UCLouvain

École polytechnique de Louvain

Développement d'une application Web pour la gestion des données pluviométriques

Auteurs: **Gaëtan Ansotte, Roland Bailly , Dorian Ricci**
Promoteurs: **Kim Mens, Sandra Soares Frazao**
Lecteurs: **Kim Mens, Sandra Soares Frazao, Yves Zech**
Année académique 2018-2019
Master [60] en sciences informatiques

Résumé

Situé en Amérique du Nord, Haïti est souvent victime de différentes catastrophes naturelles : tsunamis, inondations, séismes, etc. C'est pourquoi le Service National des Ressources en Eau (SNRE) d'Haïti est en continuelle reconstruction et peine à se remettre sur pieds.

De nombreux pays se mobilisent afin de mettre en place des projets coopératifs afin de l'aider dans la résolution de ses problèmes. Parmi ceux-ci, un problème important est la gestion des données pluviométriques. Toutes les données pluviométriques sont réparties à travers le pays dans différentes bases de données, sous différents formats et sont difficilement accessible. Ceci rend le travail sur ces données complexe, voire impossible.

Afin de résoudre ce problème, nous avons créé une application web permettant la centralisation et la visualisation des données pluviométriques. Cette application permet d'importer des données pluviométriques sous différents formats, d'afficher ces données sous la forme d'un tableau ou d'un graphique et de les télécharger au format CSV.

Pour parvenir à notre objectif, nous avons commencé par découvrir les bases de l'hydrologie. Ensuite, nous sommes passés à une analyse de l'existant afin de constater ce qui est mis en place dans des pays plus développés. À la suite de cette analyse, nous avons pu faire une première proposition des différentes fonctionnalités réalisables en tenant compte des besoins du client. À partir de cette proposition, nous avons écrit un cahier des charges définissant les différents objectifs à atteindre au terme de la conception de l'application. Ensuite, nous sommes passés par une phase de développement. Durant cette phase, nous avons eu des échanges réguliers avec le client nous permettant d'avoir son feedback. Grâce à ce feedback, nous avons pu améliorer l'application, découvrir comment le client l'utilisait et ainsi améliorer son ergonomie. Chaque échange nous a permis de redéfinir les objectifs hebdomadaires afin de satisfaire le mieux possible la demande du client.

Arrivés au terme de ce travail, nous avons un prototype stable et fonctionnel. Ce prototype permet d'effectuer les fonctionnalités essentielles demandées par le client, à savoir la centralisation, la visualisation et l'importation des données.
L'application est disponible sur un serveur de test à l'adresse suivante :
https://client-d4rk694.c9users.io/
Pour toute personne souhaitant reprendre le projet, celui-ci est disponible à cette adresse :
https://github.com/doricci/TFE4Haiti

Au vu de la charge de travail et des délais donnés, ce mémoire a été réalisé durant le premier quadrimestre de l'année académique 2018-2019 par trois étudiants en sciences informatiques en vue de l'obtention du grade de Master en Sciences informatiques (Master[60]) en janvier 2019. L'Université catholique de Louvain nous a proposé ce mémoire afin de bénéficier au plus tôt d'un prototype stable et exploitable en conditions réelles afin de bénéficier d'un retour sur le projet de la part d'utilisateurs réels. Et qu'il puisse ainsi être proposé dans le cadre d'un nouveau mémoire.

Cette application est une bonne base pour résoudre le problème de gestion des données pluviométriques. Conscient qu'il existe encore de nombreuses améliorations possibles, un effort très important a été mis en place pour que la reprise de l'application soit la plus simple possible, autant par des étudiants que par des informaticiens.

Nous souhaitons tout d'abord remercier particulièrement nos promoteurs Mr Kim Mens et Mme Sandra Soares-Frazão qui nous ont aidés et transmis les connaissances nécessaires à la réalisation de ce mémoire. Nous remercions également Mr Yves Zech pour son expertise apportée sur la partie hydrologique.

Pour terminer, nous remercions nos proches pour leur soutien.

Table des matières

Chapitre 1

Introduction

1.1 Contexte

Situé dans les Caraïbes, Haïti partage l'île d'Hispaniola avec la République dominicaine. Ce pays s'étend sur $27.750 k\text{m}^2$ et a une population d'un peu plus de 10 millions d'habitants parlant le créole et le français.

Haïti est un pays en constante reconstruction. Il a subi de nombreuses catastrophes naturelles comme, par exemple, le tremblement de terre de 2010 ou l'ouragan Matthew de 2016 qui a été la plus violente tempête des Caraïbes depuis des années. La situation politique est quant à elle instable. Cette instabilité réduit considérablement la capacité du gouvernement haïtien et des services publics à prendre des décisions sur le long terme pour répondre aux besoins fondamentaux de la population.

Ces deux facteurs fragilisent fortement l'économie haïtienne, ce qui en fait le pays le plus pauvre d'Amérique. Suite à ce contexte très fragile, de nombreux pays dans le monde se mobilisent en mettant en place des projets de coopération pour soutenir la reconstruction du pays et son développement.

1.2 Problématique

Suite aux différentes catastrophes naturelles et à l'instabilité politique du pays, le Service National des Ressources en Eau de Haïti (SNRE) manque de moyens et tente de se reconstruire. De nombreux projets ont beaucoup de mal à se concrétiser notamment celui de la gestion des données pluviométriques.
À ce jour, il existe un amas de données réparties dans différentes institutions du pays. Ces données sont réparties dans différentes banques de données et sous des formats différents. S'il fallait tout centraliser, cela entraînerait un travail énorme tant au niveau de l'uniformisation des données qu'à la création d'un outil permettant de centraliser celles-ci.

Ce travail est nécessaire, car il permettrait de faire un énorme pas en avant dans la prise de décisions stratégiques et opérationnelles, autant dans le domaine hydrologique que dans le domaine de la prévention des désastres.
De plus, il pourrait être utilisé pour dimensionner un barrage, un réseau d'égouttage, une répartition de l'eau durant les périodes de sécheresse.
Pour finir, il serait aussi d'une grande aide pour protéger l'environnement naturel, restaurer la biodiversité, réduire la vulnérabilité des communautés face aux effets du changement climatique et autres catastrophes naturelles.

En conclusion, cet outil permettrait d'améliorer les moyens de subsistance du pays.

1.3 Motivations

Notre motivation principale concernant ce projet est la création d'un outil *ex nihilo* utile et nécessaire ayant une finalité humanitaire. De plus, la perspective que cet outil soit installé et utilisé pour des applications concrètes ayant pour objectif d'améliorer le niveau de vie global d'un pays est très valorisante.

Une motivation supplémentaire est de pouvoir utiliser nos acquis pédagogiques pour la création d'une application web utilisant les dernières technologies existantes. Nous pouvons donc pu jouir d'une liberté totale en passant par l'architecture de l'application, la représentation des données, etc. Bien que cette liberté semble totale, nous avons dû nous plier à certaines demandes du client que nous expliciterons ci-après.

1.4 Objectifs

Les objectifs du projet sont multiples, mais convergent tous vers l'objectif principal : fournir un outil, à l'état de prototype stable, permettant de centraliser les données pluviométriques existantes en Haïti et de visualiser ces données de manière plus explicite que des données pluviométriques brutes.
Étant donné qu'une analyse hydrologique approfondie doit être effectuée pour certaines fonctionnalités, l'implémentation de celles-ci ne sera donc pas envisageable dans le cadre de ce mémoire. Nous allons donc nous concentrer sur les fonctionnalités primordiales, permettant d'encoder des données et d'effectuer diverses actions sur celles-ci.
Finalement, étant donné que l'application a de fortes chances d'être reprise par d'autres développeurs, nous proposons donc une base solide, permettant ainsi l'ajout de fonctionnalités supplémentaires de manière simple et rapide.

1.5 Démarche

Afin d'atteindre les objectifs fixés, nous avons établi la démarche suivante se divisant en trois étapes principales :

1. Une première phase d'analyse des besoins afin de déterminer les besoins du client, mais aussi de lui montrer ce qu'il est possible de faire.
2. Ensuite, nous avons déterminé les technologies les plus adéquates pour la réalisation de ce projet.
3. Et enfin, nous sommes passés par la phase d'implémentation.

Cette démarche, basée sur la méthode de développement *AGILE*, consiste à avoir des échanges réguliers avec le client sur le produit afin de bénéficier de son feedback. Ceci permet donc d'adapter l'application afin de lui fournir le produit modifié de telle sorte que le produit final soit le plus proche de ses attentes, sachant que celles-ci peuvent évoluer durant la phase de développement.

Concernant la répartition interne des tâches, nous avons utilisé une méthodologie *Kanban*, qui consiste à attribuer un état à chaque tâche : à faire, en cours, terminées. Chaque semaine, nous nous réunissions afin de constater l'avancement du projet et de ses différentes tâches et ainsi de définir celles à effectuer en priorité. Pour se faire, nous avons utilisé l'outil *Trello*.

1.6 Contributions

La contribution majeure est la solution fournie au client. Cette contribution est une application web permettant de centraliser et visualiser les données pluviométriques récoltées depuis des stations provenant de différentes régions d'Haïti.
Les contributions secondaires sont plutôt d'un ordre personnel et pédagogique. En effet durant ce projet, nous avons pu découvrir des techniques de gestion de projet et enrichir nos connaissances concernant les nouvelles technologies.

1.7 Structure du mémoire

Ce mémoire est structuré de la manière suivante :

2. Notions de base en hydrologie
3. Analyse de l'existant qui consiste à analyser des solutions déjà mises en place
4. Établissement du problème afin de comprendre les besoins du client
5. Présentation de l'application
6. Architecture et implémentation
7. Validation et tests
8. Conclusion
9. Améliorations possibles
10. Annexes

Chapitre 2

Notions de base

Avant de poursuivre la lecture de ce document, il est important d'introduire certains termes et notions liés à l'hydrologie.

2.1 Introduction à l'hydrologie

2.1.1 Définition

L'hydrologie est la science de la terre qui s'intéresse au cycle de l'eau, c'est-à-dire aux échanges entre l'atmosphère, la surface terrestre et son sous-sol. Au titre des échanges entre l'atmosphère et la surface terrestre, l'hydrologie s'intéresse aux précipitations (pluie et neige), à la transpiration des végétaux et à l'évaporation directe de la couche terrestre superficielle. [17]

2.1.2 Formation des précipitations

La création d'un nuage démarre par la condensation. Lorsque la température augmente, l'eau s'évapore et gagne en altitude. En montant en altitude, la température et la pression diminuent. L'humidité ne peut plus rester à l'état de vapeur invisible (point de rosée) et se condense principalement en cristaux de glace (une partie peut rester liquide). Et donc, c'est l'amas de ces cristaux qui entraîne la formation du nuage. Une fois formés, sous l'action des vents, les nuages se déplacent. Ces déplacements sont régis par les zones de haute et de basse pression. Enfin, une fois que le nuage est assez chargé de ces cristaux, il devient trop lourd pour supporter cette charge, et les cristaux commencent donc à tomber vers la terre. Durant leur chute, la température augmente et les cristaux passent de l'état solide à l'état liquide (sauf si la température est inférieure à 0°C, alors on obtient de la neige, de la pluie verglaçante, etc.).

2.1.3 Mesure des précipitations

Afin de pouvoir quantifier le volume d'eau précipitée, il existe différents instruments. Dans le cadre de ce mémoire, seuls des pluviomètres sont pris en compte. Nous nous sommes limités à l'étude du fonctionnement de ceux-ci. Nous avons pu observer deux variations de cet équipement :

FIGURE 2.1 – Pluviomètre à lecture directe [28]

— Les pluviomètres à lecture directe (cf. Figure 2.1) : dans le cas d'Haïti, il s'agit le plus souvent d'un récipient gradué qui sera relevé à intervalles réguliers. Ce type de pluviomètre permettra de mesurer le volume d'eau précipitée sur une surface donnée en un temps donné. Bien sûr, il est important de protéger le récipient contre l'évaporation et d'éviter toute erreur ou maladresse de manipulation. Il est donc moins fiable.

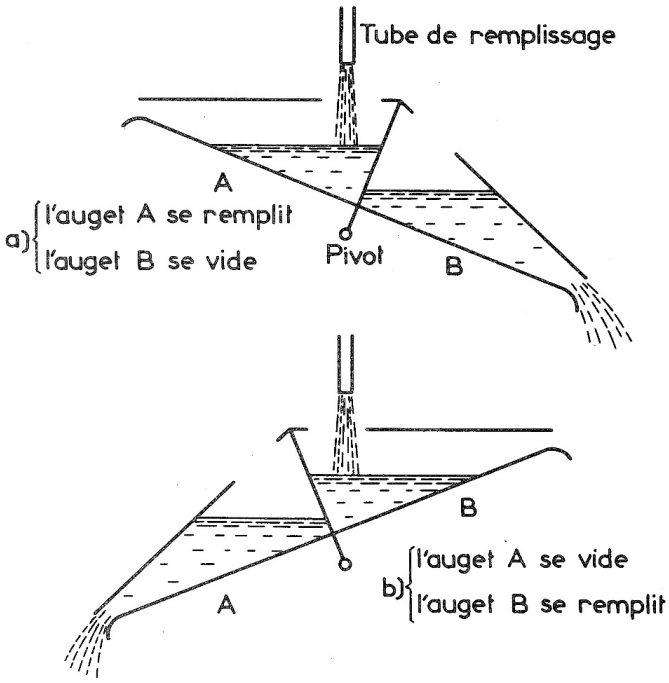


FIGURE 2.2 – Représentation d'un pluviomètre à augets basculeurs [29]

— Les pluviomètres à augets basculeurs (cf. Figure 2.2) : dans ce cas, nous ne relevons pas un volume d'eau, mais le nombre de basculements d'un volume donné d'eau fixe sur un temps donné. Ces basculements sont enregistrés de manière mécanique, optique

ou électronique. Lorsqu'il pleut, l'auget supérieur se remplit d'eau jusqu'à atteindre une certaine quantité. Une fois cette quantité atteinte, la partie mobile bascule et se dévide, ainsi l'autre partie du mécanisme peut à son tour se remplir jusqu'à ce que le basculement s'effectue dans l'autre sens. La multiplication du volume par le nombre de basculements permet de savoir la quantité d'eau tombée. Typiquement, le volume de basculement est calibré pour correspondre à 0.10 mm de pluie.

Le pluviomètre à lecture directe sera davantage utilisé dans les villages les plus reculés où les moyens sont les plus faibles tandis que les pluviomètres à augets basculeurs auront tendance à être placés dans des zones critiques (un bassin versant, une zone souvent inondée, etc.). Il est d'usage d'exprimer les précipitations cumulées en millimètres, ce qui correspond à un litre d'eau par mètre carré, et les précipitations par unité de temps (l'intensité de pluie) en mm/h. Ce sont ces unités qui seront utilisées dans la suite du document.

2.1.4 Représentation et interprétation des mesures

Maintenant que nous avons introduit les outils, nous les appellerons *stations pluviométriques* ou simplement *stations* dans la suite du document. Ces stations pluviométriques permettent de quantifier les volumes d'eau. Nous allons pouvoir passer à la représentation de ces mesures.

La représentation la plus courante est une représentation via un tableau (cf. Figure 2.3).

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
00	0	0	0	0	1.2	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
05	0	0	0	0	1	0	0	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	1	0	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0.4	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0.4	0.2	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
45	0	0	0	2	0	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50	0	0	0	1.4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
55	0	0	0	0.6	0	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Récapitulatif

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Moyenne	-	-	-	0.37	0.33	0.08	0.03	0.02	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Min (heure)	0 00:00	0 01:00	0 02:00	0 03:00	0 04:20	0 05:05	0 06:00	0 07:00	0 08:00	0 09:00	0 10:00	0 11:00	0 12:00	0 13:00	0 14:00	0 15:00	0 16:00	0 17:00	0 18:00	0 19:00	0 20:00	0 21:00	0 22:00	0 23:00
Max (heure)	0 00:00	0 01:00	0 02:00	2 03:45	1.2 04:00	0.2 05:00	0.2 06:10	0.2 07:05	0 08:00	0 09:00	0 10:00	0 11:00	0 12:00	0 13:00	0 14:00	0 15:00	0 16:00	0 17:00	0 18:00	0 19:00	0 20:00	0 21:00	0 22:00	0 23:00

Minimum absolu : 0 Maximum absolu : 0.2 Somme : 10.0 Nombre de valeurs : 288

FIGURE 2.3 – Représentation des données sous forme d'un tableau (tiré de notre application)

Cette représentation a l'avantage de pouvoir rapidement comparer les données entre elles. Un tableau récapitulatif peut faciliter la lecture. Malheureusement, cette représentation peut rapidement devenir difficile à interpréter en fonction de la quantité des données.

C'est pourquoi une autre représentation possible est une représentation sous forme de graphique en bâtonnets (cf. Figure 2.4). Celle-ci a l'avantage de pouvoir rapidement visualiser les tendances et les épisodes extrêmes et donc de voir si certaines données semblent incohérentes.

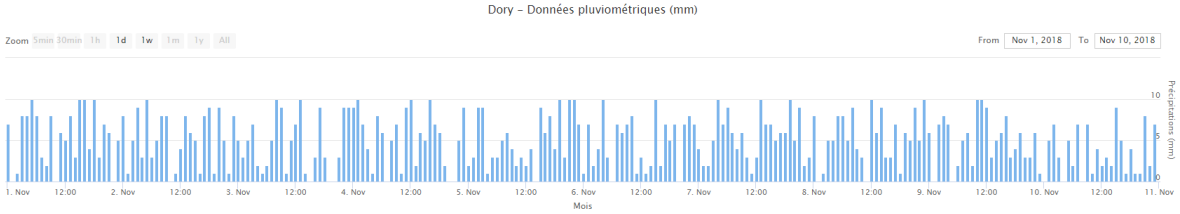


FIGURE 2.4 – Représentation des données sous forme d'un graphique en bâtonnets (tiré de notre application)

Une autre représentation possible est la représentation sous forme de tableaux ou de graphiques "Intensité-Durée-Fréquence (ISM)" (cf. Figure 2.5), qui indique, pour différents temps de retour (les fréquences) l'intensité moyenne maximum de la pluie (en mm/h) en fonction de la durée de celle-ci (par exemple en minutes). Les différents épisodes de pluie observés pendant un laps de temps donné (de préférence supérieur à 20 ans) sont analysés et représentés par des points qui, par régression, sont reliés par des courbes. Comme le montre la figure 2.5 (où les courbes sont simplifiées en segments de droite dans un diagramme *log-log*), la plus grande intensité moyenne que l'on puisse observer sur une durée donnée diminue en fonction de cette durée et augmente pour un temps de récurrence plus grand. Ces données sont très utilisées pour le dimensionnement d'ouvrage, par exemple les collecteurs d'égouts, mais cette représentation ne sera pas abordée dans les développements qui suivent.

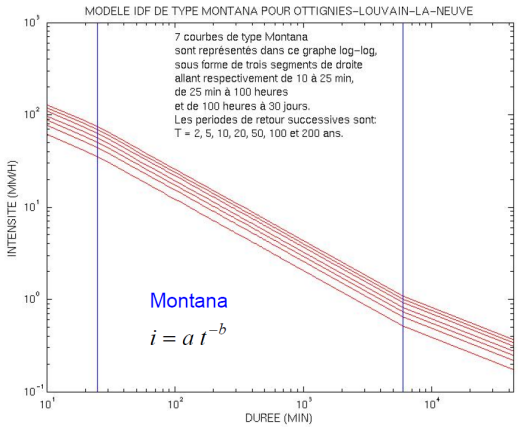


FIGURE 2.5 – Représentation des données sous forme d'un graphique ISM selon la formule de Montana [31]

Il est aussi possible de représenter les pluies sur une carte via l'utilisation d'isohyètes (cf. Figure 2.6). Les isohyètes sont des lignes imaginaires dessinées sur des cartes reliant des points d'égales quantités de précipitations tombées en une période déterminée. Il existe différents moyens de calculer ces courbes.

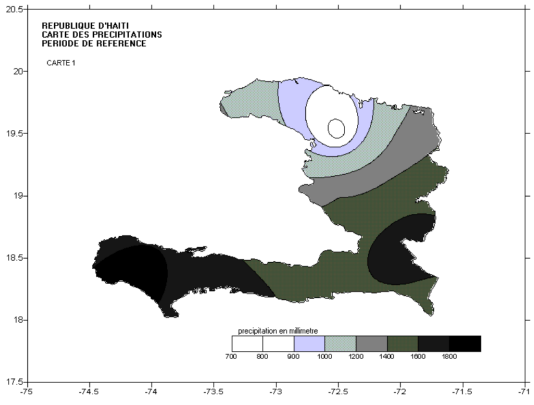


FIGURE 2.6 – Représentation des précipitations via l'utilisation d'isohyètes [30]

L'application qui sera réalisée dans le cadre de ce mémoire permet, pour le moment, de donner des représentations sous forme de tableaux, de tableaux récapitulatifs et de graphiques en bâtonnets. Bien sûr, les autres représentations sont envisageables dans le futur de l'application.

Chapitre 3

Analyse de l'existant

Afin de comprendre la demande du client, celui-ci nous a fourni un exemple d'application web belge répondant à une partie des besoins. Nous avons pu analyser cette application web, comprendre ses qualités et ses défauts afin de nous en inspirer.

3.1 Direction générale opérationnelle de la Mobilité et des Voies hydrauliques

Cette application web belge[1] appartient au service public de la Wallonie : Direction générale opérationnelle de la Mobilité et des Voies hydrauliques

La première interface de l'application est la carte :

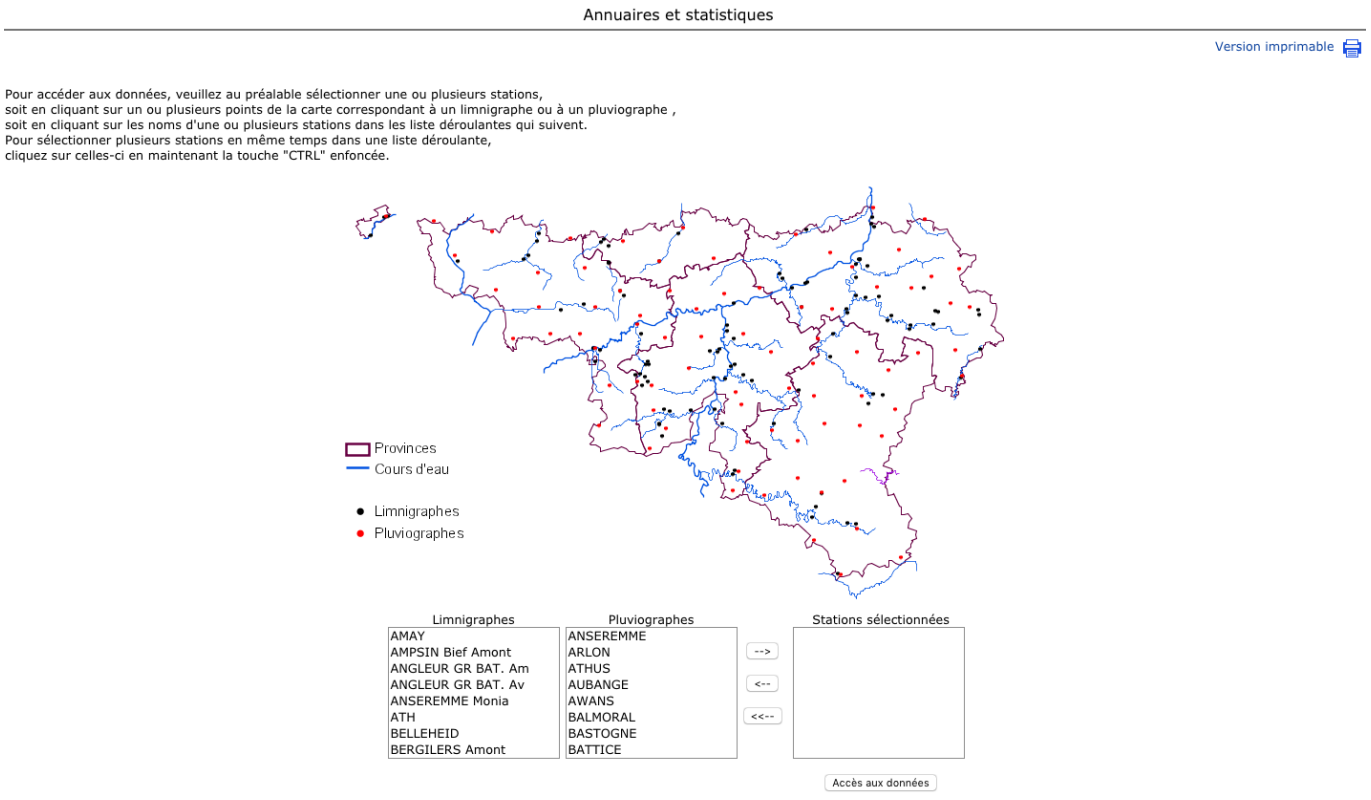


FIGURE 3.1 – Interface carte de l'application de la Wallonie [1]

Elle comporte :

- Un bouton permettant d'imprimer la carte
- Une explication pour l'utilisation de la carte
- La carte interactive qui permet de sélectionner une station et d'accéder à ses informations par la suite
- Une légende pour la carte
- Une liste qui permet de parcourir les stations disponibles et de les sélectionner pour accéder aux données
- Un bouton qui permet d'accéder aux données pluviométriques des stations sélectionnées

Cette interface est le moyen principal pour accéder aux données pluviométriques d'une station.

