
BBensors

Application mobile en milieu néonatal



Thèse de Bachelor présentée par

Monsieur Ermal FEKA

pour l'obtention du titre Bachelor of Science HES-SO en

**Ingénierie des technologies de l'information avec orientation en
Logiciels et systèmes complexes**

Septembre 2016

Professeur HES responsable TB

Florent Gluck

Mandant

HUG HEIG-VD

Cahier des charges

INGÉNIERIE DES TECHNOLOGIES DE L'INFORMATION ORIENTATION – LOGICIEL ET SYSTEMES COMPLEXES APPLICATION MOBILE EN MILIEU NEONATAL
--

Descriptif :

Dans le cadre du projet de recherche bbsensors "Un système de monitoring sans fil, portable et miniaturisé, pour la détection et l'analyse de plausibilité d'évènements vitaux en milieu néonatal" en collaboration avec la HEIG-VD et l'unité de néonatalogie des HUG, nous désirons déterminer la présence d'anomalies cardiaques et respiratoires chez les nouveaux nés sur la base de signaux en provenance d'un électrocardiogramme et d'un oximètre dans le but d'avertir, en temps réel, le personnel soignant. Le système actuel se compose d'un système embarqué portable (côté bébé) sur lequel sont connectés des capteurs (ECG, oximètre) ainsi que d'un deuxième système embarqué connecté au dispositif de monitoring de l'hôpital sur lequel sont affichés les signaux vitaux. Les systèmes embarqués communiquent via le protocole Bluetooth Low Energy. Le système embarqué connecté au moniteur réalise le traitement de signal afin de détecter les anomalies potentielles et transmet les flux de données provenant des différents capteurs au moniteur.

Le travail à réaliser se compose de deux parties. La première partie est la conception d'une application mobile servant de moniteur de données en provenance des bébés monitorés. Le but de cette application sera l'affichage des signaux en provenance des patients (ECG, respiration, et données oximétriques) ainsi que la gestion des alarmes. L'application devra être multi-plateforme, afficher les signaux en temps réel, permettre de naviguer dans l'historique des signaux, de zoomer sur ceux-ci. Il est important que l'application soit le plus simple et facile à utiliser par le personnel infirmier. De même, l'association patient/moniteur devra être aussi plus plug-and-play que possible. La deuxième partie du projet repose sur la mise en place de l'architecture serveur. Chaque système embarqué fera office de serveur sur lequel les clients mobiles viendront se connecter. Il s'agira de mettre en place le stockage des signaux vitaux, de gérer les connections clients, le protocole de transmission, etc., le tout aussi simplement et efficacement que possible. Ce projet impliquera une interaction avec le personnel des HUG et à l'issue du travail, le démonstrateur développé sera présenté au personnel de néonatalogie pour validation.

Travail demandé :

- Application mobile :
 - Affichage des signaux vitaux en temps réel (ECG, respiration, taux d'oxygénation)
 - Zooms, gestion de l'historique
 - Gestion des alarmes
 - Système d'association/configuration patient/moniteur plug-and-play
- Composante serveur :
 - Architecture de communication client (smartphone) ↔ serveur (système embarqué)
 - Chaque smartphone doit pouvoir afficher les données venant de n'importe quel serveur
 - Modèle de stockage des signaux et stockage des données.
 - Choix du protocole de communication et du format de données afin de maximiser les performances
 - Application serveur
- Analyse de performances (wifi mauvaise qualité, beaucoup de clients, etc.)
- Démonstrateur.

Candidat :

M. Feka Ermal

Filière d'études : ITI

Professeur(s) responsable(s) :

Gluck Florent

En collaboration avec : HUG
Travail de bachelor soumis à une convention
de stage en entreprise : **non**
Travail de bachelor soumis à un contrat de
confidentialité : **non**

Timbre de la direction



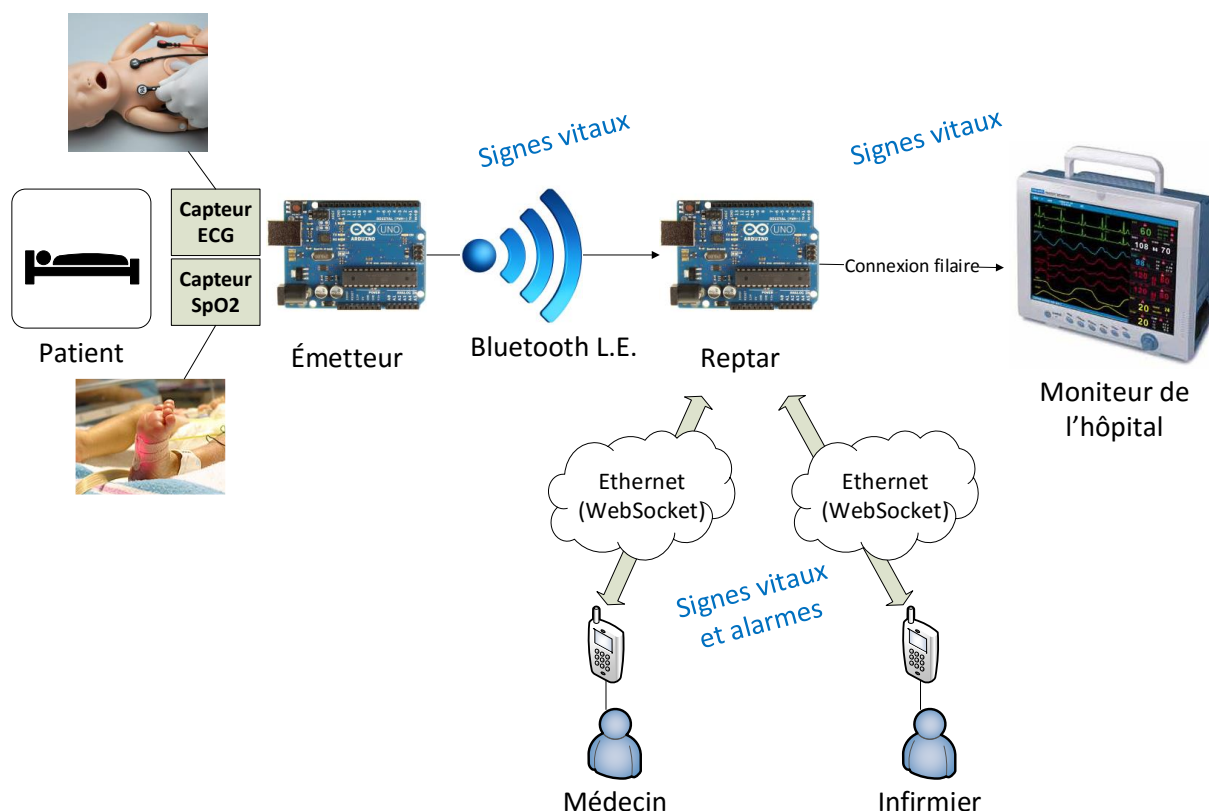
Résumé

Ce travail de diplôme consiste à créer une application mobile permettant de suivre et détecter les anomalies cardiaques et respiratoires chez les nouveau-nés prématurés.

De nos jours, les nouveau-nés sont encore des patients critiques et demandent un suivi rigoureux et constant. Pour aider les médecins et le personnel soignant dans leur métier, nous avons créé une application leur permettant de visualiser l'état des patients depuis un appareil mobile.

Cette application permet d'avoir un suivi des patients, de pouvoir visualiser en temps réel les signaux affichés sur le moniteur ECG de l'hôpital et de pouvoir analyser une alerte passée. Elle avertit les médecins et le personnel soignant au plus vite en cas de problème grave afin de minimiser les risques de complication.

Cette application est de type hybride. Elle peut être déployée sur plusieurs systèmes mobiles ainsi que sur le store.



Remerciements

Dans le cadre du travail de bachelor, je voudrais dans un premier temps, remercier M. Gluck pour m'avoir suivi et conseillé tout au long de ce projet. Ce fut une aide et un soutien primordial qui m'a permis de concrétiser mon travail.

Je tiens aussi à remercier M.Cavat pour m'avoir orienté et aidé à de nombreuses reprises.

Pour finir, je voudrais remercier ma copine Tina, qui m'a supporté durant tout ce projet. Malgré mes sauts d'humeurs et mon manque de patience dû au stress, elle m'a soutenu à chaque moment.

Table des matières

1. Introduction	1
2. Vue d'ensemble	2
2.1 Schéma global du système	2
2.2 Schéma détaillé.....	3
2.3 Objectifs de l'application	4
3. Architecture de l'application	5
4. Choix des technologies	7
4.1 Frameworks mobiles	7
4.1.1 Ionic.....	8
4.1.2 AngularJS.....	10
4.1.3 Cordova	15
4.2 Communication	19
4.2.1 Technologies possibles.....	19
4.2.2 Technologie utilisée	23
4.3 Scanner	24
4.3.1 Technologies possibles.....	24
4.3.2 Technologie utilisée	25
4.4 Stockage.....	26
4.4.1 Technologies possibles.....	26
4.4.2 Technologie utilisée	27
4.5 Notification	28
4.6 Background mode.....	29
4.7 Historique alerte	30
4.7.1 Technologies possibles.....	30
4.7.2 Technologie utilisée	31
5. Description fonctionnelle de l'application.....	32
5.1 Home	33
5.2 Ajout d'un patient.....	35
5.3 Modification des données d'un patient	36
5.4 Liste des alertes	37
5.5 Historique	38
5.6 Moniteur ECG	39
6. Mise en œuvre	40
6.1 Architecture logicielle.....	40

6.2 Vues et contrôleurs	41
6.2.1 Index & app	41
6.2.2 HomeCtrl	42
6.2.3 addpatientCtrl	43
6.2.4 modifListeCtrl	44
6.2.5 infoAlerteCtrl	44
6.2.6 historiqueCtrl	45
6.2.7 moniteurCtrl	47
6.3 Services	48
6.3.1 Liste Patient	48
6.3.2 BDD	49
6.3.3 Socket	50
6.3.4 Background	52
6.3.5 Alertes	53
7. Mesures & Performances	54
7.1 Consommation de l'application	54
7.2 Réception des données	56
8. Conclusion	57
Bibliographie	58
Annexe	60
A.A Installation et création d'un projet IONIC	60
A.B Commande d'installations des différents plugins utilisés	62
A.C Librairie Zingchart et SocketIO	62
A.D Code source	63

Table des illustrations

Figure 1 Schéma bloc global du projet	2
Figure 2 Communications entre le Reptar et le Smartphone	3
Figure 3 Schéma bloc fonctionnel	5
Figure 4 Diagramme en couche d'une application Ionic.....	7
Figure 5 Schéma représentant les différentes interactions entre le modèle, la vue et le contrôleur. Référence	10
Figure 6 Utilisation de l'attribut ng-show	11
Figure 7 Exemple d'injection de dépendance	11
Figure 8 Exemple de gestion des routes	12
Figure 9 Exemple d'utilisation de la dépendance \$ionicView.enter.....	13
Figure 10 Voici une représentation en couche des vues, services et contrôleurs.....	14
Figure 11 Les différentes couches de l'application sur le téléphone.....	15
Figure 12 Les différents plugins disponibles par Cordova suivant la version du système du téléphone.	17
Figure 13 Autorisation d'une fonctionnalité sur le téléphone.....	18
Figure 14 Communication WebSocket.....	22
Figure 15 Représentation d'un code-barre et d'un QRCode	25
Figure 16 Notification générée par le plugin Background	28
Figure 17 Représentation d'un graphique sur Zingchart	31
Figure 18 L'ensemble des vues de l'application	32
Figure 19 Vue Home et son menu.....	34
Figure 20 Scanner QRCode - Code-barre	35
Figure 21 Vue Add. Patient.....	35
Figure 22 Vue Modif. Patient - Informations générales et Seuils des alertes.....	36
Figure 23 Vue Liste des Alertes	37
Figure 24 Visualisation du second graphique	38
Figure 25 Vue Historique.....	38
Figure 26 Vue Moniteur ECG.....	39
Figure 27 Architecture de l'application	40
Figure 28 Récupération des données du code-barre.....	43
Figure 29 Suppression d'une option du menu de Zingchart	45
Figure 30 Gestion du zoom avec zingchart	46
Figure 31 Exemple d'utilisation de la méthode zingchart.exec() avec zoomin.....	46
Figure 32 Exemple d'utilisation de la méthode zingchart.exec() avec zoomto	46
Figure 33 Conversion des données reçues des capteurs de binaires en string	51
Figure 34 Conversion des données alertes de binaires en string	51
Figure 35 Activation du mode background et initialisation du texte de la notification	52
Figure 36 Graphique Temps d'envoi des données.....	56
Figure 37 Représentation des trois types d'application possible de départ avec Ionic	60

Tableau 1 Avantage et désavantage de la technologie AJAX.....	19
Tableau 2 Avantage et désavantage de la technologie SocketIO.....	20
Tableau 3 Avantage et désavantage de la technologie UDP.....	30
Tableau 4 Avantage et désavantage de la technologie WebSocket.....	22
Tableau 5 Avantage et désavantage de la technologie NFC.....	24
Tableau 6 Avantage et désavantage de la technologie scan Code-barre QRCode.....	25
Tableau 7 Avantage et désavantage de la technologie LocalStorage.....	26
Tableau 8 Avantage et désavantage de la technologie SQLite.....	27
Tableau 9 Avantage et désavantage des technologies utilisées pour la notification.....	28
Tableau 10 Avantage et désavantage des technologies utilisées pour le mode Background...	29
Tableau 11 Avantage et désavantage de la technologie Chart.....	30
Tableau 12 Avantage et désavantage de la technologie Zingchart.....	31
Tableau 13 Consommation des vues pendant 1 heure.....	54
Tableau 14 Mesure temps d'envoi.....	56

Tableau des acronymes

ACRONYME	DEFINITION
ECG	Electrocardiogramme
SPO2	Saturation pulsée en oxygène
FR	Fréquence respiratoire
FC	Fréquence cardiaque
HUG	Hôpital universitaire de Genève
HEIG-VD	Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud
BDD	Base de données
HUG	Hôpital universitaire de Genève
AJAX	Asynchronous JavaScript and XML
BLE	Bluetooth Low Energy
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
API	Application Programming Interface
HTML	Hypertext Markup Language
SDK	Software development kit
MVC	model-view-controller
URL	Uniform Resource Locator
NFC	Near Field Communication
RFID	Radio Fréquence Identification
SQL	Structured Query Language
CLI	Commande line interface

1. Introduction

De nos jours, les nouveau-nés prématurés sont encore des patients critiques et demandent un suivi rigoureux et constant.

Dans le cadre du projet de recherche "*BBSensors : un système portable sans fils et miniaturisé pour le monitoring de signaux vitaux en milieu néonatal*" en collaboration avec les HUG et la HEIG-VD, nous désirons déterminer la présence d'anomalies cardiaques et respiratoires chez les nouveau-nés prématurés. Ces anomalies se basent sur l'analyse des signes vitaux en provenance d'un électrocardiogramme et d'un oxymètre.

Actuellement, en cas d'alerte d'un patient, le moniteur ECG sonne pour prévenir le personnel soignant. Par contre, cet appareil se trouve dans la chambre du bébé et ne peut être déplacé. Comme le personnel soignant est toujours en mouvement (dû à leur travail), le temps de réaction est variable.

L'étape clé du projet est de transmettre les valeurs mesurées par les différents capteurs ainsi que les anomalies vers un périphérique mobile utilisé par les médecins et le personnel soignant. Il s'agit donc de développer une application mobile hybride (iOS & Android) qui communique avec un système embarqué.

Le périphérique reçoit les données traitées et les alertes. Ces alertes permettent d'informer le personnel soignant qu'un patient est en danger (fréquence cardiaque trop basse, arrêt cardiaque...) pour que le temps de réaction soit le plus bref possible.

La création de l'application hybride est développée via le Framework¹ Ionic. Elle est multi plateforme et déployable sur le store.

¹ Structure logicielle permettant de créer les fondations d'un programme

2. Vue d'ensemble

Pour avoir une idée du projet dans sa globalité, voici une description en bloc décrivant les différents points du projet :

2.1 Schéma global du système

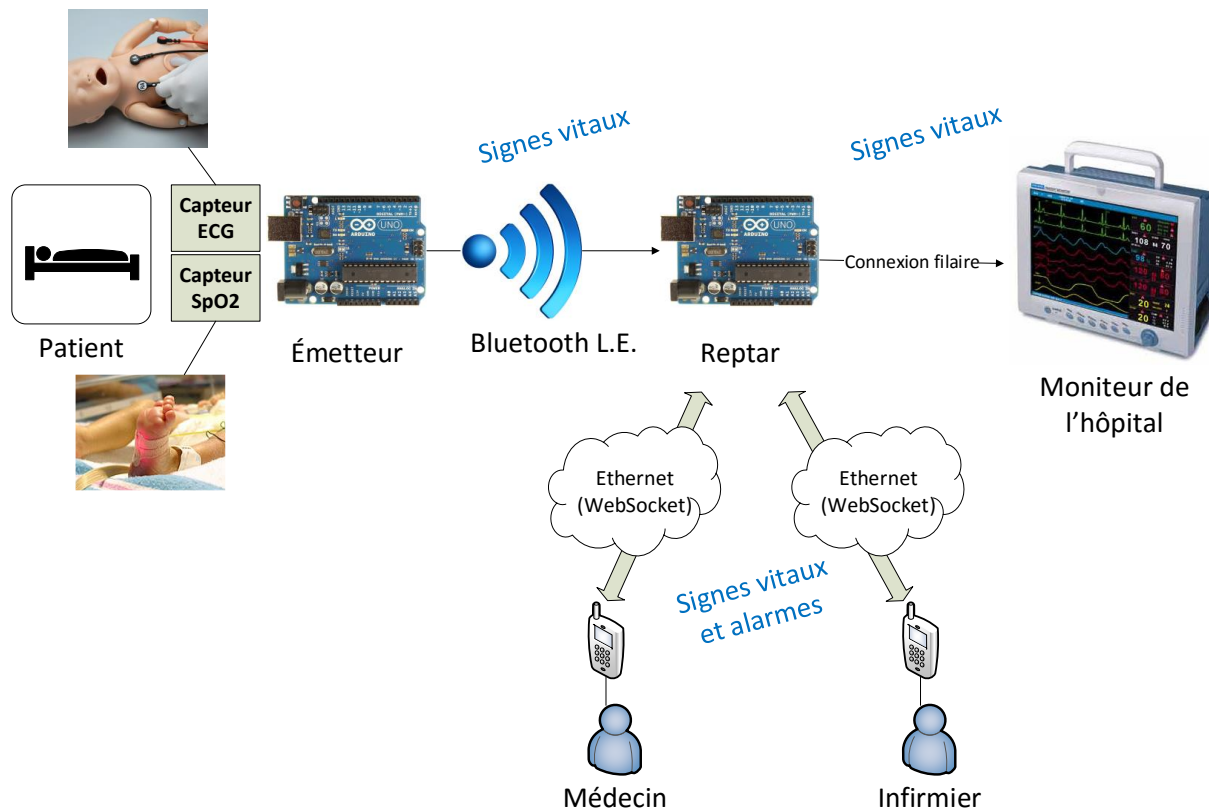


Figure 1 Schéma bloc global du projet

La figure 1 présente une vue globale du système. Le patient est connecté à un capteur ECG et SPO2. Ces capteurs sont reliés à un module (émetteur) accroché à la couche du bébé. Une fois les données des capteurs reçues, il les envoie via le BLE au second module (Reptar²). Le module (Reptar) traite ces données et les transmet au moniteur de l'hôpital et aux smartphones. Ainsi, le personnel soignant et les médecins peuvent avoir accès aux informations des patients à travers le wifi de l'hôpital. Lorsqu'un patient présente un problème, le module (Reptar) envoie des alertes au téléphone. A la réception de ces alertes, le téléphone sonne et vibre.

² Plateforme embarquée reconfigurable pour la formation et la recherche

2.2 Schéma détaillé

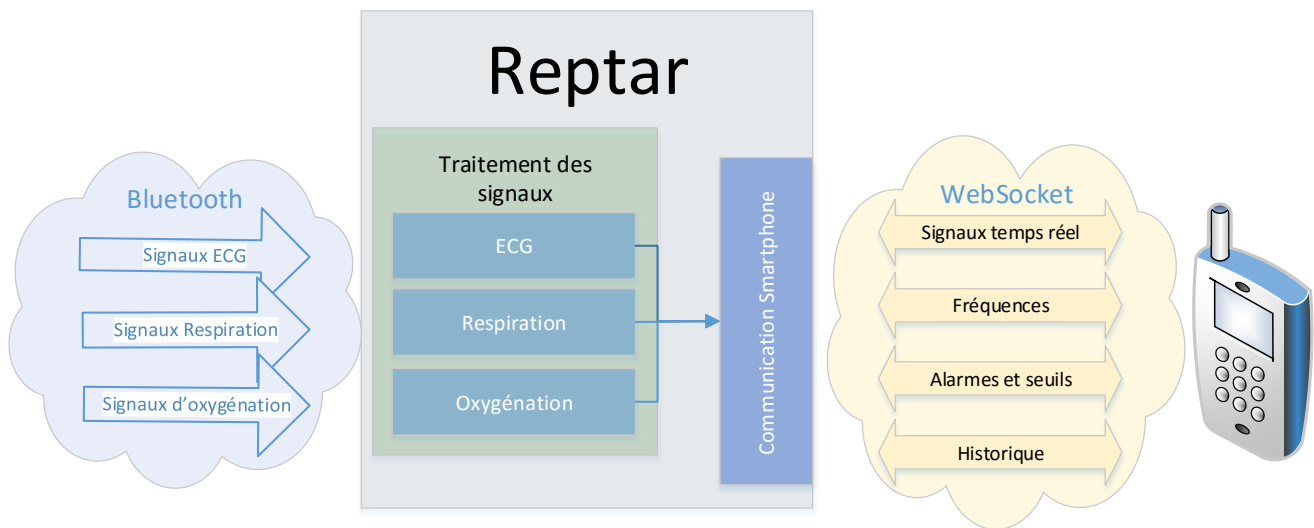


Figure 2 Communications entre le Reptar et le Smartphone

La Figure 2 décrit la communication des données entre le Reptar et le smartphone.

Une fois les données des différents capteurs reçues via BLE, le Reptar traite ces données et les envoie dans des canaux de communication websocket bidirectionnelle au smartphone.

- Le premier tunnel (signaux temps réel) permet d'envoyer les signaux des capteurs à une fréquence de 50[Hz].
- Le deuxième (fréquences) permet d'envoyer les données fréquentielles des capteurs (fréquence cardiaque, fréquence respiratoire et taux d'oxygène dans le sang) chaque seconde.
- Le troisième (alarmes et seuils) transmet les seuils d'alarme du patient. Ces seuils désignent la valeur minimale et maximale de la valeur des capteurs. Si celle-ci dépasse le seuil, une alerte est envoyée au téléphone.

Chaque capteur a ses propres seuils et ils peuvent être changés dans le téléphone. Une fois modifiés, ils sont renvoyés au Reptar pour qu'ils soient mis à jour. Il permet aussi d'adresser un message d'alerte signalant au personnel soignant et au médecin qu'un patient a un problème.

- Le dernier (historique) permet d'envoyer une partie du signal passé. Ce signal représente les données quelque minute avant et après l'apparition d'une alerte. Cela permet d'analyser celui-ci pour mieux comprendre le problème du patient.

2.3 Objectifs de l'application

Le but de ce projet est de concevoir une application mobile hybride permettant aux médecins et aux personnels soignants d'avoir une vision de l'état des patients.

Il est demandé que l'application remplisse les points suivants :

- Pouvoir communiquer avec le module Reptar qui transmet toutes les données des différents capteurs du patient.
- Permettre l'ajout, la modification ou encore la suppression des patients suivis.
- Mémoriser les patients sur le téléphone pour qu'il ne soit pas nécessaire de les rajouter à chaque activation de celle-ci.
- Afficher les données des différents capteurs connectés sur le patient comme les fréquences et les signaux, qui doivent être en temps réel comme sur le moniteur ECG de l'hôpital.
- Prévenir (sonore, vibration, visuelle) lorsqu'un patient présente des symptômes ou est en danger.
- Permettre la visualisation d'un signal d'alerte passé. Il servira au médecin et au personnel soignant de pouvoir analyser le signal d'une ancienne alerte.
- L'application doit être la plus simple et Plug&Play³ possible pour que les utilisateurs n'aient pas de difficulté à la compréhension et à l'utilisation. Le plug and Play est sollicité pour gérer l'association moniteur <-> patient.

