message = message["rendez_vous"]
    if "nom" in message and "prenom" in message and "dateNaissance" in message and "numSecuriteSociale" in message and "dateRes" in message and "nomVaccin" in message and "nbrDoses" in me
        rendez_vous = Rendez_vous(message["nom"], message["prenom"], message["dateNaissance"], message["numSecuriteSociale"], message["dateRes"], message["nomVaccin"], message["nbrDoses"]
        try :
            BaseDD.ajouter_rendez_vous(rendez_vous)
            return "Ok"
        except:
            abort(500)
    else:
        abort(406)
```

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) Application mobile

1) Navigation

Pour gérer la navigation, nous avons utilisé le composant bottom-tab-navigator.
documentation : https://reactnavigation.org/docs/bottom-tab-navigator/
Nous avons créé la fonction MyTabs() qui nous renvoie la barre de navigation.

```
function MyTabs() {

  return (
    <Tab.Navigator
      initialRouteName="accueil"
      screenOptions={{
        tabBarActiveTintColor: '#2b8eff',
        headerShown: false
      }}
    >
        <Tab.Screen
        name="Accueil"
        component={Accueil}
        options={{
            tabBarLabel: 'Accueil',
            tabBarIcon: ({ color, size }) => (
            <MaterialCommunityIcons name="home" color={color} size={size} />
            ),
        }}
      />
```

Dans le fichier App.js nous appelons la fonction précédente, ce qui permet d'avoir la navigation active sur l'intégralité des pages.

```
JS App.js > ...
1   import { StatusBar } from 'expo-status-bar';
2   import React from 'react';
3   import { StyleSheet, Text, View } from 'react-native';
4   import {NavigationContainer} from '@react-navigation/native'
5   import MyTabs from './src/navbar';
6
7   export default function App() {
8     return (
9       <NavigationContainer>
10        <MyTabs />
11      </NavigationContainer>
12    );
13  }
```

"Tab.Navigator" contient trois "Tab.Screen" qui représentent les trois pages de l'application

2) Page Accueil

La page Accueil contient un tableau créé par la balise Table.
La balise Row contient la liste des headers. Les données sont dans une liste
La balise Rows contient les données à afficher. Les données sont stockées dans liste de listes

```
  </View>
    <Table borderStyle={{borderWidth: 2, borderColor: '#c8e1ff'}}>
      <Row data={state.tableHead} style={styles.head} textStyle={styles.textheader}/>
      <Rows data={state.tableData} textStyle={styles.text}/>
    </Table>
  </View>
```

```
componentDidMount() {
  this.getPatient();
  this.getStockVaccin();
}
```

Le méthode componentDidMount() permet de lancer automatiquement des fonctions au lancement de la page.
On va alors lancer les requêtes vers l'API pour récupérer les informations que l'on souhaite :

- les informations des rendez-vous des patients
- les informations sur le niveau du stock des vaccins

La méthode getPatient() permet de récupérer les informations nécessaires pour ensuite les afficher dans le tableau.

```
async getPatient() {
  try {
    const response = await fetch('http://127.0.0.1:5000/api/rendez_vous');
    const json = await response.json();
    let liste_rendez_vous = []
    json.forEach(rendez_vous => {
      liste_rendez_vous.push([rendez_vous.nom, rendez_vous.prenom, this.convertDate(new Date(rendez_vous.date_naissance)), rendez_vous.num_secu
    });
    console.log(new Date(json[0].date_rendez_vous).toLocaleDateString('fr-FR'));
    this.setState({ tableData: liste_rendez_vous});
  } catch (error) {
    console.log(error);
  } finally {
    this.setState({ isLoading: false });
  }
}
```

La méthode getStockVaccin() nous permet d'avoir le stock des vaccins et donc de savoir si un des vaccins à un stock inférieur à 20 doses pour afficher l'alerte

```
async getStockVaccin() {
  try {
    const response = await fetch('http://127.0.0.1:5000/api/vaccin');
    const json = await response.json();
    this.setState({ data: json });
  } catch (error) {
    console.log(error);
  } finally {
    this.setState({ isLoading: false });
  }
}
```

Le component Accueil a aussi 2 méthodes qui permettent de gérer l'affichage du format des dates (Date et DateTime).