Pour accéder aux données des stations, il faut les sélectionner et appuyer sur le bouton "Accès aux données". On arrive alors sur l'interface suivante :

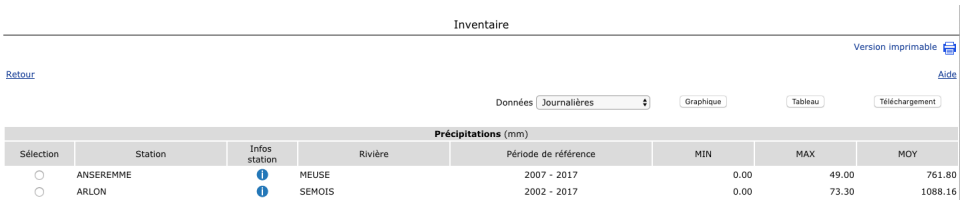


FIGURE 3.2 – Inventaire des stations sélectionnées [1]

Elle comporte :

- Des boutons permettant d'avoir la possibilité d'afficher des données de manière journalière, mensuelle, etc. via une interface de type graphique ou tableau
- Un bouton permettant de télécharger les données
- Un tableau permettant d'avoir une vue globale des stations sélectionnées, des informations relatives à la celles-ci et un bouton permettant d'avoir accès à plus d'informations

On remarque que cette interface, qui est principalement composée d'une liste de stations sélectionnées, ne permet pas simplement d'afficher des données, mais d'accéder aussi à de nouvelles interfaces : description, graphique et tableau.
Commençons par analyser l'interface description de la station :

FIGURE 3.3 – Informations d'une station [1]

Cette interface comporte des informations propres à la station ainsi qu'une photo de celle-ci.

Analysons l'interface graphique des données :

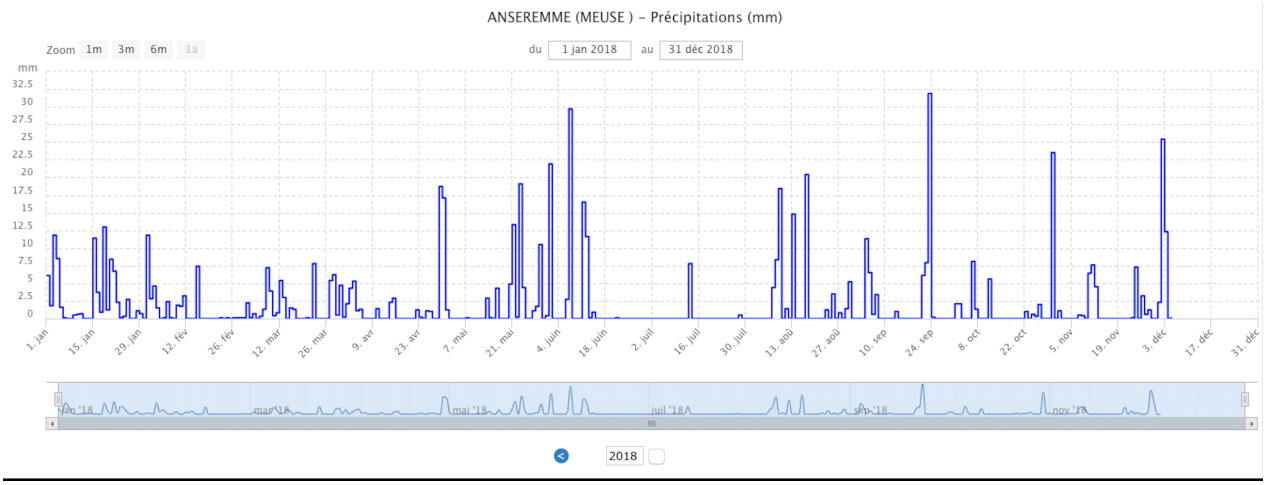


FIGURE 3.4 – Graphique des données d'une station [1]

Cette interface comporte :

- Un titre composé du nom de la station, sa rivière et l'unité
- Le graphique qui affiche les données selon une date choisie et permettant de :
  - changer l'intervalle d'affichage des données (1mois, 3mois, 6mois, etc.)
  - modifier la date à afficher
  - la possibilité de changer la période à afficher

Finissons par l'interface tableau des données :

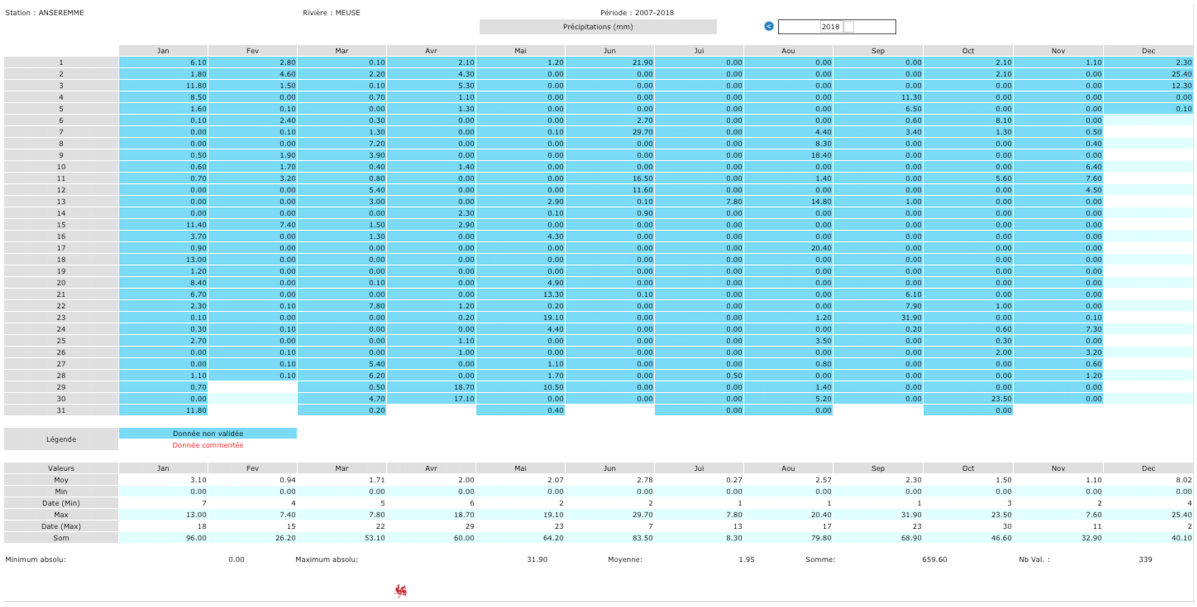


FIGURE 3.5 – Tableau des données d'une station [1]

Cette interface comporte :

- L'ensemble des données pluviométriques
- Des données récapitulatives comme le maximum, le minimum, etc.

Pour finir nous avons défini les points forts de l'application web :

- Le système pour rechercher les stations est facile d'utilisation et intéressant
- L'affichage des données en graphique et en tableau semble être le meilleur moyen pour fournir des données claires et précises
- Sur le graphique, une ligne du temps interactive permet à l'utilisateur de changer la période d'affichage
- Pour le tableau, en plus des données pluviométriques, un récapitulatif est calculé

Malheureusement, une question reste en suspens :
"Comment ont-ils rajouté les stations et les données, le tout de manière facile et rapide pour les utilisateurs ?"
Cette partie sera abordée dans le chapitre suivant : établissement du problème.

3.2 Données fournies

Un ensemble de fichiers *Excel* nous a été fourni. Celui-ci contient les données sur lesquelles nous nous sommes basés afin de définir les deux formats de données que notre application acceptera. Les données sont issues du stage de Saint Fleur Bob E., ont été fournies par notre promotrice Mme Soares Frazao, retravaillées par Monsieur Y. Zech et se concentrent principalement sur les stations présentes autour du bassin versant de Cavaillon.

Chapitre 4

Établissement du problème

Ce chapitre établit le problème à solutionner grâce à l'application web. Avant la conception de l'application web, nous avons dû établir un cahier des charges. Celui-ci permet de s'assurer de la bonne compréhension du problème entre le client et les développeurs. Ce document doit également être respecté durant la conception d'un projet.

4.1 Cahier des charges

4.1.1 Description

Étant donné sa situation géographique et économique sensible, Haïti a une situation plus que précaire quant aux différents phénomènes naturels pouvant arriver. Entre les cyclones, les menaces de tsunami, les ouragans [, etc], les inondations et les sécheresses, la *perle des Antilles* est dans une position très délicate.
C'est dans ces circonstances que nous allons tenter de créer une application web permettant de mieux répondre aux deux dernières menaces citées précédemment. Cet outil a pour but de centraliser toutes les informations relatives aux données pluviométriques et proposerait une interface claire et simple d'utilisation permettant de visualiser celles-ci.
Cette application web permettrait de centraliser, visualiser et simplifier la récupération des données pluviométriques.

4.1.2 Objectifs

L'objectif du document actuel est de bien définir les limites du travail à effectuer. L'outil à développer a pour objectif principal de centraliser des données pluviométriques existantes ainsi que de créer une plate-forme permettant d'en ajouter de nouvelles afin de les visualiser.
Nous supposons que les utilisateurs de l'application web sont soit un **Visiteur**, c'est-à-dire une personne souhaitant simplement visualiser les données à travers l'application. Soit un **Utilisateur**, c'est-à-dire une personne travaillant dans le domaine météorologique en Haïti (ou ailleurs), autant pour consulter les données que pour en importer de nouvelles. Ou enfin un **Administrateur** s'occupant de la gestion générale de l'application (gestion des utilisateurs, gestion des **Station**s, validation des actions, etc.).

Étant donné le volume de travail à effectuer, nous ne serons pas capables de faire un outil couvrant tous les besoins. Il sera donc à la charge de Haïti de poursuivre ce que nous avons commencé. Toutefois, nous fournirons une base solide, pouvant être facilement reprise et améliorée, de par sa documentation complète, une architecture robuste et l'utilisation d'un nombre réduit de langages informatiques, accessibles, simples, et ayant une communauté active.

4.1.3 Approche

Pour se faire, nous allons nous baser sur des données existantes issues principalement du bassin versant de Cavaillon provenant du stage de Saint Fleur Bob E. fournies par notre promotrice Mme Soares Frazao et retravaillées par Monsieur Y. Zech. L'intégralité des données reçues est reprise dans différents fichiers au format Excel. Ces données pourront être enregistrées afin de les rendre accessibles de telle sorte que l'utilisateur puisse avoir un aperçu simple et concis de celles-ci (nous reviendrons sur les fonctionnalités dans la partie suivante).

4.1.4 Fonctionnalités

Dans cette section, nous allons tenter de définir les limites du travail à effectuer. Nous avons divisé les fonctionnalités en trois parties :

1. Basique : ce que nous rendrons au minimum dans le produit final
2. Intermédiaire : la majorité de ces fonctionnalités seront implémentées
3. Avancé : s'il nous reste du temps, nous piocherons dans ces fonctionnalités (certaines de ces fonctionnalités nécessitent le travail d'un externe, tels un hydrologue, un géographe, etc.).

Fonctionnalités basiques

— Afficher une carte avec l'ensemble des stations ainsi qu'un système de recherche et de sélection des stations

— Afficher les données d'une station sélectionnée sous forme de

  1. Tableau représentant les données pluviométriques sur différents intervalles de temps : horaire, journalier, mensuel, annuel.
  2. Graphique en courbes et en bâtonnets des précipitations

— Filtrer les données d'une station sur base de :

  1. Date
  2. Durée (intervalle entre deux dates)

— Demande de création d'un compte pour un nouvel utilisateur qui sera validée par un administrateur

— Modification du mot de passe d'un utilisateur (dans le cas où celui-ci l'aurait oublié, ce processus sera sécurisé par l'envoi d'un mail)

— Demande de création d'une station par un utilisateur connecté

— Dans le panneau d'administration : valider des utilisateurs

— Dans le panneau d'administration : valider une demande d'ajout de données

— Importer les données via un fichier CSV (format facilement obtenu via les logiciels de tableur (p. ex. Excel, éditeur de texte))

— Importer une donnée manuellement

Fonctionnalités intermédiaires

— Validation d'une station par un administrateur

— Modification d'une station par un administrateur

— Suppression d'une station par un administrateur

— Export de données pluviométriques au format CSV

— Ajout d'une note sur une station

— Feedback via un mail lorsque l'administrateur refuse un utilisateur

Fonctionnalités avancées

— Export des données sous différents formats

— Comparaison des données entre plusieurs stations

— Graphiques Tablot et de Montana

— Affichage d'une HeatMap des données pluviométriques.

4.1.5 User stories

Les *user stories* permettent de décrire en une phrase une fonctionnalité en précisant le *"qui"*, *"quoi"* et le *"pourquoi"*.

Visiteur

1. En tant que visiteur, je souhaite voir l'emplacement de l'ensemble des stations météorologiques afin d'avoir un aperçu rapide de leur répartition.
2. En tant que visiteur, je souhaite voir les données pluviométriques récoltées par une station météorologique, sous forme de :
   — Tableau (horaire, journalier, mensuel, annuel)
   — Graphique (sous forme de graphique en courbes avec ligne du temps)

   afin d'avoir un aperçu plus simple que des données brutes.
3. En tant que visiteur, je souhaite filtrer les données pluviométriques récoltées par une station météorologique selon :
   — Une période de temps (année, mois, jour, etc.)
   — Triées dynamiquement selon l'intitulé de la colonne (via un clic)

   afin d'affiner ma recherche.

Utilisateur

1. En tant qu'utilisateur, je souhaite pouvoir ajouter une nouvelle station afin de lui ajouter des données.
2. En tant qu'utilisateur, je souhaite pouvoir supprimer ma station afin de notifier qu'elle n'existe plus.
3. En tant qu'utilisateur, je souhaite pouvoir modifier les informations de ma station afin de corriger une erreur ou ajuster des résultats erronés.
4. En tant qu'utilisateur, je souhaite pouvoir importer des données pour ma station selon un modèle prédéfini afin d'envoyer un ensemble de données sans devoir les encoder une par une.
5. En tant qu'utilisateur, je souhaite pouvoir ajouter une donnée pour ma station manuellement afin de ne pas devoir créer un fichier pour envoyer une donnée.
6. En tant qu'utilisateur, je souhaite pouvoir éditer une donnée pour ma station manuellement afin de corriger une erreur ou ajuster des résultats erronés.
7. En tant qu'utilisateur, je souhaite pouvoir télécharger les données de ma station afin d'en faire une analyse.

Administrateur

1. En tant qu'administrateur, je souhaite pouvoir ajouter, supprimer et modifier les informations et données d'une station afin de corriger/supprimer ce qu'un utilisateur a encodé.
2. En tant qu'administrateur, je souhaite pouvoir ajouter, supprimer et modifier les informations d'un utilisateur afin de rectifier des données erronées.
3. En tant qu'administrateur, je souhaite pouvoir accepter ou refuser l'ajout d'une station par un utilisateur (qui n'est pas un administrateur) afin de contrôler les stations ajoutées.
4. En tant qu'administrateur, je souhaite pouvoir accepter ou refuser l'ajout d'une action d'un utilisateur (ajouter une donnée, modifier une donnée, supprimer une station, renommer une station, etc.) (qui n'est pas un administrateur) afin de vérifier et réguler les modifications effectuées.

4.1.6 Maquette

Cette section a pour but de vous donner un aperçu de ce à quoi pourrait ressembler l'application web. Ces maquettes permettent de partager la vision de l'application entre les développeurs sur les fonctionnalités à développer de telle sorte que le client ait un support visuel et puisse donner son accord avant de passer à la phase de développement.

Page d'accueil

La page d'accueil (cf. Figure 4.1) est divisée en trois parties principales :

— Le panneau de gauche : recherche parmi les stations

— La carte centrale : affichage des stations sur une carte interactive type Google Maps

— La partie basse : un tableau affichant les données relatives aux stations sélectionnées

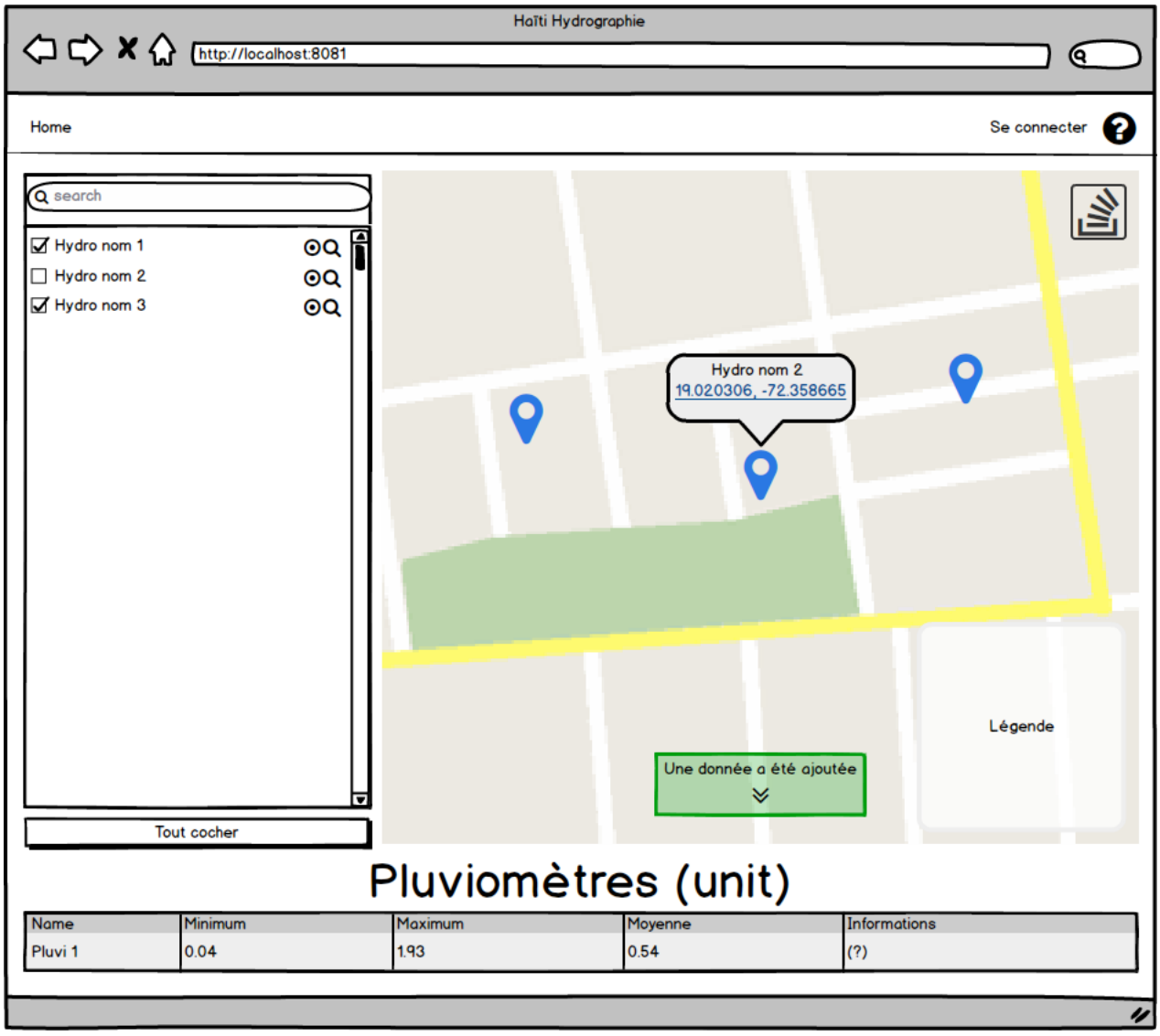


FIGURE 4.1 – Page d'accueil

Connexion/Inscription

La page de connexion (cf. Figure 4.2) est divisée en deux parties : à gauche, il est possible de se connecter et de récupérer son mot de passe. À droite, il est possible de faire une demande de création d'un compte. Voici comment se déroule la procédure de création d'un compte :

1. Une personne doit effectuer une demande de compte en remplissant le formulaire ci-dessous.
2. La demande est envoyée à l'administrateur qui devra accepter ou refuser celle-ci via sa page d'administration (cf. Figure 4.12).
3. Si l'administrateur valide le nouvel utilisateur, celui-ci reçoit un mail de confirmation afin qu'il puisse définir son mot de passe via l'interface "*Changer le mot de passe/Créer un mot de passe*" (cf. Figure 4.4).

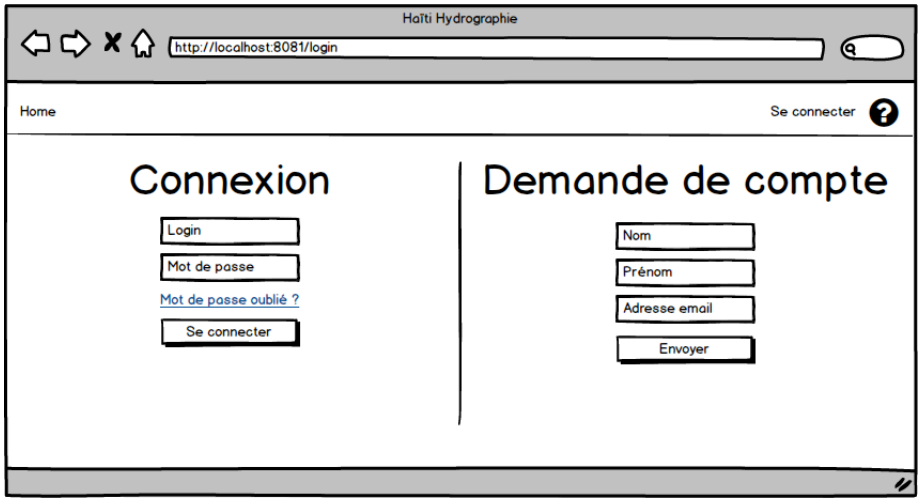


FIGURE 4.2 – Interface pour se connecter ou créer un compte

Mot de passe perdu

Il s'agit de l'interface sur laquelle l'utilisateur arrivera s'il a perdu son mot de passe (cf. Figure 4.3). Pour des raisons de sécurité, nous n'affichons pas d'erreur si l'adresse mail est inconnue. Toutefois, si cette dernière est connue, nous envoyons un mail à l'utilisateur lui permettant de changer de mot de passe. Le lien de modification du mot de passe sera valide pour une durée précise.

