Projet “Cabinet médical 2016”

0) Vérification des outils

Vérifiez que vous avez bien les outils suivant :

- **NodeJS** et **NPM**: Dans une console, les commandes node et npm devraient être comprises. Allez sur le site de NodeJS pour télécharger la dernière version le cas échéant.
- **GULP** : La commande gulp devrait être comprise dans une console. Si ça n’est pas le cas, installez le avec la commande :
  npm install -g gulp

1) Amorce de code

Un serveur HTTP ainsi qu’un squelette du site vous sont fournit. 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/amorcePAI.git`

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

```
cd amorcePAI
npm install
```

Une fois l’installation des bibliothèques terminée, vérifier que la chaîne de compilation gulp fonctionne correctement :

```
gulp
```

Vous pouvez regarder comment cette chaîne est instanciée dans le fichier `gulpfile.js`

2) 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 :

```
node   ./serverCabinetMedical.js   (si vous utilisez window)
nodejs ./serverCabinetMedical.js   (si vous utilisez Linux )
```

Par défault, le serveur écoute sur le port 8080, utilisez un navigateur pour allez sur :

```
http://localhost:8080/
```

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 ressoit 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 ● **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

Exercice : Ouvrez votre navigateur, ouvrez le debuggeur, observez dans le panneau "network" la successions d'appel HTTP lorsque vous vous connectez sur le site et vous identifiez en tant que secrétaire. Expliquez les appels que vous observez, qui en est à l'origine

3) Noyau fonctionnel

Editez le fichier `js/secretary.js` à l'aide de votre éditeur préféré (sublimeText par exemple). Nous allons commencer par réaliser le noyau fonctionnel de l'application. Voilà schématiquement ce que ce noyau doit faire :

1. Charger le fichier data/cabinetInfirmier.xml à l'aide d'une requête AJAX. Utilisez pour cela le module utils et sa fonction XHR.
   var utils = require("./utils.js");
   utils.XHR('GET', 'data/cabinetInfirmier.xml').then(function(data) {console.log(data);})
   Voir la section 4 pour une description plus précise de cette fonction.
2. Analyser ce fichier pour produire un document DOM. Pour cela vous pouvez vous référer à DOMParse.
3. A partir de ce document XML, produisez une structure de donnée associant les infirmières et leurs patients, n'oubliez pas les patients qui ne sont affectés à aucune infirmière. Appuyez vous entre autres sur les instructions suivantes :
   - querySelector que vous pouvez appeler à partir du document ou d'un élément.
   - querySelectorAll que vous pouvez appeler à partir du document ou d'un élément.
   - textContent qui vous permet d'avoir le contenu textuel d'un noeud.
   - getAttribute qui vous permet d'obtenir la valeur d'un attribut d'un noeud.
4. A la fin, vous devez produire un objet contenant :
   - patientsRestant : le tableau des patients restants.
   - infirmiers : un tableau associatif des infirmiers, indexés par leur identifiant.
   - Avec :
     i. Un patient est décris par un nom, un prénom, un sexe, une date de naissance, un numéro de sécurité social et une adresse.
     ii. Un infirmier est décris par un identifiant, un nom, un prénom, une photo et un tableau de patients qui lui sont affectés.
5. Implémentez les fonctions d'affectation (ou de désaffectations) ainsi que de création d'un patient. Faites en sorte que votre noyau se synchronise bien avec le serveur (requêtes AJAX via utils.XHR).

4) Appels asynchrones au serveur

L'objet **utils** contient une function **XHR** (pour XmlHttpRequest) qui permet de réaliser des appels HTTP de manière asynchrone. Cette fonction prend trois paramètres, le premier indique si la requête est de type GET ou POST, le second indique la ressource à qui est adressée la requête, enfin le dernier est un objet pouvant contenir les attributs suivants :

- **variables** : Référence un objet dont les attributs seront émis en tant que variables dans le cas d'une requête POST. Par exemple `{login: 'toto', password: 'xxxx'}`
- **form** : Référence un noeud du DOM de type formulaire. Les données de ce formulaire sont alors transmises en tant que variables dans le cas d'une requête POST.

La fonction renvoie une promesse (Promise) qui sera tenue si la requête réussie, la réponse sera alors transmise en paramètre de la fonction de résolution. En cas d'échec de la requête

(code HTTP >= 400), la promesse sera rejetée et la réponse sera passée en paramètre de la fonction de rejet.

Avant de passer à la suite, placez votre code dans un fichier différent de secretary.js.
Ce fichier servira de proxy de noyau fonctionnel. Il contiendra toutes les fonctions nécessaires à la communication avec le serveur et à la transformation des données.
N'oubliez pas d'exporter les functions nécessaire au bon usage de votre module (via module.exports).
Vous importerez ce module dans vos directives afin d'accéder aux données et aux opérations sur ces données.

Utilisez la fonction service des modules Angular pour définir un nouveau service qui servira de proxy pour le noyau fonctionnel :

```
var proxyNF = function($http) {
      // Ajoutez le code de construction du service
      // Cette fonction sera appelée pour instancier un objet service
      …
      // Utilisez $http pour télécharger la base de données
      $http .get ( "data/cabinetInfirmier.xml" )
            .then( function(response) {console.log(response);} )
}
proxyNF.$inject = [ "$http" ]; // Injection de dépendances

module.exports = function(mod) {
      var id = "proxyNF";
      mod.service(id, proxyNF);
      return id;
};
```

Nous verrons dans la section suivante comment utilisez ce fichier.

