

P2022 : SunShare  
Nicolas LHOMMEAU

## Dossier technique du projet - partie individuelle

### Table des matières

<b>1 - SITUATION DANS LE PROJET</b>	<b>3</b>
1.1 - RAPPEL DES TÂCHES PROFESSIONNELLES À RÉALISER	3
1.2 - PRÉSENTATION DE LA PARTIE PERSONNELLE	3
1.2.1 - Introduction	3
1.2.2 - Chronologie du projet	4
1.2.3 - Synoptique de la réalisation	5
1.2.4 - Technologies utilisées	6
1.2.5 - Diagramme de déploiement	7
1.2.6 - Diagramme d'exigences	8
1.2.7 - Aperçu de l'application	8
1.2.8 - Diagramme de cas d'utilisation	10
1.2.9 - Schéma de la base de données	10
1.2.10 - Diagramme d'activité	11
<b>2 - RÉALISATION DE LA TÂCHE PROFESSIONNELLE "RÉCUPÉRATION DES DONNÉES"</b>	<b>12</b>
2.1 - DIAGRAMME DE SÉQUENCE	12
2.2 - CAS D'UTILISATION	12
2.3 - CONCEPTION DÉTAILLÉE	13
2.4 - TEST UNITAIRE	14
2.4.1 - Procédure de test	14
2.4.2 - Rapport d'exécution	16
<b>3 - RÉALISATION DE LA TÂCHE "AFFICHER LES DONNÉES SOUS FORME GRAPHIQUE"</b>	<b>16</b>
3.1 - DIAGRAMME DE SÉQUENCE	16
3.2 - CAS D'UTILISATION	17
3.3 - CONCEPTION DÉTAILLÉE	17
3.4 - TESTS UNITAIRES	19
3.4.1 - Procédure de tests	19
3.4.2 - Rapport d'exécution	20
<b>4 - RÉALISATION DE LA TÂCHE "AFFICHER LES DONNÉES EN FONCTION D'UNE PÉRIODE CHOISIE"</b>	<b>21</b>
4.1 - DIAGRAMME DE SÉQUENCE	21
4.2 - CAS D'UTILISATION	21
4.3 - CONCEPTION DÉTAILLÉE	22
4.4 - TESTS UNITAIRES	23
4.4.1 - Procédure de tests	23
4.4.2 - Rapport d'exécution	24

<b>5 - BILAN DE LA RÉALISATION PERSONNELLE.....</b>	<b>24</b>
5.1 - RAPPEL DES TÂCHES PERSONNELLES À RÉALISÉES.....	24
5.2 - AMÉLIORATIONS POSSIBLES.....	24
5.3 - CONCLUSION PERSONNELLE.....	24
5.3.1 - Points positifs.....	24
5.3.2 - Points négatifs.....	25

## 1 - Situation dans le projet

### 1.1 - Rappel des tâches professionnelles à réaliser

Tâches	Description	Validation
FP1	<b>Visualiser un résumé de données</b> Par jours, semaines, mois ou années	En cours
FP2	<b>Visualiser les indicateurs sous forme de tableaux simples</b> En temps réel ou par période	Effectué
FP3	<b>Visualiser les indicateurs sous forme de graphiques détaillés</b> En temps réel ou par période	Effectué

### 1.2 - Présentation de la partie personnelle

#### 1.2.1 - Introduction

*Le projet consiste en un système permettant de suivre l'évolution de l'énergie (consommation, injection et production [voir page 5]) provenant de panneaux solaires ou éoliennes depuis son smartphone ou son PC. Le compteur Linky va envoyer les données dans la base de données à l'aide de la raspberry. Les applications web et smartphone permettent à l'utilisateur de suivre les données sous forme graphique ou historique, le tout avec une utilisation simple et agréable. Le projet se veut open source et simple à installer suite à un accord avec le client.*

L'objectif de ma partie personnelle est de réaliser une application pour smartphone, permettant à son utilisateur de visualiser ses données en temps réel ou par période (choisie par l'utilisateur). L'application doit être compatible avec une majorité de smartphone (versions Android et iOS) et être utilisable facilement. Enfin, elle doit permettre à l'utilisateur de choisir entre un affichage simple des données (c'est-à-dire sous forme de tableaux) et un affichage plus détaillé (sous forme de graphiques).

### 1.2.2 - Chronologie du projet

#### **3 Janvier – 21 Janvier : Découverte du projet**

- Prise de connaissance du cahier des charges et répartition des tâches
- Prise en main des différents outils de gestion (Trello et GitLab)
- Préparation des postes (installation Android Studio pour son émulateur)

#### **24 Janvier – 11 Février : Sprint 1**

- Choix de framework et création d'une application basique
  - Image de chargement de l'application
  - Création des 3 onglets 'Résumé de données', 'Temps réel', 'Période de données'
- Diagrammes SYSML

#### **14 Février – 25 Mars : Sprint 2**

- Diagrammes SYSML (fin)
- Accès à la base de données via l'application
  - Requêtes fetch() pour récupérer les valeurs
  - Affichage des valeurs sur l'application

#### **28 Mars – : Sprint 3**

- Affichage données instantanées sur l'application
- Mise en place du choix de l'affiche graphique ou historique

Les deux premières semaines du projet ont permis la découverte du cahier des charges et la répartition des tâches. Elles ont aussi permis de préparer les outils de gestion GitLab et Trello ainsi que la préparation de mon poste (installation Android Studio).

Pour le premier sprint, le principal objectif était de prendre en main les outils que j'allais utiliser tout au long du projet mais également de comprendre la demande du client : créer un moyen facile et efficace pour n'importe quel utilisateur d'accéder à ses données énergétiques en temps réel ou des jours passés via une application smartphone. Ce sprint m'a notamment permis de faire le choix constituant la base du projet : le choix du framework pour développer l'application smartphone.

Le second sprint a principalement été utilisé afin de créer/finir les différents diagrammes (déploiement, séquence, ..) permettant de mieux comprendre le projet. Cela nous a également permis de nous mettre d'accord sur certains points pas encore abordés à ce moment-là. C'est également dans ce sprint que la connexion entre la base de données et l'application est créée, permettant à cette dernière d'afficher les données sur le smartphone.

Enfin, dans le troisième et dernier sprint, l'affichage des données en temps réel est possible sur l'application smartphone. Ce sprint est plus long dû aux périodes de vacances et principalement car il constitue la phase finale de développement du projet. C'est aussi dans ce sprint que les derniers détails seront mis en place et que l'idée principale du projet sera finalisée.

### 1.2.3 - Synoptique de la réalisation

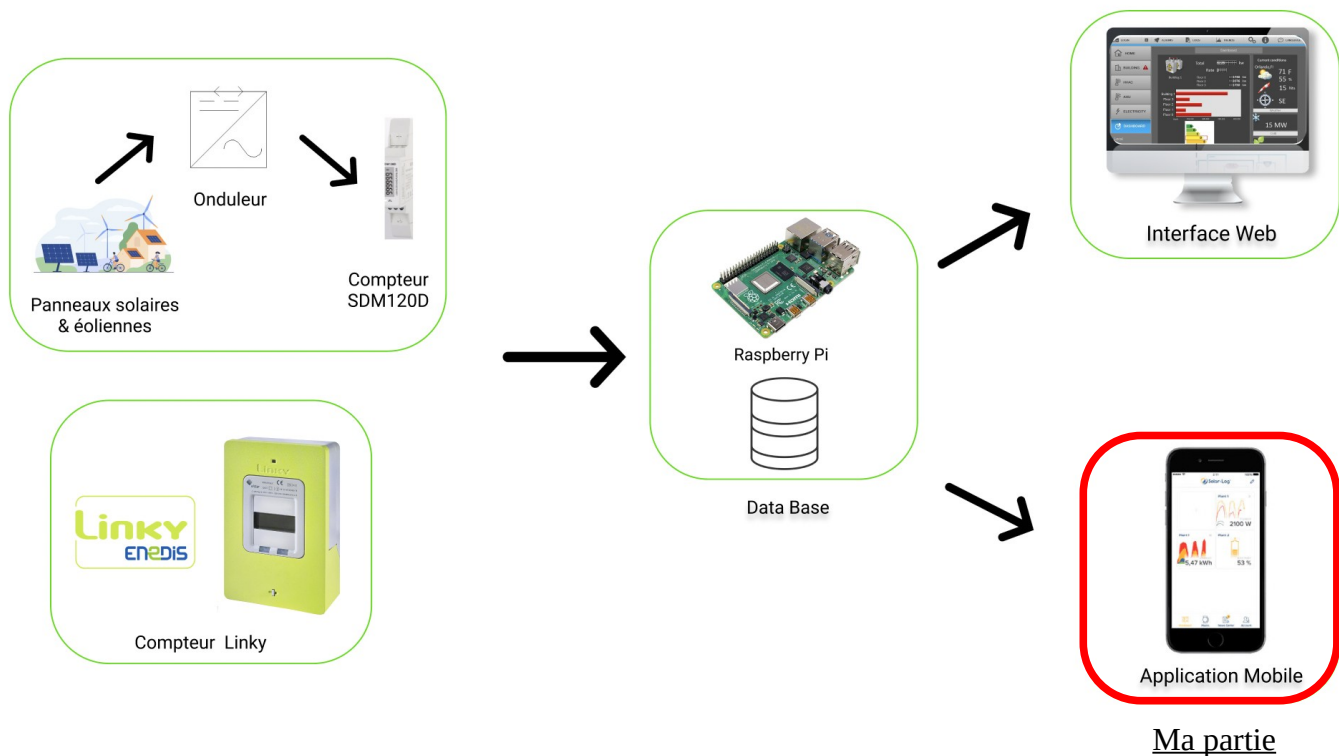


Figure 1: Synoptique général du système

Dans un premier temps, on retrouve ENEDIS, gestionnaire du réseau électrique, responsable de la distribution d'énergie électrique en France. Il y a ensuite le compteur Linky qui mesure les indicateurs de consommation des foyers pour l'envoyer à ENEDIS et permet aux particuliers d'accéder aux données grâce à la TIC (protocole de Télé-Information Client). Les indicateurs sont les suivants :

