



Haute Ecole
Libre de
Bruxelles

2021-2022

Programmation Web II

Projet HELBImmo - Rapport final

Jessy ADAM

Professeur :

M. Riggio



2021

Table des matières

1	Introduction	1
2	Technologies utilisées	2
2.1	Back-end	2
2.2	Front-end	2
3	Fonctionnalités	3
3.1	Fonctionnalités incluses avec la base fonctionnelle	3
3.2	Fonctionnalités obligatoires	5
3.2.1	Système de favoris	5
3.2.2	Système de statistiques	6
3.2.3	Système de recherches	8
3.2.4	Système de cartes	9
3.2.5	Système de notifications	10
3.2.6	Adaptation vers un site d'immobilier	11
3.3	Fonctionnalités supplémentaires	12
3.3.1	Système de Question-Réponse	12
3.3.2	Additions Diverses	13
4	Analyse	15
4.1	Analyse de l'application finale	15
4.2	Comparaison aux premiers schémas	16
5	Limitations et développement futur	17
5.1	Limitaions	17

5.1.1	Graphismes et ergonomie	17
5.1.2	Structure interne	19
5.1.3	Mon magnum opus	20
5.2	Développement futur	22

6	Conclusion	23
----------	-------------------	-----------

Dans le cadre du cours de Programmation Web II, il nous a été demandé de réaliser une application web d'immobilier, permettant de consulter et de publier des annonces immobilières, tout en adhérant à certaines contraintes.

Ceci est donc le rapport final de ma version de ce projet de fin de quadrimestre, dénommé "HELBImmo" par manque de créativité.

Dans celui-ci, je traiterai des technologies utilisées, des fonctionnalités implémentées, d'une analyse de la structure de l'application, des limitations de celle-ci, et de ce qui aurait pu être.



2.1 Back-end

Comme convenu dans les consignes, cette application a été réalisée avec le framework Django qui utilise donc le langage Python, et contient un système de templates de page réutilisables et une base de données prête à l'emploi. Elle est aussi basée sur le *tutoriel de Corey Schafer* [1] où il utilise les extensions :

- **Django-crispy-forms**, qui améliore l'aspect graphiques des formulaires;
- et **Pillow**, qui permet à Python d'utiliser des images respectivement;

django

2.2 Front-end

Aussi en plus du Html, JavaScript et CSS nécessaires à toute page web, j'ai aussi utilisé :

- **Bootstrap 4**, une librairie CSS qui facilite la présentation de la page;
- **Bootstrap Icons**, une librairie d'icônes à utiliser dans les pages html;
- **jQuery**, pour faciliter la manipulation de la page via Javascript et pour contacter la base de donnée via des requêtes AJAX;
- **Openlayers**, pour pouvoir afficher et manipuler des cartes;
- et **OpenLayers Control Geocoder**, un plugin pour OpenLayers permettant de trouver sur une carte l'endroit où se trouve une adresse;

3.1 Fonctionnalités incluses avec la base fonctionnelle

Comme dit plus tôt, cette application est basée sur un tutoriel Django créé par Corey Schafer [1]. Cependant, commencer ce projet n'a pas été sans encombres. Comme j'ai fait ce projet dans un environnement Windows, au lieu du Linux recommandé par le professeur, j'ai dû faire certaines choses au préalable. Notamment, j'ai dû modifier la variable d'environnement "PATH" afin d'inclure le chemin de mon installation python car sinon la console Windows ne la reconnaissait pas.

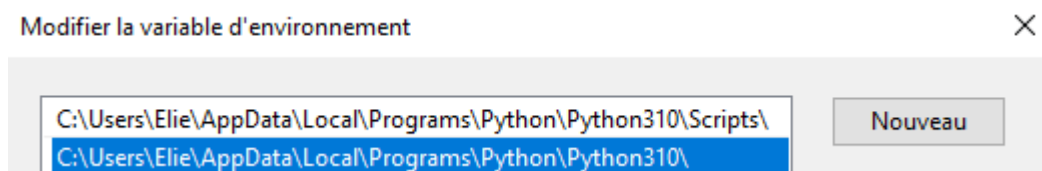


FIGURE 3.1: Chemin vers mon installation python

Ensuite, en essayant d'initialiser l'application Django j'ai découvert que la console, bien qu'elle reconnaissait pip et python, ne reconnaissait pas Django et les autres scripts installés sur l'installation python globale du PC, j'ai donc dû suivre un autre tutoriel de Corey Schafer [2], afin de créer un environnement virtuel contenant tout les scripts nécessaires au bon fonctionnement du site, grâce à virtualenv.

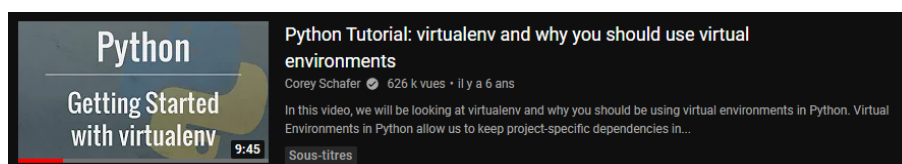


FIGURE 3.2: Le tutoriel en question

Après l'avoir fait séparément sur mon ordinateur fixe et portable, j'ai pu finalement commencer l'application, après avoir créé un repos GitHub pour le synchroniser entre les deux. J'ai donc suivi la série de tutoriel de Corey Schafer [1] et ai pu le finir (sans la partie concernant le déploiement du site) sans plus d'encombres.

Celui-ci résulte donc en une base fonctionnelle qui offre déjà certaines fonctionnalités. Celles-ci étant :

- un système d'utilisateurs pouvant se connecter, se déconnecter, mettre leur information à jour, et réinitialiser leur mot de passe;
- un système d'annonce pouvant être publiée avec un titre et du contenu textuel;
- et une collection de pages comprenant la page d'accueil montrant les annonces tout en étant paginées, une page "À propos", et les pages nécessaires à la gestion de son compte utilisateur.

Or, pour l'instant cette base reste un blog, il m'a donc fallu donc le convertir en site d'immobilier.