```
convertDate = (d) => {
  let month = String(d.getMonth() + 1);
  let day = String(d.getDate());
  const year = String(d.getFullYear());

  if (month.length < 2) month = '0' + month;
  if (day.length < 2) day = '0' + day;

  return `${day}/${month}/${year}`
}

convertDateTime = (d) => {
  let month = String(d.getMonth() + 1);
  let day = String(d.getDate());
  const year = String(d.getFullYear());
  let hour = d.getHours()-2;
  let minute = String(d.getMinutes());

  if (hour<0) hour += 24;
  hour = String(hour);
  if (hour.length < 2) hour = '0' + hour;
  if (month.length < 2) month = '0' + month;
  if (day.length < 2) day = '0' + day;
  if (minute.length < 2) minute = '0' + minute;

  return `${day}/${month}/${year} ${hour}:${minute}`
}
```

```
renderVaccinList = () => {
  return this.state.data.map((vaccin) => {
    if(vaccin.quantitee_disponible <= 20)
    return <Text style={styles.alerte}>Attention ! La quantitée de vaccin {vaccin.nom_vaccin} est inférieure à 20</Text>
  })
}
```

```
<View>
  {this.renderVaccinList()}
</View>
```

La méthode rendezVaccinList() permet d'ajouter un message d'alerte au-dessus du tableau quand le stock de l'un des vaccins est inférieur à 20.

3) Page Rendez-vous

La page rendez-vous comporte un formulaire.
On a des balises "TextInput" pour les champs de type text, la balise "Picker" pour les champs "select" (faire un choix) et la balise "Pressable" pour le bouton de validation.

```
<Picker
    selectedValue= {this.setState.nomVaccin}
    onValueChange={(itemValue, itemIndex) => this.setState({nomVaccin: itemValue})}
    style={{ padding: 10,
    marginLeft:30,
    marginRight:30,
    marginBottom:20,
    borderWidth: 1,
    borderStyle: 'solid',
    borderColor: 'black',
    borderRadius:5,
    backgroundColor: '#89c2d9'
    }}>
    {this.renderVaccinList()}
</Picker>
```

```
<View>
    <TextInput style={styles.input}
        placeholder="Nom"
        value = {this.state.nom}
        onChangeText={(nom)=> this.setState({ nom: nom })}
    />
</View>
```

```
<Pressable style={styles.button} onPress={this.onpressEnvoyer} >
    <Text style={styles.textButton}>Envoyer</Text>
</Pressable>
```

La donnée des différents champs sont stockés dans le state du component.

```
state = {
    data:[],
    nom: "",
    prenom: "",
    dateNaissance: "",
    numSecuriteSociale: "",
    dateRes: "",
    nomVaccin:"1",
    nbrDoses: "",
}
```

Le Picker est alimenté par la requête vers l'API au chargement de la page avec la méthode getStockVaccin() grâce au componentDidMount()

```
async getStockVaccin() {
    try {
      const response = await fetch('http://127.0.0.1:5000/api/vaccin');
      const json = await response.json();
      this.setState({ data: json });
    } catch (error) {
      console.log(error);
    } finally {
      this.setState({ isLoading: false });
    }
  }

  componentDidMount() {
    this.getStockVaccin();
  }
```

Quand l'utilisateur clique sur le bouton, tous les champs deviennent vides avec la méthode clearInput() et une requête vers l'API du type POST qui contient les informations du formulaire est envoyée.

