# Introduction

Pendant la phase épidémique, les professionnels de santé ont su se coordonner pour assurer l’accès à une consultation pour tous les patients suspects de Covid. La téléconsultation est un cadre pertinent pour réaliser des primo-consultations de patient potentiellement Covid-19. Toutefois, dans certaines situations, elle pourra être insuffisante pour poser l’ensemble du diagnostic et les conditions de prise en charge associées et devra donc être prolongée par une consultation présentielle. C’est au médecin qu’il revient d’apprécier à tout moment la possibilité de poursuivre la téléconsultation en cours ou d’organiser une consultation physique, notamment quand une exploration plus fine du système respiratoire est requise.

La téléconsultation permet aux professionnels médicaux (chirurgiens-dentistes, médecins généralistes, spécialistes) de réaliser une consultation à distance avec un patient. Dans le cadre de l’état d’urgence sanitaire face à l’épidémie de COVID-19, le recours à la téléconsultation et/ou au télé soin permet :

D’assurer une prise en charge à domicile pour les patients présentant les symptômes de l’infection ou reconnus atteints du COVID-19 ;

De faciliter l’accès aux soins, y compris en cas de besoin de prise en charge aigüe ;

D’assurer une continuité de prise en charge chez les patients ayant une maladie chronique et chez les femmes enceintes; en période de confinement impliquant une Limitation de déplacements

# Explication des étapes de travail

Nous avons choisi de travailler avec Python et sa bibliothèque Owlready2 qui permet une programmation objet dans laquelle les objets et les classes sont les entités d’une ontologie. La programmation orientée objet dans les ontologies est une approche à la fois plus simple et plus puissante que les API (*ApplicationProgramming Interface*) habituelles en Java tel que JENA ou la manipulation des ontologies y est beaucoup plus complexes . Donc Owlready nous permet d’assurer :

* la capacité à représenter finement et en détail des connaissances complexes, à les relier entre elles, et à effectuer des raisonnements sur celles-ci.
* La vitesse d’accès d’une base de données, avec ses capacités de stockage et de recherche rapide.
* L’agilité des langages de programmation objet comme Python

# Création de l’ontologie

Dans cette étape nous allons expliquer l’architecture de notre ontologie et comment nous nous y sommes pris afin de la réaliser.

Le script de création de notre ontologie se trouve dans le fichier « create-ontol.py ». Tout d’abord nous commençons par la création de notre ontologie et la définition de son nom de domaine (NameSpace).

Puis nous nous élançons dans le bloc de code qui contiendra nos classes pythons faisant références à nos Classes, DataProperty et. ObjectProperty.

Voici les classes que nous avons créés :

* La classe Humain avec les DP(DataProperty) Nom,Prénom,Sexe et Age (contient les infos en communs aux docteur et aux patients)
* La classe Patient (sous-classes de Humain) avec les DP ID, Poids, Taille, Durée depuis apparition des symptômes, Durée depuis dernier voyage ( ces deux dernieres DP aideront au dépistage des malades de COVID plus facilement ) ,Traitement, Situation Familiale
* La classe Medecin (sous-classes de Humain)
* La classe Localisation quant à elle est la super-classe des classes Wilayas et Dairas
* La classe Symptomes et sa sous-classe Symptômes Covids
* La classe Maladie Chronique
* La classe Orientation ainsi que ses sous classes PCdomicile(Prise en charge a domicile) pour les patients qui devront etre surveillés a leur domicile,RDV( plannification de rendez-vous avec le medecin) , RedirectionHopital

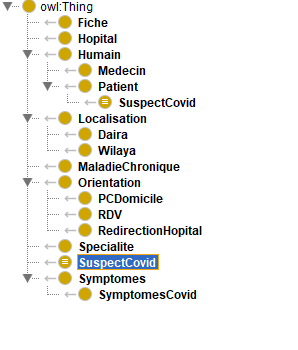


Image 1 : Représentation de nos classes dans Protégé

Puis nous avons reliés nos classes entre eux à l’aide d’ObjectProperties :

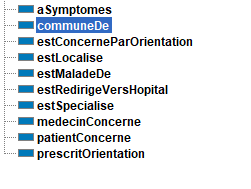


Image2 : Représentation d’Object Properties dans Protégé

Mention spéciale à la classe Suspect Covid représentant un patient ayant au moins 2 symptômes du covid .

