



TRAVAIL DE FIN D'ÉTUDES

Implémentation d'un Système de Contrôle Domotique

Travail réalisé par Tom Barbette
2^e Master en Sciences Informatiques à finalité approfondie

Année académique 2012-2013

Promoteur : Pr. Benoit Donnet

Jury : B. Boigelot (Président), Laurent Mathy, Quentin Louveaux

Travail de fin d'études présenté en vue de l'obtention du grade de Master en Sciences Informatiques à finalité approfondie

Résumé

Ce document présente le fonctionnement, le contexte, et les perspectives de la réalisation d'une interface de contrôle domotique. L'interface est flexible, pouvant s'adapter à toutes les énergies et tous les types de compteurs. Elle a été pensée pour une utilisation facile, guidant l'utilisateur par étape dans la configuration de l'interface, tout en permettant une définition relativement précise de l'enveloppe énergétique de la maison. Elle répond également à un souci de performance devant être utilisable sur les ordinateurs mais aussi sur les smartphones et tablettes. Par la combinaison de ces caractéristiques, l'interface se démarque par rapport à ce qui se fait actuellement en la matière.

Ce travail envisage également quelques perspectives de recherche liée à l'interface, comme l'utilisation des informations fournies par les divers appareils intelligents de la maison pour calculer des mesures approximées plutôt que d'utiliser des capteurs physiques, souvent coûteux.

Abstract

This paper presents the realization, the context, and the prospects of an automation control interface. The interface is flexible, capable of adapt to all energy and all types of meters. It has been designed for easy use, guiding the user by steps in the configuration of the interface, while allowing a relatively precise definition of the energy envelope of the house. It also responds to performance issues, allowing it to be used on computers but also smartphones and tablets. By combining these characteristics, the interface differs from what is currently being done in this area.

This work is also considering some research perspectives related to the interface, such as the use of information provided by the various smart devices in the house to compute approximated consumption data, rather than using real meters, which are often expensive.

Interface de démonstration et code source

Une version de démonstration de l'interface est disponible à l'adresse <http://www.usemonitoring.be>. Le code source est téléchargeable à l'adresse <http://www.usemonitoring.be/static/source.tar.gz>. Une version électronique de ce rapport est également disponible à l'adresse <http://www.usemonitoring.be/static/rapport.pdf> pour votre meilleure convenance. Veuillez noter que comme précisé plus loin dans ce travail, les données des compteurs visibles sur l'interface de démonstration ne sont pas des données réelles et les anciennes données ne sont que très disparates provoquant des erreurs uniquement dues au fait que certains relevés ont été envoyés à un stade avancé de l'interface. Elles sont cependant laissées pour que vous puissiez voir, cher lecteur, des aperçu des statistiques disponibles.

Le nom d'utilisateur est "tfe" et le mot de passe est "tombarbette2013".

Remerciements

Je tiens particulièrement à remercier mon promoteur, le professeur Benoit Donnet pour l'aide apportée, spécialement quant aux conseils concernant la rédaction.

Je désire également remercier l'équipe d'Ingénieurs de Projet qui a encadré le projet dans son ensemble. Elle a apporté à la réalisation de ce TFE, une expérience très particulière dans le cadre d'un travail en équipe pluridisciplinaire, avec ses forces et ses fragilités. Cela m'a permis de vivre une expérience unique au regard des travaux de fin d'études abordés de façon plus classique.

Je tiens aussi à remercier ma famille pour l'aide apportée au niveau de la relecture de ce manuscrit.

Enfin, je remercie l'ICEDD pour son autorisation de reproduction des figures du bilan énergétique de la Région Wallonne.

Table des matières

1	Introduction	5
2	Système d'acquisition et objectifs	9
2.1	Acquisition	12
2.1.1	Gaz et eau	12
2.1.2	Électricité et température	14
2.2	Etude de marché	16
2.2.1	Smartbox	16
2.2.2	Open Energy Monitor	17
2.2.3	Efergy Elite	17
2.2.4	Wattvision	17
2.2.5	EKM Metering	17
2.2.6	Chacon ecowatt	18
3	Choix logiciels	19
3.1	Langage : Python	19
3.2	framework : Django	21
3.2.1	Sécurité dans Django	23
3.2.2	Base de données	24
3.2.3	Technologies Web	24
4	Structure de l'interface	27
4.1	Applications	27
4.2	Structure de la base de donnée	28
4.2.1	Structure de la maison	30
4.2.2	Alertes	31
4.2.3	Authentification	32
4.3	Transfert de données entre Django et les applications Javascript . . .	33
4.4	Système de dessin du plan	37
4.5	Protocole de communication	40
4.6	TraITEMENT des données	42
4.6.1	Pre-processing	42
4.6.2	Post-processing	43
4.7	Système d'alertes	44
4.8	Compression et concaténation du texte	45

5 Compteurs virtuels	47
5.1 Génération de données aléatoires	47
5.2 Température	49
5.3 Consommation d'un ordinateur	49
6 Résultats	55
6.1 Modélisation - <i>Builder</i>	56
6.2 Consommation - <i>Consumption</i>	63
6.3 Alertes - <i>Alerts</i>	65
6.4 Statistiques - <i>Statistics</i>	66
6.4.1 Recherche en matière énergétique	66
6.5 Paramètres - <i>Settings</i>	68
6.6 Compatibilité et smartphones	70
6.7 Modularité	72
6.7.1 Ajout d'une énergie et d'un nouvel appareil de consommation	73
6.8 Performances	75
6.8.1 Base de donnée	75
6.8.2 Tailles de transferts et temps de rendu	77
6.8.3 Évaluation quantitative	80
7 Perspectives et travail futur	81
7.1 Alertes	81
7.2 Statistiques et suppression des relevés	81
7.3 Image des appareils	82
7.4 Application SmartPhone native	82
7.5 Modèle énergétique	82
7.6 Interfaçage avec un système domotique existant	84
7.7 Présence des habitants	84
8 Conclusion	87
,	

Chapitre 1

Introduction

Depuis la révolution industrielle, l'humanité ne cesse d'inventer de nouvelles machines. De la voiture à l'électroménager, de l'ampoule à l'ordinateur, nous produisons sans cesse de nouveaux objets pour satisfaire de nouveaux besoins, et de nouveaux besoins pour ensuite les satisfaire avec de nouvelles inventions. La consommation des ménages a ainsi augmenté de 12% en 15 ans, en grande partie dûe à une hausse de plus de 50% de la consommation d'électricité [23] poussée par une hausse de plus de 50% du pouvoir d'achat ces trente dernières années[12]. Un cours d'économie basique nous indique que le marketing consiste à promouvoir un produit pour en vendre plus. Ce même cours d'économie nous apprendra ensuite le rapport existant entre l'offre et la demande. Lorsqu'il n'y a pas assez d'offre pour la demande, le prix augmente. Multiplier la quantité de besoins ne serait pas un problème si la terre s'agrandissait à une vitesse proportionnelle à l'utilisation de ses ressources. Malheureusement, notre planète n'est pas en expansion et la presse nous informe quasi quotidiennement de l'augmentation des prix de l'énergie - tantôt du gaz, tantôt de l'essence, tantôt de l'électricité.

Le siècle dernier a connu une forte industrialisation. Celle-ci en lien avec l'évolution démographique a entraîné une prise de conscience à un niveau planétaire : le monde que nous habitons a ses limites, même si cette prise de conscience ne suffit pas encore toujours à provoquer les changements nécessaires à la lutte contre le réchauffement climatique (Echec de Copenhague en 2009[32] ou encore retrait du Canada de Kyoto en 2011[33]). Cette prise de conscience ne date pas d'hier. Juridiquement la grande Bretagne en fût sans doute pionnière avec le Smoke Nuisance Abatement Act de 1853 [6]. En 1970, cette prise de conscience s'affinait, avec notamment l'établissement de la journée internationale de la terre [52] et l'année suivante la naissance de greenpeace [55]. Le réchauffement climatique, la pollution, la consommation à tort et à travers sont clairement les ingrédients qui entrent en ligne de compte dans le problème majeur auquel doit faire face notre société si elle désire passer au siècle suivant. En d'autres termes, la gestion de notre consommation est un défi majeur au regard duquel l'histoire ne peut nous aider. Un défi à relever en recherchant des solutions dans l'innovation.

Que ce soit par souci écologique ou par souci économique, actuellement la plupart des gens souhaitent diminuer leur consommation. Le politique passe progressivement de l'aide à la production à l'aide à la réduction de consommation. En Belgique, on notera notamment la suppression de la prime à l'achat de panneau photovoltaïque [39] qui a laissé place aux primes à l'isolation. En effet, l'énergie la moins chère (et la moins polluante, selon les sensibilités) est celle que l'on ne consomme pas.

Plusieurs études ont montré que la simple connaissance de sa consommation par l'utilisateur a un impact non-négligeable sur la consommation globale. Nous noterons par exemple l'étude de Kohlenberg, R et al.[29] qui pointe un des principes de consommation qui cause le plus de pollution : les pics. En effet, pour pouvoir fournir de l'électricité à tous les foyers aux heures de pointes, il est nécessaire de pouvoir fournir une énergie égale au maximum du pic de consommation. L'énergie étant difficilement stockable, il est nécessaire de construire de nouvelles centrales ou d'en faire tourner de plus anciennes et plus polluantes (souvent au charbon) plusieurs mois par an toute la journée mais seulement pour ces quelques heures de pointes par jour. L'étude montre que la simple information des utilisateurs réduisait l'effet de pic de 50%.

Quant aux émissions de CO₂, si des efforts d'efficacité énergétique étaient entrepris d'ici 2030, nous pourrions diminuer de 23% notre consommation par rapport à un scénario suivant les consommations actuelles [31]. Le scénario prévoit principalement une amélioration de l'isolation, sachant que la consommation dans les bâtiments résidentiels en Belgique se situe environ 70% au dessus de la moyenne européenne.

Au niveau des bâtiments résidentiels, une action significative ne pourra être prise en compte que moyennant une implication en terme d'investissement et de conscientisation des utilisateurs. Un moyen efficace pour aider la population à réduire sa consommation d'énergie, et ne demandant qu'un investissement limité de la part des consommateurs, consiste à équiper les ménages des outils nécessaires à une visualisation et à une maîtrise de leur consommation.

Plusieurs possibilités s'offrent à l'utilisateur pour trouver les failles liées à sa consommation :

- quant à l'isolation de sa maison, il peut faire appel à un professionnel pour procéder à un audit énergétique de sa maison (ce qui est rendu obligatoire pour la vente d'un bien immobilier).
- en ce qui concerne les appareils électroménagers, il peut se fier au classement énergétique des appareils.
- il peut aussi intégrer une installation de domotique, permettant d'offrir un profil de consommation électrique. La domotique désigne l'ensemble des techniques, souvent électroniques, permettant de centraliser le contrôle et la surveillance des appareils ménagers, la gestion de l'énergie et les systèmes d'alarmes.

Mais ces solutions sont coûteuses, ou peu flexibles. Au delà des bonnes pratiques,

vaut-il mieux isoler un mur, acheter un meilleur frigo, changer le système de chauffage ou installer un panneau solaire ?

Pour tenter de répondre à ces questions, j'ai pris part à la réalisation d'un projet collectif dans le cadre du concours Ingénieur de Projet (IdP) réunissant des étudiants de plusieurs disciplines (Electro-Mécanique, Electronique et Informatique). Nous avons construit un système abordable tant du point de vue accessibilité que financier. Celui-ci permet d'étudier la consommation d'un ménage et de sensibiliser ses habitants en fonctions de leurs propres consommations. En effet une étude du SEREC sur les changements d'habitudes des Belges dans leur consommation montre parmi plusieurs dispositifs essayés, les plus efficaces étaient ceux dont la communication était personnalisée. Ainsi plus l'utilisateur reçoit une information individualisée, plus il est enclin à agir sur ses mauvaises habitudes de consommation et à investir [21, p. 133].

Le système dans son ensemble prévoit une série de capteurs et une interface. Nous avons choisi d'appeler l'ensemble du système "USE Monitoring" pour "ULg Smart Energy Monitoring". Un premier type de capteurs est fait à base de vision par ordinateur : des webcams sont placées devant les compteurs d'eau et de gaz, lisent les chiffres du compteur et l'envoient au serveur. Le deuxième type de capteur mesure les consommations électriques de la maison grâce à des pinces ampèremétriques et un circuit électronique, et la consommation de deux appareils électriques de la maison. Ces derniers ont également la possibilité d'être coupé à distance. Toutes ces données sont traitées par le serveur et affichées sur une interface. L'interface permet de visualiser la consommation actuelle de la maison, mais aussi quelques statistiques sur les consommations passées. Les utilisateurs verront les défauts de leurs habitudes de consommation et seront alertés dans certains cas qu'ils auront défini, comme par exemple le dépassement d'un certain seuil de consommation. L'interface a été particulièrement adaptée pour fonctionner sur tout type d'écran, disposant d'une souris ou non. Dès lors l'utilisateur pourra être averti s'il oublie un appareil allumé ou simplement détecter l'oubli en se connectant à l'interface via son smartphone par exemple.

L'interface permet la modélisation de la maison, afin d'intégrer un éventuel futur modèle énergétique qui sera discuté dans les perspectives. Actuellement, cela n'a donc d'utilité que pour l'affichage des appareils et des compteurs sur le plan de la maison de l'utilisateur. J'ai également développé une série de compteurs "virtuels". En effet, la température extérieure n'a par exemple, pas besoin d'un capteur réel et peut directement être récupérée en ligne. J'ai également développé un script tournant sur un ordinateur et envoyant des données de consommation simulée. En effet, la consommation d'un ordinateur peut être relativement correctement estimée et l'ordinateur peut évidemment utiliser sa connexion internet pour envoyer ces données simulées. J'étudie également quelques autres cas possibles dans la suite de ce travail. En effet, nos appareils électroménagers deviennent de plus en plus intelligents et il est évident que toute mesure pouvant être faite sans l'ajout d'un quelconque

capteur est utile.

L'idée d'une interface de monitoring énergétique dans le cadre résidentiel n'est pas neuve, et plus globalement le domaine de la domotique fait déjà l'objet de plusieurs recherches. Cependant comme nous le verrons dans l'étude de marché, il n'existe pas de système combinant la configuration de l'enveloppe énergétique de la maison, la surveillance de la consommation électrique mais aussi d'eau, de gaz, de mazout et plus globalement de toutes énergies. De plus ils ne permettent que rarement le monitoring aussi bien au niveau global, sur la consommation de toute la maison, qu'au niveau local sur un appareil ciblé. Enfin, seuls certains produits permettent le contrôle et l'extinction des appareils. Ces systèmes ont chacun isolément d'autres avantages, mais l'interface telle que présentée dans ce travail est la seule à réunir toutes ces caractéristiques et particulièrement la seule à être aussi flexible.

Ce document présentera en premier lieu le système dans son ensemble, avec notamment une description des capteurs construits par les autres membres d'Ingénieur de Projet, devant fournir les données à notre interface (le protocole de communication étant à ma charge également). Le cahier des charges prévoit le fonctionnement avec ces capteurs, mais l'interface est modulable et comme démontré dans les parties compteurs virtuels et perspectives, elle est très flexible et peut se voir greffée tout type de capteurs pour plusieurs types d'énergies. Le chapitre se termine par une étude des produits existants.

Le chapitre suivant traite des choix logiciels, du langage utilisé et du framework. Il est suivi de la description des structures internes, et de certains algorithmes plus complexes méritant une attention particulière. Il s'en suivra l'étude des compteurs virtuels et la présentation des résultats, comprenant notamment une analyse de la performance. Le chapitre 7 présentera les perspectives du travail et quelques spéculations sur les diverses intégrations d'autres produits imaginables.

Chapitre 2

Système d'acquisition et objectifs

Ce chapitre présentera le fonctionnement général du système d'acquisition pour lequel l'interface a été prioritairement conçue. Bien que mon travail de fin d'étude vise uniquement l'interface, elle a largement influencé la conception des capteurs et inversément. Comme nous le verrons, ce système est cependant destiné à s'adapter à tous types de capteurs et tous types de bâtiments. Une partie de mon travail était donc d'aider à la conception de ces capteurs (mais non la réalisation). Le chapitre se clôturera par une étude de marché couvrant ce qui existe déjà comme systèmes de monitoring énergétiques.

L'ensemble du système poursuit trois objectifs : il se veut flexible, peu coûteux, et non-intrusif. La flexibilité concerne la possibilité d'utiliser d'autre capteurs, et de facilement s'adapter à d'autres maisons, comprenant d'autres appareils ou consommant d'autres énergies comme le mazout. Le système doit être peu coûteux en prix, mais également en énergie. Il faut évidemment que le gain permi par l'information de l'utilisateur sur sa consommation soit plus grand que l'énergie consommée par le système. Enfin, il n'est pas acceptable de demander à l'utilisateur de modifier son installation électrique ou de changer son compteur, le système devant donc être non intrusif.

L'ensemble de USE Monitoring est divisé en 3 parties :

- L'acquisition des mesures par webcam (gaz et eau)
- L'acquisition des mesures électriques
- L'interface

L'acquisition se fait par des caméras qui observent les compteurs de gaz et d'eau. Ces caméras sont connectées à un ordinateur. Comme les compteurs sont souvent dans une cave, elles seront connectées à un mini-ordinateur ARM : un Raspberry Pi (Fig. 2.2).

Le Raspberry Pi est un ordinateur de 30€, un peu plus grand qu'une carte de crédit. Il est fourni avec une distribution linux : Raspbian, basée sur Debian [45]. Il dispose de deux ports USB, d'un port HDMI, d'un port ethernet et d'un

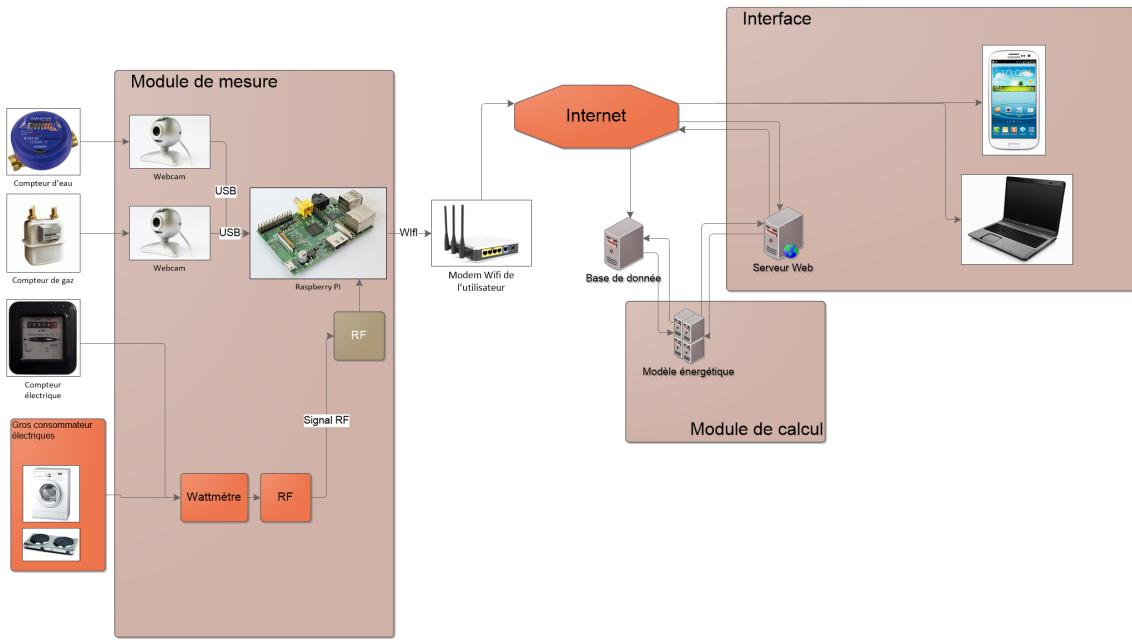


FIGURE 2.1 – Vue globale du système

emplacement pour une carte SD. Il est donc le candidat idéal pour centraliser les informations de plusieurs capteurs et les envoyer à un serveur distant sans devoir ré-implémenter les divers protocoles réseaux, tout en permettant de se baser sur la robustesse des systèmes Unix et faire tourner directement des scripts Python sur celui-ci.

Même si elle peut également se faire par webcam, la mesure de la consommation électrique globale se fait par un wattmètre, connecté également au Raspberry Pi.

Afin d'étendre les possibilités du système et de montrer sa flexibilité, il possède également deux wattmètres sans-fil pour rapporter la consommation précise de deux appareils électriques et offrir la possibilité de les couper. Deux capteurs de températures sans-fil font également partie de l'ensemble.

Le Raspberry Pi envoie les données à un serveur de stockage. Notons que ce serveur pourrait être sur le Raspberry lui-même. Il s'agit ici d'opérer un choix de séparation des éléments : le serveur peut gérer et stocker les statistiques de consommation de plusieurs maisons. De plus, l'affichage de ces données se fait par un serveur web qui présentera des difficultés d'accès s'il est hébergé chez un particulier derrière un routeur disposant d'une adresse IP dynamique, d'autant plus si celui-ci fait du NAT.

Dans le projet original, l'ensemble devait comporter un modèle énergétique utilisant l'enveloppe énergétique de la maison définie via l'interface et les mesures faites par les capteurs pour établir une série de prédictions. Le modèle a cependant dû être abandonné en cours d'année, mais l'interface garde quand même sa partie sur la



FIGURE 2.2 – Raspberry PI

configuration de l'enveloppe de la maison pour une éventuelle future intégration qui sera discutée dans le chapitre 7 perspectives et travail futur.

L'interface web a deux rôles majeurs. Elle permet d'afficher de façon conviviale et compréhensible les données générées par le modèle (Fig. 2.1), notamment sous forme de graphiques. Mais elle sert également à paramétriser la maison de l'utilisateur en lui demandant de répondre à une série de questions ayant trait au type de maison (appartement, habitation mitoyenne, date de construction, etc...) et de tracer un schéma très simplifié de l'habitation. L'interface doit également être capable de s'adapter à tous les types de maisons. Elle doit permettre l'ajout d'éventuels futurs capteurs qui ne seraient pas pris en compte ici, comme un capteur sur des panneaux solaires, la mesure du mazout consommé par une chaudière, ou le placement d'une pompe à chaleur.

Notons ici la modularité du système : il fonctionne aussi bien pour un utilisateur ne possédant que deux webcams (gaz et eau) et le module wattmètre (électricité), que pour l'utilisateur disposant de trois webcams pour chacun des compteurs. De même, l'utilisateur n'est nullement obligé d'activer un Raspberry Pi et peut faire tourner le collecteur de données sur un vieux ordinateur qu'il met au fond de sa cave. Le serveur peut se trouver également sur celui-ci sans aucun problème. Par souci de simplicité, nous utiliserons comme exemple le schéma de la figure 2.1, où le Raspberry PI est utilisé comme collecteur de données, chargé de les envoyer au serveur, qui lui est un ordinateur distinct.

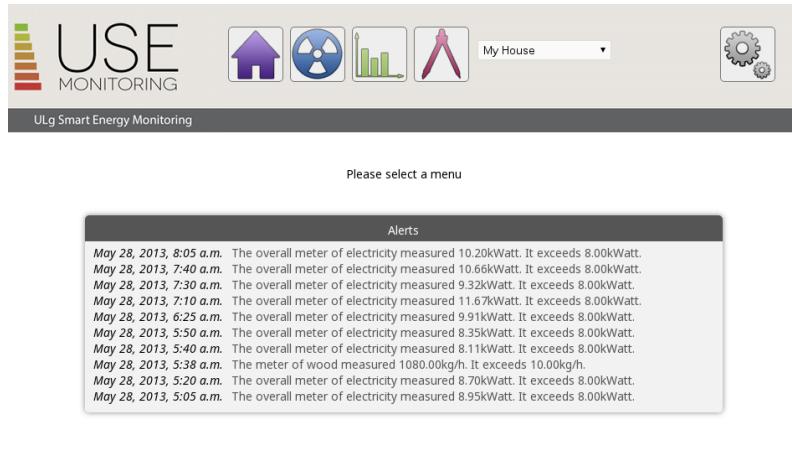


FIGURE 2.3 – Photo de l'interface

2.1 Acquisition

2.1.1 Gaz et eau

Obliger un utilisateur à installer un compteur de gaz numérique est évidemment très compliqué, voire dangereux. De même, le placement d'un compteur d'eau générera des travaux de plomberie importants et coûteux. Ces types de compteurs sont d'ailleurs très chers. La solution proposée ici est donc de prendre en image le compteur existant que tout un chacun possède, décoder les chiffres grâce à un OCR (Optical Character Recognition - logiciel de reconnaissance de caractères) et transmettre cette information au serveur.



FIGURE 2.4 – Système de capture directement collé au compteur

Les étudiants en électro-mécanique chargé de cette partie de USE Monitoring ont utilisé une webcam USB (Fig . 2.4) qu'ils ont fixée dans un boîtier en détournant l'alimentation USB pour alimenter quelques LEDs servant à illuminer le compteur. Le tout vient donc se brancher sur le Raspberry PI. Si nous avons choisi de démonter une webcam USB, ce n'est pas par facilité mais parce qu'il est toujours plus coûteux

de la fabriquer soi-même sur un PCB (Printed Circuit Board - Circuit imprimé), à moins d'en produire plusieurs centaines voir milliers.

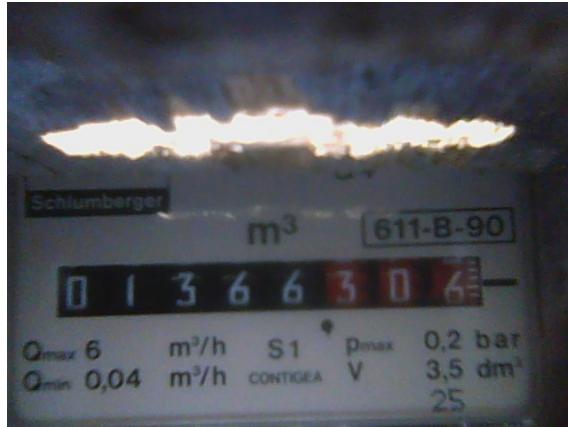


FIGURE 2.5 – Photo telle que prise par la webcam. On aperçoit le reflet des LEDs en haut de la photo



FIGURE 2.6 – Image des numéros après filtrage

Le procédé se déroule en 8 étapes :

- Prise de la photo de 640x480 par la webcam logitech c210 (Fig. 2.5)
- Identification de la zone intéressante de l'image
- Identification du nombre de numéros à lire
- Application d'un premier filtre pour faire ressortir les parties de l'image les plus claires
- Application du 2ème filtre pour éliminer le bruit(Fig. 2.6)
- Centrage de l'image (Fig. 2.7)
- Comparaison des numéros aux templates
- Envoi des numéros au serveur (le protocole est discuté dans la section 4.5 sur le protocole de communication.)



FIGURE 2.7 – Image des numéros après filtrage

Cette série d'événements est implémentée à l'aide du Python (le langage sera discuté dans la section 3.1). Le script est déclenché toutes les deux minutes via le service Cron de la distribution. Cron est un programme qui permet aux utilisateurs

d'exécuter des scripts automatiquement à un interval défini [11].

Le système d'acquisition possède une résolution maximale de $0,01 \text{ m}^3$, le dernier chiffre étant trop mouvant pour être lisible. Le traitement logiciel doit encore être amélioré car il prend 20 secondes.

2.1.2 Électricité et température

Les compteurs électriques ne possèdent pas une grande précision, dès lors il était intéressant d'avoir un wattmètre mesurant plus précisément la consommation globale de la maison. Afin d'étendre les possibilités de notre projet, nous avons intégré une équipe d'électroniciens qui en plus du wattmètre à placer sur l'arrivée électrique générale de la maison ont fabriqué deux capteurs de températures et deux wattmètres pour des appareils électro-ménagers spécifiques qui intègrent également des relais ayant la faculté de couper les appareils à distance.

Cette partie se présente sous la forme d'un boîtier contenant une carte mère (cP sur le schéma 2.8) qui se connecte au Raspberry PI. Les autres éléments communiquent par radiofréquence avec la carte mère.