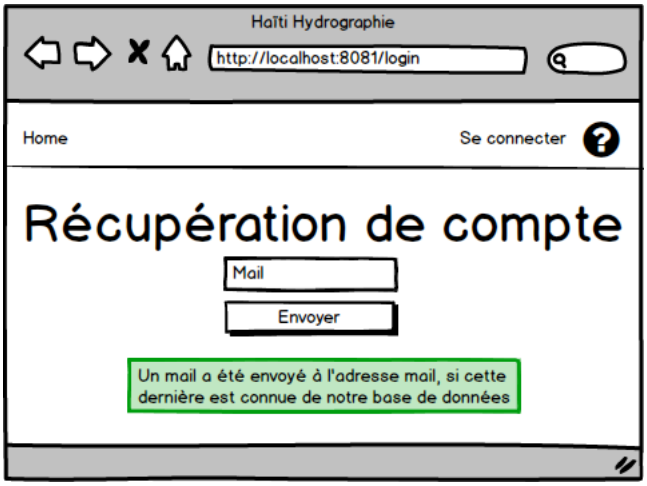


FIGURE 4.3 – Interface pour récupérer un compte

Changer le mot de passe/Créer un mot de passe

S'il s'agit d'un utilisateur ayant été accepté par un administrateur, il recevra un mail lui permettant d'arriver sur cette page qui prendra le titre *"Créer un mot de passe"*. Sinon, il s'agit d'un utilisateur ayant oublié son mot de passe, le titre sera alors *"Changer de mot de passe"*. La figure 4.4 représente l'interface qui permet de modifier le mot de passe.

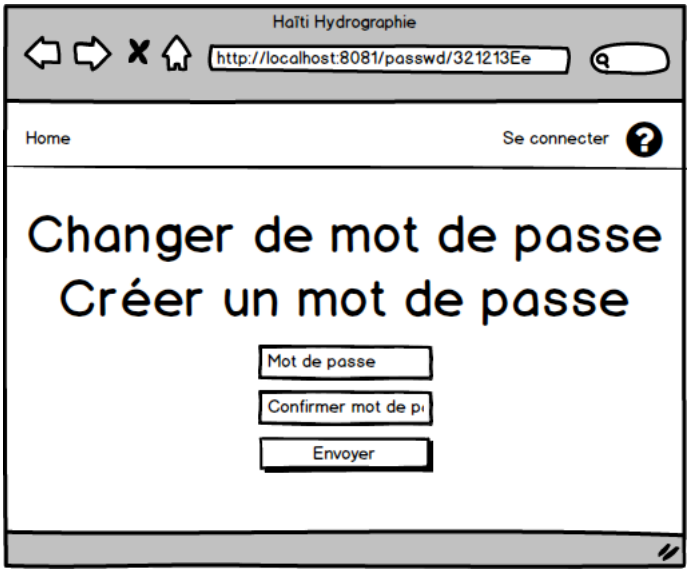


FIGURE 4.4 – Interface pour modifier le mot de passe

Gestion des stations

Afin de pouvoir accéder à cette interface (cf. Figure 4.5), l'utilisateur ayant les droits requis doit être identifié (i.e. connecté).
Il peut soit modifier une de ses stations, importer des données ou supprimer une station (il ne sera plus possible d'importer des données pour celle-ci, de plus nous conserverons les données relatives à celle-ci), soit ajouter une nouvelle station.

Mes stations

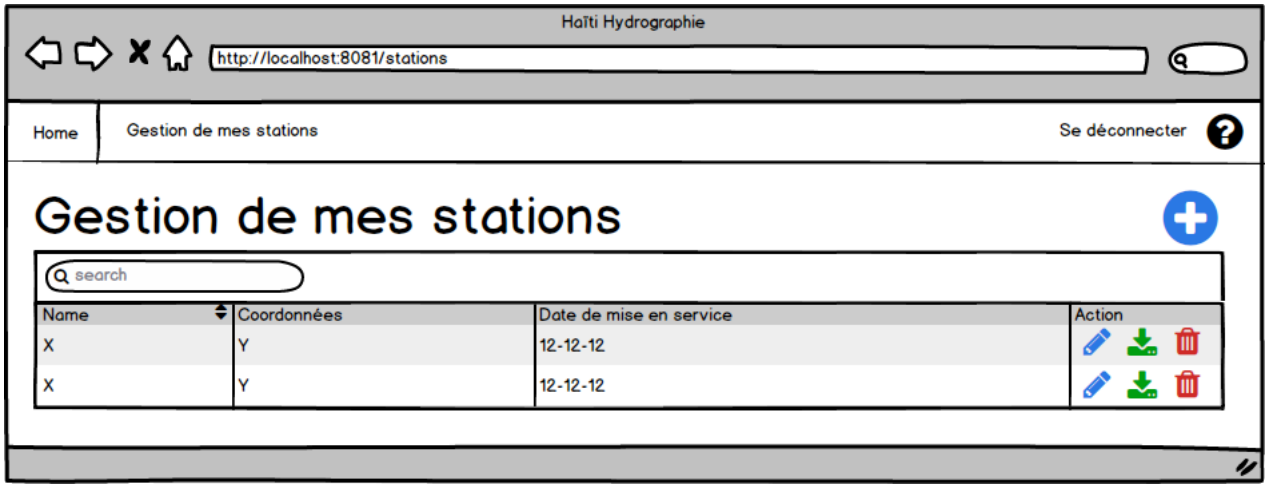


FIGURE 4.5 – Interface "Mes stations"

Ajout
En cliquant sur le bouton bleu '(+)', une fenêtre s'ouvre avec le formulaire de création contenant toutes les informations reprises sur la figure 4.6.
L'intervalle représente l'intervalle de temps entre chaque capture d'une donnée.
Attention à bien prendre l'intervalle le plus petit et existant possible, car ceci affectera directement les filtres des tableaux et graphiques. Pour définir celui-ci, nous nous sommes basés sur l'avis du professeur émérite Yves Zech de l'Université catholique de Louvain. De plus, ceci affectera aussi le format du fichier permettant l'import des données.
Cette station devra être acceptée par un administrateur.

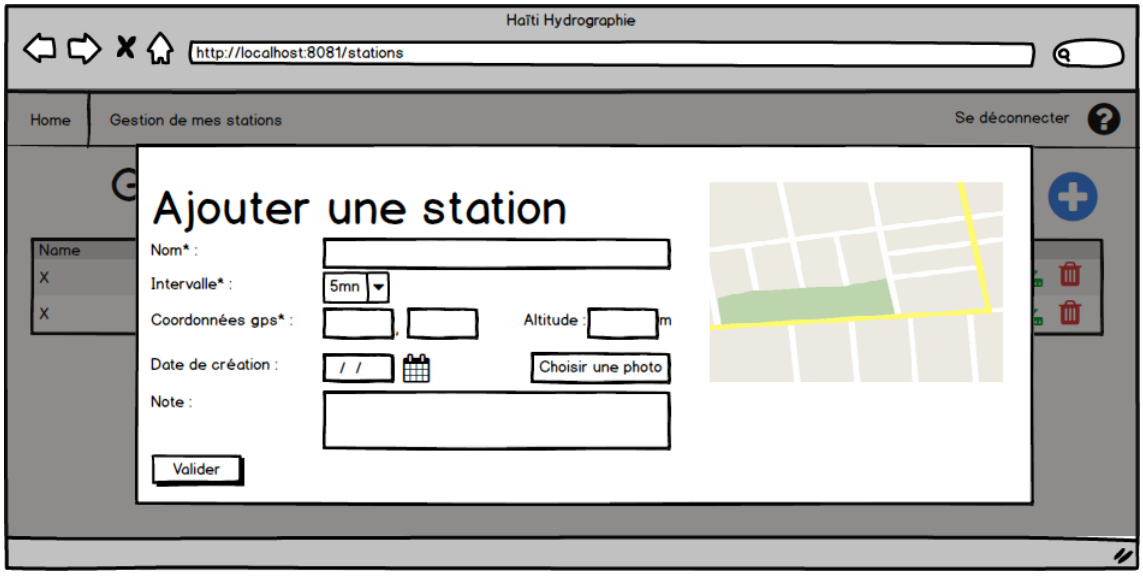


FIGURE 4.6 – Interface pour ajouter une station

Import des données
Cette interface (cf. Figure 4.7) permet d'encoder une donnée ou d'importer un ensemble de données (fichier CSV) pour une station. Une seule de ces deux actions peut être effectuée à la fois. La donnée est toujours composée d'une date, d'une heure et d'une valeur.

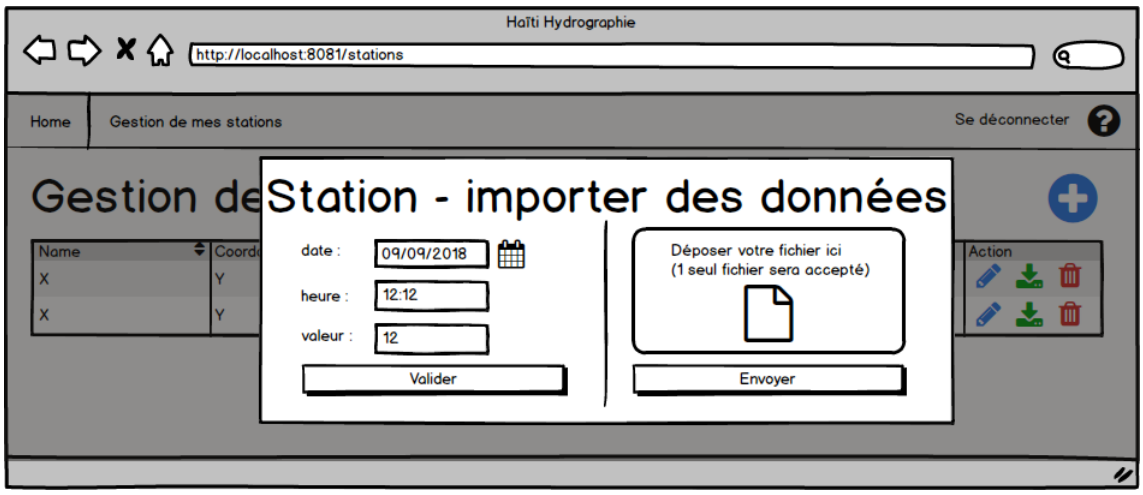


FIGURE 4.7 – Interface pour importer des données

Détail station

Cette interface (cf. Figure 4.8) affiche les détails d'une station (accessible via la page d'accueil). Il est possible de voir la localisation de la station, visualiser une photo de celle-ci (si disponible), afficher ses données sous forme d'un tableau ou d'un graphique en courbes, ainsi qu'effectuer différents filtres sur chacun de ces types de représentation.

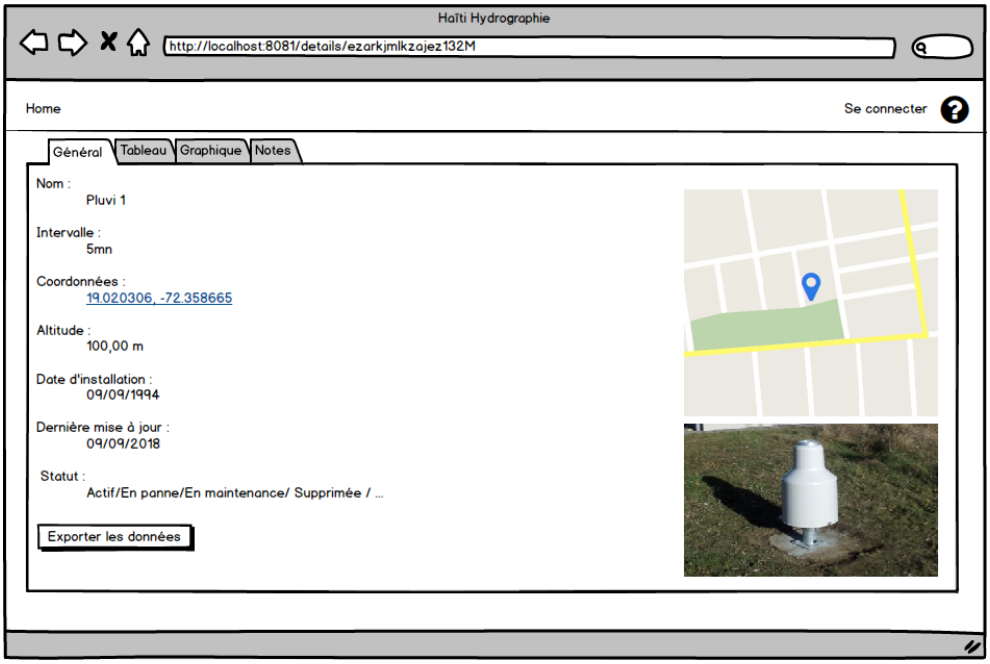


FIGURE 4.8 – Interface du détail d'une station

Tableau

Lorsque l'utilisateur (connecté et ayant les droits sur la station concernée) est dans l'interface tableau (cf. Figure 4.9) affichant les données minute par minute, il peut également modifier les données de la station et afficher les données selon le filtre choisi.

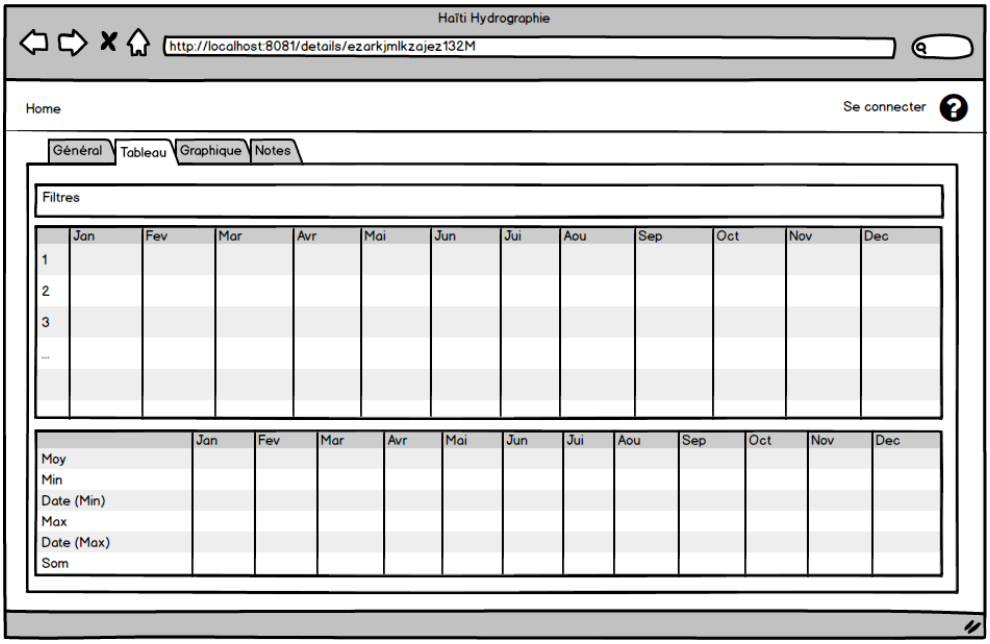


FIGURE 4.9 – Interface d'affichage des données via un tableau

Graphique en courbes
L'interface graphique (cf. Figure 4.10) représente les données pluviométriques. En abscisse sera exprimé le temps selon le format sélectionné par l'utilisateur. Et en ordonnée sera représentée la hauteur de pluie tombée exprimée en mm. Celle-ci sera toujours comprise entre le maximum et le minimum absolu pour la période donnée.

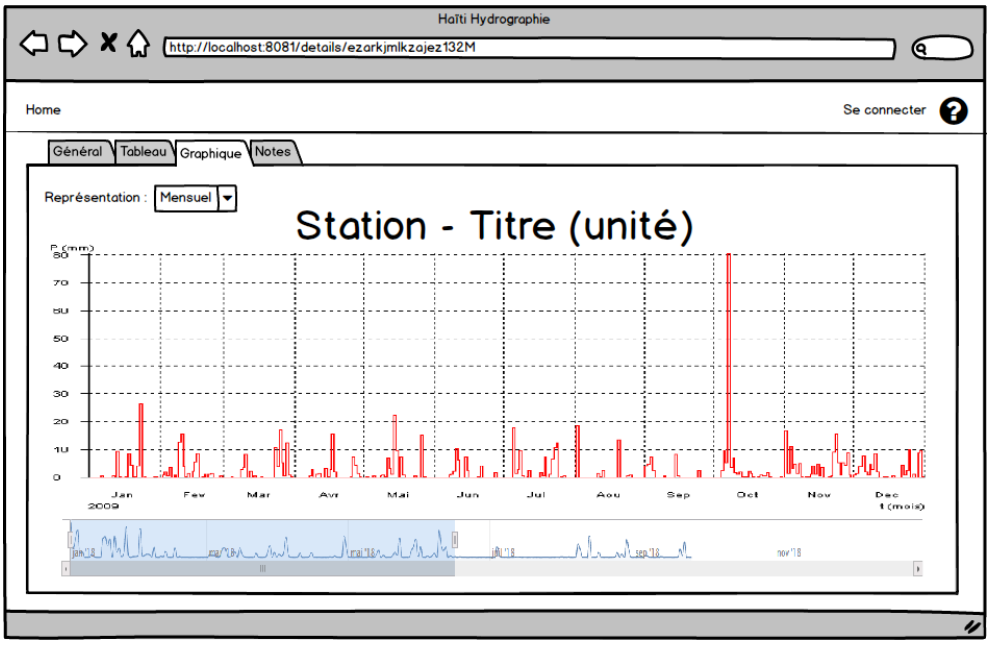


FIGURE 4.10 – Interface d'affichage des données via un graphique

Notes sur une station
Sur la figure 4.11, l'utilisateur ayant accès à cette station pourra ajouter une note sur la station pour préciser des informations spécifiques celle-ci, communiquer avec d'autres utilisateurs, signaler qu'un fichier a été envoyé, etc. Il pourra de plus visualiser toutes les notes liées à la station qui seront classées par ordre antichronologique.

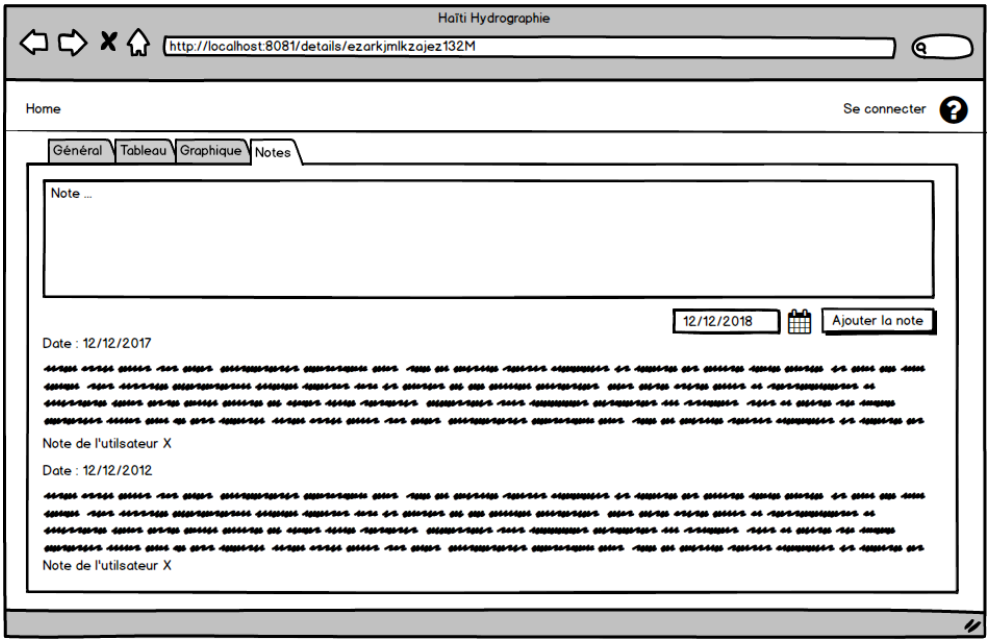


FIGURE 4.11 – Interface des notes d'une station

Tableau d'administration

L'interface tableau d'administration (cf. Figure 4.12) permet à l'administrateur d'accepter ou de refuser les stations/utilisateurs en attente de validation. De plus, l'administrateur peut visualiser ici l'ensemble des modifications effectuées par les utilisateurs et les accepter ou les refuser.

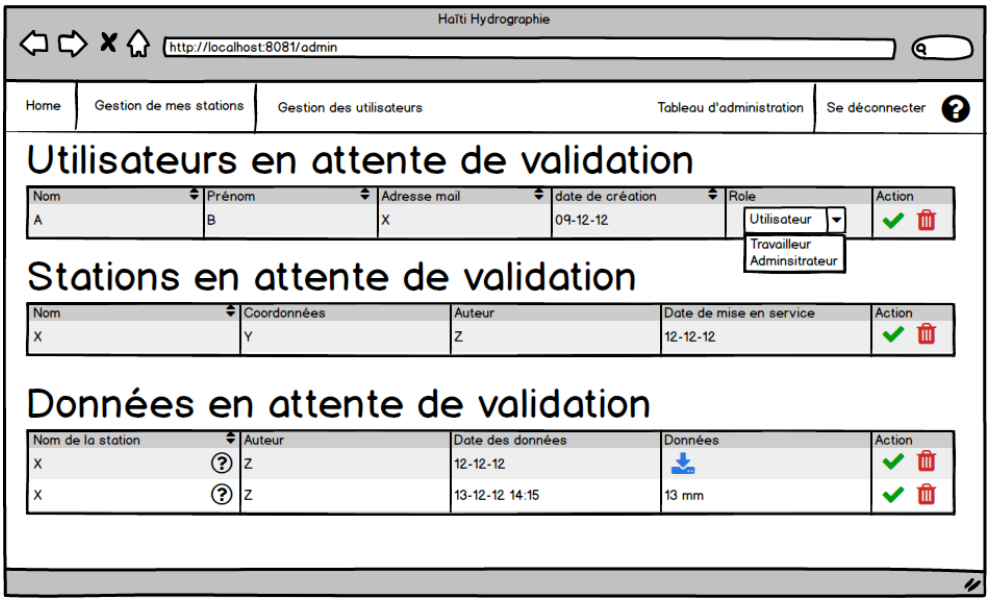


FIGURE 4.12 – Interface tableau d'administration

Lorsque l'administrateur décide de refuser un utilisateur, une modification, etc., un mail sera envoyé à l'utilisateur concerné avec la note que l'administrateur aura écrite. La figure 4.13 représente l'interface permettant à un administrateur de refuser un utilisateur.

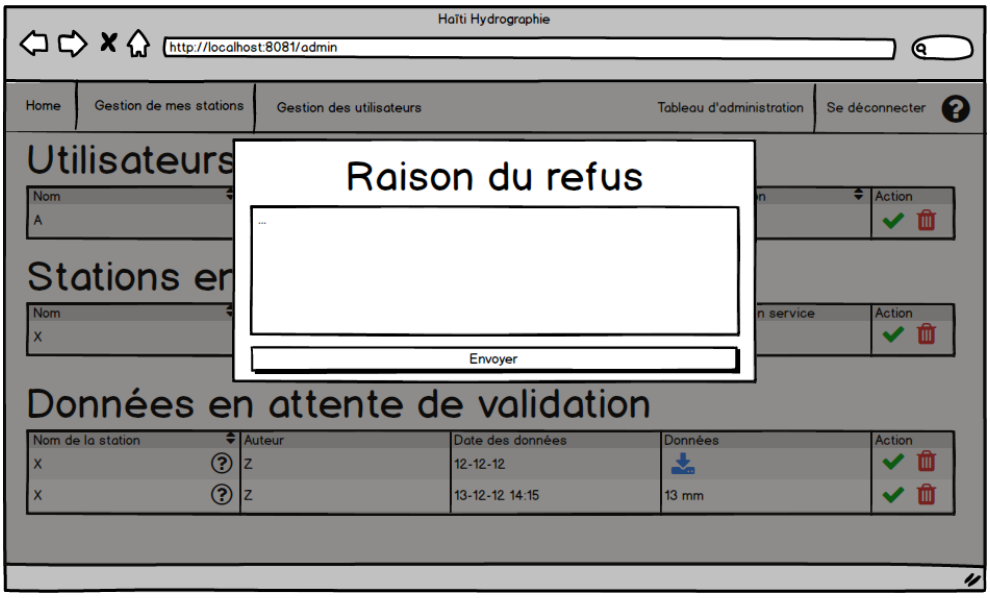


FIGURE 4.13 – Interface permettant à un administrateur de refuser un utilisateur

Liste des utilisateurs

L' interface de la liste des utilisateurs (cf. Figure 4.14) permet à l'administrateur de visualiser tous les utilisateurs afin de les modifier, pour, par exemple, changer leur rôle.