Les instances de cette classe seront devinées grâce au raisonneur Hermite.

Petit exemple :

Imaginons notre patient ci-dessous :



Pour une meilleure lisibilité :

*0001*,*Jhon*,*Doe*,*49*,*80*,*1.8*,*Homme*,*Maladie de Kawasaki*,*Anti-dépresseurs*,*Divorcé*,*1*,*25487*,*Bounouh*,*Tizi\_Ouzou*,*"Fievre,Toux seche,Maux de gorge"*

Si nous ajoutons ce patient a notre ontologie et que nous lançons le raisonneur alors notre ami « Jhon Doe  » ne sera plus repertorié en tant que patient mais plutôt en tant que SuspectCovid car il 3 SymptômesCovid.

Enfin nous terminons la création de notre ontologie en définissant la dis jointure de nos classes et en sauvegardant le tout dans notre fichier « sortie.owl » .

# 4.La manipulation de l’ontologie

Pour la manipulation de notre ontologie , nous avons utilisé un mélange de fonctions python et de requêtes SparQl afin d’interroger notre ontologie et pourvoir y créer des instances ou modifier d’autres.

On utilise des fonctions chacune avec un traitement particulier :

Enum class : une fonction qui sert a énumérer les classes de notre ontologie et les affiche afin de pouvoir instancier les classes directement et cela en utilisant une autre fonction qui est ordonner\_class qui nous permet d’ordonner les classes de notre ontologie dans une liste afin de pouvoir instancier les classes directement.

Create\_patient : une fonction qui prends en entrée les informations concernant le patient c’est-à-dire nom prénom wilaya symptômes .. etc. et en retour , nous permet de créer une instance de classe Patient dans notre ontologie . Nous y avons inclus des requêtes Sparql afin de lier les informations du patient entre elles :

* 1er requête: sert à trouver la daira et wilaya de notre patient et les mettre en lien avec le patient via une relation "estLocalise"
* requete = """  
  prefix ns1: <http://sararaouf.org/onto.owl#>   
  prefix ns2: <http://www.w3.org/2002/07/owl#>   
  prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>   
  prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>   
  prefix xml: <http://www.w3.org/XML/1998/namespace>   
  prefix xsd: <http://www.w3.org/2001/XMLSchema#>   
  SELECT ?d ?w  
  WHERE{  
  ?d rdf:type ns1:Daira .  
  ?d ns1:nomDaira ?nd .  
  ?w rdf:type ns1:Wilaya .  
  ?w ns1:nomWilaya ?wn .  
  ?d ns1:communeDe ?w .   
  FILTER regex(?nd,"var1")   
  FILTER regex(?wn,"var2")  
  }  
  """.replace("var1", daira).replace("var2",wilaya)
* 2eme requête : trouver les maladies chroniques du patient existante dans notre base rdf pour les lier , et si il n'existe pas on le crée puis on les lie .

requete = """  
 prefix ns1: <http://sararaouf.org/onto.owl#>   
 prefix ns2: <http://www.w3.org/2002/07/owl#>   
 prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>   
 prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>   
 prefix xml: <http://www.w3.org/XML/1998/namespace>   
 prefix xsd: <http://www.w3.org/2001/XMLSchema#>   
 SELECT ?m ?mn  
 WHERE{  
 ?m rdf:type ns1:MaladieChronique .  
 ?m ns1:nomMaladie ?mn.  
 FILTER regex(?mn,"var")   
 }  
 """.replace("var", j.replace(" ", "\_"))