- Soutirage : énergie fournie par ENEDIS et consommée dans le foyer
- Production : énergie produite localement (solaire, éolien) et consommée dans le foyer
- Injection : surplus d'énergie produite localement donné ou vendu à ENEDIS

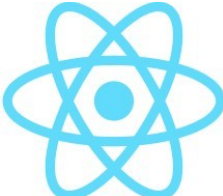






La TIC du Linky permet aux particuliers de collecter les informations relatives à la consommation, le soutirage et l'injection. La TIC se récupère par l'intermédiaire d'une interface unidirectionnelle (disponible uniquement en lecture) en liaison série RS485.

Un autre compteur (le SDM120D) sert uniquement à contrôler et certifier l'énergie produite localement (panneaux solaires, éoliennes) et met à disposition un signal en sortie sous forme d'impulsions correspondant à la production locale en kW/h.

Vient ensuite le Raspberry Pi, récupérant les indicateurs grâce à la TIC pour les stocker dans une base de données et permettre aux particuliers d'accéder aux données par l'intermédiaire d'une application (pour mobile) ou d'une interface web (pour navigateur).

### 1.2.4 - Technologies utilisées

Certaines technologies nous étaient imposées mais j'avais le choix pour le framework a utilisé, j'ai donc pu en tester plusieurs. Voici les technologies gardées et utilisées tout au long du projet.

Technologie	Description
 React Native	<ul style="list-style-type: none"> <li>Créé par Facebook</li> <li>Framework Open-source</li> <li>Permet de développer des applications pour Android, iOS et UWP (Universal Windows Platform)</li> </ul>
 MariaDB	<ul style="list-style-type: none"> <li>SGBD (Système de gestion de base de données) libre et open source</li> <li>Fork de MySQL</li> <li>Facilité de déploiement et de prise en main et très bonne intégration dans l'environnement Apache/PHP</li> </ul>
 Figma	<ul style="list-style-type: none"> <li>Plateforme collaborative qui permet de faire du prototypage</li> <li>Figma nous aide dans le développement d'interface graphique, on a d'abord fait des prototypes sur Figma avant de se lancer sur la création de notre site web et l'application mobile</li> </ul>
 Trello	<ul style="list-style-type: none"> <li>Un outil de gestion de projet gratuite, qui nous permet d'organiser notre projet sous forme de tableaux, eux-mêmes composés de listes en colonnes, qui répertorient des tâches sous formes de cartes.</li> </ul>
 GitLab	<ul style="list-style-type: none"> <li>Plateforme de développement collaborative</li> <li>Open-source</li> <li>Permet de piloter des dépôts de code source et d'en gérer les différentes versions</li> </ul>
 Visual Paradigm	<ul style="list-style-type: none"> <li>Outil de création de tous types de diagrammes SYSML/UML</li> </ul>
 TypeScript	<ul style="list-style-type: none"> <li>Langage de programmation</li> <li>Open source et libre</li> <li>Développé par Microsoft</li> <li>Pour améliorer le développement JavaScript</li> </ul>

### 1.2.5 - Diagramme de déploiement

Ici, on retrouve l'intégralité du système sous forme de diagramme de déploiement afin de mieux comprendre les relations entre chaque éléments du projet.

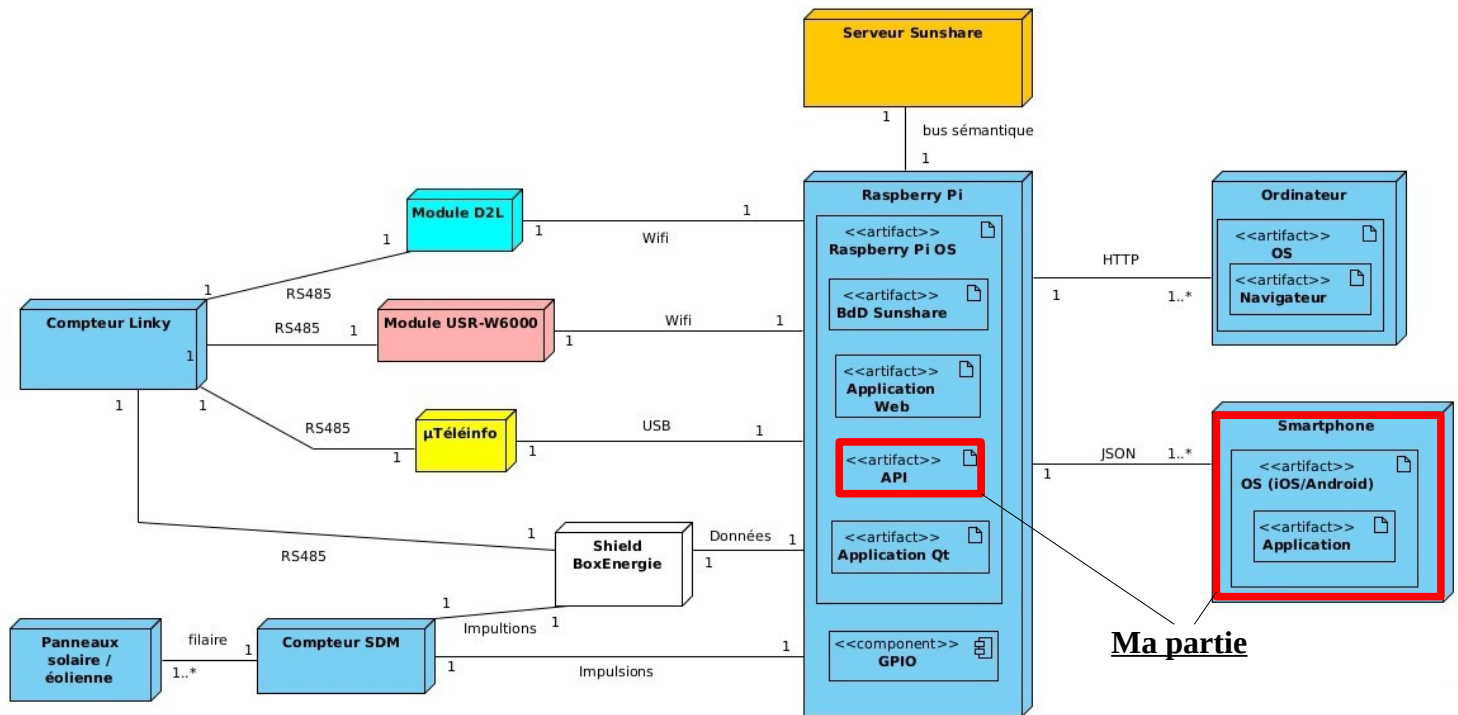


Figure 2: Diagramme de déploiement

Je me suis occupé de la partie Smartphone et j'ai contribué à la création de l'API.

Le compteur Linky permet de récupérer les informations d'injection et de soutirage et le compteur SDM permet de récupérer les informations de production.

