

**École polytechnique de Louvain**

# **HaïtiWater 2.0**

Evolution de l'application HaïtiWater vers une application entière fonctionnelle hors-ligne

Auteur: **Vincent GRADZIELEWSKI**

Promoteurs: **Kim MENS, Sandra SOARES-FRAZÃO**

Lecteurs: ,

Année académique 2020-2021

Master [120] en sciences informatiques

# Table des matières

<b>Résumé</b>	<b>2</b>	<b>5 Implémentation</b>	<b>32</b>
<b>Remerciements</b>	<b>3</b>	5.1 Application de base . . . . .	32
<b>1 Introduction</b>	<b>4</b>	5.2 Choix technologiques . . . . .	33
<b>2 Contexte</b>	<b>6</b>	5.3 Client . . . . .	35
2.1 La gestion de l'eau en Haïti	6	5.3.1 Stratégie de syn- chronisation des pages . . . . .	35
2.2 Introduction à l'application	7	5.3.2 Stratégie de syn- chronisation de la DB . . . . .	37
2.2.1 Eléments de l'ap- plication . . . . .	7	5.3.3 Gestion des chan- gements d'utilisa- teurs . . . . .	39
2.2.2 Utilisateurs de l'application . . . . .	8	5.3.4 Sytème de notifi- cation . . . . .	39
2.2.3 Accueil . . . . .	9	5.4 Interface utilisateur . . . . .	40
2.2.4 Réseau . . . . .	9	5.4.1 Changement de mode . . . . .	40
2.2.5 Carte . . . . .	11	5.4.2 Module à synchro- niser . . . . .	41
2.2.6 Gestion de zone . .	12	5.5 Serveur . . . . .	42
2.2.7 Historique . . . . .	13	<b>6 Validation</b>	<b>45</b>
2.2.8 Rapports . . . . .	14	6.1 Vérifications automatiques	45
2.2.9 Consommateurs . .	15	6.2 Vérifications utilisateurs réels . . . . .	45
2.2.10 Finances . . . . .	17	<b>7 Améliorations futures</b>	<b>46</b>
2.3 Problèmes réseaux . . . . .	18	7.1 Suite du projet . . . . .	46
<b>3 Organisation</b>	<b>20</b>	7.2 Défis rencontrés . . . . .	46
3.1 Approche de travail . . . . .	20	7.3 Propositions . . . . .	46
3.2 Méthodologie . . . . .	21	<b>8 Conclusion</b>	<b>47</b>
<b>4 Analyse des besoins</b>	<b>24</b>	8.1 Métriques . . . . .	47
4.1 Besoins fonctionnels . . . . .	25		
4.2 Besoins non-fonctionnels .	29		
4.3 Structure des données hors-ligne . . . . .	30		
4.4 Conclusion . . . . .	31		

# Résumé

Ce travail de fin d'études a été réalisé dans le cadre de mon Master en Sciences Informatiques à l'École Polytechnique de Louvain-la-neuve durant l'année académique 2020-2021.

Dans ce mémoire je vais présenter mon travail qui consistait à reprendre l'application HaïtiWater développée précédemment par Adrien Hallet, Céline Deknop et Sebastien Strebelle qui a pour but "La gestion du réseau de distribution d'eau potable en Haïti". Le but étant de faire évoluer cette application pour que celle-ci soit entièrement

afin de la faire évoluer vers une application web qui serait entièrement utilisable **hors-ligne**.

Cette application a pour but "La gestion du réseau de distribution d'eau potable en Haïti". Je commencerai par une brève introduction sur le contexte Haïtien et sur les raisons pour lesquels l'évolution de cette application était nécessaire. Ensuite je présenterai le principe des progressive web app et pourquoi j'ai choisis d'utiliser cette technologie plutôt qu'une autre. Je présenterai ensuite la validation de l'application et les feedback que j'ai reçu des utilisateurs ainsi que les conséquences de cette validation sur l'application finale. Puis je conclurai par une liste des améliorations possibles.

Tout le travail réalisé est disponible ici :

- Github : <https://github.com/exavince/HaitiWater>
- Par l'UCL : <https://haitiwater.sipr.ucl.ac.be>
- En Haïti : unknown

Si vous désirez tester l'application, il suffit de vous rendre sur un des liens cité précédemment et de vous connecter à l'aide de l'utilisateur qui vous sera communiqué si vous en faite la demande. Ces données ne seront pas révélées ici car les données ne peuvent pas être modifiées aléatoirement.

# Remerciements

# Chapitre 1

## Introduction

### Contexte

Ce mémoire appartient à un projet de développement financé par ARES-CCD avec quelques partenaires tels que Protos<sup>1</sup>, l’UCL et l’UEH.

Protos est une ONG qui vise à améliorer l’accès à l’eau potable dans plusieurs pays du monde afin de les aider à se développer.

Suite à de nombreuses crises politiques et catastrophes naturelles qui ont détruit beaucoup d’infrastructures locales, l’accès à l’eau potables est devenu difficile en Haïti. De plus, des incertitudes politiques entravent la reconstruction de ces installations et les populations ne sont pas toujours aidées par les services publics pour assurer la distribution de l’eau.

Il y a quelques années Protos est entré en contact avec l’UCL afin de réaliser un système logiciel pilote pour la gestion de la distribution d’eau potable en zone rurale. C’est pour cette raison que l’ONG Protos est active dans le pays depuis quelques années et à permis aux anciens mémorants de créer l’application HaïtiWater.

En effet, aucune gestion centralisée organisée par l’état n’existe pour ces zones, éloignées des grandes agglomérations. Des réseaux existent, constitués de points de prélèvement d’eau, de conduites de distribution d’eau et de fontaines situées dans les villages, mais la gestion publique de ceux-ci n’est pas opérationnelle.

L’application créée précédemment propose un appui à ces organismes locaux afin de mieux organiser cette distribution.

---

1. [www.protos.ngo](http://www.protos.ngo)

## **Problématiques**

Actuellement l'application HaïtiWater est prévue pour fonctionner uniquement lorsque le réseau est stable et fonctionnel. Malheureusement en Haïti le réseau est assez instable et par endroit ce réseau n'est même pas disponible. La suite du développement de l'application va donc devoir porter sur le fonctionnement hors-ligne de celle-ci.

## **Motivation**

## **Objectifs**

## **Approche**

## **Contribution**

## **Plan**

# Chapitre 2

## Contexte

### 2.1 La gestion de l'eau en Haïti

Haïti est l'un des pays les plus pauvres au monde. Le pays est situé dans une zone géographique où les risques de catastrophes naturelles sont très élevés. Ces intempéries détruisent les infrastructures et empêchent le bon développement des réseaux de distribution d'eau. Il y a quelques années, en 2010, le pays a subi un énorme séisme qui a ravagé une bonne partie du territoire. Parmi tous les défis à relever vient celui de la gestion et de la distribution de l'eau potable sur le territoire, surtout dans les zones les plus rurales. Le climat de la région rendant excessivement difficile l'exploitation directe des cours d'eau, il faut beaucoup d'infrastructures afin de pouvoir assumer la distribution de l'eau potable à tous.

En raison de la grande pauvreté et de gros problèmes organisationnels qui règnent sur la plupart de l'île, il est très difficile de maintenir et de développer le réseau de distribution d'eau potable. Il y a un gros manque de collaboration entre les entités haïtiennes ou entre les villages dû en partie à la communication qui n'est pas du tout optimisée voir inexistante dans certains cas. Dans les zones les plus rurales de l'île, le taux de recouvrement des factures est excessivement faible, jusqu'à 11%.

Pour toutes ces raisons, l'ONG Protos vient donc en aide à Haïti afin d'aider le service national des eaux à gérer la gestion des infrastructures et la facturation des clients surtout dans les zones rurales.

Si vous désirez plus d'informations sur les problèmes environnementaux, politiques, sociaux ou organisationnels du contexte haïtiens, je vous invite à aller consulter le mémoire d'Adrien Hallet, Céline Deknop et Sébastien Strebelle [2].

## 2.2 Introduction à l'application

Dans le cadre de la situation décrite dans la section 2.1 (La gestion de l'eau en Haïti), Protos a fait appel à l'UCL afin de créer une application d'aide à la gestion et la distribution de l'eau potable en Haïti. Une première version de l'application a été développée en 2018-2019 par 3 personnes dans le cadre de leur mémoire [2], elle comprend différents modules permettant de faciliter la gestion des infrastructures et les clients qui vont y chercher de l'eau.

L'application est basée sur un principe hiérarchique. Pour illustrer celle-ci je vais reprendre l'image présentée ici [2].

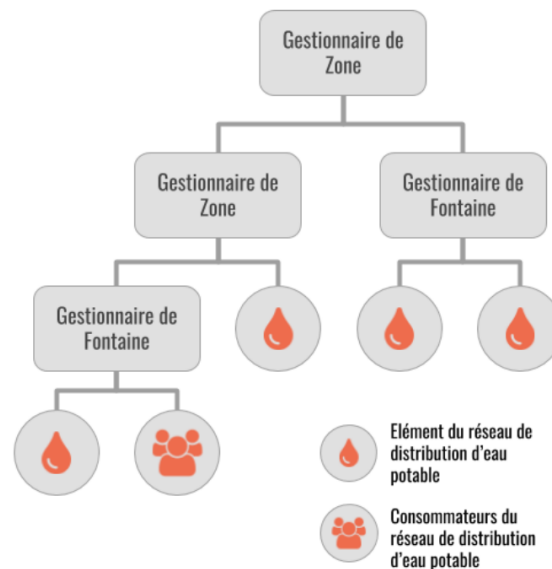


FIGURE 2.1 – Module consommateurs

### 2.2.1 Éléments de l'application

**Zone** Une zone représente une partie du territoire. Une zone peut être subdivisée en plusieurs sous-zone, la zone-parent reprendra alors toutes les données des zones enfants. Cela permet de séparer les responsabilités des différents territoires.



**Elements du réseau** Ces éléments représentent les points physiques du réseau de distribution d'eau potable. Ces éléments sont au nombre de 6 : conduites, réservoirs, sources, fontaines, kiosques et prises individuelles. Tous ensemble ils forment le réseau de distribution au complet.