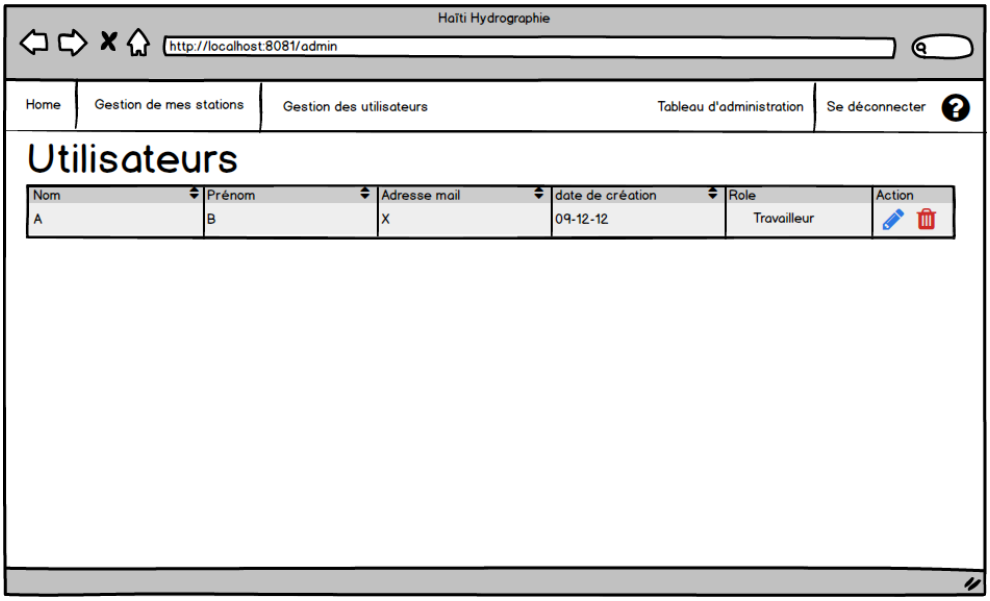


FIGURE 4.14 – Interface de la liste des utilisateurs

4.1.7 Formats

Afin de généraliser le format des données, nous avons décidé de supporter automatiquement l'import/l'export des données en suivant une structure de fichier simple dans un premier temps (le CSV).

4.2 Technologies envisagées

Afin de concevoir l'application web, nous avons analysé des technologies populaires, récentes, d'avenir, faciles à prendre en main et ayant une communauté active. Nous allons présenter ces **Frameworks** orientés web fondés sur différents langages et analyser leurs qualités.

4.2.1 Django

Ce framework est fondé sur le langage Python. Ce langage est souvent utilisé afin d'effectuer de petites tâches. Il est aussi très utilisé dans le monde scientifique pour effectuer des calculs lourds et complexes grâce à des bibliothèques spécialisées. Voici quelques caractéristiques :

- Conçu sur le modèle MTV Modèle-Template-Vue
- ORM (Object Relational Mapper) ayant une bonne performance et facile d'utilisation pour l'abstraction des objets en base de données.
- Possibilité d'une migration base de données.

4.2.2 Ruby on Rails

"*RoR est le Framework MVC libre le plus populaire basé sur le langage Ruby. Ce Framework a été conçu pour développer des applications web plus rapidement. Il permet aux développeurs de créer des fonctionnalités avec moins de code que d'autres frameworks.*"[2]

Voici quelques caractéristiques :

- Conçu sur le modèle MVC Modèle-Vue-Contrôleur
- ORM Active Record pour l'abstraction des objets en base de données
- Compatible avec la plupart des bases de données relationnelles et non relationnelles

4.2.3 J2EE

Ce framework est fondé sur le langage JAVA. Il est utilisé par de nombreuses entreprises. Il est connu pour ses bonnes performances et pour ses traitements concurrentiels.

- Conçu sur le modèle MVC Modèle-Vue-Contrôleur
- ORM avec hibernate

4.3 Technologies choisies

Le choix de la technologie ne peut pas se faire en comparant les avantages et inconvénients des différentes technologies étant donné qu'elles permettent toutes de faire une application web. Dès lors, notre choix s'est déduit selon les besoins du client. Étant donné que ce projet risque d'être repris, une des requêtes du client est de minimiser le nombre de langages utilisés.
Nous avons porté notre choix sur le **Framework MEAN** (**Mongodb**, **ExpressJs**, **Angular**, **NodeJs**). Nous considérons que ce framework utilise des technologies puissantes, d'actualités, ayant une courbe d'apprentissage forte ainsi qu'une communauté active. De plus, n'importe quelle personne ayant fait du développement web connaît le langage utilisé dans la technologie choisie.
La somme de ces avantages correspond parfaitement avec les demandes émises par le client.

4.3.1 MongoDB

Considérée comme la base de données NO SQL Open Source la plus populaire, elle est idéale pour le stockage et le traitement de gros volumes de données. Elle dispose aussi d'un système de recherche optimisé. Les données sont modélisées dans un format qui permet une grande flexibilité d'utilisation.
Sa scalabilité horizontale garantit des performances fiables. C'est-à-dire que si la charge de travail de la base de données augmente, il est très facile de rajouter un serveur supplémentaire, ce qui permet de répartir plus adéquatement cette charge de travail. Les performances globales de la base de données vont augmenter et les délais de réponse vont diminuer (sharding).

4.3.2 NodeJs

NodeJs est une technologie permettant de créer des applications web et plus précisément le côté serveur en JavaScript. Il est connu pour être très rapide principalement pour deux raisons :

- il utilise le moteur d'exécution extrêmement rapide *V8* de *Google Chrome*
- il utilise un modèle non bloquant. Voici un exemple pour illustrer ce modèle :

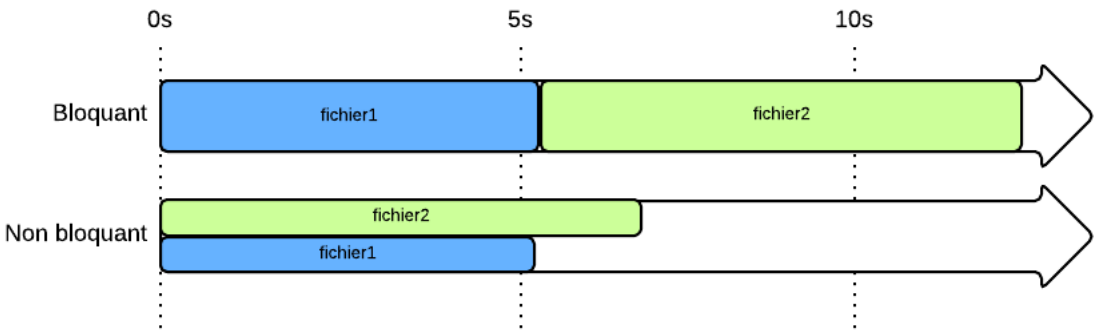


FIGURE 4.15 – *Modèle non bloquant (comme Node.js), les deux fichiers sont téléchargés en même temps et l'ensemble fini plus vite [14]*

Cette figure est un scénario représentant le téléchargement de deux fichiers. Avec le modèle bloquant, les fichiers sont téléchargés l'un après l'autre alors qu'avec le modèle non bloquant, les fichiers sont téléchargés en parallèle. Le temps total d'exécution pour le téléchargement de deux fichiers est donc plus court dans le modèle non bloquant.

De plus NodeJs fournit actuellement le plus gros dépôt de paquets tous langages confondus : Npm. Ces paquets sont des modules créés par d'autres développeurs pouvant être intégrés facilement à une application.

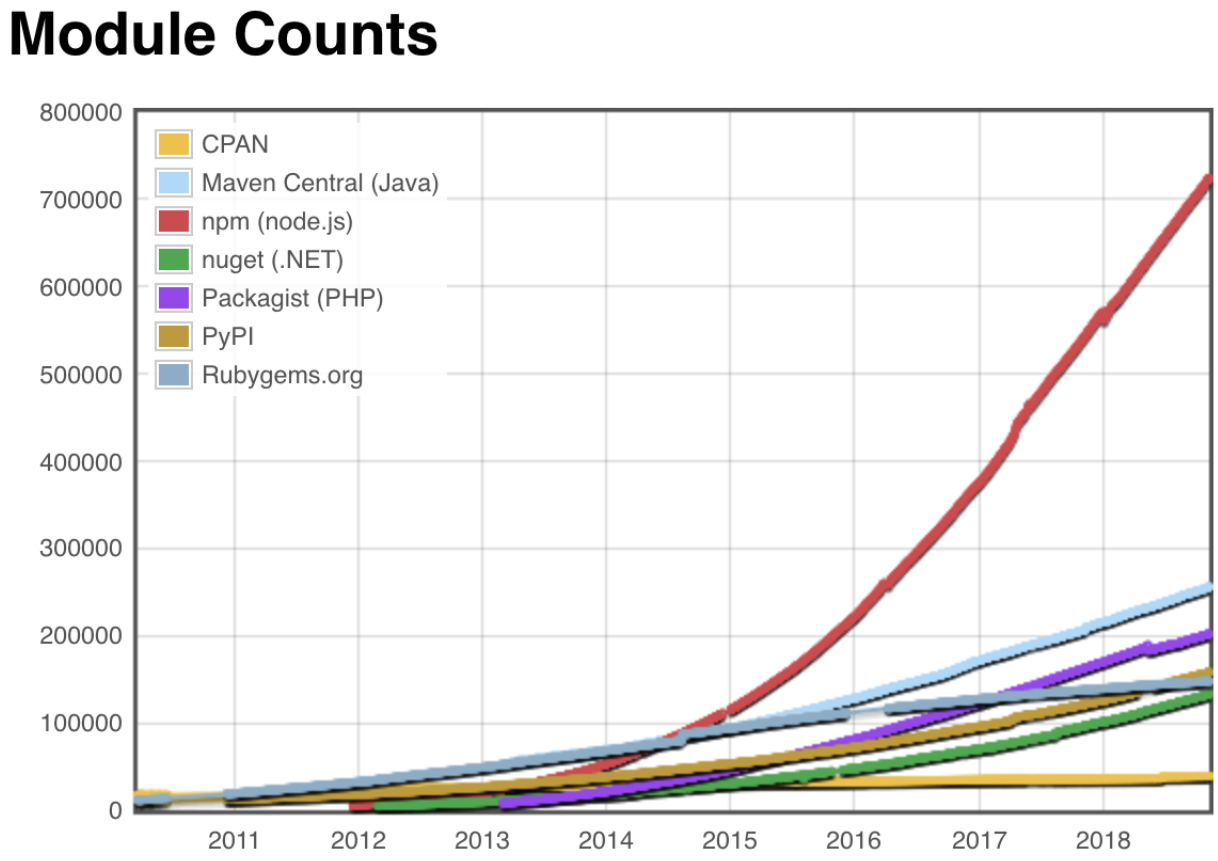


FIGURE 4.16 – *Nombre de modules par technologie*[24]

Npm est donc devenu un incontournable dans le développement d'application web en JavaScript. D'ailleurs, nous avons utilisé différents paquets que nous présentons dans la section "4.3.4 Modules utilisés".

4.3.3 Angular 6

Angular est un framework JavaScript qui permet de créer le côté visuel des applications web. Il est géré et maintenu par Google et donc il y a peu de chances qu'il disparaisse. Basé sur le concept de l'architecture MVC (Modèle Vue Contrôleur), c'est un des frameworks le plus performant et utilisé actuellement. Il offre une série d'outils permettant de développer rapidement et de manière structurée notamment grâce aux *Components*, aux *services* ou *Typescript*.

Lors de notre analyse des différentes technologies, nous avons comparé différentes technologies **frontend**. Il existe trois grands concurrents dans ce domaine :

- Angular
- ReactJs
- Vue.js

Après s'être renseigné sur les trois technologies, il se trouve qu'*Angular* et *ReactJs* sont plus ou moins semblables au niveau de la popularité et de la communauté de développeurs tandis que *Vue.js* est plus en retrait, car celle-ci est beaucoup plus récente.
Ces deux technologies sont propulsées et maintenues par de grands acteurs du domaine du web (*Google* pour **Angular** et *Facebook* pour **ReactJs**). Ce point nous semble important dans le

choix de notre technologie, car théoriquement, elles ne seront pas dépréciées dans un futur proche.

Rappelons que NodeJs et Angular sont deux technologies utilisant le même langage (JavaScript). Le côté client et le côté serveur sont donc codés avec le même langage. Ce paramètre a été pris en compte dans notre choix de technologie, car ce projet a de fortes chances d'être poursuivi par d'autres étudiants ou informaticiens.

4.3.4 Modules utilisés

Mongoose

Pour la gestion des requêtes à la base de données nous avons utilisé la technologie Mongoose. Pour une donnée, elle fournit une solution simple pour la modéliser et gère son écriture, sa validation et génère les requêtes.

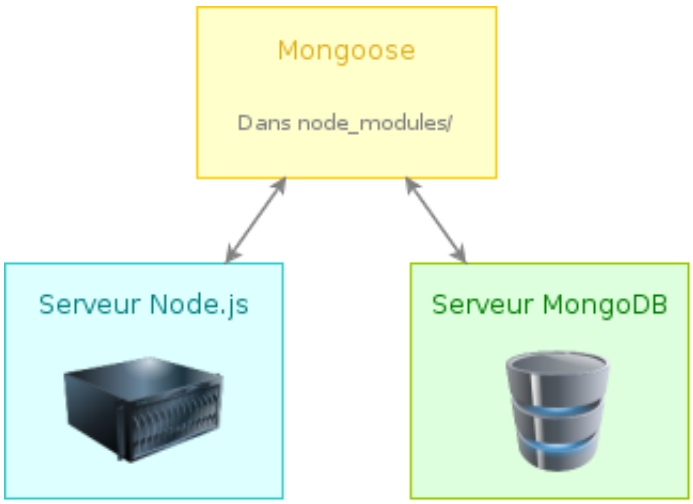


FIGURE 4.17 – *Schéma Mongoose [25]*

Elle est utilisée comme passerelle entre le serveur NodeJs et la base de données MongoDB.

Leaflet

Une des fonctionnalités de l'application est l'affichage d'une carte d'Haïti avec toutes les stations disponibles ainsi que leur état courant. Nous avons utilisé Leaflet qui est la bibliothèque principale open source pour l'affichage de cartes interactives. Elle offre toutes les fonctionnalités de cartographie dont nous avons besoin.
Leaflet a été conçu dans un souci de simplicité, de performance et de convivialité. Il fonctionne efficacement sur tous les supports (ordinateur, tablette, smartphone) et dispose d'une API facile à utiliser et bien documentée.
Grâce à cette bibliothèque, nous pouvons afficher des cartes provenant d'OpenStreetMap et aussi ajouter des interactions sur la carte. L'avantage d'OpenstreetMap par rapport aux autres concurrents est le fait que ce projet soit ouvert et collaboratif, mais aussi sa gratuité. En effet, depuis 2012, Google ne cesse de réduire le nombre d'utilisations gratuites de Google Maps par un site externe. En 2018, Google a encore changé les conditions d'utilisation de ses cartes, ce qui a fortement impacté tous les sites utilisant ce service. La plupart de ces sites cherchent donc une alternative et se sont principalement tournés vers OpenStreetMap, ce qui nous conforte dans notre choix.

Highstock

Afin de représenter les données sous forme de graphique, nous avons utilisé une autre bibliothèque : Highstock. Highstock est utilisé pour représenter des graphiques boursiers et chronologiques. Il comporte des options de navigation sophistiquées et des indicateurs techniques intégrés, le tout en JavaScript.

4.4 Licence

Pour la distribution du code de ce mémoire, nous avons décidé d'utiliser une des licences les plus permissives au niveau des droits d'utilisation.
D'une part, car nous nous sommes basés sur certains projets Open-Source pour l'architecture du back et du front end.
D'autre part, car ce projet a pour but de servir de projet pilote pour Haïti. Nous pensons donc que "bloquer" le code que nous fournissons risque d'être contre-productif au niveau de la reprise, du maintien et de l'amélioration à ce projet.
Nous avons donc opté pour la licence MIT, car celle-ci est la plus permissive pour les futurs développeurs.

Chapitre 5

Produit final

À présent que nous avons compris et analysé la demande du client, nous allons pouvoir présenter notre application finale. Celle-ci est disponible sur un serveur de test à l'adresse suivante :
https://client-d4rk694.c9users.io/
Pour toute personne souhaitant reprendre le projet ou voir le code, celui-ci est disponible à l'adresse suivante :
https://github.com/doricci/TFE4Haiti

5.1 Utilisateurs

Nous avons établi plusieurs types d'utilisateurs dans le cahier des charges. Pour rappel :

- **Un visiteur** est une personne lambda, n'ayant aucun accès à l'application
- **Un chercheur** est une personne ayant un accès, voulant consulter l'application à des fins de recherche
- **Un employé** est une personne ayant un accès, s'occupant de la gestion des stations et travaillant pour le client
- **Un administrateur** est une personne ayant un accès, s'occupant de la gestion générale de l'application

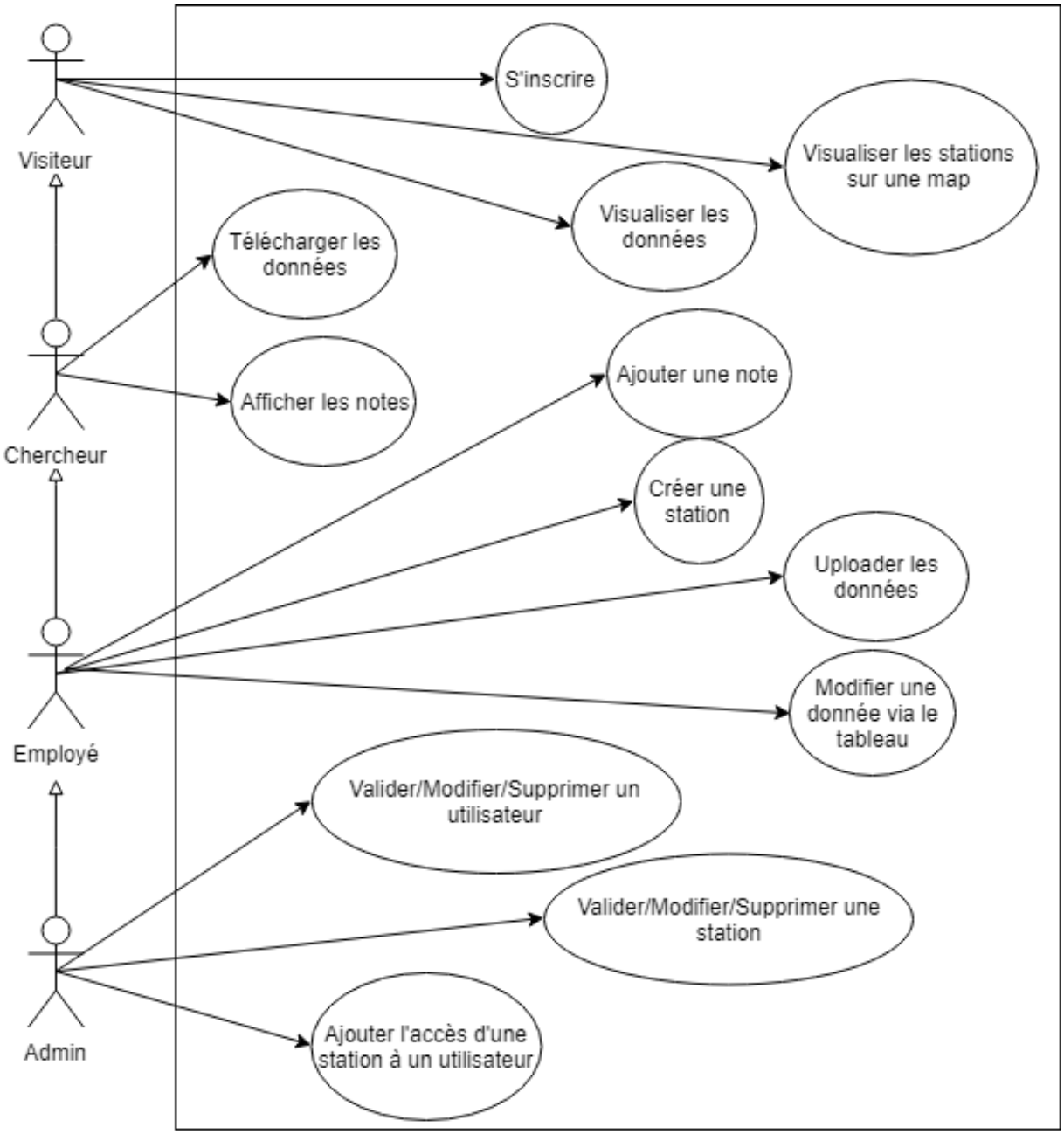


FIGURE 5.1 – Diagramme de cas d'utilisation

5.2 Accueil/Menu

FIGURE 5.2 – Interface Accueil

La figure 5.2 représente l'interface *Accueil*. Intéressons-nous au ruban gris **(1)** en haut de la page. Il représente la barre de navigation principale de l'application web (le menu). Chaque élément du menu sera présenté durant ce chapitre. *Notons que l'utilisateur actuellement connecté est un administrateur qui a donc accès à toutes les fonctionnalités.*
Les points **(2)** présentent un résumé de l'ensemble des données disponibles dans l'application.

5.3 Inscription/Connexion

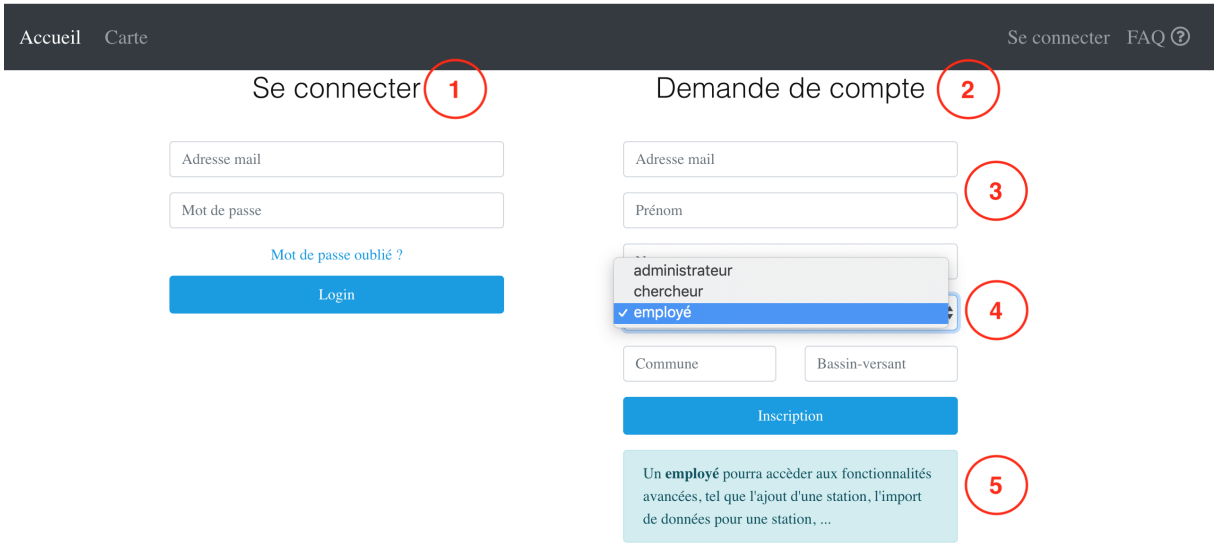


FIGURE 5.3 – Interface Inscription/Connexion

La figure 5.3 représente l'interface Inscription/Connexion. Celle-ci se présente en 2 parties : la connexion (1) et l'inscription (2). Pour se connecter, il suffit de rentrer son email et son mot de passe.
Pour l'inscription nous demandons plusieurs informations :

- Adresse mail
- Prénom
- Nom
- Le rôle du nouvel utilisateur :
  - Administrateur
  - Chercheur
  - Employé
- Une description du rôle du nouvel utilisateur

Une fois la demande envoyée, **l'administrateur** doit confirmer le nouvel utilisateur via *l'interface administrateur.*

Un mail est ensuite envoyé sur l'adresse mail renseignée par le nouvel utilisateur pour l'informer si sa demande de compte a été acceptée ou refusée.

5.4 Carte

Le second lien de navigation permet d'accéder à la fonctionnalité *Carte*. Elle permet d'avoir un aperçu visuel de toutes les stations disponibles (et leur état) ainsi que de rechercher parmi celles-ci.