* 3eme requête : trouver les symptomes/suyptomesCOVID du patient existante dans notre base rdf pour les lier , et si il n'existe pas on le crée puis on les lie
* list\_symptomes = symptomes.split(',')  
  for j in list\_symptomes:  
   requete = """  
   prefix ns1: <http://sararaouf.org/onto.owl#>   
   prefix ns2: <http://www.w3.org/2002/07/owl#>   
   prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>   
   prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>   
   prefix xml: <http://www.w3.org/XML/1998/namespace>   
   prefix xsd: <http://www.w3.org/2001/XMLSchema#>   
   SELECT ?s ?sn  
   WHERE{  
   ?s rdf:type ns1:Symptomes .  
   ?s ns1:nomSymptomes ?sn.  
   FILTER regex(?sn,"var")   
   }  
   """.replace("var",j.replace(" ","\_"))  
   result = graph.query(requete)  
   if (list(result) == []):  
   requete = """  
   prefix ns1: <http://sararaouf.org/onto.owl#>   
   prefix ns2: <http://www.w3.org/2002/07/owl#>   
   prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>   
   prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>   
   prefix xml: <http://www.w3.org/XML/1998/namespace>   
   prefix xsd: <http://www.w3.org/2001/XMLSchema#>   
   SELECT ?s ?sn  
   WHERE{  
   ?s rdf:type ns1:SymptomesCovid .  
   ?s ns1:nomSymptomes ?sn.  
   FILTER regex(?sn,"var")   
   }  
   """.replace("var",j.replace(" ","\_"))

Create\_medecin : elle nous permet de créer un objet médecin dans notre base rdf en lui attribuant l’ID, le nom prénom son sexe et la spécialité du médecin . Nous avons implémenté une requete Sparql afin de chercher la spécialité du medecin dans notre BDD et pouvoir les liées.

1er requête : pour trouver la spécialité du médecin et les mettre en lien avec le médecin via une relation estSpecialise

requete = """  
prefix ns1: <http://sararaouf.org/onto.owl#>   
prefix ns2: <http://www.w3.org/2002/07/owl#>   
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>   
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>   
prefix xml: <http://www.w3.org/XML/1998/namespace>   
prefix xsd: <http://www.w3.org/2001/XMLSchema#>   
SELECT ?s ?ns  
WHERE{  
?d rdf:type ns1:Specialite .  
?s ns1:nomSpecialite ?ns .  
FILTER regex(?ns,"var1")   
}  
""".replace("var1", specialite)

FromCSVtoRDF : Cette fonction nous permet de traduire un fichier csv et l'insérer dans notre base rdf . Notre script prendra une a une les lignes du fichier csv ( tel que une ligne représente un patient ) et utilisera la fonction create\_patient() pour ajouter les patients à notre ontologie.

Orientation : Cette fonction nous permets de relié l'orientation donné (Prise de rendez-vous , Redirection vers hôpital , Prise en charge a domicile ) par un médecin à un patient donné et les lier via les relation estConcerneParOrientation et prescritOrientation .

CréationFiche : Cette fonction nous permets de créer une fiche pour un patient en récoltant ses informations et l’orientation qu’on lui a prescrit via une requête Sparql , puis les informations seront mis dans un dictionnaire .

Requête :