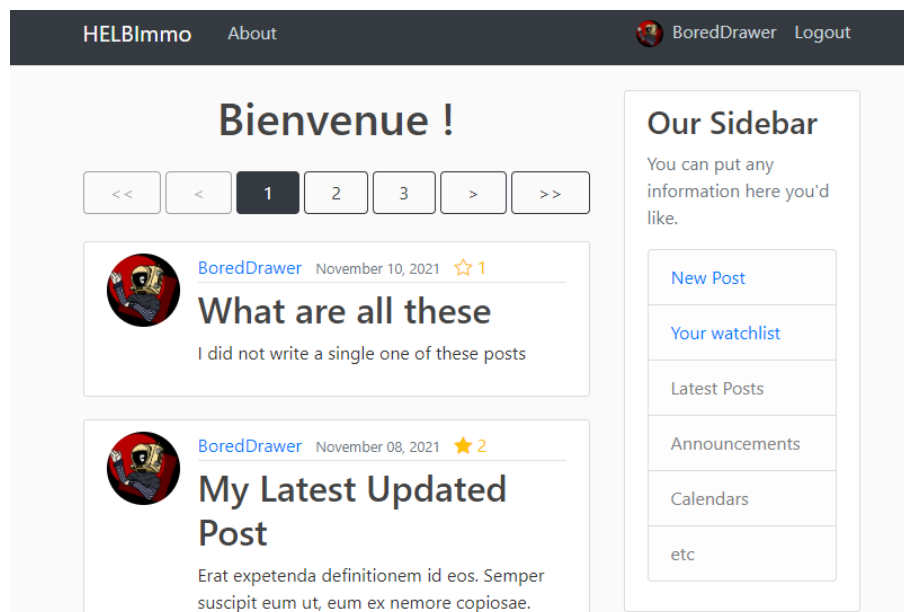


FIGURE 3.3: L'état du site peu après la fin du tutoriel

3.2 Fonctionnalités obligatoires

L'application devait donc avoir certaines fonctionnalités, dites "de base", obligatoires, celles-ci étant ¹ :

- Un système de recherche d'annonces en fonction de certains critères.
- Un système d'inscription/connexion sur la plateforme.
- Un système de profil pour les utilisateurs inscrits, utile à l'utilisation du site.
- Des notifications en cas de nouvelles annonces aux utilisateurs possiblement intéressés.
- Une liste de favoris (watchlist).
- Une carte permettant de visualiser l'emplacement des biens mis en vente.
- Un système de statistiques permettant de visualiser graphiquement (à l'aide de diagrammes) certaines données utiles aux personnes publiant des annonces.

À la fin du tutoriel, j'avais donc le système d'inscription/connexion et de profil, il a fallut donc ajouter le reste.

3.2.1 Système de favoris

La première fonctionnalité que j'ai implémenté est un système de favoris, où un utilisateur peut ajouter et enlever toutes les annonces qui lui convient à ses favoris. Il peut ensuite consulter sa liste de favoris pour retrouver les biens l'intéressait.

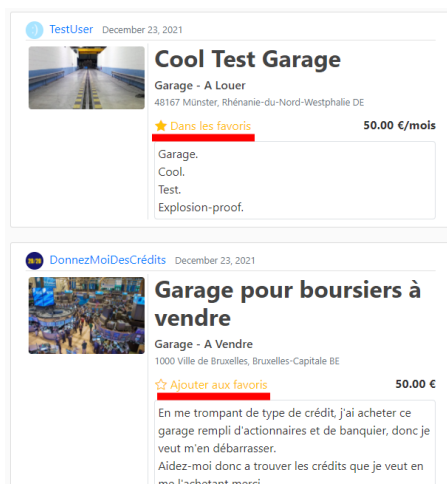


FIGURE 3.4: Exemple de la fonction des favoris

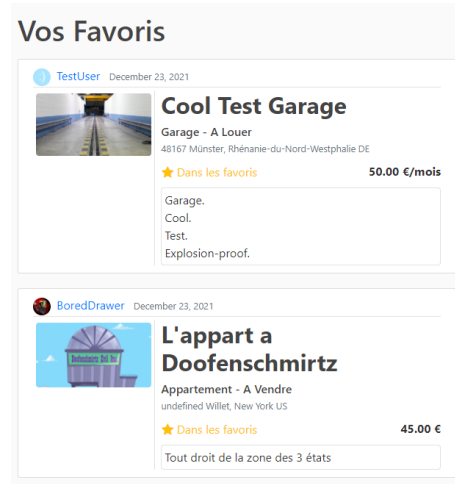


FIGURE 3.5: Aperçu de la page des favoris

Un gros problème que je voulais résoudre est que je voulais que la page ne soit pas rafraîchie un fois qu'un utilisateur décide de mettre une annonce dans ces favoris.

1. Tiré du document de consignes [3]

Il fallait donc le faire coté client avec du Javascript. C'est donc à ce moment que j'ai découvert le joyeux monde des requêtes AJAX (Asynchronous JavaScript and XML), permettant "d'effectuer des requêtes au serveur web et, en conséquence, de modifier partiellement la page web affichée sur le poste client sans avoir à afficher une nouvelle page complète" [4].

J'ai donc passé plusieurs jours à essayer de comprendre ces requêtes, en me basant sur divers tutoriels textuels et réponses de stackoverflow.com. J'y ai passé autant de temps particulièrement car les requêtes que j'ai utilisées viennent de jQuery que j'ai du apprendre dans la foulée en plus des requêtes en elles-même afin de comprendre ce qu'il se passait. Mais éventuellement j'ai pu organiser ça en envoyant une requête à une url faite expressément pour ajouter ou enlever une annonce des favoris de l'utilisateur (s'il est connecté bien sur) et rapporter les résultats. J'ai donc pu réutiliser ces requêtes pour d'autres actions qui ont rendu l'application plus dynamique.

A noter que les requêtes sont pour finir, relativement petites :

```
$(".js-fav").click(function () {  
    let btn = $(this);  
    let postid = $(this).data('postid');  
    console.log("fav " + postid + "!");  
    $.ajax({  
        url: btn.attr('fav-url'),  
        data: {  
            'postid': postid  
        },  
        success: function (result) {  
            if (result.added === true) {  
                btn.children().first().removeClass("bi-star").addClass("bi-star-fill");  
                btn.children().last().text("Dans les favoris");  
            }  
            else {  
                btn.children().first().removeClass("bi-star-fill").addClass("bi-star");  
                btn.children().last().text("Ajouter aux favoris");  
            }  
        }  
    });  
});
```

FIGURE 3.6: Requêtes Ajax permettant d'ajouter/retirer un favoris (ici appliquée a tout les objets ayant la classe "js-fav")

Plus tard, j'ai du changer les favoris afin d'y inclure une date, ce qui fait que j'ai du les changer d'un simple attribut du profil utilisateur à son propre modèle².

3.2.2 Système de statistiques

Ayant fait les favoris je me suis ensuite penché sur le système de statistiques, où il fallait représenter en tant que graphique les consultations d'une annonce.

2. Un modèle Django est l'équivalent d'une table SQL

J'ai donc commencé par faire le système de consultations avant de faire qui que ce soit. J'ai donc fait un nouveau modèle représentant une consultation qui est juste composé d'une date et d'une clé étrangère vers l'annonce concernée. J'ai aussi réutilisé une requête AJAX, cette fois-ci permettant de créer une consultations a chaque fois que la page est chargée.

J'ai aussi préparé la page où allaient se trouver les graphiques avec l'information que j'avais déjà, qui est donc le nombre de consultation et de favoris, et un bouton qui laisse l'auteur d'une annonce y accéder.



FIGURE 3.7: Aperçu de la page des statistiques ainsi que le bouton qui va avec

J'ai ensuite du faire des graphiques, que j'ai fait avec un sous-template représentant un graphique en bâtonnet dans lequel je pourvois les données à inclure dans le graphique ainsi que sa taille et couleur. Le graphique en lui-même est fait en utilisant des classes Bootstrap concernant les flex-box qui une fois imbriquées l'une dans l'autre donne l'air d'un graphique. La taille des bâtonnets est ensuite calculée avant que Django envoie la page, donnant ce résultat.