```
onpressEnvoyer = () =>{
    this.clearInput()
    this.ajout_rendez_vous()
}
```

```
clearInput = () => {
    this.setState({nom: ""});
    this.setState({prenom: ""});
    this.setState({dateNaissance: ""});
    this.setState({numSecuriteSociale: ""});
    this.setState({dateRes: ""});
    this.setState({avnomVaccinis: ""});
    this.setState({nbrDoses: ""});
}
```

```
ajout_rendez_vous = async () =>{
    const response = await fetch("http://127.0.0.1:5000/api/rendez_vous", {
        method : "POST",
        body: JSON.stringify({rendez_vous: this.state}),
        headers : {
            'Content-type': 'application/json; charset=UTF-8'
        }
    });
}
```

4) Page Stock

Comme pour la page d'accueil, nous avons un tableau mais celui-ci donne le stock des 3 vaccins différents.

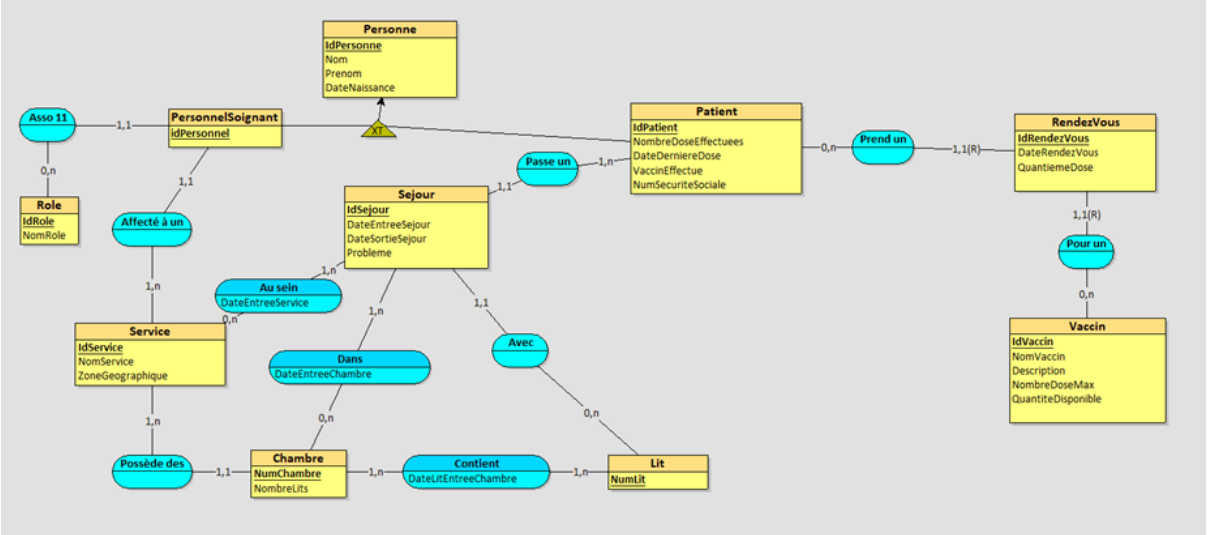
```
<View style={styles.container}>
