Projet “Cabinet médical 2017”, partie IHM

Un serveur HTTP ainsi qu’un squelette du site vous sont fournis. Pour les installer, utiliser GIT (si vous travaillez sous un système MacOS ou Linux) ou TortoiseGIT (installé ou installable sur les systèmes windows).

```
git clone https://github.com/AlexDmr/L3M-Projet.git
```

Une fois les fichiers copiés, installez les bibliothèques dont dépend le projet à l’aide de la commande suivante :

```
npm install
```

1) Comprendre le serveur

Le serveur qui vous est donné est basé sur NodeJS, il utilise le framework Express. Pour l’exécuter, placez vous dans son répertoire et taper la commande :

```
npm start
```

Ce serveur donne accès aux ressources listée ci dessous :

GET /	C'est la page d'accueil (HTML) qui permet de se connecter ensuite en tant que secrétaire ou infirmier. Cette ressource émet principalement le fichier **index.html** qui permet à l'utilisateur de s'identifier.
POST /	Cette ressource reçoit en paramètre **login** qui identifie la personne qui veut se connecter au site. Ce peut être la secrétaire ou un des infirmiers. Dans le cas de la secrétaire, le seul qu'on traitera, le fichier renvoyé est **secretary.html**.
GET /data/cabinetInfirmier.xml	C'est une ressource (un fichier XML), contenant la base de données listant les infirmiers, les patients et les visites prévues des premiers aux derniers.
POST /addPatient	Cette ressource reçoit en paramètre un ensemble de données permettant la création d'un nouveau patient, à savoir : ● **patientName** : le nom du patient ● **patientForname** : son prénom ● **patientNumber** : son numéro de sécurité social ● **patientSex** : son sexe (M ou F) ● **patientBirthday** : sa date de naissance (AAAA-MM-JJ) ● **patientFloor** : étage de son habitation ● **patientStreetNumber** : numéro dans la rue ● **patientStreet** : rue ● **patientPostalCode** : code postal ● **patientCity** : ville La ressource donne une réponse HTTP dont le code peut être 200 si tout s'est bien passé ou 400 si la requête est mal structurée.
POST /affectation	Cette ressource reçoit en paramètre un ensemble de données permettant l'affectation d'un patient à un infirmier, à savoir : ● **infirmier** : l'identifiant de l'infirmier (**"none"** si on veut désaffecter un patient) ● **patient** : le numéro de sécurité sociale du patient
POST /INFIRMIERE	Cette ressource reçoit en paramètre un identifiant (**id**) d'infirmier. Elle renvoi le XML correspondant à l'infirmier si il existe ou bien une erreur de type 400 dans le cas contraire

2) Création d'un client Angular 2

Nous allons créer un client Angular 2 à partir de NPM et d'Angular-CLI. Pour cela suivez les étapes suivantes :

1. Placez vous dans le répertoire où vous souhaitez installer votre projet
   (Prenez donc un répertoire vide)
2. Initialiser un nouveau projet avec npm :
   `npm init`
   Répondez aux questions posées, si vous n'êtes pas inspiré appuyez sur ENTRER.
3. Installez ensuite la Angular-CLI, cela nous servira a :
   `npm install --save-dev @angular/cli`
4. Renommez le fichier package.json en package.json.old.
   Nous faisons cela pour que la CLI Angular ne pense pas qu'un projet existe déjà.
5. Initiez un projet avec Angular-CLI :
   `node ./node_modules/@angular/cli/bin/ng new clientAngular`
   Cela crée un sous répertoire clientAngular qui contient le squelette d'une application.
6. Pour pouvoir développer facilement avec le serveur de développement Angular nous allons configurer un proxy pour que l'application Angular puisse communiquer facilement avec le serveur.
   a. Placez vous dans le répertoire clientAngular
   b. Créez un fichier **`proxy.conf.json`**
   c. Dans ce fichier, placez le contenu suivant:

```
{
 "/data": {
   "target": "http://localhost:8090",
   "secure": false
 },
 "/INFIRMIERE": {
   "target": "http://localhost:8090",
   "secure": false
 },
 "/check": {
   "target": "http://localhost:8090",
   "secure": false
 },
 "/addPatient": {
   "target": "http://localhost:8090",
   "secure": false
 },
 "/affectation": {
   "target": "http://localhost:8090",
   "secure": false
 }
}
```

d. Dans le fichier package.json, modifiez le script start:
```
"start": "ng serve --proxy-config proxy.conf.json"
```

