

# *DESCRIPTION DES MISSIONS REALISEES*

Application de consultation des médecins et des  
délégués médicaux

BTS SIO Option SLAM

*BASSETTE CHANCEREUL Gwendoline*

*Réalisé du 19 mars au 6 avril 2020*

## Table des matières

I-	Contexte .....	3
I-	Définition du besoin .....	3
1-	Définition de l'objet .....	3
2-	Forme de l'objet .....	3
3-	Accessibilité/Sécurité .....	3
4-	Contraintes.....	3
II-	Description du domaine de gestion .....	4
III-	Schéma physique de la base de données.....	6
IV-	Description des missions réalisées.....	7
A-	Consultation des délégués.....	8
1-	Consultation de la liste des délégués.....	8
2-	Consultation des informations d'un délégué.....	12
3-	Consultation des médecins .....	16
4-	Page de connexion à l'application.....	20

## I- Contexte

Le laboratoire Galaxy Swiss Bourdin (GSB) a décidé de réaliser un suivi des activités des délégués médicaux qui sont les interlocuteurs de GSB auprès des médecins. Pour l'instant, GSB a divisé la France en 6 secteurs : le nord, le sud, le centre, l'ouest, la région parisienne et les Dom-Tom, mais ce découpage géographique peut, bien entendu, être modifié par la suite. Chaque délégué est affecté sur un secteur spécifique et est responsable du suivi de certains médecins situés sur son secteur de rattachement. Entre 30 et 100 délégués interviennent en moyenne sur un secteur et chaque délégué est responsable en moyenne de 30 médecins. Chaque médecin sera affecté à un délégué médical qui sera donc le seul interlocuteur capable de le renseigner sur les produits vendus par GSB.

Les délégués médicaux ont pour principales missions la présentation des produits médicaux de GSB aux médecins et le recueil des caractéristiques des problèmes médicaux découverts chez les patients ayant utilisé les produits médicaux de GSB.

La direction commerciale de GSB souhaite obtenir de la part des délégués médicaux des informations concernant les visites qu'ils effectuent chez les médecins dont ils ont la charge

## I- Définition du besoin

### 1- Définition de l'objet

Vous êtes chargés de réaliser une application Ionic qui permette aux délégués médicaux de pouvoir gérer les informations souhaitées par la direction commerciale de GSB (les visites, les délégués médicaux et les médecins).

### 2- Forme de l'objet

L'application Ionic, installée sur un smartphone de type Android, sera utilisée par les délégués médicaux.

### 3- Accessibilité/Sécurité

Pour pouvoir utiliser l'application, le délégué médical devra saisir son identifiant et son mot de passe la première fois qu'il utilisera l'application ; son identifiant sera alors enregistré en local sur son smartphone.

### 4- Contraintes

#### Architecture

- Utilisation de Ionic (architecture en composants)
- Utilisation d'une API Rest pour communiquer avec la base de données gérée par MySQL

### **Ergonomie et charte graphique**

- Aucune charte n'est fournie mais vous devez avoir une uniformité dans vos formulaires.

### **Codage**

- Vous utiliserez les règles de bonnes pratiques de développement utilisées pour encadrer le développement d'applications et en faciliter la maintenance.
- Les éléments à fournir devront respecter le nommage des fichiers, des classes, des variables, des paramètres, des composants graphiques...

### **Les données**

- La base de données sera gérée par le SGBD MySQL
- L'accès aux données (ajout, consultation, etc.) sera réalisé grâce à une API Rest. L'API Rest sera sécurisée (accessible uniquement avec une clé) et une limite de 5000 accès journaliers sera appliquée.

## **II- Description du domaine de gestion**

**Les délégués médicaux :** Pour chaque délégué médical de GSB, il est nécessaire d'enregistrer son nom, son prénom, son numéro de téléphone, son adresse mail, son identifiant, son mot de passe et le secteur dans lequel il travaille. Pour chaque secteur on enregistrera son nom et une description.

**Les médecins :** Pour chaque médecin, on dispose des caractéristiques suivantes : son RPPS, son nom et son prénom, l'adresse de son cabinet (rue, ville, code postal), son numéro de téléphone. Un médecin est suivi par un seul délégué médical.

**Les produits :** Pour chaque produit, on dispose des caractéristiques suivantes : un nom, une description, le prix de vente à la pharmacie. **Les visites :** Pour chaque visite, il est nécessaire d'enregistrer la date de la visite, la durée, le produit présenté, un commentaire, le médecin concerné et le délégué médical ayant réalisé la visite.

**Les frais :** Les délégués médicaux peuvent se faire rembourser les frais occasionnés pour chaque visite. Chaque demande de remboursement concerne une visite, porte sur un type de frais précis (repas, frais kilométriques, hébergement, péage) et comporte un montant et un commentaire peut venir préciser la demande de remboursement. La direction de GSB a défini un montant maximum de remboursement pour chaque type de frais.

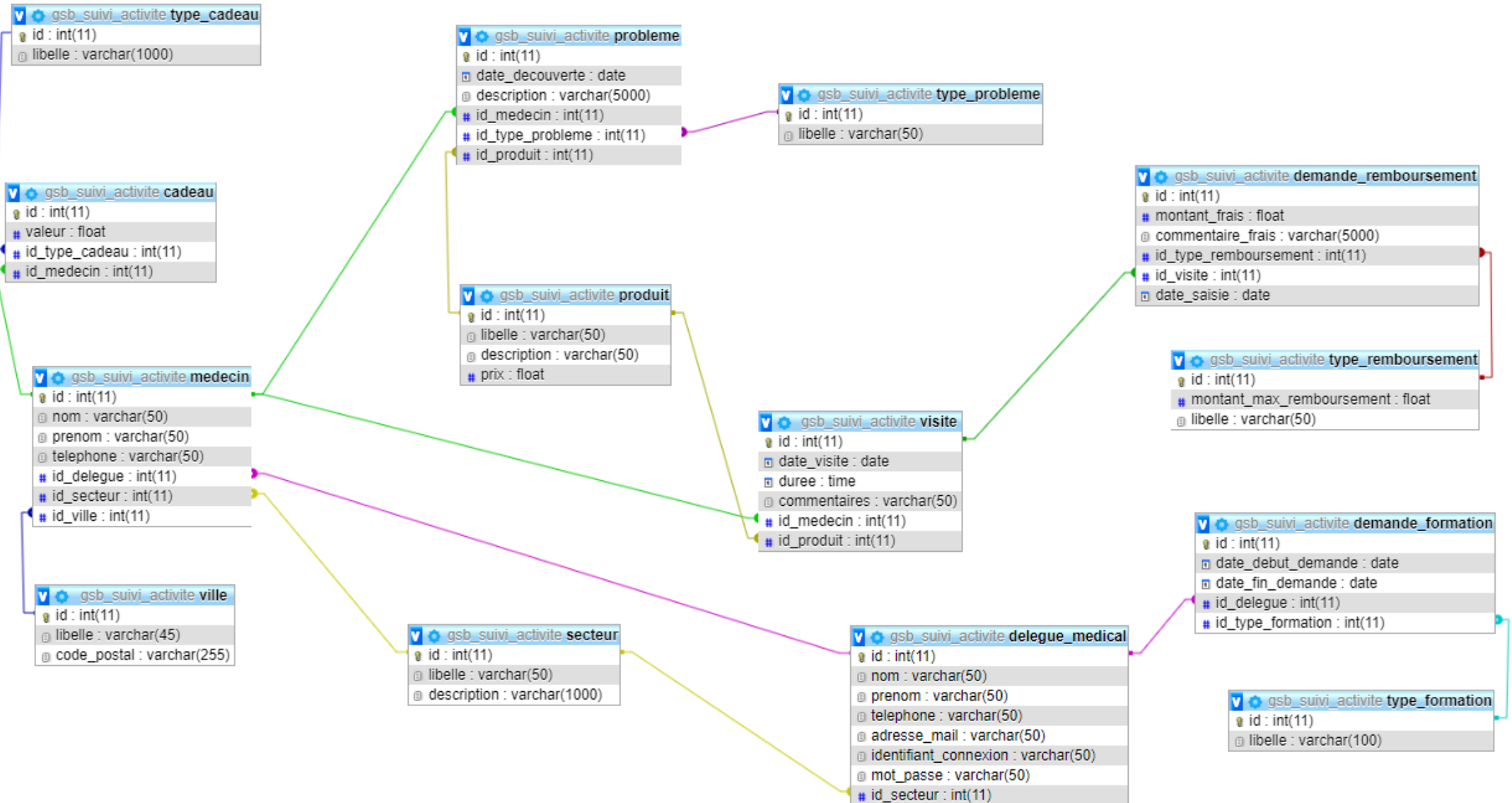
**Les cadeaux offerts aux médecins :** Les délégués médicaux peuvent offrir des cadeaux aux médecins (don de matériel, repas, voyages, frais hôteliers, invitation à une conférence, papeterie, etc.). Chaque cadeau doit être enregistré par le délégué médical : il devra contenir le médecin à qui le cadeau est destiné, la valeur et le type du cadeau et le délégué médical à l'origine du cadeau. Bien entendu, ces différents avantages en nature devront par la suite être déclarés par le laboratoire pharmaceutique auprès de l'administration fiscale. Pour en savoir plus : <https://www.transparence.sante.gouv.fr>