**Consommateurs** Le consommateur est une personne qui va utiliser le réseau de distribution d'eau potable. Chaque consommateur se voit lors de son enregistrement attribué à un seul élément de sortie d'eau du réseau : fontaine, kiosque, prise individuelle. De cette façon, il est plus simple de gérer la facturation des consommateurs.

## 2.2.2 Utilisateurs de l'application

Il y a 2 types d'utilisateurs qui peuvent se connecter à l'application

**Gestionnaire de fontaine** Ce gestionnaire va gérer la distribution de l'eau aux différents consommateurs. Il devra gérer tous les éléments du réseau qui lui sont attribués.

**Gestionnaire de zone** Ce gestionnaire va avoir la responsabilité de gérer d'autres gestionnaires de zones et/ou des gestionnaires de fontaines. Son travail est plus d'administrer et de surveiller les personnes qui sont en-dessous de lui.

Ces deux gestionnaires vont avoir des privilèges différents et peuvent donc interagir différemment sur les données. Pour illustrer plus simplement cela, voici un tableau tel que présenté ici [2] où vous pourrez retrouver plus d'informations sur les rôles des différents gestionnaires.

Permission	Gestionnaire de	
	fontaine	zone
Ajouter/modifier/supprimer un consommateur	✓	✓
Ajouter/modifier/supprimer un élément du réseau de distribution	✓	✓
Ajouter/modifier/supprimer un rapport mensuel	✓	✓
Ajouter/modifier/supprimer un paiement	✓	✓
Ajouter/modifier/supprimer un ticket de support	✓	✓
Ajouter/modifier/supprimer une zone	✗	✓
Ajouter/modifier/supprimer un gestionnaire	✗	✓*
Accepter/refuser un changement dans l'historique	✗	✓

\* : Un gestionnaire de zone ne peut pas modifier les informations personnelles (mot de passe, courrier, nom, prénom) d'un gestionnaire existant

TABLE 2.1 – Permissions dans l'application, par type d'utilisateur

### 2.2.3 Accueil

Ce module contient les informations condensées de la zone qui est attribuée à l'utilisateur. Il peut y retrouver le nombre de fontaines, de kiosques, de points de prises individuelles et de conduites que contient sa zone mais aussi le nombre de foyers et de consommateurs individuels liés à celle-ci.

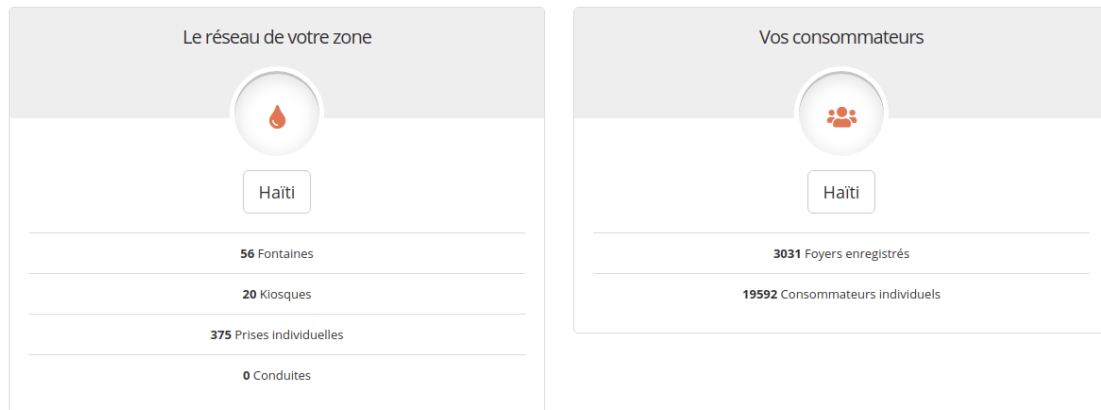


FIGURE 2.2 – Module d'accueil

### 2.2.4 Réseau

Dans ce module, l'utilisateur peut retrouver 3 éléments différents.

**Schéma** Un schéma contenant la répartition des consommateurs par genre ou le volume d'eau mensuel distribué dans chaque zone. Il peut sélectionner le schéma à afficher grâce à une liste déroulante.

**Résumé de zone** Un résumé de sa zone géographique contenant le nombre de consommateurs et de points d'eau présents ainsi que le volume d'eau distribué dans celle-ci.

**Elements du réseau** Un tableau interactif qui contient tous les différents éléments du réseau. Dans cette partie en fonction de ses privilèges, celui-ci peut supprimer, modifier ou ajouter des éléments du réseau. Il peut également trier ou faire des recherches dans ce tableau selon ses besoins.

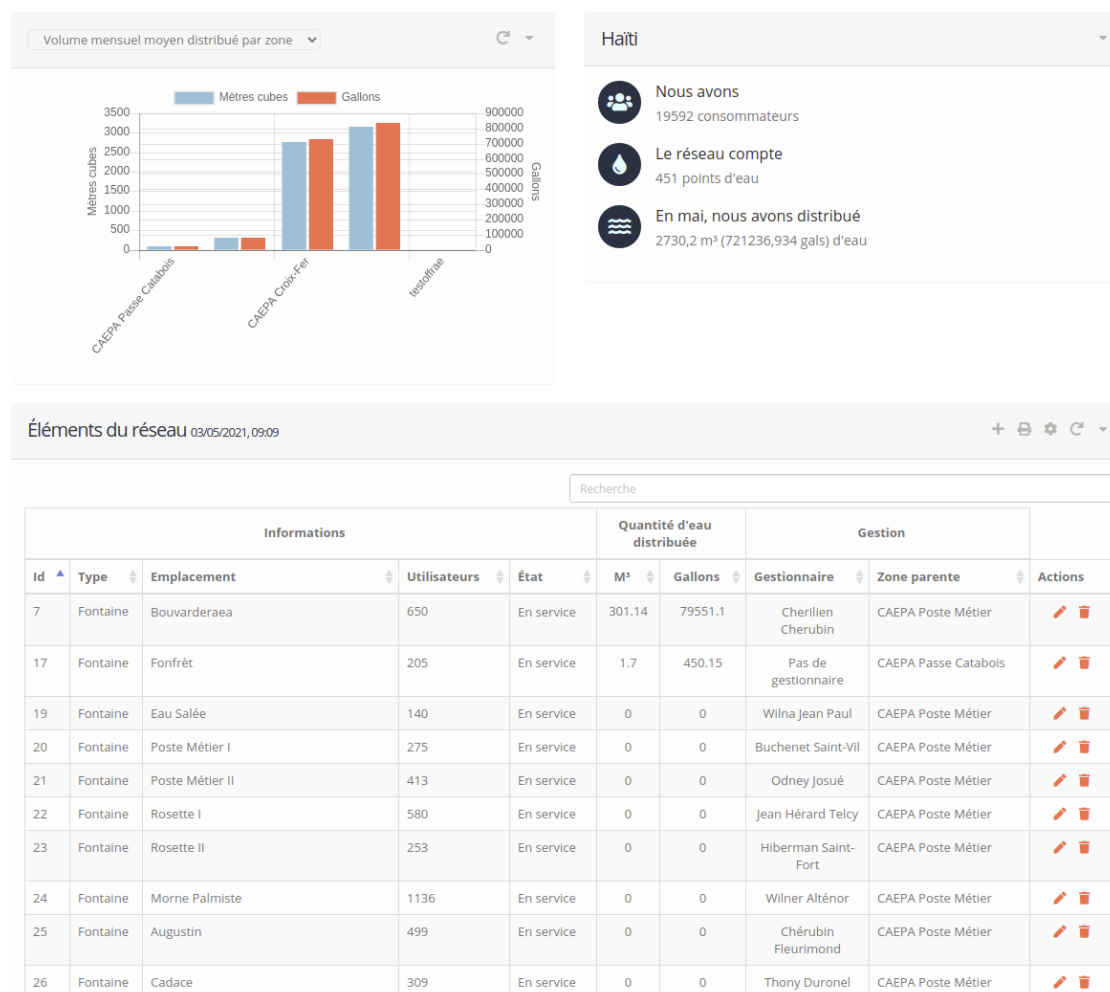


FIGURE 2.3 – Module réseau

## 2.2.5 Carte

Ce module comprend un tableau réduit des éléments du réseau ainsi qu'une carte interactive qui permet à l'utilisateur de voir où sont situés les différents éléments du réseau et de connaître ou d'encoder les coordonnées géographiques de ceux-ci.

Comme dans le module 2.2.4 "Réseau", il est également possible d'ajouter, modifier ou supprimer des éléments du réseau. De plus c'est ici que l'on peut ajouter des coordonnées géographiques à un élément du réseau. Soit en les entrant manuellement soit en utilisant la carte interactive.