7. Créez un service Angular qui servira à communiquer avec le serveur :
`node ./node_modules/@angular/cli/bin/ng generate service cabinet-medical --module app`
8. Créez un premier composant pour la secrétaire :
`node ./node_modules/@angular/cli/bin/ng generate component secretary`

3) Mise en place de la communication avec le serveur

Nous allons définir le modèle de données que nous allons utiliser dans le projet pour représenter les informations du cabinet médical. Créer un répertoire **src/app/dataInterfaces** dans lequel nous allons placer cinq fichiers :

- sexe.ts

```
export enum sexeEnum {M, F}
```

- adress.ts

```
export interface Adresse {
 ville: string;
 codePostal: number;
 rue: string;
 numéro: string;
 étage: string;
 lat: number;
 lng: number;
}
```

- patient.ts

```
import {sexeEnum} from "./sexe";
import {Adresse} from "./adress";

export interface PatientInterface {
 prénom: string;
 nom: string;
 sexe: sexeEnum;
 numéroSécuritéSociale: string;
 adresse: Adresse;
}
```

- nurse.ts

```
import {PatientInterface} from "./patient";
import {Adresse} from "./adress";

export interface InfirmierInterface {
```

```
 id: string;
 prénom: string;
 nom: string;
 photo: string;
 patients: PatientInterface[];
 adresse: Adresse;
}
```

- cabinet.ts

```
import {InfirmierInterface} from "./nurse";
import {PatientInterface} from "./patient";
import {Adresse} from "./adress";

export interface CabinetInterface {
 infirmiers: InfirmierInterface[];
 patientsNonAffectés: PatientInterface  [];
 adresse: Adresse;
}
```

Maintenant, nous allons éditer le service /src/app/cabinet-medical.service.ts et le composant secretary (Typescript et HTML).

Le but du service sera :

- D'obtenir les données stockés au format XML sur le serveur (ressource /data/cabinetInfirmier.xml) et de les traduire au format de données Typescript exposé ci-dessus.
- De communiquer avec le serveur des demandes de création et d'affectation des patients (voir la partie A: Comprendre le serveur)

Question 0) Initialisation

Allez dans le composant secretary (fichier `src/app/secretary/secretary.component.ts`) et ajouter à son constructeur une dépendance au service cabinet-medical (cela revient à ajouter un paramètre de type CabinetMedicalService au constructeur du composant).

Ajouter une instance du composant secretary dans le template du composant "principal" (fichier `src/app/app.component.html`).

Question 1) Obtenir et convertir les données.

Implémentons une méthode **`getData`** qui renvoie une Promesse d'instance de CabinetInterface

```
getData( url: string ): Promise<CabinetInterface>
```

Indications :

- Faites un appel à la ressource **HTTP GET /data/cabinetInfirmier.xml** en utilisant le service Http fourni par Angular.
  ```
  import {Http} from '@angular/http';
  ```
  Spécifiez ensuite un paramètre de type Http pour le constructeur de votre composant secretary pour pouvoir réaliser des requêtes HTTP.
  Vous pouvez alors faire des requêtes HTTP GET avec la méthode get de l'instance du service Http passé à votre constructeur.
- Dans app.module.ts :
  import {HttpModule} from "@angular/http"
  Et ajouter HttpModule dans le tableau des imports.
- Une fois le document reçu via la requête HTTP GET :
  - Construisez un tableau de tous les infirmiers à partir des infirmiers XML.
  - Construisez un tableau de tous les patients à partir des patients XML. Remarquez que les patient XML peuvent contenir une balise visite et que cette visite peut avoir un intervenant. Lorsqu'une visite est prévue pour un patient et qu'un intervenant (infirmier) est prévu pour cette visite. Alors vous pouvez affecter ce patient à la liste des patients à visiter de cet infirmier.