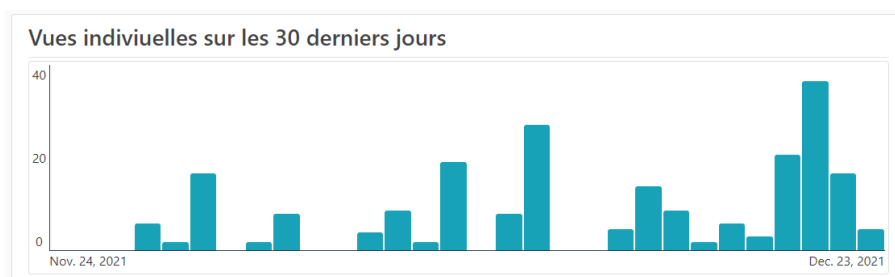


FIGURE 3.8: Graphique représentant les vues d'une annonce sur les 30 derniers jours

Ce système est probablement plus limité que des bibliothèques telles que Chart.js, mais il est amplement suffisant pour mes besoins.

3.2.3 Système de recherches

J'ai ensuite ajouté le système de recherche, en commençant par placer une barre de recherche dans la barre de navigation du site, pour quelle soit accessible dans toutes les pages. J'ai ensuite suivi un tutoriel sur le sujet [5] et ai pu l'implémenter sans trop d'embarras.



FIGURE 3.9: Aperçu de la barre de recherche

J'ai ensuite ajouté des filtres en attachant un formulaire à la barre de recherche, accessible via un menu dropdown apparaissant qu'on clique la fleche à coté du bouton de recherche. On vérifie ensuite si on filtre les annonces avec le texte si on a des informations en plus et, si oui, on les utilise pour filtrer une fois de plus les résultats.

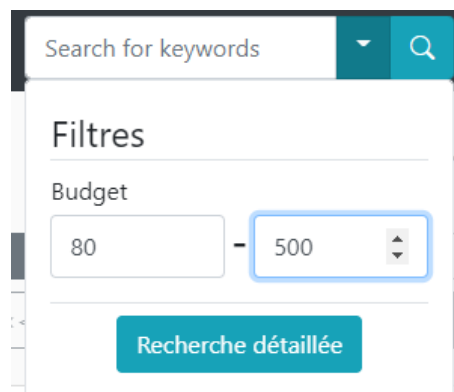


FIGURE 3.10: Aperçu de la barre de recherche avec les filtres ouverts

Un problème que j'ai eu était que comme la pagination et la recherche étaient des requêtes GET, l'un effaçait l'autre. Et, bien que ça du sens lors de la recherche, la pagination effaçant la recherche causait une erreur aussi bien technique que conceptuelle. J'ai donc trouvé une solution qui était d'utiliser un script Javascript [6] pour insérer les paramètres de la recherche dans les liens des boutons de pagination.

J'ai aussi fait en sorte que les paramètres non-utilisés ne soient pas compris dans la requête pour éviter l'encombrement.

3.2.4 Système de cartes

J'ai par après ajouté le système de carte, permettant de visualiser l'emplacement d'un bien dans le monde. J'ai commencé par ajouter la latitude, longitude et l'adresse (séparée en plusieurs champs) aux attributs du modèle des annonces. En utilisant OpenLayers, l'emplacement du bien est affiché sur une carte dans la vue détaillée d'une annonce.

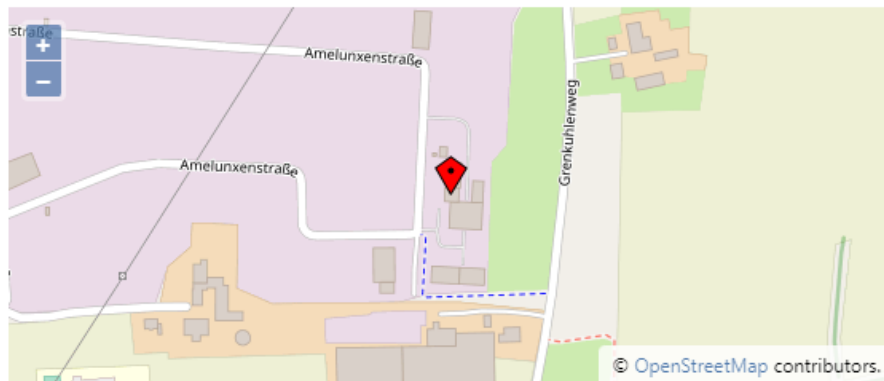


FIGURE 3.11: Carte montrant l'emplacement d'un bien

J'ai aussi utilisé la même carte ainsi que le géocodeur Openlayers mentionné au début pour créer un widget, à mettre dans les formulaires, permettant à l'utilisateur de saisir l'adresse de leur bien pour choisir son emplacement sur la carte.

Je n'ai découvert que trop tard que des solutions préfaites existent donc je l'ai créé moi-même en utilisant des entrées de formulaires cachées qui seraient remplies après l'envoi du formulaire, le processus de sa conception a pris une semaine tout au plus, mais j'en parlerai dans la section appropriée.

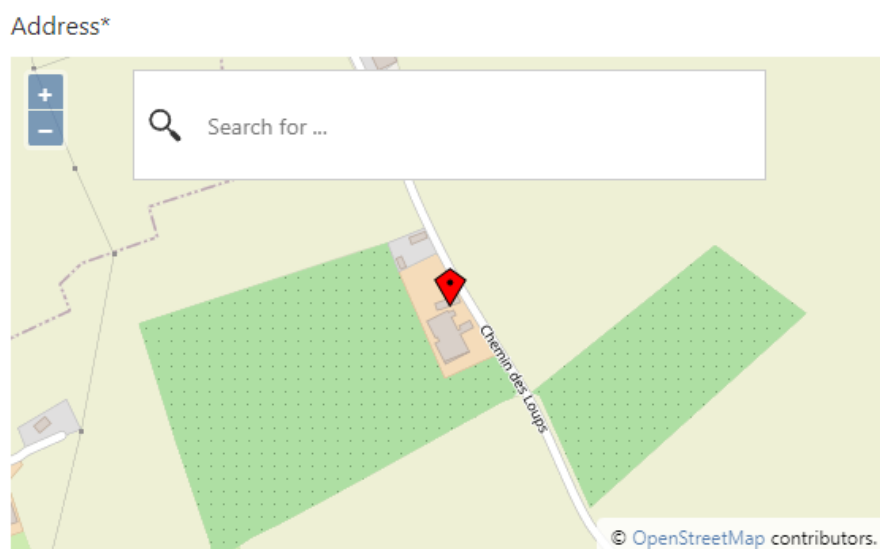


FIGURE 3.12: Exemple de widget de carte

3.2.5 Système de notifications

Enfin, comme dernière fonctionnalité obligatoire, j'ai ajouter un système de notification, qui envoie une si un utilisateur pourrait être intéressé. J'ai fait cela en donnant à l'utilisateur des critères qu'il peut mettre à jour à sa guise et qui ont donc leur page dédiée. En plus des entrées normale comme le budget ou la méthode de vente, j'ai décidé d'ajouter l'option de rendre ses critères publics ou pas, ce qui équivaut à désactiver les notifications de recommandation si besoin.