**Les problèmes constatés par les patients lors de l'utilisation des produits vendus par GSB :**

Chaque problème rencontré doit être saisi : il concerne un produit et est affecté à un des médecins suivis par le délégué médical qui réalise la saisie. Pour chaque problème on devra également saisir la date à laquelle le patient a remonté le problème au médecin, le type de problème (palpitations, rougeurs, gonflement...) et une description précise.

**Les demandes de formation :** Chaque délégué médical pourra saisir une demande de formation qu'il souhaite suivre. Chaque demande d'un délégué médical porte sur un type de formation (formation sur une maladie, formation commerciale, formation médicale) et une période souhaitée. Le délégué médical peut ajouter un commentaire permettant de décrire plus précisément la demande souhaitée.

### III- Schéma physique de la base de données

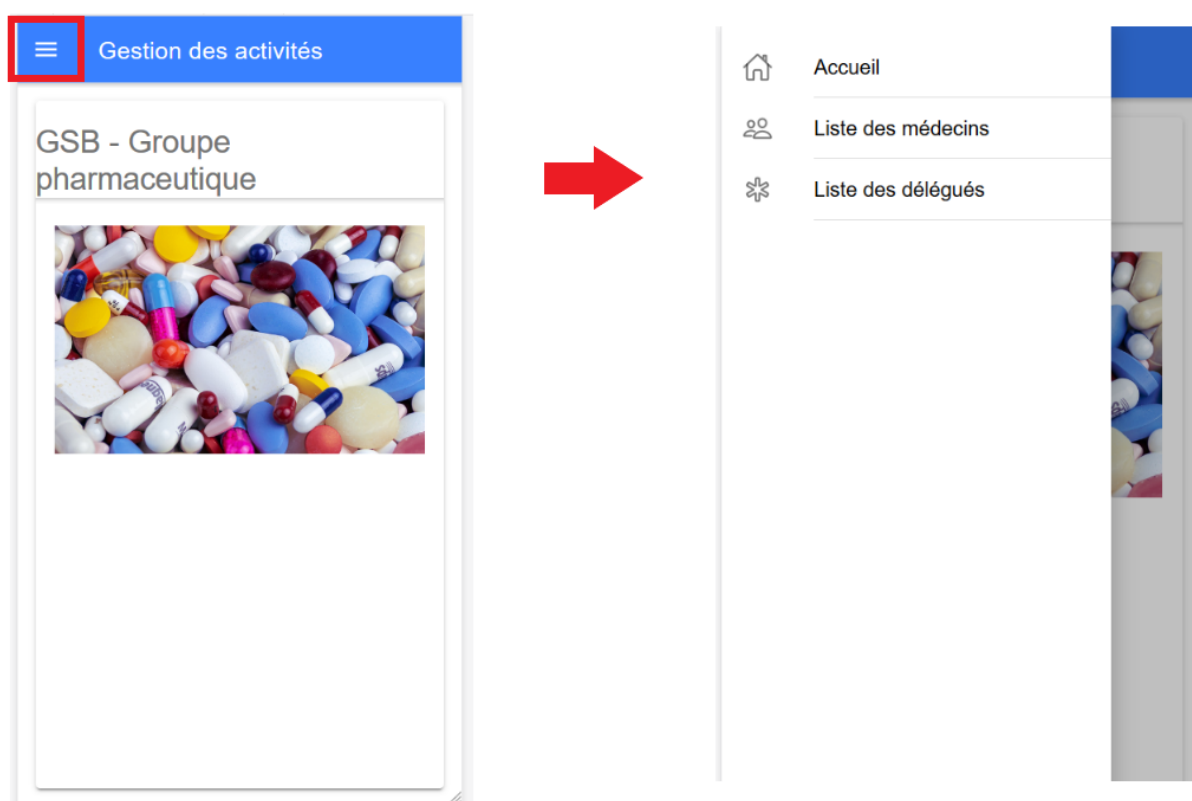


## IV- Description des missions réalisées

Au lancement de l'application et s'il est connecté l'utilisateur obtient la page d'accueil suivante :



En cliquant sur l'icône représentant trois traits horizontaux en haut à gauche de l'écran, l'utilisateur obtient le menu suivant :



Ce menu est accessible depuis toutes les pages de l'application et permet de naviguer entre les pages.

Pour réaliser le menu de navigation j'ai commencé par modifier la page app.component.html dans laquelle j'ai ajouté les instructions qui suivent. Ces instructions permettent de créer les composants du menu.

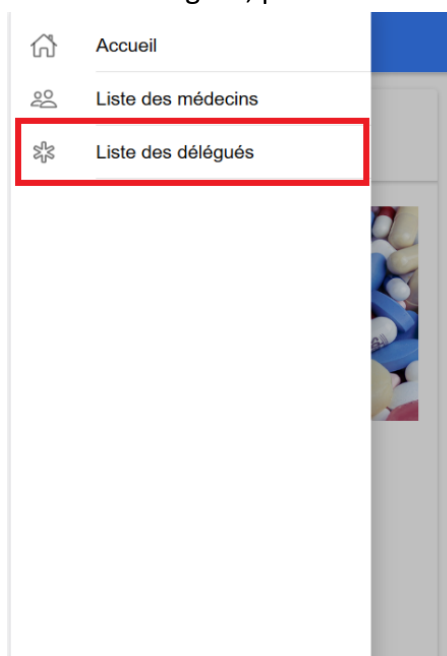
```
<ion-app>
  <ion-router-outlet id="main-content"></ion-router-outlet>
  <ion-menu contentId="main-content" type="overlay">
    <ion-content>
      <ion-list>
        <ion-menu-toggle auto-hide="false">
          <ion-item [routerLink]="['accueil']">
            <ion-icon slot="start" name="home-outline"></ion-icon>
            <ion-label>Accueil</ion-label>
          </ion-item>
          <ion-item [routerLink]="['listemedecins']">
            <ion-icon slot="start" name="people-outline"></ion-icon>
            <ion-label>Liste des médecins</ion-label>
          </ion-item>
          <ion-item [routerLink]="['listedelegues']">
            <ion-icon slot="start" name="medical-outline"></ion-icon>
            <ion-label>Liste des délégués</ion-label>
          </ion-item>
        </ion-menu-toggle>
      </ion-list>
    </ion-content>
  </ion-menu>
</ion-app>
```

Pour chaque option du menu, le [routerLink] = "[listemedecins] " permet d'accéder à la page correspondant au path qui sera dans le fichier routing du composant principal. La balise ion-icon permet d'afficher une icône devant le nom de l'option. La balise ion-label contient le libellé de l'option affichée à l'utilisateur.

## A- Consultation des délégués












### 1- Consultation de la liste des délégués

L'utilisateur peut consulter la liste de tous les délégués, pour cela il doit cliquer sur l'option du menu « liste des délégués » :





L'utilisateur obtient alors la page suivante qui liste les noms et prénoms de tous les délégués médicaux.