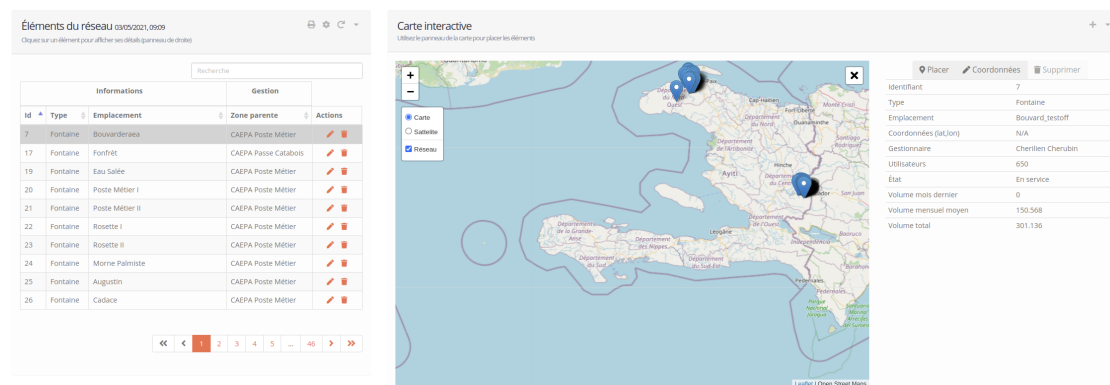
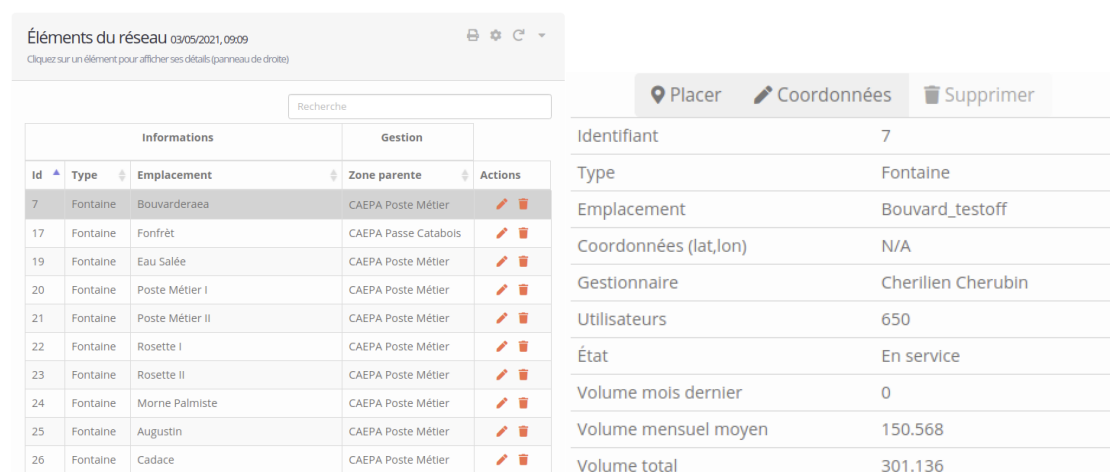


FIGURE 2.4 – Mordule carte



(a) Tableau des éléments réseaux simplifié

(b) Détails de l'élément réseau

## 2.2.6 Gestion de zone

Ce module comprend 3 tableaux permettant à l'utilisateur de gérer sa zone.

**Zones** Le premier contient les différentes zones géographiques encodées dans le système. Cliquer sur un des éléments du tableau permet de filtrer les éléments des deux autres tableaux qui seront décrits plus bas afin de ne garder que les éléments de la zone concernée. Ce tableau permet également si l'utilisateur a les privilèges requis d'ajouter, supprimer ou modifier une zone.

**Gestionnaires** Le deuxième contient la liste de tous les gestionnaires. Cliquer sur un gestionnaire permet à l'utilisateur de filtrer les éléments réseaux qui sont gérés par ce gestionnaire. De nouveau si l'utilisateur possède les privilèges nécessaires, il pourra ajouter, modifier ou supprimer des gestionnaires.

**Elements du réseau** Le dernier contient le même tableau que dans la section 2.2.4 "Réseau".

Zones

05/05/2021, 13:05

Opérateur en accès pour filtrer les gestionnaires et éléments

Recherche

<

FIGURE 2.6 – Module gestion de zone

(a) Tableau des zones

(b) Tableau des gestionnaires

## 2.2.7 Historique

Ce module contient les actions ayant été effectuées par des gestionnaires de fontaines. Ces actions doivent être validées ou refusées par un gestionnaire plus haut placé. On peut y retrouver deux tableaux.

**A valider** Le tableau du haut contient les éléments devant être validés. Ici si l'utilisateur a les privilèges requis, il peut choisir de confirmer ou non une action qui a été encodée.

**Historique** Le tableau du bas contient les éléments qui ont été validés ou refusés dans les 3 dernières semaines. Il s'agit simplement d'un historique récent.

Actions effectuées 03/05/2021, 09:09						
Cliquez sur une ligne pour afficher les détails ! Validez ou refusez les changements avec les boutons d'action.						
Id	Horodateur	Type	Utilisateur	Modification	Actions	
3800	2019-05-03	Ajouter	PC	Rapport mensuel	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3801	2019-05-03	Ajouter	PC	Rapport mensuel	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3802	2019-05-03	Ajouter	PC	Rapport mensuel	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3803	2019-05-03	Ajouter	PC	Rapport mensuel	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3804	2019-05-03	Ajouter	PC	Rapport mensuel	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3805	2019-05-03	Ajouter	PC	Rapport mensuel	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3806	2019-05-03	Ajouter	PC	Rapport mensuel	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3807	2019-05-03	Ajouter	PC	Rapport mensuel	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3808	2019-05-03	Ajouter	PC	Rapport mensuel	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3809	2019-05-03	Ajouter	PC	Rapport mensuel	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Actions effectuées 03/05/2021, 09:09						
Vous pouvez visualiser les changements acceptés ou validés des trois dernières semaines.						
Id	Horodateur	Type	Utilisateur	Modification	Action choisie	
3271	2019-04-03	Ajouter	PM	Paiement de consommateur	Validé	
3272	2019-04-05	Ajouter	PC	Ticket de problème	Validé	
3762	2019-05-02	Ajouter	PM	Paiement de consommateur	Validé	
3771	2019-05-03	Ajouter	PC	Ticket de problème	Validé	

FIGURE 2.8 – Module historique

## 2.2.8 Rapports

Dans ce module, l'utilisateur peut signaler les différents problèmes qu'il rencontre avec les infrastructures de distribution d'eau. Une fois le problème signalé, celui-ci peut consulter ce qu'il a encodé dans le tableau juste en dessous et modifier ou supprimer son signalement. Si ses privilèges sont suffisamment élevés, il pourra également gérer les signalements des autres personnes.

C'est également ici que l'utilisateur va pouvoir encoder les rapports mensuels. Si jamais la connexion internet n'est plus présente, il peut simplement enregistrer le formulaire avec les données qu'il a encodées afin de l'envoyer plus tard lorsque le réseau sera de nouveau accessible.

Envoyer un rapport

Le rapport sera visible pour tous vos gestionnaires.

Signaler un problème

Rapport mensuel

Tickets de support

03/05/2021, 09:09

⚙️ ↻ ▼

Recherche









Id ▲	Urgence ⚡	Emplacement ⚡	Type ⚡	Commentaire ⚡	Statut ⚡	Actions
26	Moyen	Fontaine Ti Charles	Mécanique	tuyau bobotyryrtyrt	Non résolu	 
30	Haut	Fontaine fon Melon	Mécanique	pa gen dlo	Non résolu	 
32	Haut	Kiosque Kote Iegliz katolik	Mécanique	Dlo a pa rive nan fonten yo	Non résolu	 
34	Bas	Prise Individuelle Samuel Etzer	Mécanique	dazf	Non résolu	 

FIGURE 2.9 – Module rapport

## 2.2.9 Consommateurs

Dans ce module, l'utilisateur retrouve 3 parties différentes :

**Schéma** Le même choix de schéma que décrit dans la section 2.2.4.

**Résumé de zone** Un résumé de la situation de votre zone (nombre de foyers consommant de l'eau, nombre de consommateurs, nombre de foyers n'ayant pas payé leur facture).

**Consommateurs** Un tableau qui reprend tous les consommateurs auxquels l'utilisateur a accès. Celui-ci peut ajouter, modifier ou supprimer des consommateurs. Ou bien il peut également juste consulter tous les détails sur ceux-ci.



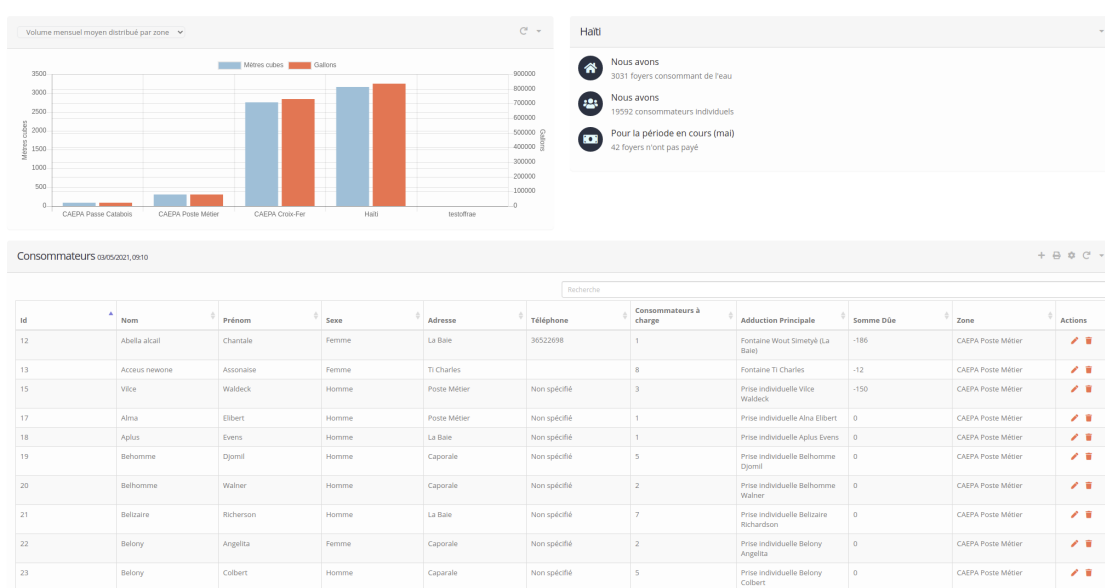


FIGURE 2.10 – Module consommateurs

Consommateurs 03/05/2021, 09:10

Recherche

Id	Nom	Prénom	Sexe	Adresse	Téléphone	Consommateurs à charge	Adduction Principale	Somme Due	Zone	Actions
12	Abella alcaïl	Chantale	Femme	La Bale	36522698	1	Fontaine Wout Simetyè (La Bale)	-186	CAEPA Poste Métier	
13	Acceus newone	Assonaise	Femme	Ti Charles		8	Fontaine Ti Charles	-12	CAEPA Poste Métier	
15	Vilce	Waldeck	Homme	Poste Métier	Non spécifié	3	Prise individuelle Vilce Waldeck	-150	CAEPA Poste Métier	
17	Alma	Elibert	Homme	Poste Métier	Non spécifié	1	Prise individuelle Alma Elibert	0	CAEPA Poste Métier	
18	Aplus	Evens	Homme	La Bale	Non spécifié	1	Prise individuelle Aplus Evens	0	CAEPA Poste Métier	

FIGURE 2.11 – Tableau des consommateurs

### 2.2.10 Finances

Dans ce module, l'utilisateur a accès à 4 tableaux différents qui lui permettent de gérer les finances de ses consommateurs. Celui-ci pourra modifier, supprimer ou ajouter des informations dans les différents tableaux en fonction de son niveau de privilèges.