5) Interface secrétaire

Codez l'interface pour la secrétaire de sorte à lui permettre :

- de visualiser quels sont les patients affectés à quels infirmiers
- de visualiser quels sont les patients non encre affectés
- d'affecter des patients à des infirmiers
- de créer de nouveaux patients

Appuyez vous sur AngularJS 1.5 et Angular material. Pour cela, vous allez

1. Installer Angular et Angular-material via NPM :
   npm install --save angular

npm install --save angular-animate
npm install --save angular-aria
npm install --save angular-messages
npm install --save angular-material

2. Instancier Angular dans votre code:

```
var angular            = require( "angular"         )
  , angularMaterial    = require( "angular-material" )
  ;
require( "angular-material/angular-material.css" );
```

3. Déclarer votre propre module et ses dépendances (ici à Angular-material)

```
var cabinetModule = angular.module( "cabinet", [ angularMaterial ]
);
```

4. Liez le service noyau fonctionnel :

```
require( CHEMIN DU SERVICE NF )(cabinetModule);
```

5. Déclarer des composants pour représenter
    a. un cabinet médical
    b. un patient
    c. un infirmier

Cas du composant repésentant le cabinet médical:

```
var template = require( "./cabinetMedical.html" );
require( "./cabinetMedical.css" );

module.exports = function(mod) {
      var proxyNF = require( CHEMIN DU SERVICE NF )(mod);

      var controller = function( proxyNF ) {
            // Cette fonction sera appelée pour instancier un cabinet
      }
      controller.$inject = [ proxyNF ]; // Injection de dépendances

      mod.component( "cabinetMedical", {
            template    : template,
            bindings    : {
                  titre : "@"
            },
            controller  : controller
      });
};
```

4) Utilisation de Google Map (!!! Mise à jour nécessaire !!!)

Pour visualiser l'adresse des patients encore plus facilement, vous pouvez utiliser le service de cartographie offert par Google Map au travers de son API javascript. Pour cela, il vous faut créer un compte développeur chez google afin d'obtenir des clefs d'accès en suivant ces instructions :

https://developers.google.com/maps/documentation/javascript/tutorial?hl=FR#api_key

Dans votre fichier HTML, insérer un lien vers google map à l'aide la balise :
<script src="https://maps.googleapis.com/maps/api/js?key=**KEY**&sensor=false"></script>

Ajouter une balise **div** dans le code HTML du formulaire d'ajout de patient, prenez soin de lui attribuer un identifiant. Editez ensuite le fichier javascript **ControlPanelInteraction.js** et ajouter dans la fonction **init** le code d'initialisation suivant :

```
new google.maps.Map(document.getElementById( IdentifiantDeVotreBalise )
                      , { center: new google.maps.LatLng(45.193861, 5.768843)
                        , zoom: 11
                        }
                      );
```

Vérifiez que la carte s'affiche effectivement.
Question : Décrivez cette ligne de code, quel est le sens de cette instruction, où est-elle définit, que signifient ses paramètres?

Nous allons maintenant intégrer un marqueur à la carte afin de marquer l'emplacement de l'adresse du patient à l'aide du code suivant :

```
new google.maps.Marker(
                      { position     : new google.maps.LatLng(0, 0)
                      , map          : LA_REF_DE_LA_MAP
                      , title        : "Je suis ici!"
                      } );
```

Question : Décrivez cette ligne de code, quel est le sens de cette instruction, où est-elle définit, que signifient ses paramètres? Où se trouve le marqueur au début? (explorez la carte)

Nous allons maintenant placer dynamiquement le marqueur à l'endroit où l'utilisateur clique, pour cela il est possible de s'abonner aux évènements utilisateurs qui surviennent sur une carte à l'aide du code suivant :

```
google.maps.event.addListener(
         LA_REF_DE_LA_MAP
       , 'click'
       , function(evt) {self.newPatientMarker.setPosition(evt.latLng);}
       );
```

Question : Décrivez cette ligne de code, quel est le sens de cette instruction, où est-elle définit, que signifient ses paramètres? Vous pouvez vous référer à la documentation.

Nous avons maintenant un marqueur sur la carte qu'il est possible de déplacer en cliquant. Nous voudrions exploiter les informations géographiques pour remplir les champs d'adresse du formulaire HTML à l'aide de Google Map. Pour ce faire, nous allons utiliser les fonction de codage géographique (geocoding et geodecoding). Créer tout d'abord dans la fonction d'initialisation un objet de codage géographique:

```
geocoder = new google.maps.Geocoder();
```

Utilisez le ensuite à l'aide du code suivant qu'il vous faudra insérer à l'endroit adéquat (cf exercice suivants)

```
geocoder.geocode(
          {'latLng': evt.latLng}
        , function(results, status) {
              if (status == google.maps.GeocoderStatus.OK) {
                       console.log(results);
                     } else {console.error("Error geocoding:", status);}
       );
```

Question : Décrivez cette ligne de code, quel est le sens de cette instruction, où est-elle définit, que signifient ses paramètres? Vous pouvez vous référer à la documentation.

Exercice : Insérez ce code à l'endroit adéquat pour obtenir l'adresse du marqueur lorsqu'il est posé. Parcourez ensuite les résultats obtenus de sorte à pré-remplir les données du formulaire relatives à l'adresse du patient.

Exercice : Vice versa, mettez à jour le marqueur et la carte lorsqu'une adresse est saisie via le formulaire.