The image shows a web form titled "Vos Critères". It contains five input fields: a dropdown menu for "Notifications" with "Activées" selected, a text input for "Budget*" containing "150,00", a dropdown menu for "Méthode de vente" with "Vente" selected, a text input for "Localité*" containing "Bruxelles", and a text input for "Localité*" containing "Bruxelles".

FIGURE 3.13: Aperçu de la page des critères

J'ai ensuite fait un modèle et une page pour les notifications, où elles seront différenciés en fonction du type de notification et auront un lien vers l'annonce concernée. La page étant accessible tant que l'utilisateur est connecté depuis la barre latérale.

J'ai, après, mis en place un système qui envoie une notifications à tous les utilisateurs dont les critères correspondent quand on poste une annonce, et a tout ceux qui l'on mit en favoris quand on modifie une annonce.

Pour finir j'ai attaché un booléen à chaque notification pour savoir si elle à été déjà lue ou non et aie créé une requête AJAX qui les mets à jour quand on consulte la page des notifications. Les notifications non lues apparaîtront en jaune.

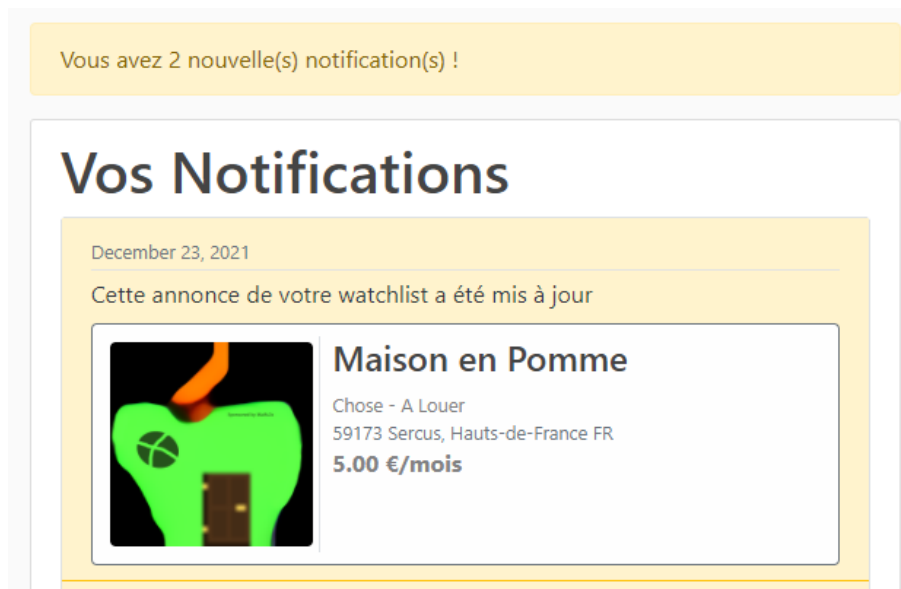


FIGURE 3.14: Aperçu d'une notification non lue

3.2.6 Adaptation vers un site d'immobilier

Pendant tout cela, j'ai aussi fait quelques changements aux annonces pour qu'elles aient plus de rapport à un bien immobilier. J'ai ajouté des informations comme le prix et la surface habitable, ainsi que la méthode de vente et le type de bien parmi d'autres. Sachant que le type de bien change certains paramètres, j'ai utilisé un script Javascript pour dynamiquement changer les informations montrées pendant la vue détaillée et formulaire de l'annonce. Le type de vente change aussi la manière dont est affiché le prix.

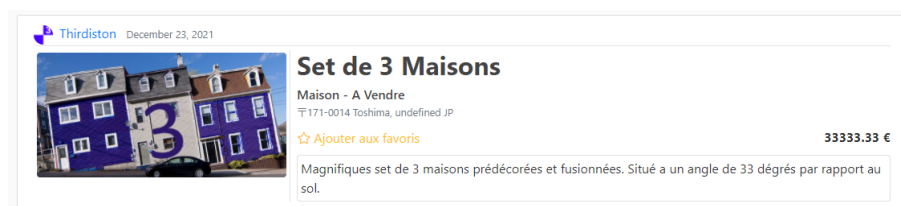


FIGURE 3.15: Exemple d'annonce simplifiée

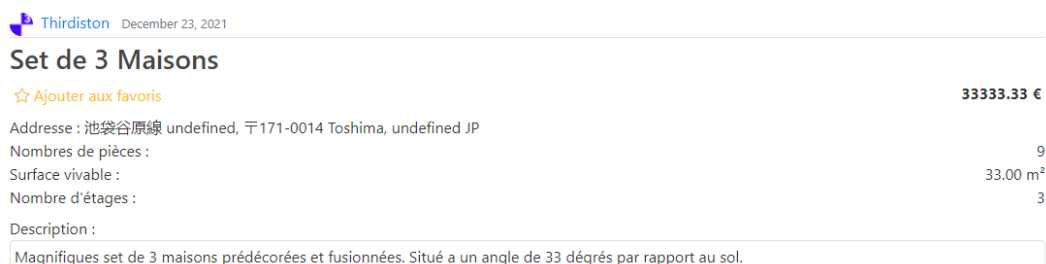


FIGURE 3.16: Exemple de l'information montrée pour un annonce de maison

3.3 Fonctionnalités supplémentaires

3.3.1 Système de Question-Réponse

Comme fonctionnalité supplémentaire principale, j'ai choisi d'inclure un système de question-réponse, où les utilisateurs intéressés peuvent demander plus de détails si ceux-ci ne sont pas déjà inclus dans l'annonce.

Donc sur toute annonce qui n'est pas la leur, les utilisateurs peuvent laisser des questions grâce à un formulaire placé en bas de la page détaillée de l'annonce. Ce formulaire est ensuite envoyé via AJAX à la base de donnée et est ensuite affichée dans la page.



FIGURE 3.17: Exemple de publication de question

L'utilisateur a aussi l'option de supprimer sa propre question si besoin, mais est d'abord averti pour éviter les accidents. Ceci utilise un système Javascript qui sert à dynamiquement cacher et afficher les bonnes sections de la page, et que je réutiliserai juste après.



FIGURE 3.18: Aperçu du système de suppression

L'auteur de l'annonce peut ensuite répondre à la question sur son annonce grâce à un bouton qui s'affiche quand il la consulte. Par le biais du même système, ce bouton affiche un formulaire qui servira à répondre à la question en envoyant une autre requête AJAX. Il a aussi l'option de la supprimer si besoin.