**Zones** Un tableau contenant les différentes zones. Le fait de sélectionner une zone permet à l'utilisateur de filtrer les consommateurs en fonction de leur zone d'attribution.

**Consommateurs** Un tableau contenant les consommateurs auxquels l'utilisateur a accès. Le fait de sélectionner un consommateur permet de faire apparaître les deux tableaux suivants.

**Détails** Un tableau contenant les coordonnées du consommateur sélectionné ainsi que la somme due par celui-ci. Il n'est pas possible de modifier des données manuellement dans ce tableau.

**Paielements** Un tableau interactif contenant tous les paiements effectués par le consommateur sélectionné. Ici l'utilisateur peut ajouter, modifier ou supprimer des paiements.

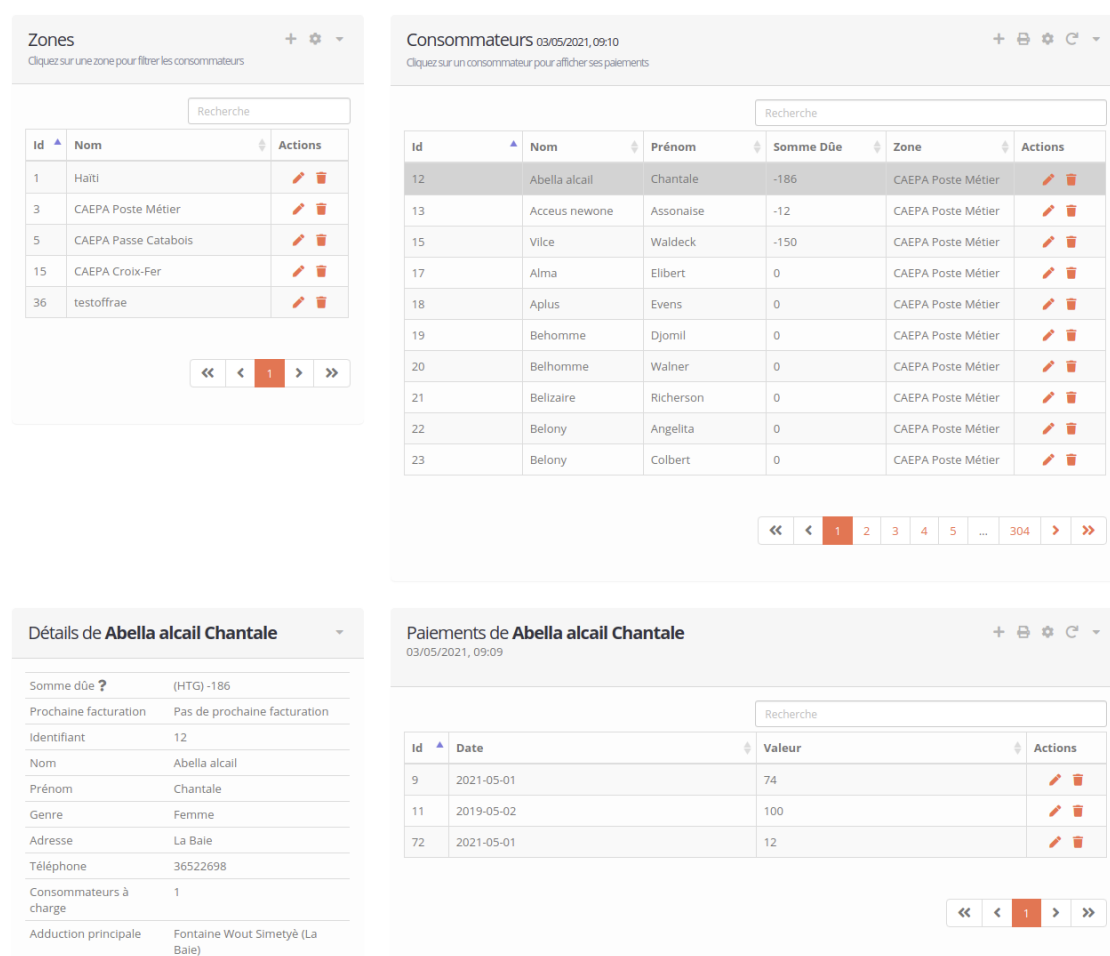


FIGURE 2.12 – Module finances

## 2.3 Problèmes réseaux

L'application HaïtiWater est une application web classique. Cela signifie que pour fonctionner, il faut qu'il y ait nécessairement une connexion à internet. Or en Haïti, cette connexion n'est pas très stable en ville et celui-ci peut aller jusqu'à être inexistant lorsque vous vous aventurez dans les zones les plus rurales.

Ces problèmes de connexions font que l'application est difficilement utilisable sur le terrain en toute sérénité car on ne sait jamais quand le réseau va devenir capricieux. Pour pallier ce problème, il a été décidé de transformer l'application web existante pour que celle-ci puisse fonctionner même lorsque le réseau est absent.

De par sa nature, une application web nécessite une connexion pour pouvoir fonctionner. Il a donc fallu étudier les différentes options qui pourraient permettre de faire fonctionner l'application hors-ligne. Parmi ces options, les deux qui ont été retenues étaient de soit créer une **application mobile** séparée de l'application web, soit de transformer l'application web existante en **Progressive Web App**.

L'avantage de créer une application dédiée aux mobiles est qu'il est très simple de gérer le fonctionnement hors-ligne de cette application. De plus on peut plus facilement utiliser les différentes capacités de l'appareil mobile comme la caméra, la localisation, ... L'inconvénient de cette solution est qu'il fallait du coup générer une nouvelle application de A à Z. De plus cela implique qu'il y aura deux codes différents dans deux langages différents à devoir maintenir.

L'avantage de la Progressive Web App est que tout se joue au niveau du navigateur. Il n'est donc pas nécessaire de devoir maintenir un deuxième code source. De plus aucune installation n'est nécessaire de la part de l'utilisateur et les mises à jour de l'application sont faites de manière transparente. L'inconvénient de cette solution est que la gestion du mode hors-ligne est un peu plus complexe, on ne peut pas profiter des capacités de l'appareil mobile aussi bien qu'avec une application native et cette technologie étant assez récente, il n'y a pas énormément de ressource d'aide en ligne et tous les navigateurs ne prennent pas en charge toutes les fonctionnalités proposées.

Au vu de la situation compliquée en Haïti et du temps qui sera alloué à ce mémoire, il a été décidé que la meilleure option était celle de la Progressive Web-App. La raison principale est qu'il aurait été difficile une fois le mémoire terminé d'avoir deux codes sources différents à maintenir et à mettre à jour.

# Chapitre 3

## Organisation

Étant le seul étudiant à réaliser ce mémoire, il n'a pas été nécessaire d'utiliser des outils de planification complets. Cette section contiendra surtout la planification des différentes tâches à accomplir permettant l'aboutissement de l'application ainsi que l'écriture de ce mémoire. Celles-ci ont été mises en place grâce aux discussions avec mes deux promoteurs.

### 3.1 Approche de travail

La première étape du mémoire a été de bien mettre en avant les différentes fonctionnalités à implémenter afin d'ajouter de la plus value au projet existant. L'idée de base étant de rendre l'application fonctionnelle hors-connexion.

Nous avons étudié la question avec mes deux promoteurs et Nahomie, l'étudiante venue d'Haïti. Celle-ci a pu nous apporter son expertise afin de sélectionner les fonctionnalités qui pourraient apporter une réelle valeur au projet et prioriser l'implémentation de celles-ci.

### Planification

**Mensuel** Dans un premier temps, il a fallu réaliser un plan général de déroulement du mémoire et mettre en avant les modules sur lesquels il fallait travailler en priorité et les différentes fonctionnalités à implémenter dans ceux-ci. Une grosse échéance était de pouvoir faire tester l'application par de vraie personne à partir de Février afin d'obtenir du feedback sur l'application.

**Hebdomadaire** Une réunion était prévue toutes les semaines en alternance avec mes deux promoteurs. Ces réunions permettaient de leur montrer l'état d'avancement du développement et d'avoir un feedback extérieur sur celui-ci. Ces réunions ont toutes eu lieu en vocal via l'application Teams. En effet en période de COVID, il était impossible de se voir régulièrement en présentiel.

**Quotidien** Etant donné que j'étais le seul à travailler sur le mémoire, un outil de suivi n'était pas nécessaire. Pour gérer l'organisation du travail une simple liste de tâches était maintenue. Cette liste était modifiée tous les jours en fonction de l'avancement de celles-ci. Les plus importantes étaient maintenues en haut de la liste afin de les prioriser.

Le plan n'a malheureusement pas pu être entièrement respecté pour différentes raisons :

- La technologie étant encore assez récente, les sources d'informations et d'aides sur le net sont encore peu présentes. Du retard a donc été pris lors du développement de certaines fonctionnalités complexes.
- L'installation du serveur en Haïti a pris plus de temps que prévu et il a donc fallu repousser la validation avec de vraies personnes.
- La période COVID-19 a apporté un manque de stimulation et de motivation qui a ralenti l'avancement du projet.

Malgré le retard pris, il a tout de même été possible de terminer le projet dans les temps.

## 3.2 Méthodologie

Même en travaillant seul, une bonne méthodologie de travail permet de garantir l'avancement régulier du projet. Elle permettra de transformer efficacement les différents besoins en fonctionnalités implémentées.