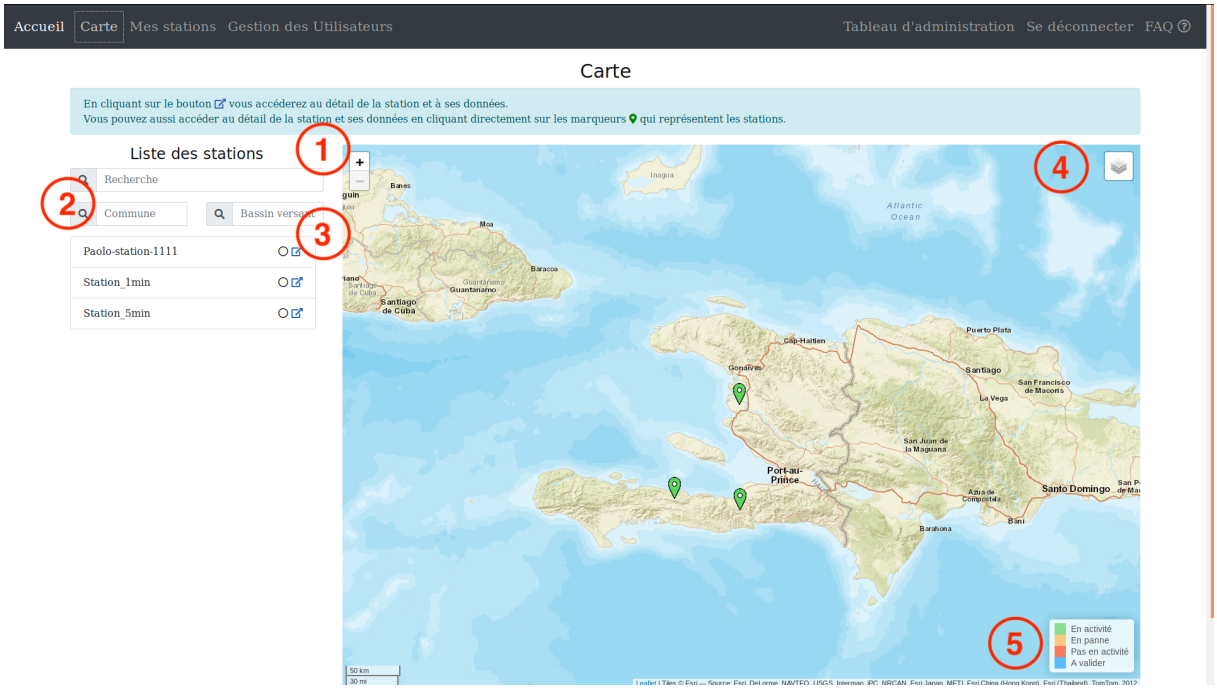


FIGURE 5.4 – Carte

1. Une barre de recherche permet de filtrer les stations à afficher dans la liste en fonction du nom.
2. Deux barres de recherche disposant d'un système d'auto complétion : une pour filtrer en fonction des communes, l'autre en fonction du bassin versant.
3. Une liste est affichée en fonction des différents résultats de la recherche. Cette liste affiche le nom de la station, ainsi que deux actions : la première permet de sélectionner les stations à afficher sur la carte. La deuxième permet d'être re-dirigé vers le détail de la station.
4. Nous avons ajouté la possibilité de choisir le thème de la carte à afficher (satellite, rue, etc.) ainsi que d'afficher uniquement les stations selon un état choisi. Si vous êtes un visiteur, vous ne pourrez pas visualiser les stations ayant l'état "À valider".
5. Une légende permet d'afficher l'état d'une station.

5.5 Mes stations

Le troisième onglet "*Mes stations*" affiche la liste des stations disponibles. Cet onglet est disponible uniquement pour un utilisateur connecté.

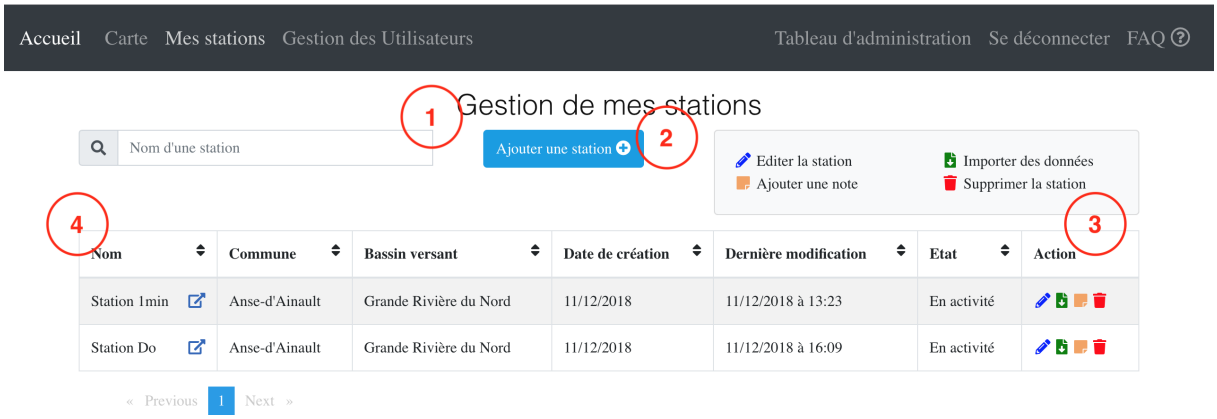


FIGURE 5.5 – Mes stations

1. Une barre de recherche pour filtrer les stations en fonction du nom.
2. Un bouton pour accéder à l'interface de l'ajout d'une station.
3. Une légende qui explique les différentes actions présentes dans la colonne "Action".
4. Un tableau affichant toutes les stations ainsi que certaines informations relatives à la station :
   - Le nom de la station avec le bouton permettant d'accéder à son détail
   - La commune de la station
   - Le bassin versant
   - La date de création
   - La date de dernière modification
   - L'état de la station (En activité, En attente, En panne, Pas en activité)
   - Les différentes actions possibles

5.5.1 Ajout d'une station

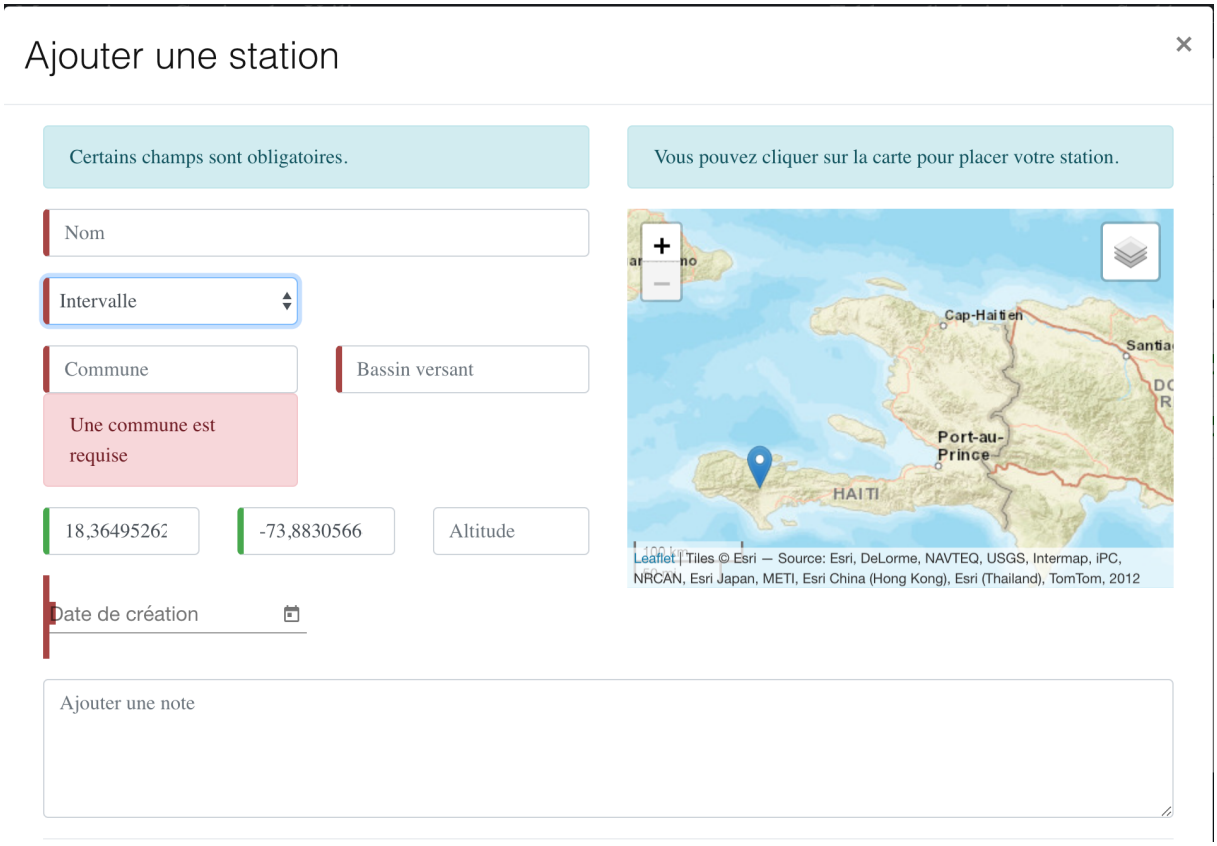


FIGURE 5.6 – Interface pour l'ajout d'une station

Afin d'ajouter une station, il faut entrer des informations :

- Le nom
- L'intervalle de collecte des données pluviométriques
- La commune
- Le bassin versant
- La latitude et la longitude. Une carte interactive permet d'entrer facilement les coordonnées en cliquant directement sur celle-ci.
- L'altitude
- La date de création
- Une première note

Lorsqu'une information est manquante, l'interface affiche un message d'erreur expliquant l'information manquante.

5.5.2 Modification d'une station

L'interface de modification d'une station est identique à l'interface d'ajout d'une station. Les données de la station concernées sont pré-complétées, il ne reste plus qu'à les modifier.

5.6 Informations d'une station

Nous proposons une interface permettant de réunir toutes les informations d'une station. Cette interface est composée de plusieurs onglets :

- Détails
- Graphiques
- Tableaux
- Notes
- Utilisateurs

5.6.1 Détails d'une station

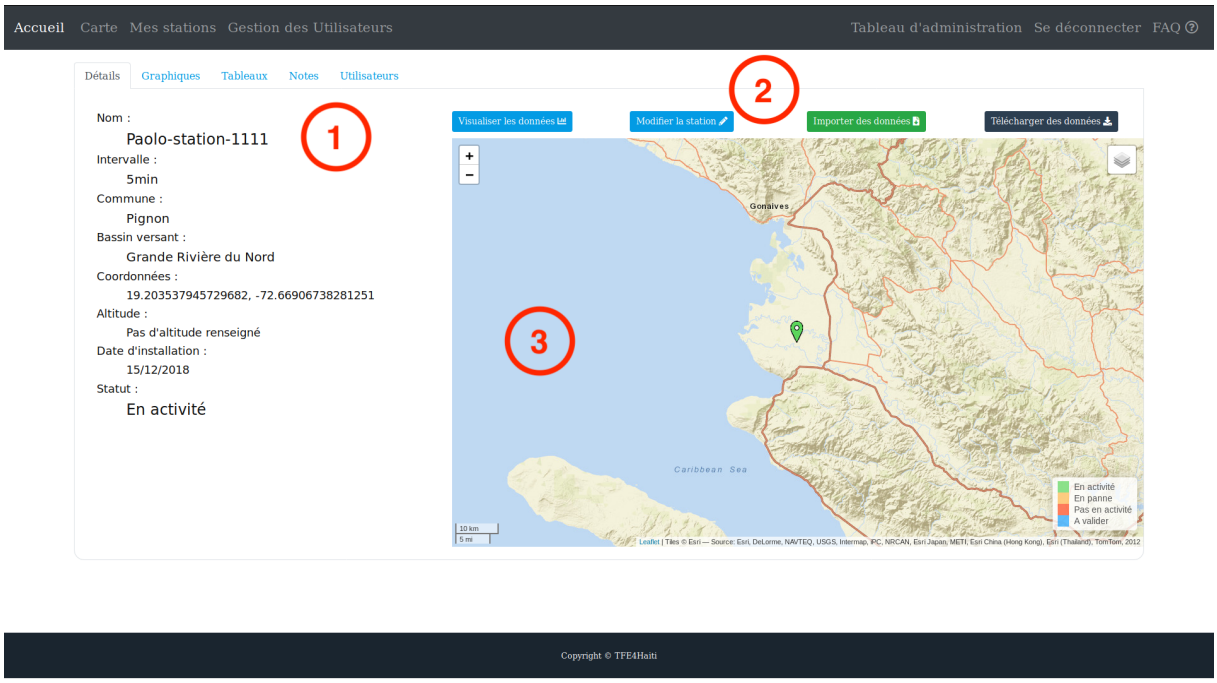


FIGURE 5.7 – Interface du détail d'une station

Cette interface contient :

1. Les informations de la station
2. Des boutons (s'affichant en fonction du rôle de l'utilisateur connecté) permettant d'accéder aux différentes fonctionnalités :
   - Visualiser des données via un graphe
   - Modifier la station
   - Importer des données pluviométriques
   - Télécharger des données pluviométriques
3. D'une carte affichant la station

5.6.2 Graphiques

Cette interface permet de visualiser les données pluviométriques sous la forme d'un graphique :

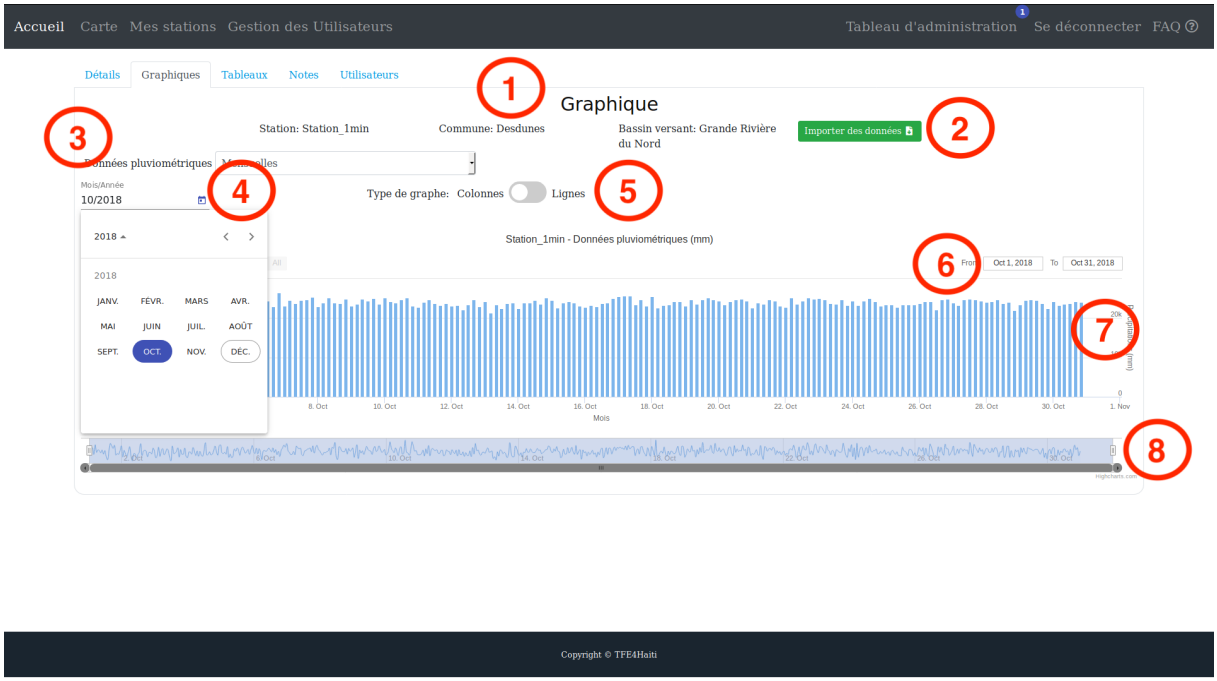


FIGURE 5.8 – Graphique des données

Cette interface est composée de :

1. Informations sommaires sur la station
2. Un bouton pour importer des données pluviométriques
3. La possibilité de choisir un intervalle pour l'affichage des données (Annuelle, Mensuelle, Journalière)
4. La date à afficher
5. La possibilité d'afficher le graphique sous forme de bâtonnets ou de lignes
6. L'affichage du *range* de date affiché par le graphique en fonction de sa ligne du temps (présente sous le graphe)
7. Les données pluviométriques
8. La ligne du temps interactive

5.6.3 Tableau

Cette interface permet d'afficher les données pluviométriques sous la forme d'un tableau :

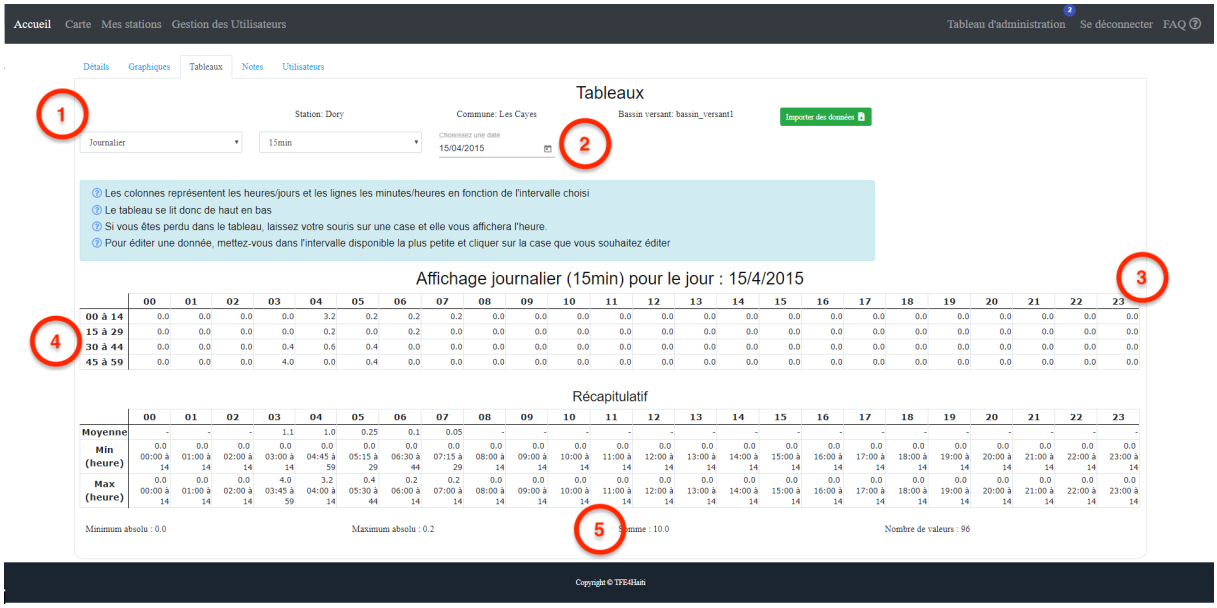


FIGURE 5.9 – Tableau des données

Cette interface comporte :

1. Liste permettant de choisir un intervalle parmi ceux disponibles pour la station
2. Le choix d'une date/d'un mois concernant les données à visualiser
3. Les heures/jours
4. Les minutes/heures
5. Un récapitulatif des données

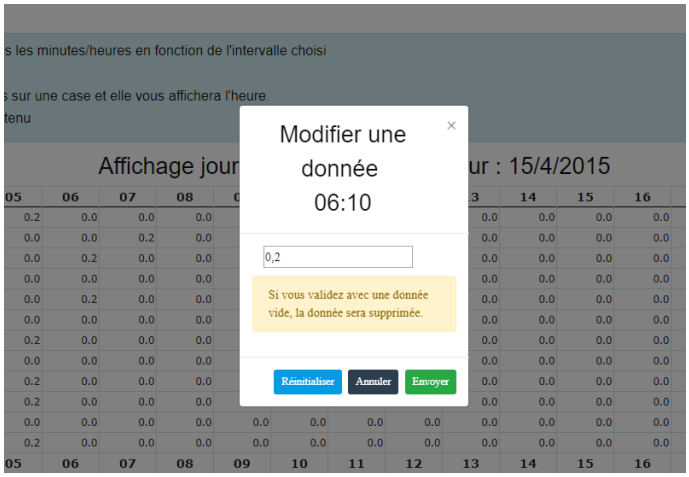


FIGURE 5.10 – Édition d'une donnée

Si vous avez sélectionné l'intervalle de la station, en cliquant sur une donnée vous pouvez éditer celle-ci. Si la case est laissée vide, un message indique à l'utilisateur que la donnée va être mise à 'vide'.

5.6.4 Notes

L'interface note permet d'ajouter des notes pour une station pour, par exemple, communiquer avec d'autres utilisateurs ou signaler qu'un fichier a été envoyé :

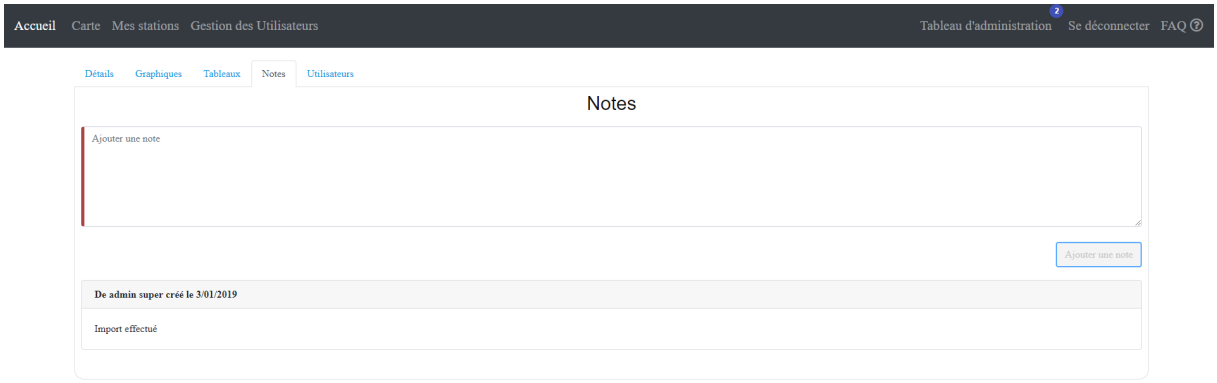


FIGURE 5.11 – Interface pour l'ajout et l'affichage de notes

5.7 Utilisateurs

Un administrateur peut gérer l'accès de la station pour les utilisateurs ayant accès à cette dernière. En cliquant sur le bouton **(1)**, il va ajouter l'utilisateur sur la station et en cliquant sur le bouton **(2)**, il va le retirer de la station. Il peut aussi filtrer la liste des utilisateurs sur base de leur nom **(3)** :

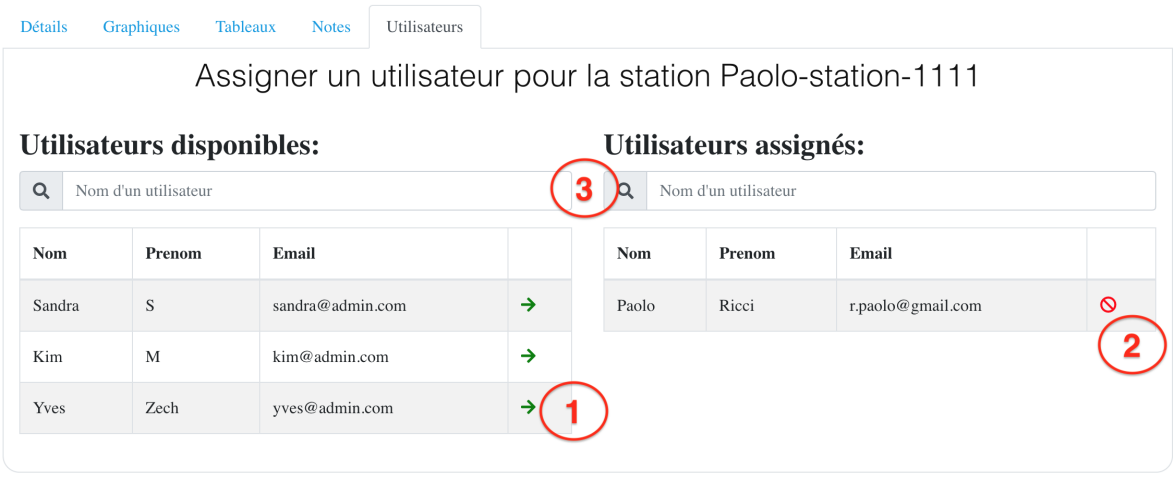


FIGURE 5.12 – Gestion des accès utilisateur pour une station

5.8 Import de données

Une fonctionnalité importante est l'ajout de données pluviométriques pour une station.