Liste des délégués	
	Joan Veronica
	Rachel Sylvester
	Kibo Ignatius
	Carl Lenore
	Nathaniel Thomas
	Uma Prescott
	Kevyn Hanae
	Demetrius Allen
	Kane Kaye
	Paro Sebas
	Joseph Mikayla

Les informations concernant les délégués médicaux sont enregistrées dans la base de données, j'ai donc commencé par créer une classe qui contiendra toutes les caractéristiques d'un délégué. Pour générer la classe `delegate` j'ai utilisé la commande **ionic generate class models/delegate**

Voici le contenu de la classe `Delegate`:

```
export class Delegate {
  id: number;
  nom: string;
  prenom: string;
  telephone: string;
  secteur: string;
}
```

Un API REST est utilisé pour obtenir la liste des délégués médicaux. Voici la méthode qui nous était fournie pour obtenir tous les délégués :

```
private function lire_les_delegates()
{
  $req = $this->_obj_base->prepare("SELECT delegate_medical.id, nom,prenom,telephone,
  libelle as secteur FROM delegate_medical join secteur on id_secteur=secteur.id");
  $req->execute();
  if ($req->rowCount() > 0) {
    $result = $req->fetchAll();
    // Statut OK + mise en forme des caractéristiques au format demandé
    //(appel de la méthode convertir_donnees)
    $this->reponse($this->convertir_donnees($result), 200);
  } else {
    // Si aucun enregistrement, retour avec le statut 404 (not found)
    $this->reponse('', 404);
  }
}
```

Cette fonction effectue un select sur la base de données qui retourne le nom, prénom, numéro de téléphone et le libellé du secteur où il exerce de chaque délégué médical. Si au moins un délégué est retourné, alors on convertit les données au format JSON, sinon on retourne le code d'erreur 404.

Il était ensuite nécessaire de demander à ionic de générer le service avec la commande suivante : **ionic generate service apirest**.

Dans le fichier apirest.service.ts j'ai ajouté les instructions entourées en rouge.

The image shows a code editor with the file `apirest.service.ts`. The code is as follows:

```
@Injectable({
  providedIn: 'root'
})
export class ApirestService {
  apiURL: string = 'http://localhost/gbsuiviact/';

  constructor(private http: HttpClient) {}

  public getDelegueListe(): Observable<Delegue[]> {
    return this.http.get<Delegue[]>(this.apiURL + 'delegates')
      .pipe(
        tap(res => console.log('Lecture des délégués OK' + res)),
        catchError(this.handleError<Delegue[]>('Lecture des délégués', []))
      );
  }

  private handleError<T>(operation = 'operation', result?: T) {
    return (error: any): Observable<T> => {
      console.error(error);
      console.log(`${operation} failed: ${error.message}`);
      return of(result as T);
    };
  }
}
```

Annotations (in French) with arrows pointing to the code:

- Importer les classes nécessaires à l'appel de l'API REST**: Points to the first four `import` statements.
- Importer la classe `Delegue`**: Points to the `import { Delegue } from './models/delegue';` statement.
- Renseigner l'URL à laquelle est située l'API REST**: Points to the `apiURL` property assignment.
- Créer un objet de la classe `HttpClient`**: Points to the `private http: HttpClient` in the constructor.
- Méthode dans votre service qui appellera l'API REST pour obtenir la liste des délégués**: Points to the `getDelegueListe` method.
- Méthode qui gérera les problèmes d'erreur lors de l'appel de l'API REST**: Points to the `handleError` method.

L'API REST sera donc appelée avec une méthode HTTP de type GET et avec l'URI suivant : **http://localhost/gbsuiviact/delegates**

J'ai ensuite modifié mon composant principal dans le fichier `app.module.ts` pour qu'il autorise l'utilisation de la classe `HttpClientModule`.

Puis j'ai ajouté les instructions suivantes au fichier `listedelegue.page.ts` :

```
import { Component, OnInit } from '@angular/core';
import { Delegate } from '../models/delegate';
import { ApirestService } from '../apirest.service';

@Component({
  selector: 'app-listedelegues',
  templateUrl: './listedelegues.page.html',
  styleUrls: ['./listedelegues.page.scss'],
})
export class ListedeleguesPage implements OnInit {
  lesDelegues: Delegate[] = [];
  constructor(private monServ: ApirestService) { }

  ngOnInit() {
    this.monServ.getDelegueListe().subscribe(
      value => {
        this.lesDelegues = value;
      },
      error => {
        console.log('Récupération liste délégués impossible');
      }
    );
  }
}
```

Import de la classe `delegate`

Import de la classe de mon service

Attribut qui contient les délégués

Attribut qui contient un objet de la classe de mon service

Appelle de la méthode de mon service (celle qui appellera l'api `rest` pour avoir la liste des délégués)

Enfin, dans le fichier `listedelegues.page.html` j'ai ajouté les instructions nécessaires à l'affichage du nom et du prénom de chacun des délégués. Voici ces instructions :

```
<ion-header [translucent]="true">
  <ion-toolbar color="primary">
    <ion-buttons>
      <ion-menu-button auto-hide="false"></ion-menu-button>
      <ion-title>Liste des délégués</ion-title>
    </ion-buttons>
  </ion-toolbar>
</ion-header>












<ion-content>
  <ion-list>
    <ion-item *ngFor="let delegate of lesDelegues">
      <ion-icon slot="start" name="people-outline"></ion-icon>
      <ion-label>
        {{delegate.prenom}} {{delegate.nom}}
      </ion-label>
    </ion-item>
  </ion-list>
</ion-content>
```

Instruction de la barre de titre et du bouton permettant d'accéder au menu

Permet de réaliser un `foreach` : on parcourt la liste `lesDelegues` et chaque élément de la liste est placé à tour de rôle dans la variable `delegate`

Pour chaque délégué on affiche une icône, son prénom et son nom

On obtient alors la page suivante :


	Liste des délégués
	Joan Veronica
	Rachel Sylvester
	Kibo Ignatius
	Carl Lenore
	Nathaniel Thomas
	Uma Prescott
	Kevyn Hanae
	Demetrius Allen
	Kane Kaye
	Paro Sebas
	Joseph Mikayla

## 2- Consultation des informations d'un délégué


Il est possible de consulter les informations du délégué en cliquant sur son nom dans la liste. Par exemple, pour obtenir les informations de Rachel Sylvester :

≡


Liste des délégués




Joan Veronica




Rachel Sylvester




Kibo Ignatius




Carl Lenore




Nathaniel Thomas




Uma Prescott




Kevyn Hanae




Demetrius Allen



Kane Kaye



Paro Sebas



Joseph Mikayla



≡

Délégué

Rachel Sylvester

téléphone : 05 16 77 77 37  
secteur : Nord

Pour obtenir la page contenant les informations du délégué j'ai ajouté une page `detaildelegue` avec la commande **ionic generate page detaildelegue**.

J'ai également modifié la route associée à cette page pour qu'elle doit recevoir l'identifiant du délégué concerné :

```
{
  path: 'detaildelegue/:id'
  loadChildren: () => import('./detaildelegue/detaildelegue.module').then( m => m.DetaildeleguePageModule)
},
```

Afin d'obtenir les informations du délégué j'ai utilisé la méthode `lire_un_delegue` de l'API REST :

```
private function lire_un_delegue($id)
{
  if (empty($id) == false) {
    // le paramètre contient une valeur
    // on prépare et on exécute la requête permettant d'obtenir les
    // caractéristiques du délégué médical
    $req = $this->_obj_base->prepare("SELECT delegue_medical.id, nom,prenom,
      telephone,libelle as secteur FROM delegue_medical
      join secteur on id_secteur=secteur.id WHERE delegue_medical.id = :par_id");
    $req->execute(array(':par_id' => $id));

    if ($req->rowCount() == 1) {
      $result = $req->fetch();
      // Status OK + mise en forme des caractéristiques au format demandé
      //(appel de la méthode convertirDonnees)
      $this->reponse($this->convertir_donnees($result),200);
    } else {
      // Si aucun enregistrement, statut "No Content"
      $this->reponse('', 204);
    }
  } else {
    // le paramètre transmis est vide: status Bad Request
    $this->reponse('', 400);
  }
}
```

Cette méthode effectue un select sur la base de données pour obtenir le nom, le prénom, le numéro de téléphone et le libellé du secteur dans lequel travaille le délégué médical dont l'identifiant est passé en paramètre. Si un enregistrement est retourné alors on convertit les données au format JSON avec la méthode `convertir_donnees` et on retourne le code statut confirmant que tout s'est bien passé. Si aucun enregistrement n'est retourné alors on retourne le code statut correspondant à « aucun contenu ». Si l'identifiant passé en paramètre est vide on retourne le code statut correspondant à « mauvaise requête ».