## Agile

Le méthodologie agile permet une réponse au changement plus flexible. Plûtôt que de planifier tout le projet directement, on développe par petit incrément que l'on termine termine à chaque itération.

Cette méthode a permis de faire évoluer le projet en fonction des différents feedback que j'ai reçu de la part de mes promoteurs et de l'étudiante venue d'Haïti. Ce qui n'aurait pas été possible avec une approche waterfall beaucoup plus séquentielle.

**Feature Driven Development.** L'application a été développée avec la méthode de développement par les fonctionnalités. Dans cette méthode on se focalise surtout sur la création d'une liste de fonctionnalités et sur leurs productions. Les différentes fonctionnalités ont été déterminées par mes deux promoteurs et l'étudiante de Haïti.

**Itérations** Dans le cadre d'une méthodologie agile, les itérations sont de courtes durées. Lors de ce mémoire, les itérations duraient deux semaines. Cela permettait d'avoir un feedback de la part des deux promoteurs que je voyais en alternance une semaine sur deux. Les avantages des itérations courtes sont que l'on peut détecter rapidement les défauts présents et de les corriger.

## Phases du mémoire

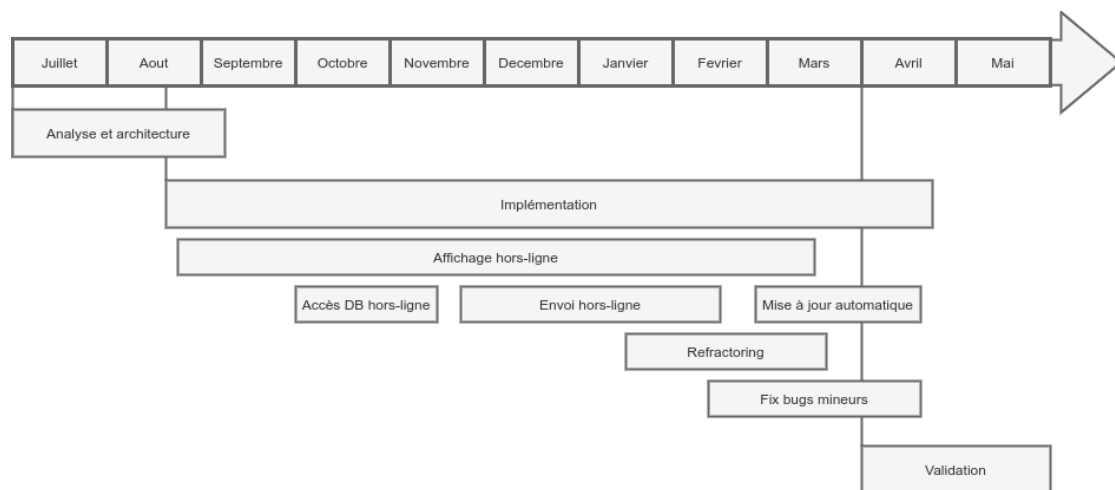


FIGURE 3.1 – Diagramme de Gantt

**Analyse et architecture.** Sur le schéma 3.1, on peut voir qu'il y a d'abord une phase d'analyse afin de déterminer les technologies qui seraient utilisées pour les différentes fonctions à implémenter. C'est dans cette phase qu'il y a eu beaucoup de discussions avec l'étudiante venue d'Haïti.

**Implémentation.** Cette phase compose la plus grosse partie du mémoire. C'est ici que toutes les décisions prises durant la phase d'analyse se sont mise en place. Cette phase se divise en plusieurs petites phases qui représentent les différentes fonctionnalités produites durant le mémoire.

**Validation.** La phase de validation n'intervient que tard dans le déroulement du mémoire. C'est simplement parce que la méconnaissance de la technologie à utiliser et le temps qu'a pris le déploiement de l'application sur les serveurs Haïtiens ont grandement retardé l'arrivée de cette phase.

**Rédaction.** La rédaction du mémoire n'a commencé que lorsque la plupart des fonctionnalités étaient terminées afin de lancer la validation au plus vite. Durant la rédaction, les parties produites ont été relues par mes deux promoteurs séparément afin d'améliorer celle-ci mais aussi par des relecteurs externes afin que le texte soit plus agréable à lire.



# Chapitre 4

## Analyse des besoins

Cette phase est très importante. C'est lors de celle-ci que nous, mes deux promoteurs, l'étudiante d'Haïti et moi-même, avons choisi quelle technologie était la plus adaptée afin de faire évoluer l'application existante de la meilleure façon possible. C'est également ici qu'ont été déterminées toutes les fonctionnalités à implémenter et leur ordre de priorité.

J'ai avec l'aide de l'étudiante Haïtienne et les bons conseils de mes promoteurs, réalisé une analyse de moscow qui sera décrite dans la section 4.1. Cette méthode permet de prioriser les tâches d'un projet en fonction de leur criticité, c'est-à-dire du niveau d'impact de la tâche sur la réalisation du projet. Cette analyse de Moscow est assez représentative de la direction que devait prendre le projet et représente bien les différentes tâches à accomplir.

## 4.1 Besoins fonctionnels

Fonctionnalités hors-ligne	M	S	C	W	Notes
Accès à l'application	✓				Indispensable
Afficher les informations de l'utilisateur connecté	✓				Tâche importante mais pas vitale
Modifier son profil				✓	N'impacte pas l'utilisation hors-ligne
Visualiser le volume mensuel distribué par zone		✓			Cela peut être fait sur le long terme
Répartir les consommateurs par genre		✓			Cela peut être fait sur le long terme
<b>Lister</b> <ul style="list-style-type: none"> <li>les éléments du réseau</li> <li>les consommateurs</li> <li>les paiements</li> <li>les tickets</li> <li>les zones</li> <li>les gestionnaires</li> <li>les logs</li> </ul>	✓ ✓ ✓ ✓	✓ ✓	✓		Indispensable au travail de terrain  Facilite le travail de terrain  N'est utile qu'aux gestionnaires de zone qui ne sont pas sur le terrain
Afficher les éléments du réseau sur la carte			✓		Cela peut être fait sur le long terme
<b>Ajouter, supprimer ou éditer</b> <ul style="list-style-type: none"> <li>les consommateurs</li> <li>les paiements</li> <li>les tickets</li> <li>les éléments du réseau</li> <li>les zones</li> <li>les gestionnaires</li> <li>les logs</li> </ul>	✓ ✓ ✓	✓	✓ ✓ ✓		Indispensable au travail de terrain  Facilite le travail de terrain N'est utile qu'aux gestionnaires de zone qui ne sont pas sur le terrain
Signaler un problème (soit mécanique, de qualité ou autres) par le gestionnaire de fontaine	✓				Indispensable pour une bonne gestion du service de distribution d'eau
Envoi de notification au gestionnaire de zone une fois un problème signalé				✓	Tâche importante mais inaccessible à cause du coût de développement qui n'est pas compatible avec le temps qui reste.
<b>Faire un rapport mensuel</b> <ul style="list-style-type: none"> <li>Encoder une consommation</li> <li>Mise à jour de l'état de paiement du consommateur</li> </ul>	✓ ✓				Indispensable pour un suivi du service de distribution d'eau Indispensable pour un suivi du service de distribution d'eau

FIGURE 4.1 – Moscow

Les besoins fonctionnels répondent aux points précis à implémenter, ils représentent toutes les tâches à accomplir pour que le projet soit réussi. Dans le tableau 4.1, les lettres ont la signification suivante :

**M** signifie "Must have", le projet serait un échec si l'application ne possédait pas cette fonctionnalité à la fin du développement.

**S** signifie "Should have", cette fonctionnalité doit être faite dans la mesure du possible. Ces tâches sont importantes mais pas vitales.

**C** signifie "Could have", cette fonctionnalité peut être faite si cela n'affecte pas les autres tâches importantes. Ce sont des tâches de confort qui peuvent être effectuées si vous avez suffisamment de temps et que les tâches des deux catégories précédentes ont été réalisées.

**W** signifie "Won't have but would like", ce qui ne sera pas fait cette fois, mais plus tard. Réaliser ces tâches est un luxe qu'on a théoriquement pas le temps de se payer.

Toutes les tâches de type **M**, **S** et **C** ont été réalisées durant ce projet. Seule les tâches **W** n'ont pas été implémentées.

Lors de l'analyse des besoins, nous avons bien entendu repris l'analyse faite par les anciens mémorants [2]. Ils proposaient trois types différents d'accès à l'application et aux données lorsque le réseau n'est pas disponible. Les 3 options proposées sont les suivantes.

**Accès aux formulaires.** Avec cette solution, l'utilisateur aurait toujours un accès aux différents formulaires d'ajouts de données. Celui-ci pourrait pré-remplir les formulaires et les enregistrer dans le cache du navigateur afin de pouvoir les envoyer plus tard dans le cas où le réseau ne serait pas disponible immédiatement. Cette solution est déjà partiellement mise en place dans l'application de base. Il est en effet déjà possible de pré-remplir son rapport mensuel et de le sauvegarder pour plus tard.

**Accès statique aux données.** Cette solution demande d'avoir une version simplifiée de la base de données en copie sur l'appareil de l'utilisateur mais sans la possibilité de modifier celle-ci. Cela permettrait aux utilisateurs de consulter les données hors-ligne mais pas d'en entrer de nouvelles. L'avantage de cette solution est que la synchronisation des tables est très simple et qu'il n'y a pas de problèmes d'inconsistances des données.

**Accès complet aux données.** Cette solution permettrait aux utilisateurs de pouvoir consulter, ajouter, supprimer et modifier des données de la base de données. Ainsi même en cas de perte de réseau, l'utilisateur pourrait continuer à réaliser ses tâches sans le moindre soucis. Cette solution est la plus compliquée à mettre en place car elle demande de synchroniser les différentes versions des données. En effet si deux utilisateurs modifient en même temps les mêmes données en étant hors-ligne, il faudrait pouvoir déterminer quelle modification conserver sans créer d'inconsistances.

L'option qui a été choisie est celle de l'accès complet. C'est la seule solution qui permettrait vraiment à l'application d'être utilisée de manière intensive sur le terrain. C'est également l'option la plus intéressante à mettre en place du point de vue technique.