Trois modules différents sont visibles sur la figure 2.9 :

- Le module mT mesure la température et renvoie les données à la carte principale cP.
- Le module mWcc transmet une mesure de la consommation électrique d'un appareil à cP (ce module se branche entre la prise électrique et l'appareil) tout en donnant la possibilité à l'utilisateur de couper l'alimentation à distance.
- cP est la carte mère, qui sera branchée directement sur le Raspberry PI. Elle possède également 3 pinces pour mesurer l'intensité électrique des 3 phases de courant à l'arrivée de la maison. Cette carte sera donc également à côté du compteur électrique.

Chacune des cartes est contrôlée par des microprocesseurs PIC.

Pour les mWcc, ces derniers utilisent un capteur de courant industriel et la tension est lue au moyen d'un transformateur par un des convertisseurs analogique-digital du pic. Il suffit alors de multiplier la tension par le courant pour obtenir la puissance consommée. La coupure du courant se fait elle grâce à un triac et son optocoupleur.

Pour le capteur de puissance principal, le système est identique si ce n'est que des pinces sont utilisées plutôt que des bornes d'entrée-sortie. En effet, le système se voulant non-intrusif, il n'était pas acceptable ni légal de demander à l'utilisateur de couper son arrivée de courant pour dénuder les fils et faire traverser le courant par la carte.

Concernant la température, le PIC est branché à un simple capteur dont la tension varie en fonction de la chaleur. Il suffit donc de convertir la tension avec un port ADC (Analog to Digital Converter - un port pour convertir une valeur analogique en un signal digital de 10 bits) du PIC.

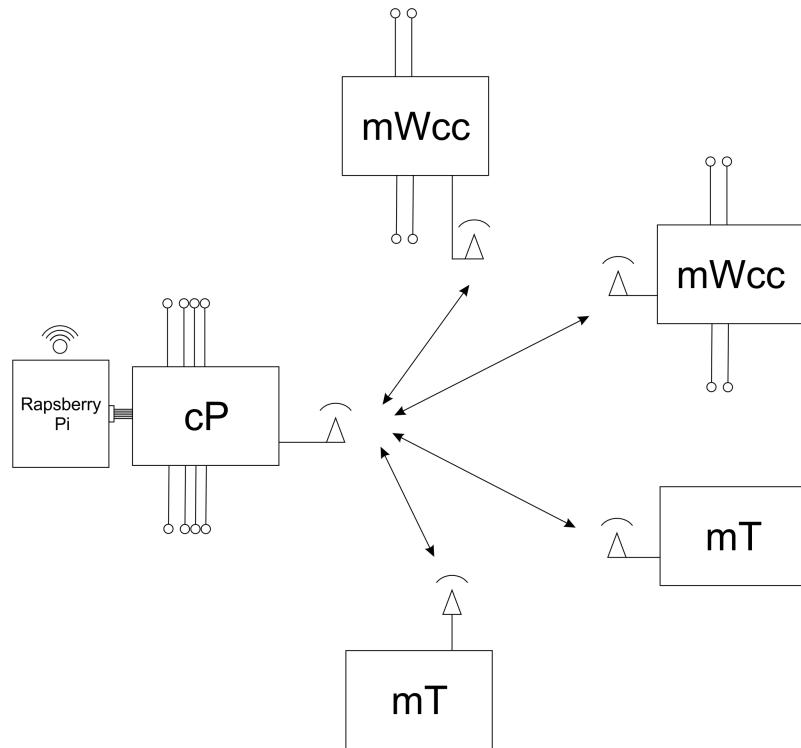


FIGURE 2.8 – Schéma général de la communication des modules de mesure électrique et thermique

Pour la communication radio, les électroniciens ont utilisé des cartes XBee [58], qui sont des modules très faciles à utiliser avec les PIC qui analysent les données des capteurs. Pour communiquer les données de chacun des modules à la carte mère, un multiplexage temporel est utilisé. La carte mère envoie un signal toutes les minutes indiquant quel module a le droit d'utiliser le canal. Chaque XBee a donc droit de parole tour à tour pour envoyer ses données à la carte mère. Lorsque la carte mère a reçu tous les résultats, elle communique ceux-ci à l'ordinateur via un port série géré par une puce MAX232.

De la même façon que pour les capteurs liés à la webcam, les données collectées par la carte mère qui ont été transmises au Raspberry Pi sont envoyées au serveur par la méthode discutée dans la section 4.5 sur le protocole de communication.

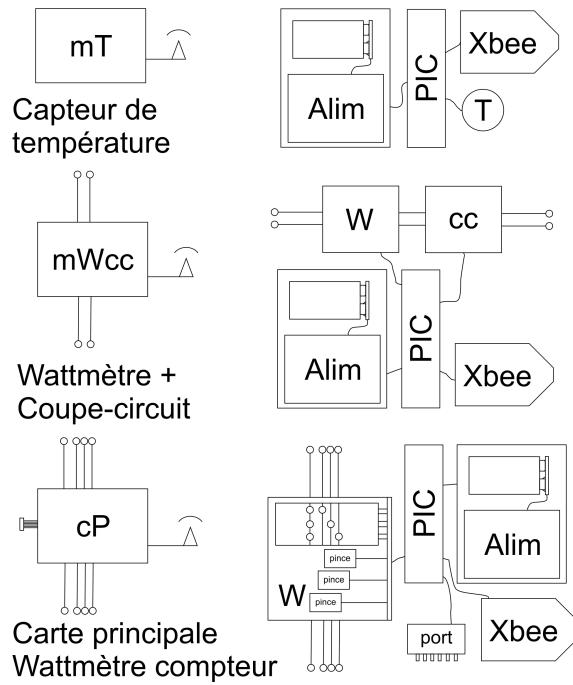


FIGURE 2.9 – Schéma des modules thermiques, du wattmètre, et de la carte mère

2.2 Etude de marché

Cette section présente une analyse de ce qui existe déjà, autant au niveau de l'interface que pour le système de capture et de monitoring dans son ensemble, comme ceux-ci vont souvent de paire. Il est important d'analyser les différences avec nos objectifs et de comparer les avantages et faiblesses de chaque systèmes (ou du moins d'un panel assez large)

2.2.1 Smartbox

Cet appareil[49] d'Electrabel permet la mesure de la consommation de 4 appareils électriques via des capteurs sans fil. Il est possible d'acheter d'autres capteurs électriques, et un capteur thermique en option. Les données sont envoyées par la smartbox à Electrabel qui propose une interface en ligne pour consulter diverses statistiques sur celles-ci. Les appareils surveillés peuvent également être coupés ou éteint.

Avantages :

- Application SmartPhone native

Inconvénients :

- Consommation d'appareils électriques uniquement.
- Pas de mesures globales ou de fort ampérage.
- Coût d'achat non négligeable (139€), et abonnement mensuel de 3€.

2.2.2 Open Energy Monitor

Open Energy Monitor[37] est une solution de monitoring Open Source. Aucun système matériel n'est produit, mais des schémas sont distribués pour fabriquer des capteurs. Comme notre système, celui-ci utilise des pinces ampermétriques pour mesurer la consommation électrique. Cependant, le système nécessite des compétences en électronique, et est limité à l'électricité et les températures uniquement. L'interface est en PHP.

Avantages :

- Open source

Inconvénients :

- Electricité uniquement
- Monitoring uniquement (pas de contrôle)

2.2.3 Efergy Elite

La solution proposée par efergy[18] consiste en un petit boîtier de mesure connecté au réseau sans fil, utilisant lui aussi des pinces pour mesurer les trois phases d'entrées. Le boîtier coûte une 50aine d'euros et envoie les données au serveur d'efergy qui propose un interface web et une application Android et iPhone.

Avantages :

- Applications SmartPhone natives

Inconvénients :

- Une seule mesure globale de l'électricité uniquement
- Propriétaire
- Monitoring uniquement (pas de contrôle)

2.2.4 Wattvision

Wattvision[53] est un appareil de mesure de la consommation électrique globale. Il propose plusieurs types de capteurs pour différents modèles de compteur, mais nécessite un compteur compatible.

Avantages :

- Support de plusieurs maisons

Inconvénients :

- USA Uniquelement
- Propriétaire et chère (250\$)
- Une seule mesure de l'électricité globale uniquement
- Interface semble très limitée
- Monitoring uniquement (pas de contrôle)

2.2.5 EKM Metering

EKM Metering[19] propose une solution de monitoring de l'électricité, du gaz et de l'eau, mais des consommations globales uniquement. EKM propose un logiciel

compatible Windows, Linux et Mac pour lire les données mesurées.

Avantages :

- Gaz, eau et électricité

Inconvénients :

- Pas d'appareils individuels
- Logiciel à 30\$, système complet entre 300\$ et 1000\$

2.2.6 Chacon ecowatt

Chacon ecowatt permet la mesure de consommation électriques globale de la maison. Le système de capteurs est composé de pinces ampermétriques et coûte environ 80€.

Avantages :

- Ecran LCD

Inconvénients :

- Electricité globale uniquement
- Logiciel uniquement windows

Cette liste n'est évidemment pas exhaustive, j'ai également trouvé de nombreux autres appareils et fournisseurs tel que OWL, Ewgeco, GEO, rmoni et bien d'autres, mais ceux-ci n'offrent pas spécialement plus de possibilités que ceux présentés ci-dessus.

Aucun des systèmes ne combine à la fois la possibilité de mesurer tous types d'énergies, de contrôler des appareils en plus du monitoring de ceux-ci et d'accéder aux données à distance sur internet. De plus, tous ces systèmes de monitoring ne permettent l'utilisation d'un modèle énergétique. Si cet objectif peut être mené à bien, nous aurons un système encore plus démarqué, pouvant tirer parti de simulations que je n'ai vu sur aucun système disponible dans le commerce, ou même à l'état de projet.

J'ai en effet cherché des solutions de modélisations énergétiques de maisons, mais tous les logiciels sont très complexes. On notera par exemple PEB[] logiciel officiel de la région wallonne destiné à cet usage, qui est très compliqué en matière d'utilisation, impliquant le recours aux services d'un professionnel.

La difficulté d'atteindre le même niveau se situe au niveau de la finition de certaines interfaces, conçues par des graphistes, et comprenant de nombreuses statistiques différentes. Evidemment, l'interface permettra l'ajout de divers graphiques parmi les différentes perspectives, mais ces interfaces font l'objet d'un développement par une équipe de designers que je ne peux évidemment égaler dans le temps qui m'est imparti. Certains systèmes disposent également d'applications natives, nous verrons que celles-ci peuvent avoir de petits avantages sur une simple version mobile d'un interface web.

Chapitre 3

Choix logiciels

Nous allons analyser dans ce chapitre les divers choix de logiciels et les choix d'implémentation.

Nous voulions un système livrable "clé sur porte", également compatible avec du matériel de récupération et divers systèmes d'exploitations. Nous pourrions recompiler nos exécutables et les adapter à ces derniers ainsi qu'à diverses architectures, mais ceci constitue un travail plutôt lourd.

Le choix du langage est en fait parti du module de calcul. Le modèle énergétique a été développé sous Matlab, qui présente une possibilité d'interfaçage plutôt complexe et une utilisation des ressources plutôt gourmandes. Nous verrons que parmi ces alternatives, l'utilisation du langage Python[41] ressort du lot.

Le module d'acquisition pour les webcams était lui aussi basé sur du code Matlab (finalement abandonné au profit d'une librairie Python). En effet, ce code laissait apparaître des problèmes similaires au module de calcul (extrêmement lent, interfaçage difficile, ...) et pour garder un même éco-système, les membres du projet IdP ont décidé d'écrire tous les codes en Python.

3.1 Langage : Python

Un des premiers avantages du Python est donc qu'il convient à toutes les parties du système :

- Pour les scripts : Notamment utile pour l'OCR et le contrôle des webcams, mais également pour les compteurs virtuels qui seront discutés dans la section 5
- Pour le modèle, la librairie numpy [35] permet le calcul matriciel et scientifique
- Pour l'interface, de nombreux frameworks et plugins sont disponibles en Python

Le Python est un langage interprété, c'est à dire qu'il n'est pas compilé. L'interpréteur Python exécute directement des instructions écrites dans un fichier source contenant du texte. Bien-sûr des séries d'optimisations permettent de garder une version entre le binaire et la source pour ne pas réinterpréter tout le texte à chaque exécution. Les applications Python sont donc directement cross-platform puisqu'il suffit d'installer l'interpréteur Python sur un des très nombreux systèmes supportés [43] pour que l'application puisse tourner.

Le Python est un langage orienté objet. En quelques mots, cela signifie que l'écriture d'une application est faite en petits modules auxquels sont délégués certaines fonctionnalités et ces modules s'appellent entre eux pour former l'application. Le Python est souvent décrit comme "hautement" orienté objet, dans le sens où il ne dispose pas de types primitifs : les entier ou caractères sont des objets. Il est également dit "de haut niveau", dans le sens où il est considéré "loin du code machine", l'utilisation ne requiert aucune connaissance sur la structure de son ordinateur, et particulièrement de la mémoire : il n'y aura jamais en Python d'utilisation de malloc ou autre. L'utilisation même de tableaux à taille fixe est très limitée, car ceux-ci sont souvent remplacés par des listes dynamiques.

Python convient très bien pour des petits scripts notamment grâce à ses nombreux plugins qui permettent d'écrire rapidement de petites applications sans "réinventer la roue" (Voir notamment la *Python Package Index* [44]), mais il convient aussi à l'écriture d'applications modestes. Il est utilisé par des grands noms tels que Google (bien que son utilisation au sein de la société ait tendance à disparaître au fur et à mesure du développement de l'entreprise) [46]. En effet le Python étant interprété, il est forcément plus lent [42] et tend à être abandonné lorsque des projets dépassent une certaine taille. Est-ce une raison pour ignorer le Python ? Non, puisque beaucoup d'applications ne requièrent pas de travail lourd, le Python permettant presque toujours un code plus concis [38], gagnant ainsi du temps de programmation. Ainsi, le Python convient à l'appel de fonction natives ou d'autres applications. S'il est de plus en plus utilisé dans le calcul scientifique via numpy[35] ou SciPy[48], c'est notamment parce que les performances ne sont pas tant impactées, car ces modules font appel à des librairies écrites en C ou fortran. L'utilisation du Python permet des codes concis, facilement compréhensibles qui sont en outre très utiles à la communauté scientifique. Celle-ci appelle tout particulièrement un grand partage des applications dans un langage facilement lisible et universel. Il est également beaucoup utilisé comme langage de plugins dans plusieurs applications connues tels que Blender [4] ou Inkscape [25] pour ces mêmes raisons.

L'utilisation du Python représente aussi un choix personnel parmi les autres possibilités offertes en matière de langage pour implémenter une interface web. En effet, je souhaitais opter pour un autre langage que le PHP avec lequel j'avais déjà beaucoup travaillé au travers la conception de nombreux sites. Je désirais donc varier. Le Java aurait également bien convenu grâce à ses frameworks pour le Web mais son

utilisation était déjà fort répandue dans le cadre de l'Université. Je l'avais également utilisé lors de mon stage.

3.2 framework : Django

Bien qu'il soit envisageable de générer directement du code HTML en Python, l'utilisation d'un framework a plusieurs utilités :

- La gestion plus fine de l'HTTP, des codes d'erreurs, des données envoyées via des requêtes POST, etc
- Un framework vient souvent avec une série de mécanismes ou d'architectures, comme le Modèle-Vue-Contrôleur (MVC) qui permet de séparer la gestion de la représentation de l'information, et les interactions de l'utilisateur.
- Souvent, les frameworks fournissent un système de template prenant part au cadre de l'architecture MVC. Les templates sont des pages écrites en HTML où le programmeur indique l'emplacement où viendront s'inscrire les données, calculées ailleurs dans le code.
- Respecter la structure proposée par un framework permet également d'assurer la continuité d'un projet. Un framework est plus facile à utiliser lorsqu'il est déjà familier à des milliers de programmeurs : s'il faut rajouter une page au site, ils n'auront pas besoin de chercher quelle fonction gère les url, quelles fonctions gèrent le rendu de cette page, où placer les images, etc.
- Les frameworks intègrent également presque tous un système de déploiement et de test pour la mise en production avec, par exemple, un mode de débogage pour le développeur.
- L'intégration de mécanismes de sécurité comme la gestion de l'authentification de l'utilisateur ou la protection contre certains types d'attaques courants

J'ai donc étudié diverses possibilités en matière de framework Python. Mes recherches préliminaires ont fait ressortir deux frameworks qui sont les plus connus et les plus cités : Pylons[40] et Django[13]. Le web ne manque pas de littérature sur le sujet "Pylons vs Django", mais peu de sources sont scientifiquement fiables car elles sont souvent basées sur des opinions et des points de vue subjectifs. Django est sans doute plus orienté "modules". Nous verrons qu'il sépare les applications et permet d'en réutiliser certaines, là où Pylons dispose d'un dossier modèles, un dossier contrôleur et un dossier templates. En réalité, il serait difficile de justifier mon choix sans refaire l'intégralité du projet sur ces deux frameworks. J'ai choisi Django notamment par sa plus grande popularité. Une communauté très large garantit plus de sources de documentations et offre de nombreux petits modules qui permettent de ne pas "réinventer l'eau chaude".

Django a été créé en 2005 et a beaucoup évolué depuis. Il dispose d'une bonne maturité, et d'une bonne stabilité grâce à son système de branche test, et stable. La documentation est claire quant aux versions et branches, ce qui est souvent oublié dans les documentations de logiciels informatiques. Les développeurs veulent souvent

garder la documentation au niveau de la dernière version.

Django suit le modèle MVC, garantissant une séparation claire de la partie logique, de la partie affichage, et de la partie donnée/modèle. Il dispose d'un système de template simple et puissant. Celui-ci permet de définir rapidement la structure de la page en HTML et de choisir l'emplacement de l'affichage du contenu d'une variable, ou l'exécution de boucles simples sur une liste, pour afficher un tableau par exemple. Il dispose du support pour plusieurs bases de données, mais dans tous les cas les données sont définies grâce à un modèle (un objet Python) qui sera enregistré comme une ligne de la base de donnée. Django génère d'ailleurs automatiquement la base de données en fonction des modèles définis en Python. Il permet également de générer automatiquement des formulaires en fonction du modèle et la validation de ceux-ci. Lors de la définition d'un modèle "Article" par exemple, le programmeur peut définir que le champ "titre" a une longueur maximale de 15 caractères. La base de donnée sera créée avec ce paramètre garantissant une optimisation maximale du stockage des données de la base, mais le formulaire affichera également un message d'erreur s'il dépasse le paramètre, tout cela avec une intervention très limitée du programmeur.

Un "site web" utilisant Django est divisé en plusieurs applications qui correspondent en général à un set de pages visant l'accomplissement d'une même fonctionnalité. Par exemple un blog simple aura probablement une application "messages" et une application "commentaires". L'appel de l'application correcte s'opère en fonction de l'URL entrée par l'utilisateur sur base d'expressions régulières (regex). Pour ce même exemple, deux regex du type "`^ message`" et "`^ comments`" re-dirigeront vers les deux applications. Chaque application dispose également d'une gestion d'URLs. Par exemple, `/message/` re-dirigera vers la liste des messages du blog mais `/message/id` permettra de voir un message particulier. L'utilisation d'expressions régulières offre la possibilité de définir sa propre structure d'url, en s'affranchissant du traditionnel `page ?var=value&var2=value2` peu "user-friendly".

Django inclut un serveur de développement. Même s'il est compatible avec presque tous les serveurs web, Django permet ainsi de déployer très rapidement un environnement pour commencer à programmer. Le serveur intégré est également adapté au développement avec Django, facilitant notamment le débogage.

Enfin, il inclut pas mal de "modules", de fonctionnalités qu'il suffit d'importer, comme par exemple une interface d'administration. En effet, pour beaucoup de sites, il est nécessaire d'avoir une interface d'administration qui suivra presque toujours la même logique. Dans l'exemple du blog, il faudra une interface administrateur pour créer le message et une autre pour modérer les commentaires. Il suffit d'inclure l'application `admin django`, et de paramétriser celle-ci en quelques lignes de codes pour avoir automatiquement cette série de formulaires d'ajout ou de suppression de messages et de modération de commentaires. De la même façon, Django inclut un système d'authentification, de gestion des utilisateurs et surtout de permissions

pour chacun de ses utilisateurs. Il dispose également de groupes pour réunir certains utilisateurs en catégories, avec les mêmes permissions par défaut. Plus globalement, il existe beaucoup de packages (à ce jour 1766 répertoriés sur [16]) qui permettent l'ajout de ce genre de fonctionnalités en quelques lignes de code.

3.2.1 Sécurité dans Django

Django dispose de plusieurs mécanismes de sécurité intégrés, tous utilisés dans l'interface et absolument nécessaires pour une interface web dont les principales sont :

- XSS : Par plusieurs techniques, un utilisateur pourrait introduire du code javascript sur une page web. Un exemple classique est une requête GET faite aussi avec un paramètre qui est affiché directement sur la page. Une personne mal intentionnée peut alors renvoyer vers notre site en mettant du code javascript dans cette requête, pouvant donc altérer la page et changer l'adresse d'envoi du mot de passe de l'utilisateur qui s'identifierait. Django échappe les caractères spéciaux des variables imprimées dans les templates, empêchant ce genre de techniques.
- CSRF : Il s'agit ici de se protéger contre une attaque de type replay. Il serait possible de renvoyer la requête HTTP faite par un utilisateur plusieurs fois, en altérant les données et se faisant passer pour lui. La technique de protection est ici de passer un code unique avec toutes les requêtes. Ce code unique n'est valide qu'une seule fois. La requête ne peut donc pas être rejouée.
- SQL injection : sans doute l'attaque Web la plus populaire, et pourtant encore bien trop souvent présente. Ainsi,(mais d'autres cas sont possibles), l'url d'un site affiche l'identifiant d'un message à afficher, par exemple `www.site.com/postid=5`. Si le développeur ne fait pas attention et introduit la variable "id" à la fin d'une requête SQL tel que `"SELECT * FROM posts WHERE id = " + id`, un utilisateur pourrait en profiter pour changer l'url par `www.site.com/postsid=5 OR 1 = 1`, entraînant l'affichage de tous les messages du site ! Dans Django, l'utilisation d'objets appelés Modèles est toujours de mise, on ne fait presque jamais appel à une requête SQL mais plutôt à des injonctions telles que `"Meter.objects.all()`" qui retournent tous les compteurs. Toutes les requêtes sont échappées et, comme les modèles définissent le type de données, elles sont converties. Dans l'exemple précédent, Django aurait retourné une erreur dûe au fait que l'ID du message n'est pas un nombre.

Le système d'authentification déjà présenté plus haut peut également être considéré comme faisant partie de la sécurité.

3.2.2 Base de données

Django supporte plusieurs backend pour différentes base de données. Le côté standard de mySql, mes connaissances de celui-ci et sa relative rapidité ont rapidement orienté mon choix vers lui. Oracle serait un peu trop gourmand en ressource pour un tel système, sa configuration étant beaucoup moins simple. L'alternative à mySql souvent présentée est SQLite, un système de base de données SQL qui stocke les données dans un fichier. Le problème est que je voulais garder un accès à la base de données depuis l'extérieur pour d'éventuels ajouts de données sans passer par l'interface. De plus, si la base de données n'est pas assez importante pour justifier Oracle, il ne faut pas oublier qu'en moyenne 10 compteurs (en ce y compris les virtuels) envoient des données toutes les 5 minutes, soit 2880 lignes par jour, soit environ 86400 entrées par mois. Ceci invite d'ailleurs l'ajout d'une deuxième table chargée de garder la consommation journalière agrégée de chaque compteur. Un nettoyage automatique de la table "brute" serait opportun mais en l'absence du modèle, il est préférable de laisser toutes les données.

Concernant le choix du moteur, nous verrons en détail les diverses tables et la structure générale de la base de données. L'interface doit pouvoir supporter plusieurs maisons, composées chacunes de plusieurs étages comportant plusieurs murs, eux-mêmes équipés de plusieurs fenêtres, ... Mon choix s'est donc porté vers InnoDB qui intègre la gestion des clés étrangères et permet une grande efficacité dans l'application des multiples jointures que nous aurons à faire.

3.2.3 Technologies Web

L'interface fait un usage extensif de l'HTML5 et du javascript. Elle utilise principalement la librairie jQuery, et quelques-uns de ses plugins. Le système le plus complexe, c'est-à-dire le dessin des plans (et la modélisation en générale), a été écrit entièrement en Javascript, dessinant grâce à l'élément Canvas introduit avec l'HTML 5.

HTML5/CSS3

L'HTML n'est plus à présenter. Sa version 5 apporte cependant son lot de nouveautés permettant d'envisager les fonctionnalités de l'interface sans recours à un plugin dans le navigateur tel que le flash ou les applets Java. Si l'HTML permet des modifications faciles du design sans recourir à des images, la nouveauté la plus utile dans notre cas est l'élément canvas, qui permet de dessiner sur un plan 2D avec des instructions de base telles que des points, des arcs de cercles, des lignes, etc. Un autre élément, le SVG, aurait été envisageable. Au niveau de la compatibilité avec les navigateurs : ils sont au même niveau. La différence est que le SVG permet de dessiner de façon vectorielle et le canvas de dessiner sur un bitmap. Cependant le canvas peut aussi bien dessiner en 2D qu'en 3D (sur les navigateurs compatibles)

tandis que le SVG n'est qu'en 2D. J'ai choisi d'utiliser le canvas pour me laisser la possibilité d'afficher le plan en 3D. Au final, j'ai modularisé totalement le système de rendu en un objet Javascript qui reçoit des instructions de dessins de fenêtres ou de murs, et qui se charge de les traduire pour dessiner sur le système sous-jacent. Au final, le SVG aurait donc mieux convenu au module uniquement 2D, puisqu'à chaque re-dimensionnement de la page le plan doit être redessiné, mais il était trop tard pour changer lorsque j'ai abandonné l'idée du plan 3D.

Les problèmes de compatibilité sont discutés dans la section 6.6.

Javascript

Une grande difficulté a été de gérer un code Javascript relativement gros et qui se veut modulable pour répondre à l'objectif de flexibilité, malgré le côté particulier de la gestion des objets en Javascript. En effet, le Javascript ne permet pas nativement d'étendre les objets. Il permet en revanche de copier les fonctions d'un objet à un autre. Ceci présente l'avantage d'avoir un mécanisme d'héritage mais pas d'étendre une fonction puisque celle-là même de l'objet parent sera remplacée par celle de l'enfant.

Pour combler ces manquements du Javascript, nombreux sites ont recours à un framework. J'ai porté mon choix sur jQuery [28] car il permet d'accéder facilement à des éléments de la page web, d'en modifier le contenu, de les animier, et facilite grandement la gestion de l'AJAX... Environ 45% des 100 000 sites les plus populaires du monde utilisent un framework, et 28% de ceux-ci utilisent jQuery, loin devant les autres [1].

L'utilisation de jQuery permet aussi d'assurer la compatibilité des instructions sur un très grand nombre d'appareils car certaines instructions se comportent différemment d'un navigateur à l'autre et jQuery s'assure d'utiliser les bonnes méthodes. Techniquement, jQuery n'ajoute évidemment aucune fonctionnalité à Javascript, mais il permet de faire beaucoup plus en beaucoup moins de lignes de code et moins de temps.

Pour les graphiques, c'est la librairie Highcharts qui a attiré mon attention. Elle n'est cependant pas libre (mais gratuite pour un usage limité tel qu'il se présente dans le cadre de mon TFE). Il existe de très nombreuses librairies, mais Highcharts était une des rares comprenant toutes les fonctionnalités que je désirais.

Chapitre 4

Structure de l'interface

Ce chapitre passera en revue la structure de la base de données, et les méthodes utilisées pour transférer les éléments de la base de données via Django, au client Javascript. Les éléments importants de l'interface seront ensuite expliqués comme le système de plan, de traitement des données ou encore le protocole de communication entre les capteurs et le serveur.

4.1 Applications

Comme expliqué dans la section 3.2, Django fonctionne en combinant plusieurs applications. Chaque application est un package Python (un dossier de fichiers Python). Les différentes applications sont :

- Alert : La page de gestion des alertes
- Builder : Les diverses étapes de modélisation
- Consumption : La page de consommation
- Data : L'échange des données avec les capteurs mais aussi l'extraction de données brutes pour des statistiques pour les graphiques
- Historic : Les pages des statistiques
- Main : La page d'accueil et de le système d'authentification
- Settings : Les pages de paramètres

Le dossier principal contient également un package monitoring qui contient les scripts de Django et la configuration de celui-ci ainsi qu'un dossier static qui comprend tous les fichiers statiques (les images, les feuilles de styles css, et les scripts javascripts) et un dossier templates qui reprend toutes les templates telles que décrites précédemment.