FIGURE 5.13 – Interface d'upload de données

Cette interface est divisée en deux parties :

- Manuelle (partie gauche) : on rentre "à la main" la date, l'heure et la valeur. Il est possible d'envoyer plusieurs données en une fois en ajoutant une ou plusieurs lignes.
- Via un fichier CSV (partie droite) : on envoie un fichier CSV pour insérer les données, si celui-ci est conforme au pattern défini pour les fichiers CSV :

```
2018-12-01 00:00:00; 0.2
2018-12-01 00:01:00; 0.1
2018-12-01 00:02:00; 0.4
2018-12-01 00:03:00; 0.3
```

FIGURE 5.14 – $Template_1$ pour importer via CSV

ou

```
1/12/2018 00:00; 0.1
1/12/2018 00:01; 0.1
1/12/2018 00:02; 0.2
1/12/2018 00:03; 0.4
```

FIGURE 5.15 – $Template_2$ pour importer via CSV

Pour rappel, chaque donnée doit être validée par un administrateur.
Si une ligne ne correspond pas au format requis, un message d'erreur est affiché à l'utilisateur, lui précisant quelle ligne pose problème.

5.9 Télécharger des données

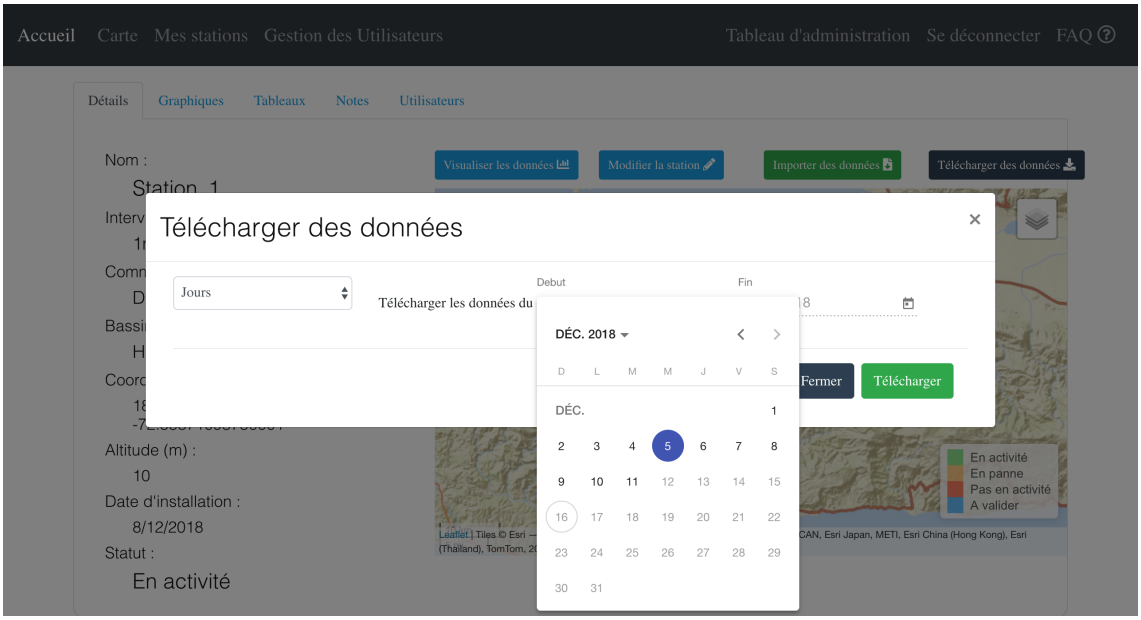


FIGURE 5.16 – Interface pour télécharger des données

Cette interface permet de télécharger les données au format CSV. Il faut d'abord sélectionner un intervalle (jour, mois, année). Ensuite, il faut sélectionner deux dates. Une fois la demande effectuée, un mail contenant le lien de téléchargement est envoyé sur l'adresse mail de l'utilisateur lorsque le fichier CSV est créé.

5.10 Administrateur

5.10.1 Gestion des utilisateurs

L'interface *Gestion des utilisateurs* permet d'avoir une vue globale des utilisateurs acceptés, de les supprimer et de les modifier :

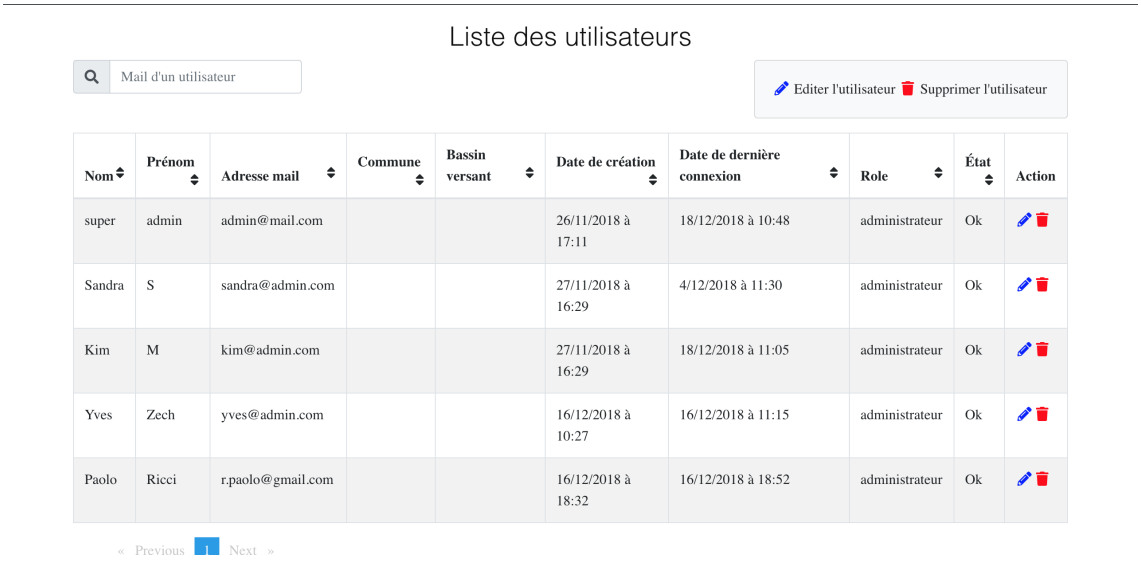


FIGURE 5.17 – Gestion des utilisateurs

À travers cette interface, il est possible de modifier les données d'un utilisateur :

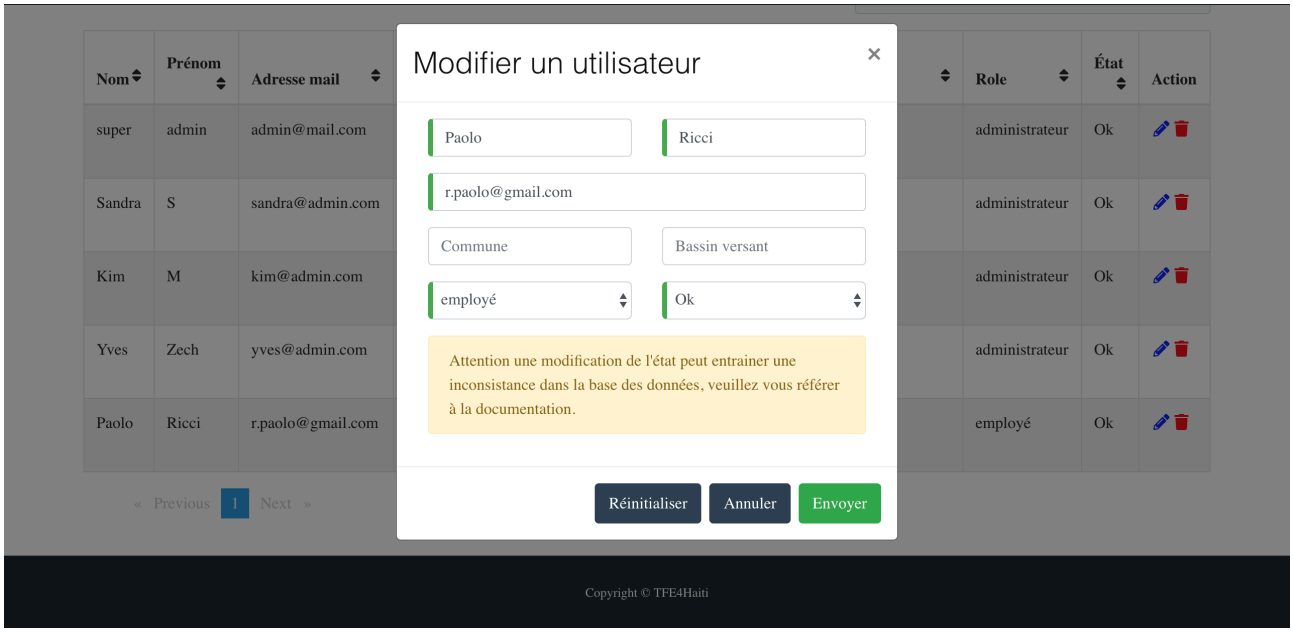


FIGURE 5.18 – Modification d'un utilisateur

5.10.2 Validation

L'administrateur peut valider ou refuser l'inscription d'un utilisateur. S'il refuse, il peut justifier la raison de ce refus. Un mail sera envoyé à l'utilisateur avec la raison du refus écrite par l'administrateur. Si l'administrateur n'a rien écrit, un message par défaut est envoyé. Ensuite, toutes les informations relatives à cet utilisateur seront supprimées. Voici l'interface de refus d'un utilisateur :

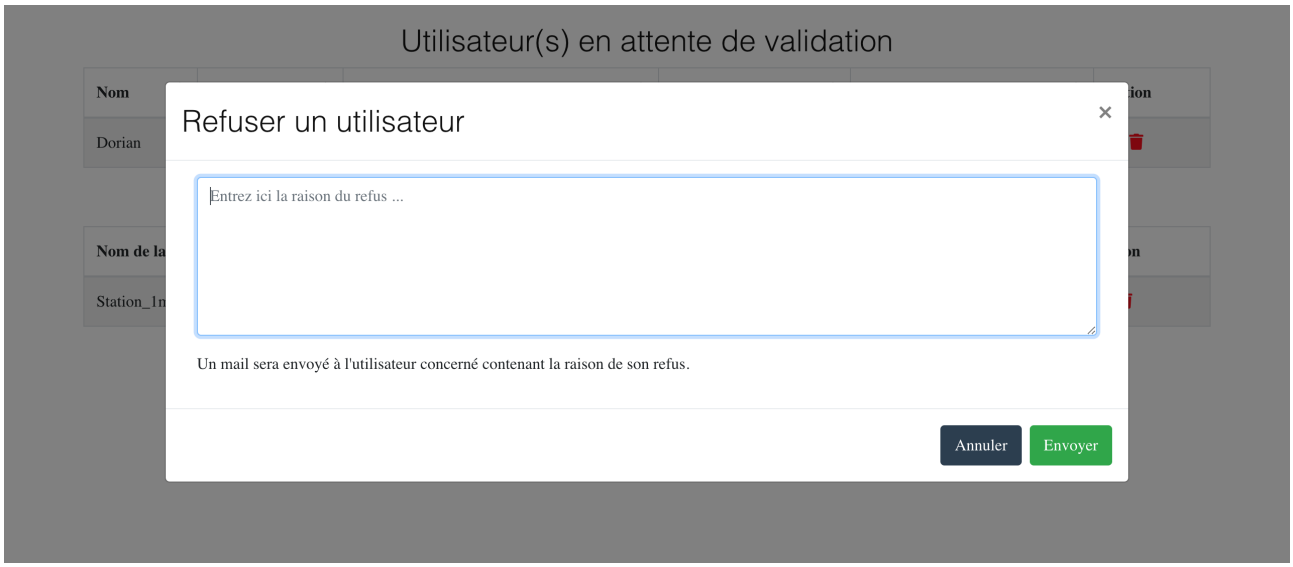


FIGURE 5.19 – Refus d'un utilisateur

L'interface *Tableau d'administration* permet de gérer toutes les demandes d'ajout (utilisateurs, stations, données) :

Cette interface est divisée en trois parties :

- La gestion des utilisateurs en attente
- La gestion des stations en attente de validation
- La gestion des données en attente de validation. Ces données peuvent être dans trois états, **Manuel** qui correspond à la mise à jour ou l'ajout d'une donnée, **Fichier** qui correspond à l'import d'un fichier (ce fichier peut être téléchargé afin de vérifier manuellement les valeurs) ou encore **Donnée à supprimée** qui correspond à la suppression d'une donnée.

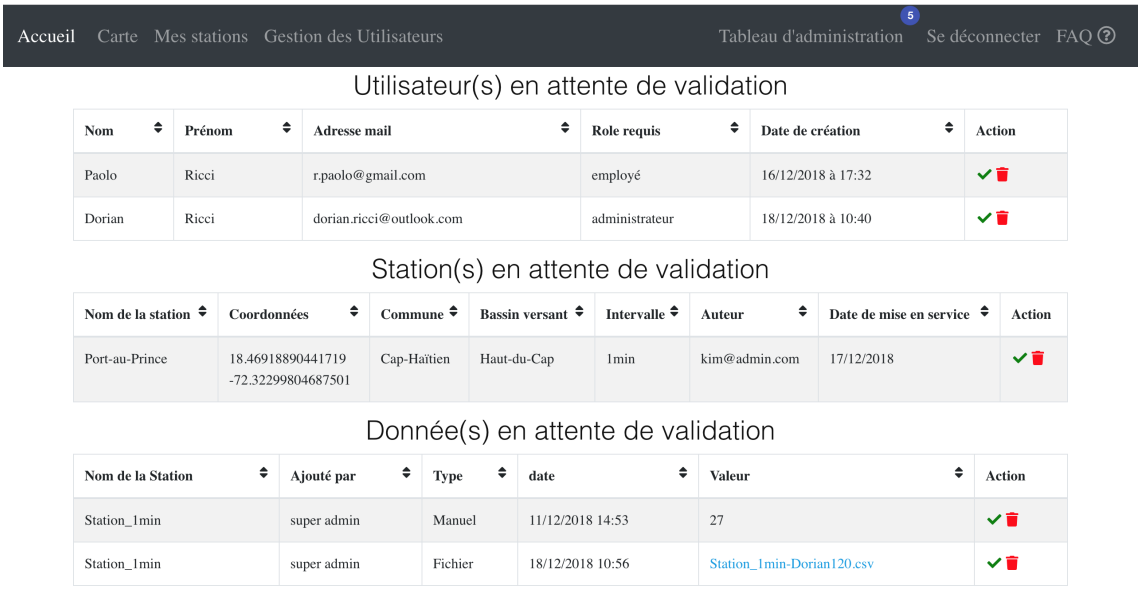


FIGURE 5.20 – Interface *Tableau d'administration*

Chapitre 6

Implémentation

À présent que nous avons présenté l'application, intéressons-nous au côté technique de l'application. Nous allons vous décrire l'architecture mise en place ainsi que certains choix effectués durant l'implémentation.

6.1 Architecture

Étant donné que l'architecture choisie est une **API REST**, la partie client (Front-end) et la partie serveur (Back-end) sont complètement séparées. L'avantage de cette séparation est que cela permet de porter l'application sur de nouveaux supports (ie : smartphone) sans devoir ré-implémenter la partie serveur. La figure 6.1 résume brièvement l'architecture mise en place :

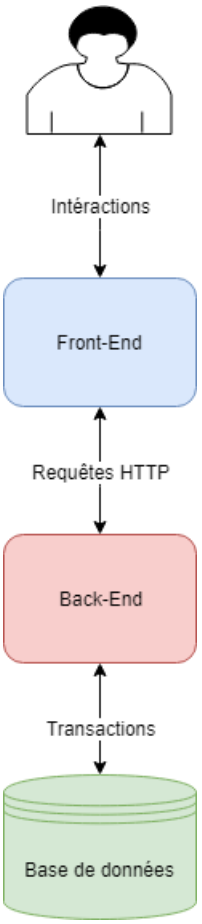


FIGURE 6.1 – Schéma de l'architecture

- Front-end : représente tout ce que peut voir le l'utilisateur. Lorsque le front-end a besoin de données, il communique avec le back-end via des requêtes HTTP.
- Back-end : représente le serveur qui permet d'effectuer des calculs et de récupérer des données.
- Base de données : permet de stocker les données de l'application. Elle communique avec le back-end via des transactions.

6.1.1 Architecture Front-End

L'architecture du Front-End a été pensée pour permettre une compréhension simple et rapide. Elle permet aussi une prise en main rapide par les personnes reprenant cette application.

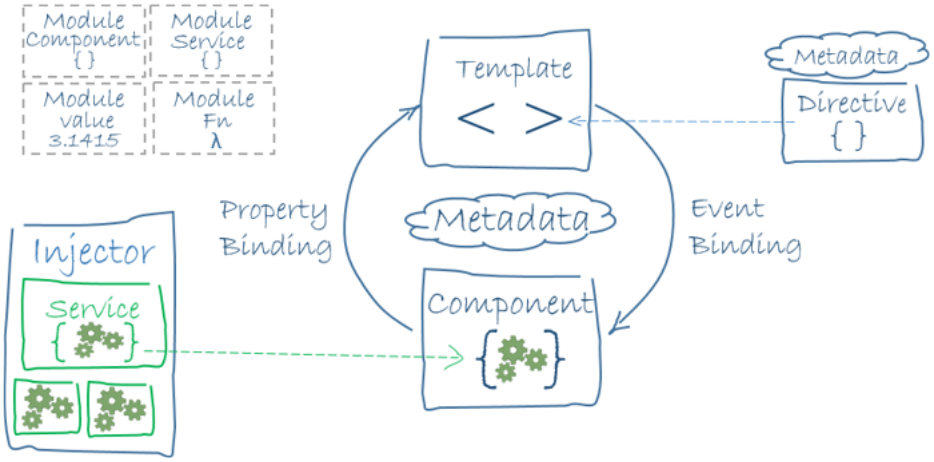


FIGURE 6.2 – Schéma architectural front-end [19]

La figure ci-dessus reprend l'architecture ainsi que les différents liens existants entre les différents composants d'Angular 6.

- Les *Templates* représentent l'interface graphique et tous les éléments visibles dans celle-ci. Angular ajoute aussi un système de *Directives* fournissant ainsi des fonctionnalités n'existant pas dans le *HTML* classique, tel que la boucle **for**, le **if**, etc.
- Les *Components* sont des contrôleurs qui permettent de gérer la logique relative aux *templates*
- Les *Injectors* sont des classes ne devant pas être instanciées manuellement. Une fois injectées, leurs fonctionnalités deviennent disponibles pour la classe dans laquelle elle a été injectée. l'exemple le plus parlant sont les *Services* qui permettent de partager des morceaux de code entre différentes parties de l'application et qui permettent aussi aux *components* de communiquer entre eux ou avec le monde extérieur (Serveur web, Service de cartographie, etc.).

Angular fonctionne sur le principe du *2 way binding*, c'est-à-dire, qu'entre le *template* et le *component* les variables sont liées dans les deux sens. Si la valeur change au niveau du *template*, elle sera mise à jour dans le *component* et inversement.
Ensuite, on retrouve les *services*, qui peuvent être "injectés" directement dans les différents composants. Ils permettent la communication entre les différents *components* ou avec le(s) serveur(s).

Division des modules/components de l'application

Dans Angular, chaque *component* peut être lié à une ***URL*** afin d'être affiché lorsqu'un utilisateur se rend sur cette dernière.
Les interfaces de notre application se divisent en trois grandes parties.
Elle reprend les parties suivantes :

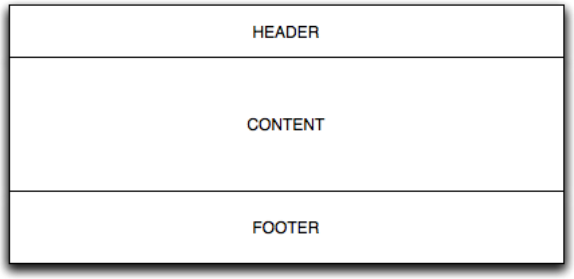


FIGURE 6.3 – Schéma Structure Front-End [27]

- Le *Header*/Menu : Cette partie est composée du *component* **menu**, il gère tout ce qui concerne l' affichage du menu de l'application.
- Le *Content* : Cette partie-ci, gère l'affichage des *components* à afficher en fonction de l'**URL** sur laquelle se trouve l'utilisateur.
- Le *Footer* : Cette partie est composée du *component* **footer**. Ce dernier ne sert qu'à l'affichage du *footer* dans le site.

6.1.2 Architecture serveur

L'architecture serveur a aussi été pensée pour être facile à prendre en main. Notre implémentation permet de rajouter de nouvelles fonctionnalités en seulement quatre lignes de code.

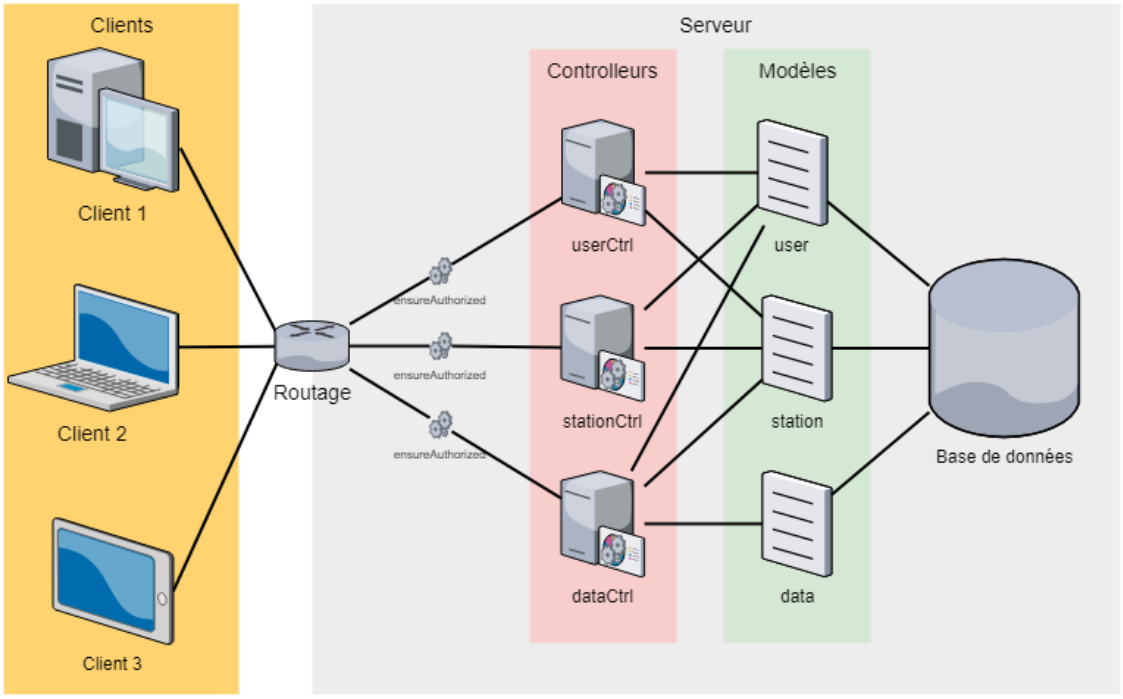


FIGURE 6.4 – Architecture du serveur

Lorsqu'un client effectue une requête et que celle-ci arrive sur le serveur, *le routage* la redirige sur le contrôleur correspondant (pour des raisons de simplicité, tous les contrôleurs ne sont pas dans la représentation). Une vérification des accès est effectuée permettant de vérifier si le client peut effectuer ce qu'il demande. Par exemple, si l'utilisateur demande à modifier une station (ce que seul un administrateur peut effectuer), et qu'il n'est pas un administrateur, la requête va être refusée. À chaque requête effectuée nécessitant une vérification des accès (p. ex. modification d'une station), nous allons récupérer l'utilisateur en base de données via son jeton d'identification, et vérifier que son rôle lui permette d'effectuer ce qu'il demande.