## Affichage des pages

Une partie importante des besoins est bien évidemment l'accès aux différentes pages de l'application lorsque la connexion internet n'est plus disponible.

C'était la partie prioritaire du projet car sans cette fonctionnalité, il aurait été impossible d'accéder aux différentes données. Les premières pages concernées par cette évolution ont été celles utiles aux gestionnaires de fontaines qui sont le plus souvent sur le terrain. Les différents points du tableau 4.1 concerné par cette partie sont :

- L'accès à l'application
- Afficher les informations de l'utilisateur connecté
- Visualiser le volume mensuel distribué par zone
- Répartir les consommateurs par genre

Comme on peut le voir les deux premiers points sont bien placés dans la catégorie "Must have". Les deux derniers sont dans le "Should have" car ces fonctionnalités ne sont que des schémas qui s'affichent sur certaines pages et ceux-ci ne sont pas absolument essentiels.

## Accès aux données

La deuxième partie importante des besoins fonctionnels est d'avoir accès aux données de la base de données même lorsque la connexion internet n'est pas disponible. En effet le simple accès aux pages hors ligne n'a que peu d'intérêt si l'on ne peut pas consulter de données sur ces pages. Dans le tableau 4.1, cela est représenté les points suivants :

- Lister les éléments du réseau
- Lister les consommateurs
- Lister les paiements
- Lister les tickets
- Lister les zones
- Lister les gestionnaires
- Lister les logs

Les quatre premiers points sont dans la catégorie "Must have" car ceux-ci représentent les données qui seront utilisées en permanence par les acteurs sur le terrain, dans notre cas les gestionnaires de fontaines. Ce sont donc les données qu'il faut avoir prioritairement dans l'application hors-ligne.

Les trois derniers sont dans la catégorie "Should have" et "Could have" car ceux-ci sont surtout utile pour les gestionnaires de zones qui seront assez peu sur le terrain. Malgré tout il reste intéressant de pouvoir accéder à ces données même lorsque le réseau internet n'est pas disponible dans le cas où un gestionnaire devrait se déplacer sur le terrain ou bien en cas de panne réseau.

Un dernier point non listé est d'afficher les éléments du réseau de distribution d'eau sur une carte interactive même lorsque l'on a pas de connexion réseau. Ce point a été mis dans les "Could have" car l'usage de la carte en mode hors-ligne est une fonction qui prendrait beaucoup d'espace sur un appareil mobile et cette fonctionnalité n'est pas essentielle à un usage sur le terrain.

## Modifications des données

La dernière partie mais non la moindre consiste à gérer l'envoi des données lorsque la connexion au réseau n'est pas disponible. Cela permettrait aux utilisateurs de pouvoir complètement se passer de la solution crayon/papier. Dans le tableau 4.1 cette partie est représentée par les points suivant :

- Ajouter, supprimer ou éditer
  - Les consommateurs
  - Les paiements
  - Les tickets
  - Les zones
  - Les gestionnaires
  - Les logs
- Signaler un problème par le gestionnaire de fontaine
- Faire un rapport mensuel

La plupart des points sont situés dans la partie "Must have" car ces fonctionnalités sont essentielles au bon fonctionnement de l'application hors-ligne. Par contre la modification des éléments du réseau, des zones, des gestionnaires et des tickets sont dans les catégories "Should have" and "Could have" car ces données sont surtout utilisées par les gestionnaires de zones qui ne vont que peu sur le terrain et pas par les gestionnaires de fontaines.

## 4.2 Besoins non-fonctionnels

Les besoins non-fonctionnels sont soit des besoins optionnels, soit des besoins liés à l'implémentation et à l'interopérabilité générale. En réalisant ces besoins non fonctionnels, on obtient un système fiable et qualitatif.

## Multi-plateforme

Ce projet se déroule dans le cadre, et est la suite, d'un mémoire universitaire. La prochaine étape pour celui-ci une fois le mémoire terminé est qu'il soit déployé et maintenu par une équipe de développeur en Haïti.

Le problème est qu'on ne connaît pas encore le temps et le budget qui seront alloués à la suite de ce projet. Par contre, ce que l'on sait, c'est qu'il faut que l'application puisse tourner sur un maximum d'appareils différents. Les choix technologiques qui seront décrits dans la section 5.2 ont donc été pris en prenant en considération le fait que l'application doit pouvoir tourner sur tout type de configuration.

## Technologies simples et populaires

Comme précisé précédemment, nous ne connaissons pas encore l'équipe de développeur qui va reprendre le projet. Il a donc fallu continuer à utiliser des technologies qui sont assez rapide à apprendre et accessible à tous. Cela afin de faciliter la maintenance et l'évolutivité de l'application lorsque celle-ci sera déployée en Haïti. De plus il faudra également fournir une documentation complète et détaillée afin d'aider ces futurs développeurs lorsqu'ils devront reprendre l'application.

### 4.3 Structure des données hors-ligne

Pour que l'utilisateur puisse accéder aux données en étant hors-ligne, il faut bien évidemment que celles-ci soient stockées sur l'appareil. Pour ce faire il a fallu trouver une solution qui prendrait peu de place sur l'appareil de l'utilisateur, qui permettrait un accès rapide en lecture et la possibilité de modifier ces données. Deux solutions ont alors été envisagées.

**JSON** Stocker les réponses au format JSON telles qu'envoyées par le serveur dans la cache du navigateur. L'avantage de cette solution est qu'il est très rapide de récupérer les données à afficher sur la page lors du chargement de celle-ci car les données nécessaires sont stockées telles quelles dans le navigateur,

elle est également très facile à mettre en place. Si l'on avait utilisé l'accès statique aux données comme décrit dans la section 4.1, cette solution aurait été suffisante. Mais comme il faut pouvoir modifier les données, cette solution n'est pas la plus adaptée car il faudrait mettre le json en mémoire, le modifier et ensuite réécrire tout le nouveau JSON dans le cache à chaque modification de données.

**DB** Utiliser un système de gestion des données intégré au navigateur. Cette option a pour avantage le fait d'avoir un vrai système de gestion de données. Une fois les données enregistrées dans le navigateur, il est très facile de faire des requêtes sur ces données ou bien d'ajouter en modifier certaines. Par contre l'inconvénient est que lors de la synchronisation des différentes tables dans le navigateur, l'appareil doit fournir un travail supplémentaire.

La solution qui a été retenue est la deuxième option. C'est en effet elle qui permet le plus facilement de gérer l'accès complet à l'application en permettant de modifier, supprimer ou ajouter des données hors-ligne.

## 4.4 Conclusion

Dans ce chapitre, une analyse poussée des besoins fonctionnels et non fonctionnels a été faite afin de déterminer les différentes fonctionnalités à implémenter et leur ordre de priorité. Une étude des différentes options de synchronisation des données a également été réalisée afin de trouver la solution qui serait la plus adaptée à un usage de l'application sur le terrain en cas de perte de connexion au réseau internet.



# Chapitre 5

## Implémentation

### 5.1 Application de base

Dans cette section, un résumé de l'application HaïtiWater de base vous sera présenté. Si vous désirez de plus ample informations sur les raisons des choix technologiques suivant et les détails de leur implémentation, vous pouvez consulter le document [2].

#### Backend

L'application HaïtiWater est une application web propulsée par le langage python. C'est ce langage qui a été choisi pour le backend car c'est un langage de programmation populaire et facile à apprendre. Il existe énormément de bibliothèques logicielles et de documentations sur internet ce qui le rend plus adapté à la situation. Vu la taille du projet, un framework a dû être utilisé. Le choix s'est porté sur Django principalement pour ses outils de gestion de base de données, particulièrement efficace pour les données géographiques.

Le SGBD utilisé pour l'application est PostgreSQL. Celui-ci a été choisi pour deux raisons :

- D'abord parce que c'est le système qui est recommandé par Django car c'est celui qui s'adapte le mieux à l'Object relational mapping. C'est Django qui va se charger d'effectuer la connexion avec la base de données et qui va se charger d'y envoyer toutes les requêtes.
- Ensuite parce qu'il existe l'extension PostGIS de PostGreSQL permettant de traiter facilement les données géographiques.

Le serveur possède un module API qui permet de séparer les différents rôles de celui-ci. Cela facilite également la récupération des données par les différentes bibliothèques du front-end.

## Frontend

Pour le front-end, aucun framework n'a été utilisé mais seulement la librairie bootstrap qui permet d'agencer l'interface graphique par blocs et de les personnaliser. La raison de son utilisation est qu'il s'agit d'une bibliothèque très connue et utilisée et donc facile à apprendre.

Deux autres bibliothèques ont été utilisées afin de gérer l'affichage des données dans l'application. La première est "datatable.js", cette bibliothèque permet l'affichage des données sous forme de table. La deuxième est Chart.JS, elle permet l'affichage des données sous forme de graphiques. Toutes les données sont récupérées via l'API qui a été créée pour l'occasion.

## Client

Du côté du client, l'application utilise la mise en cache afin d'éviter les téléchargements répétitifs des mêmes fichiers. Pour gérer cette mise en cache, on fait ici usage du Service Worker qui se place en middleware entre le client et le serveur afin d'intercepter les requêtes pour les mettre en cache.

## 5.2 Choix technologiques

Une étude a été faite pour savoir qu'elle était l'option la plus adaptée pour que l'application puisse être accessible lorsque l'utilisateur n'a plus de connexion au réseau internet. Plusieurs options ont alors été envisagées.

## **Application android**

Cette option est la plus facile à implémenter et pour gérer le fonctionnement hors-ligne de l'application. On aurait dans ce cas une application android qui cohabiterait avec l'application web existante. Cela aurait permis aux gestionnaires de fontaines d'accéder à une version limitée de l'application sur leur téléphone ou tablette quand ils sont sur le terrain.

Le problème de cette solution est qu'elle ne fonctionne que sur Android et pas sur iOS. De plus elle oblige à maintenir et à faire évoluer deux codes sources dans deux langages différents ce qui demanderait plus de temps ou deux team de développement.