Chaque application est composée de plusieurs fichiers Python :

- views.py : Ce fichier contient toutes les fonctions "contrôleur" qui reçoivent en paramètre un objet HttpRequest, contenant les informations sur la requête de l'utilisateur. Ces fonctions sont chargées de retourner un objet HttpResponse qui contiendra le code HTML à renvoyer au navigateur, et le code de réponse HTTP

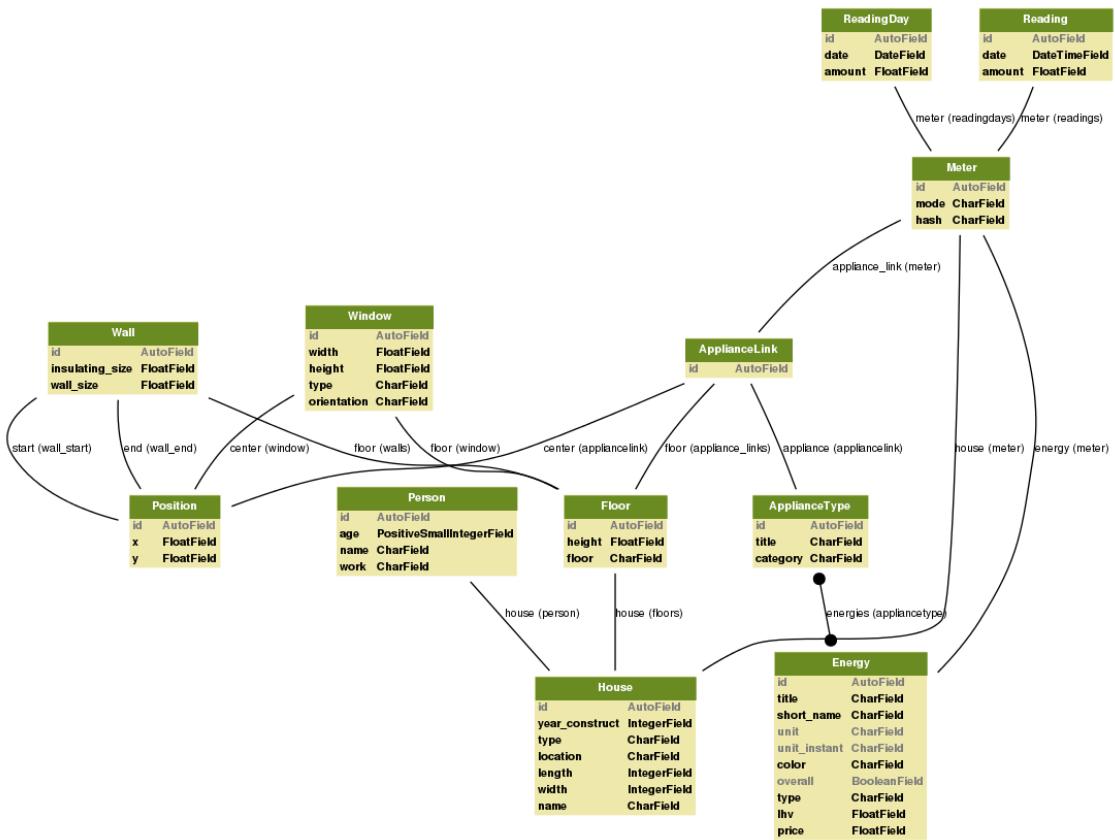


FIGURE 4.1 – Structure des données relatives à la maison

- models.py : Ce fichier contient les modèles. Ceux-ci décrivent les objets et chacun de leurs champs qui seront enregistrés dans la base de données. Seules certaines applications ont des modèles. Tous les modèles relatifs à la maison sont dans le models.py de builder. Ceux qui sont relatifs aux relevés des compteurs se trouvent dans data, et le profil utilisateur est défini dans main.
- urls.py : Ce fichier associe des expressions régulières aux vues définies dans views.py, permettant ainsi de définir quelle vue sera utilisée en fonction de l'URL

Le fichier urls.py du dossier monitoring permet de définir quelles applications seront appelées en fonction de l'URL. La partie matchée de l'expression régulière est enlevée et la partie restante est vérifiée en fonction d'un fichier urls.py dans le dossier de l'application, pour lancer la vue correspondante.

4.2 Structure de la base de donnée

La base de données peut être divisée en 3 parties distinctes : la structure de la maison 4.1, les alertes 4.2 et l'authentification 4.3.

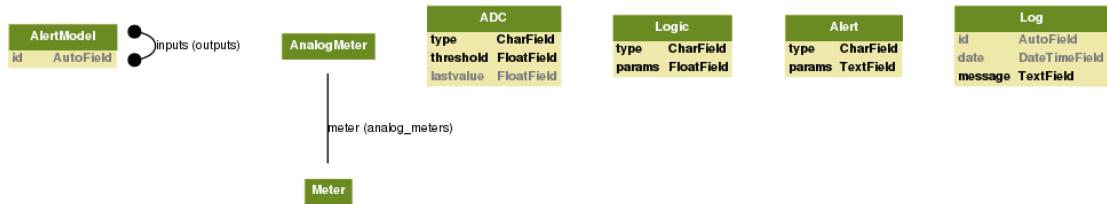


FIGURE 4.2 – Structure des données relatives aux alertes

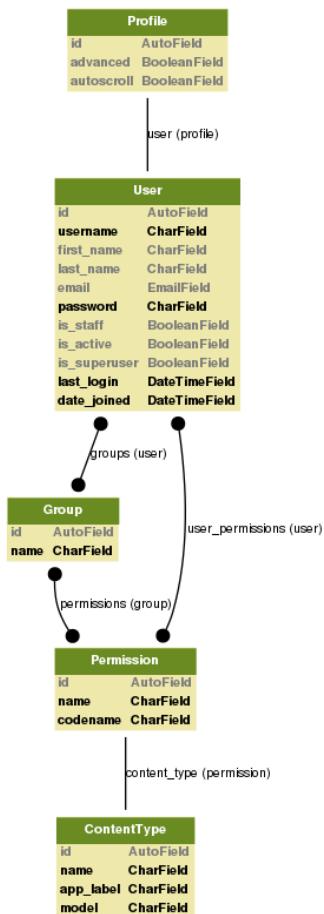


FIGURE 4.3 – Structure des données relatives aux utilisateurs

Pour rappel, Django crée automatiquement la base de données et l'extension django-extensions[15] permet également de générer les graphiques des figures 4.1, 4.2 et 4.3. Il gère également les clés étrangères et crée des relations inverses dans l'objet Python. Dès lors, si l'objet maison possède plusieurs étages, la relation etage.house, permet de remonter la relation en sens inverse.

4.2.1 Structure de la maison

L'interface supporte plusieurs maisons représentées par l'objet maison (House). Les champs de l'objet maison correspondent à ceux que l'utilisateur doit encoder à l'étape 1 décrite dans la section 6.1. Une maison est composée de plusieurs étages (Floor), eux-mêmes composés de plusieurs murs (Wall), et de fenêtres (Window). Les fenêtres ne sont donc pas liées au mur mais à l'étage. Les murs sont définis par deux positions, ils ont une épaisseur d'isolant et une épaisseur totale du mur. Les fenêtres sont définies par une position et une orientation et disposent également d'une information de largeur et de hauteur. Une série de personnes (Person) sont associées à la maison. Le fait de créer des relations vers des objets Position plutôt que de créer des champs x et y au sein des murs et fenêtres est dû à la gestion directe de la base de données par Django. Pour pouvoir garder un objet "Position" Python, les tables doivent être séparées, ce qui imposera malheureusement de nombreuses jointures qui auraient pu être évitées.

Chaque étage peut contenir plusieurs appareils (ApplianceType). Le lien entre un type d'appareil et un étage est fait par la relation ApplianceLink. Chaque appareil fonctionne avec une ou plusieurs énergies, c'est donc une relation multiple générée automatiquement par Django. En réalité, il crée une table comprenant une référence vers un ApplianceType et un objet Energy, afin de pouvoir lier plusieurs énergies à plusieurs appareils.

Les compteurs sont reliés soit à un appareil (via son lien ApplianceLink) soit à une maison (House). En effet, les compteurs peuvent être lié à un appareil, ou être défini en tant que compteur global de la maison. Chaque compteur est aussi lié à une énergie, on peut ainsi déterminer quelle énergie de l'appareil ou de la maison il est chargé de mesurer.

Le compteur dispose d'un champs "mode" qui peut prendre 3 valeurs :

- Instantané (INS) : le compteur envoie la consommation instantanée
- Total (TOT) : le compteur envoie la consommation totale depuis sa mise en activité
- Différentiel (DIF) : le compteur envoie l'énergie totale consommée depuis son dernier envoi

Les énergies disposent également d'un champs type, qui peut prendre 3 valeurs également :

- Puissance (POWER) : ce sont les énergies qui peuvent produire un travail, comme l'électricité, le gaz, ou le fuel
- Consommable (CONS) : ce type est fait pour ce que l'on voudrait mesurer, mais qui n'entre pas dans la consommation d'énergie, comme l'eau. L'eau n'est pas une énergie et n'est pas convertissable en tant que telle. Cependant la consommation d'eau est une information nécessaire pour l'utilisateur et est souvent considérée comme facteur écologique. Elle a donc toute sa place dans notre interface.
- Etat (STATE) : ceci concerne les compteurs qui représentent un état, comme la température. On ne consomme pas de la température, cependant elle est également utile dans l'évaluation de la consommation en chauffage

L'activateur on/off (énergie "switch") est également considéré comme un état. Ici, le système de compteur est abusé et ce sera l'interface qui permettra de définir une valeur au compteur "switch", dont le relais, qui est chargé d'allumer effectivement ou éteindre un appareil, viendra lire la valeur.

Les changements de traitement que cela induit seront discutés dans la section 4.6 Traitement des données.

Les relevés sont liés au compteur, ils contiennent l'heure et la date du relevé, et la valeur (amount) lue sur le compteur. Notons qu'il y a aussi une table chargée de retenir le relevé par jour. L'opération sera décrite en détail dans la section 4.6, mais les calculs en fonctions du modes de compteurs étant très nombreux, à l'ajout d'une donnée, la consommation du jour est calculée et enregistrée pour pouvoir afficher rapidement les données sur une année complète par exemple, permettant de traiter uniquement 365 relevés et non plus de 100000.

4.2.2 Alertes

Le cahier des charges imposait une interface simple mais puissante. Cette recherche m'a orienté vers un système de puzzle (Voir fig. 4.4) où l'utilisateur voit directement quelles pièces il peut emboiter, et créer une logique en assemblant des éléments du puzzle. Toutes les pièces du système de "puzzle" appartiennent à l'une des 4 classes suivantes :

- AnalogMeter : Représente une pièce qui est la première entrée : le relevé d'un compteur. Elle contient donc un lien vers un objet Meter tel que présenté dans la section précédente.
- ADC : Elle représente une pièce qui convertit une entrée analogique en une sortie digitale. Elle dispose d'un champs seuil (threshold). Le champs Lastvalue permet de garder en mémoire la dernière valeur lue afin de permettre de repérer le moment où une donnée traverse le seuil, sans devoir remonter la chaîne jusqu'au compteur.

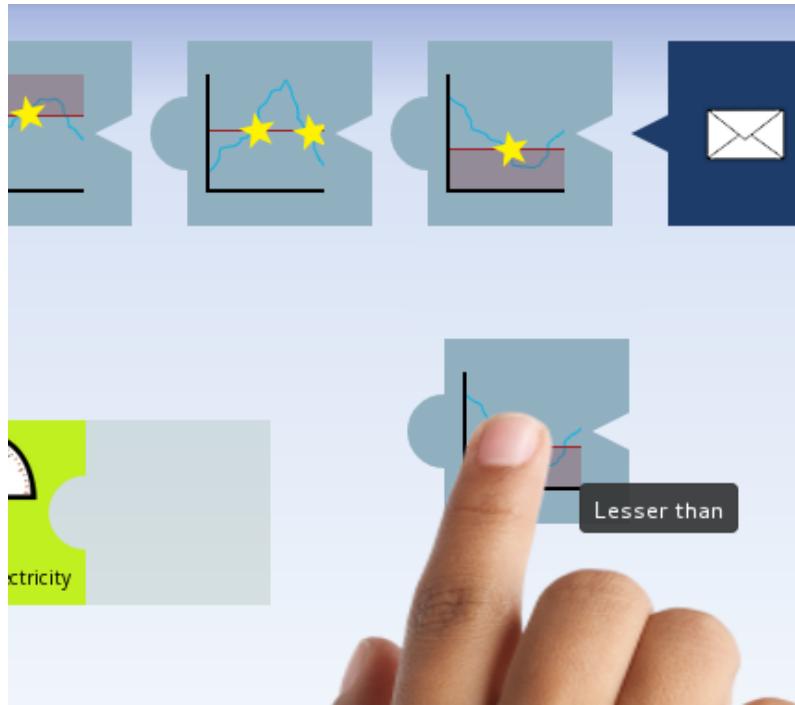


FIGURE 4.4 – Gestion des alertes à l'utilisateur.

- Logic : ce sont les pièces “digital vers digitalé qui servent à filtrer des évènement selon certaines conditions.
- Alert : ce sont les pièces finales, chargées de déclencher une action comme une notification ou un mail

Chacune de ces pièces étend (dispose d'un lien vers) un objet AlertModel. Cet objet sert à définir quelles pièces sont liées à quels autres via une relation multiple. La base de donnée permet donc des pièces avec plusieurs sorties et plusieurs entrées, mais l'interface ne le permet pas, par manque de temps et de réel intérêt. Le système est déjà puissant comme ça, et la priorité serait plutôt de définir de nouvelles pièces comme de nouvelles entrées sur la consommation journalière d'un compteur, l'heure de la journée ou tout autre donnée numérique, et d'autres sorties comme laisser la possibilité de lancer une requête HTTP en cas d'alerte, un SMS, etc. Notons simplement que toutes ces pièces sont donc possibles, mais il fallait bien mettre une limite au champs de la présente recherche.

4.2.3 Authentification

La plupart des modèles sont ici fournis directement par le package d'authentification Django. Le seul créé par mes soins est l'objet profile, permettant de définir si l'utilisateur est en mode avancé (advanced) et s'il désire l'option de défilement automatique (autoscroll). Les groupes ne sont pas utilisés dans l'interface, par contre les permissions le sont.

4.3 Transfert de données entre Django et les applications Javascript

De nombreuses données extraites de la base doivent être transférées telles quelles à l'application Javascript. En effet, si Django est particulièrement efficace pour aller récupérer des données dans la base et les traiter pour les afficher dans une template, il ne prévoit pas spécialement de passage des données à une application Javascript. La différence réside dans l'utilisation de ces données qui doivent être passées brutes au sein d'une application Javascript. Par exemple, il n'est pas question de générer côté serveur le plan en HTML, puisque celui-ci doit pouvoir être modifié.

Pour le transfert dans le sens serveur vers Javascript, c'est le JSON qui a été choisi. En effet, il peut être directement écrit sur la page dans une balise `<script>`, puisque le JSON est le format de description d'objets et de tableaux natifs du JavaScript. Le traitement de celui-ci est donc particulièrement performant.

Il reste à décider comment gérer les relations entre les objets. Il y a deux possibilités :

- soit Django donne à la page un objet House, une liste d'objet Meter, Energy, etc. Pour trouver les informations sur l'énergie que mesure le compteur, il suffit alors de parcourir la liste des énergies pour trouver l'énergie identifiée par l'id contenue dans le champ `energy_id` du compteur.
- soit écrire toute la structure étendue en JSON, en remplaçant le champs `energy_id` de l'objet Meter qui identifie l'énergie qu'il mesure directement, par un champs `energy` qui contient l'objet représentant l'énergie.

Cette dernière relation introduit une redondance, puisque si trois compteurs mesurent la même énergie comme l'électricité, l'objet Energy pour l'électricité sera écrit trois fois. La première méthode elle, implique la gestion d'un système d'index et un traitement plus lourd pour la gestion de nombreuses relations. C'est la deuxième méthode, où il s'agit de suivre la structure que j'ai choisie. En effet, tant que les données ne sont pas modifiées, la redondance n'est pas très grave, et cela simplifie grandement la gestion des relations, puisqu'on peut accéder directement à un sous objet en utilisant par exemple `meter.energy.color`. Afin d'illustrer le processus, la figure 4.5 représente les données des compteurs à transférer à l'application Javascript de dessin du plan. La figure 4.6 montre le code du contrôleur Python et la figure 4.7 le code du template HTML contenant l'instruction `{% meters|safe %}` qui sera remplacée par le contenu de la variable `meters` dans le code HTML (dans une balise `script`). Le filtre `/safe` permet de ne pas activer l'échappement automatique que fait Django pour les caractères spéciaux.

Pour écrire ces données, Django possède un mécanisme de sérialisation, mais qui ne permet malheureusement pas la sérialisation en profondeur. En d'autres termes, si on lui demande de sérialiser l'objet `Floor`, il ne décrira pas l'objet `House`, mais définira un champs `house_id` avec pour contenu l'index de la maison. Pour cela

id	energy_id	mode	hash	house_id	appliance_link_id
31	1	INS	JbXc8HB7eTmy	12	NULL
32	2	TOT	IbZfWx6EpUtS	12	NULL
44	5	INS	Ix6HFIsrvxQG	12	NULL
77	6	INS	VoxcqtDc3muX	12	168
81	3	TOT	W5rtork2blqi	12	NULL
85	1	INS	ixYmQqSrUJVn	12	168

FIGURE 4.5 – Exemple de traversée des données - Données des compteurs tels qu'enregistrés dans la base de donnée mySQL

```
#Stage 7 : Meters
@permission_required('builder.edit_house')
def stage7(request):
    house = shortcuts.get_object_or_404(House, id=request.session['house'])
    meterForm = forms.MeterForm(initial = {'house': house.pk})
    minCorner,maxCorner = house.getDimension()
    context = { 'house_id':house.pk,
                'house':house.pk,
                'floor_types':Floor.FLOOR,
                'floors':serializers.serialize("json", Floor.objects.filter(house = house),use_natural_keys=True),
                'house':house,
                'meters':serializers.serialize("json",
                                                Meter.objects.filter(house = house),
                                                use_natural_keys=True,
                                                relations=[{
                                                    'energy':(),
                                                    'appliance_link':
                                                    {
                                                        'relations':
                                                        {
                                                            'center':(),
                                                            'appliance':
                                                            {
                                                                'relations':('energies',)
                                                            }
                                                        }
                                                    }
                                                }]),
                'meter_modes':json.JSONEncoder().encode(Meter.MODES),
                ...
            }
```

FIGURE 4.6 – Exemple de traversée des données - Code Python chargé d'aller récupérer les données dans la base. Le contexte associe des variables qui seront utilisées dans une template pour être remplacé par les données.

```
<script type="text/javascript">
    //Retrieving meter list
    var meters = jsonStripModel('{{meters|safe}}');

    //Retrieving the list of modes that the meter can have
    var meter_modes = {{meter_modes|safe}};


```

FIGURE 4.7 – Exemple de traversée des données - Template contenant la variable meters qui sera remplacée par le code JSON par Django

4.3. TRANSFERT DE DONNÉES ENTRE DJANGO ET LES APPLICATIONS JAVASCRIPT35

```
<script type="text/javascript">

    //Retrieving meter list
    var meters = jsonStripModel([{"pk": 31, "model": "builder.meter", "fields": {"house": 12, "energy": "0.128", "unit_instant": "kWatt", "price": 0.128, "overall": false, "lhv": 1.0, "type": "POWER", "unit": "kWh"}, "hash": "Jbx8HB7eTmy", "mode": "INS", "appliance_link": null}, {"pk": 32, "model": "builder.meter", "fields": {"house": 12, "energy": {"pk": 2, "model": "builder.energy", "fields": {"short_name": "water", "title": "Water", "color": "#35b1d2", "unit_instant": "L/h", "price": 0.0022, "overall": false, "lhv": 0.0, "type": "CONSU", "unit": "l"}, "hash": "IbzFwx6EpUTs", "mode": "TOT", "appliance_link": null}, {"pk": 44, "model": "builder.meter", "fields": {"house": 12, "energy": {"pk": 5, "model": "builder.energy", "fields": {"short_name": "temp", "title": "Temperature", "color": "#ad1f24", "unit_instant": "\u00b0deg;C", "price": 0.0, "overall": false, "lhv": 0.0, "type": "STATE", "unit": "\u00b0deg;C"}, "hash": "Ix6HFIsvrXQG", "mode": "INS", "appliance_link": null}, {"pk": 77, "model": "builder.meter", "fields": {"house": 12, "energy": {"pk": 6, "model": "builder.energy", "fields": {"short_name": "switch", "title": "Switch", "color": "#8ACC27", "unit_instant": "", "price": 0.0, "overall": true, "lhv": 0.0, "type": "STATE", "unit": ""}, "hash": "VoxcqDc3mUX", "mode": "INS", "appliance_link": {"pk": 168, "model": "builder.applianceLink", "fields": {"appliance": {"pk": 1, "model": "builder.applianceType", "fields": {"category": "C", "energies": [{"pk": 1, "model": "builder.energy", "fields": {"short_name": "electric", "title": "Electricity", "color": "#f1c81d", "unit_instant": "kWatt", "price": 0.128, "overall": false, "lhv": 1.0, "type": "POWER", "unit": "kWh"}], "pk": 6, "model": "builder.energy", "fields": {"short_name": "switch", "title": "Switch", "color": "#8ACC27", "unit_instant": "", "price": 0.0, "overall": true, "lhv": 0.0, "type": "STATE", "unit": ""}}, "center": {"pk": 4357, "model": "builder.position", "fields": {"y": 2.5, "x": 10.5}, "floor": 96}}}, {"pk": 81, "model": "builder.meter", "fields": {"house": 12, "energy": {"pk": 3, "model": "builder.energy", "fields": {"short_name": "gas", "title": "Gas", "color": "#374973", "unit_instant": "L/h", "price": 0.728, "overall": false, "lhv": 0.09, "type": "POWER", "unit": "L"}, "hash": "W5rtork2b1qj", "mode": "TOT", "appliance_link": null}, {"pk": 85, "model": "builder.meter", "fields": {"house": 12, "energy": {"pk": 1, "model": "builder.energy", "fields": {"short_name": "electric", "title": "Electricity", "color": "#f1c81d", "unit_instant": "kWatt", "price": 0.128, "overall": false, "lhv": 1.0, "type": "POWER", "unit": "kWh"}, "hash": "ixYnQsGrUJvn", "mode": "INS", "appliance_link": {"pk": 168, "model": "builder.applianceLink", "fields": {"appliance": {"pk": 1, "model": "builder.energy", "fields": {"short_name": "electric", "title": "Electricity", "color": "#f1c81d", "unit_instant": "kWatt", "price": 0.128, "overall": false, "lhv": 1.0, "type": "POWER", "unit": "kWh"}, "hash": "IbzFwx6EpUTs", "mode": "INS", "appliance_link": null}, {"pk": 6, "model": "builder.energy", "fields": {"short_name": "switch", "title": "Switch", "color": "#8ACC27", "unit_instant": "", "price": 0.0, "overall": true, "lhv": 0.0, "type": "STATE", "unit": ""}, "hash": "VoxcqDc3mUX", "mode": "INS", "appliance_link": null}, {"pk": 4357, "model": "builder.position", "fields": {"y": 2.5, "x": 10.5}, "floor": 96}}}}}], "center": {"pk": 4357, "model": "builder.position", "fields": {"y": 2.5, "x": 10.5}, "floor": 96}}}}];

    //Retrieving the list of modes that the meter can have
    var meter_modes = [[["TOT", "Total count"], ["INS", "Instant consumption"], ["DIF", "Difference"]]];

```

FIGURE 4.8 – Exemple de traversée des données - Objets représentés en JSON tel que généré par le plugin FullSerializer

j'ai utilisé le plugin Django FullSerializers qui ajoute la possibilité d'énoncer quels modèles identifiés par des clés étrangères il faut développer.

Notons que les objets JSON sorti par Django sont nettoyés par la fonction jsonStripModel() du fichier global.js. En effet les objets JSON générés par Django contiennent quelques champs qui définissent le type de modèle et des méta-données sur la classe (Voir fig. 4.9). Les données elle-mêmes sont dans un objet fields. Cette fonction crée un objet de classe correspondant au modèle en fonction des méta-données, avec pour champs, les variables du modèles et comme valeurs des champs, leur contenu (Voir fig. 4.10). En Javascript, même les objets spécialement définis sont anonymes. Il n'y a pas de différence entre l'instruction `var person = new Person(22, 'Tom', 'STUDENT');` et `var person = {age :22, name :'Tom', work :'STUDENT'};`. En effet, demander quel est la classe de ces deux objets retournera toujours "object". Ainsi, seuls certains objets pour lesquels des fonctions sont spécifiquement définies, par exemple les murs, possèdent une fonction getOrientation() qui retourne l'orientation du mur ; ceux là sont définis dans `models.js`. Les autres sont donc créés de façon anonyme.

Lorsque les données sont modifiées, l'envoi du Javascript vers Django ne se fait pas en JSON mais directement par la fonction `objects_to_request()` qui génère les données formatées pour une requête POST standard. Les données de plusieurs positions sont ainsi renvoyées sous la forme `num_positions=X&position_x1=X1&position_y1=Y1&position_x2=X2&...`. La dé-sérialisation étant de toute façon impossible à faire automatiquement côté serveur puisque les objets sont anonymes, leur classe

```

▼ Array[6] ⓘ
  ▼ 0: Object
    ▼ fields: Object
      ▼ energy: Object
        ▼ fields: Object
          color: "#f1c81d"
          lhv: 1
          overall: false
          price: 0.128
          short_name: "electric"
          title: "Electricity"
          type: "POWER"
          unit: "kWh"
          unit_instant: "kWatt"
        ► proto_ : Object
        model: "builder.energy"
        pk: 1
      ► proto_ : Object
      hash: "3bXcBHB7eTmy"
      house: 12
      mode: "INS"
    ► proto_ : Object
    model: "builder.meter"
    pk: 31
  ► proto_ : Object
  ▷ 1: Object
  ▷ 2: Object
  ▷ 3: Object
  ▷ 4: Object
  ▷ 5: Object
  length: 6
  ► __proto__: Array[0]

```

FIGURE 4.9 – Exemple de traversée des données - Représentation interne après parsing par le navigateur du code JSON représenté dans la figure 4.8. On observe que les données utiles sont dans un sous-objet fields de l'objet Meter, et que son type est contenu dans model.

```

▼ [Meter, Meter, Meter, Meter, Meter, Meter, remove: function, removeEquals: function] ⓘ
  ▼ 0: Meter
    appliance_link: null
    ▼ energy: Energy
      color: "#f1c81d"
      lhv: 1
      overall: false
      pk: 1
      price: 0.128
      short_name: "electric"
      title: "Electricity"
      type: "POWER"
      unit: "kWh"
      unit_instant: "kWatt"
    ► proto_ : Object
    hash: "3bXcBHB7eTmy"
    mode: "INS"
    pk: 31
  ► proto_ : Object
  ▷ 1: Meter
  ▷ 2: Meter
  ▷ 3: Meter
  ▷ 4: Meter
  ▷ 5: Meter
  length: 6
  ► __proto__: Array[0]

```

FIGURE 4.10 – Exemple de traversée des données - Objets après transformation par `jsonStripModel()` de la liste de structure Meter. Par rapport à la figure 4.9, il n'y a plus de sous-structure fields, et le champs model a été utilisé pour instancier des objets de la classe correspondante.

étant perdue il n'était pas spécialement avantageux d'utiliser le JSON plutôt qu'une boucle récupérant les requêtes. Si le JSON peut être parsé nativement par tous les navigateurs, il ne peut pas être écrit nativement et doit donc faire l'objet d'un traitement plus lourd par une librairie externe. De plus le Python est moins dynamique que le JavaScript et il est plus compliqué de faire les transformations citées dans les paragraphes précédent en sens inverse.