³ Permet d'indiquer une fonctionnalité utilisable avec un minimum d'intervention (branche et utilise)

3. Architecture de l'application

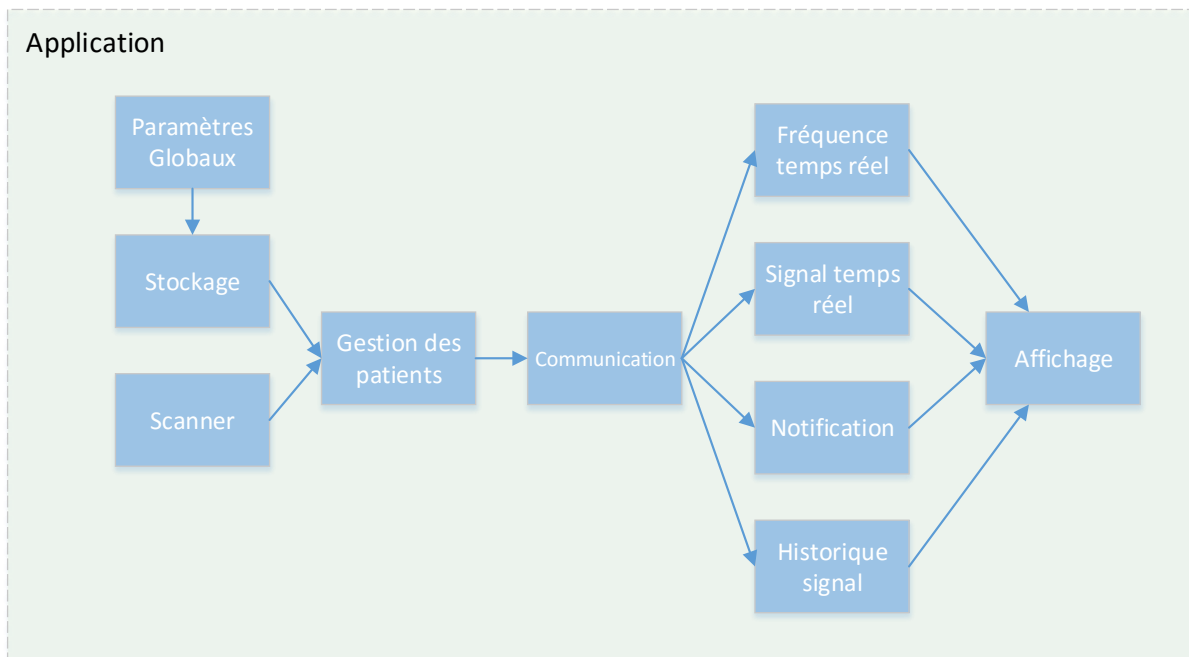


Figure 3 Schéma bloc fonctionnel

La figure 3 représente les différents blocs fonctionnels de l'application mobile BbSensors. Voici une explication des différents blocs.

- **Gestion des patients** permet d'ajouter, modifier ou supprimer un patient. Pour rendre ceci plus Plug&Play, j'utilise un scanner code-bar et QRCode.
- **Scanner** donne la possibilité d'ajouter ou modifier un patient, il faut scanner un code-barre pour avoir le nom et le numéro de la chambre et scanner un QRCode pour obtenir l'ip et le port du serveur (Reptar).
- **Stockage** mémorise les informations des patients afin qu'elles ne soient pas perdues à la fermeture de l'application.
- Comme l'application utilise des **paramètres globaux** concernant tous les patients, il faut aussi les mémoriser pour qu'à la fermeture de l'application ces informations ne se suppriment pas.
- **Communication** effectue la liaison entre le smartphone et le module(Reptar) pour recevoir les différentes données de celui-ci (fréquence temps réel, signal temps réel, notification, historique signal). A chaque ajout d'un nouveau patient, une nouvelle connexion est établie.

- **Fréquence temps réel** permet d'obtenir les fréquences (fréquence cardiaque, fréquence respiratoire et taux d'oxygène dans le sang) du patient en temps réel. Ces données, une fois reçues, sont affichées et permettent d'avoir une idée de l'état du patient.
- **Signal temps réel** reçoit les données à afficher sous forme de graphiques. Il y a un signal pour chaque donnée (fréquence cardiaque, fréquence respiratoire et taux d'oxygène dans le sang) et ces signaux sont affichés en temps réel comme sur le moniteur ECG de l'hôpital.
- **Notification** reçoit des messages d'alerte indiquant qu'un patient est en danger. A chaque réception de ces messages, le téléphone informe l'utilisateur de manière visuelle, sonore et en vibrant qu'une alerte a été reçue.
- **Historique signal** analyse le signal d'une alerte passée. Une tranche du signal (X [min] avant et X [min] après l'alerte) est affichée. Cela permet une analyse de celui-ci pour une meilleure compréhension du problème.

4. Choix des technologies

Ce chapitre présente les différentes technologies étudiées pour répondre au besoin du projet. Il indique les points forts et faibles de chacune d'entre elles. Ainsi, cela sélectionne les plus adaptées.

4.1 Frameworks mobiles

Pour la création de l'application, l'utilisation du Framework **Ionic** [1] m'a été imposée. Ce framework permet de concevoir une application hybride multiplateforme (Android, iPhone, ...). Pour cela, il utilise **AngularJS** [2] pour l'aide à la création de l'application web et **Cordova** [3] pour accéder aux fonctionnalités natives du téléphone (Bluetooth, camera, ...).

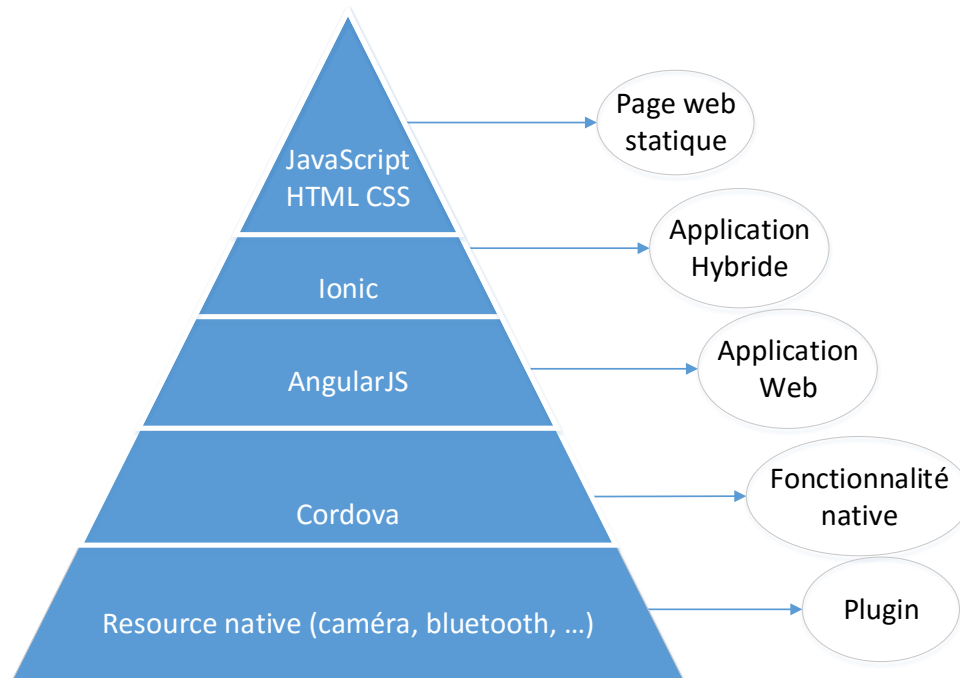


Figure 4 Diagramme en couche d'une application Ionic

La figure 4 représente une application développée sur Ionic et donne un aperçu hiérarchisé des frameworks utilisés.

4.1.1 Ionic

Description

Ionic est considéré comme un framework front-end⁴ de développements d'applications mobiles hybrides. Il permet de créer une application compatible avec un grand nombre de systèmes d'exploitation smartphone. Il dispose d'un certain nombre de composants CSS et librairie permettant de créer rapidement une application ressemblant à une application native. (référence : [4])

Il a aussi la capacité d'accéder aux éléments système du téléphone (camera, notification). C'est ce qu'on appelle une application hybride.

Application hybride

Il existe trois types d'applications. Les applications natives, les applications web et les applications hybrides.



- ✓ Accès aux API natives
- ✓ Distribution possible sur le Store
- ✗ Multi plateformes

Une application native est une application programmée dans le langage du système (java pour Android, Objectif C pour ios). Cela assure une performance maximale de celle-ci, car toutes les ressources sont utilisées. Il gère les API⁵ natives comme le Bluetooth, la camera ou les notifications, il peut distribuer l'application sur le Store, mais ne peut être utilisé que pour le système pour lequel il a été créé.



- ✗ Accès aux API natives
- ✗ Distribution possible sur le Store
- ✓ Multi plateformes

Une application Web est une application programmée en JavaScript, HTML et CSS. L'application est générée sur une WebView ce qui lui permet d'être utilisée par toutes les plateformes. Par contre, elle ne peut être distribuée dans le Store et n'a pas la possibilité de gérer les API natives. Ces applications sont utilisées pour des petits programmes demandant peu de ressource, car la performance est minime.

⁴ Basé sur le visuel et le client

⁵ Application Programming Interface

Une application hybride est un mélange entre une application native et Web. C'est une application programmée en JavaScript, HTML et CSS qui est exécuté dans la WebView du navigateur par défaut du téléphone. Elle est moins performante qu'une application native, mais peut aussi gérer les API natives comme le Bluetooth, la caméra ou les notifications. Elle donne aussi la possibilité de distribuer l'application sur le Store et peut être utilisée par un grand nombre de plateformes (Android, IOS, ...). Moins performante qu'une application native, elle reste une bonne solution pour développer rapidement une application.



Les outils d'Ionic :

L'avantage d'Ionic est le nombre étonnant d'outils et services pour l'aide au développement. Ces outils concèdent l'ajouter facile des dispositifs d'analyse à toutes applications, de compiler les applications sans avoir les SDK installés sur la machine, ou bien de mettre à jour l'application sans passer par le store. (référence : [5])

- Ionic dispose de son propre gestionnaire de commandes. Ce gestionnaire est manipulé par ligne de commande et elle donne la capacité de démarrer un projet rapidement, lancer un serveur local de développement pour aider dans la programmation, compile une application ou encore lance un émulateur de téléphone.
- Une fois l'application finie, Ionic met à disposition une plateforme accordant le partage de l'application. Elle est très utile parce qu'elle offre la possibilité de distribuer l'application à de tierces personnes qui pourront l'expérimenter ainsi que donner leurs avis. Cela assure une phase de test supplémentaire avant le déploiement final sur le store. Il est nécessaire d'installer IonicView pour y accéder. Par contre, il faut spécifier à qui on veut la partager. Ce n'est pas comme le store ou tout le monde y a accès.
- Un émulateur en ligne qui se rafraichit à chaque modification du code. Il permet alors de visualiser directement l'application et de la tester. Cependant, Il ne peut gérer les fonctionnalités natives du téléphone. Toute application utilisant cette fonctionnalité provoquera une erreur dans l'émulateur.

4.1.2 AngularJS

AngularJS est l'un des frameworks adoptés par Ionic afin de soutenir le développement de l'application web. Voici différents points importants le concernant. (référence : [6])

Description

AngularJS est un framework JavaScript open source⁶ développé par Google. Il peut créer une application web plus rapidement qu'en JavaScript, HTML et CSS, notamment grâce aux nombreuses directives proposées.

Il adhère aussi au patron de conception MVC, utilise le concept d'injection de dépendance et de gestion des routes.

Patron de conception

AngularJS adhère au patron de conception MVC . Il effectue une séparation entre la vue (ce que l'utilisateur voit), le modèle (les données du système) et le contrôleur (la "colle" entre la vue et le modèle).

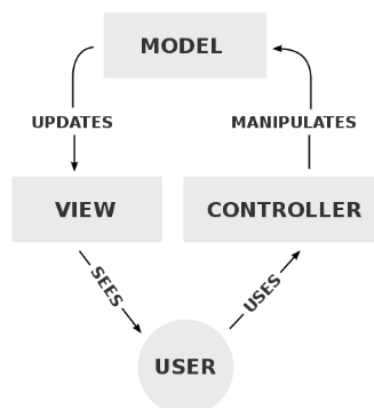


Figure 5 Schéma représentant les différentes interactions entre le modèle, la vue et le contrôleur. Référence⁷

La figure 5 présente le patron de conception MVC. L'utilisateur exploite le contrôleur pour manipuler les données (model) qui sont affichées dans la vue. AngularJS définit la dépendance « \$scope⁸ » comme étant le modèle. C'est elle qui effectue la liaison entre la vue et le contrôleur. Il ne peut y avoir qu'un seul \$scope entre un contrôleur et une vue.

Par exemple, si on prend l'élément « Input » :

```
<input type="text" ng-value="nom" ng-model="nom">
```

Il suffit d'ajouter « nom » à \$scope pour mettre la vue à jour.

```
$scope.nom = 'Ermal';
```

⁶ Code source ouvert

⁷ <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>

⁸ Espace, périmètre et portée de variable

Les directives

Dans l'exemple de l'input cité précédemment, nous pouvons constater la présence d'attribut (ng-value, ng-model) dans la balise HTML. Ces attributs sont appelés des directives. On peut les reconnaître parce qu'elles sont précédées de « ng ». Ils permettent de définir des fonctionnalités aux balises HTML sans avoir à implémenter le code.

Dans le cas de « input », ces directives signalent que la valeur à afficher est la valeur qui se trouve dans « ng-value » et si la valeur est modifiée par la vue, elle peut être récupérée dans « ng-model ».

Pour mieux comprendre, prenons l'exemple d'une liste que nous ne voulons pas afficher continuellement. Il suffit d'utiliser la directive « ng-show » pour remédier à ce problème.

La figure 6 donne un exemple d'utilisation de « ng-show ».

```
<ion-list ng-show = "afficher">
  <ion-item ng-repeat="item in items">
    Hello, {{item}}!
  </ion-item>
</ion-list>
```

Figure 6 Utilisation de l'attribut ng-show

Comme nous pouvons le voir sur la figure 6, il suffit d'ajouter la directive « ng-show » dans la balise HTML de « ion-list » pour indiquer que la liste est visible que si la variable « afficher » est à *true* (vrai). Bien sûr, il faut qu'« afficher » soit déclarée dans le contrôleur.

```
$scope.afficher = true;
```

L'injection de dépendances :

Comme expliqué ci-dessus, AngularJS utilise la dépendance \$scope pour effectuer la liaison entre la vue et le contrôleur. Par contre, il existe un grand nombre d'autres dépendances mis à disposition. Pour que ces dépendances soient faciles à mettre en place, AngularJS adhère à l'injection de dépendance.

L'injection de dépendances offre la possibilité d'instancier automatiquement les modules dont on a besoin. Il suffit d'appeler les dépendances voulues et AngularJS se charge de l'injecter et de l'instancier automatiquement.

Par exemple, pour utiliser « \$scope », il suffit d'indiquer en paramètre d'entrée du contrôleur qu'il doit l'utiliser.

```
app.controller("MyController", function($scope, $timeout, Socket){
  ...
});
```

Figure 7 Exemple d'injection de dépendance

Comme on peut le voir sur la figure 7, la dépendance `$scope` et `$timeout` sont injectés au contrôleur. Par contre, on peut aussi constater qu'il est possible d'injecter des services. Les services sont expliqués dans la suite du rapport.

La gestion des routes

Les routes permettent de gérer les transitions entre les différentes vues de l'application. AngularJS crée des applications web, cela implique que chaque vue doit avoir une URL attribuée. Ces URL permettent à la gestion des routes de connaître les différentes vues possibles. La gestion des routes est propre à AngularJS.

AngularJS possède un fichier nommé `app.js` dans lequel sont spécifiées toutes les vues que l'application possède. C'est dans ce fichier que sont indiqués l'URL de chaque vue et le contrôleur relié à celui-ci. Il aura l'avantage d'être centralisé au sein d'un seul fichier.

La figure ci-dessous donne un exemple de gestion des routes.

```
app.config(function($stateProvider,$urlRouterProvider) {
    $stateProvider.state("home", {
        url: "/home",
        templateUrl: "templates/home.html",
        controller: "homeCtrl"
    })
    $stateProvider.state("addPatient", {
        url: "/addPatient",
        templateUrl: "templates/addPatient.html",
        controller: "addPatientCtrl"
    })
    $stateProvider.state("modifList", {
        url: "/modifList/:mIndex",
        templateUrl: "templates/modifList.html",
        controller: "modifListCtrl"
    })

    // Etat par défaut (la page de démarrage)
    $urlRouterProvider.otherwise("/home")
});
```

Figure 8 Exemple de gestion des routes

Comme nous pouvons le voir sur la figure 8, nous constatons que l'application comprend trois vues possibles. Ces vues possédant chacune leur URL attribuée et sont toutes liées à un contrôleur.

Le contrôleur

Le contrôleur permet de spécifier à la vue les actions et manipulations possibles. Il comporte les différentes classes et méthodes essentielles au bon déroulement de l'application. Chaque contrôleur possède son propre « \$scope » lui permettant de faire la liaison entre la vue et lui-même. Par contre, le contrôleur est actif tant que la vue liée à celui-ci et affichée.

Avec AngularJS, comme le contrôleur est attaché à la vue, à chaque appel de celle-ci le contenu du contrôleur est rechargé. Étant une perte de temps, Ionic a décidé de fonctionner différemment.

Ionic garde une trace de l'historique de navigation. On peut considérer cela comme un travail en couche. Il permet, dans le cas où nous voulons revenir à la vue précédente, de ne pas recharger son contrôleur. Par contre, s'il vous est bénéfique d'exécuter une partie de code à chaque fois que la vue est affichée, il existe la dépendance « \$ionicView ».

« \$ionicView » gérer les vues actives. Elle comprend la méthode « enter » qui permet d'appeler une fonction à chaque appel de la vue. Vous avez ci-dessous un exemple d'implémentation.

```
// Va être appelé à chaque appel de la vue
$scope.$on('$ionicView.enter', function() {
    ...
});
```

Figure 9 Exemple d'utilisation de la dépendance \$ionicView.enter

Les services

Les services sont des singletons, c'est-à-dire des instances uniques d'objets. Cela signifie que ces objets peuvent être utilisés tout au long de l'application contrairement à un contrôleur qui lui est détruit à la fermeture de la vue. Son rôle est de fournir un ensemble de tâches utile au bon fonctionnement de l'application. (référence : [6])

AngularJS met à disposition un certain nombre de services comme \$route ou \$http et donne la possibilité de les créer. Afin d'utiliser le service dans le contrôleur, il suffit de l'injecter comme expliquer dans le chapitre "Injection des dépendances".

La figure ci-dessous illustre les différentes liaisons entre les services, les contrôleurs et les vues.

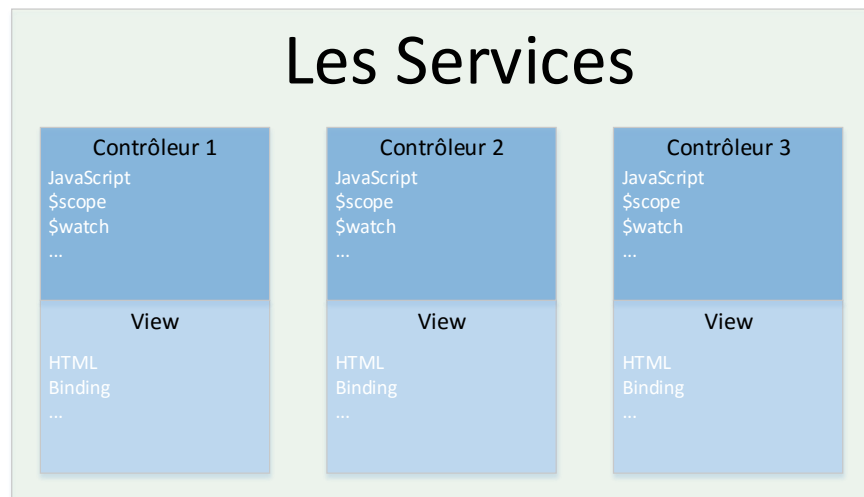


Figure 10 Voici une représentation en couche des vues, services et contrôleurs

Comme nous pouvons le voir sur la figure 10, chaque vue est liée à un contrôleur. Par contre, le service peut être injecté à tous les contrôleurs. Ce qui leur permet de tous utiliser le même service.

4.1.3 Cordova

Cordova est un framework utilisé par Ionic pour accéder aux APIs natives du téléphone et donc créer une application hybride. Voici une explication de ce framework.

Description

Cordova, autrefois appelé Apache Callback ou PhoneGap, est un framework open source de développement d'application mobile hybride. Comme Ionic, il utilise AngularJS pour la création d'application web. Il gère l'exécution de l'application sur le téléphone en faisant la liaison entre l'application web et la webView du navigateur par défaut et gère aussi les fonctionnalités natives du téléphone. Les applications qui en résultent sont de ce fait hybride.

La figure ci-dessous représente les différentes couches d'une application Ionic.

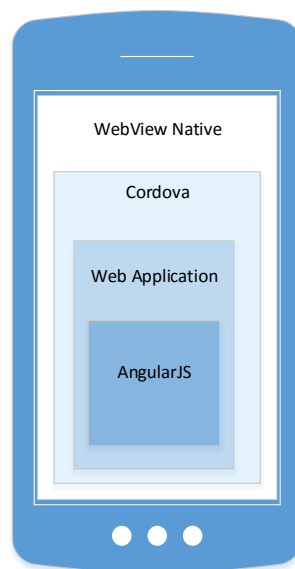


Figure 11 Les différentes couches de l'application sur le téléphone

Comme nous pouvons le voir sur la figure ci-dessus, Ionic est une surcouche de Cordova et utilise toutes les fonctionnalités de celui-ci. Par contre, grâce à ces différents composants CSS et librairie, il rend les applications qui en résulte plus faciles à programmer, plus élégant et qui réagisse comme des applications natives.

Pour pouvoir accéder aux fonctionnalités natives de l'appareil, Cordova met en place la notion de plugin. Ces plugins sont un mélange de JavaScript HTML CSS et de langage natif à la plateforme utilisée.

Les plugins

Un plugin est un moyen d'accès aux fonctionnalités natives de l'appareil. C'est un mélange de JavaScript HTML CSS pour visualiser ce que l'on veut faire dans l'application web et de langage natif à la plateforme utilisée pour que le téléphone comprenne et exécute la requête qui lui a été demandé. (référence : [7])

Par contre, les plugins mis à disposition par Cordova ne sont pas tous compatibles avec toutes les plateformes mobiles. La figure 12 montre un certain nombre de plugins et leur compatibilité avec les différentes Platform. On peut constater que tous les plugins sont compatibles avec les plateformes Android et IOS qui sont les leaders dans le domaine de la téléphonie mobile.

Les plugins présents sur internet ne sont pas tous créés par Cordova, il est aussi possible de créer son propre plugin ou encore de récupérer le plugin d'un autre programmeur qui l'aura partagé dans son git. La commande pour ajouter un plugin à son application est la suivante :

```
cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-device.git
```

Un plugin se compose de trois parties majeures :

- Le fichier **plugin.xml** qui permet de définir et de configurer le plugin
- Le dossier **src/** qui comporte le code natif spécifique aux différentes plateformes
- Le dossier **www/** qui contient l'API JavaScript que le plugin exposera à la vue

Tous les plugins de l'application se trouvent dans le dossier **plugins/**. Par défaut Ionic ajoute les plugins « device », « console » et « keyboard » à la création du projet.

Dans le fichier **plugin.xml**, il y a toutes les configurations des différentes plateformes supportées par le plugin et les autorisations des fonctionnalités. Pour ce qui est du contenu **src/**, le langage de programmation n'est plus JavaScript, mais le langage natif à la plateforme.

Le fichier **fetch.json** qui se trouve dans le dossier **plugins** contient tous les plugins utilisés par l'application. C'est grâce à ce fichier que les plugins sont téléchargés et mis en place à la compilation de l'application ou à l'ajout d'une nouvelle plateforme.

	Amazon- fireos	Android	blackberry10	Firefox OS	iOS	Ubuntu	wp8 (Windows Phone 8)	Windows (8.0, 8.1, 10, Téléphone 8.1)	paciarelli
Cordova CLI	✓ Mac, Windows, Linux	✓ Mac, Windows, Linux	✓ Mac, Windows	✓ Mac, Windows, Linux	✓ Mac	✓ Ubuntu	✓ Windows	✓	X
Embedded WebView	✓ (voir détails)	✓ (voir détails)	X	X	✓ (voir détails)	✓	X	X	X
Plug-in Interface	✓ (voir détails)	✓ (voir détails)	✓ (voir détails)	X	✓ (voir détails)	✓	✓ (voir détails)	✓	X
Accéléromètre	✓	✓	✓	✓	✓	✓	✓	✓	✓
BatteryStatus	✓	✓	✓	✓	✓	X	✓	✓ * Windows Phone 8.1 seulement	✓
Appareil photo	✓	✓	✓	✓	✓	✓	✓	✓	✓
Capture	✓	✓	✓	X	✓	✓	✓	✓	X
Boussole	✓	✓	✓	X	✓ (3 G +)	✓	✓	✓	✓
Connexion	✓	✓	✓	X	✓	✓	✓	✓	✓
Contacts	✓	✓	✓	✓	✓	✓	✓	partiellement	X
Dispositif	✓	✓	✓	✓	✓	✓	✓	✓	✓
Événements	✓	✓	✓	X	✓	✓	✓	✓	✓
Fichier	✓	✓	✓	X	✓	✓	✓	✓	X
Transfert de fichiers	✓	✓	✓ * Ne pas soutenir onprogress ni abandonner.	X	✓	X	✓ * Ne pas soutenir onprogress ni abandonner.	✓ * Ne pas soutenir onprogress ni abandonner.	X
Géolocalisation	✓	✓	✓	✓	✓	✓	✓	✓	✓
Mondialisation	✓	✓	✓	X	✓	✓	✓	✓	X
InAppBrowser	✓	✓	✓	X	✓	✓	✓	utilise les iframe	X
Media	✓	✓	✓	X	✓	✓	✓	✓	✓
Notification	✓	✓	✓	X	✓	✓	✓	✓	✓
SplashScreen	✓	✓	✓	X	✓	✓	✓	✓	X
Barre d'État	X	✓	X	X	✓	X	✓	✓ 8.1 de Windows Phone uniquement	X
Stockage	✓	✓	✓	X	✓	✓	✓ localStorage & indexedDB	✓ localStorage & indexedDB	✓
Vibration	✓	✓	✓	✓	✓	X	✓	✓ * Windows Phone 8.1 seulement	X

Figure 12 Les différents plugins disponibles par Cordova suivant la version du système du téléphone.