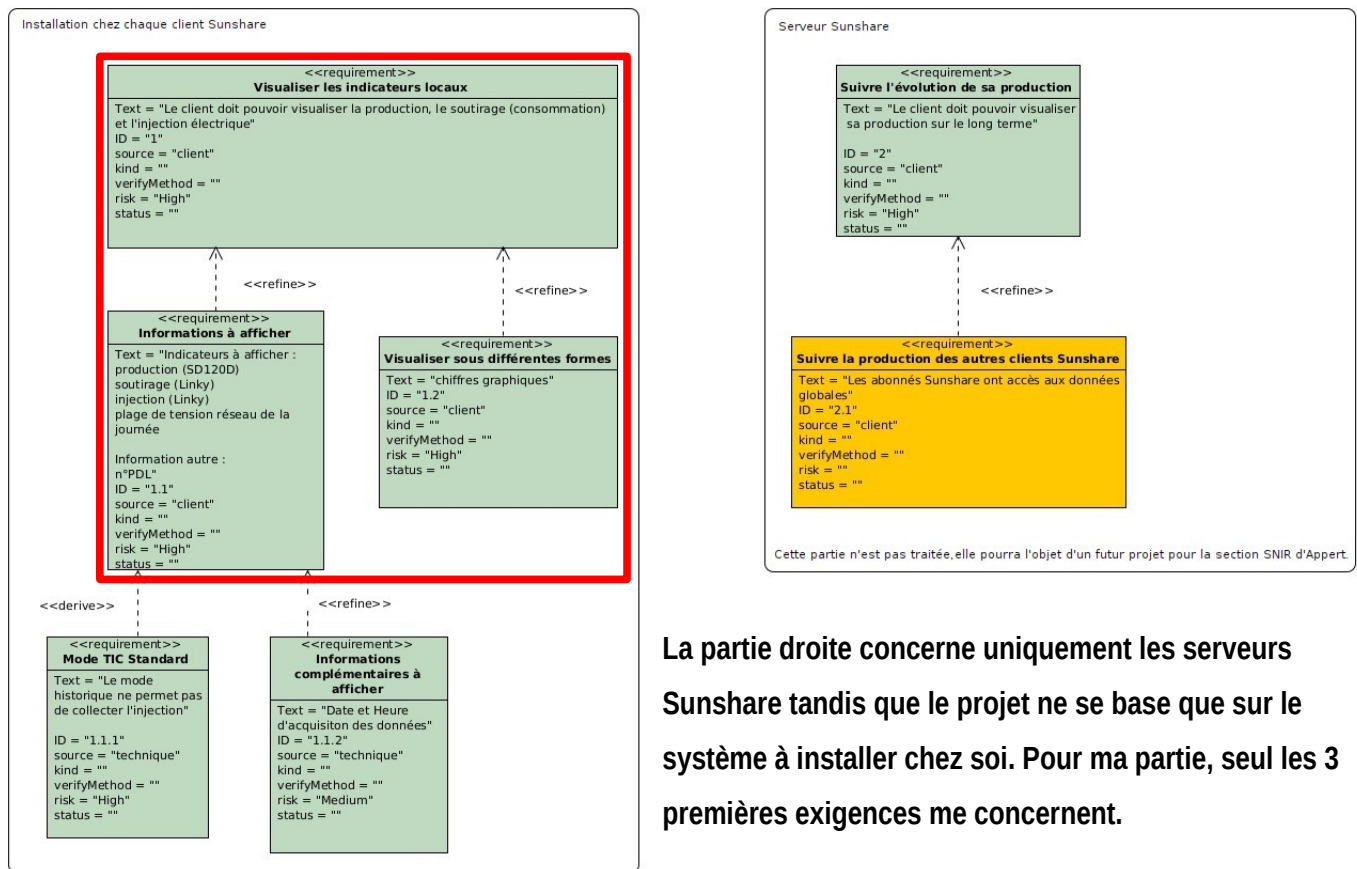
Le serveur distant appelé « Serveur SunShare » est celui qui centralise les données de tous les clients SunShare, quel que soit l'endroit où ils se trouvent dans le monde.

La base de données est hébergée sur le Raspberry et stocke les indices de consommation.

L'ordinateur accède à l'application web hébergée sur le Raspberry Pi par l'intermédiaire d'un navigateur (Chrome, Firefox...).

Le smartphone utilise une application mobile, qui pour communiquer avec la base de données, doit passer par une API convertissant les requêtes HTTP en fichier JSON.

### 1.2.6 - Diagramme d'exigences



La partie droite concerne uniquement les serveurs Sunshare tandis que le projet ne se base que sur le système à installer chez soi. Pour ma partie, seul les 3 premières exigences me concernent.

Figure 3: Diagramme d'exigences

### 1.2.7 - Aperçu de l'application

Première page : la page "Résumé de données" permet d'accéder rapidement et facilement aux dernières données enregistrées dans la base de données.

Elle permet de voir les données des 10 dernières journées, semaines, mois ou années.

Elle montre 3 graphiques différents pour chaque type d'énergie : production, consommation ou injection.

Chaque diagramme possède le même style pour garder un affichage simple et ergonomique.

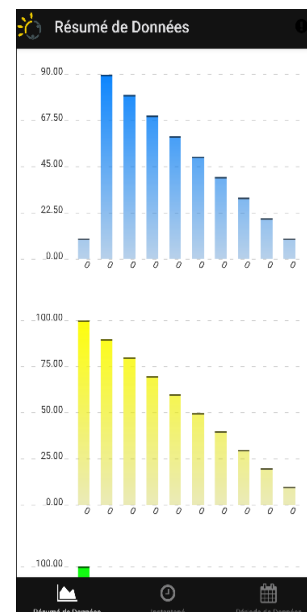


Figure 4: Page Résumé de données





Seconde page : la page "Temps réel" permet d'accéder instantanément aux données actuelles.

Elle montre 3 lignes différentes pour chaque type d'énergie : production, consommation ou injection.

L'énergie produite est exprimée en Watt (puissance active) et l'énergie consommée et injectée sont exprimées en Voltampère (puissance apparente).

En haut à droite il y a également un bouton permettant l'affichage d'astuces et conseils proposés par SunShare.



Figure 5: Page Temps réel

Troisième page : la page "Période de Données" permet d'accéder aux données stockées sur la base de données en sélectionnant une période de date.

Un bouton Calendrier se situe en haut de la page, celui-ci permet de sélectionner la période de données souhaitées. La page montre plusieurs lignes avec les 3 énergies par jour et le jour correspondant pour chacune. Le deuxième bouton permet de passer d'un mode d'affichage graphique à historique.

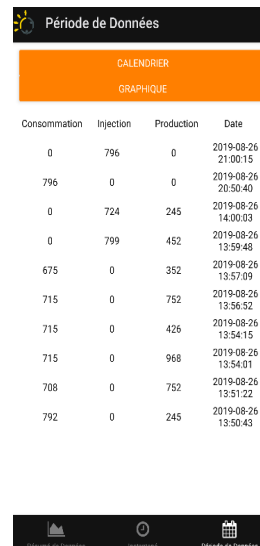


Figure 6: Page Période de données (côté historique)

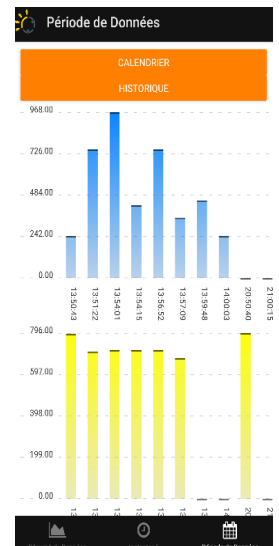


Figure 7: Page Période de données (côté graphique)

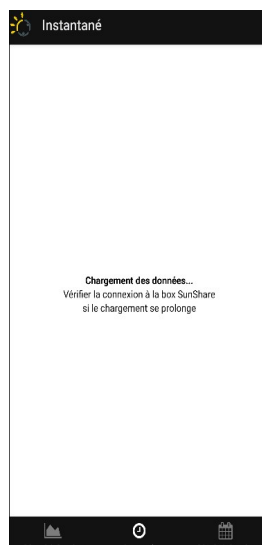


Figure 8: Page en cas d'erreur

Cas spécial : lorsque la connexion à la base de données échoue, un message d'erreur s'affiche.

Celui-ci indiquant l'erreur ainsi qu'un conseil demandant à l'utilisateur de vérifier sa connexion au réseau/à la box Sunshare.

### 1.2.8 - Diagramme de cas d'utilisation

Ici, je m'occupe de chaque cas, hormis l'affichage par interface web et client Sunshare.

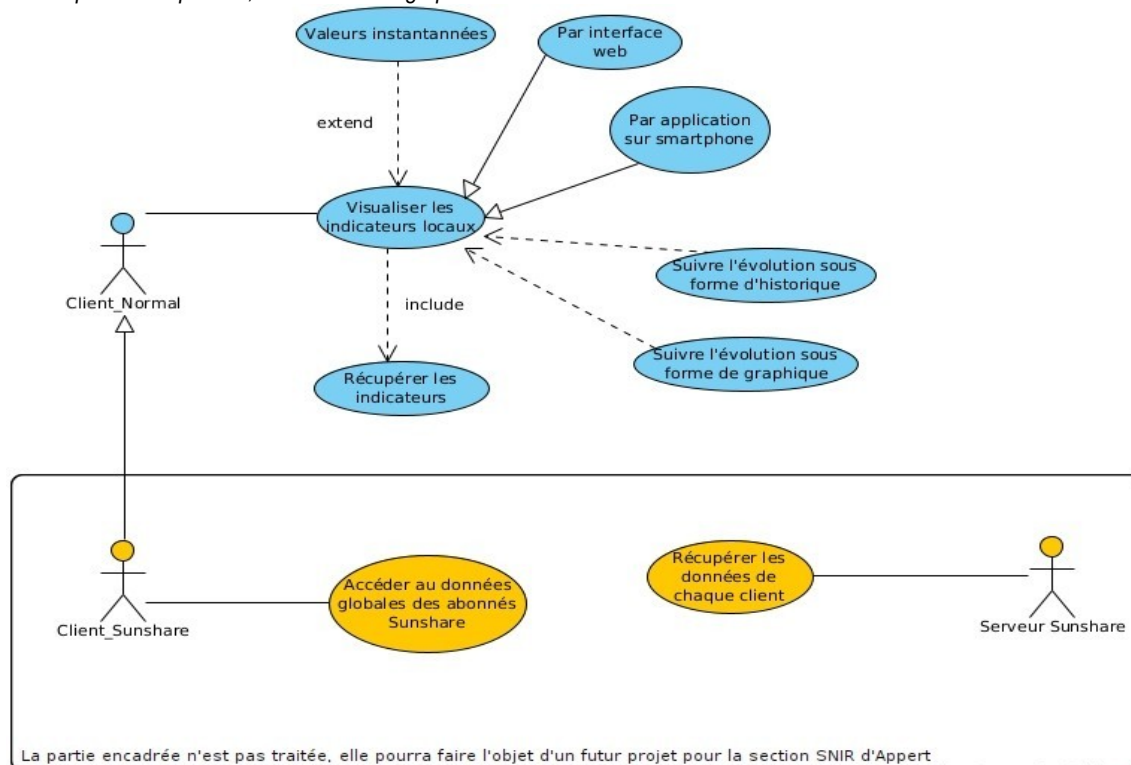


Figure 9: Diagramme de cas d'utilisation du système

### 1.2.9 - Schéma de la base de données

La base de données qui a été construite afin d'optimiser la gestion des données.