Dans le fichier `apirest.service.ts`, j'ai ajouté une méthode qui permet d'appeler l'API REST pour obtenir les caractéristiques de l'id passé en paramètre :

```
public getDelegue(id: string): Observable<Delegue> {
  return this.http.get<Delegue>(this.apiUrl + 'delegues/' + id)
    .pipe(
      tap(res => console.log('Lecture du délégué OK' + res)),
      catchError(this.handleError<Delegue>('Lecture du délégué'))
    );
}
```

L'API REST sera donc appelée avec une méthode HTTP de type GET et avec l'URI :

**http://localhost/gbsuiviact/delegates/n**

où n représente l'identifiant du délégué pour lequel on souhaite obtenir les caractéristiques

J'ai ensuite modifié le fichier listedelegates.page.html pour indiquer que lors du clic sur l'item de la liste, on appellera la route /detaildelegue/ avec l'identifiant du délégué sur lequel on a cliqué :

```
<ion-content>
  <ion-list>
    <ion-item *ngFor="let delegue of lesDelegates" [routerLink]="'/detaildelegue/' + delegue.id">
      <ion-icon slot="start" name="people-outline"></ion-icon>
      <ion-label>
        | {{delegue.prenom}} {{delegue.nom}}
      </ion-label>
    </ion-item>
  </ion-list>
</ion-content>
```

Dans le fichier detaildelegue.page.ts, j'ai ajouté les instructions entourées en rouge:

```
import { Component, OnInit } from '@angular/core';
import { Delege } from '../models/delege';
import { ApirestService } from '../apirest.service';
import { ActivatedRoute } from '@angular/router';
```

Import de la classe de mon service  
et de la classe delege

```
@Component({
  selector: 'app-detaildelegue',
  templateUrl: './detaildelegue.page.html',
  styleUrls: ['./detaildelegue.page.scss'],
})
```

Déclaration et initialisation d'un  
objet de la classe Delege

```
export class DetaildeleguePage implements OnInit {
  public unDelege: Delege = {
    id: 0,
    nom: '',
    prenom: '',
    telephone: '',
    secteur: ''
  };
}
```

Ajout d'un attribut qui contient un  
objet de la classe de mon service.

Ajout d'un attribut pour  
gérer la route

```
constructor(private monServ: ApirestService, private activatedRoute: ActivatedRoute) {}
```

```
ngOnInit() {
  this.monServ.getDelege(this.activatedRoute.snapshot.params.id).subscribe(
    value => {
      this.unDelege = value;
    },
    error => {
      console.log('Récupération délégué impossible');
    }
  );
}
```

Récupération du délégué  
retourné par l'API REST

Récupération de l'id passé  
dans la route

On appelle la méthode  
getDelege de mon service qui  
appellera l'API REST pour avoir les  
caractéristiques d'un délégué  
dont l'id est passé en paramètre

Enfin pour afficher les informations à l'utilisateur, j'ai ajouté les instructions suivantes dans le fichier **detaildelegue.page.html** :

```
<ion-header [translucent]="true">
  <ion-toolbar color="primary">
    <ion-buttons>
      <ion-menu-button auto-hide="false"></ion-menu-button>
      <ion-title>Délégué</ion-title>
    </ion-buttons>
  </ion-toolbar>
</ion-header>
```

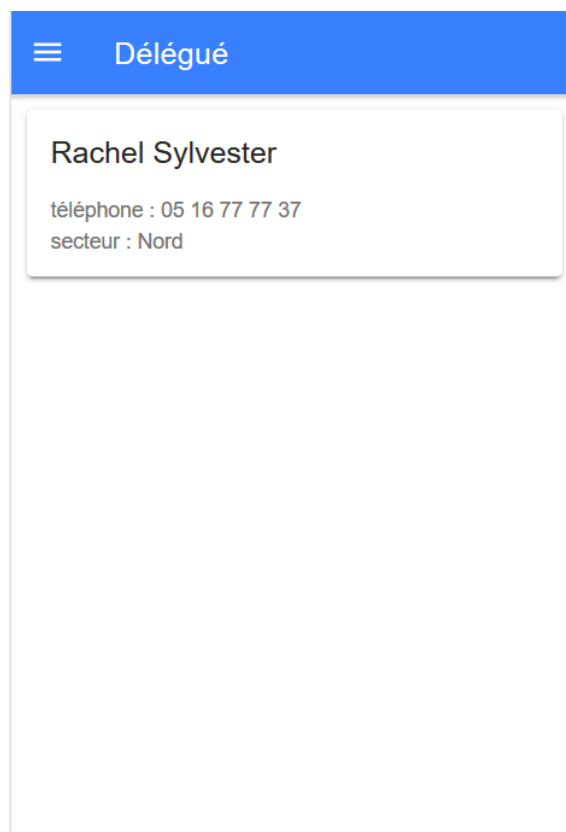
Barre bleue avec le titre de la page et le bouton permettant d'afficher le menu de navigation

```
<ion-content>
  <ion-card>
    <ion-card-header>
      <ion-card-title>{{unDelegue.prenom}} {{unDelegue.nom}}</ion-card-title>
    </ion-card-header>
    <ion-card-content>
      téléphone : {{ unDelegue.telephone }} <br>
      secteur : {{ unDelegue.secteur }}
    </ion-card-content>
  </ion-card>
</ion-content>
```

Affichage du nom et du prénom du délégué en titre de l'encadré

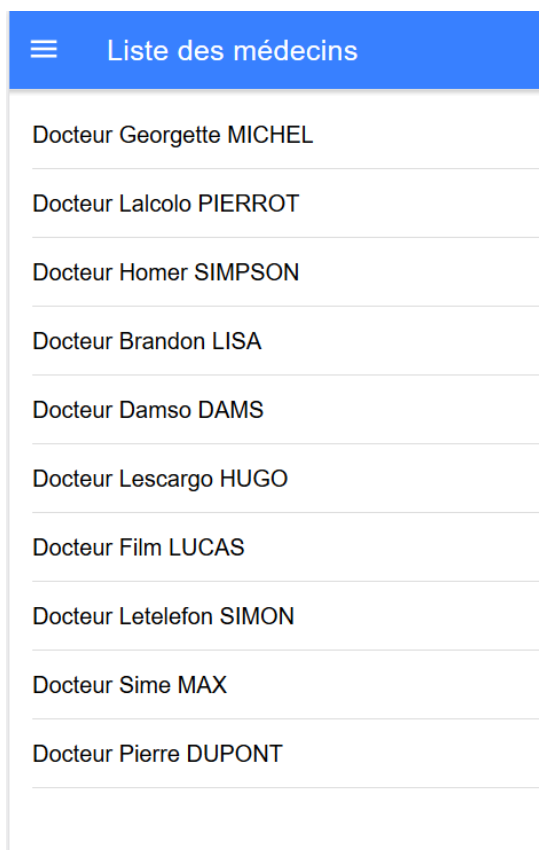
Affichage du numéro de téléphone et du secteur dans lequel travaille le délégué en contenu de l'encadré

Lorsque l'utilisateur a cliqué sur le nom d'un conseiller il obtient la page suivante :