4.4 Système de dessin du plan

La partie la plus complexe du travail a sans doute été d'écrire le système capable d'utiliser un élément HTML canvas (tel que présenté dans la section 3.2.3) pour dessiner le plan d'une maison dont les éléments sont enregistrés suivant la structure présentée dans la section 4.2.1, mais surtout permettre la définition de ces éléments, c'est à dire le placements des murs, des fenêtres, des objets composant la maison, et des compteurs. Il faut également viser une modularité à plusieurs niveaux : que le système de rendu sous-jacent soit facilement inter-changeable, et que les fonctionnalités du plan puissent être ajoutées et modifiées à souhait. En effet, il faudrait pouvoir ajouter une étape de définition des portes après celle des fenêtres par exemple.

Comme expliqué dans la section 3.2.3, le Javascript ne prévoit pas de mécanisme d'extension des fonctions (sauf par copie de celles-ci, ce qui est peu performant). J'ai donc implémenté un gestionnaire d'évènement EventManager (implémenté dans *eventmanager.js*), qui permet d'inscrire des fonctions à appeler lorsqu'un évènement est déclenché. Par exemple le plugin de dessin des murs inscrira sa fonction de dessin des murs à l'évènement "refresh" du plan. Ainsi lorsque le plan doit être redessiné, il dessine les éléments de base comme la grille ou l'arrière plan, et appelle ensuite toutes les fonctions inscrites à l'évènement "refresh".

Les pages HTML utilisant le système de plan ne contiennent que le code et les données nécessaires au dessin du plan vide, tel que la liste des étages. Chaque plugin se charge d'aller récupérer le contenu nécessaire au dessin de ses éléments en AJAX, permettant à la page de se dessiner rapidement, pendant que le chargement des données se fait. Le plugin de gestion des murs du plan (*plan_walls.js*) inscrira sa fonction de chargement à l'évènement "floorChanged", ainsi, chaque fois que l'utilisateur change d'étage, les murs seront rechargés. Ce plugin définit aussi un nouvel évènement "wallLoaded", appelé lorsqu'il a terminé de charger la liste des murs. Le dessin des fenêtres s'inscrira à ce dernier évènement mais celui des appareils également. En effet, il semble logique de charger et d'afficher les murs avant les fenêtres, mais les fenêtres et les objets peuvent se dessiner dans un ordre quelconque après les murs. Ce système permet également de n'avoir qu'un dessin partiel. En effet, lorsque l'utilisateur désire modifier les murs de sa maison, il est intéressant de n'afficher que les murs s'il n'a pas besoin des appareils et des compteurs par exemple. Il suffit de ne pas inscrire les évènements de dessins de ces derniers pour ne pas les afficher. Les fonctions concernant les compteurs, les fenêtres, les appareils

et les murs sont groupées respectivement dans des groupes de fichiers JavaScript séparés par fonctions, afin de permettre un chargement minimal en accord avec les fonctionnalités demandées.

Comme certains évènements sont asynchrones, ils peuvent finir en même temps ou à des moments très proche. La fonction `refresh()` inclu donc un mutex `plan.refreshing` pour empêcher plusieurs rafraîchissements simultané. De plus, il serait inutile de rafraîchir le plan plus de 15 fois par secondes (nous parlons bien ici de changer le contenu des pixels, pas de la fréquence d'affichage de ces pixels à l'écran). Même quand l'utilisateur passe sa souris au dessus du plan et que l'on doit dessiner la position d'une future fenêtre qui change constamment de position par exemple. Le mutex n'est donc réactivé que 66ms après la fin du rafraîchissement du plan. Si une demande de rafraîchissement du plan est faite pendant ce temps, elle ne sera exécutée qu'après les 66ms. Si une deuxième demande de rafraîchissement, elle sera ignorée.

Le schéma général est visible sur la figure 4.11. Plutôt que d'adresser directement l'élément canvas, un objet chargé du rendu (`renderer.js`) est responsable de la translation d'objets complexes en primitives 2D. Ainsi, le plugin de gestion des murs demandera au Renderer de dessiner un mur avec son épaisseur et sa position "réelle" en mètre, et le Renderer se chargera de traduire ces données en une série de fonctions de dessin de rectangles pour l'élément canvas, tout en se chargeant de convertir les positions réelles en positions à l'échelle du plan en pixels. L'idée est ici que ce Renderer soit totalement interchangeable, pour pouvoir utiliser un autre élément sous-jacent que le canvas, ou dessiner en 3D grâce à WebGL par exemple. Le JavaScript ne permet pas d'implémenter des interfaces au sens Java du terme, mais il faut considérer que le Renderer en utilise un. Notons que le dessin des compteurs se fait en HTML et non directement sur le canvas, notamment pour pouvoir utiliser des bibliothèques de graphiques en JavaScript. Là aussi, les positions des éléments sur le plan sont calculées par Renderer, sans faire de traduction par l'échelle d'un pixel/mètre directement afin de ne pas casser la modularité.

Tous les plugins ont plus ou moins la même structure : la fonction `registerYYY-Plugin()` où YYY est Wall, Window, Appliance ou Meter se chargent d'enregistrer les évènements nécessaires au fonctionnement du plugin. En général, il y en a un pour le chargement des données nécessaires qui s'inscrit à l'évènement "floorChanged" ou "wallLoaded" et un autre pour l'affichage sur le plan qui s'inscrit à l'évènement "refresh". Les pages qui se contentent d'afficher le plan appellent les 4 fonctions register. Les pages de l'application builder inscrivent en plus, des fonctions aux évènements "click" et "move" pour interagir avec l'utilisateur lorsqu'il clique sur le plan (ou la touche du doigt pour les smartphone) et lorsque la souris bouge au dessus de celui-ci.

La plupart des maisons ont la même enveloppe extérieure à chaque étage. Pour ne pas que l'utilisateur aie à recopier les murs extérieurs lorsqu'il change d'étage, lors de l'étape de définition des murs, et que l'étage inférieur est déjà défini, un algorithme

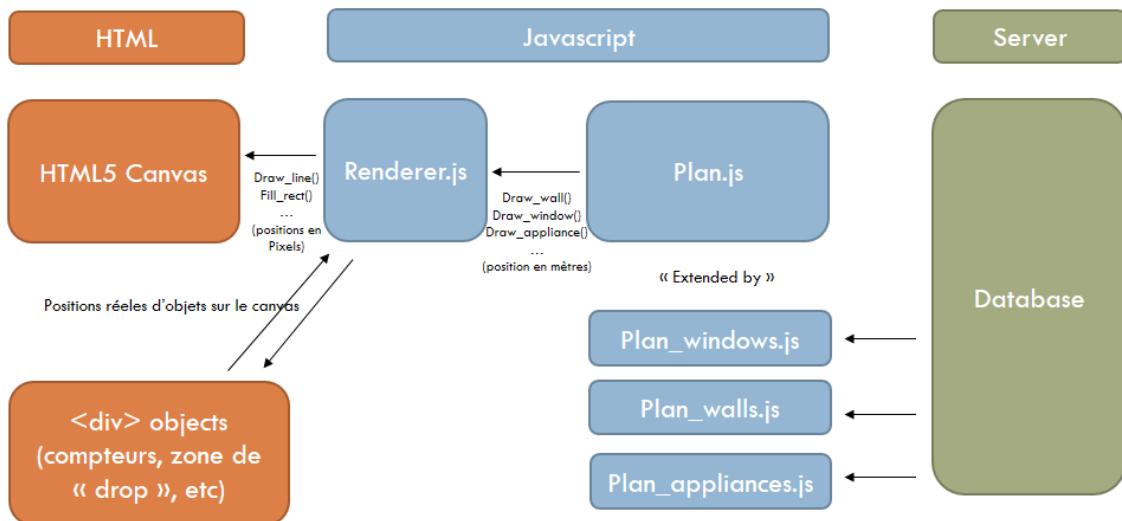


FIGURE 4.11 – Organisation du dessin du plan

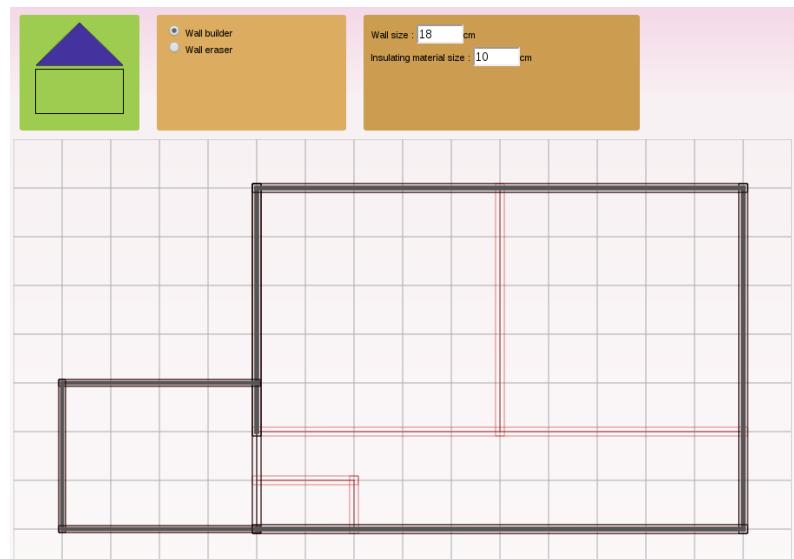


FIGURE 4.12 – Copie automatique des murs extérieurs. Les murs de l'étages du dessous sont visibles en rouge. Les murs extérieurs sont automatiquement copiés.

d'analyse en composante connexe en deux pas sera appliquée sur le plan du mur (fonction `plan_labelize()` dans `plan.js`) pour trouver les murs extérieurs (Voir fig. 4.12). Pour rappel, la méthode consiste simplement à analyser les cases dans l'ordre gauche-droite, haut-bas et à leur attribuer un numéro appelé label qui commence à 0. Le numéro est choisi comme le plus petit des numéros en haut et à gauche de la case analysée, sauf si elles sont séparées par un mur. Si les deux sont séparées par un mur, alors on augmente le label puisqu'on entre dans une nouvelle zone. La seconde passe se fait en sens inverse, et consiste à attribuer le plus petit des labels des zones à droite et en dessous de chacune de chacune des cases du plan si elles sont dans la même zone. Le résultat est un tableau correspondant à chaque case de la grille du plan, contenant le numéro de la zone à laquelle appartient la case. Les zones correspondant aux pièces, et la partie extérieure étant toujours la zone numéro 0. Il suffit alors de copier les murs ayant une zone positive et une zone égale à 0.

Cet algorithme est également utilisé pour colorer les pièces de la maison de façon différentes pour un rendu plus agréable.

4.5 Protocole de communication

Cette section discutera du protocole d'envoi des données des capteurs vers le serveur web. Plusieurs possibilités s'offraient à moi : l'utilisation d'un protocole de monitoring spécialisé comme SNMP, implémenter moi-même un protocole basé sur TCP ou UDP, ou utiliser le protocole HTTP pour directement communiquer avec l'interface. C'est ce dernier que j'ai choisi.

Pour rappel, ce travail se faisait dans le cadre d'un projet plus global, où d'autres équipes de personnes n'ayant presque aucune compétence en informatique devaient m'envoyer des données via leurs capteurs présentés dans la section 2.1. Il fallait donc un système très simple, l'implémentation d'un agent SNMP, l'authentification d'SNMPv3 nécessaire à la sécurité de ce projet, ou encore la connaissance du principe des MIB excluant d'office le choix de ce protocole. Quant à l'implémentation d'un protocole via TCP ou UDP, la question était de déterminer l'avantage réel, puisque j'avais de toute façon un serveur HTTP à disposition (celui chargé de l'interface), prenant en charge tout ce dont j'avais besoin. De plus, les divers clients peuvent faire appel aux bibliothèques HTTP disponibles dans tous les langages, ou utiliser des programmes Linux comme WGET directement appellés par un script.

Chaque compteur (représenté par l'objet Meter) dispose d'un hash, une séquence de 12 lettres majuscules, minuscules et chiffres, donnée à l'utilisateur lorsqu'il définit un compteur dans sa maison. L'utilité de ce hash est de disposer d'un identifiant "secret" pour le compteur qu'un logiciel ne pourrait pas trouver par force brute pour envoyer ou lire de fausses données. Pour ajouter un relevé, le client n'a qu'à émettre une requête HTTP(s) sous la forme :

https://server.com/data/put/HASH/123,456 où 123,456 est la valeur lue dans l'unité standard tel que définie dans les paramètres de l'interface.

L'interface dispose d'un système d'authentification avec nom d'utilisateur et mot de passe, mais /data en est dépourvu, car le hash est suffisant pour garantir l'authentification de la source. L'authentification requiert un envoi via une requête POST.

Cependant, le protocole http est interceptable : c'est pour cette raison qu'il est conseillé d'utiliser l'HTTPS et non l'HTTP, afin d'empêcher quiconque de lire les données en écoutant le trafic sur le réseau, mais surtout de trouver le hash. Notons qu'il faut également un certificat authentifié si l'on veut se protéger de l'attaque de l'homme au milieu. En effet, un ordinateur pourrait se mettre entre le client et l'interface, et utiliser un serveur HTTPS qui comprend la requête et la renvoie à l'interface de façon invisible. Cependant, il ne pourra pas avoir un certificat authentifié par une autorité valable pour notre serveur.

Pour lire la dernière donnée en date, le client peut utiliser :

https://server.com/data/get/hash/

Cela est notamment utile pour les boutons qui doivent lire la valeur du compteur afin de savoir si l'appareil est sensé être allumé ou éteint. Pour avoir une bonne réactivité, il doit donc le faire souvent. La méthode peut étonner à première vue puisque une requête de la part du serveur au client à l'instant même du changement mènerait à une bien plus grande réactivité. Là encore, le choix vient de l'expérience réelle : le client sera derrière un routeur, pratiquant le NAT. Nous voulions un système non intrusif ne requérant pas de compétences, excluant donc la redirection de port manuelle. La redirection de port pourrait se faire en UPNP automatiquement, mais de nouveau cela impose un client assez compétent. Ici, l'utilisateur n'a qu'à configurer le serveur de l'interface et le hash.

Un dernier avantage de l'HTTP est qu'il passe très bien les firewall, mieux qu'un protocole TCP maison sur un port inconnu (bloqué par exemple à l'université où était notre serveur), ou qu'un protocole UDP, lui aussi souvent refusé. Le désavantage premier est qu'il rend plus difficile le monitoring. En effet, de plus en plus de développeurs aboutissent à la même conclusion et utilisent l'HTTP, obligeant les administrateurs à faire du "deep packet inspection", pour voir quels sont réellement les types de données échangées.

De plus, cette simplicité permet de mettre rapidement en place des compteurs virtuels qui seront présentés dans le chapitre 5, ou même de parser les données d'autres capteurs ou de systèmes semblable au nôtre pour les envoyer à notre interface, permettant ainsi des "couches de compatibilité".

4.6 Traitement des données

A la réception d'une donnée, l'application data se charge de l'enregistrer, tout en mettant à jour le compteur de consommation de la journée.

Le calcul de la consommation totale doit se faire en fonction du type du compteur. En effet, si le compteur est en mode instantané, la consommation totale à ajouter sera la valeur indiquée en instantané, multipliée par le temps depuis le dernier relevé. Pour les compteurs qui envoient un total de consommation depuis la mise en service, il suffit de soustraire à la dernière valeur envoyée celle de la dernière valeur envoyée la veille, pour avoir la consommation de la journée. Quant aux compteurs différentiels, il suffit d'ajouter la consommation envoyée à la journée courante. Tous ces calculs se font en interceptant la fonction `save()` du modèle des relevés, *Reading* (voir `data/models.py`).

Un choix plus correct pour l'envoi de la mesure en mode instantané consisterait à considérer la moyenne de la valeur envoyée et celle de la dernière valeur, et de multiplier cette consommation instantanée par le temps entre les deux consommations. En réalité, considérer que la consommation de la dernière période est celle du dernier relevé permet au client de définir la consommation qui est apparue en cas de panne ou d'arrêt du compteur. Imaginons qu'un compteur envoie des consommations instantanées toutes les 5 minutes. Si la connexion est coupée 30 minutes mais que l'appareil continue de consommer, alors la consommation instantanée sera multipliée par 30 minutes dès le rétablissement de la connexion et le total sera correct. Cependant dans le cas d'une panne électrique générale où l'appareil s'est éteint également, il ne faut pas que lorsque l'appareil et son compteur redémarrent, le système considère que l'appareil a consommé pendant cette période. La solution est d'envoyer une consommation instantanée de 0 au redémarrage de la machine, et reprendre ensuite normalement. Ainsi la période d'arrêt sera multipliée par 0, et les périodes suivantes seront multipliées normalement. Cette méthode permet de s'affranchir de systèmes de timeout, et de laisser le client gérer la reprise en cas d'échec, en fonction du scénario. Cette méthode est utilisée pour la consommation d'un ordinateur qui se mesure lui-même. Lorsqu'il démarre, il envoie un 0 pour signifier qu'il n'a rien consommé depuis le dernier relevé (en pratique, il envoie une valeur très faible pour signifier une consommation de veille).

4.6.1 Pre-processing

Certaines données peuvent cependant être ignorées pour alléger la base de données. Par exemple, si le compteur est en mode total, il est inutile d'enregistrer plus de deux fois la même valeur. Il est cependant utile de garder en mémoire la première et la dernière fois que cette valeur a été lue, afin que l'utilisateur puisse constater qu'il y a eu un palier (en théorie la consommation d'eau en comporte beaucoup, lorsqu'il n'y a personne dans la maison). Sur la figure 4.13 représentant une consommation instantanée d'une telle situation, les trois points au centre du palier sont inutiles. Pour ne pas les enregistrer, le système vérifie avant d'enregistrer un relevé, que les



FIGURE 4.13 – Exemple de données de consommations instantanée reçues. Les 3 points centraux du plateau sont ignorés.

deux précédents ne sont pas équivalents. Si c'est le cas, il supprime le dernier en date et ajoute un nouveau relevé à l'heure actuelle. Ainsi, trois valeurs identiques n'apparaissent jamais d'affilée, et la première apparition du relevé est conservée, ainsi que la dernière en date.

Remarquons que la méthode s'applique aux compteurs instantanés, mais pas aux compteurs différentiels, puisque la suppression d'une valeur impliquerait un recalcul du total de la journée incorrecte. Les valeurs pouvant être ignorées pour ce dernier étant le 0.

Il est important de noter que les relevés enregistrés comme Reading sont dans le format original du compteur, mais ceux de ReadingDay présentent toujours la consommation totale de la journée. L'objet Energy contient les sigles des unités de consommation instantanée et de consommation totale pour l'affichage de celle-ci (ie. pour l'électricité, la consommation instantanée est en kW, la consommation totale en kWh).

4.6.2 Post-processing

A l'affichage des données dans l'interface, certains traitements sont également effectués sur la page d'affichage des consommations. Ce qui importe sur cette page (à l'opposé de la page statistique), c'est de donner un aperçu de sa consommation à l'utilisateur en un seul coup d'oeil : il doit voir une tendance et non une courbe exacte. Ainsi, on préfèrera les graphiques de la figure 4.15 à ceux de la figure 4.14. Pour cette raison, s'il y a plus de 30 relevés pour les dernières 24 heures, elles seront moyennées par tranche de $N/20$ arrondies au chiffre supérieur. Etant donné que les valeurs de températures d'une journée sont souvent discrètes, les compteurs de température sont toujours moyennés, quelque soit le nombre de relevés, afin de supprimer l'effet de palier qui ne représente pas la réalité. Notons que la dernière valeur est copiée en cas de moyennage afin que l'utilisateur ait l'heure du dernier relevé et la dernière valeur exacte sur le graphique.

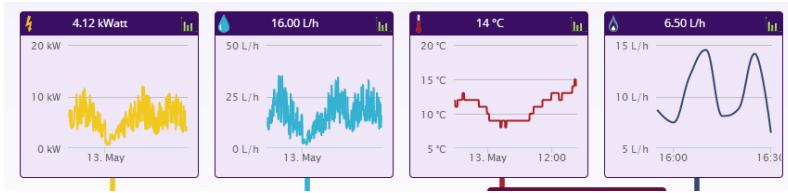


FIGURE 4.14 – Données des dernières 24 heures sans moyennage

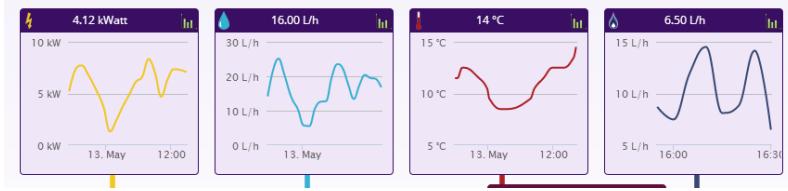


FIGURE 4.15 – Données des dernières 24 heures avec moyennage

4.7 Système d'alertes

La page servant à la définition du système d'alertes est composée d'une part des plans de chaque étage de la maison pour que l'utilisateur clique sur le compteur à mesurer, et d'autre part de la zone de placement des pièces, qui contient toujours une pièce AnalogMeter liée au compteur sélectionné.

Lorsque l'utilisateur choisit un compteur, la chaîne de pièces existantes est chargée depuis le serveur en AJAX par une requête GET. La vue Django correspondante (fonction *get* dans *alert/views.py*) imprime toute la structure de la chaîne en JSON, en créant une liste "outputs" contenant tous les éléments pointant vers l'AnalogMeter correspondant au Meter sélectionné. Il fait de même pour chacune des pièces trouvées, suivant ainsi toute la chaîne.

Comme pour le transfert des données de la structure de la maison, les objets anonymes reçus en JSON sont lus et changés par des objets JavaScript correspondant à chacun des types de pièces (voir *js/alert/alert.js*). Chacune de ces pièces étend également, comme pour l'implémentation de la base de données, un objet AlertModel. Chacune de ces classes implémente des fonctions tels que *getImage()* qui retournera l'image de la pièce du puzzle, *getText()* qui retournera un texte éventuel à afficher, etc. L'objet Board est chargé d'imprimer sur un canvas la chaîne de pièces, en appelant ces fonctions.

L'enregistrement de la chaîne de pièces se fait cette fois en JSON, la structure étant très commune avec les modèles Django. Une requête post contenant la structure au format JSON est envoyée au serveur et traitée par la fonction *save* (Voir *alert/views.py*). Cette vue exécute un travail très comparable à celui que fait le code JavaScript. Le format JSON est transformé en une série de liste Python et de structures du type dictionnaires, qu'il faut enregistrer en Modèles (qui sont pour

```


    /**
     * Initialise a new Plan
     * @param params object of parameters
     */
    function Plan(params) {
        this.params = params;
        var w = params.length;
        var w = params.width;
        if (params.precision == undefined)
            params.precision = grid_precision;
        this.precision = params.precision;
        if (params.canvas != params.precision)
            params.canvas = $('#Plan');
        params.canvas = params.canvas;
        //-----
        // Parameters
        //-----
        this.displayLabels = false;
        if (params.display_grid != undefined) this.draw_grid = params.display_grid; else this.draw_grid = true;
        if (params.display_global_meters != undefined) this.display_global_meters = params.display_global_meters; else this.display_global_meters = false;
        if (params.meterSize != undefined) this.meterSize = params.meterSize; else this.meterSize = 2;
        if (params.onemeter_max != undefined) this.onemeter_max = params.onemeter_max; else this.onemeter_max = 100;
        if (params.showBackwalls != undefined) this.showBackwalls = params.showBackwalls; else this.showBackwalls = false;
        if (params.enable_keys != undefined) this.enable_keys = params.enable_keys; else this.enable_keys = true;
        if (params.onemeter_min != undefined) this.onemeter_min = params.onemeter_min; else this.onemeter_min = 49;
        if (params.resize_height != undefined) this.resize_height = params.resize_height; else this.resize_height = true;
        if (params.enable_mouse != undefined) this.enable_mouse = params.enable_mouse; else this.enable_mouse = true;
        //-----
        // Canvas and sizes
    }


```

FIGURE 4.16 – Code JavaScript avant minification

rappel des objets Python tout à fait normaux) et enregistrer dans la base de données avant d'enregistrer chacun des ses "outputs".

Pour la génération des alertes et le suivi de la chaîne proprement dit, lorsqu'un relevé est enregistré, la fonction save() vérifie si un AnalogMeter est associé au compteur. Si c'est le cas, sa fonction fire() est appelée avec la valeur de la consommation instantanée en paramètre. Cet objet AnalogMeter appellera la fonction fire() de tous les AlertModel pointant vers lui, en toute logique des ADC. Les fonctions fire() des ADC vérifieront que la consommation antérieure est au-delà, égale ou en deçà du seuil en fonction du type choisi. S'il y a un événement, il appellera la fonction fire() de tous ses enfants, et ainsi de suite pour toute la chaîne jusqu'aux pièces Alert, qui produiront la notification définie par l'utilisateur.

4.8 Compression et concaténation du texte

Bien que le JavaScript ne permette pas d'inclusion nativement, il est nécessaire de séparer le code en plusieurs fichiers qui seront inclus via des balises HTML pour conserver une organisation propre. En effet, le code de dessin du plan par exemple contient plus de 1800 lignes.

Le problème de la séparation des fichiers JavaScript et CSS, est qu'à chaque fichier, il faut effectuer une requête HTTP supplémentaire. Même si ces requêtes peuvent être faites en parallèle, elle ne peuvent être plus courtes qu'un RTT (RoundTripTime, temps qu'il faut pour qu'une requête soit envoyée au serveur et que la réponse soit reçue). Dès lors, il faut en théorie, faire le choix de la performance (en écrivant tout le code dans le même fichier), ou de la lisibilité (en séparant le code en plusieurs fichiers).

La solution, apportée par le plugin django-pipeline[17] est la concaténation des fichiers en groupes. Par exemple, le groupe plan combine tous les fichiers relatifs au dessin du plan en un seul fichier.

De plus, ces fichiers sont au passage minifiés. La minification est un procédé

FIGURE 4.17 – Code JavaScript après minification

permettant la compression en taille d'un fichier. Le minificateur ([yuglify](#) [59]) supprime tous les espaces et retours à la ligne inutiles du fichier et les commentaires. En JavaScript, il renomme également toutes les variables locales avec des noms courts tel que `a`, `b`, `c`... Ainsi, le code de la figure 4.16 est transformé en code de la figure 4.17. Les fichiers CSS sont également minifiés.

Le code HTML généré par les différentes vues Django est également minifié, et tous les fichiers servis sont compressés au format gzip pour diminuer au maximum la taille des nombreux échanges.

Chapitre 5

Compteurs virtuels

Pour certains compteurs il est possible d'approximer une consommation de façon très correcte. Pour cela j'ai mis en place sur le serveur quelques petits scripts appelés toutes les 5 minutes via l'échéanceur Cron déjà présenté dans la section 2.1.1.

Tous les scripts impriment sur la sortie standard, une valeur de consommation à envoyer à un compteur. Il suffit ensuite d'utiliser le programme wget[56] pour lancer la requête HTTPS au serveur. Un exemple d'appel est la commande `wget -spider https://server.com/data/hash/'Python path/temperature.py BEXX0011'`. Le paramètre `-spider` de wget permet de ne pas réellement télécharger la page (envoie une requête HEAD plutôt que GET) puisque le résultat ne nous intéresse pas. L'utilisation des symboles apostrophe française ‘ permet de remplacer la commande entre les symboles par la sortie que celle-ci aura produite, et donc remplacer la fin de l'URL par la sortie du script `temperature.py`.

Pour envoyer des données toutes les 5 minutes, il suffit d'ajouter la ligne suivante au fichier de configuration de cron (en général `/etc/crontab`) :

`*/5 * * * * root commande où commande est à remplacer par la ligne citée ci-dessus.`

5.1 Génération de données aléatoires

Les premiers scripts servaient à générer des données aléatoires (en fonction des heures de pointe tout de même), afin de tester l'interface, tandis que les équipes IdP terminaient les capteurs. Ils sont disponibles dans le dossier `scripts` du code source.