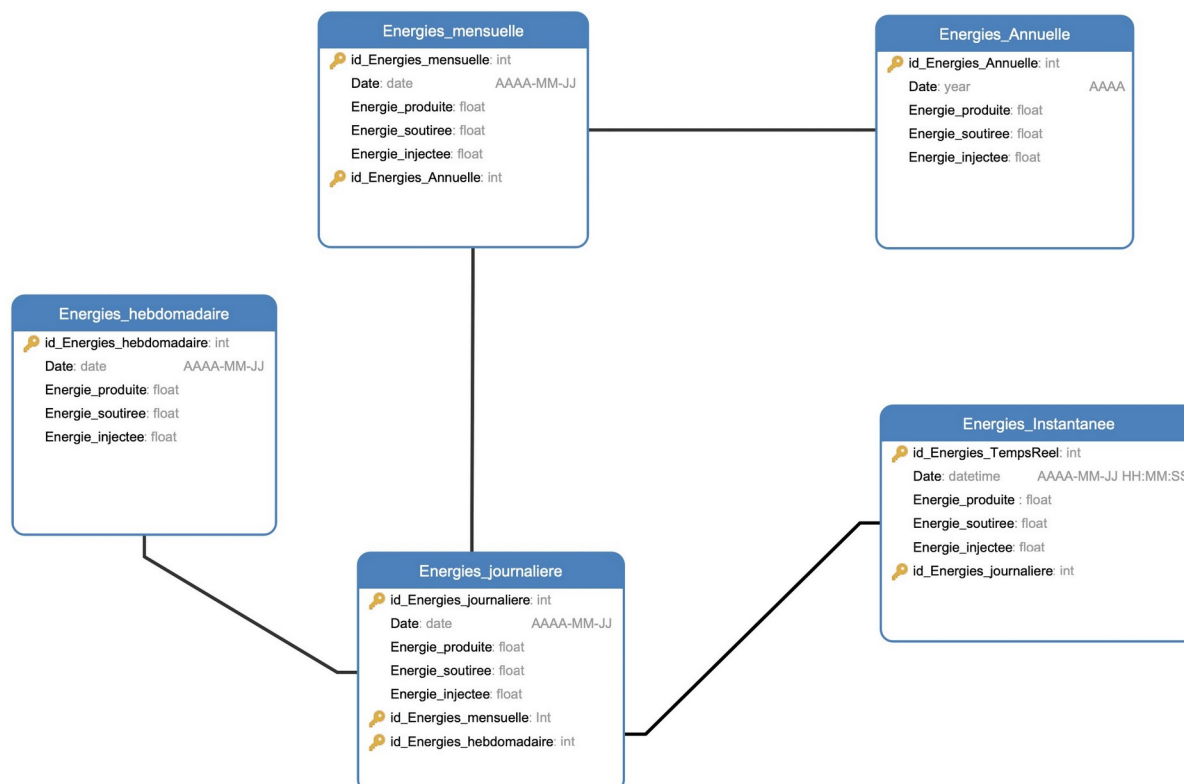


Figure 10: Schéma de la base de données

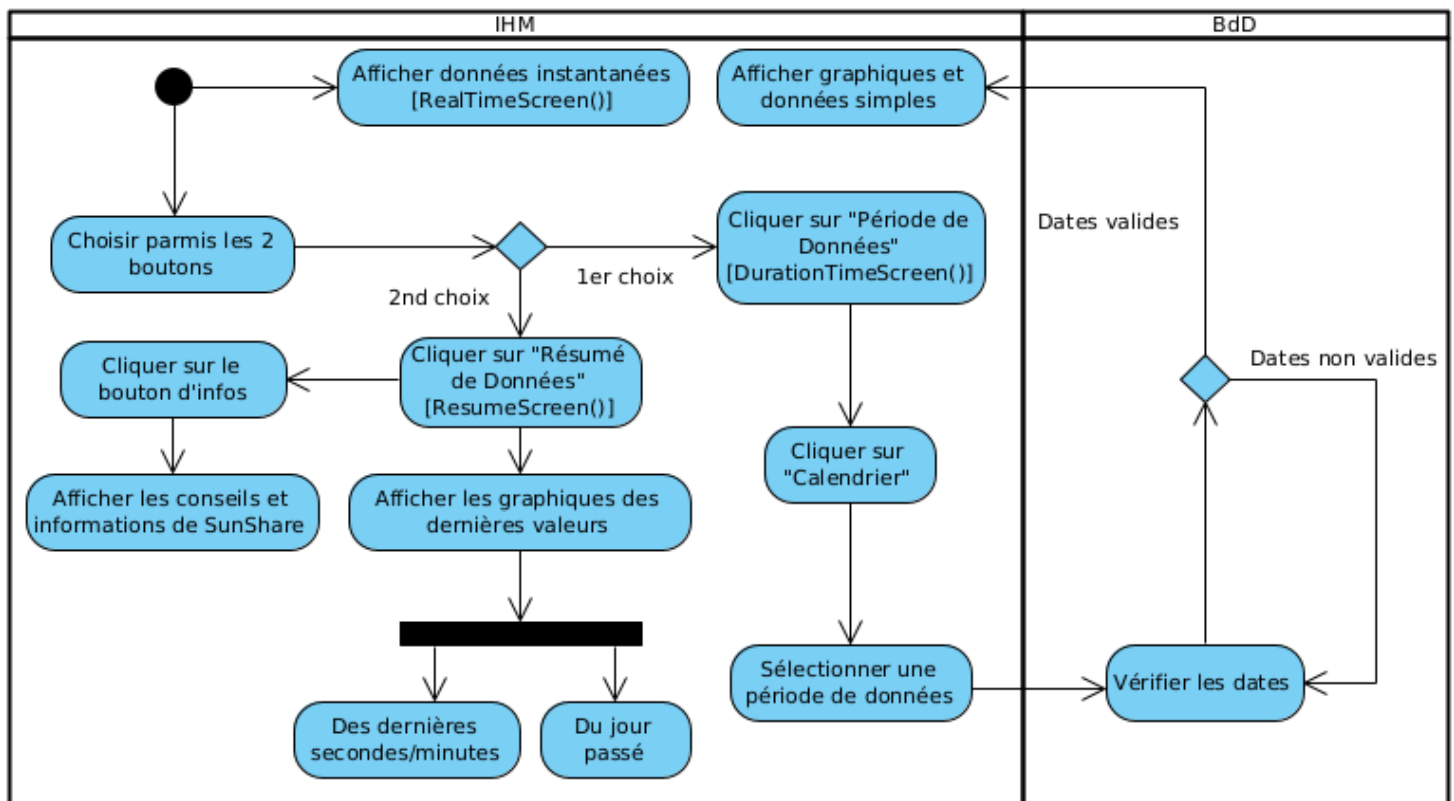
Cette base de données est implémentée dans la Raspberry pi. Avec celle-ci les données sont stockées le plus simplement possibles pour que le client puisse consulter le plus facilement et rapidement ses données.

Le système utilise une base de données composé de 5 tables :

- La table **Energies\_Instantanee** qui contient les énergies que les compteurs (Compteur Linky et Compteur SD120D) vont envoyer à la base de données, mesurées en temps réel ainsi que la date (Date) à laquelle les mesures ont été faites. Ces mesures comprennent donc l'énergie produite (energie\_produite), l'énergie consommée (energie\_soutiree), l'énergie injectée (energie\_injectee).
- La table **Energies\_journaliere** qui contient les énergies (Energie\_produite, Energie\_soutiree et Energie\_injectee) moyennes d'une journée. Ces valeurs seront calculées chaque jour et permettent de gagner du temps pour la consultation journalière.
- La table **Energies\_hebdomadaire** contient les énergies (Energie\_produite, Energie\_soutiree et Energie\_injectee) moyennes d'une semaine. Les données de chaque jour durant 7 jours, calculées en valeur moyenne.
- La table **Energies\_mensuelle** contient les énergies (Energie\_produite, Energie\_soutiree et Energie\_injectee) moyennes de chaque mois
- La table **Energies\_annuelle** contient les énergies (Energie\_produite, Energie\_soutiree et Energie\_injectee) moyennes d'une année

### 1.2.10 - Diagramme d'activité

Diagramme d'activité montrant la navigation entre les pages.