  <Table borderStyle={{borderWidth: 2, borderColor: '#c8e1ff'}}>
    <Row data={state.tableHead} style={styles.head} textStyle={styles.textheader}/>
    <Rows data={state.tableData} textStyle={styles.text}/>
  </Table>
</View>
```

La méthode getStockVaccin() nous permet de récupérer le stock des vaccins pour qu'ils puissent être affichés dans le tableau.

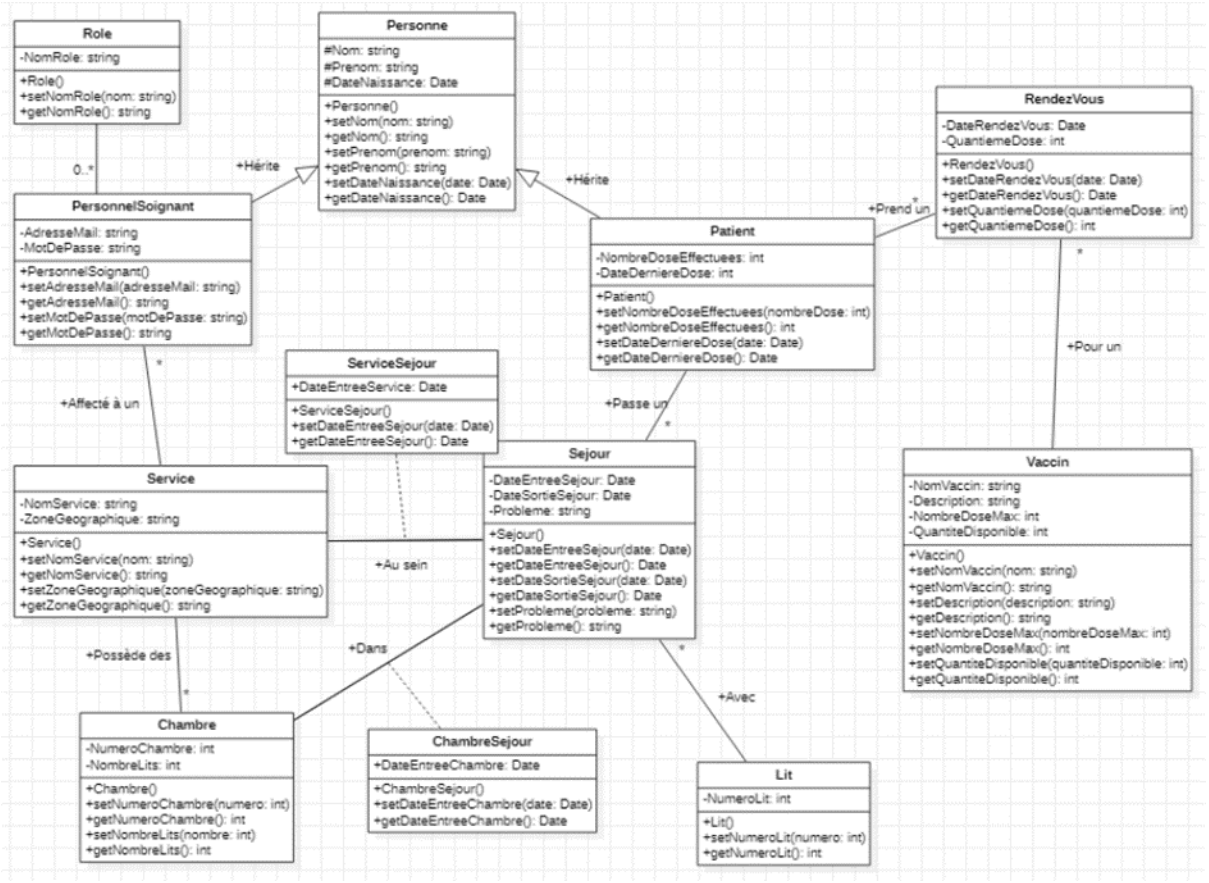
```
async getStockVaccin() {
  try {
    const response = await fetch('http://127.0.0.1:5000/api/vaccin');
    const json = await response.json();
    let liste_vaccin = []
    json.forEach(vaccin => {
      liste_vaccin.push([vaccin.nom_vaccin, vaccin.quantitee_disponible])
    });
    console.log(liste_vaccin)
    this.setState({ tableData: liste_vaccin});
  } catch (error) {
    console.log(error);
  } finally {
    this.setState({ isLoading: false });
  }
}
```

c) Base de données