Le premier est `raval.py` qui prend en paramètre une valeur, et multipliera cette valeur par deux nombres variant tous deux entre 0.5 et 1.5. Le premier multiplicateur est en fonction de la journée, c'est une simple fonction qui permettra de générer 3 pics à 8h, 12h, 16h et 20h comme sur le profil de consommation typique de la figure 5.1. Le multiplicateur final est visible sur la figure 5.2. Le deuxième multiplicateur est simplement un nombre aléatoire entre 0.5 et 1.5, ce qui permet de donner un air plus réaliste aux valeurs, bien que beaucoup plus oscillantes puisque lors de chaque

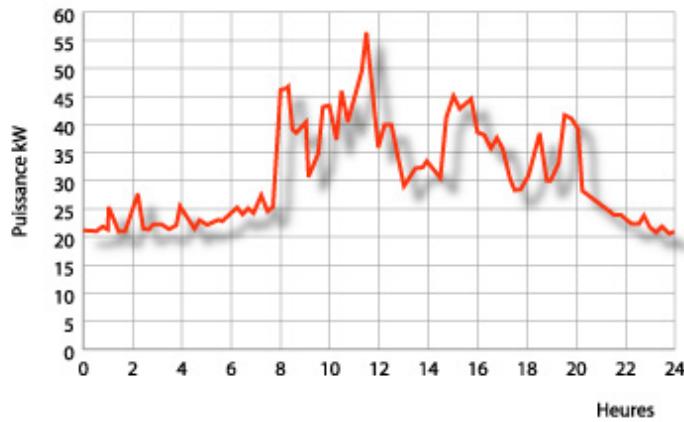


FIGURE 5.1 – Profil de consommation typique d'une journée. Source :[51]

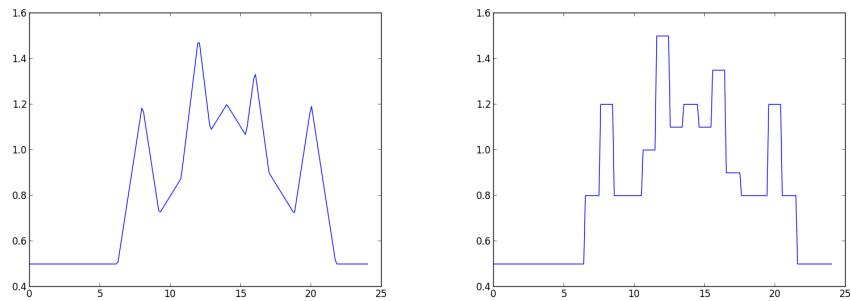


FIGURE 5.2 – Premier multiplicateur. En fonction de l'avancement de la journée et en fonction de l'heure (version utilisée)). Axes verticaux : valeur générée. Axes horizontaux : heure de la journée.

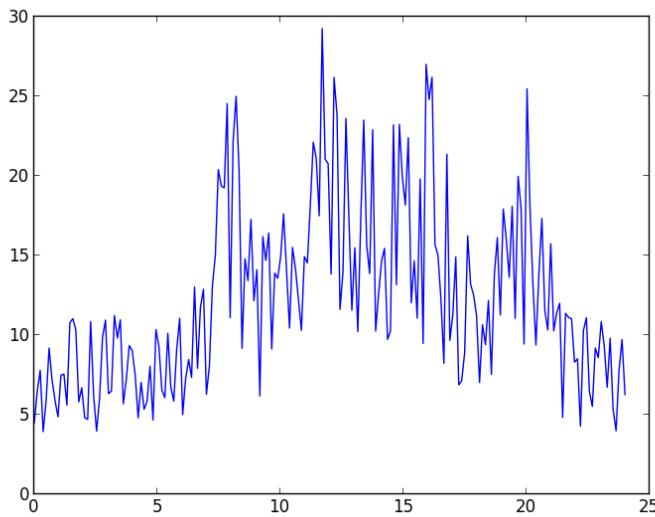


FIGURE 5.3 – Profil final généré avec une valeur de base de consommation de 15. Axe vertical : valeur générée. Axe horizontal : heure de la journée.

appel du script la nouvelle valeur n'aura aucun lien avec la précédente, si ce n'est la base et l'heure. La figure 5.3 montre le résultat pour une valeur de base de 15.

Le script *getval.py* qui permet de récupérer la valeur courante d'un compteur grâce à un appel GET via la librairie *httplib*, et *rangetval.py* combine les deux afin de récupérer la valeur actuelle d'un compteur et lui ajouter la valeur générée. Ce script est utile pour simuler une consommation sur un compteur en mode "total", puisque les consommations envoyées doivent être ajoutées au dernier total.

5.2 Température

J'ai également fait un script qui va chercher la température extérieure sur weather.com grâce à la librairie *pywapi*. En effet, il serait inutile de placer un thermomètre intelligent à l'extérieur pour mesurer la température, puisque la température extérieure est déjà disponible en ligne. Il est utilisé de la même façon que les autres au sein de cron pour publier la température toutes les 5 minutes sur un compteur "global" de température de la maison.

5.3 Consommation d'un ordinateur

L'idée ici repose sur la capacité de l'ordinateur à estimer sa propre consommation. Lorsqu'il est allumé, il envoie une consommation instantanée en fonction de l'état de

ses appareils, de son nombre de disque dur, de son écran, etc... Lorsqu'il démarre, il est nécessaire d'envoyer la valeur moyenne de la consommation en veille. Comme expliqué dans la section 4.2, la consommation totale est calculée en fonction de la consommation instantanée pour la période précédent l'envoi. Ainsi, si l'ordinateur a été éteint 3 heures, s'il démarre en envoyant 1.5Watts, le système considérera que l'ordinateur a consommé 1.5Watts pendant 3 heures.

Ma recherche en matière de documentation sur la consommation d'un ordinateur a été peu fructueuse. Je me suis basé sur quelques analyses générales dont l'article [5], mais soit ce sont des analyses de profil sur l'année ou de la consommation totale de l'ordinateur, soit ce sont des analyses très fines d'un composant précis dont l'accès aux paramètres comme le temps de "spin-up" d'un disque dur (démarrage d'un disque dur) n'est pas accessible de façon automatique.

J'ai donc étudié chacun des composants d'un ordinateur un par un pour essayer d'obtenir une consommation totale approximée de l'ordinateur. Le but de cette recherche est surtout de montrer les possibilités d'extensions que la facilité du protocole permet, et d'explorer quelques pistes pour se passer de certains capteurs. Beaucoup d'autres idées seraient réalisables, telle qu'activer la webcam d'un ordinateur pour vérifier si la lumière de la pièce est allumée toutes les 5 minutes et envoyer alors la consommation d'une ampoule. Le script final *scripts/computer.py* n'a donc pas pour vocation d'être complet et n'est testé que sur quelques machines. Il ne supporte que les systèmes Unix.

Le premier élément à prendre en compte est le type d'ordinateur. En pratique, il faudrait au moins séparer les ultra portables, les portables, les ordinateurs de bureau et les serveurs. J'ai décidé de ne garder que deux catégories les plus courantes dans une maison : les portables et les ordinateurs de bureau. Pour détecter si l'ordinateur est un mobile, le script teste l'existence du dossier `/proc/acpi/button/lid/` qui indique si l'écran de l'ordinateur portable est ouvert ou fermé. Si ce dossier n'existe pas, c'est qu'il ne s'agit probablement pas d'un ordinateur portable. Le type d'ordinateur n'implique pas de consommation mais sera utilisé par le reste du script.

L'étape suivante est le calcul de la consommation des disques durs. Le script liste les périphériques dans `/dev/sd*` et utilise `hdparm` pour voir si c'est un SSD, un HDD ou autre chose. En réalité, on utilisera le fait que `hdparm` retourne une erreur si le disque n'est ni un SSD ou un HDD, il s'agit alors probablement d'une clé usb ou d'un lecteur CD. Les consommations sont les moyennes tirées des produits de WesternDigital[54] des consommations de ses disques durs 3.5 pouces(6W) pour les ordinateurs de bureau et 2.5 pouces (2W) pour les portables. Concernant les SDD, ce sont celles d'OCZ[36] qui ont été calculées (0.8W) et une moyenne de 0.5W pour les autres appareils pour indiquer en moyenne une légère consommation de la clé usb ou du lecteur cd, la plupart du temps en veille.

Pour la carte mère et la mémoire, après quelques recherches il semble correct de considérer 8Watts pour les cartes mères d'ordinateurs portables, 20Watts pour celles des ordinateurs de bureau d'entrée de gamme et 30Watts pour les plus grosses avec les derniers chipsets. Ne pouvant pas détecter si la carte mère est "grosse", je compte le nombre de slot de RAM qu'elle possède en parseant le retour de la commande *dmidecode*. Je la considère petite si elle a 2 RAM ou moins, et elle sera considérée comme grosse si elle a plus de 2 RAM. Je compte également 2 Watts par Go de mémoire pour les desktops et 1W par Go de mémoire pour les mobiles. L'utilisation en électricité d'une mémoire RAM ne varie pas énormément en fonction de la charge puisqu'elle se réécrit perpétuellement de toute façon. Il est de plus difficile de mesurer le nombre d'opérations IO de mémoire sans accès au kernel.

Concernant l'écran, j'ai observé une différence de 3Watts entre un écran éteint (en veille profonde) ou allumé sur les deux ordinateurs portables à ma disposition et d'environ 8Watts de consommation pour mes trois LCD. L'état de l'écran est récupéré via la sortie de "wset -q dpms", indiquant si l'écran est en veille ou non. Cependant cette commande ne fonctionne que si l'accès à l'écran est configuré via la variable d'environnement Unix *DISPLAY*, ce qui n'est pas le cas avec Cron, à moins d'effectuer un *export DISPLAY* avant de lancer le script via Cron.

Pour la carte graphique, la variation peut bouger beaucoup selon le modèle. Je me suis uniquement focalisé sur les cartes NVIDIA n'ayant que des ordinateurs équipés de celles-ci ou de chipsets graphiques intégrés. Les autres cartes que les NVIDIA sont considérés comme des chipsets graphiques intégrés (1.5W pour un mobile, 3W pour un desktop). Le modèle de la carte graphique est détecté grâce à *lspci*. Notons que le script supporte plusieurs cartes graphiques (une configuration dite SLI). La consommation des cartes est peu précisée sur le site des constructeurs, en revanche on dispose souvent du TDP (Thermal Design Power) des chipsets qui indique la puissance dissipée. Ce n'est pas équivalent à la puissance consommée, mais cela peut être utilisé pour avoir un ordre de grandeur ou une proportion. En parcourant la liste des cartes graphiques NVIDIA sur *Wikipedia - Comparison of Nvidia graphics processing units* [57], on observe que de façon générale la puissance dissipée augmente avec le numéro du modèle au sein de la série, et que les maxima de chaque série augmentent avec au fur et à mesure du renouvellement des séries. A l'observation du résultat, j'ai considéré que la consommation maximale d'une carte mobile était de 5W à 60W en fonction de la gamme, et de 20W à 200W pour les desktops. Les cartes plus anciennes (les modèles disposant de 4 chiffres) sont considérés comme ayant une consommation maximale 70% plus petite. Les modèles mobiles finissent tous par M après les chiffres du modèle ceci permettant de les détecter. La consommation de la carte (à charge pleine) étant choisi dans l'intervalle cité en proportion de la gamme. Par exemple, un modèle 8600M a pour numéro de série 8, de gamme 6 et est mobile. La gamme 8 étant une ancienne, on considère que le plus petit modèle (la 8000) consomme 5W, et la 8900 consomme 60W. Comme c'est un ancien modèle (4 chiffres) cette consommation est rabaissée de 30%, la consommation de la série variant donc

entre 4W et 42W. Comme la gamme est “6”, on utilisera comme consommation une proportion de 6/10 entre 4 et 42W, soit 26W. Il conviendrait en pratique d’utiliser la même méthode au moins pour les cartes AMD ATI.

Ceci nous donne une information sur la consommation maximale de la carte, mais il est difficile de trouver la charge du GPU. A la place, j’ai utilisé la constante de 10%. Puisque le script tourne sur un système Unix, il est rare que la carte soit réellement utilisée, hormis pour la lecture d’une vidéo.

La détection des ventilateurs se fait par le nombre d’occurrences de *sysclassthermalcooling_device/cur_state* à 1. Il se peut que le driver de support des ventilateurs ne soit pas compatible avec le système. Si aucun ventilateur n’est détecté, on considère qu’il y en a au moins 1 (pour le processeur). Les desktops ont également un ventilateur ajouté pour l’alimentation. En effet celui-ci n’est presque jamais monitoré et donc indétecté par cette méthode. Chaque ventilateur est considéré comme consommant 1.5Watts.

La consommation finale est divisée selon l’efficience de l’alimentation. En effet la conversion du 220V en 12V et 5V a un coût. Sur les desktops, l’efficacité est en moyenne de 80%, c’est à dire que si le résultat de notre calcul de consommation est de 80Watts, l’alimentation en consommera en réalité 100. De plus, l’alimentation induit une légère perte constante (assumée de 3Watts). Sur les mobiles, l’alimentation est considérée comme, en moyenne, plus efficace : 90% avec une perte constante de 0.5Watts.

La partie la plus difficile est probablement le CPU. J’ai étudié le profil de consommation de 4 ordinateurs à ma disposition. De façon générale, j’ai envisagé deux cas de consommation : la consommation lorsque l’ordinateur ne travaille pas et la consommation lorsque tous les corps du CPU sont à 100%. J’ai utilisé un wattmètre pour mesurer la consommation de certains ordinateurs en ma possession et le logiciel cpuburn [10] pour faire tourner le processeur au besoin. L’établissement d’un profil de consommation en fonction de l’utilisation du CPU m’a permis de voir que la consommation était proportionnelle à la charge du CPU, sauf quand un mécanisme de down-clocking est utilisé (descente de fréquence du CPU s’il n’est pas utilisé) auquel cas la consommation a plus l’allure d’une courbe exponentielle. Ceci est confirmé par d’autres observateurs tels que [22]. En pratique, il est difficile de voir avec un simple script si le processeur possède un tel mécanisme (il serait possible d’analyser dans le temps le processeur pour voir si la fréquence diminue parfois). J’assume donc que le mécanisme d’économie d’énergie est actif sur les ordinateurs portables mais pas sur les ordinateurs de bureau. Il reste à trouver la consommation maximale du processeur, le script parcourt */proc/cpuinfo* à la recherche du nom du modèle. La consommation d’un processeur peut varier énormément d’un modèle à l’autre, et aucune caractéristique ne peut en laisser deviner la consommation. Je me suis uniquement focalisé sur les processeurs Intel, n’ayant aussi que ceux-ci sous la main (tout comme les GPU, il conviendrait en pratique d’appliquer la méthode pour tous les constructeurs). Le script recherche sur le site ARK d’intel [26] le modèle du

processeur. Il parse la page pour trouver le lien vers la page d'aspect technique et charge cette deuxième page. Sur cette dernière est indiqué le TDP du processeur déjà discuté concernant les GPU. C'est sur base de ce TDP que l'on pourra considérer la consommation du processeur. En réalité, je considère que la consommation maximale du processeur est de 60% du TDP pour un mobile et 75% pour un desktop. Cette valeur est quelque peu arbitraire et elle a été choisie en regardant "ce qu'il manque" entre la consommation simulée de mes 4 appareils et leurs consommations réelles ainsi qu'en récoltant diverses informations sur les forums et divers articles. La consommation actuelle d'un processeur est calculée ensuite comme la somme de 10% de sa consommation maximale (même sans aucune charge, il consomme) et des 90% restants majorés en fonction de sa charge.

Voici le résultat des estimations et des mesures réelles :

Type	Portable	Desktop	Portale	Desktop	Mobile
Processeur	T9300	i5 3550	P6200	G1610	G1610
Idle réel	20	143	20	29	-
Idle simulé	25	123	22	65	30
Full réel	33	193	39	35	-
Full simulé	44	195	44	119	62

Le premier mauvais résultat du Celeron G1610 s'explique par le fait qu'il est dans une plateforme utilisée comme serveur. L'alimentation est une alimentation haut de gamme très efficace tandis que la carte mère est d'entrée de gamme même si elle comprend un bon chipset assurant une consommation minime. Le processeur Celeron consomme très peu malgré un TDP du CPU annoncé de 55 Watts. La tour complète sans écran ne consomme que maximum 27 Watts à pleine charge, du coup, ma présomption sur le TDP semble incorrecte dans ce cas. Cet ordinateur particulier peut en fait être considéré comme un ordinateur portable, j'ai donc décidé après ces résultats de déclarer l'ordinateur comme mobile également lors de la détection d'un CPU Celeron. Les consommations sont plus correctes mais le TDP du CPU continue d'influencer la consommation en charge complète en l'augmentant de façon trop élevée à cause du chiffre du TDP de 55W donné par Intel, même majoré par la conversion TDP vers sa consommation maximale réelle.

Quelques essais m'ont permis d'assumer que la consommation par un vrai programme (avec des accès disques, des échanges mémoires, etc) ne changeait pas les résultats par rapport à l'utilisation du logiciel cpuburn pour simuler la charge. Ceci est explicable par le fait que si effectivement un logiciel qui crée beaucoup d'accès disque augmente la consommation au sein de celui-ci, il réduit sa consommation du processeur qui est en état d'attente.

En conclusion, le script permet d'avoir une idée de la consommation estimée, raisonnablement exacte. Ce qui permet à l'utilisateur de donner une approximation relativement correcte sur l'année. Un utilisateur plus scrupuleux peut paramétriser le script en fonction de son ordinateur, en l'utilisant comme une base s'adaptant

grossièrement à cet ordinateur. Ici, on pourrait par exemple déclarer que les Celeron ne consomment au maximum que 20% de leurs TDP. Il ne serait de toute façon pas possible de trouver automatiquement la consommation exacte de l'ordinateur, même si ce compteur virtuel était l'unique sujet de ce travail. Même en envisageant un parsing du site des constructeurs comme pour les CPU, nous ne pourrions jamais avoir toutes les données automatiquement. La consommation d'une carte mère par exemple est très hasardeuse et dépend de chacun de ses composants, il n'est même pas sûr que les fabricants la connaissent eux-mêmes. Le boîtier est également un problème insoluble. Certains boîtiers d'ordinateur s'alimentent sur l'alimentation générale pour des éclairages, des ventilateurs ou des petits écrans LCD de contrôle mais les modèles du boîtier sont souvent inconnus de l'ordinateur lui-même et rien n'est monitoré par l'ordinateur lui-même mais bien par un circuit indépendant. Seul un wattmètre permettrait de mesurer cette consommation de façon exacte.

Pour rappel, le but de ce script était d'une part de me permettre de tester mon interface avec des données qui ne sont pas aléatoires en attendant les capteurs des équipes IdP, et, d'autre part, d'explorer un peu plus en profondeur les possibilités de compteurs virtuels. Je pense avoir atteint ces objectifs. D'autres idées à implémenter seraient la détection de périphériques sur le réseau : modem, switch, imprimantes, ... et publier également leurs consommations sur l'interface. Il existe également beaucoup de possibilités dans le domaine de la vision par ordinateur : voir si une lampe est allumée (la consommation d'une lampe étant en général constante), ou, plus largement la détection de l'état de certains appareils ménagers. Par exemple, beaucoup de fours à micro-ondes possèdent une lumière qui pourrait être détectée sur une image, ou encore les leds d'état des télévisions et décodeurs.

Chapitre 6

Résultats

Cette section présentera l'interface terminée et la réalisation pratique des différents éléments discutés. Les questions de performances seront discutées dans le chapitre suivant.

Des trois objectifs du groupe IdP (Flexibilité, Accessibilité/Non-intrusif et peu coûteux), seules la flexibilité et l'accessibilité concernent vraiment l'interface. Le seul élément financier est celui de ne pas requérir des performances exagérées (ce but est relativement atteint et discuté dans la section évaluation des performances 6.8. Concernant l'accessibilité, un des objectifs majeurs de l'interface est de pouvoir fonctionner sur les smartphones, tablettes et PC. Les mécanismes mis en place pour atteindre cet objectif sont nombreux et discutés tout au long de ce travail. La flexibilité, elle, concerne la possibilité de pouvoir s'adapter à toutes les maisons, à de nouvelles énergies, ou encore à de nouveaux appareils électro-ménagés. Au niveau du code, il faut également une totale modularité, le moins possible de "hard codage", afin de permettre ces changements d'horizons.

L'utilisateur est catégorisé comme "standard" ou "avancé", il peut à tous moments changer son mode dans les paramètres.

La page d'accueil affiche un menu à l'utilisateur lui offrant 4 possibilités, correspondant aux 4 cas d'utilisation du logiciel :

- Consommation - Affichage des de consommation actuelle de l'utilisateur
- Modélisation du bâtiment - Création et configuration de la maison de l'utilisateur
- Alertes - Configuration du système d'alertes
- Statistiques - Affichage du profil de consommation sur les dernières jours, semaines ou mois.

La disposition des éléments a été soigneusement étudiée afin que l'utilisateur débutant ne se perde pas dans les diverses possibilités du système. Chaque icône du menu a une couleur correspondant à la page liée, et cette page est dans le même ton,

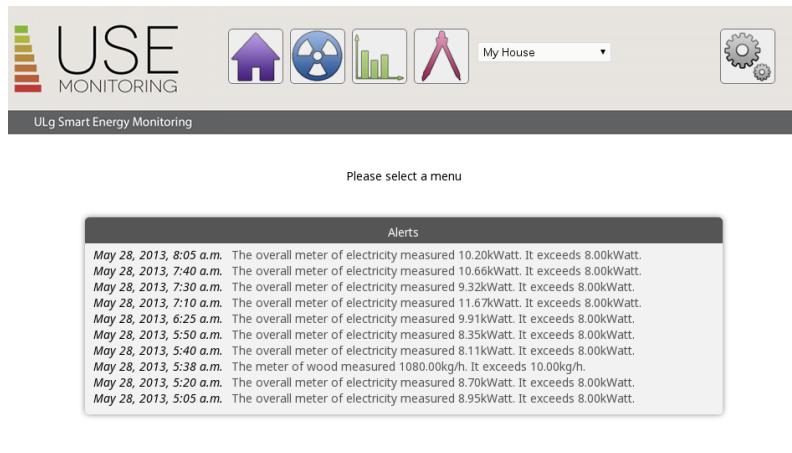


FIGURE 6.1 – Accueil

assurant un effet mémoire à l'utilisateur.

Pour cheminer en cohérence avec l'objectif d'accessibilité par l'utilisateur, presque tous les boutons ont été remplacés par des images. Cela permet également de limiter l'adaptation nécessaire pour les smartphones : les boutons carrés sont faciles à cibler du doigt à n'importe quel niveau de zoom, même sur des smartphones pourvus de petits écrans.

Les choix opérés au niveau de la forme de l'interface peuvent sembler anodins mais la programmation a été précédée de 3 mois de réflexion, en compagnie principalement des 3 étudiants en 2ème bachelier rattachés à notre groupe IdP. Ils m'ont aider à recueillir toutes les informations nécessaires pour le modèle énergétique (voir 7.5), ce qui était réellement attendu de l'interface, ou encore de décider du protocole de communication (discuté dans la section 4.5). Ils m'ont également aidé en apportant un avis extérieur, non-informaticien et non-initié sur mes idées (jusqu'alors sur papier), pour que tant l'utilisateur lambda que l'utilisateur avancée trouve ce qu'il cherchait dans USE Monitoring. Au final, j'ai fabriqué plus de 55 symboles originaux pour ne pas souffrir de copyright. Ma formation n'étant absolument pas celle d'un graphiste, j'ai utilisé un système de silhouettes, contenant peu de couleurs et des formes simples afin de les réaliser en utilisant des courbes de bézier, tenant plus de l'art de la manipulation de la souris que de l'art du pinceau (Fig 6.2).

6.1 Modélisation - *Builder*

La première tâche de l'utilisateur lorsqu'il accède à l'interface est de définir l'enveloppe énergétique de la maison. Biensûr la plupart des utilisateurs, même expérimentés ne connaissent pas le volume exact de leur habitation et moins encore le coefficient de perte énergétique. La solution retenue est de guider l'utilisateur à

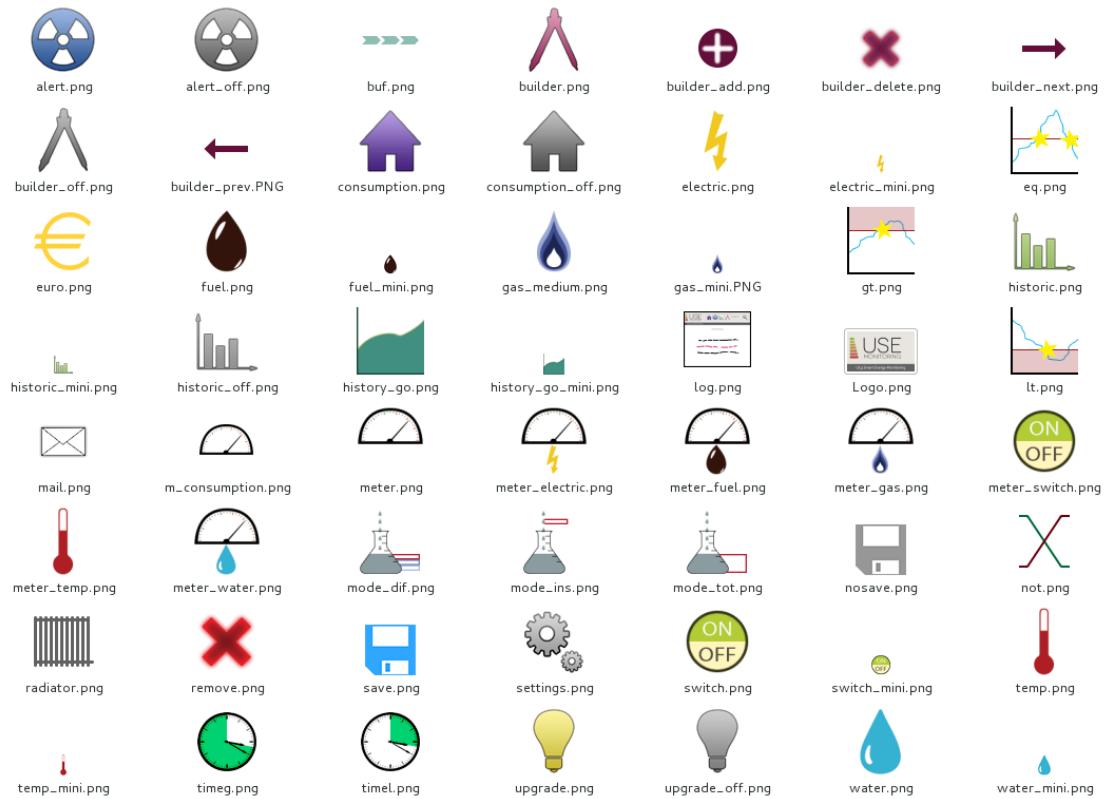


FIGURE 6.2 – Icônes, images et logos créés pour l’interface

travers 7 étapes légèrement différentes selon son niveau d’expertise.

Étape 1

La première étape (Fig. 6.3) définit la maison et ses caractéristiques générales.

L’utilisateur est invité à entrer :

- Un nom pour la maison : utilisé uniquement pour choisir la maison active dans la liste visible en haut dans le menu principal.
- L’année de construction : utile pour le modèle, mais également pour définir automatiquement le niveau d’isolation des murs.
- Type de maison : l’utilisateur a le choix entre un appartement et une maison 2, 3 ou 4 façades. Concernant celles-ci, nous parlons ici de maisons mitoyennes. La maison deux façades n’étant pas un mur seul mais une maison coincée entre deux autres.
- Situation : Urbaine, péri-urbaine ou rurale. Cette information est uniquement destinée au modèle énergétique.
- Longueur et largeur de la maison : ces données ne seront pas directement

utilisées mais permettront de définir la taille du plan. La superficie exacte de la maison est calculée à partir des murs extérieurs dessinés sur le plan.

Il serait également possible de changer tous les menus déroulant par une liste d'images pour les smartphones. L'utilisateur n'aurait qu'à toucher une image pour faire son choix. Les zones de chiffres auraient également pu être remplacées par des "sliders", une barre à déplacer pour choisir un nombre entre 0 et 100 par exemple. En pratique, les derniers navigateurs mobiles affichent un pop-up pour faire le choix aisément quand l'utilisateur touche la zone du menu déroulant. Concernant les zones de chiffres, l'affichage du clavier est toujours une opération pénible et peu fluide même sur les derniers modèles, me poussant à imaginer cette solution. Cette fonctionnalité peut être affichée au chapitre des "améliorations possibles".