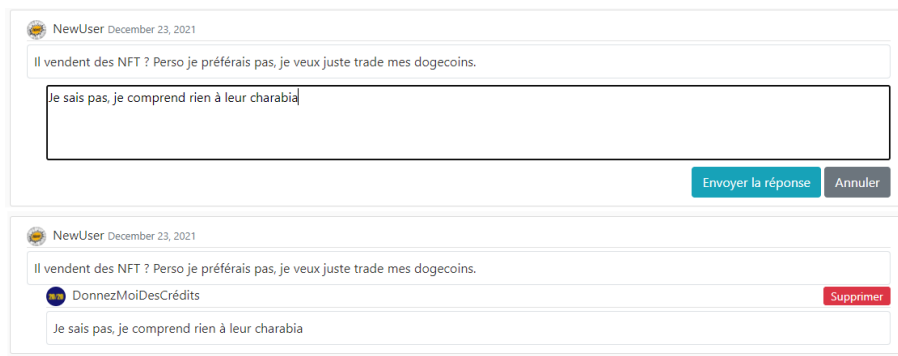


FIGURE 3.19: Exemple de publication de réponse

3.3.2 Additions Diverses

J'ai aussi fait d'autres ajouts divers mais moindre en complexité, je vais donc en parler un peu ici. J'ai donc :

- Fixé la barre latérale à l'écran et l'ai rendu plus fonctionnelle avec divers liens tout en prenant compte du type d'utilisateur.

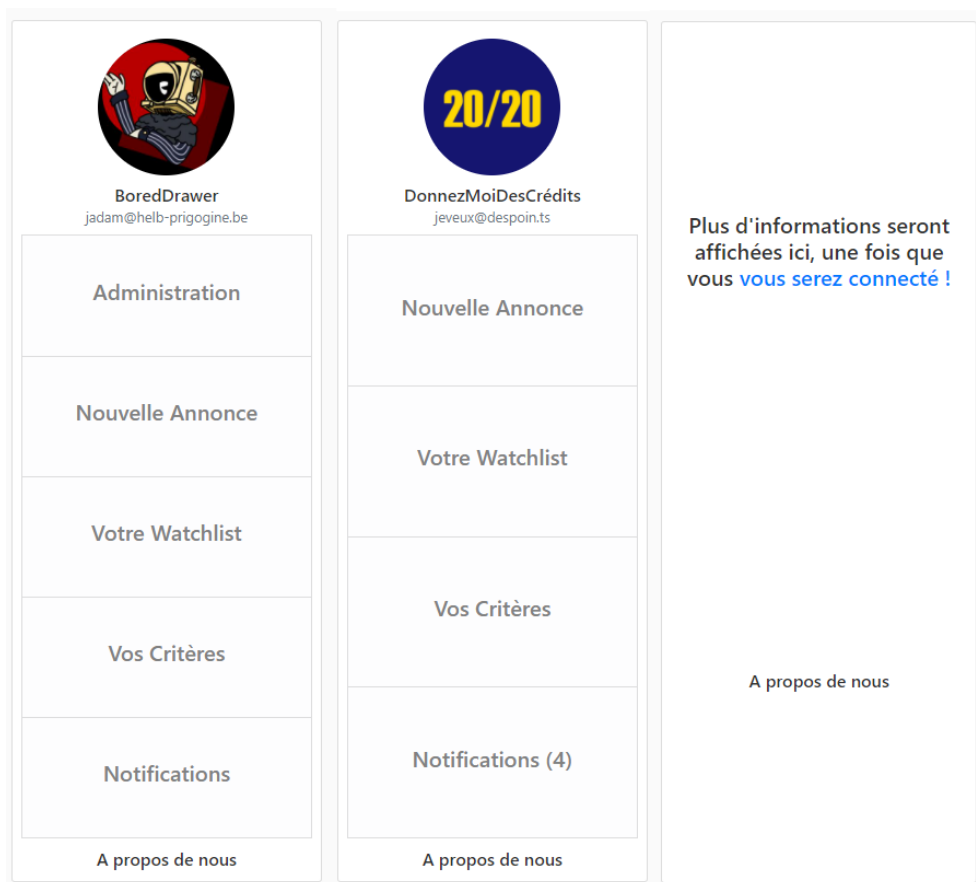


FIGURE 3.20: Aperçu de la barre latérale pour un administrateur, utilisateur normale, et utilisateur non connecté

- Amélioré la pagination en laissant le choix à l'utilisateur de sélectionner le nombre d'annonces par page.

A screenshot of a pagination control. It features a 'Per page :' label followed by four buttons: '5' (selected), '10', '25', and '50'. Below these are six navigation buttons: '<<', '<', '1' (selected), '2', '>', and '>>'.

FIGURE 3.21: Aperçu du système de pagination

- Permet à l'utilisateur de désactiver les recommandations, s'il en a envie.

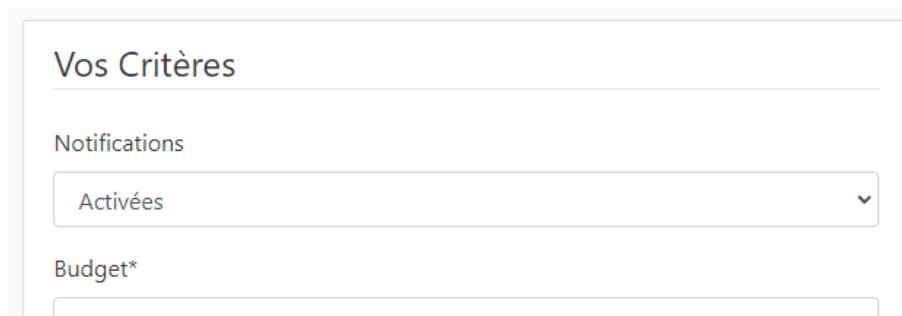
A screenshot of a settings panel titled 'Vos Critères'. It contains a 'Notifications' section with a dropdown menu currently set to 'Activées'. Below this is a 'Budget*' section with a text input field.

FIGURE 3.22: Aperçu du réglage de notification dans les critères

- Crée des mixins pour éviter la duplication de code à chaque fois qu'on doit paginer des éléments ou récupérer les favoris d'un utilisateur.

4.1 Analyse de l'application finale

La structure globale de cette application web est assez simple, elle est composée de deux applications Django, une qui gère les utilisateurs et ce qui y est lié dénommé "users", et une autre, dénommé "blog", qui gère les annonces et tout ce qui y est lié.