1) Modèle conceptuel de données



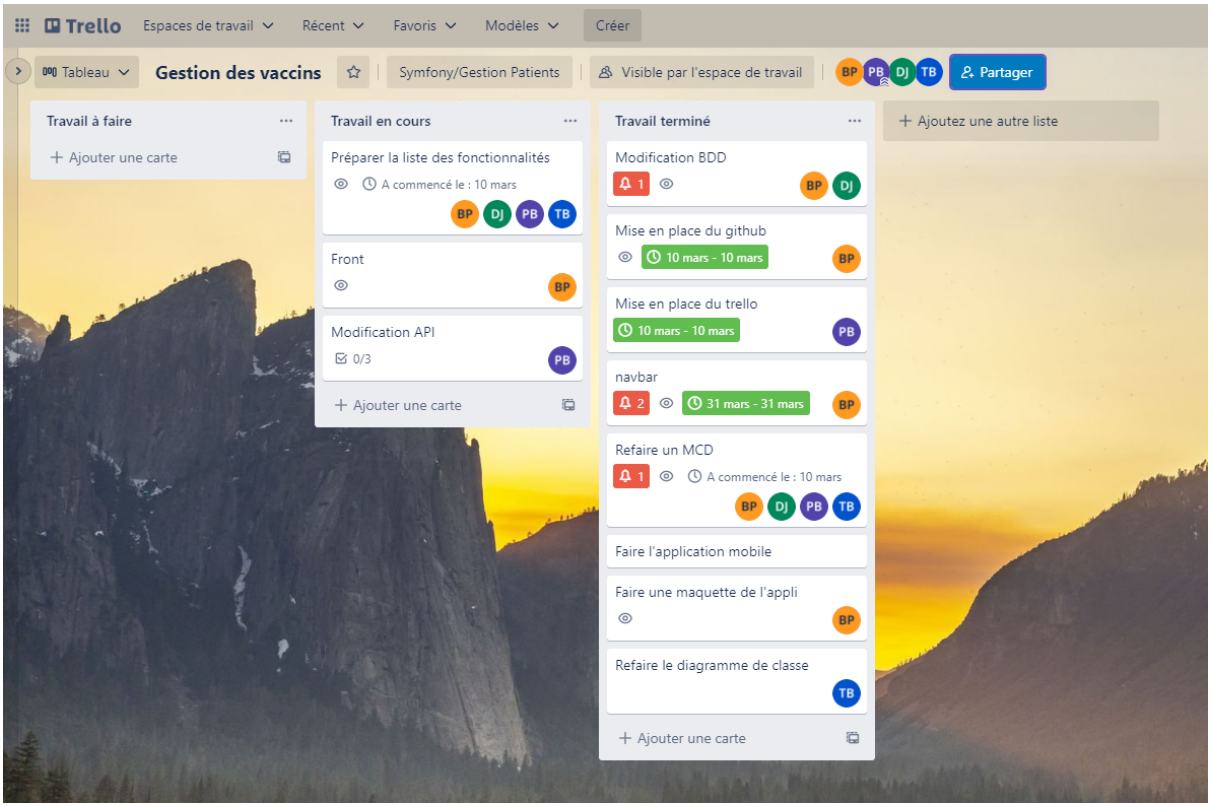
2) Diagramme des classes



d) IHM et enchaînement des écrans

Voir la documentation utilisateur

e) Trello


Trello
Espaces de travail
Récent
Favoris
Modèles
Créer
Tableau
Gestion des vaccins
Symfony/Gestion Patients
Visible par l'espace de travail
BP PB DJ TB
Partager
Travail à faire
Ajouter une carte
Travail en cours
Préparer la liste des fonctionnalités
A commencé le : 10 mars
BP DJ PB TB
Front
BP
Modification API
0/3
PB
Ajouter une carte
Travail terminé
Modification BDD
1
BP DJ
Mise en place du github
10 mars - 10 mars
BP
Mise en place du trello
10 mars - 10 mars
PB
navbar
2
31 mars - 31 mars
BP
Refaire un MCD
1
A commencé le : 10 mars
BP DJ PB TB
Faire l'application mobile
Faire une maquette de l'appli
BP
Refaire le diagramme de classe
TB
Ajouter une carte
Ajoutez une autre liste