Les autorisations

Il est possible que certaines fonctionnalités réclament une autorisation depuis la mise à jour d'Android (version 6.0.1). Cette mise à jour est toute récente et certains plugins ne se sont pas encore adaptés. Pour ajouter ou modifier une autorisation dans un plugin, il faut modifier le fichier « AndroidManifest.xml » qui se situe dans le répertoire :

plugins\phonegap-plugin-barcodescanner\src\android\LibraryProject

Pour autoriser l'application à utiliser ces fonctionnalités sans changer le code natif du plugin, il faut se rendre dans les paramètres du téléphone, gestionnaire d'application, sélectionner l'application, autorisation puis cocher la fonctionnalité à autoriser comme indiquer en figure 13.

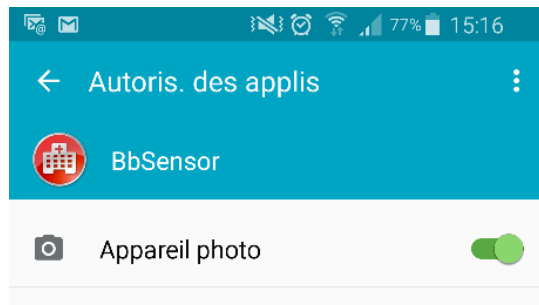


Figure 13 Autorisation d'une fonctionnalité sur le téléphone

4.2 Communication

En termes de communication des données du patient (fréquence cardiaque, fréquence respiratoire, ...) ainsi que des alertes entre le Reptar et le smartphone, les solutions potentielles retenues sont AJAX, SocketIO, WebSocket ou SocketUDP.

4.2.1 Technologies possibles

AJAX

AJAX est un ensemble de technologie visant à effectuer une communication entre un navigateur et un serveur. La différence principale entre elle et une application web classique est qu'à chaque modification de la vue, seules les données modifiées sont renvoyées par le serveur. Tandis qu'une application classique recharge toute la vue. Il permet ainsi de limiter le nombre de données envoyé par le serveur à chaque modification de la page internet. (référence : [8])

Par contre, pour recevoir des données continuellement depuis le serveur, le client doit effectuer une requête pour chaque demande. Ceci peut être gênant, car il y a un risque de goulot d'étranglement⁹.

Le tableau ci-dessous présente les privilèges et les inconvénients de la technologie AJAX.

Avantage	Désavantage
Communication avec serveur sans recharger toute la page	Demande une requête pour chaque demande de données

Tableau 15 Avantage et désavantage de la technologie AJAX

⁹ Dépassement des limites de performance.

SocketIO

Socket.io [4] est une librairie Node.js [5]¹⁰ utilisée pour mettre en place une communication en TCP quasi temps réel bidirectionnelle synchrone entre un serveur et un ou plusieurs clients.

Il se base sur plusieurs techniques différentes pour effectuer la communication. De base, Socket.IO utilise webSocket. Par contre si le navigateur ne le supporte pas, Socket.IO va automatiquement choisir la meilleure technique mise à disposition supportée par celui-ci.

Les différentes techniques sont :

- WebSocket
- Adobe Flash Socket
- AJAX long polling
- AJAX multipart streaming
- Forever Iframe
- JSONP Polling

Le protocole TCP permet d'informer le client d'un changement d'état de la connexion et d'assurer l'envoi de toutes les données. En cas de perte des paquets, le serveur le renvoie tant que le client ne l'a pas acquitté.

Par contre, ce protocole n'est pas vraiment adapté pour une communication temps réel. Une communication UDP l'est plus, car elle ne prend pas en charge la gestion de la bonne réception des données. Cela rend la communication TCP plus lente que UDP.

Le tableau ci-dessous présente les avantages et les inconvénients de la technologie SocketIO.

Avantage	Désavantage
Facile à utiliser	Trop lourd pour notre utilité
Différentes méthodes prises en charge pour la communication des données	Communication TCP et non UDP
Adaptée à tous les navigateurs	
Permet une communication quasi temps réel	
Permet une communication bidirectionnelle	

Tableau 16 Avantage et désavantage de la technologie SocketIO

¹⁰ Node.js est une plateforme logicielle libre et événementielle en JavaScript orientée vers les applications réseau

Socket UDP

Contrairement à TCP, UDP¹¹ est un protocole qui ne gère pas la connexion et ne garantit pas la réception des données. Cela signifie qu'il est impossible de savoir si la connexion a été établie ou rompue. Elle ne permet pas non plus de savoir si toutes les données envoyées ont bien été reçues. Il y a donc un risque de perte des données.

Ce protocole est bien adapté aux données pour lesquelles la perte d'un paquet n'est pas importante, par exemple, la vidéo streaming. Il permet d'avoir un protocole plus léger que TCP et donc une communication plus rapidement. (référence : [9])

Pour que Ionic puisse utiliser le protocole UDP dans la transmission des données, il doit appliquer le plugin « chrome-apps-sockets-udp [10] » qui va exploiter les ressources système du téléphone. Ce plugin est obligatoire parce qu'il est impossible d'utiliser ce protocole avec une application Web. C'est pour cela que nous avons décidé de ne pas utiliser cette technologie vue que nous voulons par la suite déployer l'application sur un site web.

Le tableau ci-dessous présente les privilèges et les inconvénients de cette technologie.

Avantage	Désavantage
Permet une communication quasi temps réel	Communication sans garantie de réception des paquets
Envoi broadcast	Pas de gestion de connexion
Latence moins importante que TCP	Impossible à utiliser si on veut faire un site internet

Tableau 17 Avantage et désavantage de la technologie UDP

¹¹ User Datagram Protocol = protocole de datagramme utilisateur

WebSocket

WebSocket est une technologie permettant une communication en TCP quasi temps réel entre un client et un serveur. Cette communication est bidirectionnelle. Cela signifie qu'elle utilise un seul canal (chemin) pour envoyer et recevoir les données. Elle est aussi asynchrone, offrant la possibilité d'envoyer plusieurs paquets sans devoir attendre l'accusé de réception. Il met aussi en place une connexion persistante qui permet l'envoi de plusieurs paquets sans devoir faire de requête.

La figure ci-dessous illustre une communication websocket comprenant une connexion persistante (référence : [11])

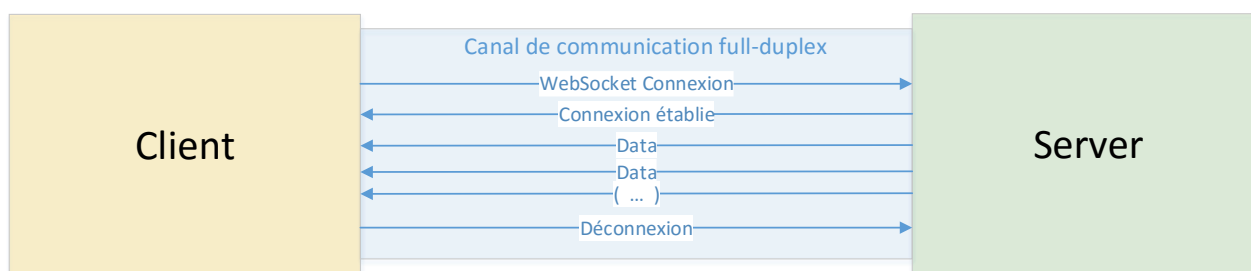


Figure 14 Communication WebSocket

Comme Socket.IO, websocket utilise le protocole TCP. Il permet de prendre en charge la gestion de la connexion et assure l'envoi de toutes les données (pas de perte de paquet).

Ne requérant pas d'en-tête pour chaque envoi de données (seulement à l'initialisation), la bande passante est fortement réduite, ce qui permet une communication quasi temps réel.

Cette technologie est simple à mettre en place. Elle ne sollicite pas d'installation de plugin contrairement à la technologie socket UDP. Il suffit d'ajouter la librairie au programme. De plus, cette bibliothèque est facile à utiliser de par le fait qu'elle ne comprend que trois méthodes principales. Elle offre la possibilité de créer un socket, recevoir et envoyer des paquets, sans avoir à les gérer.

Le tableau ci-dessous présente les privilèges et les inconvénients de cette technologie.

Avantage	Désavantage
Facile à utiliser	Latence plus importante que UDP
Moins lourd que Socket.io, car ne prennent pas en charge les autres techniques de communication.	
Permet une communication quasi temps réel	
Permet une communication bidirectionnelle	

Tableau 18 Avantage et désavantage de la technologie Websocket

4.2.2 Technologie utilisée

Suite à la création de petites applications tests des différentes technologies proposées ci-dessus, la plus appropriée pour le transfert de données est **WebSocket**. Elle autorise un transfert de donnée continue comparé à AJAX qui lui impose une requête pour chaque nouvelle donnée. De plus, elle est moins lourde que SocketIO vue que WebSocket n'utilise pas toutes les techniques de communication que propose SocketIO. Puis, elle effectue une gestion de la connexion et des paquets, ce qu'il aurait fallu mettre en place si nous avions utilisé UDP. Enfin, pour recourir à WebSocket, il suffit de télécharger et incorporer le package « socket-io.js » [12] dans le projet.

4.3 Scanner

A l'ajout d'un patient, l'application doit connaître l'IP et le port du Reptar sur lequel il va se connecter. C'est pourquoi nous avons décidé de rendre cette manipulation la plus plug&play possible, car ces informations sont méconnues auprès des médecins et du personnel soignants. Cela permet aussi d'effectuer cette manipulation sans avoir à mémoriser ces données.

Les méthodes permettant cela sont les suivantes. La première est l'utilisation de la technologie NFC du téléphone, en ajoutant à chaque patient un tag NFC. La seconde est l'emploi de l'appareil photo pour scanner un code-barre afin de récupérer les données du patient et un QRCode pour les informations du Reptar.

4.3.1 Technologies possibles

NFC

Est une technologie de communication sans fil à courte portée et haute fréquence dérivée de la technologie RFID. Il permet l'échange d'informations entre un téléphone et un tag NFC d'une distance d'environ 10 cm. De plus, les tags ne nécessitent aucune source d'énergie et sont faciles à mettre en place. De nos jours, la plupart des téléphones possèdent cette technologie. Par contre, il faut que chaque patient ait un tag NFC où sont stockées les informations de celui-ci. (référence : [13])

Pour utiliser cette technologie dans l'application, il faut employer le plugin « nfc.plugin » [14].

Le tableau ci-dessous présente les avantages et les inconvénients de la technologie NFC.

Avantage	Désavantage
Rend l'ajout des patients plus Plug & Play	Oblige à avoir des Tags NFC (achat, maintien)
Dispositif passif	

Tableau 19 Avantage et désavantage de la technologie NFC

Scan Code-barre QRCode

Etant donné que l'hôpital ne souhaite pas avoir du matériel supplémentaire (tag NFC) et qu'il utilise déjà les codes-barres pour l'identification des patients, la seconde solution est d'utiliser un scanner de Code-barre et de QRCode.

La figure ci-dessous illustre d'un code-barre et d'un QRCode.



Figure 15 Représentation d'un code-barre et d'un QRCode

Comme illustrées sur la figure ci-dessus, le code-barre et le QRCode sont des images cryptées contenant des données. Pour ajouter ou modifier des patients, il faut scanner le code-barre depuis l'application afin d'avoir les renseignements du patient (nom, chambre). Puis, scanner le QRCode pour avoir les informations du Reptar (ip, port).

Concernant la création des codes-barres et des QRCode, il existe des sites gratuits qui le proposent [15].

Pour cette technologie, Cordova propose le plugin « BarcodeScanner » [16] qui offre la possibilité de scanner les codes-barres et les QRCode.

Le tableau ci-dessous présente les avantages et les inconvénients de cette technologie.

Avantage	Désavantage
Rend l'ajout des patients plus Plug & Play	Oblige à avoir des codes-barres et des QRcodes (maintient)
Création des QRcode et des codes-barres gratuitement	

Tableau 20 Avantage et désavantage de la technologie scan Code-barre QRCode

4.3.2 Technologie utilisée

Pour une question de simplicité et de non-ajout de matériel supplémentaire, nous avons décidé de choisir l'option Scan Code-barre et QRCode. Cela n'oblige pas l'achat de tags NFC, car la création de QRCode est gratuite et l'hôpital utilise déjà les codes-barres pour identifier les patients.

4.4 Stockage

L'application doit garder en mémoire les patients avec lesquels elle communique, pour ne pas faire une demande de connexion à chaque initialisation de celle-ci. C'est pourquoi il a fallu effectuer des recherches sur les différentes méthodes de mémorisation possibles. Les technologies trouvées sont Web Storage et SQLite.

4.4.1 Technologies possibles

Web Storage

Il permet de stocker des données dans le navigateur. Il est plus puissant que les cookies qui sont limités à une capacité de mémorisation de quelques Ko contre plusieurs Mo pour le Web Storage. (référence : [17])

Il met à disposition deux interfaces :

- sessionStorage
- localStorage

sessionStorage mémorise les données tant que l'application est exécutée. Une fois celle-ci fermée, les données sont automatiquement supprimées.

localStorage enregistre sans limitation de durée et a une mémoire de 5 Mo. Par contre, les données ne sont pas cryptées, ce qui implique qu'elles sont visibles par tous.

Cette technologie fonctionne sur le concept "clé-valeur". Pour chaque valeur sauvegardée, une clé lui est dédiée. Par exemple, lorsque nous voulons sauvegarder le nom d'un patient, il faut procéder de la manière suivante :

```
window.localStorage.setItem('nom', 'Ermal');
```

Cela peut rapidement compliquer son utilisation dans le cas où nous avons beaucoup de données à mémoriser.

L'emploi de cette technologie ne nécessite aucun plugin puisqu'il est géré par Ionic.

Le tableau ci-dessous présente les privilèges et les inconvénients de la technologie localStorage.

Avantage	Désavantage
Permet de mémoriser dans le téléphone	Peut-être un problème si beaucoup de données à sauvegarder
Facile à utiliser (clé-valeur)	Que 5 Mo de mémoire
Pas de plugin à installer	

Tableau 21 Avantage et désavantage de la technologie localStorage

SQLite

« *SQLite* est une bibliothèque écrite en C proposant un moteur de base de données accessible par le langage SQL. » (référence : [18])

La caractéristique principale est qu'elle n'est pas implémentée sur un serveur externe (comme le sont les bases de données standard), mais directement dans le programme. Cela réduit considérablement le temps de sauvegarde et de récupération des données.

SQLite occupe moins de 300[Ko] et est directement intégrée dans l'application. Il utilise sa bibliothèque logicielle, avec son moteur de base de données. Les données mémorisées sont sauvegardées dans un fichier externe propre à chaque base de données. Ce fichier comprend donc les données, mais aussi les déclarations, les tables et les indexes de celle-ci. Comme il est accessible par le langage SQL, la création et la manipulation de la BDD sont très simples. Il suffit d'effectuer la bonne requête SQL pour y parvenir. C'est pour toutes ces raisons qu'elle est très appréciée par les systèmes embarqués.

L'utilisation de SQLite se fait grâce au plugin « *sqlite-storage* » [19].

Le tableau ci-dessous présente les privilèges et les inconvénients de la technologie SQLite.

Avantage	Désavantage
Permet de mémoriser dans le téléphone	Prends un certain temps pour charger les données
Facile à utiliser (Requête SQL)	
Extrême légèreté (moins de 300 Ko)	

Tableau 22 Avantage et désavantage de la technologie SQLite

4.4.2 Technologie utilisée

Pour une question de simplicité d'utilisation (requête SQL) et une possibilité de sauvegarder un nombre important de données, nous choisissons SQLite pour la mémorisation des informations des patients.

Par contre, on utilise la technologie locale Storage pour les données globales. Par exemple, le nombre d'alerte à mémoriser par patient ou le délai indiquant une alerte récente.

4.5 Notification

Une alerte peut survenir à n'importe quel moment indiquant qu'un patient a un problème. Il faut donc informer le personnel de cette urgence.

La notification donne la possibilité de visualiser qu'une alerte est parvenue. Par contre, si le téléphone est dans la poche, il est impossible pour l'utilisateur d'être immédiatement informé qu'un de ces patients est en danger.

Afin de remédier à ce problème, l'idée est qu'à chaque alerte le téléphone sonne et vibre. Pour ce faire, Cordova propose le plugin « cordova-plugin-vibration » [20] pour faire vibrer le téléphone pendant X [ms] voulu et le plugin « cordova-plugin-dialogs » [21] pour le faire sonner.

En ce qui concerne la notification, Cordova fournit le plugin « background-mode » [22] pour avertir de manière visuelle lorsqu'un patient est en danger.

La figure ci-dessous montre les 2 états d'une notification.

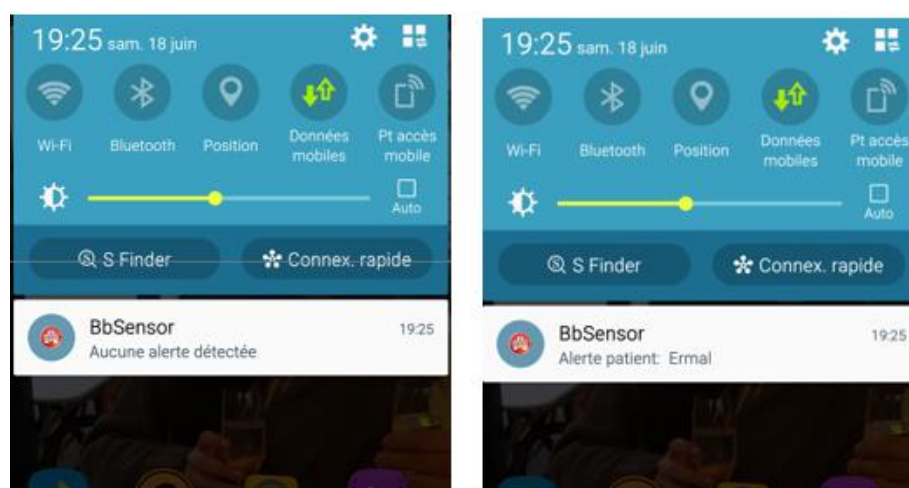


Figure 16 Notification générée par le plugin Background

Comme le montre la Figure 16, il génère une notification, une fois le mode Background activé (voir chapitre suivant) il affiche le nom du patient en danger dans la notification.

Le tableau ci-dessous présente les privilèges et les inconvénients de la technologie Notification.

Avantage	Désavantage
Informe l'utilisateur d'une alerte de manière sonore, visuel et vibre	Utilisation de trois plugins différent
Facile à mettre en place	

Tableau 23 Avantage et désavantage des technologies utilisées pour la notification

4.6 Background mode

Quand les patients sont ajoutés et mémorisés dans la base de données, ils envoient continuellement des informations (fréquence cardiaque, fréquence respiratoire, alertes, ...).

Cependant l'application ne doit pas monopoliser le téléphone et permettre au soignant d'utiliser celui-ci pour d'autre fonctionnalité. Malheureusement, à la fermeture de l'application, la connexion WebSocket se rompt et les alarmes ne sont plus communiquées à l'utilisateur.

Comme il est impératif de recevoir les alertes à n'importe quel moment, la solution pour pallier à ce problème est d'utiliser le plugin « background-mode » [22] de Cordova.

Donc, ce plugin met l'application en arrière-plans sans pour autant la fermer et peut utiliser le téléphone comme voulu. Puisque l'application n'est pas fermée, la connexion n'est pas rompue et les alertes sont toujours reçues.

Le tableau ci-dessous présente les privilèges et les inconvénients de cette technologie.

Avantage	Désavantage
L'application ne monopolise pas le téléphone.	L'application consomme plus d'énergie vue qu'elle n'est pas fermée
La connexion pour la réception des alarmes n'est pas rompue.	

Tableau 24 Avantage et désavantage des technologies utilisées pour le mode Background

4.7 Historique alerte

A la réception d'une alerte, celle-ci est mémorisée dans une liste désignant le motif de l'alerte (fréquence cardiaque trop élevée, ...) la date et l'heure d'apparition.

Pour visualiser le signal au moment de l'alerte il faut l'afficher dans un graphique. Les technologies offrant ces possibilités sont Chart et Zingchart.

4.7.1 Technologies possibles

Chart

Chart.js est une librairie JavaScript permettant la création de graphiques. Elle admet utilisation Angular.chart.js qui se base sur AngularJS pour faciliter la programmation. Par contre, dans l'application, la possibilité de sélectionner et visualiser un fragment du signal est très importante, cette fonctionnalité est difficile à mettre en place avec Char.js.

Ionic ne supporte pas non plus la dernière version de chart.js rendant la recherche d'information plus compliquée.

Le tableau ci-dessous présente les privilèges et les inconvénients de la technologie Chart.

Avantage	Désavantage
Permet de créer facilement un graphique	La dernière version n'est pas prise en charge par Ionic
	Documentation difficile à comprendre et à trouver

Tableau 25 Avantage et désavantage de la technologie Chart

Zingchart

Zingchart.js [23] est une librairie JavaScript permettant aussi la création de graphiques. Elle gère la notion de zoom en sélectionnant la partie du graphique que nous voulons voir et permet de visualiser la valeur de n'importe quel point du graphique. Sur leur site internet, il donne un exemple de graphique comprenant 100000 données. Cela convient parfaitement à notre projet vu que nous voulons afficher un signal d'au moins 100000 données pour avoir une plage de temps de cinq minutes.

Comme nous pouvons le voir sur la figure 17, Zingchart donne aussi l'opportunité d'avoir un second graphique montrant la totalité du signal. Celui-ci est muni de quatre curseurs avec lesquels on peut corriger l'amplitude et l'écart de temps du signal que nous voulons afficher sur le graphique principal.

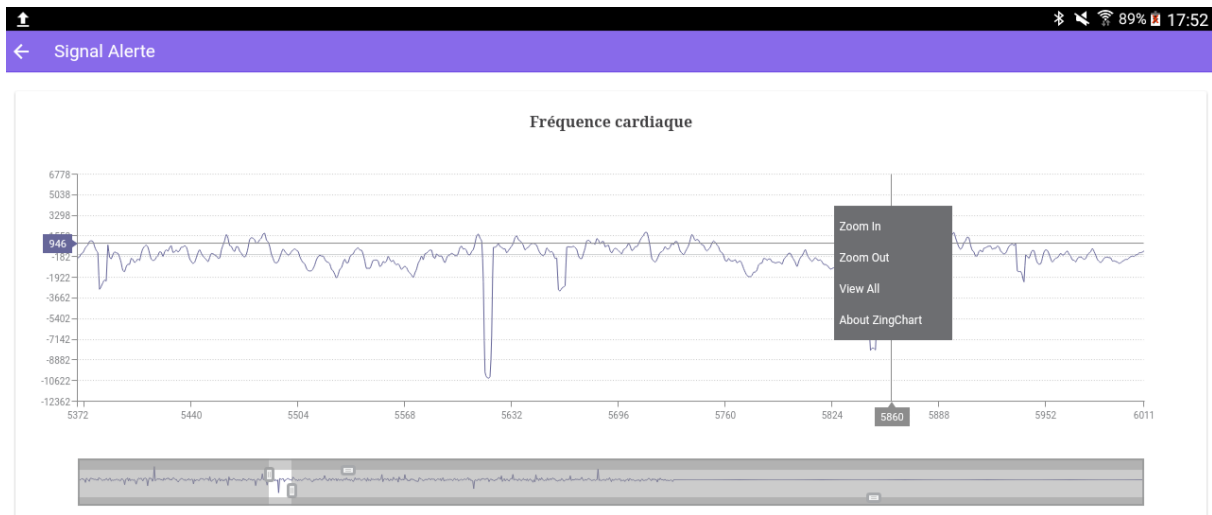


Figure 17 Représentation d'un graphique sur Zingchart

Il met aussi à disposition un menu, qui donne la possibilité d'avoir un autre moyen de gérer le zoom grâce aux différentes options proposées. De plus, il est possible d'avoir d'autres fonctionnalités dans ce menu comme le téléchargement du graphique en format PDF. Par contre, cette fonctionnalité comme beaucoup d'autres nécessite d'avoir la version payante de Zingchart.

Le tableau ci-dessous présente les privilèges et les inconvénients de la technologie Zingchart.

Avantage	Désavantage
Incorpore automatiquement une barre graphique de sélection	Requiers une licence pour certaines fonctionnalités
Permet le zoom	

Tableau 26 Avantage et désavantage de la technologie Zingchart

4.7.2 Technologie utilisée

Zingchart incorpore automatiquement la barre graphique de sélection d'une partie du signal. De plus, la documentation de Zingchart est plus claire et plus facile à trouver que Chart.js. Avec Zingchart, l'option de zoom est prise en compte et elle possède un grand nombre de méthodes pour gérer le graphique. Sans compter qu'elle permet aussi l'affichage d'un grand nombre de données. Alors, la technologie utilisée est Zingchart.