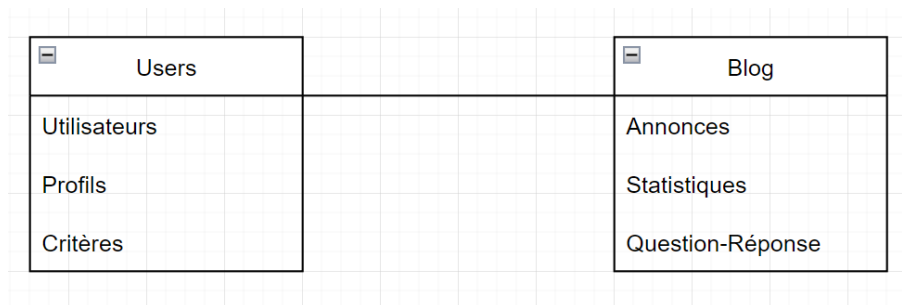


FIGURE 4.1: Schéma de la structure globale de l'application

Pour aller plus en détail voici un schéma reprenant tous les modèles qui ont été créés pour cette application :

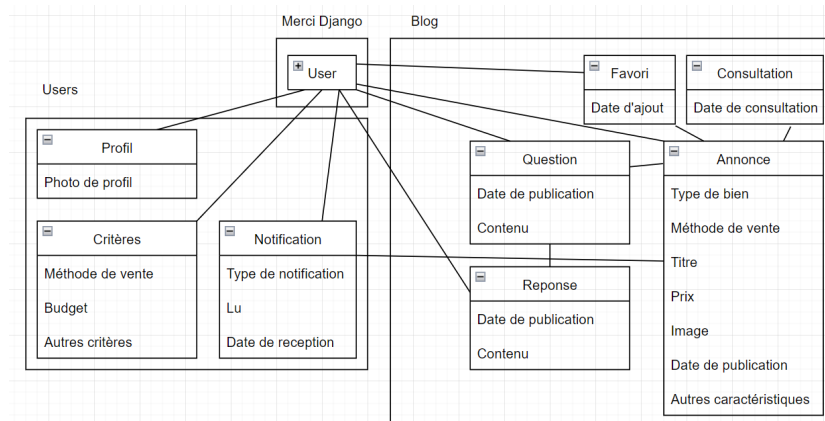


FIGURE 4.2: Schéma de la structure globale de l'application en détail

Ici vous pouvez voir que le modèle utilisateur inclus avec Django a été étendu grâce au modèle de profil et de critères, qui sont gardés séparés pour garder la responsabilité des critères au critères, tout en laissant la possibilité d'ajouter des attributs à l'utilisateur.

Les questions et réponses ont dû être séparés afin de pouvoir joindre une réponse à sa question au lieu de l'annonce sur lequel il est tandis que la question fait le lien avec l'annonce.

Les consultations ne sont liés qu'à l'annonce à laquelle elles sont liées.

4.2 Comparaison aux premiers schémas

Si on compare cela au premier plan que j'avais créé on peut voir certaines différences.

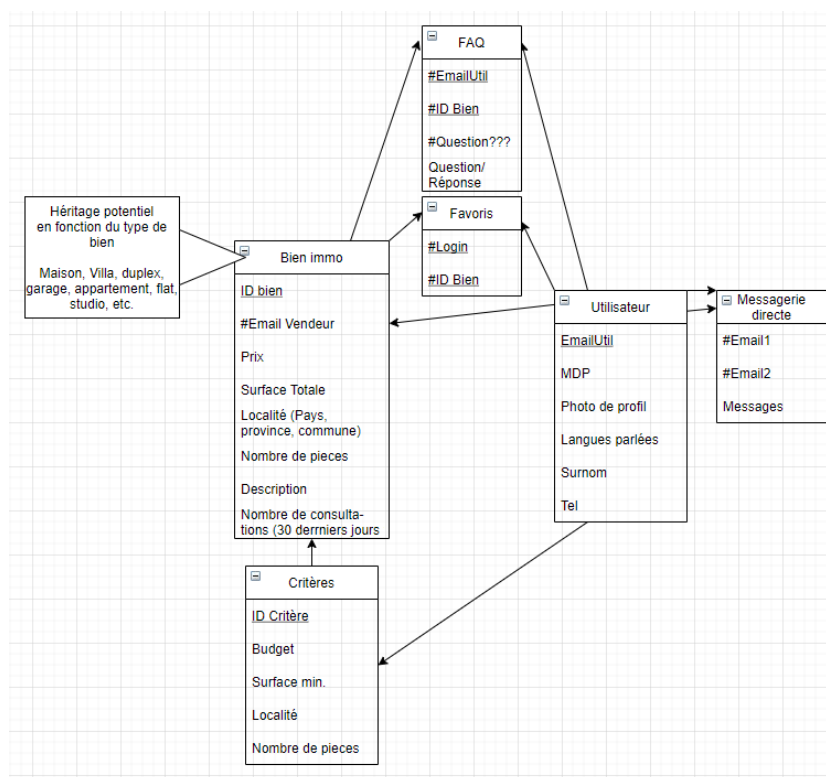


FIGURE 4.3: Schéma de structure créé avant la réalisation de l'application

On peut remarquer que beaucoup de choses sont manquantes, notamment la séparation entre question et réponse, les consultations et les notifications.

J'ai aussi opté contre l'héritage pour différencier les types d'annonces par héritage car la même chose a pu être faite grâce à un attribut de celles-ci, tout en rendant la réalisation plus simple. Les critères ne sont plus hérités d'une annonce car je les ai considérés trop différents l'un de l'autre pour que ça en vaille la peine.

La messagerie directe a aussi été abandonnée par manque de temps.

Limitations et développement futur

Bien sur cette application n'est pas parfaite, il convient donc que je parle des limitations et autres point négatifs de celle-ci.

5.1 Limitaions

5.1.1 Graphismes et ergonomie

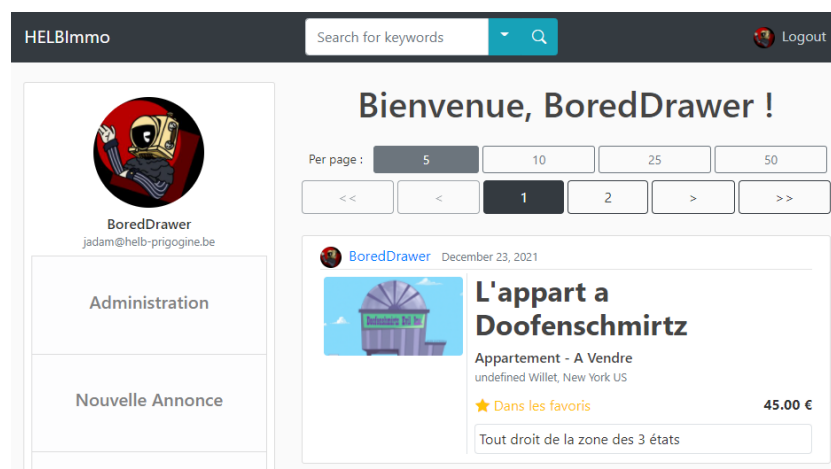


FIGURE 5.1: Aperçu global du site

Si vous regardez l'état du site maintenant, vous pourrez remarquer qu'il n'a pas l'air si différent que ça de la base fonctionnelle, mise à part quelques ajustement de disposition. En manquant de temps, je n'ai pas pu y faire grand chose ce qui fait que le site n'est pas le plus responsive.

Par exemple j'ai mentionné que la barre latérale était fixée à l'écran et défilait avec, cela veut aussi dire que l'effet reste peut importe la taille de l'écran, ce qui à l'air un peu ridicule sur téléphone. L'idéal aurait été de faire en sorte que l'on puisse cacher la barre latérale mais je n'ai pas eu le temps de le faire.