Vous pourrez utiliser les fonctions et objets HTML5 suivants :

- DOMParser : Pour construire un document DOM à partir d'une chaîne de caractères. Utilisez en particulier la méthode parseFromString qui rend en paramètre le texte à analyser et le type du document (ici: "text/xml").
- querySelector et querySelectorAll : Appliquées à partir d'un document ou d'éléments construit à partir du DOM parser.

Pour visualiser le contenu du cabinet médical que vous avez produit, vous pouvez utiliser la console du navigateur.

Question 2) Composants Infirmier et Patient

Utilisez Angular CLI pour créer deux nouveaux composants : Infirmier et Patient.
Prenez le temps de réfléchir à la façon dont vous allez utiliser ces composants pour concevoir l'interface du secrétaire médical (probablement en créant un composant infirmier et un composant patient).

Pour cela, vous pouvez utiliser les bibliothèques suivantes :

- Angular Material : un ensemble de composants d'interactions de haut niveau.
- alx-dragdrop : Une bibliothèque pour la gestion du drag&drop.

N'oubliez pas qu'il faut pouvoir :

- Créer de nouveaux patient (cela peut passer par la conception d'un composant dédié) et mettre à jour les données du serveur via une requête HTTP POST sur la ressource /addPatient
  Exemple :

```
this._http.post("/addPatient", {
    patientName: patient.nom,
    patientForname: patient.prénom,
```

```
    patientNumber: patient.numéroSécuritéSociale,
    patientSex: patient.sexe === sexeEnum.M ? "M" : "F",
    patientBirthday: "AAAA-MM-JJ",
    patientFloor: patient.adresse.étage,
    patientStreetNumber: patient.adresse.numéro,
    patientStreet: patient.adresse.rue,
    patientPostalCode: patient.adresse.codePostal,
    patientCity: patient.adresse.ville
})
```

- Affecter ou réaffecter les patients aux infirmiers et mettre à jour les données du serveur via une requête HTTP POST sur la ressource /affectation
  Exemple :

```
this._http.post( "/affectation", {
    infirmier: infirmierId,
    patient: patient.numéroSécuritéSociale
})
```

- Désaffecter un patient (il va alors dans la liste des patients non affectés) et mettre à jour les données du serveur via une requête HTTP POST sur la ressource /affectation
  Exemple :

```
this._http.post( "/affectation", {
    infirmier: “none”,
    patient: patient.numéroSécuritéSociale
})
```

Question 3) Intégration de Google Map (seulement si le reste fonctionne)

Pour visualiser où se trouvent les patients, les infirmiers et le cabinet médical, nous allons utiliser une carte Google à l'aide du composant AGM.

```
npm install --save @agm/core
npm install --save-dev @types/googlemaps
```

Dans app.module.ts :

```
import { AgmCoreModule } from '@agm/core';
```

Puis, dans le tableau des imports, ajouter le module suivant :

```
AgmCoreModule.forRoot({
 apiKey: VOTRE_CLEF_GOOGLE_API
})
```

apiKey fait référence à une clef d'application google. Il y a des limites d'utilisations. Créer votre propre clef en suivant les instructions de cette page : https://developers.google.com/maps/documentation/javascript/get-api-key?hl=Fr

Une fois le module Google Maps importer, il faut l'utiliser. Dans un premier temps, nous allons l'utiliser pour trouver les coordonnées spatiales des adresses. Dans le service CabinetMedical :

```
import {MapsAPILoader} from "@agm/core";
import {} from '@types/googlemaps';

...

export class CabinetMedicalService {
 private Pgapi: Promise<any>; // Indique quand l'API est disponible
 private geocoder: google.maps.Geocoder;
```

```
 constructor(private http: Http, mapsAPILoader: MapsAPILoader) {
   this.Pgapi = mapsAPILoader.load().then(() => {
     console.log('google script loaded');
     this.geocoder = new google.maps.Geocoder();
   } );
 }
```