L'utilisateur a également la possibilité de supprimer la maison s'il édite une maison existante plutôt qu'une nouvelle.

USE
MONITORING

ULg Smart Energy Monitoring

Builder
House

Name :	<input type="text" value="My House"/>
Year construct :	<input type="text" value="1950"/>
Type of house :	<input type="text" value="4 Face House"/>
Location :	<input type="text" value="Urban"/>
Maximum length of the house :	<input type="text" value="15"/>
Maximum width of the house :	<input type="text" value="5"/>

→ X

FIGURE 6.3 – Informations principales sur la maison

L'étape 1 est la seule obligatoire pour accéder à la suivante. Une fois la maison enregistrée, l'utilisateur peut naviguer entre les étapes et en sauter l'une ou l'autre à sa guise. Toutes les pages présentées ici s'ouvrent naturellement en mode "édition" ou en mode "création", en fonction de la sélection d'une maison actuelle ou non.

Étape 2

La seconde étape concerne les habitants de la maison et leur appartenance socio-professionnelle (Fig. 6.4). Celle-ci permettra au modèle d'évaluer un taux de présence et donc d'utilisation des appareils de la maison.

L'utilisateur définit un nom, un âge et un emploi. Il clique sur le bouton d'ajout pour inscrire l'habitant à la liste et peut enlever un habitant à tous moments.

The screenshot shows the USE MONITORING software interface. At the top, there is a navigation bar with icons for home, radiation, energy consumption, and people, followed by a dropdown menu 'My House' and a settings gear icon. Below the header, a dark banner reads 'Builder' and 'People living in the house'. The main area contains a form for adding a new inhabitant with fields for Name, Age, and Work status, and a '+' button. To the right, three existing inhabitants are listed: Tom (22, Student), Mum (55, Employee), and Dad (57, Employee), each with a 'Remove' link. At the bottom are left and right navigation arrows.

FIGURE 6.4 – Habitants de la maison

Étape 3

L'utilisateur doit ensuite définir les étages de sa maison : s'il dispose d'un grenier, le nombre d'étages, d'une cave, etc...

Il lui suffit de cliquer sur un type d'étage à droite (voir fig. 6.5) et celui-ci viendra s'ajouter sur le profil de la maison à gauche. Il peut également définir la hauteur d'un étage. L'utilisateur n'a pas la possibilité d'intégrer des étages à demi niveau. Si cela constitue une grande simplification au niveau du code, cela ne change rien au niveau du modèle énergétique. Celui-ci n'utilise que le volume globale des pièces et n'est donc pas influencé par un étage à un niveau décalé par rapport aux autres qui serait ramené au niveau inférieur ou supérieur.

Étape 4

Pour les quatre dernières étapes, l'utilisateur passe en mode plan (Fig. ??) afin de définir les murs, les fenêtres, les appareils et les éventuels compteurs énergétiques

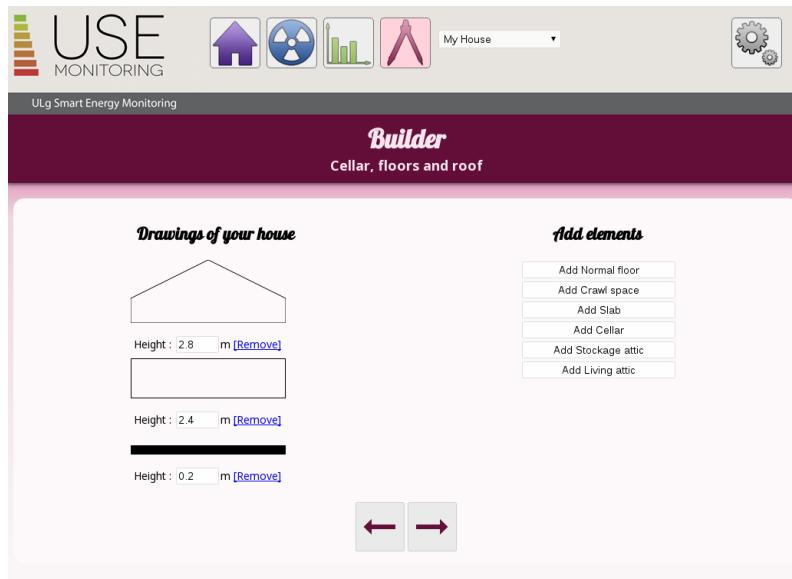


FIGURE 6.5 – Définition du profil de la maison

installés dans la maison. Le plan est à l'échelle (le quadrillage est un mètre). Sur toutes ces pages, il dispose d'un menu en haut à gauche pour naviguer entre les étages. Toutes les modifications sont enregistrées à chaque changement d'étage, et à chaque changement de page. Ces 4 étapes sont les seules de l'interface qui sont moins adaptées aux petits smartphones. En effet pour atteindre une précision suffisante l'utilisateur devra constamment zoomer et dé-zoomer. Cependant, peu d'utilisateurs configureront vraiment leur maison sur leur smartphone. Cette étape est plutôt destinée à être réalisée sur un PC et précède l'utilisation du smartphone pour la consultation de la consommation.

L'utilisateur est donc amené à configurer les murs de sa maison. En mode basique, il clique simplement sur deux points du plan pour tracer un mur entre les points. Pour simplifier les différentes étapes et le système, il ne peut faire que des murs verticaux et horizontaux et ce, sur les intersections de la grille du plan. Comme seul le volume général sera utilisé par le modèle, cela n'est pas un problème. Lorsque l'utilisateur passe à l'étage supérieur, une analyse en composante connexe est effectuée par le programme (sur base de la grille) afin de détecter quels sont les murs extérieurs et les dupliquer à l'étage supérieur. L'étage inférieur est affiché en transparence rouge afin que l'utilisateur puisse se repérer tel que représenté sur la figure 4.12.

En mode avancé, le menu visible sur la figure 6.6 est remplacé par celui visible sur la figure 6.7, pour qu'il puisse définir lui-même la taille du mur et de l'épaisseur de l'isolation. Le type d'isolant ne change que peu le facteur d'isolation et n'est donc pas demandé à l'utilisateur. En mode basique, le partitionnement de l'espace est calculé après chaque mur afin de détecter s'il est intérieur ou extérieur. La taille du mur et de l'isolant sont choisis en fonction de l'âge de la maison.

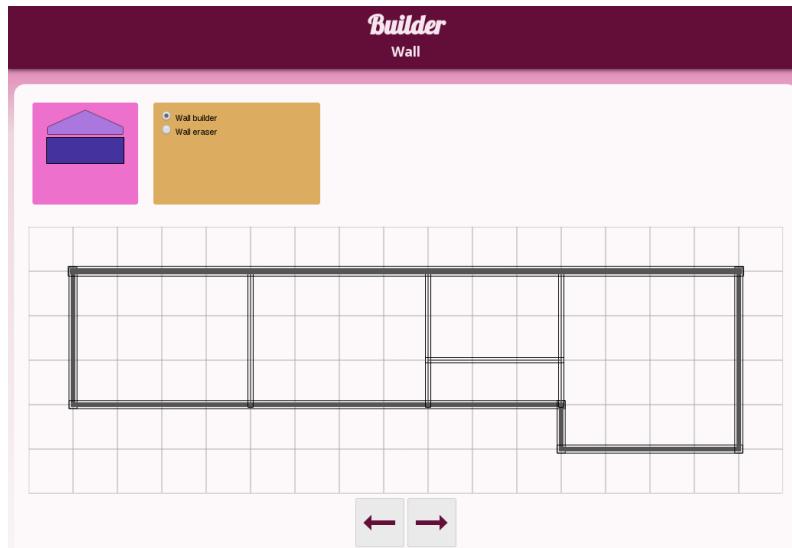


FIGURE 6.6 – Définition des murs



FIGURE 6.7 – Menu de définition des murs en mode avancé

Étape 5

L’utilisateur doit ensuite disposer les fenêtres, leurs tailles et leurs types : simple vitrage, double vitrage, double-vitrage haute efficacité ou triple vitrage (Fig. 6.8). La précision de la position des fenêtres est ici au demi-mètre près (bien que leur taille soit définie en virgules flottantes par l’utilisateur). Le fait qu’il n’y ait pas de support pour les fenêtres de toit simplifie également l’interface utilisateur sans impacter la justesse de la perte énergétique de l’enveloppe.

Étape 6

Pour la partie placement des appareils électroniques, l’utilisateur dispose d’un menu par catégorie pour placer ses appareils (voir Fig. 6.9)

Les catégories ne sont pas changeables par l’utilisateur (mais les différentes catégories ne sont mises qu’à un seul endroit côté serveur, donc facilement modifiables et l’interface s’adaptera automatiquement), cependant l’utilisateur a la possibilité d’ajouter d’autres appareils, de changer les énergies qu’il utilise, leurs noms, etc... Ceci sera discuté dans la section 6.5.

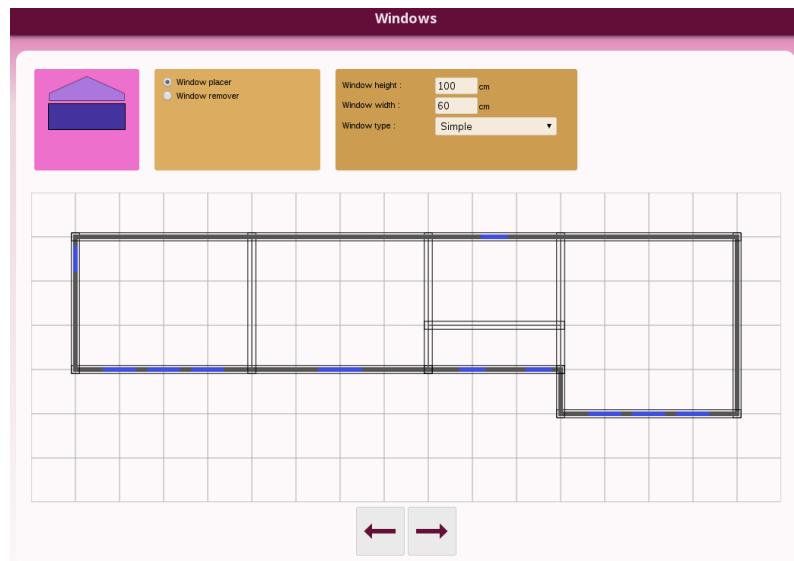


FIGURE 6.8 – Exemple de murs de l'étages supérieurs automatiquement créés. On aperçoit également l'étage inférieur en transparence.

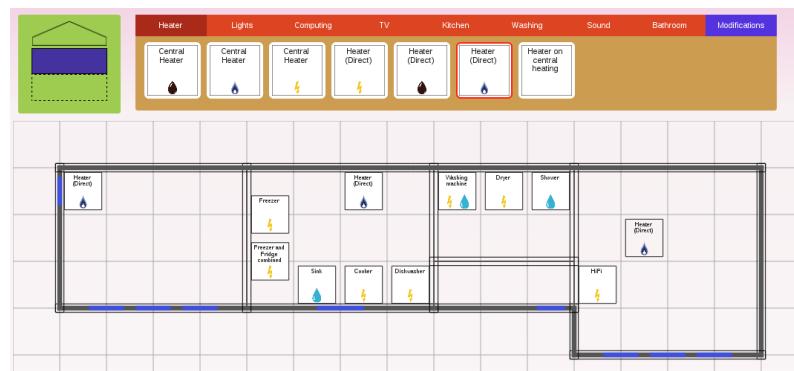


FIGURE 6.9 – Placement des appareils ménagers

Pour placer un appareil, il clique sur celui de son choix dans le menu, et clique à un endroit du plan.

Le type d'énergie est indiqué pour chaque appareil. L'utilisateur doit choisir un appareil et le placer sur le plan, étage par étage. Beaucoup d'informations dérivées seront tirées de ce placement comme la surface chauffée et les énergies utilisées.

Étape 7

L'étape suivante permet à l'utilisateur de cibler les compteurs dont il dispose et les relier aux appareils dont ils mesurent la consommation. Les compteurs peuvent également être identifiés en tant que compteur général de la maison.

Pour cette dernière étape, les grilles du plan disparaissent, les pièces sont colorées avec des fonds légèrement différents et le système calcule la taille réelle de la maison

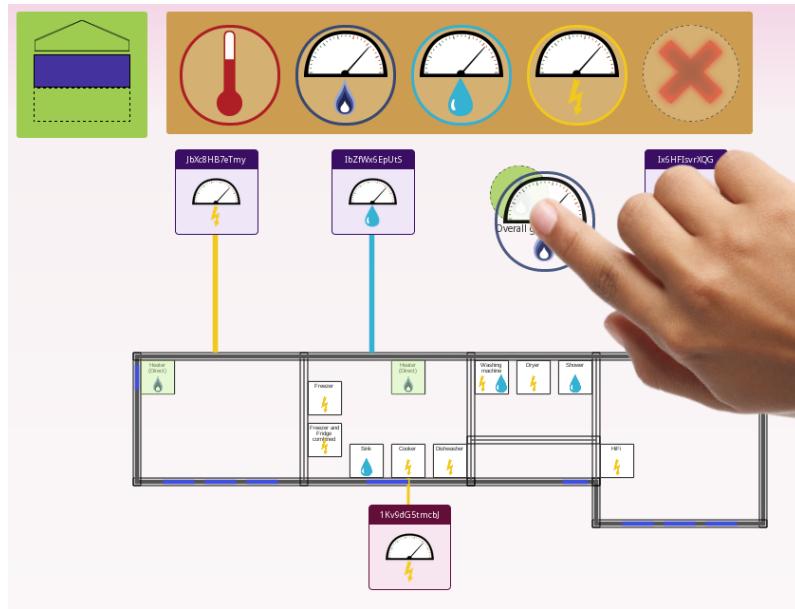


FIGURE 6.10 – Placement des compteurs. Comme l'utilisateur déplace un compteur gaz, seuls les appareils au gaz entrent en surbrillance et le compteur général de gaz.

pour l'agrandir au maximum sur l'écran. Nous avons voulu ainsi couper avec le reste des étapes précédentes, l'utilisateur ne définissant plus le plan mais l'utilisant. Ici, l'utilisateur est invité à faire un drag-n-drop qui est plus intuitif. Lorsqu'il commence le déplacement d'un compteur, les appareils et compteurs globaux de l'énergie du compteur se mettent en vert pour indiquer à l'utilisateur qu'il peut déposer le compteur à cet endroit tel que visible sur la figure 6.10. La méthode des appareils de l'étape précédente fonctionne toujours, à savoir un clic sur un compteur et puis sur l'appareil qu'il mesure, notamment pour une question de compatibilité avec le navigateur d'origine Android depuis sa version 4 qui casse le plugin javascript de drag-n-drop (ceci sera discuté plus longuement dans la section compatibilité 6.6).

Quand l'utilisateur place un compteur, une boîte de dialogue (fig. 6.11) lui permet ensuite de choisir le mode du compteur tel que défini dans la section 4.2, structure de la base de donnée.

Une fois le compteur placé sur le plan, la page affiche le "hash" utile au protocole d'échange présenté dans la section 4.5 protocole de communication. C'est un code de 14 chiffres/caractères que l'utilisateur emploiera pour paramétrier son appareil de mesure. Le compteur physique envoie les données mesurées associées au hash ; ainsi, le système peut savoir à quel compteur appartiennent les données qui lui arrivent.

6.2 Consommation - Consumption

La page consommation est destinée à donner un aperçu à l'utilisateur de la consommation de sa maison. A ce titre, il peut voir la consommation instantanée

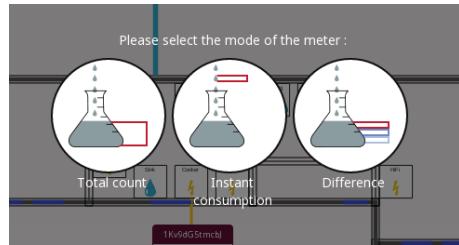


FIGURE 6.11 – Choix du type de compteur

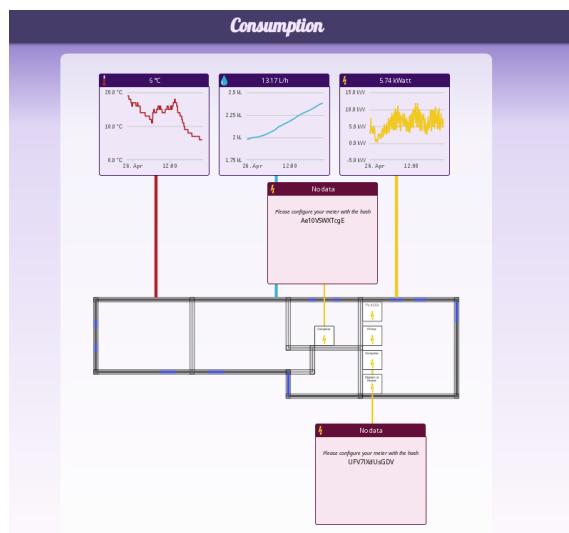


FIGURE 6.12 – Affichage de la consommation



FIGURE 6.13 – Affichage de la consommation en mode double

de tous ses appareils monitorés et un graphique de l'évolution de la consommation instantanée des dernières 24 heures pour chacun d'eux. Notons que la consommation sera instantanée quelque soit le type du compteur. S'il est en mode différentiel ou total, elle sera calculée. L'utilisateur peut également allumer ou éteindre les appareils qui disposent d'un allumage à distance. Chaque compteur dispose également d'un bouton qui renvoie vers un graphique de la consommation instantanée mais non limitée dans le temps (avec une période d'affichage libre) ainsi que la consommation totale journalière.

Pour atteindre cet objectif qui vise à ce que l'utilisateur ait un aperçu de sa consommation "au premier coup d'oeil", cette page a été l'objet d'une attention toute particulière au redimensionnement, au recoupage des zones du quadrillage vide et au placement des éléments que ce soit sur des écrans d'ordinateur ou smartphone. Si la taille de l'écran le permet, l'affichage passera en mode dual (Fig. 6.13).

6.3 Alertes - Alerts

L'utilisateur a la possibilité de spécifier des alertes en fonction de la consommation (instantanée) d'un appareil. Il peut recevoir une alerte par mail ou une notification sur la page d'accueil. Nous désirions une interface simple mais puissante, cette recherche nous a orientés vers un système de puzzle (voir fig. 6.14) où l'utilisateur voit directement quelles pièces il peut emboiter. S'il commence le glisser-déplacer, l'emplacement possible pour une pièce se met en surbrillance (Voir fig. 4.4). Sur les compteurs sortant des valeurs numériques (un bout rond), il devra forcément placer une pièce "Seuil" qui activera une sortie digitale (un bout en triangle) selon une valeur choisie. Ensuite viendra la pièce avec une entrée digitale qui permet d'activer une alerte. Il y a quelques pièces logiques digitales-digitales : NOT qui inversent le résultat d'une sortie, et deux filtres de temps. Un qui laisse passer une activation uniquement s'il est plus tôt qu'une certaine heure définie et l'autre uniquement s'il est plus tard. L'utilisateur en plaçant ces deux pièces l'une après l'autre peut donc

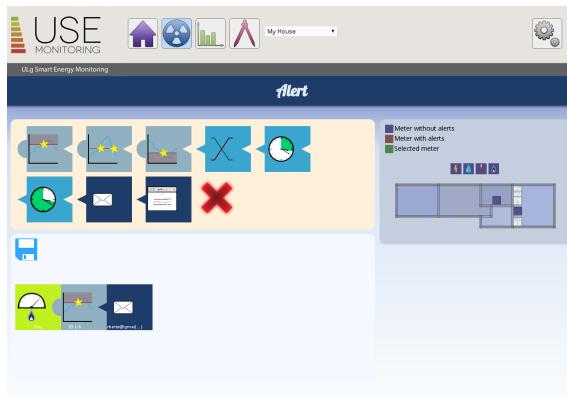


FIGURE 6.14 – Page de définition des alertes

définir un interval.

6.4 Statistiques - *Statistics*

La page de statistique 6.15 permet de visualiser la consommation de chaque jour en fonction du temps, selon une période choisie par l'utilisateur.

La page dispose également de deux graphiques montrant la consommation totale des 12 mois précédents, ainsi qu'un diagramme circulaire pour l'année écoulée. Ces deux graphiques peuvent afficher la consommation énergétique globale de la maison sous forme de plusieurs grandeurs :

- La consommation totale dans l'unité de l'énergie
- Le montant payé pour cette consommation
- Une comparaison en kWh de la consommation énergétique pure, en utilisant le pouvoir calorifique inférieur (PCI). De façon intuitive la consommation de chaque compteur est divisée par la capacité de l'énergie mesurée à produire de la chaleur en fonction de son unité. Cela permet une comparaison entre les énergies utilisées dans la maison.

6.4.1 Recherche en matière énergétique

C'est dans cette catégorie que le travail au sein d'une équipe pluridisciplinaire prend son sens, puisque l'expertise de mes collègues énergéticiens m'a permis de gagner pas mal de temps.

Concernant le PCI, le coefficient varie selon la pureté et le type exact de l'énergie (lorsque applicable). L'utilisateur plus scrupuleux peut donc changer son PCI, quand l'utilisateur basique gardera le PCI par défaut. En regardant le tableau 6.16, on voit que le gaz le plus utilisé est le gaz naturel et c'est donc le PCI de ce dernier qui est

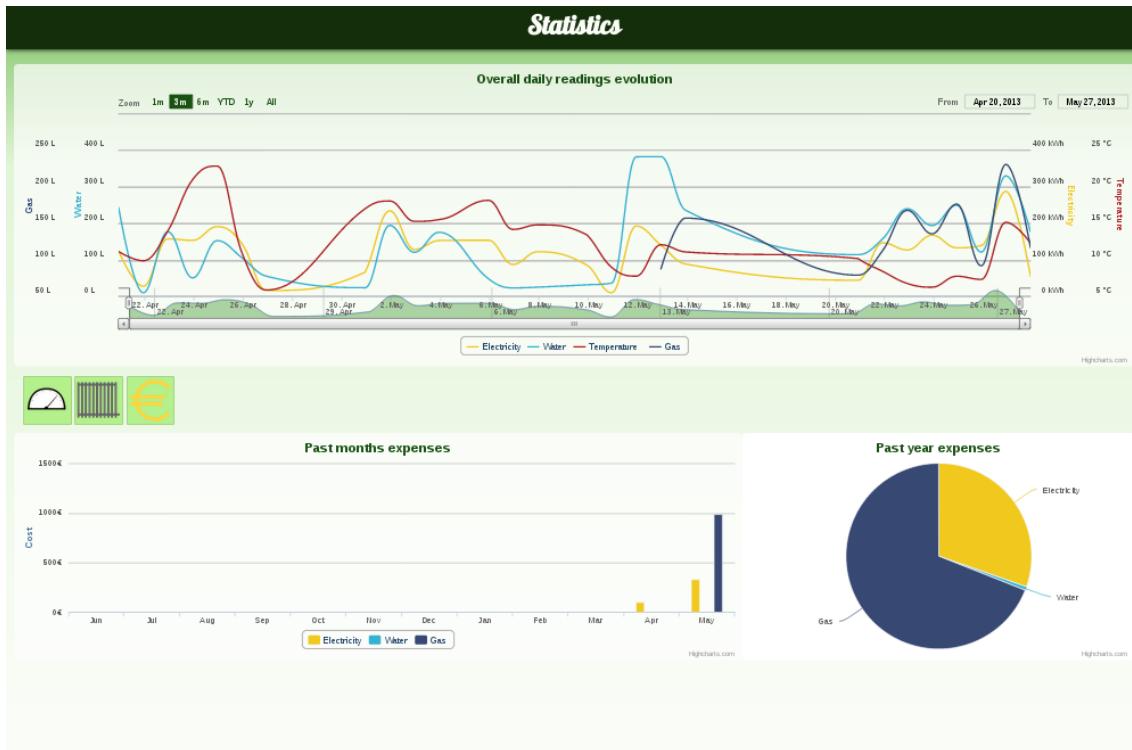


FIGURE 6.15 – Historique de la consommation

défini par défaut (le choix de simplification de ne proposer d'utiliser qu'un gaz ayant été fait à la base). L'utilisateur peut modifier le PCI des énergies ou peut rajouter des énergies et donc des PCI différents pour chacune d'elles. En pratique, plusieurs sources se contredisent ou visent simplement des produits légèrement différents mais dont les alliages ne sont pas spécifiés. J'ai pris les sources se rapprochant des énergies les plus utilisées et j'ai fait des moyennes pour les autres cas. Les principales sources sont ACQUALYS [2] et AXEIYA [3].

Le problème est plus complexe pour les prix des énergies. En effet, le prix varie beaucoup entre chaque fournisseur, il existe différentes offres, et pour une même offre d'un même fournisseur, ces prix sont souvent progressifs en fonction de la consommation annuelle.

	Gasoil	Bois	Charbon	Gaz naturel	Butane propane	Electricité	PAC et solaire	Géoth.	Vapeur Cogen	Total	% du Total
Chauffage	13 333	598	392	7 922	332	721	15.2	3	20.5	23 336	69%
	57.1%	2.6%	1.7%	33.9%	1.4%	3.1%	0.1%	0.0%	0.1%	100%	
Hors Chauffage	1 229	760	25	1 994	652	5 992	17.4		3.3	10 672	31%
Total	14 562	1 358	417	9 916	984	6 713	33	3	24	34 008	100%
	42.8%	4.0%	1.2%	29.2%	2.9%	19.7%	0.1%	0.0%	0.1%	100%	

FIGURE 6.16 – Consommation réelle du logement en 2006. Source :ICEEDD [23]

Ma première idée était d'implémenter un système d'interpolation. Par défaut (et toujours modifiable par l'utilisateur), en allant chercher des données sur le site de la *Société Wallone Des Eaux* [50] pour l'eau, on peut avoir le prix payé pour chaque interval de consommation annuel comme sur la figure 6.17. On peut aussi trouver le prix exact payé par l'utilisateur en prenant sa consommation des 365 derniers jours ou en multipliant les données pour obtenir un équivalent annuel. A cela il faut ajouter le prix de la redevance annuelle. Comme l'interface affiche le prix payé par jour ou par mois, il faudrait donc la diviser de façon équivalente sur les mois.

Une recherche plus poussée permettrait sans doute de trouver les catégories de consommation automatiquement. La consommation d'eau est par exemple presque proportionnelle au nombre d'habitants du ménage. C'est une information qui est définie par l'utilisateur en début de modélisation de sa maison.

Mais aucun de ces procédés n'est parfait car le prix varie toujours en fonction du fournisseur ou de la région. En pratique je manquais de temps et je n'ai pas développé un système aussi compliqué, laissant simplement une valeur unique de coût par unité. Que mettre alors comme prix par litre d'eau par exemple ? Un rapide coup d'oeil aux statistiques de la région wallonne montre qu'une large majorité des utilisateurs se trouvent dans la catégorie 30 à 5000, sachant que la moyenne de consommation s'élève à 110m³ par raccordement[23]. Au final c'est donc cette catégorie prix qui a été utilisée, légèrement augmentée pour tenir compte de la redevance.

De façon générale, les autres énergies sont soumises aux mêmes problèmes. Il appartient donc à l'utilisateur de changer le coût de ses énergies en fonction de sa dernière facture de régularisation annuelle par exemple. Au final, le prix unique corrigé par l'utilisateur donne même un meilleur résultat que le système de catégorie. En utilisant mon avant-dernière facture de régularisation d'électricité pour avoir un coût au kWh et calculer le prix de la suivante, j'obtiens un meilleur résultat qu'en utilisant un système de catégorie standard utilisant une approximation des catégories de deux fournisseurs (catégories qui sont d'ailleurs différentes), chacune avec une moyenne des prix de 2 fournisseurs. Ceci est notamment dû au fait que l'électricité est souvent sur un schéma bi-horaire, et dans mon cas, un tri-horaire. L'implémentation et la configuration d'un tel système (lui faire entrer toutes les plages de prix de son fournisseur), ne vaudrait probablement pas l'amélioration apportée et emporterait le travail fourni lié à la simplification de l'encodage de la maison.