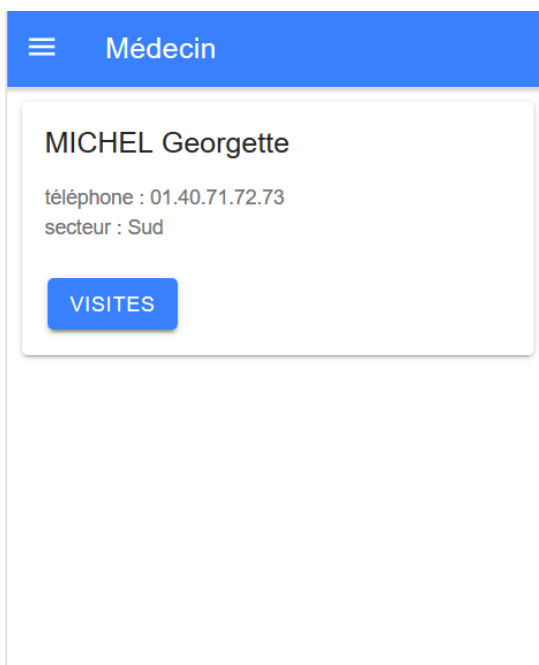
## B- Consultation des médecins

Comme pour la consultation des délégués, il est possible de consulter la liste des médecins. La page s'affiche de la manière suivante :



Docteur Georgette MICHEL
Docteur Lalcolo PIERROT
Docteur Homer SIMPSON
Docteur Brandon LISA
Docteur Damso DAMS
Docteur Lescargo HUGO
Docteur Film LUCAS
Docteur Letelefon SIMON
Docteur Sime MAX
Docteur Pierre DUPONT

Lorsqu'on clique sur le nom d'un médecin, on obtient ses informations :

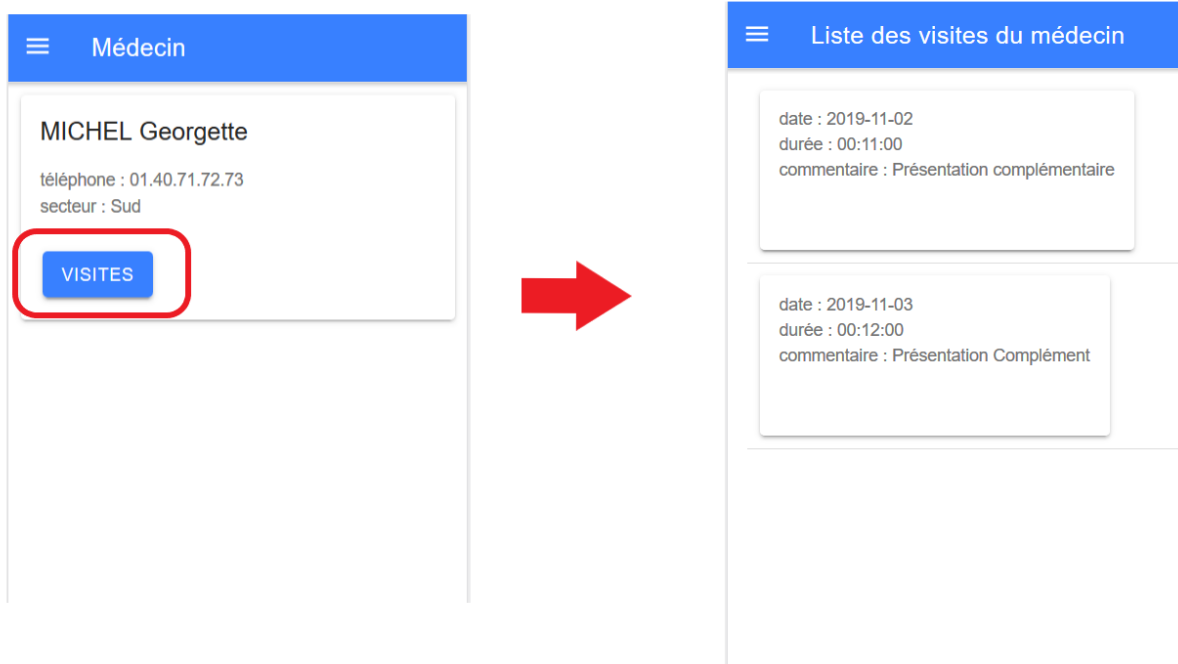


<b>MICHEL Georgette</b> téléphone : 01.40.71.72.73 secteur : Sud <a href="#">VISITES</a>
---

Pour réaliser ces deux pages, j'ai procédé exactement de la même manière que pour les délégués.



Les médecins reçoivent des visites, on peut les voir en cliquant sur le bouton « visites » :



Pour obtenir les visites du médecin j'utilise l'API REST. J'ai donc écrit la méthode suivante qui réalise un select dans la base de données pour obtenir toutes les visites du médecin passé en paramètre, on convertit les données au format JSON (méthode convertir\_donnees) et on retourne les données et le code statut qui informe que tout s'est bien passé. S'il n'y a aucun enregistrement alors on retourne le code statut 404.

```
private function lire_les_visites_un_medecin($id_medecin)
{
    $req = $this->_obj_base->prepare("select date_visite, commentaires,
    duree from visite where id_medecin = :par_id");
    $req->execute(array(':par_id' => $id_medecin));
    if ($req->rowCount() > 0) {
        $result = $req->fetchAll();
        // Statut OK + mise en forme des caractéristiques au format demandé
        // [appel de la méthode convertir_donnees]
        $this->reponse($this->convertir_donnees($result), 200);
    } else {
        // Si aucun enregistrement, retour avec le statut 404 (not found)
        $this->reponse('', 404);
    }
}
```

Dans le fichier apirest.service.ts, j'ai ajouté une méthode qui permet d'appeler l'API REST pour obtenir les visites du médecin dont l'id est passé en paramètre :

```
public getVisitesMedecin(id: string): Observable<Visite[]> {
    return this.http.get<Visite[]>(this.apiUrl + 'medecins/' + id + '/visites')
    .pipe(
        tap(res => console.log('Lecture des visites OK' + res)),
        catchError(this.handleError<Visite[]>('Lecture des visites', []))
    );
}
```

L'API REST sera donc appelée avec une méthode HTTP de type GET et avec l'URI :

<http://localhost/gsb suivi/act/medecins/n/visites>

où n représente l'identifiant du médecin pour lequel on souhaite obtenir les caractéristiques des visites

J'ai ensuite modifié le fichier detailmedecin.page.html pour indiquer que lors du clic sur le bouton « visites », on appellera la route /visitesmedecin/ avec l'identifiant du médecin sur lequel on a cliqué :

```
<ion-content>
  <ion-card>
    <ion-card-header>
      <ion-card-title>{{ unMedecin.nom }} {{ unMedecin.prenom }}</ion-card-title>
    </ion-card-header>
    <ion-card-content>
      téléphone : {{ unMedecin.telephone }} <br>
      secteur : {{ unMedecin.secteur }}
      <br><br>
      <ion-button [routerLink]="'/visitesmedecin/' + unMedecin.id">Visites</ion-button>
    </ion-card-content>
  </ion-card>
</ion-content>
```

Dans le fichier detaildelegue.page.ts et j'ai ajouté les instructions entourées en rouge :

```
import { Component, OnInit } from '@angular/core';
import { Visite } from '../models/visite';
import { ApirestService } from '../api.rest.service';
import { ActivatedRoute } from '@angular/router';
```

Import de la classe visite

Import de la classe de mon service

```
@Component({
  selector: 'app-visitesmedecin',
  templateUrl: './visitesmedecin.page.html',
  styleUrls: ['./visitesmedecin.page.scss'],
})
```

Ajout d'un attribut qui contient les visites du médecin

```
export class VisitesmedecinPage implements OnInit {
```

```
  lesVisites: Visite[] = [];
```