Pour utiliser ZingChart, il suffit de télécharger et incorporer le fichier « zingchart.min.js » [24] dans le projet.

5. Description fonctionnelle de l'application

Pour comprendre comment fonctionne l'application, voici une description des différentes vues et options de celle-ci :

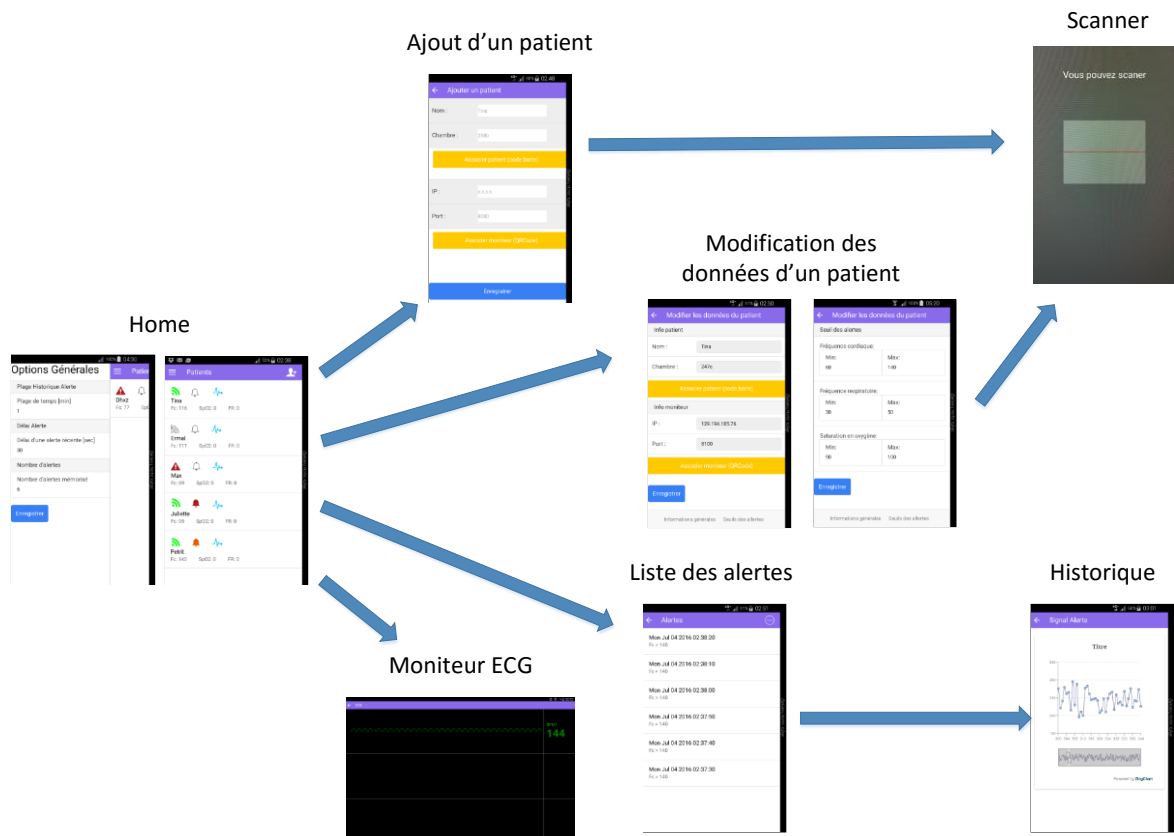


Figure 18 L'ensemble des vues de l'application

La figure 18 illustre les différentes vues de l'application. Nous pouvons remarquer que la vue de départ est Home et que les flèches indiquent les transitions possibles.




5.1 Home

Comme nous pouvons le voir sur la figure 19, la vue home contient tous les patients suivis.

Le patient comporte le nom, la valeur de sa fréquence cardiaque « FC », la valeur du taux d'oxygénation « SpO2 » et la valeur de la fréquence respiratoire « FR ».





Nous pouvons aussi distinguer qu'il y a trois icônes à gauche du patient.

- La première icône indique l'état de la connexion.
Elle peut se représenter sous trois formes :

- Connecté 
- Déconnecté 
- Erreur de connexion 

En cliquant dessus, nous gérons la connexion et la déconnexion.

- La deuxième signale lorsque le patient a un problème.
Elle peut se représenter sous trois formes :

- Pas d'alerte 
- Alerte récente   (change de couleur toutes les 500ms)
- Alerte non récente 

En cliquant dessus, nous appelons la vue « Liste des alertes » où nous pouvons visualiser les alertes parvenues.

- Puis la dernière nous permet, quand on clique dessus, d'aller à la vue « Moniteur ECG » pour avoir une vision instantanée des signaux comme sur le moniteur ECG de l'hôpital.



Il y a aussi la possibilité d'ajouter un nouveau patient grâce au bouton "Ajouter" qui appelle la vue « Ajout d'un Patient ».




Pour supprimer ou modifier un patient, il vous suffit de glisser le doigt vers la gauche sur le patient.



- Va à la vue « Modification donnée » pour changer les paramètres du patient
- Supprimer le patient.



Comme visualisé à la figure 19, 'il y a aussi un bouton "Menu" en haut à gauche de l'écran. Il permet d'ouvrir le menu des options général. 

- La première option spécifie la plage de temps du signal passé, à afficher en cas d'alerte. Ce signal sera vu dans « Historique Alerte ».
- Le deuxième permet d'indiquer le temps d'une alerte en seconde.
- La troisième permet d'indiquer combien d'alertes nous gardons en mémoire dans la vue « Liste des Alerte ».

Il est aussi possible d'ouvrir le menu en glissant le doigt de gauche à droit sur l'écran.

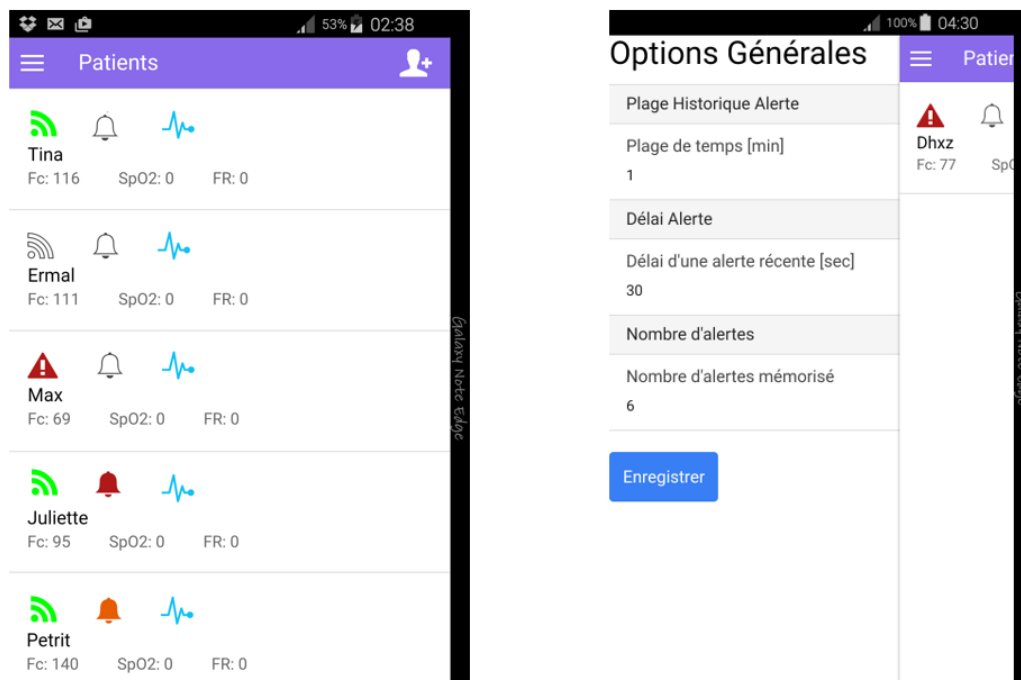


Figure 19 Vue Home et son menu

La figure ci-dessus illustre la liste des patients et des différents états de connexion et d'alertes que peut avoir un patient. Il donne aussi une vision du menu.

5.2 Ajout d'un patient

Cette vue permet d'ajouter un nouveau patient. Pour ce faire, il y a deux possibilités. Soit utiliser le scanner, soit rentrer les informations manuellement.

- Pour le faire manuellement, il suffit de donner un nom et un numéro de chambre pour les informations du patient. Puis l'ip et le port pour le Reptar.
- Pour utiliser le scanner, il suffit de cliquer sur le bouton « Associer patient » et scanner le code-barre du patient ou « Associer moniteur » et scanner le QRCode.

La figure ci-dessous affiche la vue à l'activation de scanner.

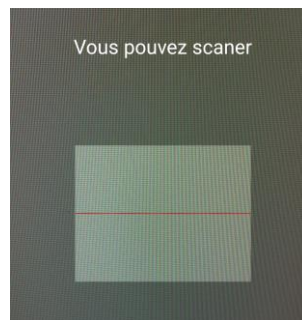


Figure 20 Scanner QRCode - Code-barre

Pour ajouter un nouveau patient, il faut spécifier un nom, un numéro de chambre, un ip et un port. L'ip et le port permettent de se connecter au patient. Une fois les informations rentrées, il suffit de cliquer sur le bouton « Ajouter ». L'application se charge de créer la connexion avec le patient et revient à la page « Home » automatiquement.

La figure ci-dessous représente la vue de l'application pour ajouter un patient.

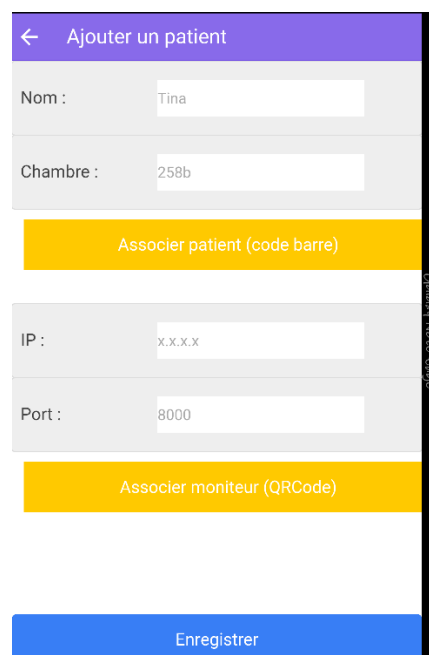


Figure 21 Vue Add. Patient

5.3 Modification des données d'un patient

Cette vue gère la modification des données d'un patient. Pour ce faire, il y a les mêmes possibilités que sur la vue « Add. Patient ».

Par contre, comme on peut le voir sur la figure 22, il y a deux boutons en bas de la vue qui affichent les informations générales ou les seuils des alertes du patient.

- **Informations générales** est la vue initiale
- **Seuils des alertes** permet de modifier les seuils des alertes du patient.

The figure displays two mobile application screens for modifying patient data. Both screens have a purple header with a back arrow and the title 'Modifier les données du patient'. The status bar at the top shows 100% battery and 05:25.

Left Screen (Informations générales):

- Info patient:** Nom: Tina, Chambre: 247c.
- Associer patient (code barre):** A yellow button.
- Info moniteur:** IP: 129.194.185.76, Port: 8100.
- Associer moniteur (QRCode):** A yellow button.
- Enregistrer:** A blue button.
- Bottom bar:** Informations générales (selected), Seuils des alertes.

Right Screen (Seuils des alertes):

- Seuil des alertes:** The active tab.
- Fréquence cardiaque:** Min: 60, Max: 140.
- Fréquence respiratoire:** Min: 30, Max: 50.
- Saturation en oxygène:** Min: 90, Max: 100.
- Enregistrer:** A blue button.
- Bottom bar:** Informations générales, Seuils des alertes (selected).

Figure 22 Vue Modif. Patient - Informations générales et Seuils des alertes

La figure ci-dessus illustre les deux vues possibles pour la modification des données d'un patient.

5.4 Liste des alertes

Comme nous pouvons le voir sur la figure 23, cette vue affiche les dernières alertes du patient. Ces alertes sont constituées de la date et l'heure à laquelle elles ont eu lieu et le motif de celles-ci.

Le nombre d'alertes affiché est paramétré par l'option générale de la vue « Home ». Si une nouvelle alerte arrive et que le nombre d'alertes mémorisées est au maximum, la plus ancienne alerte se supprime pour laisser place à la nouvelle. Toutes les nouvelles alertes sont ajoutées en haut de la liste.

Pour supprimer une ou toutes les alertes, il suffit de cliquer sur le bouton "supprimer" en haut à droite de l'écran. Cela affiche un bouton "Tout supprimer" en haut de la liste et des boutons de suppression sur chaque alerte.

- En cliquant sur le bouton "Tout supprimer", toutes les alertes se suppriment.
- En cliquant sur le bouton de suppression de l'alerte, seule cette alerte se supprime.

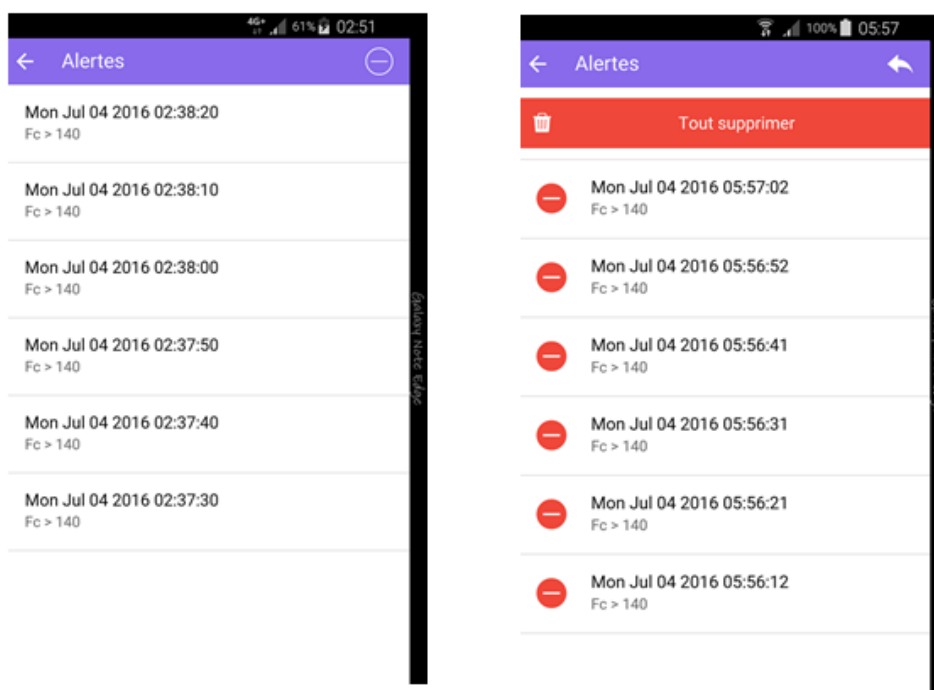


Figure 23 Vue Liste des Alertes

5.5 Historique

Cette vue affiche le signal d'une alerte passée. Elle est constituée de deux graphiques.

- Le premier, affiche en grand, une partie du second graphique. Comme nous pouvons le voir sur la figure 25, il est utile pour une meilleure vision du signal. Cela facilite les mesures et l'analyse. Il donne aussi la possibilité de zoomer. Pour ce faire, il suffit de sélectionner avec le doigt la partie du signal à afficher. L'autre méthode est de maintenir le doigt sur le graphique. Un menu s'affiche donnant la possibilité de zoomer/dézoomer par palier.
- Le second affiche l'intégralité du signal de l'alerte passée. Il est constitué de quatre curseurs qui permettent de sélectionner la plage de temps et l'amplitude du signal à afficher dans le premier graphique. La figure ci-dessous illustre la manipulation de ces curseurs.



Figure 24 Visualisation du second graphique

Il y a aussi deux boutons, en bas à gauche permettant de changer l'amplitude du second graphique sans utiliser les curseurs.

Le dernier bouton tout à droite donne la possibilité de revenir au moment de l'alerte et afficher le signal avec une plage de temps de 6 secondes (trois secondes avant et après l'alerte) sur le premier graphique.

La figure ci-dessous représente la vue historique.

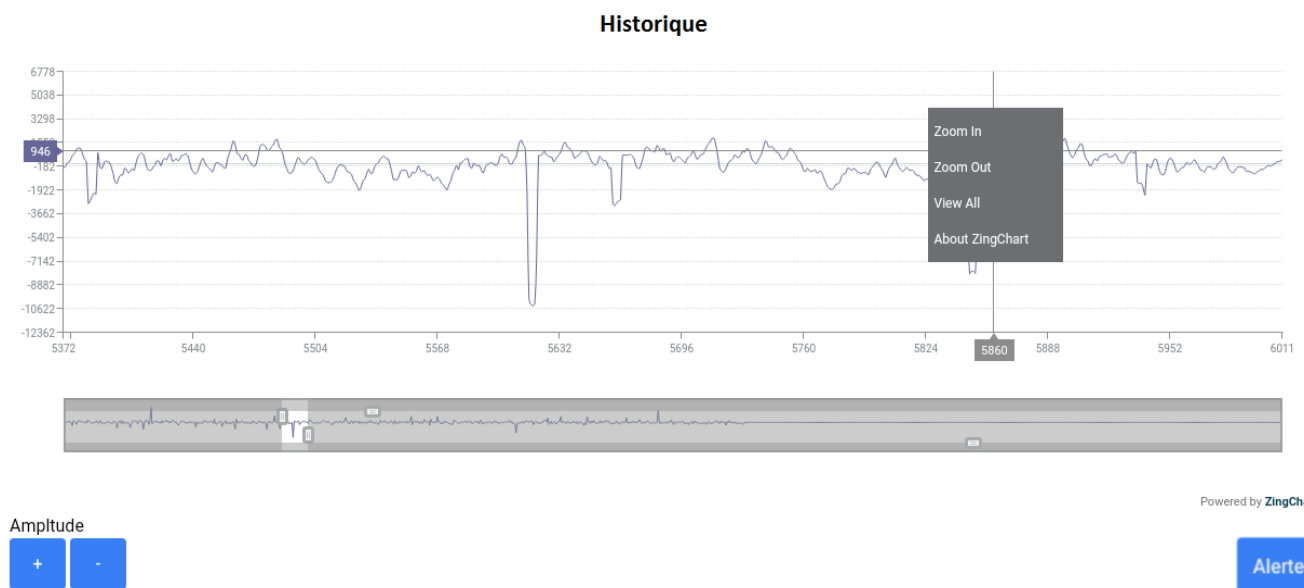


Figure 25 Vue Historique

5.6 Moniteur ECG

Comme affiché dans la figure 26, cette vue permet de visualiser les signaux en temps réel comme sur le moniteur de l'hôpital. Elle affiche les signaux et les fréquences des capteurs ECG et SpO2. Actuellement, on n'affiche que le signal et la valeur de la fréquence cardiaque, car la partie matérielle (Reptar) n'est pas encore finie.

La vue ci-dessous représente ce qui est affiché à l'ouverture de la vue Moniteur ECG. Elle comporte le signal et la fréquence du capteur cardiaque. Ces données sont les mêmes que celles affichées sur le moniteur ECG de l'hôpital.

La figure ci-dessous représente la vue Moniteur ECG.

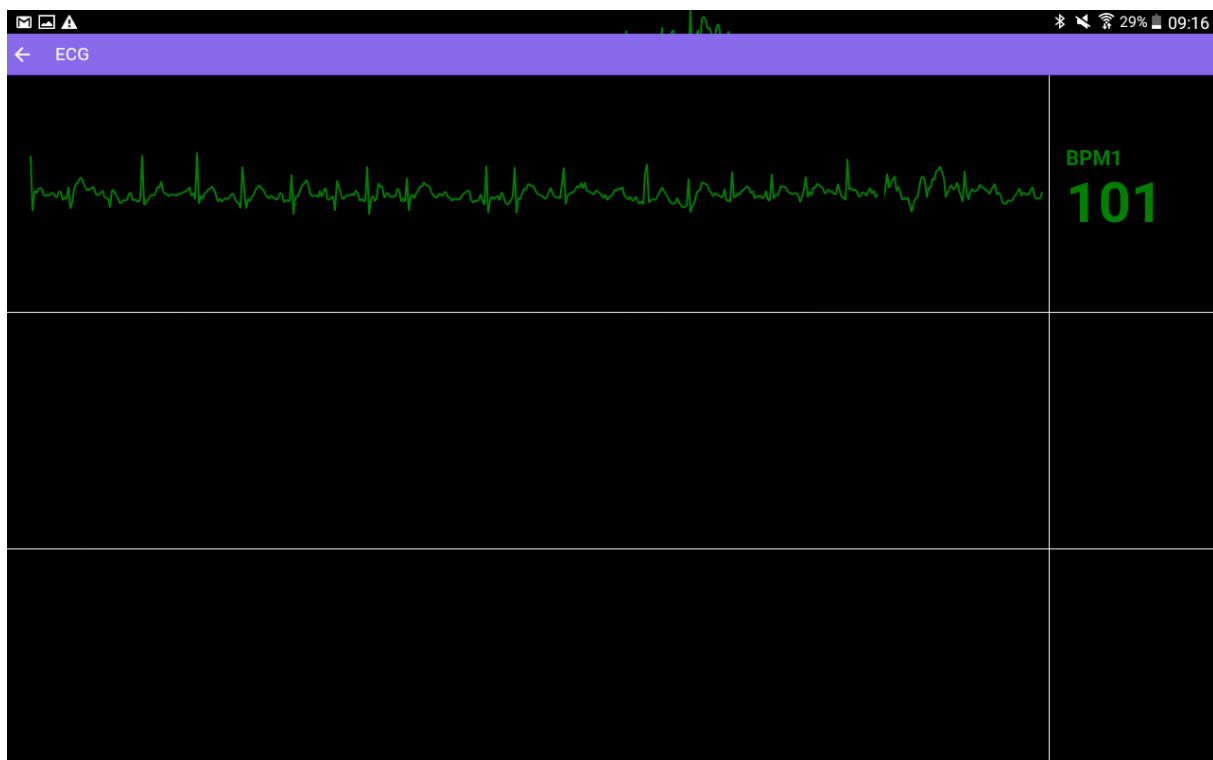


Figure 26 Vue Moniteur ECG

6. Mise en œuvre

Ce chapitre explique la structure du code et l'utilisation des différentes technologies.

6.1 Architecture logicielle

La figure ci-dessous illustre l'architecture de l'application.

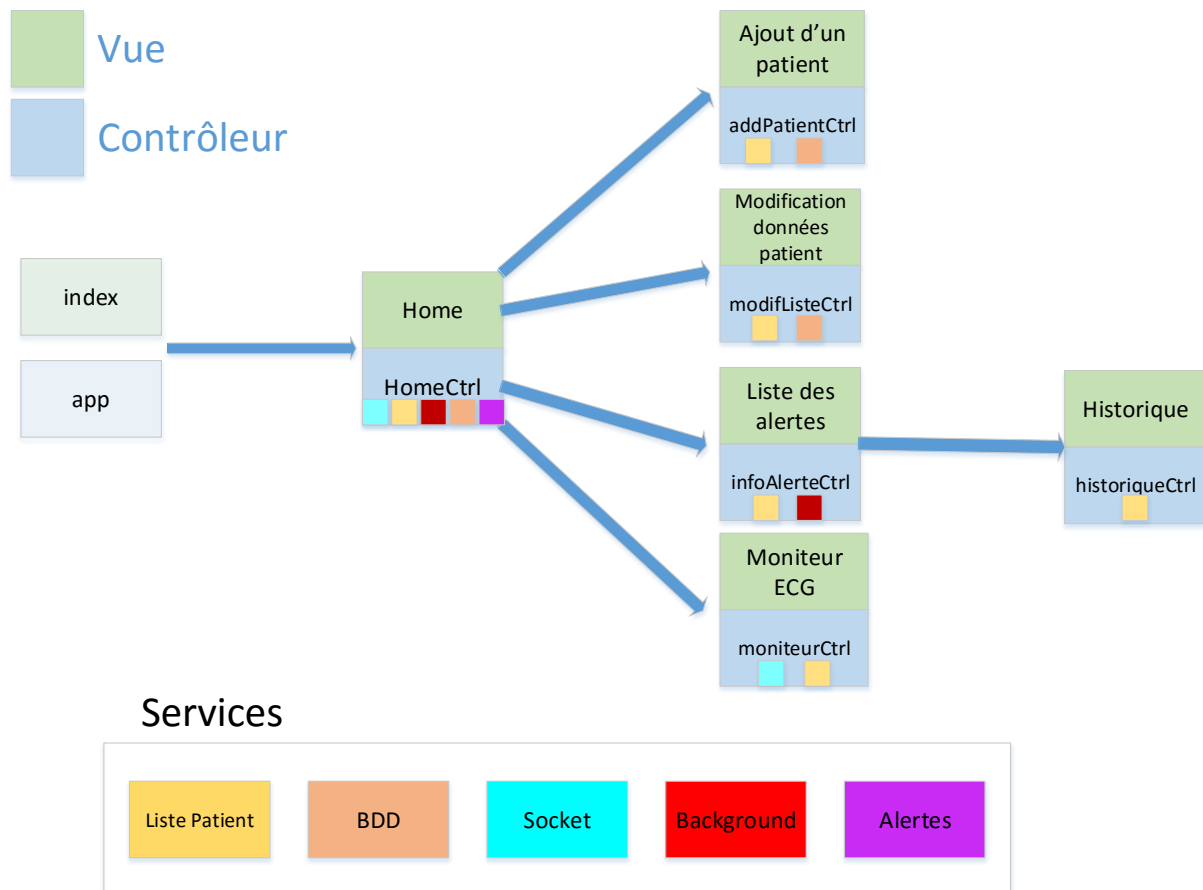


Figure 27 Architecture de l'application

Comme nous pouvons le voir sur la Figure 27, l'application est constituée de différentes vues, contrôleurs et services. Le déroulement des vues se fait de gauche à droite, chaque vue est liée à un contrôleur (Home – HomeController) et chaque contrôleur est injecté par un ou plusieurs services.