6.5 Paramètres - *Settings*

La page des paramètres permet de définir plusieurs préférences de l'utilisateur et de modifier certaines parties de la base de données. Pour rappel, l'utilisateur a 2 options par rapport à son profil : s'il est avancé ou basique et s'il désire activer le défilement automatique. En effet, sur les pages qui affichent un plan, par défaut

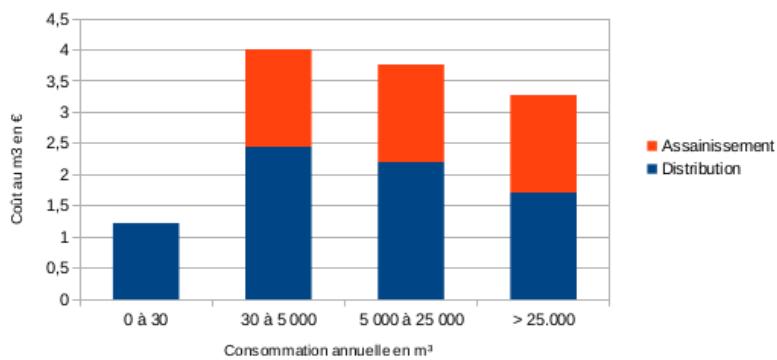


FIGURE 6.17 – Prix de l'eau au kWh en fonction de la consommation annuelle. Source des données :Société Wallone Des Eaux [50]

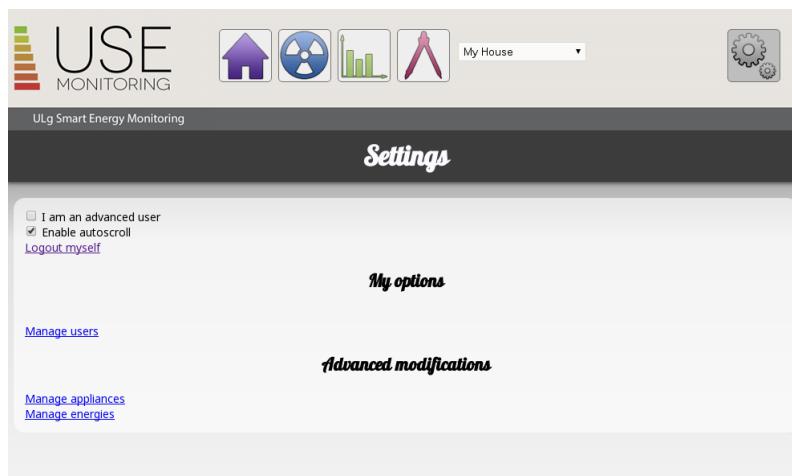


FIGURE 6.18 – Page des paramètres

si l'écran est petit et que le plan n'est pas visible en entier, le navigateur va défiler automatiquement jusqu'à ne plus voir le menu et le ruban de titre.

L'utilisateur peut ensuite configurer les utilisateurs (Voir fig.6.19). Les utilisateurs ont 3 niveau d'accès :

- Vue uniquement : Il a accès à la page de consommation et de statistiques uniquement
- Vue et alertes : Il peut en plus voir et changer les alertes
- Tous les accès : Il peut aussi accéder à la modélisation

De la même façon que les utilisateurs, il peut modifier la liste d'énergies et la liste d'appareils électro-ménagers. Cette fonctionnalité sera plus facilement compréhensible après la révision de la structure interne qui est expliquée par l'exemple dans la section 6.7.1.

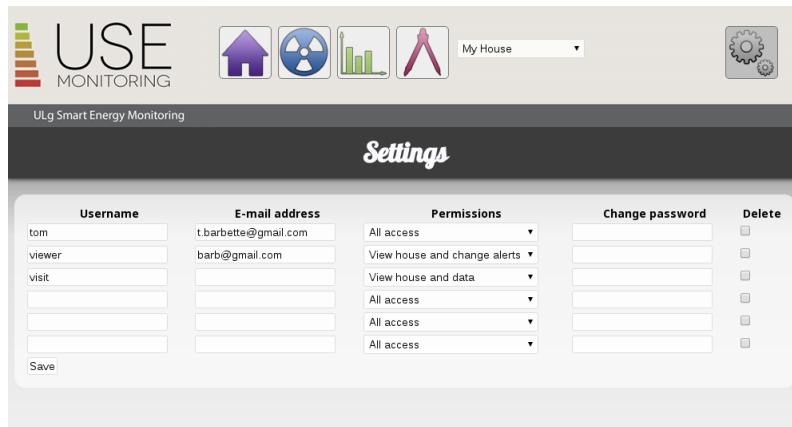


FIGURE 6.19 – Édition des utilisateurs



FIGURE 6.20 – Compatibilité de l’élément canvas avec les navigateurs. Source : caniuse.com [7]

6.6 Compatibilité et smartphones

La compatibilité avec les navigateurs de l’élément canvas est visible sur la figure 6.20. La compatibilité est plutôt bonne, sauf pour Internet Explorer. La question est de voir si c’est réellement un problème, en fonction de la part de marché des versions incompatibles : celles avant la version 9. La figure 6.21 nous montre la part de marché des différents navigateurs. Malheureusement la version 8 est encore beaucoup utilisée (23%) alors que la 7 a presque disparu (à peine 1,2%). La part de marché d’internet explorer 6 est toujours de 6%, cependant ce taux élevé vient en grande partie de la Chine[24] et des entreprises qui ne mettent pas à jour leurs ordinateurs. En belgique, tout comme aux USA, la part de marché est de 0,2%[24]. En résumé, il ne semble pas utile d’assurer la compatibilité pour les versions 6 et 7.

Concernant l’élément canvas, la librairie ExCanvas[20] permet d’assurer la compatibilité de l’élément canvas avec les navigateurs 6 à 8.

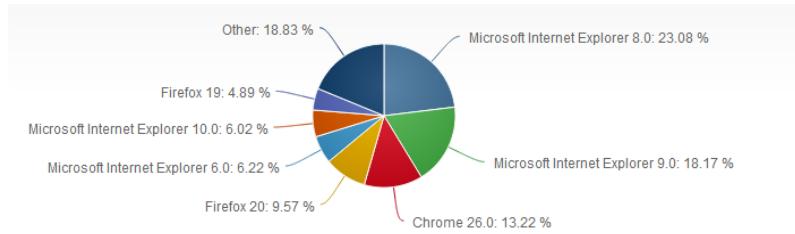


FIGURE 6.21 – Part de marché des navigateurs. Source : *Net Market Share* [34]

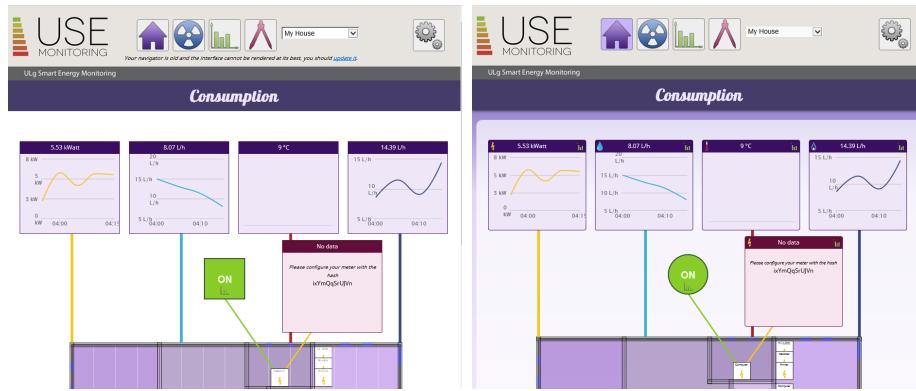


FIGURE 6.22 – Comparaison de l'affichage entre Internet Explorer 8 et 10

Enfin, un plugin de compatibilité dénommé TouchPunch [27] amène la compatibilité de jQuery ui et son système de drag'n drop sur les smartphones.

La compatibilité est donc assurée sur la plupart des systèmes récents, avec un affichage presque unique. En effet, le CSS3 ne passe pas toujours, principalement sur Internet Explorer mais des solutions de contournement sont mises en place. Il faut comparer les désavantages à obliger l'utilisateur à upgrader son navigateur à celui d'installer un programme inconnu et/ou dépendant d'une plateforme. Un utilisateur installera plus facilement une mise à jour de son navigateur.

En pratique, sauf pour quelques éléments de design, toutes les couches de compatibilité permettent le fonctionnement avec internet explorer 6 et 7. S'il était réellement important d'assurer leur compatibilité, il suffirait d'ajouter des feuilles de style CSS spécifiques à ces versions et d'ajouter quelques vérifications au sein du code JavaScript.

Une attention toute particulière a été donnée au système de redimensionnement. A vrai dire, presque toutes les pages s'adaptent à l'écran et sa taille, ne modifiant pas uniquement les tailles et les dimensions, mais également la position des objets. La page consommation par exemple dispose même d'un affichage en deux passes. Une première passe affiche les objets mais de façon invisible (ou plutôt sans le contenu des "boîtes" HTML div). J'utilise le résultat pour disposer mieux les éléments en fonction de l'écran et refait une deuxième passe pour réellement les afficher. Ceci implique une certaine utilisation du processeur mais procure un résultat très satisfaisant au niveau de l'agencement (la performance sera discutée dans la section 6.8). La taille

des nombreuses données du plan à télécharger est compensée par la grande utilisation de l’AJAX. Comme expliqué dans la section 4.2.1, dans la plupart des dessins de plan, Django n’écrit directement dans le code HTML de la page que les étages. C’est ensuite une requête AJAX qui va récupérer la liste des murs, en parallèle avec la liste des appareils. Une fois les murs chargés, les fenêtres le sont également. De même, les données des graphiques ne sont jamais écrites dans le code HTML de la page, permettant à l’utilisateur d’obtenir un chargement progressif mais sans attente.

Concernant les navigateurs mobiles, je n’ai pas testé l’iPhone et l’iPad, ne disposant pas de l’appareil et l’émulateur n’étant pas accessible sur PC. Cependant, leurs navigateurs utilisent le moteur Webkit, tout comme Google Chrome et la compatibilité devrait être atteignable sans trop de soucis si des problèmes se manifestaient. Concernant les navigateurs mobiles sous Android, j’ai du faire face à plusieurs problèmes au sein des navigateurs et du système d’exploitation lui-mêm :

- Comme discuté précédemment, j’ai souvent utilisé les éléments HTML canvas. J’ai rencontré un problème étonnant sous Google Chrome Mobile avec un mobile Samsung Galaxy S2. Tous les petits éléments canvas s’affichaient correctement, mais pas les grands. En réalité, les éléments canvas d’une taille inférieure à 256*256 sont dessinés par le CPU, tandis que ceux d’une taille supérieure sont dessinés par le GPU. En désactivant totalement le rendu GPU, la page s’affiche correctement. Le problème est reporté et fait l’objet d’un bug report en pleine activité [9]. Le problème en cause est clairement le rendu GPU et non le code. Le rendu est correct sans désactivation du GPU sur l’émulateur Android et sur un HTC Desire X. Probablement parce que ceux-ci n’ont pas d’accélération matérielle du navigateur.
- Le navigateur Android d’origine souffre lui d’un autre bug [8], de sorte qu’un canvas ne peut pas être nettoyé avant d’être redessiné, laissant de nombreux artefacts sur la page. Pour contrer ce bug apparu avec la version 4 d’Android et qui n’est toujours pas fixé à ce jour, j’ai dû peindre une zone blanche sur l’entièreté du canvas avant chaque rafraîchissement du plan. Ce qui a pour effet d’abîmer un peu le design.

La version mobile de Firefox, elle, ne présente aucun défaut d’affichage. Bref, il semble que Google ait encore du travail sur son système et qu’il y ait eu un peu trop de précipitation dans la publication de ses mises à jours.

6.7 Modularité

Les sections précédentes ont porté leur attention sur la modularité de certains éléments tel que le plan qui pourrait avoir aisément un autre module de rendu. La flexibilité était un des principaux aspects du cahier des charges. Une maison peut

Title	Short name	Unit	Unit instant	Color	Overall	Type	Lhv	Price
Electricity	electric	kWh	kWatt	#1c81d4	<input type="checkbox"/>	Power	1.0	0.128
Water	water	L	l/h	#3591d2	<input type="checkbox"/>	Consumable	0.0	0.0622
Gas	gas	L	l/h	#734973	<input type="checkbox"/>	Power	0.09	0.728
Fuel	fuel	L	l/h	#591a22	<input type="checkbox"/>	Power	0.1	0.979
Temperature	temp	°C	°C	#ad1f24	<input type="checkbox"/>	State	0.0	0.0
Switch	switch			#8acc27	<input checked="" type="checkbox"/>	State	0.0	0.0
Wood	wood	kg	kg/h	#401300	<input type="checkbox"/>	Power	4.0	0.0928
					<input type="checkbox"/>	Power		
					<input type="checkbox"/>	Power		
					<input type="checkbox"/>	Power		

Save

FIGURE 6.23 – Ajout d'une nouvelle énergie : le bois

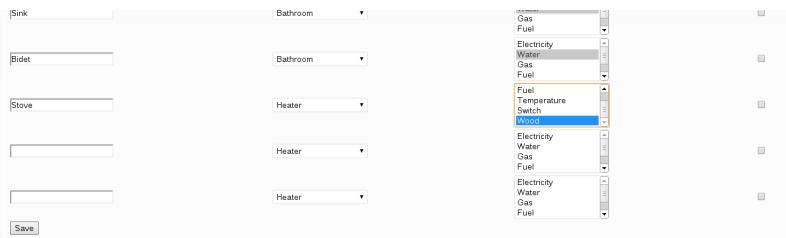


FIGURE 6.24 – Ajout d'un nouvel appareil : le poêle à bois

être très différente d'une autre, certains utilisateurs peuvent avoir différents appareils ménagers (cela a également déjà été traité dans la section 6.5) ou peuvent utiliser différentes énergies. Cette modularité techniquement simple avec un schéma de base de données et une programmation respectant le paradigme MVC, requiert pas mal d'attention à plusieurs niveaux de l'interface. Cette section met en valeur certains aspects et problèmes auxquels il a fallu prêter une attention spéciale.

6.7.1 Ajout d'une énergie et d'un nouvel appareil de consommation

L'utilisateur peut s'il le désire, ajouter de énergies. Comme nous l'avons vu le type d'énergie (avec ses unités, sa couleur, sa conversion PCI ou encore le prix par unité) est un simple modèle Django. La section paramètre permet d'ajouter une énergie (voir fig. 6.23). Nous allons suivre les détails nécessaire à cette flexibilité en prenant l'exemple du bois. Le poêle à bois n'était pas prévu dans le projet initial, notamment parce qu'il est très difficile de compter l'énergie consommée (une solution sera développée à la fin de la section). Une petite recherche nous apprend que le bois relativement sec à un pouvoir calorifique inférieur de 4 kWh par kg. Il se vend environ 65€ le stère, et un stère de chêne fait environ 700kgs, soit 9,28 cents par kg.

L'habitation que j'occupe est chauffée par un poêle à bois. Ceci m'a permis de créer un nouvel appareil (Fig 6.24) que j'ai ensuite ajouté dans ma maison. L'interface de l'étape 7 s'adapte automatiquement pour afficher le nouvel appareil. Comme le

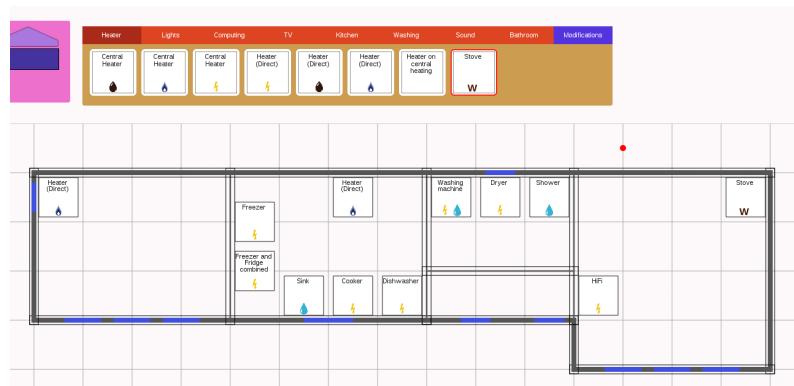


FIGURE 6.25 – Interface de configuration des appareils, avec le nouvel objet poêle à bois"

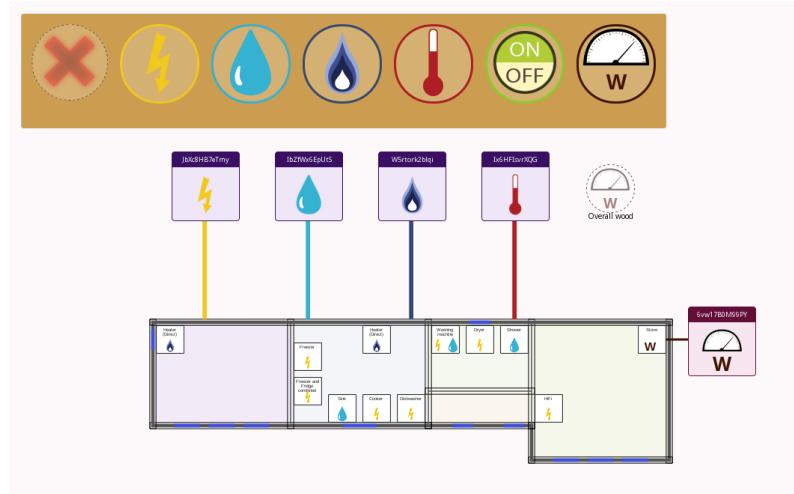


FIGURE 6.26 – Interface de configuration des compteurs avec la nouvelle énergie : le bois

bois est une énergie ajoutée par l'utilisateur, il n'y a pas d'image. Lorsque l'interface détecte qu'il n'y a pas d'image pour le bois, il écrit la première lettre de l'énergie dans sa couleur. Un symbole de compteur par défaut est également prévu pour cette situation. Dans l'étape 7 de configuration des compteurs (Voir fig. 6.26) j'ai ajouté un compteur de bois sur le poêle en mode "différentiel". Cela signifie que lorsque j'enverrai une donnée, elle s'ajoutera à la consommation totale.

Concernant le problème de quantité de données pour le bois j'ai créé un raccourci "petite buche", "moyenne buche" et "grosse buche", envoyant chacun une requête d'ajout de respectivement 0.8, 1.2 et 1.6 sur le compteur que j'ai créé. L'unité étant définie comme le kg, cela correspond aux valeurs moyennes des petites moyennes et grosses bûches. Il suffit que je clique sur le favoris de mon navigateur correspondant à la taille de la bûche, lorsque je fais mon feu.

Le résultat des statistiques après une soirée de chauffage au bois est visible sur la



FIGURE 6.27 – Statistiques du compteur bois

figure 6.27. Nous observons que le graphique des consommations instantanées affiche des pics très hauts avec des valeurs improbables. C'est normal, puisque j'ai ajouté 3 bûches d'affilées dans le poêle. L'interface considère donc que j'ai consommé plusieurs kgs de bois par seconde, donc plusieurs milliers par heure. En réalité, l'erreur vient du fait que je ne mesure pas la consommation de bois, mais bien son approvisionnement. Ce qui devrait être envoyé dans le mode différentiel est la quantité de bois consommée toutes les 5 minutes par exemple. La consommation journalière elle, est correcte puisqu'elle est l'addition des quantités consommées sur la journée.

6.8 Performances

6.8.1 Base de donnée

J'ai utilisé le plugin django “django-toolbar” [14], permettant de voir les requêtes exécutées et le temps processeur qu'elles ont nécessité. Cet outil magnifique combiné au manuel d'optimisation des requêtes Django, permet de tirer parti du système de cache des requêtes pour minimiser celles-ci et m'a permis de diviser par 3 mon nombre de requêtes. Django utilisant un système pour construire automatiquement les requêtes, il est plus que nécessaire d'en connaître son fonctionnement. Par exemple, récupérer la liste des maisons se fait avec `liste = House.objects.all()`. Si on veut vérifier qu'il n'y a aucune maison avant de parcourir la liste des maisons, alors il vaut mieux utiliser `len(liste)` que `liste.count()`, car dans le premier cas Django exécutera la requête complète et la mettra en cache, tandis que dans le deuxième cas, il exécutera une requête du type `SELECT COUNT(*)`. Et lorsque le parcours de la liste des maisons commencera, il devra alors refaire une requête du type `SELECT *`. Cela semble anodin, mais apporte un gain de 0,30ms sur le chargement de la

page d'accueil par exemple. Dans une boucle ou répété, ce genre de détail peut faire toute la différence. Les divers temps de requêtes sont pris sur mon serveur de développement local (ordinateur portable), le serveur dédié est généralement plus rapide.

Accueil

La page d'accueil ne fait évidemment presque pas de requêtes si ce n'est l'affichage des 10 dernières entrées de la table log. Cette requête s'exécute en 0.70ms. Les autres requêtes sont au nombre de 3 et sont faites sur toutes les pages : une pour l'affichage de la liste des maisons dans le menu déroulant, une pour l'identification de l'utilisateur et enfin une pour la récupération de la session dans la base de données. Le temps cumulé d'exécution des 4 requêtes est d'environ 2,3ms.

Consommation

La aussi l'analyse de mon nombre de requêtes m'a permis de passer de 88 requêtes sur la page des consommations exécutées en 120ms à 19 faites en 8.2ms. L'algorithme qui prend un nombre important de requêtes est ici celui de détection de la taille réelle de la maison. En effet, il analyse la position de toutes les extrémités des murs pour trouver les points extrêmes de chaque étage, et finalement l'espace maximal pris par la maison afin que le plan puisse commencer le rendu. Ceci, en coupant le plan "à ras bord" avant d'avoir chargé les données des murs (pour rappel, les données sont chargées en AJAX). L'algorithme de sérialisation est en réalité le problème : pour sérialiser la liste des compteurs avec certaines données de clés étrangères il prend 11 requêtes, alors qu'il pourrait évidemment le faire en une. Ceci est en fait dû à ce que Django appelle les liens "ManyToMany", soit les relations où plusieurs objets d'un même type peuvent être liés à plusieurs autres objets d'un second type, ce qui en pratique nécessite de créer une 3ème table liant un objet de la classe 1 à la classe 2. Django ne permet pas de faire une requête qui ira rechercher les données de la 1ère et de la deuxième classe en même temps. Ceci dit, il n'est pas possible d'assurer avec certitude que faire ces requêtes en une serait forcément plus rapide, la taille du joint unique pouvant être très conséquente.

Voici la liste du temps d'exécution des requêtes de récupération des données du plan, exécutées pour répondre aux appels AJAX :

Liste des murs : 1 requête, 0.80ms

Liste des fenêtres : 1 requête, 0.76ms

Liste des appareils : 25 requêtes, 8ms

Le nombre de requêtes pour les appareils dépend du nombre d'objets de la maison à récupérer car il y a une relation "ManyToMany".

La récupération des données des compteurs pour les 24 dernières heures prend 2 requêtes (une pour les données et une pour retrouver l'ID du Meter en fonction de

son hash) et s'exécute en 2.5ms.

Autres données importantes

La requête pour récupérer tous les relevés bruts d'un compteur (Reading) depuis son installation prend 30ms, et les données agrégées par jour (ReadingDay) prennent 2ms. Cependant, actuellement la table des relevés bruts ne contient que 10000 entrées. Pour l'envoi de données d'un compteur dont la consommation fluctue (et donc dont le preprocessing ne supprime pas de relevés) et dont les relevés sont pris toutes les 5 minutes cela ne représente que 34 jours de relevés. On est en droit de se demander à partir de quand il faut supprimer les relevés pour éviter que la performance de cette requête ne se dégrade. En réalité, j'attends d'avoir les données des vrais capteurs pour décider de la suppression, mais je pense que la conservation des relevés bruts plus de deux semaines n'est pas utile à l'utilisateur. Plus loin, ce qui l'intéressera sera sa consommation agrégée par jour, voire uniquement par mois. Une solution serait éventuellement d'agréger les anciens résultats par heure, afin de garder un profil précis si l'utilisateur le désire, sans toutefois surcharger la base de données.

Les autres pages suivent les mêmes proportions et ne seront pas discutées en détail. Une page avec un plan prendra une vingtaine de requêtes exécutées en une petite dizaine de millisecondes, les autres pages un peu moins.

6.8.2 Tailles de transferts et temps de rendu

Pour les essais de performances, j'ai d'abord analysé la taille des fichiers téléchargés pour les principales pages. J'ai fait trois analyses : le nombre total d'octets transférés sans l'utilisation de cache, le nombre total d'octets transférés lors de la première visite et lors de la deuxième visite (et les suivantes). Le résultat est visible sur la figure 6.28. Les navigateurs possèdent tous des caches, et la majorité du contenu ici échangé avec l'utilisateur est statique : les fichiers JavaScript, le CSS, et les nombreuses images. Dès lors, l'interface tire parti au maximum du cache de ces navigateurs, pour réduire le temps de récupération de ces fichiers et donc le temps de rendu (nous le verrons plus tard). Le nombre d'octets téléchargés lors de la première visite d'une page n'est pas non plus égal au nombre total d'octets transférés sans aucun cache. En effet, d'une part, la librairie jQuery est incluse depuis les serveurs de google, or nous l'avons vu, le taux de pénétration de jQuery atteint 30% des sites webs les plus populaires, et beaucoup de sites utilisent la même technique assurant que jQuery soit déjà dans le cache de l'utilisateur. De plus, toutes les librairies JavaScript ne sont pas incluses sur toutes les pages. Le système de dessin du plan par exemple est uniquement inclus sur les pages consommation, alertes et les 4 dernières étapes de modélisation. Ainsi, lorsque l'utilisateur arrive sur la page d'accueil, il ne charge en fait que les librairies communes à toutes les pages. L'utilisateur ne téléchargera jamais plus de 300Ko par page. Comme il passe toujours d'abord par l'accueil, il n'aura jamais la situation "sans cache" de la page consommation qui demande plus de 680Ko.

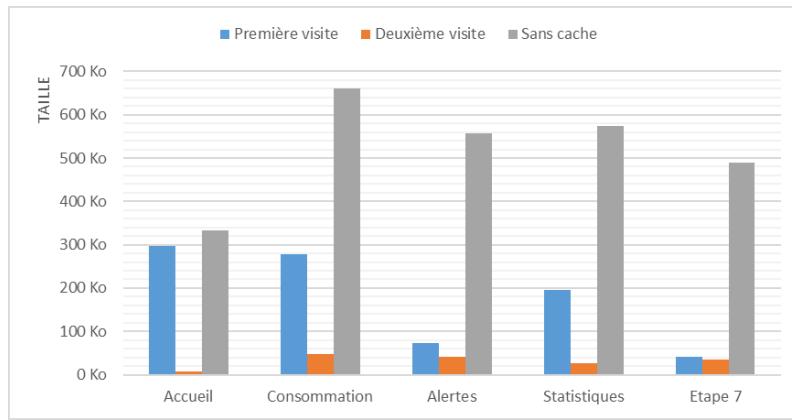


FIGURE 6.28 – Quantité de données transférées pour l'affichage de diverses pages

Une fois toutes les pages visitées, aucun chargement n'engendre plus de 40Ko de transfert, ce qui est très raisonnable. A titre de comparaison, la page d'accueil de la RTBF [47] engendre un chargement de plus de 700Ko au premier chargement, et 55Ko au suivant, soit plus du double de notre situation.

Ce qui compte au final pour l'utilisateur, ce n'est cependant pas la taille transférée, s'agissant de si petites quantités, mais le temps de rendu. J'ai donc comparé le temps de rendu de chacun des 3 grands navigateurs de bureau pour quelques pages caractéristiques également. En réalité, les navigateurs ne donnent que le moment où ils ont fini de télécharger la dernière ressource. Ce moment n'est pas tellement différent du moment où le rendu est terminé dans notre cas, puisque toutes les pages lourdes effectuent des requêtes en AJAX à la fin du rendu. En effet, la page de consommation ne charge les données des compteurs qu'à la fin de l'affichage du plan, et concernant la page statistique, les graphiques s'affichent au fur et à mesure du traitement des données, une fois celles-ci reçues. L'appel des données de la page statistique ne se fait également qu'au moment où toute la page et les librairies ont été chargées, soit quand le rendu est déjà presque fini. Le résultat est visible sur la figure 6.29. J'ai également fait les mêmes mesures sur la page d'accueil de Google à titre de comparaison. Les batonnets intitulés "SC" sont les temps de chargement sans cache. Fatalement, les temps de chargement sans cache sont plus longs puisqu'il y a plus de données à transférer. Il est important de noter que le temps indiqué est celui de la fin du rendu du dernier élément. Mais le rendu est toujours progressif. Pour les plans par exemple, j'ai choisi d'afficher le plan à chaque nouvel élément téléchargé. Il est donc dessiné, à la réception des murs, des fenêtres, des appareils, puis des compteurs. Ceci allonge le temps de rendu total mais permet à l'utilisateur de voir le chargement progresser. Toutes les mesures ont été faites avec la fonction de défilement automatique activée. Firefox est fort impacté par cette option et les autres navigateurs ne semblent pas en souffrir.