requete = """  
prefix ns1: <http://sararaouf.org/onto.owl#>   
prefix ns2: <http://www.w3.org/2002/07/owl#>   
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>  
prefix xml: <http://www.w3.org/XML/1998/namespace>  
prefix xsd: <http://www.w3.org/2001/XMLSchema#>  
SELECT ?id ?nom ?prenom ?age ?sexe ?situationfam ?dureevoyage ?dureesymp ?traitement ?ndaira ?nwilaya ?poids ?taille ?s ?maladiechronique ?norientation ?idmed ?nommed  
WHERE{  
?p rdf:type ns1:Patient .  
?p ns1:patientID ?id .  
?p ns1:Nom ?nom .  
?p ns1:Age ?age .  
?p ns1:Sexe ?sexe .  
?p ns1:DureeDepuisDernierVoyage ?dureevoyage .  
?p ns1:DureeDepuisApparitionDesSymptomes ?dureesymp .  
?p ns1:Prenom ?prenom .  
?p ns1:Poids ?poids .  
?p ns1:Taille ?taille .  
?p ns1:estMaladeDe ?maladiechronique .  
?p ns1:Traitement ?traitement .  
?p ns1:SituationFamiliale ?situationfam .  
?p ns1:aSymptomes ?s .  
?p ns1:estLocalise ?daira .  
?daira ns1:communeDe ?wilaya .  
?daira ns1:nomDaira ?ndaira .  
?wilaya ns1:nomWilaya ?nwilaya .  
?p ns1:estConcerneParOrientation ?orientation .  
?m ns1:prescritOrientation ?orientation .  
?orientation ns1:nomOrientation ?norientation .  
?m ns1:medecinID ?idmed .  
?m ns1:Nom ?nommed .  
FILTER regex(?id , "^var")  
}  
 """.replace("var", IDpatient)

Génération fiche : Afin de regrouper toutes les fiches des patients et les afficher , pour cela nous utilisons une liste de dictionnaires en bouclant sur la fonction CréationFiche. La sortie sera sous forme de fichier csv contenant les informations du patient , l’orientation donné par le médecin , ainsi que les informations sur le médecin ayant prescrit l’orientation.

# 5.Enrichissement de l’ontologie

Nous nous ne sommes pas contentés de créer les classes de notre ontologie et la manipuler !

En effet, nous avons enrichi notre ontologie grâce à des fonctions récoltant des informations depuis un fichier csv/txt

Enrichissement Wilayas : Nous avons créé 48 instances de classes Wilayas depuis un fichier « wilaya.csv »

Enrichissement Commune : à travers une fonction d’enrichissement qui concerne les daïras depuis un fichier csv , sauf qu’en plus de ça , on lie les Dairas à leur création à leur Wilayas respectives en utilisant la relation « commune De  » et en utilisant le code ID de la wilaya (Au lieu de laisse owlready définir des iris pour les daira et wilayas , nous les avons modifiés tel que l’iri d’une wilaya finit par son Id wilaya ( ex : Alger 16) et la commune par son code postale

PS : nous savons , nous avons confondues daira et communes , on est pas trop doués en éducation civique désolé .

Enrichissement Symptomes et SymptomesCovid: Grace a un code de web scrapping , nous avons pu receuillir depuis un site web la liste des symptomes , puis nous avons filtrer les symptomes du covid en se basant sur les sources de l’OMS.

Enrichissement Maladie Chronique : pour les maladies chroniques qu’un patient peut avoir