## 2 - Réalisation de la tâche professionnelle "Récupération des données"

Cette tâche consiste à récupérer les données via une requête effectuée à la base de données. Pour cette tâche, je n'ai pas utilisé d'API existante telle que Axios, mais nous avons créé (à l'aide de Ayaad Ahamada, qui gère la partie base de données) notre propre API pour récupérer les données au format JSON.

### 2.1 - Diagramme de séquence

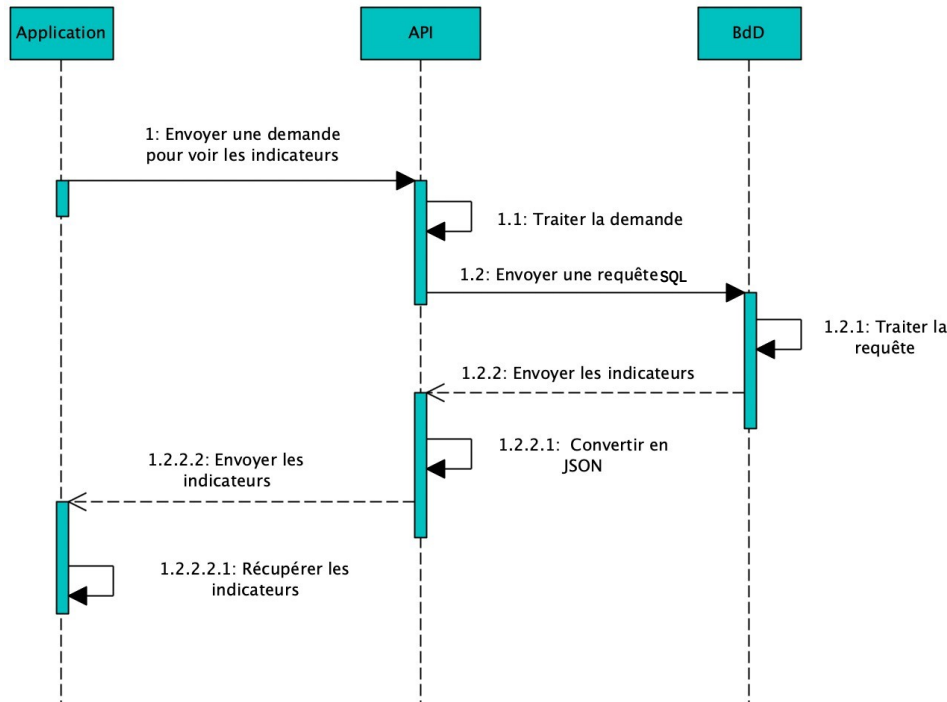


Figure 11: Diagramme de séquence du scénario nominal

Une API nous est indispensable afin de récupérer les données de la base de données et les traiter facilement dans l'application. Celle-ci a été construite uniquement via le langage PHP.

Dans un premier temps l'application va envoyer une requête HTTP vers l'API, puis le PHP accède à notre base de données, ce qui va ensuite permettre dans un deuxième temps de renvoyer les données souhaitées (temps réel/résumé/période) sous le format JSON (format lisible plus facilement pour une application smartphone). Enfin, on trie les données afin de les afficher normalement directement sur l'application au format souhaité (graphique ou tableau).

### 2.2 - Cas d'utilisation

Le cas d'utilisation correspondant à la récupération des données sur l'application.

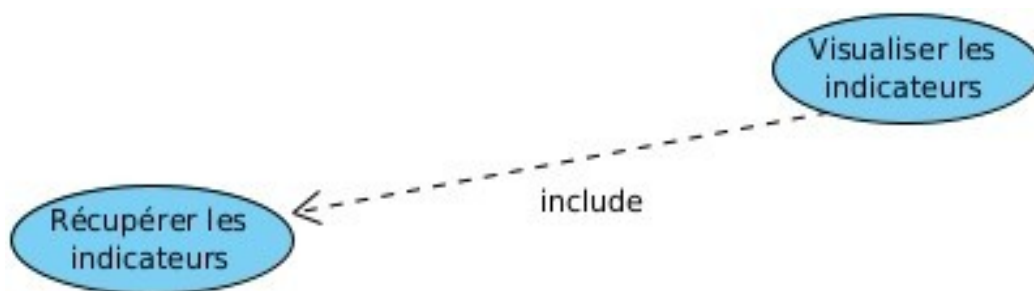


Figure 12: Cas d'utilisation : Récupération des données

La récupération des données n'est concernée que pour pouvoir les visualiser dans l'application, celle-ci se fait uniquement lorsque l'application est lancée et que l'utilisateur se trouve sur la page correspondante aux données souhaitées.

## 2.3 - Conception détaillée

L'application doit effectuer une requête HTTP à l'aide de la fonction `fetch()`, celle-ci doit permettre de récupérer uniquement les données souhaitées par l'utilisateur et aucunes de plus, pour éviter un chargement trop long ou une erreur d'acquisition de celles-ci.

L'utilisation de la fonction `fetch()` permet l'ajout de paramètres, et c'est ceux-ci qui permettront de choisir sur quelle période afficher les données. Sur l'exemple ci-dessous on peut voir que les deux dates sélectionnées par l'utilisateur sont ajoutées directement à la suite de l'URL contenant nos données en JSON afin de créer une URL sous cette forme :

" http://10.0.122.166/Ma\_page\_web\_/AppMobile/Historique.php?DebutDate=2022-02-22&FinDate=2022-03-08 "

```
var url = new URL("http://10.0.122.166/Ma_page_web_/AppMobile/Historique.php"),
    params = {DebutDate: selectedRange.firstDate,
              FinDate: selectedRange.secondDate+" 23:59:59"}
Object.keys(params).forEach(key => url.searchParams.append(key, params[key]))
```

Figure 13: Utilisation de la fonction `fetch()` afin d'afficher des données sur une période choisie

La requête récupère alors (dans cet exemple) toutes les données existantes entre le 22 Février 2022 à 00h00 et le 8 Mars 2022 à 23h59, sans exceptions. Si la période sélectionnée ne contient aucunes valeurs, alors il n'y aura simplement aucuns résultats. Enfin, ces données sont stockées par lignes dans un tableau nommé `tableUpdate`.

```
fetch(url).then(response => response.json())
  .then((responseJson) => {
    console.log(tableData[0]);
    for(let i=0; i<=9; i++)
    {
      global.tableUpdate[i][0] = responseJson[i].energie_produite;
      global.tableUpdate[i][1] = responseJson[i].energie_soutiree;
      global.tableUpdate[i][2] = responseJson[i].energie_injectee;
      global.tableUpdate[i][3] = responseJson[i].date;
    }
    console.log(tableData[0]);
    //tableData = global.tableUpdate;
  })
  .catch((error) => {
    console.error(error);
  });
```

Figure 14: Insertion des données reçues dans le tableau `tableUpdate`

Le cahier des charges proposait d'utiliser Axios comme API afin de récupérer les données, nous avons finalement décidé de créer notre propre API Rest. Côté serveur c'est un script PHP qui permet la récupération des données et les renvoie au format JSON afin que l'application smartphone puisse les traiter.