Ajout d'un attribut qui pour gérer la route

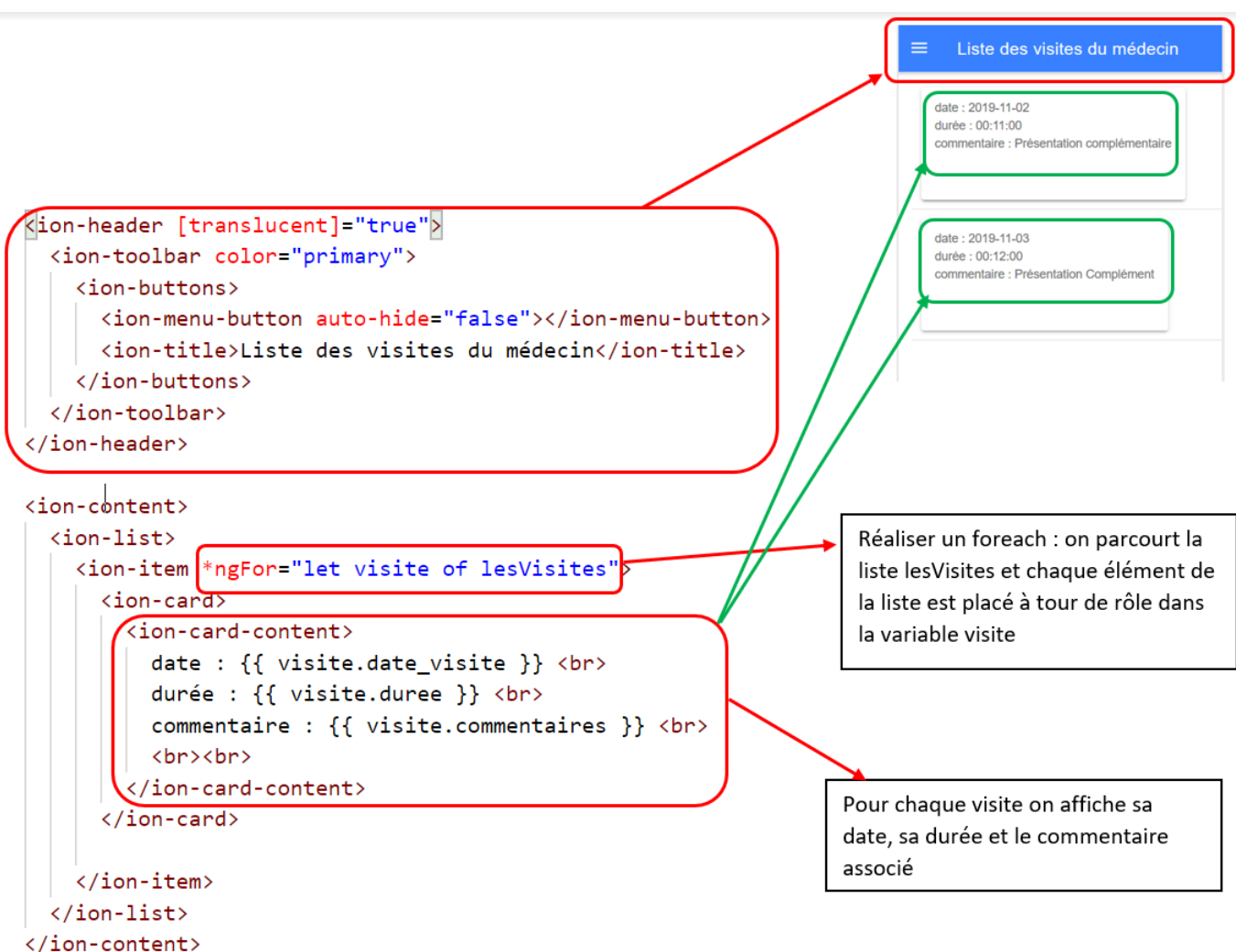
```
  constructor(private monServ: ApirestService, private activatedRoute: ActivatedRoute) { }
```

Ajout d'un attribut qui contient un objet de la classe de mon service

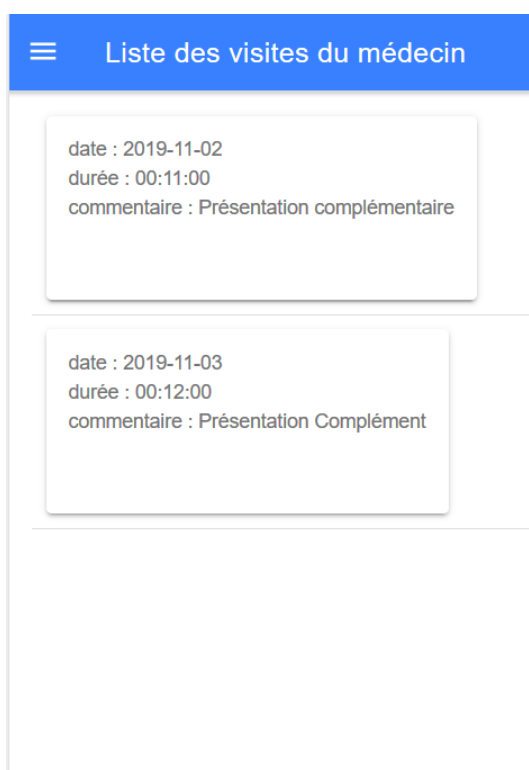
```
  ngOnInit() {
    this.monServ.getVisitesMedecin(this.activatedRoute.snapshot.params.id).subscribe(
      value => {
        this.lesVisites = value;
      },
      error => {
        console.log('Récupération liste visites impossible');
      }
    );
  }
}
```

Appelle de la méthode de mon service qui appellera l'api rest pour avoir la liste des visites du médecin

Enfin pour afficher les informations à l'utilisateur, j'ai ajouté les instructions suivantes dans le fichier **visitesmedecin.page.html** :



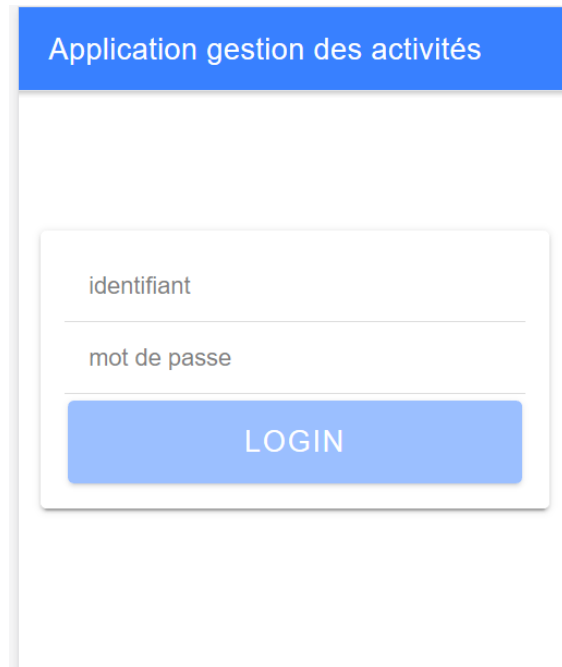
L'utilisateur obtient alors la page suivante :



## C- Page de connexion à l'application

L'application est accessible aux délégués médicaux, pour cela ils doivent se connecter avec un identifiant et un mot de passe.

Voici la page permettant de se connecter :



J'ai commencé par modifier l'API REST pour gérer la nouvelle URI permettant d'obtenir l'identifiant du délégué correspondant à un couple identifiant/mot de passe transmis :

```
case "delegates" :
switch($this->_methode_req)

// méthode GET (consultation de données)
case "GET" :
// la variable $id existe-t'elle ?
if (isset($id) == true) {
// OUI (cela signifie que l'on souhaite obtenir les caractéristiques d'un délégué
// on appelle la méthode lire_un_delegue en lui donnant le numéro du délégué à consulter
$this->lire_un_delegue($id);
} else {
if(isset($this->_donnees_req['identifiant'])==true && isset($this->_donnees_req['motdepasse'])==true)
{
$this->connexion_un_delegue($this->_donnees_req['identifiant'], $this->_donnees_req['motdepasse']);
}
else{
// NON (cela signifie que l'on souhaite lire tous les délégués)
// on appelle la méthode lire_les_delegates
$this->lire_les_delegates();
}
}
break;
default :
```

La nouvelle URI permettant d'obtenir l'identifiant du délégué est la suivante :