Enrichissement Spécialité: pour les spécialités d’un médecin donné

# Développement de l’interface

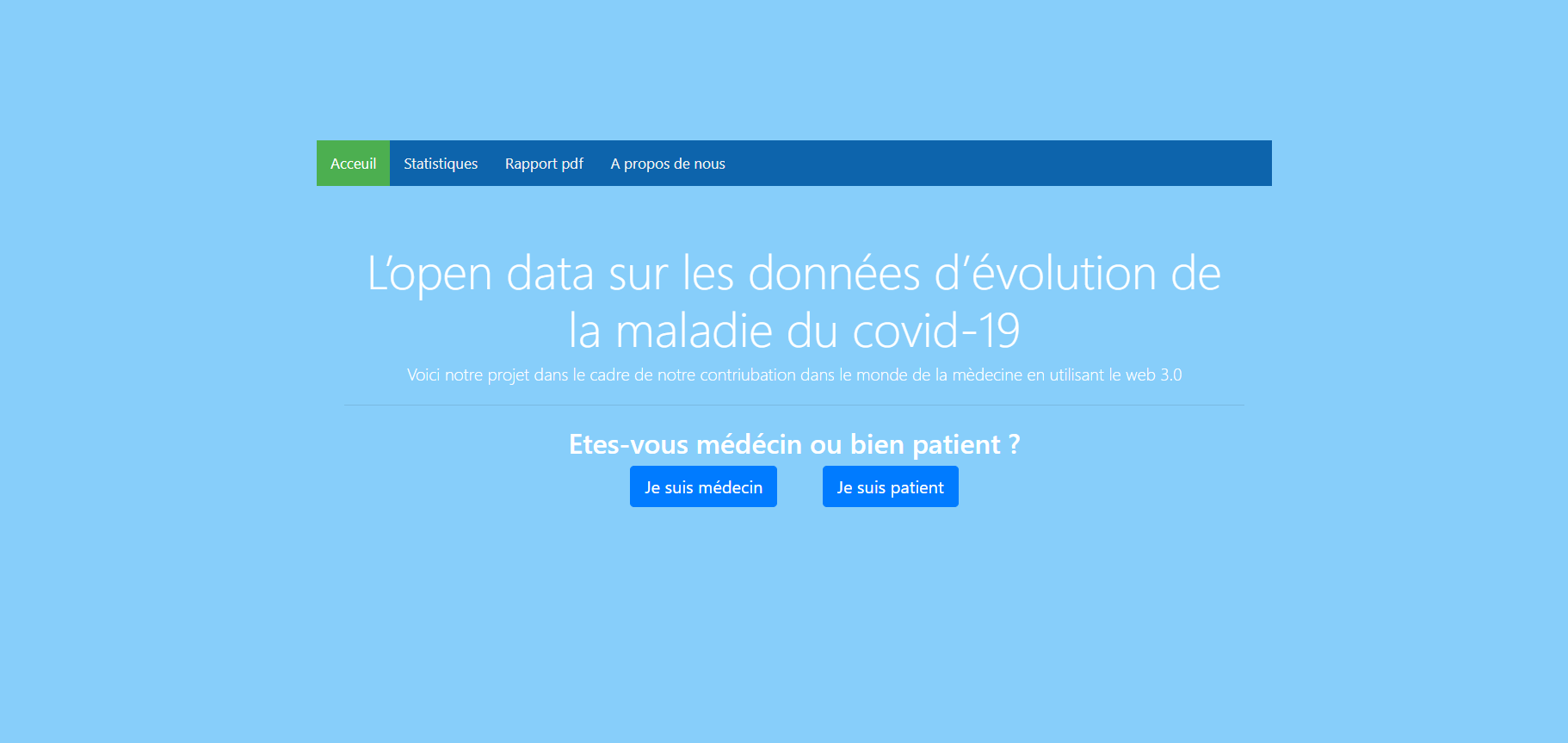
Pour exploiter le Web 3.0 , nous avons décidé de créer une application web pour déployer notre ontologie . En effet , coder une application de bureau aurait minimaliser l’impact que peut avoir le Web sémantique.

Nous avons utilisé le framework Flask de python qui nous permet de déployer notre application au niveau local .