6.2 Vues et contrôleurs

Dans ce chapitre, nous expliquons les points importants du code concernant les différentes vues et/ou contrôleurs du programme.

6.2.1 Index & app

infex.html et app.js sont les codes initiaux de l'application. Voici une explication des points importants les concernant.

index.html est la vue principale. Elle est utilisée comme modèle à toutes les autres vues. Il permet de signaler au programme où se trouvent les sources des contrôleurs, des services et des librairies utilisées (js/zingchart.min.js) et met en place la gestion des routes.

app.js est le premier code exécuté par l'application. Il active Cordova et initialise les plugins. Il vérifie aussi s'il y a une base de données existante contenant les données des patients. Si c'est le cas, il récupère les données et les mémorise dans le tableau « patients » qui se trouve dans le service « listePatient ». Sinon, il crée la base de données. Les différentes méthodes pour gérer la base de données se trouvent dans le service « BDD ». Les deux services que nous venons de citer sont expliqués dans le chapitre « Services » à la suite du rapport.

Pour ajouter un module Angular à l'application (par exemple : chart.min.js) il faut la spécifier dans « angular.module ». Cela permet à Angular de l'initialiser pour pouvoir l'utiliser.

6.2.2 HomeCtrl

Les points importants dans ce contrôleur sont la gestion : des connexions, du bouton retour en arrière du téléphone et du mode mise en arrière-plan de l'application.

La gestion des connexions

Elle gère la connexion des patients, lors de l'activation de l'application, de l'ajout ou la modification d'un patient. Puis, le contrôleur va faire appel au service « Socket » pour effectuer la connexion. Ce service effectue la connexion WebSocket, l'envoi et la récupération des différentes données. Il est expliqué dans le chapitre « Services » à la suite du rapport.

La gestion du bouton retour en arrière du téléphone

Elle est utilisée pour ne pas quitter immédiatement l'application quand on active le bouton « retour en arrière » du téléphone. On met en place un message de confirmation qui permet de garantir que l'action voulue est de quitter l'application. Pour gérer ce bouton, il faut utiliser « `$ionicPlatform.registerBackButtonAction` ».

Cette dépendance permet de configurer l'action du bouton retour en arrière du téléphone. Elle prend en paramètre une fonction qui est appelée à chaque action du bouton et une priorité. La priorité est une valeur numérique indiquant pour quelle action nous effectuons une configuration.

Les différentes actions sont les suivantes :

- 100 = retour à la vue précédente
- 150 = rejeter le modèle
- 200 = fermer l'action feuille
- 300 = rejeter le Popup
- 400 = rejeter le chargement overlay

Cependant, pour que cette action ne s'effectue pas sur toutes les vues, il faut lui spécifier qu'il faut exécuter cette action que pour la vue home. Il est possible de faire ce test grâce à « `$ionicHistory.currentTitle` » qui permet de connaître la vue active.

La gestion du mode background met en arrière-plans l'application sans la quitter. Pour ce faire, nous utilisons le plugin « background-mode » [22]. Une fois le plugin installé, la gestion de celle-ci s'effectue dans le service « Background ». Il contient toutes les fonctions de manipulations. Ce service est expliqué dans le chapitre « Services » à la suite du rapport.

6.2.3 addpatientCtrl

Le point important dans ce contrôleur est la gestion du plugin « barcodescanner ». Il gère le scanner des codes-barres et des QRcodes.

La gestion du scanner Code-barre et QRcode

Il rend l'ajout des informations des patients plus Plug&Play. Pour ce faire, on utilise le plugin « barcodescanner » [16].

Pour gérer le scanner, on utilise deux méthodes.

- QRcode
- Code-barre

QRcode et Code-barre permettent de scanner un QRcode ou un code-barre. Pour cela, il faut lancer le plugin qui une fois le scanne effectué renvoi le message qui y est associé. Le message stocké dans le QRcode est de la forme suivante :

"ip129.194.185.76-port8100"

Une fois le QRcode scanné, nous testons si le message contient bien un ip et un port. Lorsque c'est le cas, l'application mémorise les données et les associe au patient.

La figure suivante donne un exemple d'utilisation du plugin CordovaBarcodeScanner.

```
// lence le Scanner
$cordovaBarcodeScanner.scan().then(function(imageData)
{
    // Si le message contient un ip et un port
    if((imageData.text.search("ip") != -1) && (imageData.text.search("port") != -1)){
        var text = imageData.text.replace("ip", "");
        text = text.replace("port", "");
        var val = text.split("-");

        // Recupère la valeur de l'ip
        $scope.newPatient.ip = val[0];
        // Recupère la valeur du port
        $scope.newPatient.port = parseInt(val[1]);
    }
}, function(error) {
    console.log("An error happened -> " + error);
});
```

Figure 28 Récupération des données du code-barre

Pour ce qui est du code-barre, il est impossible de se procurer des codes-barres de l'hôpital HUG, car ils contiennent les informations réelles des patients, qui sont tenues au secret professionnel. C'est pourquoi nous avons utilisé un exemple d'internet comprenant le numéro de série suivant : 5901234123457.

Si nous scannons un code-barre et que le numéro de série est celui pris d'internet, nous forçons un nom (baby) et un numéro de chambre (185b) au patient.

6.2.4 modifListeCtrl

Ce contrôleur gère le scan des QRCode et des codes-barres, mais permet aussi de modifier les seuils des alertes. Une fois les seuils changés, ils sont renvoyés au Reptar pour le mettre à jour. L'envoi de ces données se fait dans le service « Socket ». Ce service est expliqué dans le chapitre « Services » à la suite du rapport.

6.2.5 infoAlerteCtrl

Les points importants de ce contrôleur sont : l'affichage, la suppression des alertes reçues et la gestion du clignotement de l'icône indiquant une alerte.

L'affichage et la suppression des alertes

A la réception d'une alerte, celle-ci est affichée dans une liste. Cette liste contient les dates et heures d'apparition et les types d'alertes.

La date et l'heure sont récupérées grâce à la classe « Date ». Cette classe met à disposition la méthode « .toLocaleTimeString » qui retourne la date et l'heure actuelle.

Pour supprimer les alertes indésirables, nous utilisons le composant « show-delete » d'Ionic. Ce composant met en place automatiquement et de manière élégante des boutons de suppression à chaque alerte. Ces boutons sont affichés uniquement quand on clique sur le bouton de suppression se trouvent en haut à droite de la vue.

Gestion du clignotement de l'icône

Lorsqu'une alerte est reçue, l'icône alerte se met à clignoter. Pour mettre en place ce clignotement, nous avons utilisé la dépendance « \$interval ». Il donne la possibilité d'appeler une fonction à l'intervalle de temps voulu. Comme nous pouvons le voir ci-dessous, elle prend en paramètre la fonction à appeler et l'intervalle de temps en milliseconde.

```
$interval(clignoteFunc, 500);
```

Une fois toutes les alertes supprimées, il faut stopper ce clignotement pour que l'icône repasse à l'état initial (pas d'alerte). Pour ce faire, AngularJS propose la méthode « .cancel ». Par contre, il est possible que l'intervalle ne s'arrête pas immédiatement et effectue encore un cycle. Ainsi, pour assurer l'arrêt immédiat, AngularJS propose d'utiliser « \$destroy » pour garantir la destruction de celle-ci.

6.2.6 historiqueCtrl

Pour afficher le signal d'une alerte passée et pouvoir l'analyser, nous utilisons la librairie « zingchart ». Elle donne la possibilité de créer un graphique. Les points importants de la librairie sont les suivants.

Création du graphique

Pour créer et afficher le graphique, nous appelons la méthode « .render » de zingchart. Cette méthode prend en paramètre un objet construit de la manière suivante.

```
zingchart.render({  
  id: 'myChart',  
  data: myConfig,  
  height: 400,  
  width: "100%",  
});
```

- L'id du tableau spécifié dans la vue « historique.html »
- Les données et la configuration du graphique
- La hauteur
- La largeur

myConfig est constitué de « gui » et « graphset ».

- « gui » gère le menu du graphique qui s'affiche quand on maintient le doigt sur le graphique. Un grand nombre d'options est disponible, mais certaines demandent d'avoir la version payante de zingchart. Elles sont alors dans notre cas inutilisable. Pour les retirer, il faut configurer « behaviors ». L'exemple ci-dessous montre comment enlever l'option de téléchargement du graphique en format PDF.

```
"behaviors": [  
  {  
    // retire l'option DownloadSVG  
    "id": "DownloadSVG",  
    "enabled": "none"  
  },  
]
```

Figure 29 Suppression d'une option du menu de Zingchart

- « graphset » permet d'indiquer quel type de graphique nous voulons, le titre, le sous-titre, les données et les différentes options (le zoom).

Option zoom

Zingchart propose l'option zoom qui offre la possibilité de zoomer sur le graphique. Nous avons trois moyens pour le faire.

- En sélectionnant sur le graphique quelle partie du signal nous voulons visualiser.
- En utilisant des curseurs pour sélectionner quelle partie du signal nous voulons visualiser.
- Grâce à deux boutons "+" et "-" qui change l'amplitude de l'affichage du signal.

Pour utiliser cette option, il faut le spécifier dans « scale-x » et « scale-y ». La figure ci-dessous montre comment activer l'option zoom sur l'axe X et d'initialiser la position des curseurs.

```
"scale-x": {
  "values": "0:100000:1", // valeur à afficher sur l'axe X
  "max-items": 11, // nombre de valeur sur l'axe X
  "zooming": true, // Active le zoom
  "zoom-to": [500, 600], // position des curseurs
  "item": {
    "font-size": 10 // taille des valeurs affichées sur l'axe X
  }
}
```

Figure 30 Gestion du zoom avec zingchart

Le changement d'amplitude est possible grâce à la méthode « zingchart.exec() » qui modifie la valeur du zoom. Elle prend en paramètre le nom du graphique et l'action à effectuer (zoomin ou zoomout). La figure ci-dessous donne un exemple d'utilisation de cette méthode.

```
zingchart.exec('myChart', 'zoomin', {
  graphid: 0,
  zoomx: false,
  zoomy: true
})
```

Figure 31 Exemple d'utilisation de la méthode zingchart.exec() avec zoomin

Par contre, cette méthode change la valeur du zoom par pallié. Pour pouvoir spécifier la valeur du zoom à effectuer, modifier le paramètre " zoomin " par " zoomto ". La figure ci-dessous donne un exemple d'utilisation de cette méthode.

```
zingchart.exec('myChart', 'zoomto', {
  graphid : 0,
  xmin : 10,
  xmax : 40,
  ymin : 1500,
  ymax : 3000
});
```

Figure 32 Exemple d'utilisation de la méthode zingchart.exec() avec zoomto

6.2.7 moniteurCtrl

Ce contrôleur gère l’affichage des signaux en temps réel. Pour une question de meilleure fluidité d’affichage du signal, nous n’utilisons pas la librairie « zingchart » comme dans le chapitre « Historique ». Nous créons le graphique dans un canvas en JavaScript.

Création du graphique

La fonction **drawWave** crée le contour du graphique pour que celui-ci ressemble au maximum au moniteur ECG de l’hôpital.

La classe **Signal** gère l’affiche des signaux et des données fréquentielles en temps réel.

Il prend en paramètre :

- La position du moniteur
- La position du signal
- Le type de la valeur à afficher (fréquence ou signal)
- Le nom du signal ou de la fréquence à afficher
- La couleur du signal
- Une fonction pour l’affichage des limites dans un slot

Cette classe possède les fonctions « drawFlowSignal » et « basicSignal ». La première fonction crée le signal à afficher de la couleur et l’épaisseur voulue, à partir des données reçues du Reptar. La deuxième affiche la valeur fréquentielle du capteur à la suite du signal.

La connexion au Reptar est gérée par le service « Socket ». Il est expliqué dans le chapitre suivant.

6.3 Services

Voici une explication des différents services mis en place dans le programme.

6.3.1 Liste Patient

Ce service contient un tableau avec toutes les données du patient. Cela permet de les centraliser pour les récupérer plus facilement.

- **Id** : id du patient dans la BDD
- **Nom** : le nom du patient
- **Chambre** : le numéro de chambre du patient
- **Fc, SpO2 et FR** : sont les fréquences mesurées par les capteurs (fréquence cardiaque, taux d'oxygène dans le sang et la fréquence respiratoire).
- **Ip et port** : sont les données pour se connecter au Reptard
- **etatConnection** : permet d'indiquer l'état du patient (0 = non connecté, 1 = connecté, 2 = erreur de connexion)
- **alarme** : permet d'indiquer si le patient est en alerte et quel type d'alerte (0 = pas d'alerte, 1 = alerte récente, 2 = alerte non récente).
- **Alertes** : est un tableau contenant le type et la date des alertes d'un patient
- **Socket** : est le socket permettant la communication avec le Reptard du patient
- **timeoutAlarm** : les promesses de \$timeout et \$interval pour le clignotement et le changement d'icône en cas de réception d'alerte d'un patient.
- **Seuils** : permettent de configurer à partir de quelle valeur de fréquence celle-ci est considéré comme une alerte. Elle contient FC minimum, FC maximum, SpO2 min, SpO2 max, FR min et FR max.

Elle contient aussi la méthode **addNewPatient()**. Cette méthode permet d'ajouter un nouveau patient au tableau.

Puis la méthode **getAllPatientBDD()**. Cette méthode récupère tous les patients mémorisés dans la BDD et les ajoute dans le tableau à chaque lancement de l'application.

6.3.2 BDD

Le service BDD permet de centraliser toute la manipulation liée à la base de données SQLite. Elle comprend les fonctions suivantes :

- **CreatBDD** : Permet de créer la BDD. Elle prend en paramètre le nom de la BDD et « locat » qui est un paramètre de location (obligatoire). Elle retourne un boolean afin d'indiquer un succès ou un échec.
- **CreatTable** : Permet de créer une table à la BDD. Elle prend en paramètre une requête SQL. Dans notre cas, nous utilisons la requête :

```
CREATE TABLE IF NOT EXISTS BbSensor (id integer primary key, Nom text, Chambre text, IP text, Port integer)
```

- **SelectAllInBDD** : Permet de reprendre tous les patients mémorisés dans la BDD. Elle saisit en paramètre une requête SQL. Dans notre cas, nous utilisons la requête :

```
SELECT * FROM BbSensor
```

- **InsertPatientInBDD** : Permet d'insérer un patient à la BDD. Elle prend en paramètre une requête SQL, le nom du patient, sa chambre, son ip et son port. Dans notre cas, nous utilisons la requête :

```
INSERT INTO BbSensor (Nom, Chambre, IP, Port) VALUES (?, ?, ?, ?)
```

- **Delete** : Permet de supprimer un patient de la BDD. Elle prend en paramètre une requête SQL et l'id du patient à supprimer. Dans notre cas, nous utilisons la requête :

```
DELETE FROM BbSensor where id=?
```

6.3.3 Socket

Le Service Socket permet d'effectuer la connexion WebSocket entre le téléphone et le Reptar. Elle comprend les méthodes et classes suivantes :

- **JoelSocket** : est une classe prenant en charge la connexion WebSocket, la gestion des envois et les réceptions des données. Elle est constituée des méthodes suivantes :
 - **Register** : Cette méthode permet d'effectuer la connexion WebSocket. Elle prend en paramètre le type de donnée que nous voulons recevoir ("freq", "lead", "alarm" ou "history"), la fonction à appeler à la réception de donnée et l'index du patient concerné. Une fois la connexion établie, le WebSocket reçoit trois types de messages
 - onopen** : pour indiquer que la connexion est bien établie.
 - onclose** : pour indiquer que la connexion n'a pas pu s'effectuer ou qu'elle s'est rompue.
 - onmessage** : pour la réception des données. Le format des données est en binaire, car cela réduit le nombre de paquets envoyés et donc augmente la vitesse de transmission.
 - **MySocketDictionary** : Est un dictionnaire qui, par rapport au type de donnée que nous voulons recevoir ("freq", "lead", "alarm" ou "history"), contient le socket, un tableau contenant les fonctions à effectuer à la réception des données et l'index du patient.
 - **Send** : pour envoyer un message au Reptar. Il prend en paramètre le type de données et les données à envoyer.
 - **connectSendAndClose** : Est utilisée pour la réception du signal d'une alerte passé. Une fois l'alerte sélectionnée, une connexion WebSocket est mise en place, permettant d'effectuer la requête et la réception des données.

La requête est constituée de deux paramètres.

- Temps écoulés entre l'alarme et le temps actuel (valeur en tick)
- L'intervalle de temps du signal (valeur en tick)

Les ticks sont le nombre de cycles que doit effectuer le CPU. Pour transformer une valeur minute en tick, il faut multiplier la valeur par 60 puis par 250.

- **Construct** : permet de construire l'objet JoelSocket à l'appel de la vue « Moniteur ECG » puis de gérer la connexion et la réception des données (signaux et fréquences).

- **Connect** : permet de construire l'objet JoelSocket à chaque ajout ou modification d'un patient ainsi que de gérer la connexion et la réception des données (fréquences et alarmes).

Format des données réceptionnées

Les données sont réceptionnées en format binaire puis elles sont converties en chaîne de caractère. Cette manipulation s'effectue de la manière suivante :

```
var packet = new DataView(signal.data);
if(signalName === "lead" || signalName == "freq"){
    var num1 = packet.getInt16(0, true);
    var num2 = packet.getInt16(2, true);
    signal = num1 + ";" + num2;
```

Figure 33 Conversion des données reçues des capteurs de binaires en string

Comme la partie hardware n'est pas encore finie, les seules valeurs reçues des capteurs sont les données cardiaques (signal, fréquence). Ces données sont envoyées par paire, car il y a deux capteurs qui effectuent la même mesure (num1 et num2).

Pour ce qui est des alertes, il y a trois types de messages

- NEW_FREQ : permet d'indiquer la valeur des seilles de la fréquence cardiaque.
- MIN_FREQ : Indique que la fréquence cardiaque est plus basse que le seuil minimum.
- MAX_FREQ : Indique que la fréquence cardiaque est plus haute que le seuil maximum.

```
}else if(signalName === "alarm"){
    var type = packet.getInt16(0, true);
    var num1 = packet.getInt16(2, true);
    var num2 = packet.getInt16(4, true);
    if(type === MessageType.NEW_FREQ){
        signal = "NEW_FREQ;" + num1 + ";" + num2;
    }else if(type === MessageType.MIN_FREQ){
        signal = "MIN_FREQ;" + num1 + ";" + num2;
    }else if(type === MessageType.MAX_FREQ){
        signal = "MAX_FREQ;" + num1 + ";" + num2;
    }
}
```

Figure 34 Conversion des données alertes de binaires en string

Cette figure ci-dessus montre le format des données finales après conversion.

6.3.4 Background

Le Service Background permet d'activer le mode background et de gérer le texte à afficher dans la notification. Il comprend les fonctions `start`, `addNameAlert` et `removeName`.

Start

Il initialise le mode background qui s'activera à la mise en arrière-plans de l'application.

Pour cela, il faut indiquer le message par défaut à afficher dans la notification, puis l'initialiser. La figure ci-dessous montre comment initialiser et activer le mode background.

```
cordova.plugins.backgroundMode.setDefaults({
  title: 'BbSensor',
  ticker: 'BbSensor',
  text: 'Aucune alerte détectée'});
// Enable background mode
cordova.plugins.backgroundMode.enable();
```

Figure 35 Activation du mode background et initialisation du texte de la notification

Une fois l'application mise en arrière-plans, le mode background s'active et créer la notification avec le texte voulu.

```
cordova.plugins.backgroundMode.onactivate = function() {
```

addNameAlert

Il permet de modifier le texte de la notification en ajoutant le nom du patient qui est en alerte. Pour modifier le texte de la notification, il faut utiliser la méthode « configure » comme expliquée ci-dessous.

```
cordova.plugins.backgroundMode.configure({
  text: 'Aucune alerte détectée'
});
```

removeName

Il supprime de la notification, le nom d'un patient.

6.3.5 Alertes

« Le Service Alertes » permet d'indiquer à l'utilisateur qu'une alerte est parvenue. Il gère le clignotement, le changement de couleur de l'icône alerte et de faire sonner et vibrer le téléphone à l'arrivée d'une alerte.

Clignotement et changement de couleur

Afin d'indiquer à l'utilisateur qu'une alerte récente est parvenue, l'icône alerte clignote à une fréquence de 2[Hz]. Pour effectuer ce clignotement, on utilise la dépendance « \$interval ». Il donne la possibilité d'appeler une fonction à l'intervalle de temps voulu. Comme nous pouvons le voir ci-dessous, elle prend en paramètre la fonction à appeler et l'intervalle de temps en milliseconde.

```
$interval(clignoteFunc, 500);
```

Une fois que l'alerte n'est plus récente, l'icône alerte ne clignote plus et devient orange. Pour signaler que l'alerte n'est plus récente, on utilise la dépendance « \$timeout ». Cette dépendance permet d'appeler une fonction après un temps spécifié. Son utilisation se fait comme suite :

```
$timeout(maFonction, 500);
```

Sonner et vibrer

Pour faire vibrer et sonner le téléphone, il faut utiliser le plugin « Dialogs » et « Vibration ».

- **Dialogs** : fait sonner le nombre de fois voulu le téléphone, avec la sonnerie par défaut de celui-ci.

```
// fais sonner 1 fois  
navigator.notification.beep(1);
```

- **Vibration** : fait vibrer le téléphone pendant un temps voulu.

```
// Fait vibrer 1sec  
navigator.notification.vibrate(1000);
```

7. Mesures & Performances

7.1 Consommation de l'application

Ionic ne propose aucune fonctionnalité pour mesurer la consommation de batterie des applications. Pour effectuer cette mesure, nous utilisons la fonctionnalité « Batterie » mise à disposition par le téléphone. Cette fonctionnalité offre la possibilité de connaître la consommation de chaque application utilisée. Les valeurs sont données en mAh et en pourcentage. Vous pouvez l'avoir trouvé dans les paramètres batterie du téléphone.

Condition de mesure

Les mesures sont effectuées avec un téléphone Samsung Galaxy Note Edge avec la version 6.0.1 d'Android. Seules les applications obligatoires sont actives et la batterie du téléphone est rechargée à 100%. Comme le téléphone n'est plus tout récent, il offre une autonomie de plus ou moins 14 heures.

Mesure

Des mesures ont été effectuées pour les différentes vues de l'application. Elles comprennent le pourcentage de batterie utilisé et la consommation en mAh.

Comme nous pouvons le voir sur le tableau ci-dessous, chaque mesure représente la consommation d'une vue. Ces mesures ont été effectuées avec l'application ouverte pendant une durée de 1 heure.

Vue	Home	Ajout d'un patient	Modification données patient	Liste alertes	Historique	Moniteur ECG	Scanner
Consommation	170mA/h 6%	169 mA/h 6%	170 mA/h 6%	170 mA/h 6%	171 mA/h 6%	184 mA/h 8%	221 mA/h 11%

Tableau 27 Consommation des vues pendant 1 heure

Comme nous pouvons le constater, la vue qui consomme le plus d'énergie est la vue « Scanner ». Cela est compréhensible, car elle utilise la caméra. Par contre, l'utilisation de cette vue est utilisée uniquement pour ajouter ou modifier un patient, ce qui représente quelques secondes. On peut donc considérer sa consommation comme dérisoire.

Pour ce qui est de la vue Moniteur ECG, elle ne consomme que 8% de la batterie par heure. Ce qui offre une plage totale de 12,5 heures d'utilisation. Bien sûr, cette valeur n'est Jamme atteinte, car d'autre application tourne sur le téléphone qui elle aussi consomme de l'énergie. De plus, l'application n'est pas faite pour s'exécuter continuellement. Elle a la possibilité de

fonctionner en arrière-plan. Une fois ce mode activé, la consommation de l'affichage est nulle, réduisant aussi la consommation de l'application.

Pour confirmer ces mesures, nous avons testé l'application pendant une journée. L'utilisation du téléphone a été minime (pas de vidéo ni musique) pour ne pas utiliser la batterie. L'application a fonctionné pendant 9h. Par contre, elle était en arrière plan pendant 60% du temps.