## 2.4 - Test unitaire

### 2.4.1 - Procédure de test

I.D.	Méthode testée Description de celle-ci	Procédure de test
		Résultats attendus
U1.0	<b>Se rendre sur l'application</b>  <i>A l'aide de l'émulateur intégré d'Android Studio.</i>	<p>Ouvrir un terminal puis :</p> <pre>&gt; cd Documents/android-studio/bin &gt; ./studio.sh</pre> <p>Pour lancer Android Studio, et lancer l'émulateur à l'aide de la flèche verte « Run 'app' »</p> <p>Ouvrir un second terminal <b>[Figure 16]</b> :</p> <pre>&gt; cd Documents/Sunshare_Supervisor &gt; sudo expo start ou &gt; sudo npm start</pre> <p>Puis entrer <b>r</b> et <b>a</b> pour reload l'application et la relancée</p> <p>L'application s'ouvre sur la page instantanée.</p>
U1.1	<b>Requête afin de récupérer les données</b>  <i>Avec la fonction fetch().</i>	<p><b>*Se réalise automatiquement*</b></p> <p>Avant l'affichage on fait une requête à l'aide de fetch(), pour les données en temps réel on aura :</p> <pre>fetch('http://10.0.122.166/Ma_page_web/AppMobile/Energie.php') <b>[Figure 13/14]</b></pre> <p>Les données sont récupérées au format JSON <b>[Figure 15]</b> et stockées dans une variable nommée response.</p>
U1.3	<b>Affichage des données sur l'application</b>  <i>En affichant le tableau invisible précédemment rempli par les valeurs à l'aide du module 'react-native-table-component-2' et de la fonction map().</i>	<p><b>*Se réalise automatiquement*</b></p> <p>Pour afficher le tableau on utilise le module "react-native-table-component-2" qui nous permet de créer aisément des tableaux, ici on affiche 2 colonnes pour 3 lignes, soit une ligne par valeurs et leur intitulé :</p> <pre>&lt;Table&gt;   {tableData.map((rowData, index) =&gt; (     &lt;Row style={styles.head}       key={index}       data={rowData}       widthArr={this.state.widthArr}       textStyle={styles.text} /&gt;   ))} &lt;/Table&gt;</pre> <p>Les valeurs sont affichées en face de leurs intitulés.</p>

```
[{"0":"32","id":"32","1":null,"id_utilisateur":null,"2":"2022-03-25 09:29:02","date":"2022-03-25 09:29:02","3":"30.2","energie_produite":"30.2","4":"1.2","energie_soutiree":"1.2","5":"21.2","energie_injectee":"21.2"}, {"0":"31","id":"31","1":null,"id_utilisateur":null,"2":"2022-03-25 09:28:35","date":"2022-03-25 09:28:35","3":"30.2","energie_produite":"30.2","4":"20.2","energie_soutiree":"20.2","5":"21.2","energie_injectee":"21.2"}, {"0":"28","id":"28","1":null,"id_utilisateur":null,"2":"2022-03-22 12:11:16","date":"2022-03-22 12:11:16","3":"20","energie_produite":"20","4":"121","energie_soutiree":"121","5":"220","energie_injectee":"220"}, {"0":"27","id":"27","1":null,"id_utilisateur":null,"2":"2022-03-22 12:11:00","date":"2022-03-22 12:11:00","3":"20","energie_produite":"20","4":"121","energie_soutiree":"121","5":"220","energie_injectee":"220"}, {"0":"26","id":"26","1":null,"id_utilisateur":null,"2":"2022-03-22 11:56:40","date":"2022-03-22 11:56:40","3":"50","energie_produite":"50","4":"100","energie_soutiree":"100","5":"20","energie_injectee":"20"}]
```

Figure 15: Données au format JSON (exemple)

Ici, les données reçues suite à la requête au format JSON, dans lesquelles ont récupère les valeurs et les dates.

```
local@1116-PC11-SNIR:~/Documents/Sunshare_Supervisor$ sudo expo start
[sudo] Mot de passe de local :

There is a new version of expo-cli available (5.4.6).
You are currently using expo-cli 5.0.3
Install expo-cli globally using the package manager of your choice;
for example: `npm install -g expo-cli` to get the latest version

Starting project at /local/Documents/Sunshare_Supervisor
Developer tools running on http://localhost:19002
Some dependencies are incompatible with the installed expo package version:
- react-native-svg - expected version: 12.1.1 - actual version installed: 12.3.0
Your project may not work correctly until you install the correct versions of the packages.
To install the correct versions of these packages, please run: expo install [package-name ...]
Starting Metro Bundler

> Metro waiting on exp://172.31.6.111:19000
> Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

> Press a | open Android
> Press w | open web

> Press r | reload app
> Press m | toggle menu
> Press d | show developer tools
> shift+d | toggle auto opening developer tools on startup (disabled)

> Press ? | show all commands

Logs for your project will appear below. Press Ctrl+C to exit.
> Opening on Android...
> Opening exp://172.31.6.111:19000 on
```

Figure 16: Lancement de l'application

Le lancement de l'application (l'émulateur d'Android Studio doit être ouvert). Le QR Code permet de lancer l'application depuis un smartphone physique (celui-ci doit posséder l'application expo).



## 2.4.2 - Rapport d'exécution

I.D.	OK	!OK	Observations
U1.0	X		L'application est ouverte sur l'émulateur d'Android Studio
U1.1	X		Les valeurs s'affichent sur l'application, de manière ordonnée, en face de leurs libellés respectifs. (La base de données doit être opérationnelle)
U1.2	X		
U1.3	X		

## 3 - Réalisation de la tâche "Afficher les données sous forme graphique"

Cette tâche consiste à afficher les données sous forme de graphiques simples et lisibles. Pour cette tâche, j'ai utilisé un module ("react-native-chart-kit") à ajouter directement dans mon projet, celui-ci me permettant d'afficher tout type de graphiques.

### 3.1 - Diagramme de séquence

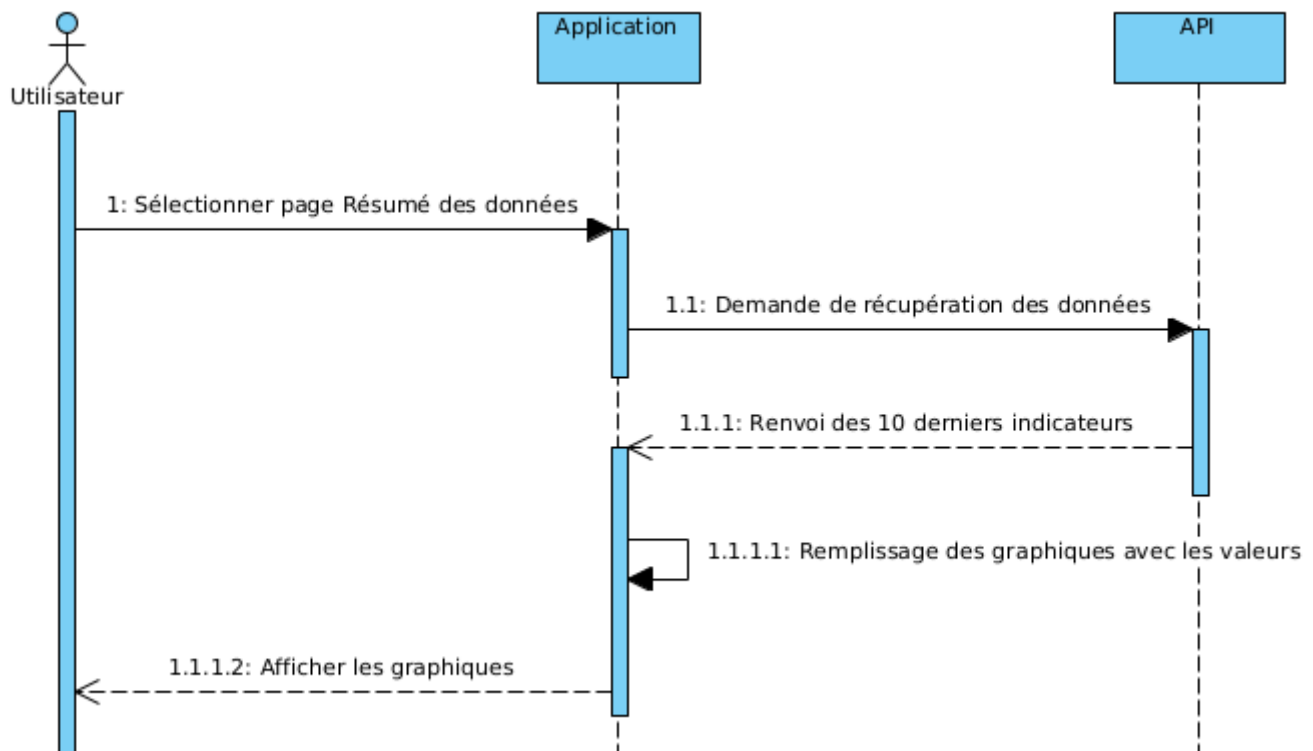


Figure 17: Diagramme de séquence d'affichage des données sous forme graphique



### 3.2 - Cas d'utilisation

Le cas d'utilisation correspondant à l'affichage graphique des données.

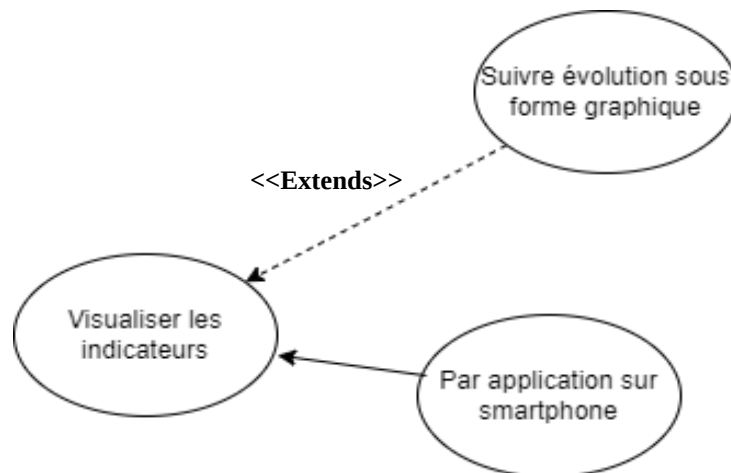


Figure 18: Visualiser les indicateurs sous forme graphique sur smartphone

### 3.3 - Conception détaillée

L'application doit afficher les données de la base de données sous forme de graphique afin de les rendre plus lisibles pour l'utilisateur. Pour cela on va utiliser un module à installer directement dans notre projet (avec la commande "sudo npm install react-native-chart-kit"), celui-ci nous permettant d'afficher des données dans 6 types de graphiques différents (ici on utilisera les graphiques en barre nommés "BarChart").