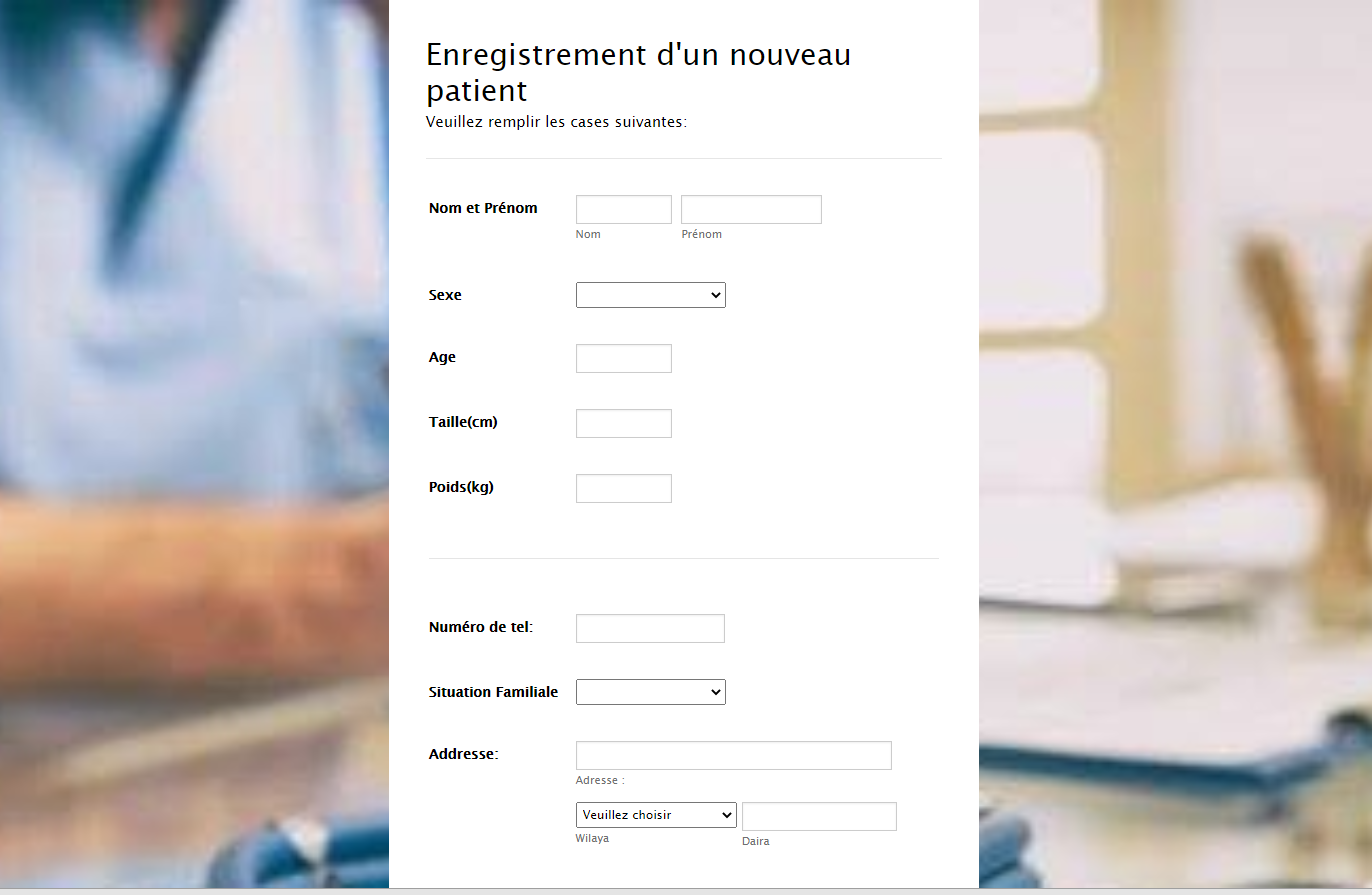
Voici une petite présentation de notre interface :

1. **La page d’accueil** : Une fois que allez lancé le script « app.py » cette page s’ouvrira sur votre navigateur , elle sert d’index a notre site web .