## **Xamarin**

Xamarin est une plateforme open source qui permet de créer des applications modernes et performantes pour android, iOS et Windows grace au .NET [3]. Le gros avantage de cette solution est qu'il suffit d'utiliser la partie API existante du serveur pour pouvoir créer une application qui serait disponible sur tous types d'appareils.

Le gros défaut de Xamarin est qu'il faut un temps d'apprentissage avant utilisation. Cette technologie n'est pas ultra répandue et il sera donc plus difficile pour Haïti de faire maintenir et évoluer le projet. De plus la méconnaissance totale de Xamarin où moment du choix à fait que cette solution n'était pas viable.

## **React-native**

Cette technologie permet d'écrire des applications qui seront disponible à la fois pour Android et iOS. Il est même possible de dériver facilement du React-native pour en faire une application web react. Cela permettrait de ne pas trop diversifier les codes sources et ainsi faciliter leur maintenance.

La difficulté ici réside de nouveau dans le fait qu'encore peu de développeur connaissent REACT. Bien que cette solution commence à être de plus en plus utilisée elle reste plus complexe à maintenir. Et même si on peut relativement facilement transformer React en React-native, il faudra tout de même maintenir deux codes sources différents à l'avenir.

## Progressive web-app

Les Progressive web-app sont de simple application web utilisant les nouvelles capacités des navigateurs modernes. Grâce aux service workers et à l'app-manifest, une application web devient fiable même sans réseau et "installable" sur un appareil mobile. Une fois installée, une progressive web-app peut tourner en full screen pour donner l'illusion qu'il s'agit d'une application native. Jusqu'à récemment, seules les applications spécifiques à une plateforme pouvaient utiliser ce genre de fonctionnalité.

C'est cette solution qui a été choisie pour faire évoluer le projet. Les raisons de ce choix sont que :

- Il n'est pas nécessaire de devoir maintenir deux codes sources différents contrairement aux autres solutions citées précédemment.
- On peut entièrement réutiliser le code de l'application de base et l'améliorer.
- L'usage de service worker permet de diminuer la charge du serveur et d'améliorer l'expérience utilisateur sans que celui-ci ne doivent se soucier d'installer ou de mettre à jour l'application puisqu'il suffit de se connecter une fois à l'application pour que celle-ci s'installe automatiquement.

### 5.3 Client

Toute cette partie décrira le fonctionnement du service worker. Un service worker joue le rôle de serveur proxy entre une application web et le navigateur. C'est lui qui va permettre une utilisation de l'application déconnectée du réseau. Il va intercepter toutes les requêtes faites par le navigateur et va effectuer différentes actions en fonction de la requête interceptée. C'est via ce service worker que l'on accède aux nouvelles APIs des navigateurs [5].

#### 5.3.1 Stratégie de synchronisation des pages

Toutes les pages de l'application sont synchronisées à l'aide du service worker et sont stockées dans un des caches du navigateur. Les fichiers nécessaires à l'affichage des pages suivent différentes stratégies de synchronisation en fonction de leur usage.

## Fichiers statiques

Les feuilles de styles et les fichiers javascripts ne sont synchronisés qu'une seule fois lors de l'installation du service worker. Ceux-ci seront mis à jour à chaque fois que le service worker est remplacé dans le navigateur. Cela se produit lorsqu'une nouvelle version de celui-ci est disponible.

Si les développeurs veulent forcer la récupération de nouveaux fichiers, il leur suffit de changer la variable "cacheVersion" dans le code du service worker. Si jamais certains fichiers de styles ou certains fichier n'ont pas pu être téléchargés lors de l'installation, ceux-ci seront automatiquement mis en cache lorsque le navigateur de l'utilisateur les téléchargera pour afficher les pages demandées.

## Pages personnalisées

Toutes ces pages ne contiennent pas de contenu qui changent dynamiquement. Par contre, elles contiennent du contenu "personnalisé" en fonction de l'utilisateur qui est connecté à l'application (son nom, sa zone, les résumés de sa zone, ...). Comme le côté personnalisé de ces pages est créé par Django, il faut pouvoir mettre à jour ces pages régulièrement au cas où les données à afficher sur celles-ci auraient changé.

C'est la raison pour laquelle on va utiliser le mode de synchronisation "Stale While Revalidate". Le principe est qu'à chaque fois qu'une requête va être faite pour récupérer ces pages, le service worker d'abord répondre avec la page qui est stockée dans le cache prévu à cet effet. Il va dans le même temps transmettre la requête au serveur afin de pouvoir mettre à jour l'ancienne page stockée dans le cache. Ces pages sont mises en cache lorsque l'utilisateur se connecte pour la première fois à l'application et elles sont supprimées lorsque celui-ci se déconnecte. Les pages concernées par ce mode de synchronisation sont les suivantes :

- /accueil
- /offline
- /aide
- /profil/editer
- /consommateur
- /reseau

## Pages fixes

Ces pages ne contiennent aucun élément personnalisé en fonction de l'utilisateur ou aucun élément qui doit être mis à jour. Le côté dynamique de ces pages est entièrement géré par les bibliothèques `datatable.js` et `graph.js` qui vont récupérer les données à afficher directement via l'API du serveur.

Toutes ces pages sont également chargées lorsque l'utilisateur se connecte pour la première fois et elles se suppriment également lorsque celui-ci se déconnecte. La suppression de ces pages lorsque l'utilisateur se déconnecte n'est pas réellement nécessaire. Mais pour une raison de sécurité ces pages sont tout même supprimées à la déconnexion. Les pages concernées par ce type de synchronisation sont les suivantes :

- /carte
- /gestion
- /historique
- /rapport
- /finances

### 5.3.2 Stratégie de synchronisation de la DB

Pour gérer l'accès aux données hors-ligne, j'ai utilisé l'API de bas niveau `indexedDB`. Il s'agit d'un système de gestion de bases de données orientée objet basé sur javascript, on y stocke des objets qui sont indexés avec une clé.

Les opérations effectuées par `indexedDB` sont réalisées de manière asynchrone et transactionnelle. Cela permet de ne pas bloquer le fonctionnement de l'application lorsque des données sont en cours de chargement [4]. Pour faciliter l'usage d'`indexedDB` qui n'est pas forcément simple à prendre en main, j'ai utilisé la bibliothèque "Dexie.js" [1]. Celle-ci offre des fonctionnalités qui facilitent le développement :

- Une meilleure gestion des erreurs
- Une meilleure gestion des requêtes
- Un code simplifié et plus court

## Récupération des données

Lors du premier chargement de l'application, le service worker va envoyer des requêtes à l'API du serveur afin de récupérer toutes les données nécessaires à l'affichage des différentes tables. Toutes ces requêtes sont faites de manière asynchrone afin de ne pas bloquer l'application. Au delà de la première synchronisation automatique, l'utilisateur peut décider lui-même via l'interface de mettre à jour toutes les tables ou bien juste certaines d'entre elles en fonction de ses besoins.

Pour éviter d'avoir un dédoublement des données tout en conservant le principe de la base de données non-relationnelles, les données qui sont liés et qui doivent s'afficher dans la même table sont toutes regroupées dans la même collection. Cela permet de conserver une grande vitesse de lecture des données et de ne pas ralentir le fonctionnement de l'application.

Pour accélérer le processus, toutes les requêtes pour récupérer les données sont envoyées simultanément. Ainsi dès qu'une réponse est reçue, le service worker peut commencer à ajouter les données dans l'indexedDB à l'aide de la bibliothèque "dexie.js" [1]. Pour ce faire, le service worker parcourt simplement tout le fichier JSON reçu de l'API et ajoute toutes les données présentes dans la collection adéquate.

## Envoi des données

Pour gérer l'envoi des données, la première solution envisagée a été d'utiliser la nouvelle API "background-sync". Quand le service worker détecte qu'une requête réseau a échoué, il peut l'enregistrer comme un événement sync qui sera délivré une fois que le navigateur a récupéré le réseau.

L'avantage de cette solution est que le service worker peut envoyer les données même si la page a été fermée ce qui permet d'éviter que les données restent indéfiniment non envoyées. Malheureusement cette API n'est pour l'instant prise en charge que par certains navigateurs sur PC et Android mais n'est pas du tout prise en charge par iOS. Il a donc été décidé de laisser cette solution de côté.

Du coup pour pallier à ce problème, j'ai implémenté une solution qui est compatible avec tous les navigateurs. Lorsque l'utilisateur essaie d'ajouter, modifier ou supprimer des données, celles-ci sont dans un premier temps stockées dans

l'indexedDB avant d'être envoyées. Cela permet en cas de perte de connexion au réseau internet, de ne pas perdre les données que l'utilisateur vient d'encoder. Une fois enregistrées dans l'indexedDB, le service worker va automatiquement essayer d'envoyer les données une première fois. En cas de réussite, les données sont supprimées de l'indexedDB et en cas d'échec d'envoi, les données restent stockées afin de pouvoir être envoyées plus tard. A chaque fois que l'utilisateur tentera de récupérer des données sur le serveur, le service worker va d'abord essayer d'envoyer toutes les données en attente avant de récupérer les données du serveur. Cela permet d'éviter d'avoir des données non envoyées qui restent en attente.

Si l'envoi des données réussit, le serveur répond avec les nouvelles données mises à jour. Lorsque le service worker reçoit cette réponse, il va mettre à jour les données qui sont dans le navigateur afin que l'utilisateur puisse voir directement les changements qui ont été effectués même s'il utilise le mode hors-ligne de l'application.

### 5.3.3 Gestion des changements d'utilisateurs

Afin de conserver une sécurité maximale, à chaque déconnexion d'un utilisateur, toutes les données et les fichiers (hors fichiers statiques) sont supprimés du navigateur. Il aurait été possible de ne pas supprimer ces données et de gérer des fonctionnalités multi-utilisateurs mais cela aurait impliqué de demander à l'utilisateur d'effectuer des démarches supplémentaires pour supprimer ces données.