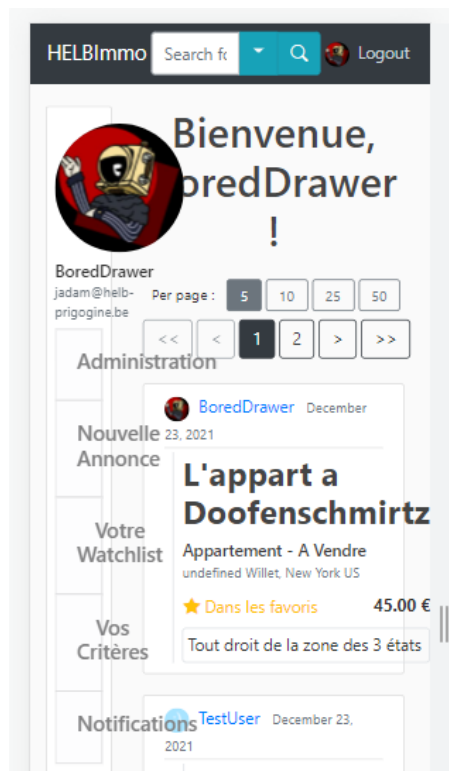


FIGURE 5.2: Aspect du site sous mobile

De plus, il n'y a pas de limite à la taille que peut prendre une image, ni aucune considération pour le ratio d'aspect ce qui veut dire que l'on peut envoyer des images beaucoup trop grandes sans transformation de celles-ci et un aspect carré ou vertical peut causer des problèmes d'affichage.

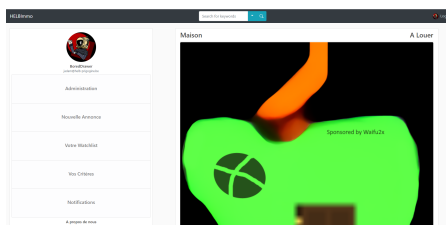


FIGURE 5.3: Exemple de bug graphique lié au ratio d'aspect

Il y a aussi le fait que les filtres ne sont pas particulièrement ergonomiques, étant mis en plein milieu de la barre de recherche. En plus de cela, ils n'ont pas leur section dédiée sur la page de recherche, ce qui fait qu'il faut retourner dans le même menu à chaque fois.

Vous avez probablement remarqué aussi qu'il y a très peu de choix de filtres, vu qu'on ne peut mettre qu'une fourchette de prix. Bien que ça ne prendrait que peu de temps, je n'ai pas eu le temps de le faire moi-même.

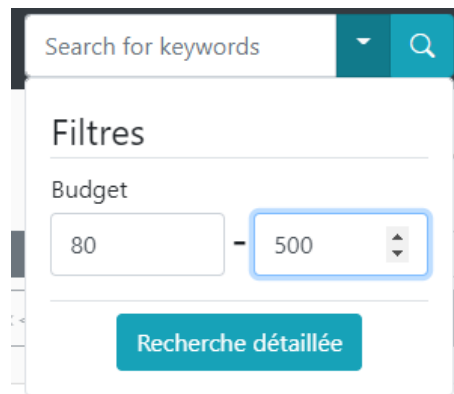


FIGURE 5.4: Aperçu de la barre de recherche avec les filtres ouverts

Un autre problème est que l'application utilise beaucoup de Javascript pour faire des choses relativement importantes comme afficher les bonnes informations en fonction du type de bien. Il aurait fallu le faire côté serveur avant d'envoyer la page, au cas où l'utilisateur a désactivé Javascript ou essaye de casser l'application.

Un dernier assez gros problème est que l'application est encore plongée dans un mixe d'anglais et français, ce qui fait qu'elle ne peut être utilisée efficacement que par des bilingues.

5.1.2 Structure interne

Il y a aussi quelques problèmes au niveau interne, un des plus grand étant la vulnérabilité au spam. Par exemple, une consultation n'est liée qu'à l'annonce qui la concerne, et n'utilise pas le système de session de Django on peut donc en faire en boucle en rafraîchissant juste la page. Pire comme la requête pour la consultation se fait par une requête AJAX basée en Javascript, en trouvant la bonne fonction, on pourrait juste faire une boucle for pour essayer de surcharger le serveur, ou juste faire gonfler ses statistiques.

Ça n'aide pas que les consultations sont stockées de manière individuelles, alors qu'on pourrait les regrouper par jour pour éviter de bourrer la base de données.

Un autre problème est qu'il y a très peu qui différencie les types de biens dans les annonces, on ne vérifie donc pas qu'il y ait de l'information seulement là où il en faut. Par exemple, un garage a des étages, on ne peut juste pas le voir.

Aussi a ma mauvaise habitude le code manque certainement de commentaires, j'ai fait ce que j'ai pu pour corriger cette erreur mais ce n'est sûrement pas assez.

Il y a aussi le fait que tout a essentiellement été fourré dans l'application "blog" sans considération pour la lisibilité, idéalement il y aurait une application pour les statistiques et une pour le système de question réponse, afin de faciliter leur réutilisation.

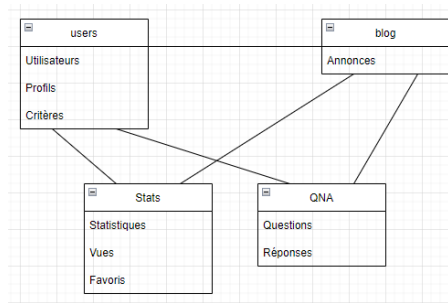


FIGURE 5.5: Structure idéale pensée après création de l'application, non-implémentée

5.1.3 Mon magnum opus

"Je n'ai découvert que trop tard que des solutions préfaites existent donc je l'ai créé moi-même en utilisant des entrées de formulaires cachées qui seraient remplies après l'envoi du formulaire, le processus de sa conception a pris une semaine tout au plus, mais j'en parlerai dans la section appropriée." [7] Nous voici dans "la section appropriée" donc laissez-moi vous parler de cette magnifique, magnifique carte.

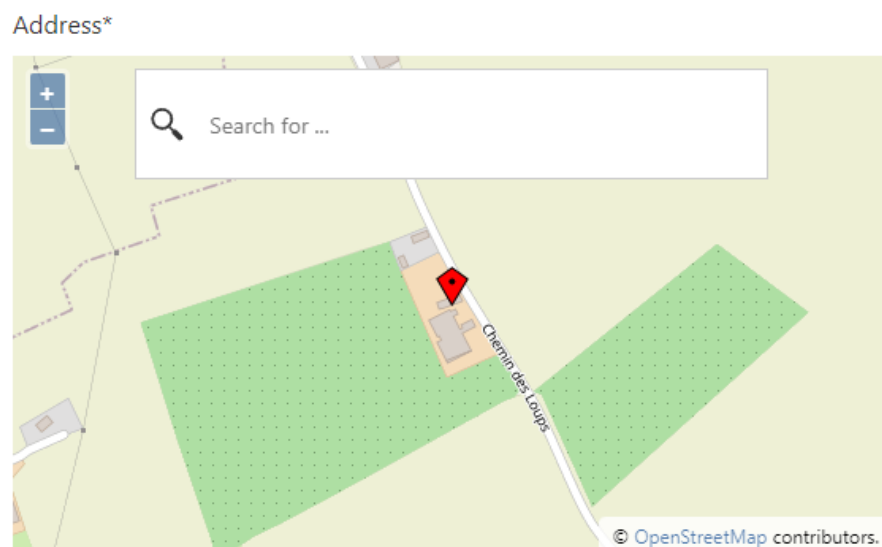


FIGURE 5.6: Exemple de widget de carte

Ce chef d'oeuvre contemporain, m'a pris une semaine, à cause d'un API assez complexe mélangé à un manque cruel de direction. J'ai pu mettre la carte dans la page détaillée d'une annonce sans problème, il suffisait juste de faire un copier coller, c'est vraiment le fait de le mettre dans le formulaire qui posait problème. Parce qu'il y a beaucoup à considérer, comme les formulaires sont préfaits il faut savoir quoi remplacer, un field ? un widget ? Maintenant ça me paraît évident mais quand on a pas tout compris ça prend du temps et beaucoup d'onglets ouverts.