Créons enfin une méthode qui permettra de calculer les coordonnées géographiques (latitude et longitude) de tout objet ayant un attribut adresse de type Adresse :

```
private getLatLngFor( adressables: {adresse: Adresse}[] ) {
  adressables = adressables.slice(); // Copie pour éviter problèmes récursions
  this.Pgapi.then( () => {
   if (adressables.length) {
     const itemWithAdress = adressables.pop();
     const A = itemWithAdress.adresse;
     const address = `${A.numéro} ${A.rue}, ${A.codePostal} ${A.ville}`;
     this.geocoder.geocode({address}, (res, status) => {
       // console.log(itemWithAdress, "=>", status, res);
       if (status === google.maps.GeocoderStatus.OK) {
         const place = res[0].geometry.location;
         itemWithAdress.adresse.lat = place.lat();
         itemWithAdress.adresse.lng = place.lng();
         console.log( itemWithAdress.adresse );
       }
       switch (status) {
         case google.maps.GeocoderStatus.OVER_QUERY_LIMIT:
           adressables.push(itemWithAdress);
           this.getLatLngFor(adressables);
           break;
         default:
           this.getLatLngFor(adressables);
       }
     });
   }
 });
}
```

Au niveau des templates, vous pouvez utiliser (entre autres) les composants agm-map et agm-marker. **Attention à bien spécifier dans le style la HAUTEUR (height) de la balise agm-map car elle vaut 0 par défaut**. Voici un exemple d'usage:

```
<agm-map [latitude]="getCabinetLat()" [longitude]="getCabinetLng()"
(mapReady)="mapReady($event)">
 <agm-marker [latitude]  = "getCabinetLat()"
```

```
              [longitude] = "getCabinetLng()"
              iconUrl     = "/data/hospital-icon.png"
 >
 </agm-marker>

 <agm-marker *ngFor      = "let nurse of getNurses()"
              [latitude]  = "nurse.adresse.lat"
              [longitude] = "nurse.adresse.lng"
              iconUrl     =
"https://d1nhio0ox7pgb.cloudfront.net/_img/v_collection_png/48x48/shadow/
nurse.png"
 >
 </agm-marker>

 <agm-marker *ngFor      = "let patient of getPatients()"
              [latitude]  = "patient.adresse.lat"
              [longitude] = "patient.adresse.lng">
 </agm-marker>
</agm-map>
```

Vous pouvez aussi utiliser les fonctions de décodage d'une coordonnées géographique pour obtenir l'adresse correspondante, il s'agit du géocodage inversé. Vous aurez besoin de la référence à la carte. Dans le code présenté ci dessus, cette référence sera passé à la méthode mapReady dont voici une implémentation possible pour gérer ensuite le click sur la carte :

```
mapReady(map: google.maps.Map) {
 console.log("mapReady", map);
 map.addListener("click", evt => {
   this.cabinetService.getAdresseFromLatLng( evt.latLng );
 });
}
```

Et dans votre service :

```
async getAdresseFromLatLng(latLng: google.maps.LatLng): Promise<Adresse> {
 function getCompValue(comps: google.maps.GeocoderAddressComponent[], name: string): string {
   const comp = comps.find( c => c.types.indexOf(name) >= 0 );
   return comp ? comp.long_name : "";
 }
```

```
 await this.Pgapi;
 return new Promise<Adresse>( (resolve, reject) => {
   this.geocoder.geocode( {location: latLng}, (res, status) => {
     console.log(latLng, status, res);
     if (status === google.maps.GeocoderStatus.OK) {
       const comps: google.maps.GeocoderAddressComponent[] = res[0].address_components;
       const ad: Adresse = {
         ville: getCompValue(comps, "locality"),
         codePostal: parseInt( getCompValue(comps, "postal_code"), 10),
         rue: getCompValue(comps, "route"),
         numéro: getCompValue(comps, "street_number"),
         étage: "",
         lat: latLng.lat(),
         lng: latLng.lng()
       };
       resolve(ad);
     } else {reject(status);}
   });
 });
}
```