La première étape consiste à personnaliser le graphique, chaque graphique peut-être modifier à l'aide de multiples options.

```
const chartConfigBar1 = {
```

Figure 19: Personnalisation du graphique

Ici, on modifie la couleur de fond du graphique,

```
backgroundColor: "#FFFFFF",  
backgroundGradientFrom: "#FFFFFF",  
backgroundGradientTo: "#FFFFFF",
```

Figure 20: Couleur de fond avec backgroundColor

La couleur de celui-ci,

```
color: (opacity = 1) => `rgba(0, 0, 0, ${opacity})`,
```

Figure 21: Couleur principale du graphique

La couleur des textes,

```
labelColor: (opacity = 1) => `rgba(0, 0, 0, ${opacity})`,
```

Figure 22: Couleur des textes sur le graphique (labels)

La largeur des barres,

```
barPercentage: 0.5,
```

Figure 23: Largeur des barres avec barPercentage

Et la couleur de celles-ci.

```
fillShadowGradient: '#0080FF',
fillShadowGradientOpacity: 1,
};
```

Figure 24: Couleur des barres en dégradé avec fillShadowGradient

Il y a bien d'autres options dont je n'avais pas l'utilité pour ces graphiques, mais il y a suffisamment de possibilités pour rendre cette extension très puissante, malgré la simplicité de son utilisation.

```
return fetch('http://10.0.122.166/Ma_page_web_/AppMobile/Energie.php')
.then((response) => response.json())
.then((responseJson) => {
  this.setState({
    loading: false,
    dataSource: responseJson,
  }, function() {
    for(let i=0; i<10; i++)
    {
      global.dataProduite.datasets[0].data[i] = responseJson[i].energie_produite;
      global.dataProduite.labels[i] = responseJson[i].date.slice(-9);

      global.dataConsommee.datasets[0].data[i] = responseJson[i].energie_soutiree;
      global.dataConsommee.labels[i] = responseJson[i].date.slice(-9);

      global.dataInjectee.datasets[0].data[i] = responseJson[i].energie_injectee;
      global.dataInjectee.labels[i] = responseJson[i].date.slice(-9);
    }
    global.dataProduite.labels = global.dataProduite.labels.reverse();
    global.dataProduite.datasets[0].data = global.dataProduite.datasets[0].data.reverse();

    global.dataConsommee.labels = global.dataConsommee.labels.reverse();
    global.dataConsommee.datasets[0].data = global.dataConsommee.datasets[0].data.reverse();

    global.dataInjectee.labels = global.dataInjectee.labels.reverse();
    global.dataInjectee.datasets[0].data = global.dataInjectee.datasets[0].data.reverse();

    global.dateJson = responseJson[0].date;
  })
})
.catch((error) => {
  console.error(error);
  global.showError = true;
});
```

Figure 25: Récupération des données et gestion de celles-ci

Ensuite, on rentre nos données dans un tableau vide et on modifie la date en séparant l'heure et le jour, puis pour afficher la date au format français (AAAA-MM-JJ devient donc JJ-MM-AAAA).

```
<FlatList
  style={ styles.MainContainer }
  data={ this.state.dataSource }
  renderItem={({item}) =>
    <View style={styles.View}>
      <BarChart
        data={dataProduite}
        width={screenWidth}
        height={250}
        chartConfig={chartConfigBar1}
        verticalLabelRotation={30}
        xLabelsOffset={-15}
        style={styles.graphBar}
        fromZero={true}
      />
    </View>
  }
/>
```

Figure 26: Affichage et personnalisation du graphique

Pour terminer, on affiche le graphique en lui indiquant quelles données utilisées, quelle largeur de l'écran occupée, l'emplacement des textes, et plusieurs autres options, ce qui permet encore plus de personnalisation en fonction de nos préférences.

### 3.4 - Tests unitaires

#### 3.4.1 - Procédure de tests

I.D.	Méthode testée Description de celle-ci	Procédure de test
		Résultats attendus
U2.0	Navigation vers la page Résumé de données.	Dès le lancement de l'application, appuyer sur le bouton " Résumé de données ". La page Résumé de données s'affiche.
U2.1	Les données sont récupérées à l'aide d'une requête fetch(). [Figure 25]	<b>*Se réalise automatiquement*</b> Dès l'affichage de la page, la requête fetch() est effectuée. Les données sont bien récupérées grâce à la requête et elles sont stockées dans la variable response.
U2.2	Gestion des données au format JSON et attribution  <i>On utilise la fonction response.json() afin d'obtenir notre résultat puis on attribuera dans des tableaux invisibles les valeurs récupérées.</i>	<b>*Se réalise automatiquement*</b> On crée une variable responseJson contenant nos quatre valeurs (production, injection, soutirage et date) puis on attribue ces valeurs un tableau invisible chacun nommés dataProduite, dataConsommee et dataInjectee. [Figure 25] Les valeurs sont stockées dans les différents tableaux à leurs places attribuées.
U2.3	Affichage des données.  <i>Sous forme de graphiques.</i>	<b>*Se réalise automatiquement*</b> Le module " react-native-chart-kit " récupère les données et les affiche [Figure 26] à l'aide du graphique et ses options choisies [Figure 18 à 24]. Les différents graphiques s'affichent sur la page avec les dix dernières valeurs. [Figure 27]

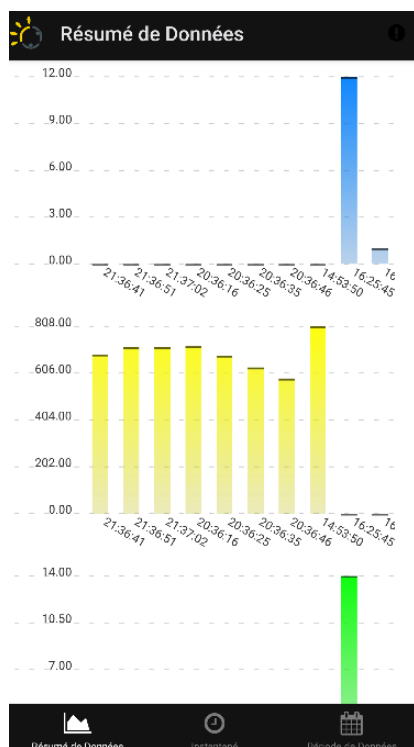


Figure 27: Page Résumé de données et graphiques

### 3.4.2 - Rapport d'exécution

I.D.	OK	Non OK	Observations
U2.0	X		La page Résumé de données s'affiche bien.
U2.1	X		Les graphiques s'affichent sur l'application, avec les bonnes valeurs. (La base de données doit être opérationnelle)
U2.2	X		
U2.3		X	Les graphiques s'affichent autant de fois que le nombre de valeurs par graphiques (ici 10).

U2.3 : Les graphiques s'affichent autant de fois que le nombre de valeurs à afficher

- Début de solution : méthode d'affichage différente = nombre de graphiques normal (trois, un par type de valeur) mais valeurs non affichées

## 4 - Réalisation de la tâche "Afficher les données en fonction d'une période choisie"

Cette tâche consiste à accéder à certaines données en fonction d'une période choisie par l'utilisateur, celle-ci permet d'obtenir des données précises sur une plage de temps.

### 4.1 - Diagramme de séquence

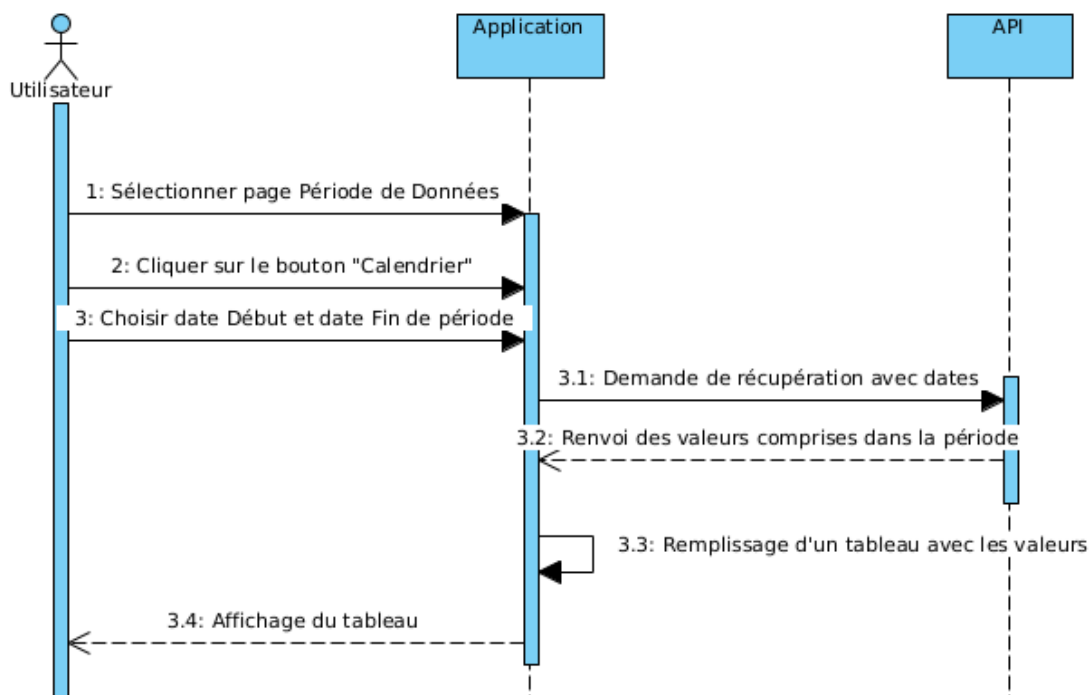


Figure 28: Diagramme de séquence de l'affichage en fonction d'une période

### 4.2 - Cas d'utilisation

Le cas d'utilisation correspondant à l'affichage historique des données.