De plus à l'heure actuelle chaque utilisateur possède son propre appareil et il n'est donc pas nécessaire de gérer le multi-utilisateur au détriment de la sécurité. En effet si on décide de laisser les données dans le navigateur même après la déconnexion d'un utilisateur, les données restent accessibles via l'inspection du navigateur. Du coup si une personne mal intentionnée passait sur l'appareil après d'un utilisateur, il pourrait récupérer toutes les données qu'il souhaite.

### 5.3.4 Système de notification

Le système de notification, comme indiqué dans le tableau 4.1, ne faisait pas partie des fonctionnalités à implémenter pour l'instant. Toutefois, un petit système de notification local a tout de même été mis en place pour notifier l'utilisateur lorsqu'il y a des données qui sont en attente d'envoi dans le navigateur. Cela afin que ces données ne restent pas en attente indéfiniment.



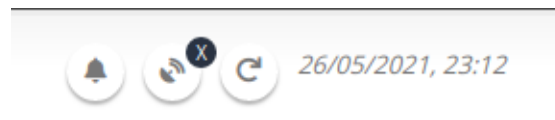
## 5.4 Interface utilisateur

L'interface utilisateur existante a du subir quelques modifications afin de pouvoir accueillir le mode hors-ligne. La nouvelle application permet en effet deux modes différents d'utilisations.

### 5.4.1 Changement de mode

Afin de passer d'un mode d'utilisation à l'autre un bouton de mode a été ajouté en haut de l'interface à coté de la cloche de notification. Quand ce bouton affiche un rond vert, cela signifie que l'on se trouve dans le mode online de l'application. Lorsque le rond est noir alors ça signifie que l'application est en mode offline.

A coté de ce bouton de changement de mode a été ajouté un bouton permettant de mettre à jour toutes les données présentes dans le navigateur. Cela afin de ne pas obliger l'utilisateur à mettre à jour toutes les tables une par une. Juste à droite de ce bouton, on peut lire la date et l'heure des données les plus anciennes présentes dans le navigateur. Si il n'y pas de données chargées, par exemple quand on lance l'application pour la première fois, alors on pourra lire la phrase "Pas encore de données". Cela permet à l'utilisateur de savoir à quel point les données du navigateur sont à jour ou non.



### Mode online

Il s'agit du mode par défaut lorsque l'on se connecte pour la première fois à l'application. Il faut du temps à toutes les données pour se charger dans le navigateur afin que l'application puisse fonctionner en mode hors-ligne. Du coup mettre ce mode-ci par défaut permet à l'utilisateur de déjà utiliser l'application en attendant que les données soient bien toutes chargées. Dans ce mode les différentes pages vont aller récupérer les données à afficher directement dans la base de données grâce à l'API et pas dans l'indexedDB du navigateur.

Ce mode est très utile lorsque l'on a besoin d'avoir les dernières données disponible et que le réseau le permet. Si vous désirez plus d'information sur ce mode de fonctionnement vous pouvez consulter le mémoire de la première version de l'application [2].

## Mode offline

Ce mode permet d'afficher les données qui sont présentes localement dans l'indexedDB du navigateur. C'est dans ce mode que l'utilisateur pourra utiliser l'application lorsque la connexion au réseau internet n'est pas disponible. Lorsque l'on passe dans ce mode, la couleur du bandeau de l'application change pour passer du bleu au rouge et ce afin d'indiquer qu'on est bien dans le mode hors-ligne, voir 5.1, et que les données ne sont pas récupérées directement sur le serveur.

FIGURE 5.1 – Mode



Dans ce mode, lorsque du travail sur des données n'a pas pu être envoyé, les données concernées apparaissent en mauve dans le tableau pour que l'utilisateur sache que des modifications sont en attente sur celles-ci. Une fois que les données ont pu être envoyées et que le serveur a répondu un status 200, les données hors-ligne que nous avons modifiées sont mise à jour localement pour refléter les changements faits sur le serveur et ces données n'apparaissent plus en mauve dans les tables, voir 5.2.





11	Agenordera	SAraj	Femme	Voie Cardijn	2478844888	6	Fontaine Siwèl	-100	CAEPA Passe Catabois	 
12	Abella	Chantale	Femme	La Baie	36522698	6	Fontaine Wout Simetyè (La Baie)	-195	CAEPA Poste Métier	 

FIGURE 5.2 – Donnée pas synchronisée

### 5.4.2 Module à synchroniser

Il s'agit d'un nouveau module qui a été ajouté pour les besoins du mode hors-ligne. Dans ce module, l'utilisateur pourra retrouver un tableau qui contient toutes les données qui n'ont pas été envoyées vers le serveur. Dans ce tableau, il peut retrouver différentes informations, voir 5.3.

Actions effectuées

Cliquez sur une ligne pour afficher les détails ! Validez ou refusez les changements avec les boutons d'action.

Date	Table	Type	Id	Status	Actions
30/05/2021, 15:36	Rapport mensuel	Ajouter	?	En attente	<input checked="" type="checkbox"/> <input type="checkbox"/>
<div>ID élément réseau : 77</div> <div>Status : Actif</div> <div>Jours de fonctionnement : 18</div> <div>Heures de fonctionnement : 13</div> <div>Mois : mai</div> <div>M³ : 1241</div> <div>Prix M³ : 31231€</div> <div>Total : 1312€</div>					
30/05/2021, 15:35	Historique	Accepter	3805	En attente	<input checked="" type="checkbox"/> <input type="checkbox"/>
30/05/2021, 15:14	Consommateur	Editer	11	En attente	<input checked="" type="checkbox"/> <input type="checkbox"/>

FIGURE 5.3 – Module à synchroniser

L'utilisateur peut choisir de supprimer des données du tableau, auquel cas il ne pourra plus essayer de les envoyer car celle-ci seront supprimées de l'indexedDB. Ou bien il peut choisir d'essayer d'envoyer une donnée en particulier ou toutes en même temps avec le bouton prévu dans cette optique dans l'entête du tableau.

## 5.5 Serveur

Dans cette partie je vais décrire le fonctionnement du serveur. Je ferai d'abord un petit résumé sur le fonctionnement de base du serveur et ensuite j'expliquerai les changements que j'ai apporté au serveur afin de récupérer les données nécessaires à l'affichage hors-ligne.

### Requêtes

**Fichiers statiques.** Le serveur répond a ces requêtes en envoyant tous les fichiers CSS et Javascripts.

**Documents.** Le serveur renvoie les pages HTML à afficher à l'utilisateur. Ce sont les documents qui vont importer les fichiers statiques.

**Données.** Le serveur renvoie toutes les données demandées. Pour ce faire, on utilise l'Object-relational mapping. Django se charge d'effectuer la connexion avec la base de données et de lui renvoyer toutes les requêtes. Cela permet d'interagir avec les données sans avoir à toucher au SQL.

Les fichiers statiques et les documents permettent au navigateur d'afficher toute l'interface graphique. Chaque module du serveur permet de desservir une page de l'application. Ces pages peuvent avoir des éléments personnalisés en fonction de l'utilisateur connecté. Les données elles sont téléchargées après le chargement de la page. Toutes ces données sont récupérées via le module API du serveur. Cela permet de mieux gérer la séparation des rôles.

Le fonctionnement global du serveur n'a pas été modifié car ce n'est pas sur cette partie que portait ce mémoire.

## API

Le module API a subi quelques modifications afin que le service worker puisse télécharger toutes les données dont il a besoin lorsque la connexion au réseau n'est pas disponible. La version de base du serveur ne permettait que de télécharger une partie des données, celle à afficher directement par la page. Par exemple, lorsque l'application doit afficher la page consommateur, l'API ne répond que les 10 éléments doivent être affichés sur la page et une nouvelle requête est nécessaire à chaque fois que l'on veut voir des données différentes (la page 2 du tableau par exemple).

Pour résoudre ce problème, j'ai implémenté des fonctions supplémentaires qui permettent de récupérer en une seule requête toutes les données nécessaires à l'affichage d'une table. Cela permet au service worker de récupérer toutes les données dont-il a besoin afin de les sauvegarder dans l'indexedDB lors de l'installation de celui-ci. Bien sûr ces nouvelles fonctionnalités demandent tout de même à l'utilisateur d'être connecté avec un compte utilisateur valide pour pouvoir récupérer les données.

De plus afin d'aider le mode offline à être plus précis dans l'affichage des données, lorsque qu'une demande d'ajout, de modification ou de suppression de données, le serveur ne vas pas juste répondre avec un status 200 mais il va également renvoyer

la donnée modifiée sous forme de json afin que le service worker puisse mettre à jour directement les données localement.

Une autre problématique est que lorsque que les utilisateurs ne sont plus connectés au réseau internet, il est possible qu'ils modifient en à des moments différent la même donnée. Vu que pour l'instant le travail des gestionnaires de fontaines doit être validé par un gestionnaire plus au placé via le module "Historique" décrit au point 2.2.7, le serveur va simplement accepter les données comme elles viennent et les ajouter dans la liste des données qui doivent être validées.

# Chapitre 6

## Validation

### 6.1 Vérifications automatiques

Tests unitaires

### 6.2 Vérifications utilisateurs réels

Méthodologie

Résultats obtenus

Modifications apportées

# Chapitre 7

## Améliorations futures

### 7.1 Suite du projet

### 7.2 Défis rencontrés

### 7.3 Propositions

# Chapitre 8

## Conclusion

### 8.1 Métriques



# Bibliographie

- [1] dexie.org. dexie.js, 2021. Consulté pour la dernière fois le 24 mai 2021.
- [2] Celine Deknop, Sebastien Sterbelle et Adrien Hallet. Haïtiwater : Développement d’une application web pour gérer la distribution de l’eau en haïti, Juin 2019.
- [3] Microsoft. Xamarin, 2021. Consulté pour la dernière fois le 20 mai 2021.
- [4] Mozilla.org. Indexeddb, 2021. Consulté pour la dernière fois le 24 mai 2021.
- [5] Mozilla.org. Service worker, 2021. Consulté pour la dernière fois le 23 mai 2021.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN

École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | [www.uclouvain.be/epl](http://www.uclouvain.be/epl)