7.2 Réception des données

Nous pouvons constater que les données envoyées peuvent avoir un temps de latence par rapport à la puissance du signal wifi. Comme notre communication est en TCP et qu'elle gère la gestion des paquets, certaines données peuvent être envoyées plusieurs fois avant que le téléphone les reçoive. Des mesures ont été effectuées pour connaître le seuil du signal wifi auquel nous pouvons nous attendre à avoir un temps de latence.

Condition de mesure

Les mesures sont effectuées avec un téléphone Samsung Galaxy Note Edge avec la version 6.0.1 d'Android.

Mesure

Les mesures consistent à mesurer le temps d'envoi de différente quantité de donnée à plusieurs seuils du signal wifi.

nb donnée	Signal Wifi			
	100%	80%	50%	20%
75000	4,5	4,8	5	8
150000	6	6,5	7,2	11
225000	8	8,9	9,4	16

Tableau 28 Mesure temps d'envoi

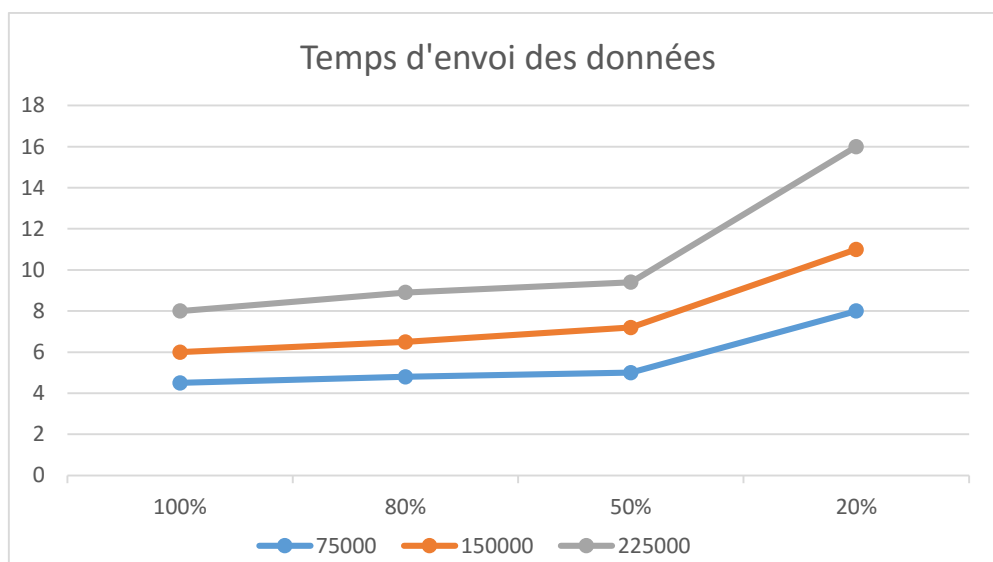


Figure 36 Graphique Temps d'envoi des données

Comme nous montre le graphique ci-dessus, le temps d'envois des données commence à être important à partir de 20 %. Pour avoir une réception de donnée convenable, il faut que l'utilisateur capte un signal wifi plus grand ou égal à 50 %.

8. Conclusion

Bilan

Ce projet de bachelor avait pour but la création d'une application mobile permettant aux médecins et aux personnels infirmiers d'avoir un suivi de leur patient. Par rapport aux différents besoins stipulés pour ce projet, nous pouvons conclure que l'application réalisée les remplit.

Elle offre la possibilité de visualiser les signes vitaux en temps réel. Elle avertit son utilisateur en cas d'urgence et permet d'afficher le signal d'une alerte passé pour pouvoir l'analyser et mieux comprendre le problème.

L'application met aussi en place un système de scanner. Cela permet de récupérer les informations des patients (nom, numéro de chambre) et du Reptar (ip, port) très facilement en peu de manipulation. Pour ce faire, il suffit de scanner un code-barre et un QRCode.

Suite à des mesures de performance, nous pouvons considérer que l'application est fonctionnelle. La réception des données étant quasi temps réel, elle permet d'avoir un affichage fluide. En ce qui concerne l'autonomie, l'application peut être utilisée pendant 9 heures sans devoir recharger le téléphone.

Améliorations possibles

- Le signal temps réel s'affiche, par contre il n'est actuellement pas possible de modifier son amplitude.
- Le code de l'application n'est pas très optimisé. Cela peut s'expliquer, car l'apprentissage du langage de programmation et des différents frameworks a été acquis durant le déroulement du projet.
- La gestion des curseurs utilisés pour le zoom d'un signal d'alerte passé peut être difficile à manipuler. Une amélioration de cette méthode est à voir.

Perspectives

Il serait intéressant d'avoir un suivi médical des patients. Nous entendons par là, de connaître les différents traitements et médicaments pris par le patient. Cela permettrait d'expliquer la cause de certaines alertes inattendues.

Actuellement, l'application peut seulement être utilisée dans l'enceinte de l'hôpital, car il faut être connecté au réseau wifi de celui-ci pour recevoir les différentes données. Rendre l'application utilisable depuis n'importe quel réseau serait judicieux. Bien sûr, un système de sécurité supplémentaire serait requis empêchant la divulgation des informations des patients.

Bibliographie

- [1] Ionic, «Framework Ionic,» [En ligne]. Available: <http://ionicframework.com/>.
- [2] AngularJS, «Framework AngularJS,» [En ligne]. Available: <https://angularjs.org/>.
- [3] Cordova, «Framework Cordova,» [En ligne]. Available: <https://cordova.apache.org/>.
- [4] F. CLEMENT, «Ionic,» [En ligne]. Available: <http://blog.ippon.fr/2015/11/02/rex-cordova-grunt-angularjs-la-stack-ionic-pour-le-developpement-dapplications-mobiles-hybrides/>.
- [5] N. PAGESY, «Introduction à Ionic Framework,» [En ligne]. Available: <http://www.supinfo.com/articles/single/179-introduction-ionic-framework>.
- [6] R. Michel, «OpenClassRooms,» [En ligne]. Available: <https://openclassrooms.com/courses/developpez-vos-applications-web-avec-angularjs/le-modele-mvc-2>.
- [7] L. Knuchel, «Créer un plugin cordova,» [En ligne]. Available: <http://loic.knuchel.org/blog/2015/03/20/creer-un-plugin-cordova/>.
- [8] Wikipédia, «AJAX,» [En ligne]. Available: [https://fr.wikipedia.org/wiki/Ajax_\(informatique\)](https://fr.wikipedia.org/wiki/Ajax_(informatique)).
- [9] Wikipédia, «User Datagram Protocol,» [En ligne]. Available: https://fr.wikipedia.org/wiki/User_Datagram_Protocol.
- [10] nutella, «Plugin github_Socket_UDP,» [En ligne]. Available: <https://github.com/nutella/ionic-cordova-chrome-udp-example>.
- [11] wikipédia, «websocket,» [En ligne]. Available: <https://fr.wikipedia.org/wiki/WebSocket>.
- [12] btford, «github_Socket.IO,» [En ligne]. Available: <https://github.com/btford/angular-socket-io/tree/master/mock>.
- [13] Wikipedia, «NFC,» [En ligne]. Available: https://fr.wikipedia.org/wiki/Communication_en_champ_proche.
- [14] don, «Plugin github_NFC,» [En ligne]. Available: <https://github.com/don/ionic-nfc-reader>.
- [15] -, «Générateur de QRCode,» [En ligne]. Available: <http://fr.qr-code-generator.com>.
- [16] wildabeast, «Plugin github_BarcodeScan,» [En ligne]. Available: <https://github.com/wildabeast/BarcodeScanner.git>.

- [17] Dev, «Stockage des données,» [En ligne]. Available:
<http://www.alsacreations.com/article/lire/1402-web-storage-localstorage-sessionstorage.html>.
- [18] Wikipédia, «SQLite,» [En ligne]. Available: <https://fr.wikipedia.org/wiki/SQLite>.
- [19] sundaravelit, «Plugin SQLite,» [En ligne]. Available:
<http://phonegappro.com/tutorials/phonegap-sqlite-tutorial-with-example-apache-cordova/>.
- [20] cordova, «Plugin Vibration,» [En ligne]. Available:
<http://cordova.apache.org/docs/en/6.x/reference/cordova-plugin-vibration/index.html>.
- [21] cordova, «Plugin Dialogs,» [En ligne]. Available:
<http://cordova.apache.org/docs/en/6.x/reference/cordova-plugin-dialogs/index.html>.
- [22] katzer, «Plugin github_Background-mode,» [En ligne]. Available:
<https://github.com/katzer/cordova-plugin-background-mode.git>.
- [23] Zingchart. [En ligne]. Available: <https://www.zingchart.com/>.
- [24] zingchart, «Package github_Zingchart,» [En ligne]. Available:
<https://github.com/zingchart/ZingChart/tree/master/client>.
- SocketIO, <https://openclassrooms.com/courses/des-applications-ultra-rapides-avec-node-js/socket-io-passez-au-temps-reel>
- AJAX, <https://openclassrooms.com/courses/simplifiez-vos-developpements-javascript-avec-jquery/premiers-pas-avec-ajax>
- WebSocket, <http://www.jmdoudoux.fr/java/dej/chap-websockets.htm>
- Introduction aux Sockets, <https://openclassrooms.com/courses/introduction-aux-sockets-1>
- Cordova, <https://cordova.apache.org/docs/fr/latest/guide/overview/index.html>
- Image Titre, <http://www.levif.be/actualite/sante/mon-smartphone-me-soigne/article-normal-322387.html>

Annexe

A.A Installation et création d'un projet IONIC

Installation Ionic

L'installation d'Ionic se fait en trois étapes.

1. Installer le gestionnaire de paquet Node.js
2. Installer Cordova

```
$ sudo npm install -g cordova
```

3. Installer Ionic

```
$ sudo npm install -g ionic
```

Bien sûr, Il faut aussi installer le JDK et le SDK d'Android. Pour plus d'information, le site d'Ionic décrit toutes ces étapes plus précisément [1].

Création d'un projet

Ionic vous permet d'avoir trois types de programmes de départ. Comme affiché sur la figure 36, le premier nommé « blanc » est vierge, ce qui implique de créer votre application de A à Z. Le deuxième « tabs » vous permet d'avoir des onglets mis en place et le troisième « sidemenu » vous permet d'avoir un menu.

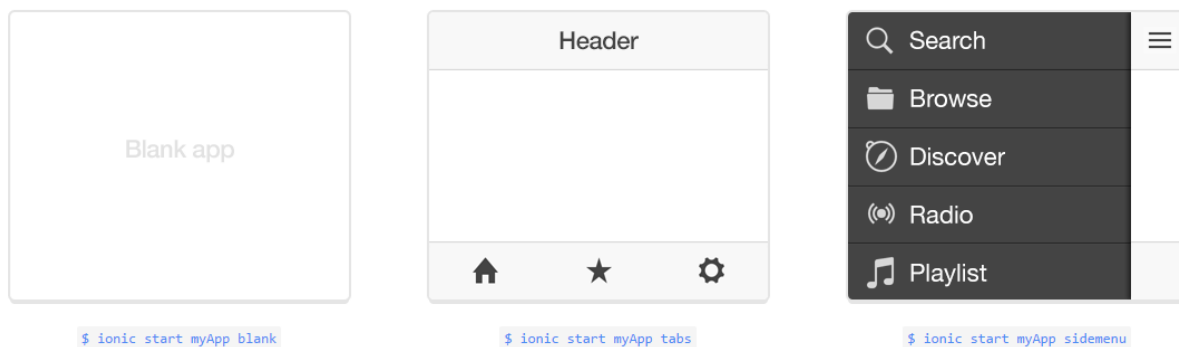


Figure 37 Représentation des trois types d'application possible de départ avec Ionic. Référence¹²

La commande à appeler pour créer un nouveau projet est :

```
ionic strart nom_de_projet type_application
```

Une fois l'application créée, vous avez la possibilité de l'exécuter sur le smartphone ou l'émuler grâce aux commandes :

```
ionic run <Platform>
```

```
ionic emulate <Platform>
```
















¹² <http://ionicframework.com/getting-started/>

La version logicielle est une version graphique d'Ionic CLI. Pour le moment, elle est uniquement disponible sur OS X, mais la version Windows devrait être disponible prochainement.






La version graphique d'Ionic Lab, permet de contrôler votre application, de la lancer sur le navigateur, l'émulateur ou sur un périphérique connecté à votre ordinateur. Elle permet également de déboguer une application, ajouter ou supprimer des plugins et lancer l'éditeur de code.

Les fichiers et dossiers générés par Ionic :

A la création d'un projet avec Ionic, un certain nombre de fichier et dossier sont générés automatiquement. Pour bien démarrer un projet, il faut avoir une idée de l'utilité de ceci. Voici une petite explication des fichiers et dossiers les plus importants.

 hooks	
 ionic-app-base-master	
 platforms	platforms : Il contient les dossier et fichier dont l'application a besoin pour fonctionner pour les plateformes demandées.
 plugins	plugins : Il contient les dossier et fichier de tous les plugins utilisé par l'application.
 resources	resources : Il contient toutes les ressources liées à l'application comme les icônes ou les splash.
 scss	
 www	www : C'est ici que se trouve tout le code de l'application. Il est lui-même organisé en sous dossier et fichier pour faciliter la compréhension et la programmation.
 .bowerrc	
 .editorconfig	
 .gitignore	
 bower.json	
 config.xml	config.xml : Il permet le contrôle des différents aspects de l'application que ce soit le nom, les icônes ou les splash utilisé.
 gulpfile.js	
 ionic.project	
 package.json	

Dossier www :

 css	Pour ce qui est du dossier www , il contient tout le code JavaScript, HTML et CSS de notre application. Cela comprend les vues, les contrôleurs et les services.
 img	
 js	
 lib	Pour mieux s'y retrouver, on conseille d'ajouter le dossier contrôleur qui comportera tous les contrôleurs, service pour les services et templates pour les vues.
 index.html	

Index.html et la vue principale de l'application. Elle permet d'avoir des options globales à toutes les autres vues comme la barView ou un menu.

Dans le dossier **js**, il y a le fichier **app.js**. Ce fichier permet d'instancier le module principal. C'est ici que la gestion des routes et l'insertion des packages AngularJS se font.

A.B Commande d'installations des différents plugins utilisés

Pour installer les plugins, il vous faut ouvrir un invité de commande dans le dossier de votre projet. Puis rentrer les commandes suivantes :

Vibreur : `cordova plugin add cordova-plugin-vibration`

Sonner : `cordova plugin add cordova-plugin-dialogs`

SQLite : `cordova plugin add cordova-sqlite-storage`

Background :

`cordova plugin add https://github.com/katzer/cordova-plugin-background-mode.git`

BarcodeScanner :

`Cordova plugin add https://github.com/wildabeast/BarcodeScanner.git`

A.C Librairie Zingchart et SocketIO

Pour pouvoir utiliser la librairie Zingchart, il faut télécharger le fichier "**zingchart.min.js**". Vous pouvez trouver ce fichier sur le site suivant :

<https://github.com/zingchart/ZingChart>

Pour la librairie SocketIO, le fichier "**socket-io.js**" se trouve sur le site suivant :

<http://socket.io/download/>

Une fois le fichier téléchargé, il faut les ajouter dans le dossier **/www/js**. Pour finir, il faut indiquer à Ionic que nous utilisons ces librairies. Pour cela il faut ajouter ces deux lignes de code dans le fichier **/www/index.html** :

```
<script src="js/socket-io.js"></script>
<script src="js/zingchart.min.js"></script>
```

A.D Code source

index.html

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="initial-scale=1, maximum-scale=1, user-
scalable=no, width=device-width">
    <title></title>

    <link href="lib/ionic/css/ionic.css" rel="stylesheet">
    <link rel="stylesheet" href="css/angular-chart.css">
    <link href="css/style.css" rel="stylesheet">

    <!-- IF using Sass (run gulp sass first), then uncomment below and remove the
CSS includes above
    <link href="css/ionic.app.css" rel="stylesheet">
    -->
    <!-- socket-io js -->
    <script src="js/socket-io.js"></script>
    <!-- ionic/angularjs js -->
    <script src="lib/ionic/js/ionic.bundle.js"></script>

    <script src="js/ng-cordova.min.js"></script>

    <!-- cordova script (this will be a 404 during development) -->
    <script src="cordova.js"></script>

    <!-- Chart.min pour la creation des graphique -->
    <script src="js/Chart.min.js"></script>
    <script src="js/angular-chart.min.js"></script>

    <script src="js/zingchart.min.js"></script>

    <script src="js/lodash.min.js"></script>

    <!-- your app's js -->
    <script src="js/app.js"></script>

    <!-- Mes
services..... -->
    <script src="service/Socket.js"></script>
    <script src="service/Background.js"></script>
    <script src="service/BDD.js"></script>
    <script src="service/ListePatients.js"></script>
    <script src="service/Alertes.js"></script>

    <!-- Mes
contrôleurs..... -->
    <script src="controllers/homeCtrl.js"></script>
    <script src="controllers/addPatientCtrl.js"></script>
    <script src="controllers/modifListCtrl.js"></script>
    <script src="controllers/infoPatientCtrl.js"></script>
    <script src="controllers/infoAlerteCtrl.js"></script>
    <script src="controllers/graphAlertCtrl.js"></script>

  </head>

  <body ng-app="starter">

    <ion-side-menus>

```



```

<ion-side-menu-content>

  <ion-nav-bar class="bar-royal mynavbar">
    <ion-nav-back-button></ion-nav-back-button>
    <ion-nav-buttons side="left">
      <button class="button button-icon button-clear ion-navicon" menu-
toggle="left"></button>
    </ion-nav-buttons>
  </ion-nav-bar>
  <ion-nav-view></ion-nav-view>
</ion-side-menu-content>
<ion-side-menu side="left">
  <h2> Options Générales </h2>
  <div class="list">
    <div class="item item-divider">
      Plage Historique Alerte
    </div>

    <label class="item item-input item-stacked-label">
      <span class="input-label">
        Plage de temps [min]
      </span>
      <input type="number" ng-
model="Paramgeneral.plagetemp">
    </label>

    <div class="item item-divider">
      Délai Alerte
    </div>

    <label class="item item-input item-stacked-label">
      <span class="input-label">
        Délai d'une alerte récente [sec]
      </span>
      <input type="number" ng-
model="Paramgeneral.delaisAlertRes">
    </label>

    <div class="item item-divider">
      Nombre d'alertes
    </div>

    <label class="item item-input item-stacked-label">
      <span class="input-label">
        Nombre d'alertes mémorisé
      </span>
      <input type="number" ng-
model="Paramgeneral.nbAlertMem">
    </label>
  </div>

  <button menu-close class="button button-positive" ng-
click="enregistrParamGeneral()">
    Enregistrer
  </button>
</ion-side-menu>
</ion-side-menus>

</body>
</html>

```

APP.js

```

// Ionic Starter App

// angular.module is a global place for creating, registering and retrieving
Angular modules
// 'starter' is the name of this angular module example (also set in a <body>
attribute in index.html)
// the 2nd parameter is an array of 'requires'
var app = angular.module('starter', ['ionic', 'ngCordova'])

app.run(function($ionicPlatform, $rootScope, BDD, ListPatients) {
  /*
  // Setup the loader
  $ionicLoading.show({
    content: 'Loading',
    animation: 'fade-in',
    showBackdrop: true,
    maxWidth: 200,
    showDelay: 0
  });
  */

  $ionicPlatform.ready(function() {
    $rootScope.mesPatients = [];
    $rootScope.bdd_ok = false;

    if(window.cordova && window.cordova.plugins.Keyboard) {
      // Hide the accessory bar by default (remove this to show the accessory bar
above the keyboard
      // for form inputs)
      cordova.plugins.Keyboard.hideKeyboardAccessoryBar(true);

      // Don't remove this line unless you know what you are doing. It stops the
viewport
      // from snapping when text inputs are focused. Ionic handles this
internally for
      // a much nicer keyboard experience.
      cordova.plugins.Keyboard.disableScroll(true);
    }
    if(window.StatusBar) {
      StatusBar.styleDefault();
    }

    // Crée la base de donnée
    //$rootScope.myDB = window.sqlitePlugin.openDatabase ({name:
"mySQLite.db", location: 'default'});
    var mBdd = BDD.CreatBDD("mySQLite.db", 'default');
    if(mBdd != false)
    {
      if(BDD.CreatTable('CREATE TABLE IF NOT EXISTS BbSensor (id integer primary
key, Nom text, Chambre text, IP text, Port integer)') != false)
      {
        BDD.SelectAllInBDD('SELECT * FROM BbSensor');
      }
    }
  });
})

// Configuration des differante etat (pages) et l'etat (page) par deffaut
app.config(function($stateProvider,$urlRouterProvider) {

  // Etat home

```

```
$stateProvider.state("home", {
    url: "/home",
    templateUrl: "templates/home.html",
    controller: "homeCtrl"
})

// Etat Mes patients
$stateProvider.state("addPatient", {
    url: "/addPatient",
    templateUrl: "templates/addPatient.html",
    controller: "addPatientCtrl"
})

// Etat Modifier Liste patients
$stateProvider.state("modifList", {
    url: "/modifList/:mIndex",
    templateUrl: "templates/modifList.html",
    controller: "modifListCtrl"
})

// Etat Info patient
$stateProvider.state("infoPatient", {
    url: "/infoPatient/:mIndex",
    templateUrl: "templates/infoPatient.html",
    controller: "infoPatientCtrl"
})

// Etat Info patient
$stateProvider.state("infoAlerte", {
    url: "/infoAlerte/:mIndex",
    templateUrl: "templates/infoAlerte.html",
    controller: "infoAlerteCtrl"
})

// Etat graphAlert
$stateProvider.state("graphAlert", {
    url: "/graphAlert",
    templateUrl: "templates/graphAlert.html",
    controller: "graphAlertCtrl"
})

// Etat par défaut (la page de demarage)
$urlRouterProvider.otherwise("/home")
});
```

Home.html

```

<!-- Indique le nom de la vue (home) -->
<ion-view view-title = "Patients">
  <!-- Indique le contenu de la vue -->
  <ion-content >
    <!-- permet d'avoir une liste de patient -->
    <!-- Que les éléments de la liste sont autorisés à être
glissée pour révéler des boutons d'option -->
    <!-- Permet
d'identifier (en donnant un nom "liste-patients") et de gérer la list
(glissement) -->
    <ion-list can-swipe = 'true' delegate-handle="liste-patients" >
      <!-- Donne un nom a mon item -->
      <!-- ng-
repeat me permet d'avoir autant d'item que j'ai de patient -->

      <!-- quand je clique sur un
item, je vais dans la page infoPatient avec le num de l'item -->
      <ion-item class="info-patient" ng-repeat="p in lesPatients"
ng-show="p.supprimer != true" >
        <!-- Me permet de changer d'icone quand
patient.notification devient true. Si notification = false, bas la classe est
'icon ion-person-stalker' sinon 'icon ion-ios-star' -->
        <!-- je mets tout ça dans une balise "i = italique"
pour me simplifier la vie -->
        <!-- Cela me permet de
changer d'icone quand notification est a true ou a false-->
        <!-- Cela me
permet de clignoter l'icone quand clignote est a true-->
        <div class= "bouton-du-patient" >
          <i class="icon" ng-class="{ 'ion-social-rss-
outline': p.etatConnection == 0, 'ion-social-rss': p.etatConnection == 1, 'ion-
alert-circled': p.etatConnection == 2}" ng-click="connectionOnOff($index)"></i>
          <i class="icon " ng-class="{ 'ion-ios-bell-
outline': p.alarme == 0, 'ion-ios-bell': p.alarme == 1, 'ion-ios-bell
bell_orange': p.alarme == 2}" ng-click="infoAlerte($index)"></i>
          <i class="icon ion-ios-pulse-strong calm" ng-
click="ecg($index)" ></i>
        </div>
        <div class= "info-du-patient" >
          <h2>
            {{p.nom}}
          </h2>
          <p>
            <span>Fc: {{p.Fc}}</span>
            <span>SpO2: {{p.SpOz}}</span>
            <span>FR: {{p.FR}}</span>
          </p>
        </div>

        <!-- ajoute un option bouton a ma liste
et lui donne une icone. Quand je clique sur le bouton,
effectuer la fonction modifPatient du controleur homectrl et donne l'index de la
liste en entrée -->
        <ion-option-button
          class = "button-positive icon ion-gear-a" ng-
click="modifPatient($index)">
        </ion-option-button>
        <!-- ajoute un option bouton a ma liste
et lui donne une icone. Quand je clique sur le bouton,
effectuer la fonction deletePatient du controleur homectrl et donne l'index de la
liste en entrée -->
        <ion-option-button
          class = "button-assertive icon ion-trash-a" ng-
click="deletePatient($index)">

```

```
                </ion-option-button>
            </ion-item>
        </ion-list>
    </ion-content>

    <!-- Ajoute un nouveau bouton a droite pour l'ajout de patient -->
    <ion-nav-buttons side="right" >
        <!-- indique que c'est un bouton menu et qui ammenne a
"addPatient" -->
        <button class="button button-icon icon ion-person-add" ui-
sref="addPatient"></button>
    </ion-nav-buttons>

</ion-view>
```

homeCtrl.js

```

// controleur du Home
app.controller("homeCtrl", function( $scope, $rootScope, $ionicHistory,
$ionicPopup, $state, $ionicListDelegate, $interval, $timeout, Socket,
ListPatients, Background, $ionicLoading, $window, BDD, Alertes, $ionicPlatform) {
    $scope.lesPatients = [];