Si l'utilisateur a les droits pour faire la requête demandée, le *contrôleur* effectue son calcul. Via les *modèles*, *le contrôleur* récupère les données requises à la requête via la base de données. Une fois les données récupérées, il valide les modifications (si nécessaire). Si aucune contrainte sur les données n'est violée, il renvoie la réponse au client.

Pour une explication plus technique du back-end, référez-vous à l'annexe :
Explication détaillée de l'architecture back-end.

6.2 Schéma de la base de données

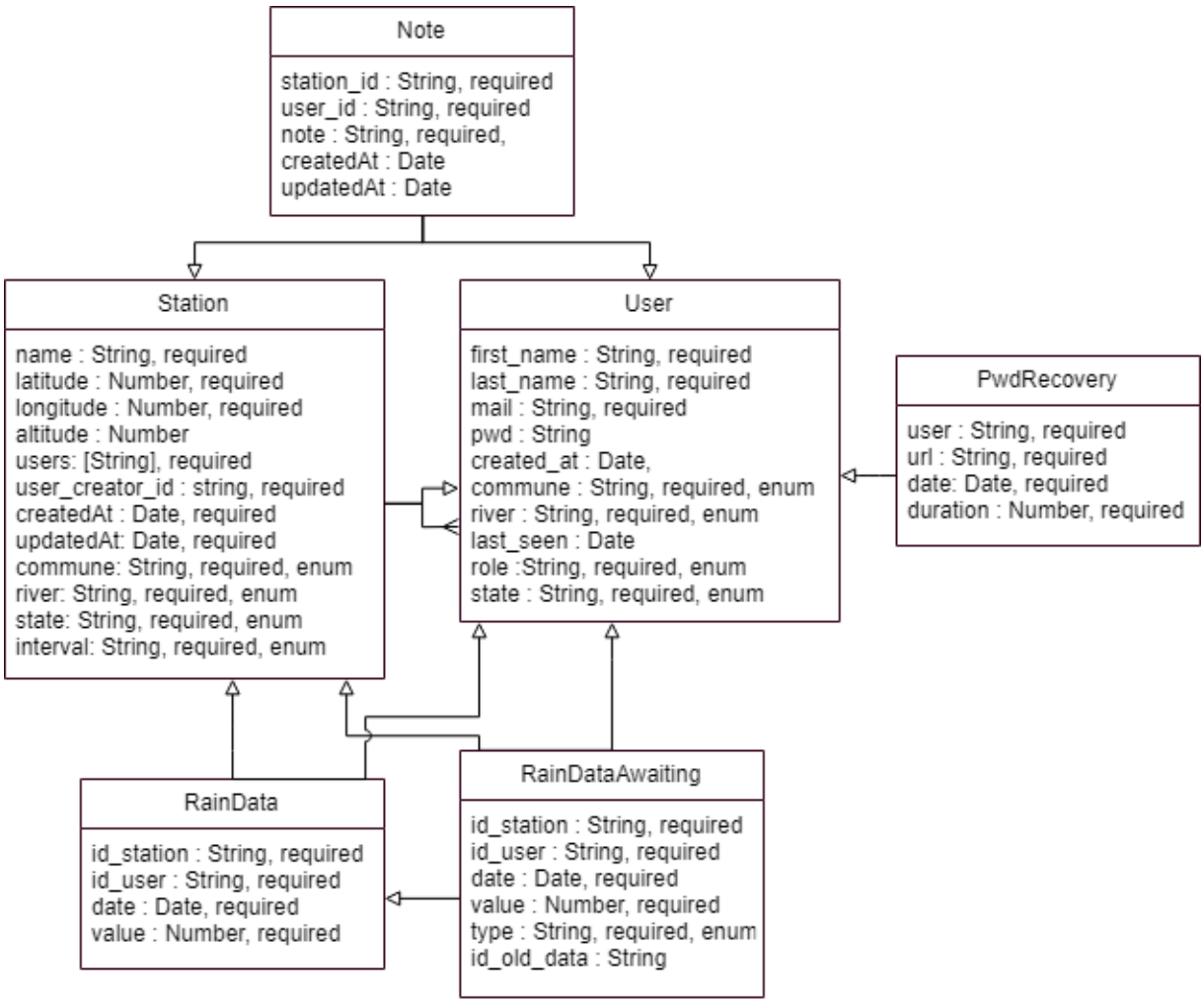


FIGURE 6.5 – Schéma de la base de données

Notre base de données contient six objets différents (de haut en bas) :

- La **Note** représente une note qu'un employé peut écrire sur une station.
- La **Station** contient toutes les informations directement liées à cette dernière (son nom, sa latitude et longitude, etc.), la personne ayant créé la station et une liste d'utilisateurs ayant accès à cette station. Un état a été aussi ajouté permettant de savoir si cette station est en panne, en activité, etc.
- Le **User** représente un utilisateur (défini par son adresse mail, son nom, etc.). De plus, nous retenons dans cette table son rôle (administrateur, employé, ou chercheur) et son état qui permet de savoir si l'utilisateur est en attente de confirmation par l'administrateur, s'il est supprimé, etc. Tous les mots de passe sont stockés de manière cryptée grâce à l'utilisation d'une bibliothèque spécialisée.
- La table **PwdRecovery** concerne toute la partie permettant à un utilisateur de définir ou de changer son mot de passe.
- La table **RainData** contient une valeur et la date de la donnée pluviométrique, ainsi que la personne ayant inséré cette donnée et la station pour laquelle cette donnée s'applique.
- La table **RainDataAwaiting** permet de retenir les données pluviométriques en attente de confirmation par l'administrateur. Cette table est plus complexe que la table **RainData** :
  - Import manuel : une donnée est insérée seule dans l'application. Chaque donnée importée manuellement sera donc validée individuellement.
  - Import de fichier : lorsque l'utilisateur importe un ensemble de données dans un fichier, c'est le fichier entier qu'il faut traiter lors de la validation. Nous devons donc garder le fichier complet.
  - Mise à jour d'une donnée : Lors de la mise à jour d'une donnée, celle-ci doit aussi être approuvée par un administrateur. En attendant d'être validée, l'ancienne donnée doit toujours rester visible.

6.3 Choix d'implémentation

Durant le développement de l'application, nous avons été confrontés à plusieurs obstacles. Dans cette section, nous allons vous énoncer certains choix d'implémentation qui nous ont permis de surmonter ces obstacles.

6.3.1 Les dates

Durant le développement de l'application, et au vu de la nature de l'application, nous avons dû créer et manipuler des dates. Certains cas, tels que le fuseau horaire, sont gérés de base.
Lors de la création d'une date, le langage utilise une **Variable d'environnement** lui permettant de savoir dans quel fuseau horaire la personne, créant la date, se situe. En fonction de la localisation de l'utilisateur, la date affichée risque de changer. Nous avons pu observer ce problème grâce à un membre de l'équipe ayant une variable d'environnement différente. Pour la même date, nous obtenions donc deux affichages différents.

Au vu de la nature de l'application, il faut que les dates soient identiques pour toute personne utilisant l'application peut importe sa localisation. Pour parvenir à fixer ces dates, nous avons fait le choix d'enregistrer les dates sans tenir compte du fuseau horaire. Si un utilisateur, se situant dans un autre fuseau horaire que celui d'Haïti, choisit d'entrer des données, il doit donc absolument entrer des données contenant l'heure qu'il était en Haïti. Ainsi quiconque consultant l'application visualisera les données à l'heure locale d'Haïti.

6.3.2 Les contraintes sur les données

Pour une question de sécurité, chaque donnée envoyée par l'utilisateur doit être vérifiée. Une vérification uniquement du côté client n'est pas suffisante, étant donné qu'il existe une **API REST**. Nous nous sommes donc tournés vers le système de contrainte de Mongoose et avons découvert que nous pouvions ajouter des contraintes directement dans *le modèle* avec des messages d'erreurs spécifiques pour chaque contrainte violée :

```
1  latitude: {
2    type: Number,
3    required: [true, 'Veuillez fournir une latitude'],
4    min: [-90.0, 'La latitude doit etre superieure a -90'],
5    max: [90.0, 'La latitude doit etre inferieure a 90']
6  }
```

Si, lors de la création ou modification d'un objet, une de ces contraintes n'est pas respectée, *Mongoose* affiche le message d'erreur lié à la violation de la contrainte au client.

6.3.3 La documentation

Un projet informatique est composé de plusieurs milliers voire dizaine de milliers de lignes de codes. Pour s'y retrouver facilement, les informaticiens ont développé un moyen : la documentation. Celle-ci permet d'expliquer des bouts de codes longs et complexes. Étant donné que ce projet est un prototype et sera sûrement repris par d'autres informaticiens, nous avons ajouté une documentation précise et complète. Nous avons donc choisi d'utiliser *JsDoc*. Il s'agit d'un outil permettant de générer automatiquement une documentation sous forme de page HTML. Les prochains informaticiens pourront donc reprendre et comprendre notre projet beaucoup plus facilement. Voici un exemple :

```
1  /**
2   * nom_methode - Description methode
3   *
4   * @param {type} nom_param  Description param
5   * @return {type}           Description du retour
6   */
```

6.3.4 Gestion des erreurs

Dans une application web, de nombreux bugs peuvent surgir. Afin de garder la trace de ceux-ci, nous utilisons un *logger*.
En 2017, l'**OWASP** fait entrer le manque de log et de monitoring dans le top des 10 des vulnérabilités des applications web. Cette vulnérabilité reste encore à ce jour (en 2018) dans ce classement. C'est pourquoi nous avons porté une attention particulière au log des erreurs. Dès qu'une erreur survient, celle-ci est conservée. Chaque log respecte un format très précis :
la date \t le niveau de l'erreur : [le nom du fichier] le nom de la méthode : l'erreur survenue
L'informaticien peut donc comprendre d'où vient l'erreur et la corriger plus aisément. Si plusieurs erreurs ont lieu dans la même méthode, le nom de la méthode sera suivi d'un chiffre afin de retrouver plus facilement la cause de l'erreur.
Certaines erreurs ne sont pas affichées à l'utilisateur pour des raisons des sécurités.

Chapitre 7

Validation

Dans cette partie, nous aborderons les tests effectués et la validation des fonctionnalités développées tout au long de ce projet. La validation des fonctionnalités s'est déroulée en deux parties : les **tests fonctionnels** et les **tests utilisateurs**.

7.1 Tests fonctionnels

Au fur et à mesure du développement des fonctionnalités, nous les avons directement testées afin d'être sûrs qu'elles soient fonctionnelles et que nous ayons bien pris en compte l'ensemble des cas qu'il nous est possible de tester via l'interface graphique ou via un outil permettant de simuler des requêtes au serveur. Cet outil nous permet de tester ce que nous ne pouvions pas tester via l'interface graphique.

Pour s'assurer que chaque fonctionnalité terminée soit correcte, chaque membre du groupe la testait afin de couvrir le maximum de cas possible (différents navigateurs, etc.).
Nous avons également mis en place une série de tests *Selenium*. Ceux-ci sont basés sur l'ensemble des tests utilisateurs. *Selenium* est un moyen d'automatiser les tests depuis un navigateur web réel. Ils nous permettent de nous assurer qu'entre chaque nouveau déploiement, aucune fonctionnalité n'a été altérée.

7.2 Tests utilisateurs

Lors de chaque réunion hebdomadaire avec le client, nous lui envoyions une version contenant les dernières fonctionnalités, afin qu'il puisse les tester et s'assurer qu'elles correspondent bien à ses attentes. Nous avons aussi fait tester une série de fonctionnalités à plusieurs personnes de notre entourage (amis, famille et proches). Ces personnes ont chacune d'elles plus ou moins d'affinité avec l'utilisation des dernières technologies.
Ceci nous a permis d'avoir un retour le plus complet possible sur la manière dont une personne utilisera notre application, mais aussi sur l'ergonomie de nos interfaces. Pour ce faire nous disions à l'utilisateur, ce qu'il devait faire sans lui dire comment. Par exemple, nous lui demandions de modifier une station, en commençant de la page d'accueil sans que ce dernier ne soit connecté. Si ce dernier avait une question, nous le redirigions d'abord sur la section FAQ. S' il ne trouvait pas la réponse dans cette section, nous l'aidions. Si nous l'aidions, nous faisions en sorte de rajouter davantage d'aide dans la FAQ afin que la prochaine fois qui nous testions la même fonctionnalité avec un autre utilisateur, le test se passe sans poser de problème.

Nous avons divisé les fonctionnalités à tester en trois parties, chaque partie correspondant à un type d'utilisateur :

Visiteurs

- Visualisation de l'ensemble des stations
- Visualisation des détails d'une station
- Visualisation des données sous chaque forme existante (à l'utilisateur de trouver les différentes possibilités)
- Création d'un compte utilisateur avec le rôle de *Chercheur*

Chercheur

- Téléchargement des données
- Afficher l'ensemble des notes d'une station
- Mot de passe oublié

Employé

- Ajout d'une station
- Import d'une donnée (via fichier et manuellement) pour une station créée par soi-même.
- Import d'une donnée (via fichier et manuellement) pour une station étant dans le bassin versant où l'employé a accès.
- Import d'une donnée (via fichier et manuellement) pour une station étant dans la commune où l'employé a accès.
- Import d'une donnée pour une date qui possède déjà une valeur.
- Mise à jour d'une donnée

Administrateur

- Mise à jour d'un utilisateur
- Mise à jour d'une station
- Validation d'une donnée
- Validation d'une modification de donnée (update/delete)
- Validation d'un utilisateur
- Validation d'une station

7.3 Tests unitaires

Étant donné que nous avons privilégié le développement des fonctionnalités demandées par le client, nous n'avons pas pu mettre en place un système de tests unitaires complets tel que nous l'aurions voulu. Un test unitaire est un test permettant de tester une portion précise de code afin de s'assurer qu'elle ait le comportement attendu. Néanmoins, nous avions déjà prévu les technologies et un exemple de code pour ceux-ci.
Au niveau des technologies envisagées pour effectuer ces tests, nous avions prévu d'utiliser *Mocha* et *Chai,* car ceux-ci sont très populaires dans le monde du test en JavaScript, de plus il existe beaucoup de documentation et d'exemple permettant ainsi une mise en place simple et compréhensible. Afin d'obtenir les statistiques de couverture de code, nous avions aussi prévu d'utiliser la libraire *nyc*.

Chapitre 8

Conclusion

Pour rappel, l'objectif principal de l'application web était de proposer une solution permettant de centraliser et visualiser l'ensemble des données hydrologiques du pays.

Suite aux derniers feedbacks de la part de notre client, nous pouvons affirmer que l'ensemble des objectifs fixés ont été atteints. Notre application permet d'ajouter des données pluviométriques de manière simple et rapide. Ces données peuvent être aisément visualisées via des tableaux et des graphiques. Quel que soit le type d'affichage choisi, l'utilisateur a toujours la liberté de changer la date et l'intervalle à afficher. Un système de téléchargement permettant d'exporter les données, comprises entre deux dates, a également été mis en place. Autour de ces fonctionnalités, l'utilisateur profite d'une carte interactive qui rend l'application plus conviviale. Chaque ajout ou modification de données (pluviométriques, stations ou utilisateurs) doit être confirmé par un administrateur. Ce système ajoute une sécurité pour l'ensemble des données stockées dans l'application.

L'application web est basée sur des technologies fiables et puissantes. Le front-end est géré par Angular 6 qui est le framework de *Google*. Le back-end est implémenté avec NodeJS, une technologie qui évolue rapidement et qui est utilisée par de grandes sociétés telles que *Netflix* ou *Uber*. L'architecture côté serveur a été pensée de telle sorte que l'ajout de fonctionnalité soit le plus simple possible. Ainsi, pour un développeur débutant, poursuivre notre travail ne sera pas très complexe. Et si par hasard, ce dernier rencontre des difficultés, les annexes de ce mémoire lui fourniront une aide supplémentaire.

Cependant, l'application web peut encore être améliorée. Par exemple, l'ergonomie de l'interface graphique. Bien que nous utilisons une technologie éprouvée, réaliser un design à la fois beau et simple reste très complexe et nécessite une interaction plus poussée avec les personnes utilisant l'application de manière régulière. Il faut anticiper et comprendre comment l'utilisateur va utiliser notre application. Dû à ce manque d'interaction avec les utilisateurs, la façon dont nous pensons l'application, n'est pas toujours la plus pratique en termes d'utilisation quotidienne.

Bien que nous étions trois pour réaliser ce projet, le temps a été un facteur limitant concernant l'importance du travail à implémenter. En effet, avec plus de temps, nous aurions pu effectuer une analyse plus poussée et donc éviter certains problèmes menant à des améliorations présentes dans la partie suivante de ce mémoire. Néanmoins, l'avantage d'être à trois permet de toujours avoir une majorité absolue dans la prise de décision.

Sachant que ce projet a de fortes chances d'être repris par d'autres informaticiens, nous avons mis une importance primordiale à la documentation et à la structure du code. Nous pensons que si ce projet est repris par d'autres personnes, par exemple dans le cadre d'une autre mémoire, ceux-ci seront heureux d'utiliser notre projet.

Commencer un projet à partir de rien était un facteur important quant au choix de ce mémoire. Nous voulions avoir notre indépendance par rapport au choix de la technologie, de l'architecture et de la méthodologie de développement. Ceci nous a permis d'apprendre de nombreuses choses concernant le processus de création d'une application.

C'est aussi la première fois que nous réalisions un projet de cette ampleur. Nous avons pu découvrir les différentes interactions existantes entre les acteurs internes, mais aussi avec les acteurs externes, par exemple, le client, durant le développement d'un projet. En effet, nous avons dû gérer notre développement à trois en établissant certaines règles et à l'aide d'une communication simple nous avons pu avancer ensemble. Nous avons établi des itérations d'une semaine. À chaque itération, nous nous réunissions pour définir les objectifs à réaliser avant l'itération suivante. La mise en place des objectifs était aussi liée aux commentaires de nos promoteurs. Une fois l'itération arrivant à son terme, nous discutions de ce qu'il c'était bien passé ou non et redéfinissions de nouveaux objectifs.
Nous avons dû aussi gérer des réunions hebdomadaires avec nos promoteurs. Elles ont joué un rôle important dans la réalisation de ce mémoire. Elles nous ont permis de comprendre les besoins du client, de modifier certaines erreurs commises durant le développement et de rédiger ce rapport final. Nous jugeons que ces réunions ont été constructives et se sont, dans l'ensemble, bien déroulées. Celles-ci ont permis de rendre l'application plus aboutie.

Nous sommes donc fiers de rendre cette application et nous pensons être plus aptes à entamer notre prochain challenge.

Chapitre 9

Améliorations possibles

Dans cette section, nous allons parler des améliorations possibles du projet. Certaines de ces améliorations font suite à des problèmes rencontrés durant le développement, d'autres améliorations proviennent de réflexions faites durant l'utilisation de l'application. Il est important de noter que toutes les améliorations proposées sont faisables dans les technologies choisies.

9.1 Le versionning

Il est possible que plusieurs utilisateurs modifient en même temps la même information (exemple : deux administrateurs modifient le nom de la même station en même temps).
Pour résoudre ce problème, il suffit simplement d'avoir un système de gestion des versions des données : le versionning (ce que la technologie choisie permet).
Il s'agit d'une première solution concernant les modifications simultanées d'une même donnée. Une gestion des versions plus poussée pourrait être mise en place (voir la section "9.4 Corbeille").

9.2 La base de données intermédiaire

Lors de la récupération d'un volume important de données pour l'affichage sous forme de tableau ou de graphique (affichage annuel), le serveur prend un long moment pour calculer et récupérer les données. Ceci est dû au fait que le serveur doit agréger l'ensemble des données avant de les renvoyer.

La solution consisterait à faire une table en base de données contenant les données pré-calculées. Ceci implique qu'à chaque ajout ou modification de données, il faut recalculer le contenu des tables intermédiaires. Cette amélioration permettrait donc de réduire le temps d'attente de l'utilisateur, de pouvoir effectuer le tableau récapitulatif de la page "*Carte*", mais aussi de diminuer la charge de calculs du serveur lors de l'affichage des données.

9.3 Ergonomie

Une autre amélioration consisterait à améliorer le design et le confort d'utilisation. Ce qui prend du temps, temps que nous avons préféré investir dans le développement des fonctionnalités, le débug, la documentation et les tests effectués plutôt que dans le design de l'application.
Une amélioration ergonomique relevée durant la phase de développement serait un moyen de visualiser les données en cours (sous les différents formats présents dans l'application) de validation intégrées aux données déjà présentes.

9.4 Corbeille

La corbeille serait un outil de double vérification de suppression définitive des données. Elle permettrait de restaurer des versions précédentes des données, d'un utilisateur, etc. Elle permettrait aussi de restaurer l'ancienne donnée en cas d'erreur.

9.5 Citizen Science

Les sciences participatives (un habitant ayant une station météorologique chez lui et souhaitant partager ses données) ont été abordées durant la présentation du projet, mais se sont très vite révélées moins utiles que le reste des fonctionnalités. Toutefois, cet angle de vue permettrait d'ouvrir complètement l'outil aux particuliers et permettrait donc d'acquérir davantage de données pluviométriques.

9.6 Des statistiques plus poussées

Certaines statistiques plus poussées pourraient être envisagées dans le domaine de l'hydrologie, mais aussi concernant l'utilisation de cette application.
Une fonctionnalité intéressante pourrait être, par exemple, la possibilité de pouvoir comparer les statistiques entre plusieurs stations.

9.7 Autre

Il serait aussi possible de détourner le but premier de l'application pour centraliser les données relatives à d'autres types de catastrophes telles que les séismes, etc. Il serait aussi possible de déployer l'application pour d'autres pays.

Glossaire

Administrateur Personne de confiance ayant un accès complet à l'application pouvant effectuer des actions qui impacteront directement le bon fonctionnement de l'application. 12

Angular Framework fondé en JavaScript permettant d'implémenter le Front-End d'une application web. 24

BodyParser Parse les requêtes entrantes afin de les faire correspondre aux paramètres donnés. 58

ExpressJs Plugin NodeJS permettant de créer un serveur web souple et robuste. 24

Framework Ensemble de composants et d'outils.. 24

MEAN Mongodb, ExpresJs, Angular, Nodejs. 24

Modele Description abstraite des données. 58

Mongodb Base de données **NoSQL**. 24

NodeJs Plate-forme open source basée sur JavaScript permettant le développement d'applications côté serveur. 24

NoSQL Système de gestion de base de données. 55

OWASP Acronyme pour *Open Web Application Security Project*. Il s'agit d'une communauté en ligne travaillant sur la sécurité des applications web. 48

Station Instrument de mesure des données pluviométriques. 12

Utilisateur Personne accédant à des fonctionnalités avancées de l'application devant s'identifier. 12

Variable d'environnement Variable utilisée pour communiquer des informations au programme suivant une convention de nommage. 47

Visiteur Personne accédant à l'application sans devoir s'identifier. 12

Chapitre 10

Annexes

10.1 Manuel d'installation

10.1.1 Installation du programme

Pour installer la stack Mean, allez voir sur le site mean.io et commencez par installer les prérequis. Ensuite clonez le repository git suivant : https://github.com/doricci/TFE4Haiti. Une fois celui-ci cloné, déplacez-vous dans le dossier et effectuez la commande : *npm install* dans le sous-dossier **serveur** et dans le sous-dossier **client**. Cette commande permet d'installer tous les paquets requis. Une fois ceci effectué, lancez le serveur et le client via la commande *npm start* (il vous faudra donc deux invités de commandes).
Dans les paquets installés, nous avons fait le choix d'utiliser *nodemon*. Ce dernier permet qu'à chaque sauvegarde effectuée le serveur correspondant soit relancé.
Si vous désirez ajouter un paquet, il faut utiliser la commande *npm install <nom du paquet> -g*. L'option -g permet d'ajouter le paquet installé dans la liste des paquets de l'application. Si vous ajoutez un paquet et que d'autres personnes travaillent aussi sur ce projet, lors de la récupération du projet, les personnes devront faire un *npm install* afin de récupérer le paquet que vous avez ajouté.
Pour toutes informations supplémentaires, référez-vous à la documentation de npm.