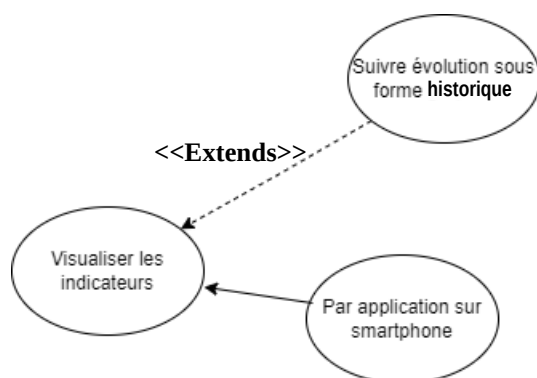


Figure 29: Cas d'utilisation de l'affichage sous forme historique

### 4.3 - Conception détaillée

L'application doit pouvoir donner à l'utilisateur toutes les données enregistrées durant une certaine période. Pour faire ceci, on effectue une requête HTTP avec comme paramètres nos deux dates (début et fin), la requête se fait directement vers la base de données, qui va renvoyer uniquement les données contenues entre les deux dates. Il ne restera plus qu'à récupérer ces données et les afficher dans un tableau.

```
var url = new URL("http://10.0.122.166/Ma_page_web/AppMobile/Historique.php"),
    params = {DebutDate: selectedRange.firstDate,
              FinDate: selectedRange.secondDate+" 23:59:59"}
Object.keys(params).forEach(key => url.searchParams.append(key, params[key]))
```

Figure 30: Initialisation avec dates en paramètres de l'URL

En premier lieu, on initialise la première ligne du tableau contenant les libellés Production, Consommation, Injection et Date. Ensuite on effectue la requête à l'aide de fetch() et on place les valeurs dans leur colonne respective.

```
fetch(url).then(response => response.json())
  .then((responseJson) => {
    console.log(tableData[0]);
    for(let i=0; i<=9; i++)
    {
      global.tableUpdate[i][0] = responseJson[i].energie_produite;
      global.tableUpdate[i][1] = responseJson[i].energie_soutiree;
      global.tableUpdate[i][2] = responseJson[i].energie_injectee;
      global.tableUpdate[i][3] = responseJson[i].date;
    }
    console.log(tableData[0]);
    //tableData = global.tableUpdate;
  })
  .catch((error) => {
    console.error(error);
  });
```

Figure 31: Utilisation de fetch() avec l'URL

Enfin, il suffit d'afficher le tableau rempli avec les valeurs (ici nommé tableUpdate), pour cela on utilise le module "react-native-table-component-2".

```
<Table>
  {global.tableUpdate.reverse().map((rowData, index) => (
    <Row
      style={styles.head}
      key={index}
      data={rowData}
      textStyle={styles.text}
    />
  ))}
</Table>
```

Figure 32: Affichage du tableau

## 4.4 - Tests unitaires

### 4.4.1 - Procédure de tests

I.D.	Méthode testée Description de celle-ci	Procédure de test
		Résultats attendus
U3.0	Navigation vers la page Période de données. [Figure 33]	Dès le lancement de l'application, appuyer sur le bouton " Période de données ". La page Période de données s'affiche.
U3.1	Sélection des dates de début et de fin	Appuyer sur le bouton Calendrier, puis sélectionner les deux dates constituant le début et la fin de la période (peu importe l'ordre de sélection). Enfin, cliquer sur accepter. Les dates sont alors utilisées comme paramètre de la requête envoyer à la base de données.
U3.3	Affichage des données. Sous forme d'historique.	<b>*Se réalise automatiquement*</b> Le module " react-native-table-component " récupère les données et les affiche à l'aide de tableUpdate. [Figure 32] L'historique s'affiche avec toutes les données et leurs dates se trouvant entre celles précédemment choisies.

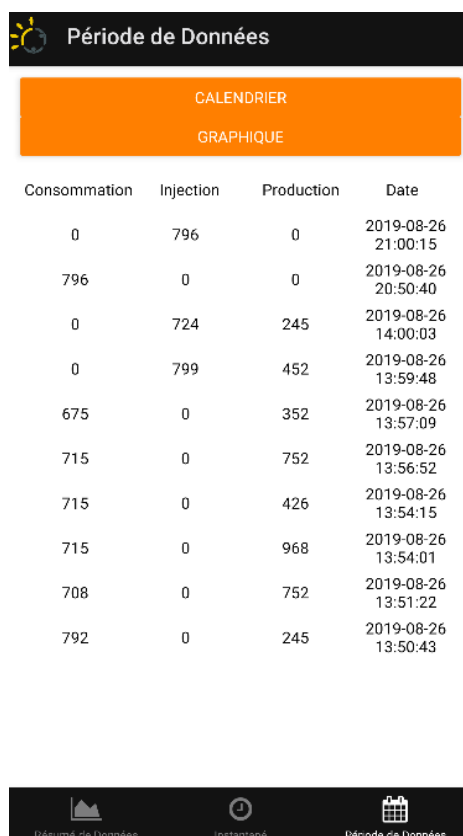


Figure 33: Page Période de données

#### 4.4.2 - Rapport d'exécution

I.D.	OK	Non OK	Observations
U3.0	X		La page Période de données s'affiche bien.
U3.1	X		Le menu de sélection des dates s'affichent bien lors de l'appui sur le bouton Calendrier.
U3.2	X		Les données s'affichent bien sous forme historique.
U3.3	X		

## 5 - Bilan de la réalisation personnelle

### 5.1 - Rappel des tâches personnelles à réalisées

Les tâches principales à réalisées durant le projet pour ma partie personnelle.

Tâches	Description	Validation
FP1	Créer une application simple pour Android et iOS	Effectué
FS1	Afficher 3 onglets différents pour l'affichage des données <ul style="list-style-type: none"> <li>Résumé de données</li> <li>Temps réel</li> <li>Période de données</li> </ul>	Effectué
FS2	Afficher des données sous forme graphique	Effectué
FS2.1	Afficher chaque tableau sans problèmes de re-rendering	Effectué
FS2.2	Afficher chaque tableau sans problèmes de récupération des données	En cours
FS3	Afficher des données sous forme de tableau	Effectué
FS4	Permettre à l'utilisateur de choisir l'affichage des données	En cours
FP2	Recevoir les données d'une base de données	Effectué
FS1	Créer un résultat de données en JSON	Effectué
FS2	Récupérer à l'aide d'une requête les données en JSON	Effectué
FS3	Traiter et afficher les données sur l'application	Effectué

Réalisation : environ 85%

### 5.2 - Améliorations possibles

- Mettre en place un système de refresh de la page temps réel
  - L'application prend les dernières valeurs enregistrées dans la base de données au moment de son lancement, pour que les valeurs se mettent à jour, l'application doit être relancée
- Permettre à l'utilisateur de mettre à jour la page temps réel à l'aide d'un "swipe" (autre idée pour régler le problème énuméré au-dessus)
- Réduire la taille de l'application (actuellement autour de 60 Mo)
- Rendre l'API plus performante (effectue actuellement une requête sur l'entier des valeurs disponibles dans la table Energies\_TempsReel)



## **5.3 - Conclusion personnelle**

Ce projet m'a permis de comprendre et d'apprendre le développement d'une application smartphone à l'aide d'un framework puissant. J'ai également pu apprendre un nouveau langage de programmation (TypeScript), celui-ci est puissant mais compliqué à prendre en main au début. Grâce à ce projet j'ai aussi appris à mieux travailler en groupe, tout en gardant l'entiereté du système fonctionnelle tout au long de son évolution.

### **5.3.1 - Points positifs**

- Découverte du framework React Native
  - Framework puissant, simple à prendre en main et très utile avec l'utilisation d'Expo
- Découverte du langage TypeScript
  - Langage puissant, plus complexe à prendre en main avec une syntaxe mélangeant JavaScript et balises
- Création d'une API
  - Fonctionnant sous PHP afin de récupérer les données stockées sur la base de données
- Gestion et organisation du groupe (Trello)

### **5.3.2 - Points négatifs**

- Beaucoup de modules à ajouter à l'application pour avoir un résultat satisfaisant
- Poids de l'application conséquent comparer à son utilisation
- Certains problèmes sans solutions rencontrés
  - Affichage des valeurs sur la page résumé de données