    // Paramerte General
    $rootScope.Paramgeneral = {};
    angular.copy(Alertes.configGlobalAlert, $rootScope.Paramgeneral);

    // Variable tempon du nb patient dans le tableau pour savoir si il faut
    effectuer une connection
    $scope.nbPatientTemp = 0;
    // variable indiquand quand la bdd est prête
    $scope.bdd_ok = false;

    $rootScope.nbalert = 0;

    /**
     * Appelé quand la BDD est prête
     * @param l'evenement.
     * @param l'argument.
     * @return --.
     */
    $scope.$on('BddOK', function(event, args){
        // recupère les patients de la base de donnée
        $scope.lesPatients = ListPatients.patients;
        // Recharge la vue
        $state.go($state.current, {}, {reload: true});
    });

    /**
     * Appelé à chaque fois qu'on va venir ou revenir sur la vue Home
     * @param --.
     * @param la fonction a executer.
     * @return --.
     */
    $scope.$on('$ionicView.enter', function()
    {
        // Si la base de donnée est prête
        if($scope.bdd_ok)
        {
            $scope.newConnection();
        }
        // Effectue le code que à la deuxième fois
        $scope.bdd_ok = true;
    });

    // ----- Fonctiopn -----

    /**
     * Fonction appelé pour créer une nouvelle connection des patient pas encor
    connecté
     * @param --.
     * @return --.
     */
    $scope.newConnection = function(){

```

```

// Indique qu'il y a plus d'alerte dans le mode Background donc
changer le text
$scope.$broadcast('bbAlerteFin');

// effectuer la conection des patient pas encor connecté
if($scope.lesPatients.length > $scope.nbPatientTemp){
    for (var i = $scope.nbPatientTemp; i <
$scope.lesPatients.length; i++){
        $scope.connexionSocket( $scope.lesPatients[i].ip,
        $scope.lesPatients[i].port,      i      );
    }
    // Mais a jour la variable tempon
    $scope.nbPatientTemp = $scope.lesPatients.length;
}

/**
 * Fonction qui effetue la conection WebSocket
 * @param ip du patient.
 * @param port du patient.
 * @param index du patient dans le tableau ListPatients.patients
 * @return --.
 */
$scope.connexionSocket = function(ip, port, id)
{
    Socket.connect("ws://" + ip + ":" + port, id)
    {
        ListPatients.patients[id].socket.register("freq", $scope.freq,
id);

        ListPatients.patients[id].socket.register("alarm",$scope.alarm, id);
    }
}

//----- Fonction appelé par l'objet JoelSocket -----
//-----
/**
 * Fonction appelé a la reception de données du tunnel fréquences du
webSocket
 * @param les datas.
 * @param index du pacient pour qui sont les frequances.
 * @return --.
 */
$scope.freq = function(data, id) {
    if(data == 'connect')
    {
        console.log('connection');
        $scope.lesPatients[id].etatConnection = 1;
    }

    if(data == 'disconnect')
    {
        console.log('deconnection');
        if(ListPatients.patients[id].conectionOn){
            $scope.lesPatients[id].etatConnection = 2;
        }
        else{
            $scope.lesPatients[id].etatConnection = 0;
            ListPatients.patients[id].conectionOn = true;
        }
    }
    // si ce n'est pas le message de conection ni le message de
déconnection
    if((data != 'connect') && (data != 'disconnect'))
    {
        var seuil;
        var res = data.split(";");
        $scope.lesPatients[id].Fc = res[0];
    }
}

```

```

    }

    // Mais a jour la vue
    if(!$scope.$$phase)
    {
        $scope.$apply();
    }

}

/**
 * Fonction appelé a la reception de données du tunnel alarmes du webSocket
 * @param les datas.
 * @param index du patient pour qui sont les frequances.
 * @return --.
 */
$scope.alarm = function(data, id){
    if(data == 'connect')
    {
        console.log('alarm connection');
    }

    if(data == 'disconnect')
    {
        console.log(' alarm deconnection');
    }

    // si ce n'est pas le message de conection ni le message de
    déconnection
    if((data != 'connect') && (data != 'disconnect'))
    {

        if(data.search("NEW_FREQ;") != -1)
        {
            console.log('New Freq reçu');
            var val = data.replace("NEW_FREQ;", "");
            val = val.split(";");
            ListPatients.patients[id].seuils.FCmin =
            parseInt(val[0]);
            ListPatients.patients[id].seuils.FCmax =
            parseInt(val[1]);
        }

        if(data.search("MIN_FREQ;") != -1)
        {
            console.log('MIN_FREQ');
            seuil = "< " + ListPatients.patients[id].seuils.FCmin;
            Alertes.indiqueAlerte(id, seuil);
        }

        if(data.search("MAX_FREQ;") != -1)
        {
            console.log('MAX_FREQ');
            var seuil = "> " +
            ListPatients.patients[id].seuils.FCmax;
            Alertes.indiqueAlerte(id, seuil);
        }
    }
}

//----- Gestion Menu -----
-----

//Memorise les parametre general
if(window.localStorage.getItem('plagetemp') == undefined)
{
    window.localStorage.setItem('plagetemp',Alertes.configGlobalAlert.plagetemp);
} else{

```



```

    $rootScope.Paramgeneral.plagetemp =
    parseInt(window.localStorage.getItem('plagetemp'));
    Alertes.configGlobalAlert.plagetemp =
    parseInt(window.localStorage.getItem('plagetemp'));
  }
  if(window.localStorage.getItem('delaisAlertRes') == undefined)
  {
    window.localStorage.setItem('delaisAlertRes',
    Alertes.configGlobalAlert.delaisAlertRes);
  } else{
    $rootScope.Paramgeneral.delaisAlertRes =
    parseInt(window.localStorage.getItem('delaisAlertRes'));
    Alertes.configGlobalAlert.delaisAlertRes =
    parseInt(window.localStorage.getItem('delaisAlertRes'));
  }
  if(window.localStorage.getItem('nbAlertMem') == undefined)
  {
    window.localStorage.setItem('nbAlertMem',
    Alertes.configGlobalAlert.nbAlertMem);
  } else{
    $rootScope.Paramgeneral.nbAlertMem =
    parseInt(window.localStorage.getItem('nbAlertMem'));
    Alertes.configGlobalAlert.nbAlertMem =
    parseInt(window.localStorage.getItem('nbAlertMem'));
  }

  // quand je clique sur enregistrer
  $rootScope.enregistrerParamGeneral = function() {
    window.localStorage.setItem('plagetemp', $rootScope.Paramgeneral.plagetemp);
    window.localStorage.setItem('delaisAlertRes',
    $rootScope.Paramgeneral.delaisAlertRes);
    window.localStorage.setItem('nbAlertMem',
    $rootScope.Paramgeneral.nbAlertMem);
    angular.copy($rootScope.Paramgeneral, Alertes.configGlobalAlert);

    for (var i = 0; i < ListPatients.patients.length; i++) {

      ListPatients.patients[i].alertes.splice(Alertes.configGlobalAlert.nbAlertMem, ListPatients.patients[i].alertes.length - Alertes.configGlobalAlert.nbAlertMem);
    }
  }

  // Fonction pour supprimer un passient du tableau de ListPatients
  $scope.deletePatient = function(index)
  {
    var confirmPopup = $ionicPopup.confirm({
      title: 'Supprimer patient',
      template: "Etes-vous sûr de vouloir supprimer ce patient ?",
      buttons: [
        {
          text: 'Non',
          type: 'button-default'
        }, {
          text: 'Oui',
          type: 'button-positive',
          onTap: function(e) {
            if((ListPatients.patients[index].alarme == 1) || (ListPatients.patients[index].alarme == 2)){
              Background.remouveName(ListPatients.patients[index].nom);
              $rootScope.nbalert -= 1;
            }
            // deconnecte un ellement de la liste des
            sockets

```

```

        ListPatients.patients[index].socket.MySocketDictionary["freq"].socket.close
    );

    ListPatients.patients[index].socket.MySocketDictionary["alarm"].socket.close
    e();

    // cache le patient
    ListPatients.patients[index].supprimer =
true;
    BDD.Delete("DELETE FROM BbSensor where
id=?", ListPatients.patients[index].id);

    // Glisser pour ne plus afficher les
boutons d'option
    $ionicListDelegate.$getByHandle('liste-
patients').closeOptionButtons();

    }
    }
    }
    });
}

// Fonction pour modifier les données d'un passient du tableau de
ListPatients
$scope.modifPatient = function(index)
{
    // va a la page modifListe en donnant l'id du passient celectionné
    $state.go("modifList",{mIndex:index});
    // Glisser pour ne plus afficher les boutons d'option
    $ionicListDelegate.$getByHandle('liste-
patients').closeOptionButtons();
}

$scope.connectionOnOff = function(index)
{
    if(ListPatients.patients[index].etatConnection == 1)
    {
        ListPatients.patients[index].conectionOn = false;
        // deconnecte un ellement de la liste des sockets

        ListPatients.patients[index].socket.MySocketDictionary["freq"].socket.close
    );

        ListPatients.patients[index].socket.MySocketDictionary["alarm"].socket.close
    e();
    }
    else {
        $scope.connexionSocket(ListPatients.patients[index].ip,
ListPatients.patients[index].port, index);
    }
}

// Fonction pour afficher les alertes d'un passient du tableau des alertes
$scope.infoAlerte = function(index)
{
    // va a la page infoAlerte en donnant l'id du passient celectionné
    $state.go("infoAlerte",{mIndex:index});
}

// Fonction pour afficher les alertes d'un passient du tableau des alertes
$scope.ecg = function(index)
{
    if(ListPatients.patients[index].etatConnection == 1)
    {
        // va a la page infoAlerte en donnant l'id du passient
celectionné

```

```
        $state.go("infoPatient",{mIndex:index});
    }
}

// Gere la mise en veille de l'application
document.addEventListener('deviceready', Background.start, false);

// Gère le bouton "retoure en arrière du téléphone"
$ionicPlatform.registerBackButtonAction(function() {
    console.log($ionicHistory.currentTitle());
    if($ionicHistory.currentTitle() == 'Patients')
    {
        var confirmPopup = $ionicPopup.confirm({
            title: 'Quitter',
            template: "Etes-vous sûr de vouloir quitter
l'application ?",
            buttons: [
                {
                    text: 'Non',
                    type: 'button-default'
                }, {
                    text: 'Oui',
                    type: 'button-positive',
                    onTap: function(e) {
                        ionic.Platform.exitApp();
                    }
                }
            ]
        });
    }
    else{
        $ionicHistory.goBack();
    }
}, 100);
});
```

Addpatient.html

```

<ion-view view-title = "Ajouter un patient">
  <ion-content>
    <div class="list">
      <label class="item item-input-wrapper">
        <span class="input-label">Nom :</span>
        <input type="text" placeholder='Tina' ng-value="newPatient.nom" ng-
model="newPatient.nom">
      </label>
      <label class="item item-input-wrapper">
        <span class="input-label">Chambre :</span>
        <input type="text" placeholder='258b' ng-
value="newPatient.chambre" ng-model="newPatient.chambre">
      </label>
      <button class="button button-full button-energized" ng-click =
"codeBar () ">
        Associer patient (code barre)
      </button>
    </div>
    <div class="list">
      <label class="item item-input-wrapper">
        <span class="input-label">IP :</span>
        <input type="text" placeholder='x.x.x.x' ng-
value="newPatient.ip" ng-model="newPatient.ip">
      </label>
      <label class="item item-input-wrapper">
        <span class="input-label">Port :</span>
        <input type="number" placeholder='8000' ng-
value="newPatient.port" ng-model="newPatient.port" >
      </label>
      <button class="button button-full button-energized" ng-click =
"QRCode () ">
        Associer moniteur (QRCode)
      </button>
    </div>
    <div class="btEnregistre">
      <button class = "button button-block button-positive" ng-click
= "addNewPatient () ">
        Enregistrer
      </button>
    </div>
    <!--</form> -->
  </ion-content>
</ion-view>

```

addpatientCtrl.js

```

// controleur du Weather
app.controller("addPatientCtrl", function($scope, $rootScope, $state,
$stateParams, $interval, ListPatients, BDD, $ionicHistory,
$cordovaBarcodeScanner){

    $scope.newPatient = {};
    angular.copy(ListPatients.modelPatient, $scope.newPatient);

    $scope.QRCode = function()
    {
        // lence le Scanner
        $cordovaBarcodeScanner.scan().then(function(imageData)
        {
            // Si le message contient un ip et un port
            if((imageData.text.search("ip") != -1) &&
(imageData.text.search("port") != -1)){
                var text = imageData.text.replace("ip", "");
                text = text.replace("port", "");
                var val = text.split("-");

                // Recupère la valeur de l'ip
                $scope.newPatient.ip = val[0];
                // Recupère la valeur du port
                $scope.newPatient.port = parseInt(val[1]);

            }
        }, function(error) {
            console.log("An error happened -> " + error);
        });
    }

    $scope.codeBar = function()
    {
        $cordovaBarcodeScanner.scan().then(function(imageData)
        {
            if(imageData.text == '5901234123457'){

                $scope.newPatient.nom = 'Baby';
                $scope.newPatient.chambre = '185b';

            }
        }, function(error) {
            console.log("An error happened -> " + error);
        });
    }

    $scope.addNewPatient = function()
    {
        if(($scope.newPatient.ip != '') && ($scope.newPatient.port != null))
        {
            // Ajoute mon nouveau patient
            ListPatients.addPatient($scope.newPatient);
            // Ajoute le patient dans la base de donnée
            BDD.InsertPatientInBDD("INSERT INTO BbSensor (Nom, Chambre,
IP, Port) VALUES (?,?,?,?)", $scope.newPatient.nom, $scope.newPatient.chambre,
$scope.newPatient.ip, $scope.newPatient.port);
        }
        else {
            alert("Vous n'avez pas scanné le QRCode");
        }
    }
    });

```

modifList.html

```

<ion-view view-title = "Modifier les données du patient">
  <ion-content>

    <div class="General" ng-show="test == false">
      <div class="list" >
        <div class="item item-divider">
          Info patient
        </div>
        <div class="item item-input-inset">
          <span class="input-label">Nom :</span>
          <label class="item item-input-wrapper" ng-
style="{ 'height': '20px' }">
            <input type="text" ng-value="patient.nom" ng-
model="patient.nom">
          </label>
        </div>
        <div class="item item-input-inset">
          <span class="input-label">Chambre :</span>
          <label class="item item-input-wrapper" ng-
style="{ 'height': '20px' }">
            <input type="text" ng-
value="patient.chambre" ng-model="patient.chambre" >
          </label>
        </div>
        <div class="item item-divider">
          Info moniteur
        </div>
        <div class="item item-input-inset">
          <span class="input-label">IP :</span>
          <label class="item item-input-wrapper" ng-
style="{ 'height': '20px' }">
            <input type="text" ng-value="patient.ip"
ng-model="patient.ip">
          </label>
        </div>
        <div class="item item-input-inset">
          <span class="input-label">Port :</span>
          <label class="item item-input-wrapper" ng-
style="{ 'height': '20px' }">
            <input type="number" ng-
value="patient.port" ng-model="patient.port" >
          </label>
        </div>
        <button class="button button-full button-energized" ng-
click = "qrCode()" >
          Associer moniteur (QRCode)
        </button>
      </div>
      <button class="button button-positive" ng-
style="{ 'float': 'left' }" ng-click = "modifPatient()" >
        Enregistrer
      </button>
    </div>

    <div class="Seuil" ng-show="test == true">
      <div class="list">
        <div class="item item-divider">
          Seuil des alertes
        </div>
        <label class="item">
          Fréquence cardiaque:
          <table>
            <tr>

```

Hes·SO  **GENÈVE**
Haute Ecole Spécialisée
de Suisse occidentale

```

                </tr>
            </table>
        </label>
    </div>

    <button class="button button-positive" ng-
style="{ 'float': 'left' }" ng-click="modifSeuil()">
        Enregistrer
    </button>
</div>
</ion-content>
<div class="tabs tabs-icon-only">
    <a class="tab-item" ng-click="associer()">
        Informations générales
    </a>
    <a class="tab-item" ng-click="seuil()">
        Seuils des alertes
    </a>
</div>
</ion-view>
```


modifListCtrl

```

// controleur du Home
app.controller("modifListCtrl", function( $scope, $rootScope, $state,
    $stateParams, $ionicHistory, $ionicPopover, $cordovaBarcodeScanner, ListPatients,
    BDD, Background)
{
    $scope.test = false;

    // l'id du patient a modifier
    $scope.mId = $stateParams.mIndex;

    $scope.patient = {};
    angular.copy(ListPatients.modelPatient, $scope.patient);
    // on recupère les nouvelles données du patient
    $scope.patient.id = ListPatients.patients[$scope.mId].id;
    $scope.patient.nom = ListPatients.patients[$scope.mId].nom;
    $scope.patient.chambre = ListPatients.patients[$scope.mId].chambre;
    $scope.patient.ip = ListPatients.patients[$scope.mId].ip;
    $scope.patient.port = ListPatients.patients[$scope.mId].port;

    $scope.seuils = {};
    angular.copy(ListPatients.patients[$scope.mId].seuils, $scope.seuils);

    // option bouton
    $scope.associer = function() {
        $scope.test = false;
    }
    $scope.seuil = function() {

        $scope.test = true;
    }

    $scope.qrCode = function(){
        $cordovaBarcodeScanner.scan().then(function(imageData)
        {
            if((imageData.text.search("ip") != -1) &&
(imageData.text.search("port") != -1)){
                //var text = imageData.text.slice(2);
                var text = imageData.text.replace("ip", "");
                text = text.replace("port", "");
                var val = text.split("-");
                // Test si le QRCode est bon (Si il posaide une
virgule)

                // Recupère la valeur de l'ip
                $scope.patient.ip = val[0];
                // Recupère la valeur du port
                $scope.patient.port = parseInt(val[1]);
            }
        }, function(error) {
            console.log("An error happened -> " + error);
        });
    };

    $scope.codeBar = function(){
        $cordovaBarcodeScanner.scan().then(function(imageData)
        {
            var val = imageData.text.split(",");

            if(imageData.text == '5901234123457'){

                $scope.patient.nom = 'BbSensor';
                $scope.patient.chambre = '185b';
            }
        }, function(error) {
            console.log("An error happened -> " + error);
        });
    };
}

```

```
};

$scope.modifPatient = function() {
    // deconnaicte un ellement de la liste des sockets

    ListPatients.patients[$scope.mId].socket.MySocketDictionary["freq"].socket.
close();

    ListPatients.patients[$scope.mId].socket.MySocketDictionary["alarm"].socket
.close();

    // cache le patient
    ListPatients.patients[$scope.mId].supprimer = true;
    BDD.Delete("DELETE FROM BbSensor where id=?", $scope.patient.id);

    // Ajoute mon nouvau patient
    ListPatients.addPatient($scope.patient);
    // Ajoute le patient dans la base de donnée
    BDD.InsertPatientInBDD("INSERT INTO BbSensor (Nom, Chambre, IP,
Port) VALUES (?, ?, ?, ?)", $scope.patient.nom, $scope.patient.chambre,
$scope.patient.ip, $scope.patient.port);

    Background.remouveName(ListPatients.patients[$scope.mId].nom);
    $rootScope.nbalert -= 1;
};

$scope.modifSeuil = function() {
    //ListPatients.patients[$scope.mId].seuils = $scope.seuils;
    ListPatients.patients[$scope.mId].seuils.FCmax =
parseInt($scope.seuils.FCmax);
    ListPatients.patients[$scope.mId].seuils.FCmin =
parseInt($scope.seuils.FCmin);
    ListPatients.patients[$scope.mId].socket.send('alarm',
$scope.seuils.FCmin + ';' + $scope.seuils.FCmax);
    $ionicHistory.goBack();
};
});
```

infoAlerte.html

```

<!-- Indique le nom de la vue-->
<ion-view view-title = "Alertes">
  <!-- Indique le contenu de la vue -->
  <ion-content >
    <button class="button button-full button-assertive icon-left ion-
trash-a" ng-if='data.showDelete' ng-click="deleteAll()">Tout supprimer</button>
    <ion-list show-delete="data.showDelete">
      <ion-item class="item-remove-animate info-alert" ng-
repeat="al in lesAlertes" ng-click="signalAlerte($index)">
        <h2>{{al.date}}</h2>
        <p>
          <span>
            {{al.descriptif}}
          </span>
        </p>
        <ion-delete-button
          class="ion-minus-circled" ng-
click="deleteAlerte($index)">
      </ion-delete-button>
    </ion-item>
  </ion-list>

  </ion-content>

  <!-- Ajoute un nouveau bouton a droite pour supprimer les alertes -->
  <ion-nav-buttons side="right" >
    <button class="button button-icon icon ion-ios-minus-outline"
ng-class="{ 'ion-reply': data.showDelete == true}" ng-click="data.showDelete =
!data.showDelete"></button>
  </ion-nav-buttons>
</ion-view>

```

infoAlertCtrl.js

```

// controleur de infoAlerte
app.controller("infoAlerteCtrl", function( $scope, $rootScope, $state,
$ionicLoading, $stateParams, $timeout, ListPatients, Background, $interval,
Alertes)
{
    // POur afficher ou pas l'option de suppression des alertes
    $scope.data = {
        showDelete: false
    };

    $scope.index = $stateParams.mIndex;
    $scope.lesAlertes = ListPatients.patients[$scope.index].alertes;

    var deleteFunc = function(){
        //var repPromisel =
        $interval.cancel(ListPatients.patients[$scope.index].timeoutAlarm.promiseTimeoutC
lignot);
        var repPromisel =
        $interval.cancel(ListPatients.patients[$scope.index].timeoutAlarm.promiseTimeoutC
lignot);
        var repPromise2 =
        $timeout.cancel(ListPatients.patients[$scope.index].timeoutAlarm.promiseTimeoutAl
armNonRes);
        if((repPromisel)&&(repPromise2)){
            ListPatients.patients[$scope.index].clignote = false;
            ListPatients.patients[$scope.index].alarme = 0;
        }
        else{
            if(ListPatients.patients[$scope.index].clignote == false){
                ListPatients.patients[$scope.index].alarme = 0;
            }
        }
        Background.remouveName(ListPatients.patients[$scope.index].nom);
        $rootScope.nbalert -= 1;
    }

    // Fonction pour supprimer une alerte du tableau de ListAlertes
    $scope.deleteAlerte = function(index)
    {
        // Supprime un element de la liste
        $scope.lesAlertes.splice(index,1);

        if(($scope.lesAlertes.length == 0) &&
((ListPatients.patients[$scope.index].clignote == true) ||
(ListPatients.patients[$scope.index].alarme == 2)))
        {
            deleteFunc();
        }
    }
    $scope.deleteAll = function()
    {
        $scope.lesAlertes.splice(0, $scope.lesAlertes.length);
        if((ListPatients.patients[$scope.index].clignote == true) ||
(ListPatients.patients[$scope.index].alarme == 2))
        {
            deleteFunc();
        }
        $scope.data.showDelete = false;
    }

    // Fonction pour afficher les alertes d'un passient du tableau des alertes
    $scope.signalAlerte = function(index)
    {
        var d = new Date();
        /*

```

```

        $ionicLoading.show({
            template: 'Loading...'
        });
    */

    var mtime = $scope.getTimeAlerte(index);
    var started = null;

    var getdataAletre = function(signal){
        var diff = (new Date()).getTime() - started;
        alert("arrive" + diff);
        /*
        Alertes.dataSignalAlerte.nbData = signal.length/2;
        console.log(Alertes.dataSignalAlerte.nbData);

        var arrayDataTemp = [];
        console.log(d.getTime());
        var i;
        for (i = 0; i < signal.length; i+2) {
            if(i < signal.length)
            {
                arrayDataTemp[i/2] = signal[i];
            }
            else{
                console.log("coucou");
            }
        }

        Alertes.dataSignalAlerte.data = arrayDataTemp;
        //Alertes.dataSignalAlerte.dataMin = Math.min.apply(null,
signal);

        //Alertes.dataSignalAlerte.dataMax = Math.max.apply(null,
signal);

        $state.go("graphAlert");
        */
    };
    started = (new Date()).getTime();

    ListPatients.patients[$scope.index].socket.connectSendAndClose("history",
getdataAletre, mtime + ";" + (Alertes.configGlobalAlert.plagetemp * 60 * 250));
}
$scope.getTimeAlerte = function(index){
    var mdate = new Date();
    var tttt = mdate.toLocaleTimeString();
    var time1 = tttt.split(":");

    var time2 = $scope.lesAlertes[index].date;
    time2 = time2.split(" ");
    time2 = time2[4];
    time2 = time2.split(":");

    var min;
    var sec;
    if(parseInt(time1[0]) > parseInt(time2[0]))
    {
        min = parseInt(time1[1]) + (60 - parseInt(time2[1]));
    }
    else{
        min = parseInt(time1[1]) - parseInt(time2[1]);
    }
    sec = parseInt(time1[2]) - parseInt(time2[2]);

    var mtime = (min * 60 * 250) + (sec * 250);
    return mtime;
};
});

```

graphAlert.html