Concernant les mobiles, j'ai utilisé Chrome dans sa version Mobile. La méthodologie est la même. J'ai comparé les temps de rendu entre un Samsung Galaxy S2,

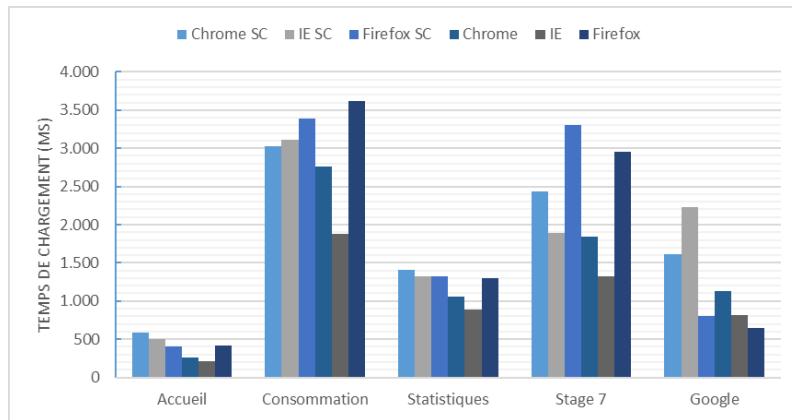


FIGURE 6.29 – Temps de rendu pour diverses pages - Moyennes de 5 rendus - Chrome 27, IE 10, Firefox 13

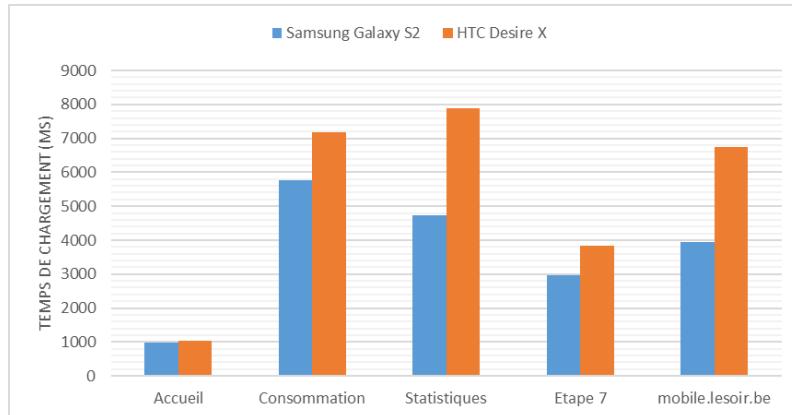


FIGURE 6.30 – Temps de rendu pour diverses pages - Chrome Mobile

Smartphone haut de gamme mais datant de 2 ans et demie (la version actuelle est le S4), et un HTC Desire X, smartphone récent mais de bas de gamme (bien que l'un des meilleurs de sa gamme). En plus des pages du site, j'ai comparé cette fois le temps de chargement des pages avec le temps de chargement de la version mobile du site du journal le Soir. Les résultats sont visibles sur la figure 6.30. On voit ici que les Smartphones sont plus impactés par la lourdeur du traitement JavaScript, mais restent dans une moyenne acceptable. De plus, l'affichage partiel prend ici mieux son sens. Si la page consommation prend 7 secondes à être rendue, en réalité après 3,5 secondes le plan est déjà affiché, et la page attend simplement le chargement des données du graphique. C'est important, car laisser patienter l'utilisateur 7 secondes sur un écran de chargement serait par contre frustrant pour lui. Remarquons que le HTC Desire X est légèrement plus lent que le Samsung Galaxy S2, mais ce smartphone, quoique bas de gamme, ne s'en sort pas trop mal.

6.8.3 Évaluation quantitative

Cette section présente quelques chiffres concernant le code source.

Code Python : 2287 lignes, 6486 mots.

Code JavaScript (sans les librairies externes) : 3060 lignes, 9969mots.

Librairies JavaScript externes : 14 Modules Django externes importées : 6 Templates

HTML : 2266 lignes, 5766 mots.

CSS : 1051 lignes, 1597 mots.

Scripts Pythons (compteurs virtuels) : 340 lignes, 1179 mots.

Nombre d'images : 64.

Le nombre assez restreint de lignes de code démontre l'avantage des choix d'implémentation discutés dans la section 3. Les nombreux modules permettent de se concentrer sur le cœur des algorithmes, Python étant un langage particulièrement concis et jQuery permettant des enchainements de fonctions en une seule ligne sans pour autant perdre en lisibilité. Cependant ce résultat est évidemment à relativiser car l'écriture de chaque ligne est plus longue, puisqu'il y a plus d'implications et de fonctionnalités. De plus, l'utilisation d'une librairie ou de l'un des nombreux imports Python requiert une étape de recherche importante, suivie de la lecture approfondie de la documentation avant l'implémentation. Django est présenté ici comme un framework permettant de tout faire très rapidement. C'est exact, mais comprendre son fonctionnement, et surtout comprendre le fonctionnement de son système de modèle, de requêtes, l'utilisation des templates, etc...a nécessité en réalité presque un mois.

Chapitre 7

Perspectives et travail futur

Tout au long de l’élaboration du projet, l’équipe IdP n’a cessé d’émettre de nouvelles idées quant à l’interface, mais le temps est limité et nous avons dû faire des choix. Cependant, chaque groupe a veillé à garder une certaine modularité dans la conception des éléments qui leurs étaient attribués. Le projet en son état actuel, bien que totalement fonctionnel, est encore une infime partie de ce qu’il peut devenir. Je vais explorer dans ce chapitre diverses possibilités pour améliorer l’interface et les données qui lui parviennent.

7.1 Alertes

Un premier travail intéressant serait de permettre à l’interface de combiner plusieurs pièces de puzzle, afin de créer des conditions sur le dépassement simultané de deux capteurs par exemple. Rappelons que la combinaison de plusieurs pièces est déjà possible au niveau de la base de données, mais c’est le manque de temps pour implémenter les nombreuses routines de dessin des pièces combinables qui m’ont empêché de l’implémenter du côté JavaScript/HTML.

De nombreuses pièces finales intéressantes pourraient également être imaginées : appel d’une requête HTTP, notifications sur une application smartphone, lancement d’un script bash ou même Python, ...

7.2 Statistiques et suppression des relevés

La page statistiques fera l’objet d’une évaluation avec le groupe IdP en septembre : après deux mois d’envoi de données avec de vrai capteurs il sera intéressant de savoir quelles seront les données utiles ainsi que la pertinence des graphiques.

Actuellement, aucune donnée des relevés n’est supprimée si ce n’est les doublons

discutés dans la section 4.6.1. En réalité, conserver plus de deux semaines la consommation précise d'une journée n'est probablement pas nécessaire. L'utilisateur sera intéressé par le détail de sa consommation dans un passé proche, mais les données agrégées par jour sont suffisantes sur le long terme, car ce qui importe pour l'utilisateur, c'est de pouvoir comparer la consommation par mois et par an, afin de voir si ses efforts en matière de consommation ont porté leurs fruits.

7.3 Image des appareils

Je prévoyais l'ajout d'une image pour chaque appareil à placer dans la maison. Cependant, cela représentait un travail assez conséquent, étant donné qu'il y a plus de 35 appareils actuellement. Je me suis débrouillé pour créer déjà une cinquantaine de symboles pour l'interface, et je n'ai pas eu le temps de dessiner chacun des appareils pour remplacer la vue actuelle constituée d'un simple carré portant le nom de l'appareil et le symbole des énergies qu'il utilise.

7.4 Application SmartPhone native

Le développement d'une application pour Android permettrait d'assurer une plus grande fluidité, et de contourner les problèmes de compatibilité rencontrés. Pour le développement de leurs applications smartphone, beaucoup d'entreprises ont choisi pour solution de développer une application qui est en réalité un simple navigateur, avec toutefois quelques ajouts comme un menu ou un système de notification, mais dont tous les éléments sont simplement rendus en HTML par un serveur et rendus par une librairie disponible sur le smartphone. L'utilisation de cette technique dans notre cas permettrait d'avoir un menu en code natif, qui ne se recharge pas au changement de pages, et d'ajouter une nouvelle pièce de puzzle qui créerait une notification sur le smartphone.

Au vu de la section performances, l'interface ne souffre pas spécialement du manque d'application dédiée, même si cela représente une piste intéressante.

7.5 Modèle énergétique

Le projet tel que défini en début d'année et présenté au groupe Ingénieur de Projet prévoyait l'intégration d'un modèle énergétique, pour placer l'utilisateur entre deux scénarios simulés : l'un au niveau de la consommation optimale, un but à atteindre pour l'utilisateur, et l'autre au niveau de consommation minimale. Il devait également permettre une désagrégation des mesures globales en secteurs de consommation, et la simulation de la consommation en améliorant certains paramètres de la maison (comme l'isolation d'un mur). Cependant, le modèle devait être conçu par des acteurs en dehors de l'équipe IdP et n'a pas pu m'être fournis à temps. Il est donc présenté

ici et pourrait faire l'objet d'une intégration future à l'interface, notamment par rapport au concours Ingénieur de Projet qui se termine en Novembre.

Cet abandon du modèle explique pourquoi l'interface possède toute une partie de modélisation qui n'a actuellement d'autre vocation que l'esthétique de l'affichage des résultats sur un plan. En réalité le modèle a besoin d'un utilisateur qui lui fournisse des données relatives aux types d'appareils présents dans la maison, type de chaudière, mais aussi des informations sur la disposition, l'épaisseur des murs, des fenêtres, etc. L'utilisateur doit pouvoir choisir avec quelle précision il fournit ces données. Plus le modèle reçoit de données, plus ses estimations seront précises. C'est pour cela que l'interface présente un mode basique et avancé. L'utilisateur basique est évidemment celui qui a le moins de données à rentrer et qui surtout,nécessite le moins de connaissances techniques, par exemple celles concernant l'isolation des murs de sa maison.

Le but principal du modèle était de définir , avec les informations acquises par les composants présentés précédemment, une série de statistiques :

- Les pertes, et le lieu des pertes (isolation, etc)
- Une courbe de la consommation dans le meilleur des cas
- Une courbe de la consommation dans le pire des cas
- Une désagrégation de la consommation globale en divers postes de consommation : chauffage, éclairage, etc.

Le modèle devait sortir 2 courbes de consommation : une courbe représentant la consommation en cas d'utilisation rationnelle de l'énergie, l'autre correspondant à un comportement abusif. En comparant la consommation provenant des mesures, l'utilisateur peut visualiser sa position et, en essayant d'atteindre la courbe inférieure, trouver la motivation nécessaire pour réduire sa consommation, et donc sa facture.

Idéalement, le modèle serait en mesure de modifier ses prévisions en fonction de travaux d'amélioration à effectuer dans la maison : ceci permettrait d'estimer les économies potentielles, et de guider l'utilisateur dans le choix des rénovations les plus efficaces à effectuer sur le plan énergétique.

La liste des variables nécessaires au paramétrage du modèle tel que défini avec l'encadrant est disponible en annexe 1. Techniquement, mon interface devait donc fournir des valeurs pour toutes ces variables. L'interface aurait alors appelé une fonction matlab en passant ces données sous forme de matrice. En pratique, la partie basique devait être faite après la partie avancée,et, comme dans la plupart des cas, il aurait suffi de supprimer des paramètres à rentrer par l'utilisateur. Vu que le modèle n'a jamais été disponible, la version basique de l'input n'a pas été spécialement implémentée. Actuellement, l'interface permet donc la définition des données "avancé" définies dans l'annexe, hormis l'orientation du terrain (Nord/Sud/...), et les informations associées aux objets comme le classement des électroménagers de l'utilisateur (A, A+, ...) ou la date de la chaudière. En effet, voyant que toute

cette partie serait probablement inutile puisque le modèle ne viendrait probablement jamais, j'ai préféré me concentrer sur les compteurs virtuels et la partie "alertes et statistiques" de l'interface. Cependant l'ajout d'informations liées aux objets du plan est prévu dans l'implémentation et facilement réalisable, si ces informations devaient être utilisées.

Même sans le modèle, le projet reste intéressant et atteint son but d'information. L'interface permet notamment le contrôle d'appareil, ce qui est une fonctionnalité supplémentaire par rapport au cahier des charges.

7.6 Interfaçage avec un système domotique existant

Nous l'avons vu dans l'étude de marché, la spécificité de notre système est son adaptation non-intrusive aux divers compteurs déjà en place dans la maison. Cependant, certaines maisons sont déjà équipées en matière de domotique. Ces systèmes sont extrêmement chers et je n'ai pu m'en procurer. Cependant les différents modes de compteurs et la simplicité du protocole permettent d'intégrer facilement diverses couches de compatibilité avec des systèmes existants. Par exemple, la smartbox d'électrabel [49] dispose d'une interface web. Il ne serait pas très compliqué de créer un script à la manière des compteurs virtuels (Voir chapitre 5) pour parser le code de la page HTML avec des expressions régulières et envoyer les données de consommation à l'interface.

De façon générale, le chapitre 5 a déjà présenté de nombreuses possibilités quant à l'extraction d'informations d'autres systèmes pour en tirer des données de consommation à envoyer aux compteurs de l'interface. Beaucoup d'approximations peuvent également être faites via la vision par ordinateur, puisque la plupart des appareils disposent de LED pour indiquer leur état et que leur consommation est souvent fixe en fonction de celui-ci.

7.7 Présence des habitants

Pour le modèle énergétique, il est important de connaître le nombre d'habitants au sein de la maison. Avant son abandon, j'ai exploré plusieurs possibilités pour affiner le taux de présence des utilisateurs, en étudiant notamment sa position et donc sa présence dans la maison.

Une première idée serait d'inter-connecter le système avec un système d'alarme. Lorsque l'alarme est branchée, l'utilisateur n'est pas chez lui (sans oublier les systèmes d'alarmes de nuit). On peut également utiliser les différents détecteurs pour avoir un fonctionnement par pièce, on aurait alors la possibilité de détecter un appareil.

Une autre possibilité est d'installer un logiciel sur le smartphone de l'utilisateur. La plupart de ceux-ci disposant d'une puce GPS, il est possible de savoir s'il est à la maison ou ailleurs. Le WiFi pourrait également être utilisé, l'utilisateur pouvant définir un réseau WiFi comme celui de son travail ou de sa maison, que ce soit via son smartphone ou son ordinateur portable.

La section précédente envisageait un interfaçage avec des installations domotiques. Certaines nouveautés sont apparues en matière de serrures électroniques comme Lockitron[30]. Lockitron est une serrure activable par WiFi ou Bluetooth. Elle possède une API afin d'être contrôlée via n'importe quel système connecté à internet. L'utilisation de Lockitron permettrait assez facilement de savoir si l'utilisateur est chez lui ou pas.



FIGURE 7.1 – Lockitron

Chapitre 8

Conclusion

L'objectif de ce travail était d'implémenter une interface de présentation des statistiques de consommation d'une maison. L'originalité réside dans sa capacité à configurer l'enveloppe énergétique de la maison, sa grande flexibilité quant aux types de compteurs, et le support de multiples entrées. Parallèlement, l'interface devait respecter trois axiomes :

- Accessibilité : une interface facile d'utilisation et intuitive mais néanmoins complète
- Flexibilité : au niveau de la maison, des objets, des capteurs, ou encore des énergies
- Performance : l'interface doit être fluide, et utilisable sur les smartphones

L'interface a été pensée entièrement pour une facilité d'accès pour tous les utilisateurs. Le système de plan est simple mais permet la définition correcte de l'enveloppe de la maison. Le placement des appareils est aussi souple et l'utilisation du glisser-déplacé pour les compteurs permet une approche intuitive. Les nombreux symboles procurent une vision claire pour l'utilisateur de bureau, et permettent d'utiliser le doigt pour sélectionner un élément sur un smartphone.

L'interface a également été démontrée comme assez modulable : l'utilisateur peut ajouter de nouvelles énergies, de nouveaux appareils et de nouveaux compteurs. Les nouveaux compteurs peuvent même être envisagés pour mesurer une production. Les différents modes d'envoi de données permettent d'accepter plusieurs méthodes de consommation : autant des données envoyées par des compteurs de consommation totale que des compteurs de consommation instantanée. L'interface peut également être utilisée comme système de contrôle des appareils de l'utilisateur.

La structure interne de l'interface a également été pensée modulable, autant du côté serveur grâce à Django et ses préceptes bien pensés utilisant le paradigme MVC, mais aussi du côté client où le code JavaScript est en général divisé en modules remplaçables et modifiables. J'ai aussi veillé à coder en utilisant des objets dans les limites du JavaScript, ou du moins des mécanismes réutilisables en proscrivant le

copier-coller. L'ajout d'une nouvelle page bénéficiera par exemple, du système de récupération automatisé des Modèles établis côté serveur en objets JavaScript, et du renvoi de ceux-ci vers le serveur. Le système de dessin du plan est également basé sur un système de "plugin", permettant l'ajout facile de nouvelles étapes de définition du plan comme la définition de portes ou d'arbres autour de la maison pour calculer l'ensoleillement si le modèle énergétique le nécessitait.

Du côté des performances, les divers tests montrent que même les pages les plus lourdes n'ont qu'un impact léger sur le temps de rendu. La génération et l'affichage de la page d'accueil est plus rapide sur tous les navigateurs que celle de Google. L'algorithme de redimensionnement des plans ne souffrirait cependant pas de certaines optimisations, mais son calcul par étapes permet un affichage progressif de la page malgré des calculs de dimensionnement importants. L'interface fonctionne correctement sur les smartphones, moyennant quelques concessions en attendant la publication des correctifs du système Android.

Ce travail a également exploré les possibilités en matière de capteurs virtuels, notamment pour démontrer les avantages de l'utilisation de l'HTTP, protocole très mature, rarement bloqué par les firewall et dont le nombre de libraires et implémentations permet la mise en place très rapide de petits scripts pour envoyer ou lire des données depuis l'interface.

Pour conclure, il me semble que les objectifs ont été atteints, même si l'interface pourrait encore bénéficier d'améliorations. Par exemple, au niveau de la page des statistiques qui pourrait afficher plus de graphiques, de comparaisons par saison ou comparer la consommation d'un jour avec celle du même jour de l'année ou de la saison précédente, sans toutefois noyer l'utilisateur sous une trop grande quantité d'informations. La page des alertes pourrait également avoir de nouvelles "pièces" pour permettre plus d'actions et d'automatisations en fonction des événements se produisant autour de la consommation de l'utilisateur.

Au regard de l'état de l'art de ce qui existe déjà, l'interface de USE Monitoring semble être la seule à combiner tous les éléments cités au fur et à mesure de ce travail. Certaines interfaces sont très belles mais ne permettent que la mesure de consommations électriques. D'autres permettent de mesurer toutes les énergies, mais seulement de façon globale et ne permettent pas de cibler des appareils en particulier. Aucune interface ne procure une souplesse aussi développée que dans ce travail. L'intégration future du modèle énergétique permettrait d'amener chez l'utilisateur des résultats scientifiques jusque là réservés uniquement aux professionnels, et lui permettant de trouver les pertes énergétiques au sein de sa maison.

Alors, est-ce que l'interface et le système complet avec les capteurs pourront amener l'utilisateur à réduire sa consommation ? La réponse est positive si on en croit les divers articles cités en introduction liant informations et réductions de consommation, ou si, simplement j'observe mon réflexe lorsque je peux constater, grâce aux

compteurs virtuels placés sur mes ordinateurs, ce qu'il m'en coûte concrètement. Si, par exemple, il est évident que mon ordinateur portable consomme moins que mon ordinateur de bureau, la vision de la consommation en euro force le passage à l'action. Je suis intimement convaincu que cet exemple est extrapolable à la consommation de chacun des appareils ménagers, ainsi qu'à la consommation générale de la maison.

Bibliographie

- [1] *45% of the popular websites use a javascript framework.* URL : <http://elie.im/blog/web/45-of-the-popular-websites-use-a-javascript-framework/#.UaWrPOddU9t>.
- [2] ACQUALYS. *Tableau comparatif PCI des énergies.* URL : <http://www.acqualys.fr/page/tableau-comparatif-pouvoir-calorique-inferieur-pci-des-energies>.
- [3] AXEIYA. *Comptage de l'énergie : kWh, kWh PCI, kWh PCS.* URL : <http://bes.axeiya.com/index.php?id=31>.
- [4] *Blender 2.6, Python Manual.* URL : <http://wiki.blender.org/index.php/Doc:2.6/Manual/Extensions/Python>.
- [5] Megan BRAY. *Review of Computer Energy Consumption and Potential Savings.* URL : http://dssw.co.uk/research/computer_energy_consumption.html.
- [6] P. BRIMBLECOMBE. *The Big Smoke (Routledge Revivals) : A History of Air Pollution in London Since Medieval Times.* Routledge revivals. Taylor & Francis Group, 2011. ISBN : 9780415671835.
- [7] caniuse.com. URL : <http://caniuse.com/>.
- [8] *Canvas clearRect failing to clear.* URL : <https://code.google.com/p/android/issues/detail?id=39247>.
- [9] *Canvas line drawing on Android failure.* URL : <https://code.google.com/p/chromium/issues/detail?id=231082>.
- [10] *CPU Burn.* URL : <http://manpages.ubuntu.com/manpages/hardy/man1/cpuburn.1.html>.
- [11] *Cron.* URL : <http://www.gnu.org/software/mcron/>.
- [12] IDD - Institut pour un DÉVELOPPEMENT DURABLE. *30 années d'évolutions des revenus et des prix.* URL : <http://users.skynet.be/idd/documents/indicateurs/indic05-1.pdf>.
- [13] *Django.* URL : <http://www.djangoproject.com/>.
- [14] *Django Debug Toolbar.* URL : <https://github.com/django-debug-toolbar/django-debug-toolbar>.
- [15] *Django Extension.* URL : <https://github.com/django-extensions/django-extensions>.

- [16] *Django Packages database*. URL : <https://www.djangoproject.com/>.
- [17] *Django-Pipeline plugin*. URL : <http://django-pipeline.readthedocs.org/en/latest/>.
- [18] *Efergy*. URL : <http://www.efergy.com/index.php/default/products-uk-1/energy-gateways-1/engage-hub-uk.html>.
- [19] *EKM Metering*. URL : <http://www.ekmmetering.com/>.
- [20] *ExCanvas*. URL : <https://code.google.com/p/explorercanvas/>.
- [21] K. Gram-Hanssen D. Maes M. Cantaert B. Spies J. Desmedt F. BARTIAUX G. Vekemans. *Socio-technical factors influencing Residential Energy Consumption SEREC*. 2006.
- [22] the green GRID. *Five Ways to Save Server Power*. URL : http://www.thegreengrid.org/~media/TechForumPresentations2008/Five_Ways_to_Save_Server_Power.pdf?lang=en.
- [23] INSTITUT DE CONSEIL ET D'ETUDES EN DEVELOPPEMENT DURABLE ASBL ICEDD. *Bilan énergétique wallon 2006, consommation du secteur domestique 2006*. Rapport statistique. Ministère de la Région Wallonne DGTR, 2008. URL : http://energie.wallonie.be/servlet/Repository/baa080930-domestique_2006_1423.pdf?ID=11526&saveFile=true.
- [24] *ie6countdown.com*. URL : <http://www.ie6countdown.com/>.
- [25] *Inkscape Wiki - Python modules for extensions*. URL : http://wiki.inkscape.org/wiki/index.php/Python_modules_for_extensions.
- [26] *Intel ARK*. URL : <http://ark.intel.com/>.
- [27] *jQUery UI Touch Punch*. URL : <http://touchpunch.furf.com/>.
- [28] *jquery.com*. URL : <http://www.jquery.com/>.
- [29] R KOHLENBERG, T PHILLIPS et W PROCTOR. « A behavioral analysis of peaking in residential electrical-energy consumers. » In : *J Appl Behav Anal* 9.1 (1976), p. 13–8.
- [30] *Lockitron*. URL : <https://lockitron.com>.
- [31] MCKINSEY&COMPANY. *Vers une efficacité énergétique de niveau mondial en Belgique*. 2009.
- [32] Le MONDE. *Dans la confusion, Copenhague s'achève sur un échec*. 2009. URL : http://www.lemonde.fr/le-rechauffement-climatique/article/2009/12/18/un-accord-non-contraignant-obtenu-in-extremis-a-copenhague_1282914_1270066.html.
- [33] Le MONDE. *Le Canada quitte le protocole de Kyoto*. 2011. URL : http://www.lemonde.fr/planete/article/2011/12/13/le-canada-quitte-le-protocole-de-kyoto_1617695_3244.html.
- [34] *Net Market Share*. URL : <http://www.netmarketshare.com/>.

- [35] *NumPy*. URL : <http://www.numpy.org/>.
- [36] *OCZ*. URL : <http://www.ocz.com/>.
- [37] *Open Energy Monitor*. URL : <http://openenergymonitor.org/emon/>.
- [38] *Performance Comparison - C++ / Java / Python / Ruby/ Jython / JRuby / Groovy*. URL : <http://blog.dhananjaynene.com/2008/07/performance-comparison-c-java-python-ruby-jython-jruby-groovy/>.
- [39] P.P. « La fin des primes photovoltaïques ». In : *La libre* (2009). URL : <http://www.lalibre.be/actu/belgique/article/535798/la-fin-des-primes-photovoltaïques.html>.
- [40] *Pylons*. URL : <http://www.pylonsproject.org/>.
- [41] *Python*. URL : <http://www.python.org>.
- [42] *Python 3 vs C++ g++*. URL : <http://benchmarksgame.alioth.debian.org/u32q/benchmark.php?test=all&lang=python&lang2=gpp&box=1>.
- [43] *Python - Other platforms download*. URL : <http://www.python.org/getit/other/>.
- [44] *Python Package Index*. URL : <http://www.python.org/getit/other/>.
- [45] *Raspberry Pi Website*. URL : <http://www.raspberrypi.org/about>.
- [46] *Recommendation against Python ?* URL : <https://groups.google.com/forum/?fromgroups#!topic/unladen-swallow/TtvEBvVEZD4>.
- [47] *RTBF*. URL : <http://www.rtbf.be>.
- [48] *SciPy*. URL : <http://www.scipy.org>.
- [49] *SmartBox*. URL : <https://www.electrabel.be/fr/particulier/prix-gaz-electricite-fournisseur/smartenergybox>.
- [50] *Société Wallone Des Eaux*. URL : <http://www.swde.be/>.
- [51] UCL. *Pré-Dimensionnement*. URL : <http://www-energie.arch.ucl.ac.be/cogeneration/CDRom/cogeneration/evaluer/frames/cbcogenepredim.htm>.
- [52] UVED. *Les Années 70... Un tournant décisif : développement et croissance ne vont plus de pair!* 2006. URL : http://www.uved.fr/fileadmin/user_upload/modules_introduction/module4/site/html/1-approche_1-2-contexte_2.html.
- [53] *WattVision*. URL : <http://www.wattvision.com/info/products>.
- [54] *WesternDigital*. URL : <http://www.wdc.com/fr/products/internal/desktop/>.
- [55] R. WEYLER. *Greenpeace : How a Group of Ecologists, Journalists, and Visionaries Changed the World*. Rodale Books, 2004. ISBN : 9781594861062. URL : <http://books.google.be/books?id=M1GW445y2n4C>.

- [56] *Wget*. URL : <http://www.gnu.org/software/wget/>.
- [57] *Wikipedia - Comparison of Nvidia graphics processing units*. URL : http://en.wikipedia.org/wiki/Comparison_of_Nvidia_graphics_processing-units.
- [58] *Xbee*. URL : <http://www.digi.com/xbee/>.
- [59] *yuglify*. URL : <https://github.com/yui/yuglify>.