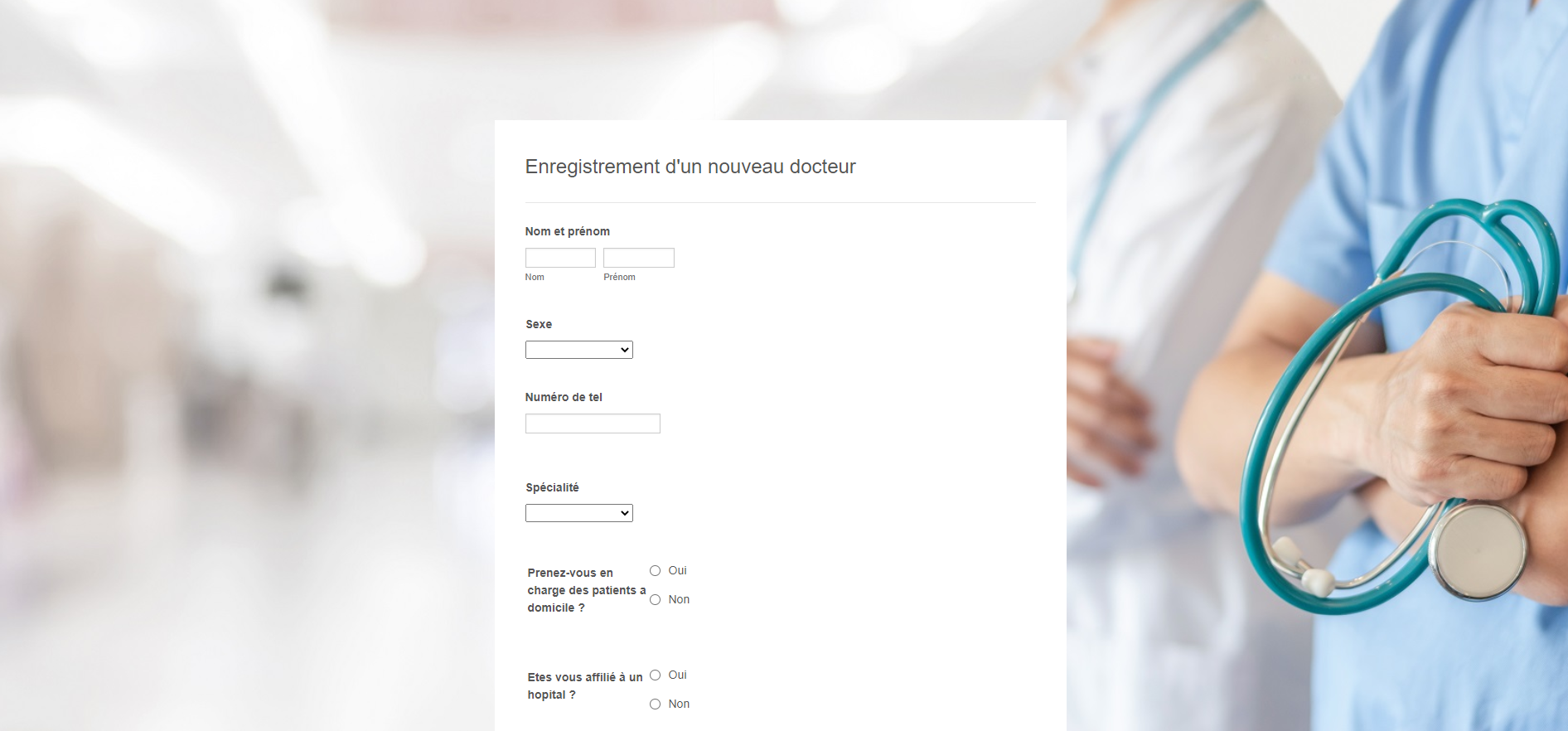
En cliquant sur les boutons « Je suis médecin » et « Je suis patient  » vous serez redirigé vers les pages d’inscriptions respectives



1. **Inscription Patient :** C’est un formulaire qu’un nouveau patient dans notre base de données doit remplir afin d’y être ajouter. Une fois que vous appuyez sur le bouton Submit ( après avoir rempli les champs bien sûr ) , la fonction create\_patient() sera lancé et le patient sera créé dans notre base de données , puis vous serez redirigez vers la page d’index



1. **Inscription Médecin :**  Même procédure que pour le patient sauf qu’ici ce sera la fonction create\_medecin() qui sera lancé.



1. **Suite de l’interface :** Bien que nous soyons peu inexpérimenté en Développement web , nous comptons continuer à travailler sur notre interface en attendant le jour de la présentation du projet .

# Conclusion