```
<ion-view view-title = "Signal Alerte">
  <ion-content>
    <div class="card">
      <div class="item item-text-wrap">
        <div id='myChart'></div>
        <h2> Amplitude </h2>
        <div>
          <button class = "button button-positive" ng-click =
"clickBottom()" style="fontSize:200%"> + </button>
          <button class = "button button-positive" ng-click =
"clickBottom2()" style="fontSize:200%"> - </button>
        </div>
      </div>
    </div>
  </ion-content>
</ion-view>
```

graphAlertCtrl.js

```

app.controller("graphAlertCtrl", function($scope, $stateParams, $ionicLoading,
$rootScope, Alertes) {
    var min = Alertes.dataSignalAlerte.dataMin;
    var max = Alertes.dataSignalAlerte.dataMax;
    console.log(min);
    console.log(max);
    //var tab_date = ["2016-06-03 14:43:12","2016-06-03 14:43:13","2016-06-03
14:43:14","2016-06-03 14:43:15",
    "2016-06-03 14:43:16","2016-06-03 14:43:17","2016-06-03 14:43:18","2016-06-
03 14:43:19","2016-06-03 14:43:20",
    "2016-06-03 14:43:21","2016-06-03 14:43:22"];

    $scope.clickBottom = function(){
        var val = ((max - min) * 10) / 100;
        min = min + val;
        max = max - val;

        zingchart.exec('myChart', 'zoomin', {
            graphid: 0,
            //ymin: min,
            //ymax: max

            zoomx: false,
            zoomy: true
        })
    };

    $scope.clickBottom2 = function(){
        var val = (((max - min) * 10) / 100) / 2;
        min = min - val;
        max = max + val;

        zingchart.exec('myChart', 'zoomout', {
            graphid: 0,
            //ymin: min,
            //ymax: max
            zoomx: false,
            zoomy: true
        })
    };

    var myConfig = {
        "gui":{
            "behaviors":[
                {
                    // retire l'option
                    "id":"DownloadSVG",
                    "enabled":"none"
                },
                {
                    "id":"Reload",
                    "enabled":"none"
                },
                {
                    "id":"SaveAsImage",
                    "enabled":"none"
                },
                {
                    "id":"Print",
                    "enabled":"none"
                },
                {
                    "id":"HideGuide",
                    "enabled":"none"
                }
            ]
        }
    }

```

```

        "id": "ViewSource",
        "enabled": "none"
    },
    {
        "id": "About",
        "enabled": "none"
    },
    {
        "id": "ShowAll",
        "enabled": "none"
    },
    {
        "id": "DownloadPDF",
        "enabled": "none"
    }
    /*,
    { // Ajoute ceci au menu
        "id": "CrosshairHide",
        "enabled": "all"
    }
    */
]
},
"graphset": [{
    "type": "line",
    "title": {
        "text": "Historique",
        "font-family": "Georgia",
        "font-size": 16
    },
    "plot": {
        "aspect": "spline",
        "line-width": 1, // Taille de la ligne
        "line-color": "#666699", // couleur du traits
        "marker": {
            "size": 3, // taille des points
            "background-color": "#64b4ce #e0e0eb", // couleur des point
            "border-width": 1,
            "border-color": "#666699" // couleur du contour des points
        },
        "tooltip": {
            "visible": false
        }
    },
    "plotarea": {
        "margin-top": "15%", // Marge en decu du tableau
        "margin-bottom": "35%" // Marge en dessous du tableau
    },

    "scale-x": {
        "values": "0:" + Alertes.dataSignalAlerte.nbData + ":1", // data
a afficher dans le second graph
        "max-items": 11, // nombre de point sur l'axe des X
        "zooming": true, // Indique q'on peut zoomer
        "zoom-to": [Alertes.dataSignalAlerte.nbData/2 -1000,
Alertes.dataSignalAlerte.nbData/2 +1000], // valeur a afficher dans le tableau
zoomé entres les 2 val
        "item": {
            "font-size": 10 // tailles des valeur x
        }
    },
    "preview": {
        "margin-bottom": "12%"
    },
    "crosshair-x": {
        "plot-label": {
            "text": "%v" // Affiche la valeur du point quand on clique decu

```



```

    },
    "scale-label": {
      "visible": true // affiche la valeur en x du point sélectionné
    },
    "marker": {
      "size": 3, // taille du point quand on selectionne un point
      "background-color": "#64b4ce #ff3300", // couleur du point
      "border-width": 1,
      "border-color": "#666699" // Coiuleur du contour du point
    }
  },
  "scale-y": {
    "zooming": true,
    // "zoom-to": [min+1, max-1],
    "values": min + ":" + max + ":500", // valeur a afficher de 150 à
350 avec des saut de 50
    "guide": {
      "line-style": "dotted"
    },
    "item": {
      "font-size": 10 // taille des val axe Y
    }
  },
  "crosshair-y": {
    "type": "multiple",
    "scale-label": {
      "visible": true // affiche la valeur en y du point sélectionné
    }
  },
  "series": [{
    "values" : Alertes.dataSignalAlerte.data
  }]
}];
zingchart.render({
  id: 'myChart',
  data: myConfig,
  height: "100%",
  width: "100%",
});

zingchart.complete = function() {
  $ionicLoading.hide();
}
});

```

infoPatient.html

```
<ion-view cache-view="false" view-title = "ECG">
  <ion-content>
    <div class="le_graph">
      <canvas id="monitor" height="750" width="1280"
style="background-color: black;"> </canvas>
    </div>
  </ion-content>
</ion-view>
```

infoPatientCtrl

```
// controleur du Home
app.controller("infoPatientCtrl", function( $scope, $rootScope, $state,
$stateParams, Socket, ListPatients, $ionicHistory)
{
  $scope.index = $stateParams.mIndex;

  // Global function showing the signals
  *****
  function monitorFunction(document) {
    var navbar = document.getElementsByClassName("mynavbar");

    const monitor = document.getElementById("monitor");
    const ctx = monitor.getContext("2d");
    ctx.fillStyle = "#dbbd7a";
    ctx.fill();

    const MARGIN_MONITOR_RIGHT = Math.floor(monitor.width - monitor.width / 7);
    // changer de 6 a 7
    const MARGIN_MONITOR_LEFT = Math.floor(monitor.width / 50);
    const MARGIN_BLOCK = 8;
    const SIZE_INFO = 60; // 15 a 30
    const NB_BLOCK = 3;
    const OVERLAP_SIZE = 4;
    const PIXEL_STEP = 2;

    // Information sur le côté du moniteur
    function drawInfo(txt, color, style, positionX) {
      ctx.fillStyle = color;
      ctx.font = style;
      ctx.fillText(txt, monitor.width - 160, positionX); // 40 a 160 indique de
combien il dois se decaler verre la droite (text info)
    }

    function drawWave() {
      function positionCenterY(no_block) {
        return ((no_block * 2) + 1) / (NB_BLOCK * 2) * monitor.height;
      }
      function positionTopY(no_block) {
        return no_block * 2 / (NB_BLOCK * 2) * monitor.height;
      }
      function positionBottomY(no_block) {
        return (no_block * 2 + 2) / (NB_BLOCK * 2) * monitor.height;
      }
    }

    function drawLine(x0, y0, x1, y1) {
      ctx.beginPath();
      ctx.moveTo(x0, y0);
      ctx.lineTo(x1, y1);
      ctx.stroke();
    }

    function drawLines(){
```

```

        ctx.strokeStyle = 'white';
        ctx.lineWidth = "1.0";

        // +0.5 avoiding antialiasing !
        var tab = [0,1,2,3,4,5,6,7,8,9];
        tab.forEach(
            //Array.from(Array(10).keys()).forEach(
                function (i) {
                    drawLine(0, positionBottomY(i) + 0.5, monitor.width,
positionBottomY(i) + 0.5);
                }
            );
        drawLine(MARGIN_MONITOR_RIGHT + 5.5, 0, MARGIN_MONITOR_RIGHT + 5.5,
monitor.height);
    }

    function minMaxDraw(min, value, max) {
        return Math.max(min + MARGIN_BLOCK / 2, Math.min(max - MARGIN_BLOCK /
2, value));
    }

    function minMax(min, value, max) {
        return Math.max(min, Math.min(max, value));
    }

    drawLines();

    // Class Signal :
    // - monitorPosition: monitorPosition du signal (slot) sur le moniteur
    // - signalPosition: position du signal dans le message reçu par le
websocket
    // - info: Nom du signal abrégé
    // - signals: liste des informations reçues à afficher (flux continu de
données et informations chaque seconde)
    // - color: Couleur du signal
    // - scaleLead: fonction pour l'affichage des limites du signal dans son
slot
    function Signal(monitorPosition, signalPosition, info, signals, color,
scaleLead)
    {
        this.info = info;
        this.x = MARGIN_MONITOR_LEFT;
        this.color = color;
        this.value = [positionCenterY(monitorPosition), 0];

        // Flux du signal
        this.drawFlowSignal = function (signal) {
            if((signal[0] != 'c') && (signal[0] != 'd')){
                ctx.lineWidth = "2";
                this.x = (this.x - MARGIN_MONITOR_LEFT + PIXEL_STEP)
                    % (MARGIN_MONITOR_RIGHT - MARGIN_MONITOR_LEFT)
                    + MARGIN_MONITOR_LEFT;
                ctx.strokeStyle = this.color;

                signal = parseInt(-signal.split(";")[signalPosition]);
                this.value[1] = scaleLead(signal);
                this.value[1] = minMaxDraw(
                    positionTopY(monitorPosition),
                    this.value[1] + positionCenterY(monitorPosition),
                    positionBottomY(monitorPosition) + monitorPosition - 1);

                drawLine(this.x - PIXEL_STEP, this.value[0], this.x, this.value[1]);
                this.value[0] = this.value[1];
                this.clearOverlap(this.x);
            }
        }
    }

```

```

    };

    // Information ponctuelle
    this.basicSignal = function (signal) {
        if((signal[0] != 'c') && (signal[0] != 'd')){
            const topMargin = 20;
            ctx.clearRect(MARGIN_MONITOR_RIGHT + 10,
positionTopY(monitorPosition) + topMargin, monitor.width - MARGIN_MONITOR_RIGHT,
monitor.height / NB_BLOCK - MARGIN_BLOCK / 2 - topMargin);

            // Information textuelle du signal
            drawInfo(this.info, this.color, "bold 22px Arial",
positionCenterY(monitorPosition) - SIZE_INFO / 2); // 12 a 22
            signal = parseInt(signal.split(";")[signalPosition]);
            drawInfo(signal, this.color, "bold " + SIZE_INFO + "px Arial",
positionCenterY(monitorPosition) + SIZE_INFO / 2);
        }
    };
    this.basicSignal("0;0");

    // Effaçage du signal précédent
    this.clearOverlap = function (x){
        // Condition utilisée pour effacer le début du signal
        const N = (x === MARGIN_MONITOR_LEFT) ? x - 4 : x;
        ctx.clearRect(

            minMax(0, N + 1, MARGIN_MONITOR_RIGHT), positionTopY(monitorPosition)
+ 2, OVERLAP_SIZE + PIXEL_STEP, positionBottomY(monitorPosition) -
positionTopY(monitorPosition) - 4
        );
    };

    var js = Socket.construct("ws://" + ListPatients.patients[$scope.index].ip
+ ":" + ListPatients.patients[$scope.index].port);
    js.register(signals[0], this.drawFlowSignal.bind(this));
    js.register(signals[1], this.basicSignal.bind(this));

}

// INSTANCES DES SIGNAUX AVEC FONCTION CALLBACK POUR AFFICHER LE SIGNAL
new Signal(0, 0, "BPM1", ["lead", "freq"], "green", function (value) {
    return value / 200;
});
/*
    new Signal(1, 1, "BPM2", ["lead", "freq"], "blue", function (value) {
        return value / 100;
    });
*/
}
drawWave();
}
monitorFunction(document);
});

```

Alertes.js

```

app.service('Alertes', function($rootScope, $interval, $timeout, ListPatients,
Background){
    this.modelAlert = {type:'', valeur:0, date:''};
    this.configGlobalAlert = {plagetemp:1, delaisAlertRes:30, nbAlertMem:6};
    this.dataSignalAlerte = {data:[], nbData:0, dataMin:0, dataMax:0}

    this.indiqueAlerte = function(id, seuil){
        var temp = [];
        var date = new Date();
        date = date.toString() + " " + date.toLocaleTimeString();
        // Ajoute le nouvelle alerte
        ListPatients.patients[id].alertes.splice(0, 0, {descriptif: "Fc "+ seuil,
date: date});
        // Si il y plus de X alerte memorisé
        if(ListPatients.patients[id].alertes.length >
this.configGlobalAlert.nbAlertMem)
        {
            // Supprime alerte en trop
            var nbAlertM = this.configGlobalAlert.nbAlertMem;
            $timeout(function() {
                ListPatients.patients[id].alertes.splice(nbAlertM,
ListPatients.patients[id].alertes.length - nbAlertM);
            }, 0);
        }

        // Fonction de clignotement
        var clignoteFunc = function(){
            if(ListPatients.patients[id].clignote == true)
            {
                if(ListPatients.patients[id].alarme == 1){
                    $timeout(function() {
                        ListPatients.patients[id].alarme = 0;
                    },0);
                }
                else{
                    $timeout(function() {
                        ListPatients.patients[id].alarme = 1;
                    },0);
                }
                //ListPatients.patients[id].timeoutAlarm.promiseTimeoutClignot =
$timeout(clignoteFunc, 500);
            }
        }
        var alertNonResFunc = function() {
            var repPromise =
$interval.cancel(ListPatients.patients[id].timeoutAlarm.promiseTimeoutClignot);
            if(repPromise){
                $timeout(function() {
                    ListPatients.patients[id].clignote = false;
                    ListPatients.patients[id].alarme = 2;
                },0);
            }
        }
        if((ListPatients.patients[id].alarme == 0) &&
(ListPatients.patients[id].clignote == false)){
            ListPatients.patients[id].clignote = true;
            // Fais clignoter
            ListPatients.patients[id].timeoutAlarm.promiseTimeoutClignot =
$interval(clignoteFunc, 500);

            // Indique que alerte pas recente
            ListPatients.patients[id].timeoutAlarm.promiseTimeoutAlarmNonRes =
$timeout(alertNonResFunc, this.configGlobalAlert.delaisAlertRes * 1000);

            // Fait vibrer 1sec
            navigator.notification.vibrate(1000); // Fait vibrer X milisec

```

```
// fais sonner 1 fois
navigator.notification.beep(1); // fais sonner n fois

// Affiche un message aletre
/*
navigator.notification.alert(
    message,          // message
    callback,         // callback
    title,            // title
    buttonName        // buttonName
);
*/
Background.addNameAlert(ListPatients.patients[id].nom);
$scope.nbalert += 1;
}
});
```

Background.js

```

app.service('Background', function($rootScope, $timeout){
  this.textAlerte = 'Alerte patient:  ';
  var self = this;

  this.start = function () {
    // customization
    cordova.plugins.backgroundMode.setDefaults({
      title: 'BbSensor',
      ticker: 'BbSensor',
      text: 'Aucune alerte détectée'});
    // Enable background mode
    cordova.plugins.backgroundMode.enable();
    cordova.plugins.backgroundMode.onactivate = function()
    {
      if(self.textAlerte == 'Alerte patient:  ')
      {
        // il faut faire un timeout parceque quand on rentre dans onactivate, le
        // plugin remait le texte par défaut
        // donc pour pas rentrer en conflit il faut le faire après
        $timeout(function()
        {
          cordova.plugins.backgroundMode.configure({
            text: 'Aucune alerte détectée'
          });
        }, 30);
      }
      else{
        // il faut faire un timeout parceque quand on rentre dans onactivate, le
        // plugin remait le texte par défaut
        // donc pour pas rentrer en conflit il faut le faire après
        $timeout(function()
        {
          cordova.plugins.backgroundMode.configure({
            text: self.textAlerte
          });
        }, 30);
      }
    };
  };
  this.addNameAlert = function(nom){
    this.textAlerte += (nom + ', ');
    cordova.plugins.backgroundMode.configure({
      text: this.textAlerte
    });
  };
  this.removeName = function(nom){
    this.textAlerte = this.textAlerte.replace(nom + ', ', '');
    if(this.textAlerte == 'Alerte patient:  ')
    {
      cordova.plugins.backgroundMode.configure({
        text: 'Aucune alerte détectée'
      });
    }
    else{
      cordova.plugins.backgroundMode.configure({
        text: this.textAlerte
      });
    }
  };
  this.stopAlert = function(){
    this.textAlerte = 'Alerte patient:  ';
  }
});

```

BDD.js

```

app.service('BDD', function($rootScope, ListPatients, $ionicHistory){

    this.bdd = null;

    /* Crée la BDD. -----
    param: le nom de la BDD
    param: location
    return: la BDD si ok, sinon false.
    */
    this.CreatBDD = function(nomBDD, locat){
        this.bdd = window.sqlitePlugin.openDatabase ({name: nomBDD, location:
locat});
        if(this.bdd != null)
        {
            //alert('BDD créé');
            return this.bdd;
        }
        return false;
    };

    /* Crée une table dans la BDD. -----
    param: La requette SQL (CREATE TABLE IF NOT EXISTS)
    return: true si ok, sinon false.
    */
    this.CreatTable = function (SQL_Comande) {
        // Crée la table dans la base de donnée
        this.bdd.transaction(function(transaction)
        {
            transaction.executeSql(SQL_Comande, [],
            // Accept
            function(tx, result)
            {
                return true;
            },
            // Error
            function(error) {
                alert("Erreur lors de la création de la table");
                return false;
            }
        ));
    };

    /* Recupère les données dans la BDD. -----
    param: La requette SQL (SELECT)
    return: --.
    */
    this.SelectAllInBDD = function(requet){
        this.bdd.transaction(function(transaction)
        {
            // Recupère tous de la table BbSensor
            transaction.executeSql(requet, [], function (tx, results)
            {
                ListPatients.getAllPatientBDD(results.rows);
                // indique a home que la BDD est prête
                $rootScope.$broadcast('BddOK');
                //return results.rows.length;
                //results.rows.item(i).id;
                //results.rows.length;
            }, null);
        });
    };

    /* Insérer dans la BDD. -----
    param: La requette SQL` (CREATE TABLE IF NOT EXISTS)

```



```

    return: --.
    */
    this.InsertPatientInBDD = function(requet){
        var tabVal = [];
        for (var i = 1; i < arguments.length; i++){
            tabVal.push(arguments[i]);
        }

        // Incertion dans la BDD
        this.bdd.transaction(function(transaction) {
            //transaction.executeSql(arguments[0], tabVal
            transaction.executeSql(requet, tabVal, function(tx, result) {
                // Recupère l'id du patient
                transaction.executeSql('SELECT * FROM BbSensor', [], function (tx,
results) {
                    //ListPatients.patients[ListPatients.patients.length-1].id =
results.rows.item(results.rows.length-1).id;
                    var id = results.rows.item(results.rows.length-1).id;
                    ListPatients.patients[ListPatients.patients.length-1].id = id;
                    $ionicHistory.goBack();
                }, null);
            },
            function(error){
                console.log('Error occurred');
            });
        });
    };
    /* Supprime un patient de la BDD. -----
    param: La requette SQL (SELECT)
    return: --.
    */
    this.Delete = function(requet, id){
        this.bdd.transaction(function(transaction)
        {
            transaction.executeSql(requet, [id],
            //On Success
            function(tx, result) {
            },
            //On Error
            function(error){
                alert('Erreur de suppression DB');
            });
        });
    };
});

```

ListePatient.js

```

app.service('ListPatients', function($rootScope){

    this.Paramgeneral = {};

    this.patients = [];
    // joel
    //$scope.patient = {id: null, nom:'', Fc: 0, SpOz: 0, FR: 0, mip:
    "192.168.2.7", mport: 5000, clignote:false, supprimer: false, etatConnection: 0};
    // moi
    //$scope.newPatient = {id: null, nom:'', Fc: 0, SpOz: 0, FR: 0, ip:
    "129.194.185.76", port: null, clignote:false, supprimer: false, etatConnection:
    0};
    this.modelPatient = {id: null, nom:'', chambre:'', Fc: 0, SpOz: 0, FR: 0,
    ip: "", port: null, clignote:false, supprimer: false, etatConnection: 0,

    alarme: 0, timeoutAlarm:{}, alertes:[], seuils:{}, intervalFunc:null,
    socket:null, conectionOn:true};
    this.modelPatient.seuils = {FCmin: 60, FCmax:100, FRmin: 30, FRmax: 50,
    SP02min: 90, SP02max: 100};
    this.modelPatient.timeoutAlarm = {promiseTimeoutClignot: null,
    promiseTimeoutAlarmNonRes: null};

    /* Ajoute un patient dans le tableau de patient. -----
    -----
    param: le patient
    return: --.
    */
    this.addPatient = function (patient) {
        this.patients.push(patient);
    };

    /* Recupère tous les patient de la BDD et les ajoute au tableau de patient.
    -----
    param: valeur retourné pare la BDD contenant les patients <.item(i)>
    return: --.
    */
    this.getAllPatientBDD = function (valeur) {
        //ajout au tableau des Patient les patient mémorisé dans la BDD
        for (var i = 0; i < valeur.length; i++)
        {
            var mpatient = {};
            angular.copy(this.modelPatient, mpatient);
            mpatient.id = valeur.item(i).id;
            mpatient.nom = valeur.item(i).Nom;
            mpatient.chambre = valeur.item(i).Chambre;
            mpatient.ip = valeur.item(i).IP;
            mpatient.port = valeur.item(i).Port;

            this.addPatient(mpatient);
        }
    };
});

```

Socket.js

```

app.service('Socket', function($rootScope, $q, ListPatients){
    this.construct = function(adresse) {
        return new JoelSocket(adresse);
    };

    var ServerEnum = {
        REPTAR_WIFI: 0,
        REPTAR_WIRE: 1,
        PROXY: 2
    }

    var MessageType = {
        NEW_FREQ: 1,
        MIN_FREQ: 2,
        MAX_FREQ: 4
    }

    this.connect = function(adresse, id){
        var js = new JoelSocket(adresse);
        ListPatients.patients[id].socket = js;
    };

    // The JoelSocket simplify the websocket management
    *****
    function JoelSocket(adresse){
        // "use strict";
        var updateAndSocketDictionary = {};
        this.MySocketDictionary = updateAndSocketDictionary;

        this.connectSendAndClose = function(signalName, func, message){
            var socket = new WebSocket(adresse, signalName);
            socket.onopen = function(e){
                socket.send(message);
                socket.binaryType = 'arraybuffer';
                socket.onmessage = function(signal){

                    var historics = [];
                    var packet = new DataView(signal.data);

                    for(var i = 0; i < packet.byteLength/4; i++){
                        historics.push( packet.getInt16(i*4,
true), packet.getInt16(i*4+2, true));
                    }
                    func(historics);
                    socket.close();
                }
            }

            // Register a function with a signal name. If the signal
            // already exists, it register only the function and it doesn't
create
            // new sockets.
            this.register = function(signalName, updateFunction, id)
            {
                // si la connection n'est pas encore etablie
                if(!(signalName in updateAndSocketDictionary)){
                    var socket = new WebSocket(adresse, signalName);
                    socket.binaryType = 'arraybuffer';
                    updateAndSocketDictionary[signalName] = {socket:
socket, lstFunc: [], id: id};
                }

                // Add the new update function

                updateAndSocketDictionary[signalName].lstFunc.push(updateFunction);

```

```

        updateAndSocketDictionary[signalName].socket.onopen =
function(signal){
        updateAndSocketDictionary[signalName].lstFunc.forEach(
            function (update){
                update('connect',
updateAndSocketDictionary[signalName].id);
            }
        );

        // Change the callback function calling when a message arrive :
update all functions
        // register to the signal name.
        updateAndSocketDictionary[signalName].socket.onmessage =
function(signal){
            /* This section reproduce the old string buffered
protocol. */
            var packet = new DataView(signal.data);
            if(signalName === "lead" || signalName == "freq"){
                var num1 = packet.getInt16(0, true);
                var num2 = packet.getInt16(2, true);
                signal = num1 + ";" + num2;
            }else if(signalName === "alarm"){
                var type = packet.getInt16(0, true);
                var num1 = packet.getInt16(2, true);
                var num2 = packet.getInt16(4, true);
                if(type === MessageType.NEW_FREQ){
                    signal = "NEW_FREQ;" + num1 +
";" + num2;
                }else if(type === MessageType.MIN_FREQ){
                    signal = "MIN_FREQ;" + num1 +
";" + num2;
                }else if(type === MessageType.MAX_FREQ){
                    signal = "MAX_FREQ;" + num1 +
";" + num2;
                }
            }

            _.forEach(
                updateAndSocketDictionary[signalName].lstFunc,
                function(updateFunc){
                    updateFunc(signal,
updateAndSocketDictionary[signalName].id);
                }
            );

            }.bind(this);

            // Send -1 in the function if socket closed
            updateAndSocketDictionary[signalName].socket.onclose =
function(){
                updateAndSocketDictionary[signalName].lstFunc.forEach(
                    function (update){
                        update('disconnect',
updateAndSocketDictionary[signalName].id);
                    }
                );
            }

            this.send = function (signalName, value) {
                var socket = updateAndSocketDictionary[signalName].socket;
                socket.send(value);
            }
        }
    });

```