**<http://localhost/gsb-suivi-act/delegates?identifiant=.....&motdepasse=....>**

Puis j'ai écrit la méthode connexion\_un\_delegue qui fait un select dans la base données pour obtenir l'identifiant du délégué connecté dont l'identifiant et le mot de passe sont passés en paramètre. S'il existe un enregistrement alors on convertit les données, on les retourne et on retourne le code statut indiquant que tout s'est bien passé. S'il n'y a aucun enregistrement, alors on retourne le code statut « aucun contenu ». Si le paramètre transmis est vide alors on retourne le code statut « mauvaise requête ».

```
private function connexion_un_delegue($identifiant, $mot_passe)
{
    if (empty($identifiant) == false && empty($mot_passe) == false ) {
        // le paramètre contient une valeur
        // on prépare et on exécute la requête permettant d'obtenir les caractéristiques du délégué médical
        $req = $this->_obj_base->prepare("select id from delegue_medical
            where identifiant_connexion = :par_ident and mot_passe = :par_mot_passe");
        $req->bindValue(':par_ident',$identifiant, PDO::PARAM_STR);
        $req->bindValue(':par_mot_passe',$mot_passe, PDO::PARAM_STR);
        $req->execute();
        // $req->execute(array(':par_ident' => $identifiant,':par_mot_passe' => $mo_passe ));

        if ($req->rowCount() == 1) {
            $result = $req->fetch();
            // Status OK + mise en forme des caractéristiques au format demandé (appel de la méthode convertirDonnees)
            $this->reponse($this->convertir_donnees($result),200);
        } else {
            // Si aucun enregistrement, statut "No Content"
            $this->reponse('', 404);
        }
    } else {
        // le paramètre transmis est vide: status Bad Request
        $this->reponse('', 400);
    }
}
```

J'ai ajouté une méthode verifconnexion dans le service apirest qui va permettre d'appeler l'API REST afin d'obtenir l'identifiant du délégué à partir de son identifiant et de son mot de passe :

```
public verifConnexion(ident: string, motdepasse: string): Observable<any> {
    const params = new HttpParams()
        .set('identifiant', ident)
        .set('motdepasse', motdepasse);
    return this.http.get<any>(this.apiUrl + 'delegates', {params})
        .pipe(
            tap(res => console.log('connexion' + res)),
            catchError(this.handleError<Delegue>('Verif connexion'))
        );
}
```

Création d'une collection params qui contient les paramètres http à passer à l'API REST

On passe la collection de paramètres lors de l'appel de l'API REST

On indique « any » comme type retourné par l'API REST car cela peut être un id si le couple pseudo/mot de passe saisi est valide ou la valeur null si le couple pseudo/mot de passe saisi n'existe pas

Puis j'ai demandé à ionic de générer la page de connexion avec la commande suivante : **ionic generate page connexion**.

Pour qu'un message d'erreur s'affiche s'il y a une erreur d'identifiant ou de mot de passe, j'ai créé un Toast.

Tout d'abord, j'ai ajouté un attribut de type ToastController dans mon constructeur :

```
constructor(private monService: ApirestService, private router: Router, private monToast: ToastController, ) {
```

Puis j'ai créé la méthode asynchrone toastErreurConnexion. Cette méthode crée un Toast avec un message d'information et l'affiche au milieu de l'écran pendant trois secondes:

```
async toastErreurConnexion() {  
  const toast = await this.monToast.create ({  
    message: 'Vos informations de connexion sont incorrectes',  
    duration : 3000,  
    position: 'middle'  
  });  
  toast.present();  
}
```

J'ai ensuite ajouté la méthode login exécutée lors de la soumission du formulaire dans le fichier connexion.page.ts :

```
login(form) {  
  this.monService.verifConnexion(form.value.ident, form.value.password).subscribe((res) => {  
    if (res != null) {  
      this.router.navigateByUrl('accueil');  
    } else {  
      this.toastErreurConnexion();  
    }  
  });  
}
```

On passe l'identifiant et le mot de passe saisis sur le formulaire à la méthode verifConnexion du service apirest

Si la valeur retournée (res) par la méthode verifConnexion du service n'est pas null cela signifie que les données saisies sont valides.

On affiche la page d'accueil.

Crée et affiche le Toast si le couple identifiant/mot de passe est incorrect

J'ai ensuite modifié le fichier de routing de l'application pour qu'au lancement de l'application le formulaire de connexion s'affiche :

```
{ path: '', redirectTo: 'connexion', pathMatch: 'full' },
```

Enfin, j'ai réalisé le formulaire de connexion dans la page connexion.page.html :

```
<ion-header>
  <ion-toolbar color="primary">
    <ion-title>Application gestion des activités</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content padding>
  <form #form="ngForm" (ngSubmit)="login(form)">
    <ion-grid style="height: 100%">
      <ion-row justify-content-center align-items-center style="height: 60px"></ion-row>
      <ion-row justify-content-center align-items-center style="height: 100%">
        <ion-col size="8" offset="2">
          <ion-card ion-text-center>
            <ion-card-content>
              <div padding>
                <ion-item>
                  <ion-input name="ident" type="text" placeholder="identifiant" ngModel required></ion-input>
                </ion-item>
                <ion-item>
                  <ion-input name="password" type="password" placeholder="mot de passe" ngModel required></ion-input>
                </ion-item>
              </div>
              <div padding>
                <ion-button size="large" type="submit" [disabled]="form.invalid" expand="block">Login</ion-button>
              </div>
            </ion-card-content>
          </ion-card>
        </ion-col>
      </ion-grid>
    </form>
  </ion-content>
```

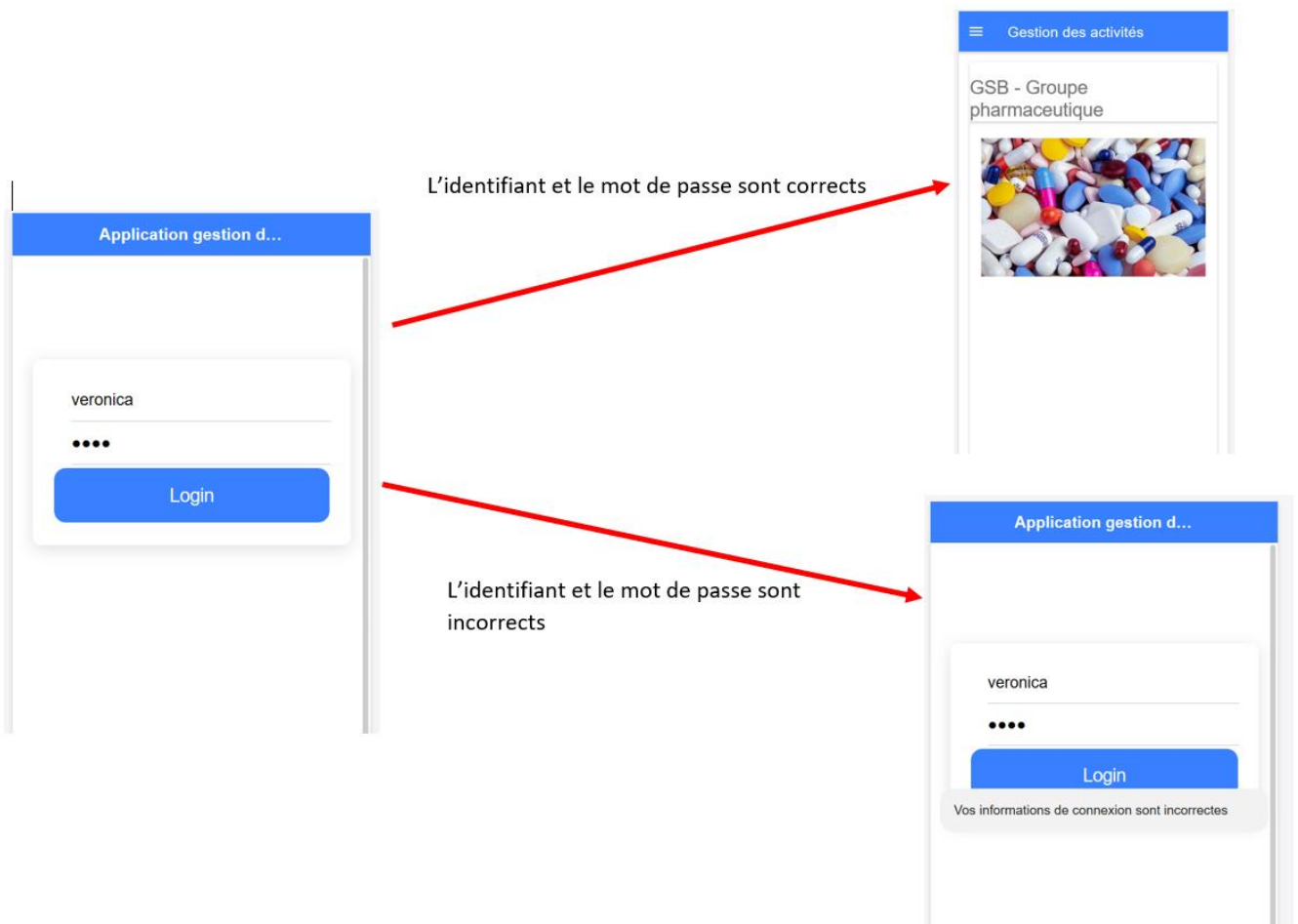
Lorsque l'on appuie sur le bouton de type `submit`, la méthode `login` située dans le fichier `connexion.page.ts` sera appelée. Elle recevra le formulaire.