10.1.2 La configuration

Lors du lancement de l'application, si le fichier de configuration n'est pas trouvé, celui-ci est généré avec des valeurs par défaut. Ces valeurs doivent impérativement être changées pour des raisons de sécurité et pour assurer le bon fonctionnement de l'application. Ce fichier se situe dans le dossier "*server*" du projet et se nomme config.json. Afin de modifier ces valeurs, il faut être sûr de bien comprendre l'utilité de chacun de ces champs.
Dans cette partie, nous allons reprendre tous les champs du fichier de configuration et les expliquer :

- development (Boolean) : Permet de signifier si l'application est en développement ou non (attention, si true ici, les mots de passe ne seront pas cryptés)
- server : Informations relatives au serveur
  - host (String) : Adresse IP ou URL sur laquelle écouter les requêtes entrantes.
  - port (Number) : Port du serveur sur lequel l'application est joignable.
- database : Informations liées à la base de données
  - host (String) : L'adresse de la base de données
  - port (String) : Le port de la base de données
  - name (String) : Le nom de la base de données

    - login (String) : Le login utilisé pour se connecter à la base de données
    - password (String) : Le mot de passe utilisé pour se connecter à la base de données
- token : Informations liées au jeton
    - privateKey (String) : La clé privée utilisée pour générer/vérifier les jetons d'authentification.
    - expiration (String) : La durée de validité/d'expiration du jeton, celui-ci peut être exprimé en millisecondes, sauf si une unité de temps est donnée (P. ex. : "2 days", "10h", "7d", etc.).
- mail : Informations liées au serveur de mail.
    - host (String) : L'adresse du serveur de mail
    - port (String) : Le port du serveur de mail
    - user (String) : L'utilisateur utilisé pour se connecter au serveur mail
    - pwd (String) : Le mot de passe de l'utilisateur utilisé pour se connecter
    - subjectCreationAccOk (String) : Le titre du mail lorsqu'un utilisateur est accepté par l'administrateur
    - subjectCreationAccRefused (String) : Le titre du mail lorsqu'un utilisateur est refusé par l'administrateur
    - secure (Boolean) : liée à l'utilisation de HTTPS
    - changePwd (String) : Le titre du mail lorsqu'un utilisateur demande de changer de mot de passe
- user : information liée à la modification et la mise en place du mot de passe d'un utilisateur
    - accountAcceptedTime (Number) : Durée de validité durant laquelle le lien envoyé à l'utilisateur à la suite de son acceptation par un administrateur est valide
    - changePwdTime (Number) : Durée de validité durant laquelle le lien envoyé à l'utilisateur lorsque celui-ci demande de changer de mot de passe.
- uploadFolder (String) : Dossier public dans lequel les fichiers de données sont importés en attente de validation.
- downloadFolder (String) : Dossier public dans lequel est créé chaque fichier de demande de téléchargement des données d'une station.

10.1.3 Compilation Front-end

L'application front-end étant différente de l'application back-end, il faut donc un moyen de la rendre accessible depuis le même serveur. Afin de rendre l'accès le plus simplifié possible de l'application à un utilisateur lambda, c'est donc la même URL qui fournira l'accès à l'API et à l'application client. Pour ce faire nous avons donc décidé que toutes les URL qui ne commençaient pas par le mot clé *API* seraient renvoyées vers le dossier de l'application client et *API* ferait un appel à l'API.
Lorsque nous sommes en *développement*, Angular nous fournit un mini serveur web qui analyse les modifications et change automatiquement les fichiers *HTML* et *JavaScript* envoyés au client. Mais en production, cela est complètement différent, car seulement un port est disponible vers l'extérieur (en tant normal et par défaut le port 80 ou 443 pour le HTTP et HTTPS). Il nous faut donc compiler les fichiers *TypeScript* pour les rendre lisibles par un navigateur web et pouvoir "fournir" ces fichiers compilés au navigateur web qui les demande.
L'exécution de la commande suivante s'occupera de compiler et de déplacer le contenu généré vers le dossier *Public* du serveur. Cette partie *Public* sera celle envoyée pour toutes les requêtes ne commençant pas par le mot clé *API*.

```
1    ng build --prod --build-optimizer
```

10.2 Explication détaillée de l'architecture back-end

Tout le code lié au serveur se situe dans le dossier *server* du projet. Celui-ci est divisé comme ceci :

— */config/* : Dossier concernant la configuration du serveur. C'est ici que la configuration est gérée, les différentes constantes, le système de log et le système de vérification des jetons d'identification.

— */controllers/* : Dossier reprenant l'ensemble de la logique des actions effectuées suite à l'appel d'une route.

— */models/* : Dossier reprenant l'ensemble des **Modele**s utilisés dans l'application (par les controlleurs).

— */node_modules/* : Dossier généré à l'installation du projet.

— */routes* : Dossier reprenant l'ensemble des routes existantes de l'application ainsi que le système de vérification des droits.

— */package.json* : Fichier utilisé lors de l'installation du projet.

— */app.js* : Fichier lançant le serveur.

10.2.1 Lancement de l'application

Lorsque vous effectuerez la commande permettant de lancer le serveur, c'est le fichier *app.js* qui sera lancé. Celui-ci va effectuer différentes vérifications avant que le serveur ne soit disponible. Tout d'abord, un système de configuration assez poussé a été mis en place. Si le fichier de configuration n'existe pas (au premier lancement de l'application par exemple), celui-ci va être créé en remplissant les champs avec des valeurs par défaut. Dès lors, le serveur s'affichera un message indiquant qu'il faut aller mettre à jour le fichier de configuration (tous les champs de ce fichier sont expliqués dans le *Manuel d'installation*), car certaines valeurs par défaut peuvent poser des problèmes de sécurité. L'administrateur réseau devra donc arrêter le serveur, et aller éditer le fichier de configuration.
Au lancement du serveur, une vérification de la configuration sera effectuée pour être sûre que tous les champs requis soient présents. Si c'est le cas, le serveur affichera un message à l'administrateur lui indiquant que des champs ont étés générés avec des valeurs par défaut et qu'il doit aller les modifier. Tout cela se passe sans arrêter le serveur.
Lorsque le fichier de configuration a été vérifié et est validé, le serveur est configuré. C'est dans cette partie, que nous configurons le **BodyParser**, c'est celui-ci qui va s'occuper de vérifier que les requêtes puissent être correctement traitées par le serveur (pour plus d'informations sur le fonctionnement exact de chaque paramètre, consultez la documentation).
Enfin, après que la configuration du serveur ait été correctement effectuée, nous vérifions que la base de données soit accessible, et donc qu'elle ait correctement été renseignée dans le fichier de configuration.
Le serveur est désormais accessible et est en attente des requêtes HTTP.

10.2.2 Comment ajouter une nouvelle fonctionnalité ?

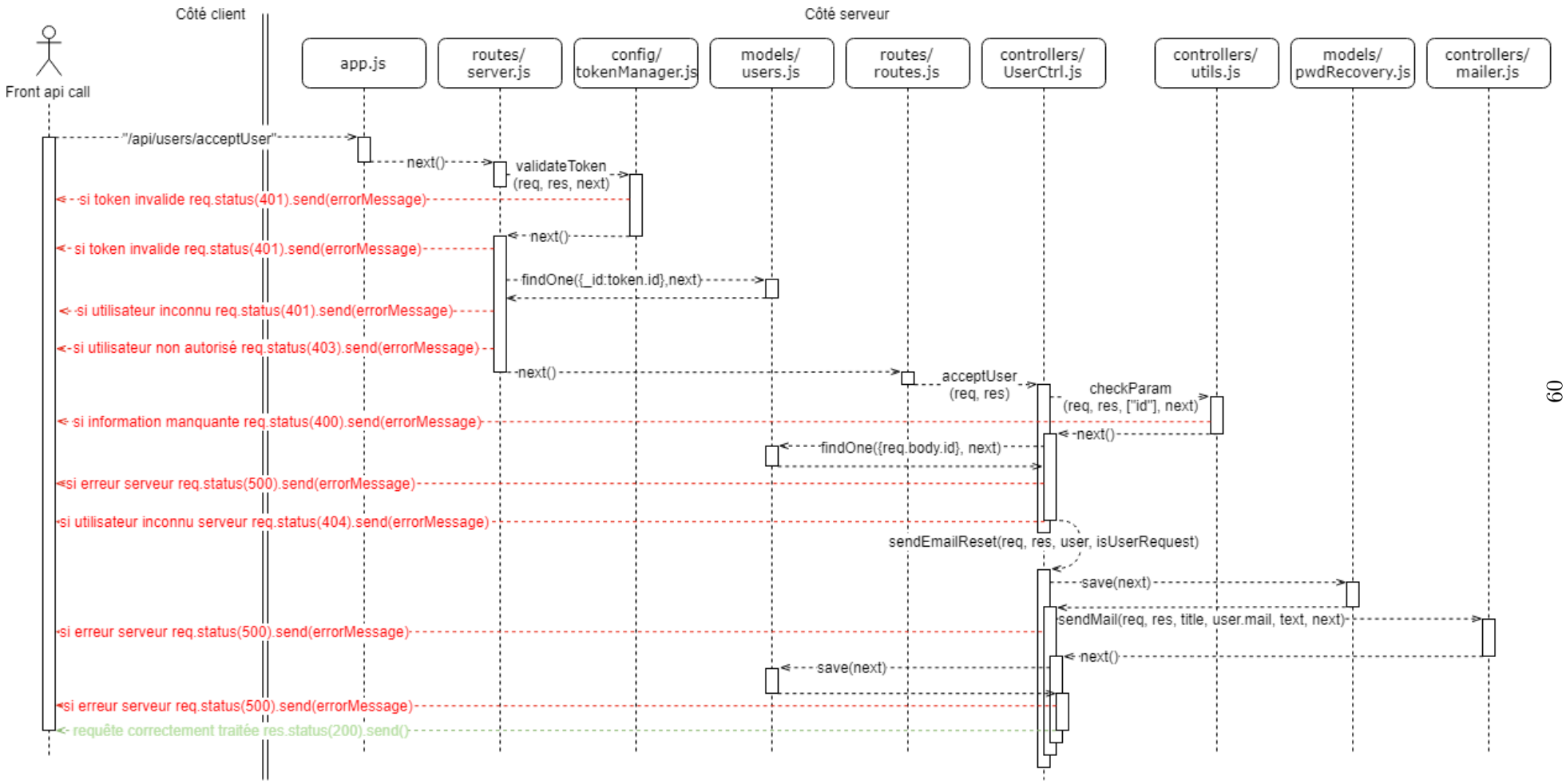
Afin de comprendre comment ajouter une route, et dans le but de vous exposer l'architecture mise en place, nous allons commencer par un exemple.
L'exemple choisi est intentionnellement complexe de telle sorte qu'il puisse montrer le maximum de mécanismes techniques existantes.

Exemple d'un trajet pour la route : "/api/users/acceptUser"

L'exemple suivant se produit lorsqu'un administrateur clique sur le bouton permettant d'accepter un utilisateur.
Plusieurs étapes se produisent :

1. Vérification du format de la requête (*app.js*)
2. Vérification des droits (*server.js* et *tokenManager.js*)
3. Vérification des paramètres requis (*utils.js*)
4. Création de l'objet concernant l'URL permettant à l'utilisateur de créer un mot de passe (*pwdRecovery.js*)
5. Envoi du mail à l'utilisateur avec l'URL lui permettant de créer un mot de passe (*mailer.js*)
6. Changement de l'état de l'utilisateur (*users.js*)

Dans le diagramme suivant, toutes les flèches rouges représentent une erreur. Dès lors la méthode est stoppée et la suite (de la méthode) ne s'exécute pas. C'est pourquoi il est possible qu'un utilisateur ait un objet en base de données lui permettant de créer un mot de passe sans avoir reçu de mail ou bien que le mail lui permettant de créer un mot de passe ne fonctionne pas. Toutefois, ce genre d'erreur est très peu probable comme une erreur serveur (erreur de communication avec la base de données) ou une requête erronée (mauvaise formulation, pas les droits, etc.). Dans le cas où ce genre d'erreurs se produit, l'administrateur, à l'aide du panneau d'administration, aurait très vite fait de rectifier l'incohérence en modifiant l'état de l'utilisateur.


Côté client
Côté serveur
Front api call
app.js
routes/ server.js
config/ tokenManager.js
models/ users.js
routes/ routes.js
controllers/ UserCtrl.js
controllers/ utils.js
models/ pwdRecovery.js
controllers/ mailer.js
"/api/users/acceptUser"
next()
validateToken (req, res, next)
si token invalide req.status(401).send(errorMessage)
next()
si token invalide req.status(401).send(errorMessage)
findOne({_id:token.id},next)
si utilisateur inconnu req.status(401).send(errorMessage)
si utilisateur non autorisé req.status(403).send(errorMessage)
next()
acceptUser (req, res)
checkParam (req, res, ["id"], next)
si information manquante req.status(400).send(errorMessage)
next()
findOne({req.body.id}, next)
si erreur serveur req.status(500).send(errorMessage)
si utilisateur inconnu serveur req.status(404).send(errorMessage)
sendEmailReset(req, res, user, isUserRequest)
save(next)
sendMail(req, res, title, user.mail, text, next)
si erreur serveur req.status(500).send(errorMessage)
next()
save(next)
si erreur serveur req.status(500).send(errorMessage)
requête correctement traitée res.status(200).send()

Ajout d'une nouvelle route

Au vu du diagramme précédent, l'ajout d'une route semble complexe, et pourtant grâce à l'architecture mise en place, cela se résume à l'ajout de cinq lignes dans le code, ainsi que, bien évidemment, le code devant être exécuté lorsque la route est appelée.
La première chose à faire consiste à ouvrir le fichier *routes.js* et ajouter une nouvelle entrée dans le tableau "routes". Cette entrée est composée de trois éléments obligatoires et un élément optionnel :

- path : la route en question. Dans le cas ou une nouvelle route est créée, veuillez respecter la nomenclature existante, à savoir commencer par "/api/" suivi du modèle principal qui va être impacté et enfin le but de la route. Attention à bien insérer votre route avant les routes plus générales telles que "/api/users/ :id", sinon il pourrait y avoir collision, et la méthode utilisée serait alors la méthode de la première route rencontrée dans le tableau.
- httpMethod : les types GET, PUT, POST ou DELETE sont les seuls types supportés.
- middleware : Tableaux reprenant les middlewares devant être appliqués. Le dernier middleware étant la méthode qui va être exécutée à l'appel de la route. Celle-ci doit respecter la nomenclature "controllers." suivie du nom du contrôleur (celui-ci doit être présent dans le tableau des contrôleurs présent au début du fichier), suivi de "." et enfin le nom de la méthode qui va être appelée.
- access (optionnel) : Tableau de rôles ayant accès à la route. Si cet argument est omis, l'architecture mise en place considérera que tout monde a accès à cette route. Si ces arguments sont mis, l'architecture vérifiera automatiquement que l'utilisateur a un jeton valide et qu'il a les droits pour accéder à cette route.

Après avoir ajouté l'entrée dans le tableau des routes, il suffit simplement d'ajouter la méthode dans le contrôleur indiqué dans l'option "*middleware*" comme ceci :

```
1 exports.nom_de_la_methode = function(req, res) {
2     // some code
3 }
```

Premièrement, on observe que le premier mot est "*exports*". Ceci va permettre d'exporter la fonction "*nom_de_la_methode*" et la rendre accessible à n'importe quel module qui importe le fichier dans laquelle la méthode se situe.
Ensuite, on remarque que la fonction comporte deux arguments :

- req : représente la requête de l'utilisateur et contient beaucoup d'informations, mais seront principalement utilisées pour récupérer les paramètres passés dans l'URL (par exemple :id dans la requête "/api/users/ :id", si de tels paramètres existent) ou dans le corps de la requête (lors d'une requête de type POST).
- res : utilisé pour transmettre la réponse au client. Permet de définir le code de retour ainsi que la réponse qui va être envoyée au client. Celle-ci sera traitée par la partie Front-End. Bien évidemment, il faut faire attention à ne renvoyer aucune erreur générée par le serveur, car cela pourrait être la source d'une faille de sécurité.

Pour revenir aux middlewares présents dans l'option *middleware* de routes.js, nous en avons créé un qui va beaucoup simplifier le code. En effet, durant le développement, nous avons remarqué que la vérification d'accès d'un utilisateur à une station (pour la modifier, importer des données, etc.) revenait très souvent. Voici son fonctionnement :

1. Récupération de l'utilisateur via son token
2. Récupération de la station passée dans l'URL ou le corps de la requête et vérification que celle-ci existe.

3. Si l'utilisateur est administrateur, il a accès à toutes les stations, ceci est effectué après la vérification de l'existence de la station, car cela permet de s'assurer que la station existe pour toutes les méthodes après l'appel à cette méthode.
4. Vérification que l'utilisateur a accès à la commune
5. Vérification que l'utilisateur a accès au bassin versant
6. Vérification que l'utilisateur est dans la liste des utilisateurs assignés à cette station.
7. Si l'utilisateur n'est accepté par aucune des vérifications, il n'a donc pas le droit et une erreur 403 (accès refusé) lui est renvoyée.

L'ajout de cette méthode dans la liste des middlewares garantit donc que l'utilisateur ait accès à la station pour toutes les méthodes du tableau présentes après cet appel.

10.3 Comment modifier la liste des communes et des bassins versants ?

Pour modifier la liste des communes ou des bassins versants de l'application, il suffit de modifier un seul fichier : *server/models/station.js*
Dans ce fichier, deux variables sont présentes :

1. communes : la liste des communes
2. bassin_versants : la liste des bassins versants

Ces deux variables sont représentées via un tableau. Attention à bien échapper chaque caractère spécial tel que l'apostrophe. Si vous retirez une commune ou un bassin versant de ce tableau, veuillez mettre à jour la base de données (pour que les stations appartiennent à une commune ou un bassin versant existant dans la liste).
Enfin, relancez le serveur.

Bibliographie

Sites internet

[1] Région wallonne, *Direction générale opérationnelle de la Mobilité et des Voies hydrauliques.* (s.d.). En ligne http://voies-hydrauliques.wallonie.be/opencms/opencms/fr/hydro/Archive/annuaires/index.html, consulté de septembre 2018 à décembre 2018

[2] Paupier F. (24/10/2017), *Développement web en 2018 : serveur ou client ?*. En ligne https://www.appvizer.fr/magazine/services-informatiques/framework/django/developpement-web-serveur-ou-client, consulté le 10 octobre 2018

[3] Google 2010-2018, *Angular*. En ligne https://angular.io/, consulté de septembre 2018 à décembre 2018

[4] Human Rights Watch. (2018). *Haïti Événements de 2017*. En ligne https://www.hrw.org/fr/world-report/2018/country-chapters/313098, consulté le 9 décembre 2018

[5] L'OBS. (13/01/2010 à 10h09) *Haïti, le pays le plus pauvre des Amériques*. En ligne https://www.nouvelobs.com/monde/seisme-en-haiti/20100113.OBS3467/haiti-le-pays-le-plus-pauvre-des-ameriques.html, consulté le 9 décembre 2018

[6] Wikipedia. (2018). *Haïti*. En ligne https://fr.wikipedia.org/wiki/Haïti, consulté le 9 décembre 2018

[7] Wikipedia. (17/10/2018). *Communes d'Haïti*. En ligne https://fr.wikipedia.org/wiki/Communes_d%27Ha%C3%AFti, consulté le 20 octobre 2018

[8] Wikipedia. (27/08/2017). *Rivières d'Haïti*. En ligne https://fr.wikipedia.org/wiki/Liste_des_cours_d%27eau_d%27Ha%C3%AFti, consulté le 20 octobre 2018

[9] OpenClassrooms. (s.d.). *Des applications ultra-rapides avec Node.js*. En ligne https://openclassrooms.com/fr/courses/1056721-des-applications-ultra-rapides-avec-node-js, consulté le 03 octobre 2018

[10] Wikipedia. (2018). *Comparison of programming languages*. En ligne https://en.wikipedia.org/wiki/Comparison_of_programming_languages, consulté le 03 octobre 2018

[11] Tripier F. (13/06/2017). *NodeJS vs Java EE pour une application web en 2017*. En ligne https://blog.axopen.com/2017/06/nodejs-vs-java-ee-application-web-2017/, consulté le 03 octobre 2018

[12] OpenClassrooms. (s.d.). *Découvrez le fonctionnement de MongoDB*. En ligne https://openclassrooms.com/fr/courses/4462426-maitrisez-les-bases-de-donnees-nosql/4474601-decouvrez-le-fonctionnement-de-mongodb, consulté le 03 octobre 2018

[13] CNIGS. (s.d.). *Les bassins versants*. En ligne http://cnigs.ht/pitdd/index.php?option=com_content&view=article&id=20&Itemid=8, consulté le 03 décembre 2018

[14] OpenClassrooms. (s.d.). *Node.js : mais à quoi ça sert ?*. En ligne https://openclassrooms.com/fr/courses/1056721-des-applications-ultra-rapides-avec-node-js/1056866-node-js-mais-a-quoi-ca-sert, consulté le

[15] *MIT License (Expat)*. (2014). En ligne https://tldrlegal.com/license/mit-license, consulté le 26 décembre 2018

[16] CariWatNet. (s.d.). *Project CariWatNet*. En ligne https://www.wur.nl/nl/show/CariWatNet.htm, consulté le 03 décembre 2018

[17] Actu-Environnement. (s.d.). *Hydrologie*. En ligne https://www.actu-environnement.com/ae/dictionnaire_environnement/definition/hydrologie.php4, consulté le 03 décembre 2018

[18] Eve, C. (1993). *Les précipitations*. En ligne http://www.meteo.org/phenomen/pcpn.htm, consulté le 23 décembre 2018

[19] Google 2010-2018. (s.d.). *What is Angular ?*. En ligne https://angular.io/docs, consulté le 10 novembre 2018

[20] Wikipedia. (20/12/2018). *Pluviomètre*. En ligne https://fr.wikipedia.org/wiki/Pluviom%C3%A8tre, consulté le 02 janvier 2019

[21] tpepluviometre. (s.d.). *I – Présentation du pluviomètre*. En ligne http://tpepluviometre.e-monsite.com/pages/i-presentation-du-pluviometre.html, consulté le 02 janvier 2018

[22] Wikipedia. (20/10/2018). *Isohyète*. En ligne https://fr.wikipedia.org/wiki/Isohy%C3%A8te, consulté le 02 janvier 2019

Images

[23] Google 2010-2018. (s.d.). *Architecture overview*. En ligne https://angular.io/guide/architecture, consulté 10/12/2018

[24] DeBill E. (s.d.) *Module Counts*. En ligne http://www.modulecounts.com, consulté le 10 novembre 2018

[25] Chopin S.(9/02/2013 - Mis à jour le 20/02/2013). *Utiliser MongoDB avec Node.js grâce à Mongoose*. En ligne https://atinux.developpez.com/tutoriels/javascript/mongodb-nodejs-mongoose, consulté le 10 novembre 2018

[26] Quest, C. (2018). *Don't be evil... until...*. En ligne Qhttps://medium.com/@cq94/dont-be-evil-until-95f2e8dfaaad, consulté le 23 décembre 2018

[27] The Apache Software Foundation. (2018). *Creating layouts using markup inheritance*. En ligne https://wicket.apache.org/learn/examples/markupinheritance.html, consulté le 27 décembre 2018

[28] Wikimedia Foundation. (2010). *Pluviomètre*. En ligne http://fracademic.com/dic.nsf/frwiki/1347924, consulté le 02 janvier 2018

[29] Med-Hycos. (s.d.). *LA PLUVIOMETRIE*. En ligne http://medhycos.mpl.ird.fr/en/data/hyd/Drobot/3C.htm, consulté le 02 janvier 2018

[30] Coopération Technique GEF/UNEP No GF/2200-97-16/97-49 (2001). *PREMIERE COMMUNICATION NATIONALE SUR LES CHANGEMENTS CLIMATIQUES*. En ligne https://unfccc.int/sites/default/files/resource/1er%20communication%20nationale.pdf, consulté le 02 janvier 2019

[31] Soares-Frazao S. (2018). *Basics of Hydrology*. Document non publié, Université catholique de Louvain, Louvain-la-Neuve.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**