Champs de saisie du formulaire

Permet au fichier `.ts` de la page de pouvoir lire les champs

Bouton de type `submit` pour soumettre le formulaire

Voici les pages qu'obtient le délégué médical en fonction du scénario :



Pour que l'utilisateur n'ait pas à s'authentifier à chaque fois qu'il lance l'application, il ne sera pas invité à ressaisir son identifiant et son mot de passe s'il s'est authentifié il y a moins d'une heure.

Pour cela j'ai ajouté une nouvelle classe `gesact` qui va décrire l'objet contenant les variables à stocker en mémoire. Cet objet contiendra l'identifiant du délégué qui s'est authentifié ainsi que la date et l'heure à laquelle il s'est authentifié pour la dernière fois. Voici son contenu :

```
export class Gesact {
  id: number;
  datederniereconnexion: Date;
}
```

J'ai ensuite modifié le fichier `localstore.service.ts` afin de créer les méthodes qui permettront de créer l'objet dans la mémoire du smartphone, de le fournir à une page extérieure et de le supprimer. Voici le contenu du fichier :

```
import { Injectable } from '@angular/core';
import { Storage } from '@ionic/storage';
import { Gesact } from '../models/gesact';
```

```
@Injectable({
  providedIn: 'root'
})
```

```
export class LocalstoreService {
  key: string = 'gesact';
  constructor(public store: Storage) { }
```

La clé utilisée pour accéder à l'objet stocké en mémoire est déclarée et on lui affecte la valeur `gesact`

```
  public async lire() {
    return await this.store.get(this.key);
  }
```

Méthode qui permet d'obtenir la valeur (enregistrée sous forme d'objet) correspond à la clé.

```
  public async creer(gesact: Gesact) {
    return await this.store.set(this.key, gesact);
  }
```

Méthode qui permet d'enregistrer l'objet associé à la clé

```
  public async supprimer() {
    return await this.store.remove(this.key);
  }
```

Méthode qui permet de supprimer l'objet associé à la

Puis j'ai modifié le fichier `connexion.page.ts`.

J'ai ajouté les attributs suivants :

```
export class ConnexionPage implements OnInit {
  constructor(private monService: ApirestService, private router: Router,
    private monToast: ToastController, private monServiceStore: LocalstoreService) { }
  public gesacts: Gesact;
```

Attribut de type `LocalstoreService`

Objet qui sera créé en mémoire



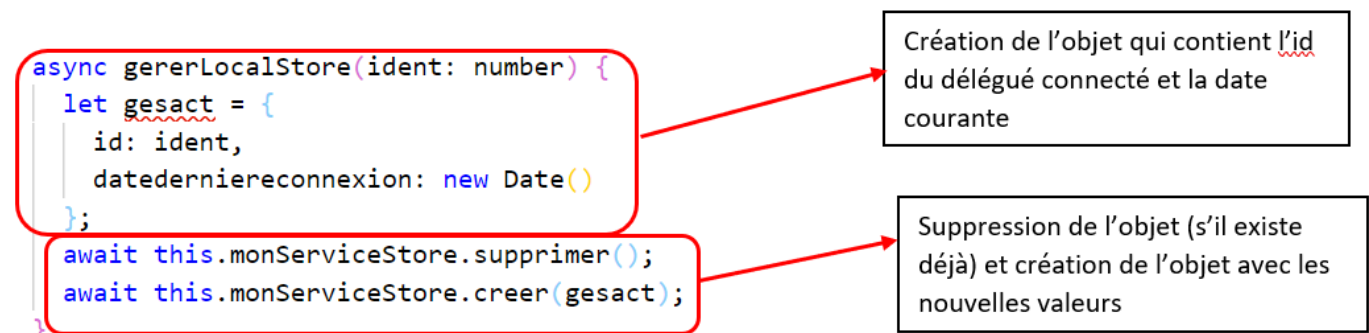
J'ai ensuite créé la méthode asynchrone `ngOnInit`. Cette méthode est appelée au chargement de la page et va réaliser les actions suivantes :

- demander au service `localStorage` de lui donner l'objet stocké dans la mémoire du smartphone pour la clé `gesact`
- si l'objet existe, récupération de la date/heure de dernière connexion
- récupérer la date et l'heure courante
- calculer l'écart en minutes entre la date et l'heure courante et la date de dernière connexion enregistrée dans l'objet stocké dans la mémoire du smartphone
- si l'écart est inférieur à 60, la connexion a alors eu lieu il y a moins de 60 minutes : on affiche la page d'accueil à l'utilisateur il n'a donc pas à s'authentifier.

Voici ses instructions :

```
async ngOnInit() {  
  // lecture des valeurs stockées dans le local storage pour la clé gesact  
  this.gesacts = await this.monServiceStore.lire();  
  // la local storage contient-elle une entrée pour la clé gesacts ?  
  if (this.gesacts != null) {  
    // la local storage contient une entrée pour la clé gesacts  
  
    // récupération de la date et l'heure de dernière connexion stockée (conversion au format date)  
    let dateLocalStorage = new Date (this.gesacts.datederniereconnexion);  
  
    // récupération de la date et l'heure courante  
    let dateHeureActuel = new Date();  
  
    // calcul du nombre de minutes écoulées depuis la dernière connexion  
    let dureeConnexionMinute = (dateHeureActuel.getTime() - dateLocalStorage.getTime()) / 60000;  
  
    // si la connexion a eu lieu il y a moins de 60 minutes (1h), alors on  
    // affiche directement la page d'accueil( l'utilisateur n'aura pas besoin de s'authentifier)  
    if (dureeConnexionMinute < 60) {  
      this.router.navigateByUrl('accueil');  
    }  
  }  
  // la local storage n'a pas d'enregistrement pour la clé gesact : c'est donc  
  // la page login qui sera affichée  
}
```

Puis, j'ai créé une méthode asynchrone chargée d'enregistrer dans la local storage (mémoire du smartphone) l'objet contenant l'identifiant et la date de connexion :



J'ai ensuite modifié le contenu de la méthode login de la page connexion, afin d'appeler la méthode présentée ci-dessus :

```
login(form) {  
  this.monService.verifConnexion(form.value.ident, form.value.password).subscribe((res) => {  
    if (res != null) {  
      this.gererLocalStore(res.id);  
      this.router.navigateByUrl('accueil');  
    } else {  
      this.toastErreurConnexion();  
    }  
  });  
}
```

On peut voir que l'identifiant du délégué et la date de connexion sont bien enregistrés dans la local storage :

id	id	key	value
1	1	gesact	{"id":"1","datederniereconnexion":"2020-05-06T17:28:35.478Z"}

Annotations:

- Clé de l'enregistrement ajouté (points to 'gesact')
- L'identifiant (points to 'id' in the value)
- La date et l'heure de dernière connexion ici : 6 mai 2020 17 :28 :35 (points to the timestamp in the value)

Si l'utilisateur quitte l'application Ionic puis la relance, il n'aura pas à s'authentifier tant qu'il ne sera pas 18:28:35, il sera directement dirigé sur la page d'accueil.