Il y a aussi le fait que je n'étais pas sûr comment faire le widget que je devais remplacer. Est-ce qu'il fallait placer un marqueur là où on clique? Taper une adresse pour qu'un programme le retrouve? Comment s'appellerait ce programme? Comment est-ce que ça marcherait? Beaucoup de questions où je n'étais pas sûr de la réponse.

Il faut aussi ajouter à cela que les exemples faits au cours, bien que très utiles, ne correspondaient à aucune des documentations que j'avais pu trouver, que ce soit sur le site d'Openlayers, même ou autre part. Tout ça, avec le fait qu'Openlayers n'est juste pas l'API la plus simple, a fait que j'étais clairement perdu.

Donc j'ai fait quelque chose de simple, j'ai tout jeté à la poubelle, tout supprimer, et j'ai recommencé depuis le début. J'ai voulu faire une démo où j'étais sûr que tout fonctionnait exactement comme je le voulais en prenant méticuleusement des exemples fournis par Openlayers pour faire le widget parfait. Certes, ça ne veut toujours pas dire que j'avais une idée claire de la situation, mais c'était le début de la fin.

J'avais donc une carte qui fonctionnait décemment bien, j'avais le geocodeur qui fonctionnait bien, la vie était belle, mais maintenant il fallait la mettre dans l'application, et dans un formulaire surtout. Dans mes recherches sporadiques, j'ai donc découvert qu'il fallait créer un widget à insérer dans le formulaire. Mais comme ça ne ressemblait à aucun widget préexistant, j'ai essentiellement dû en faire un depuis la classe de base. C'est aussi à ce moment que j'ai appris à propos des solutions qui existaient déjà, mais le têtard que je voulais être s'était dit que c'était trop tard.

C'est donc en rapiécant très lentement partie par partie les fichiers qu'il fallait à partir de divers tutoriels et ressources, j'avais réussi, je l'ai affiché dans le formulaire, hurra. Maintenant il fallait encore faire les fonctionnalités, parce que certes la carte elle-même fonctionnait, mais elle n'envoyait encore rien au formulaire. Surtout qu'il fallait qu'elle serve pour plusieurs champs du formulaire. Pour j'avais trouvé une réponse sur stackoverflow.com assez intéressante à ce sujet, le problème est qu'elle était très vague [?].

Mais je l'ai suivi, j'ai caché les champs qu'il fallait, j'ai trouvé où mettre la fonction clean après maintes recherches et j'ai utilisé du Javascript pour structurer les données à envoyer. Tout va bien qui finit bien. Mais ça n'empêche pas que cette œuvre d'art moderne est maintenue par du papier collant et de la colle, idéalement il serait remplacé par une solution préfabriquée. Or, le jour que je m'avais imposé comme limite, je l'avais fait fonctionner et donc je le garde, comme mémoire d'une semaine remplie de cris, de pleurs et de frustrations.

Voici donc la raison pour laquelle cette application n'est pas de meilleure qualité. La prochaine fois, j'essayerai de faire preuve de plus de bon sens.

5.2 Développement futur

Si j'aurais eu plus de temps à passer sur ce projet, je pense qu'une des premières choses que je voudrais faire serait de mettre de l'ordre. Parce que bien que la plupart des fonctionnalités que je voulais inclure sont là, elles manquent beaucoup de cohésion, par exemple les questions et réponses devraient envoyer des notifications, les critères et les filtres de recherches ont des champs qui manquent. Et même au niveau de la disposition des fichiers j'aimerais séparer les statistiques et la FAQ du blog pour aider à la lisibilité et passer plus de temps à vérifier qu'on ne puisse pas faire des choses interdites.

Ensuite je voudrais refaire l'aspect du site et essayer de le rendre plus intéressant à regarder. Car déjà qu'il est assez basique, vous en avez probablement déjà vu plusieurs versions. Une idée que j'aimais beaucoup était de faire ressembler les annonces à des vraies pancartes immobilière qu'on peut voir accrochée aux bâtiments à vendre. C'est d'ailleurs pour ça que j'avais opté pour du noir dans la barre de navigation. Il faudrait aussi adapter cette application à plusieurs types d'écrans, notamment les téléphones.

Je voudrais aussi enfin me décider sur la langue de l'application. Une grande chose que j'ai dû délaisser par manque de temps est l'internationalisation du site. J'aurais aimé faire en sorte de pouvoir changer la langue et de pouvoir catégoriser les biens et utilisateurs en fonction des langues qu'ils pouvaient parler mais ça aurait pris trop de temps. Je sais qu'il ne faut pas abandonner des fonctions d'accessibilité en dernière minute mais c'est le choix que j'ai fait.

Une dernière chose que j'avais voulu faire était une messagerie directe, je pense que ça aurait été une fonctionnalité supplémentaire parfaite, car c'était quelque chose qui avait du sens et que j'avais déjà observé chez des agents immobiliers qui utilisent messengers pour contacter des clients potentiels. Mais j'ai finalement opté pour la FAQ d'abord, parce qu'elle paraissait plus simple.

Ceci conclut donc ce rapport de projet, je vous remercie de votre précieux temps et espère avoir gardé votre attention. Dans celui-ci nous avons parler des fonctionnalité de cette application, ce qu'elle utilise, ses limitations et de ce qui aurait pu être. Mis à part les images multiples par annonces cette applications devrait avoir toutes les fonctions demandées et une section FAQ. Et comme j'ai du mal avec les conclusion, je vous avec cette citation de mon cerveau fatigué et beaucoup trop dramatique. Merci encore d'avoir lu.

"Mes rêves de réseau social immobiliers sont écroulé et je m'attriste devant mon twitter avec un prix, mais je me réjoui de mon futur, car grâce à HELBImmo, j'ai appris que tout faire soi-même est une mauvaise idée et que la prochaine fois, je ferai tout ce qui est en mon pouvoir pour passer un Noël, sans code illisible, sans erreurs incessantes, dans le vide qu'est mon crâne et la réjouissance qu'est ma famille, et pour cela mon cher monsieur, merci." [7]

Bibliographie

- [1] Corey SCHAFER. Django Tutorials - YouTube. URL : <https://www.youtube.com/playlist?list=PL-osIE80TeTtoQCKZ03TU5fNfx2UY6U4p>.
- [2] Corey SCHAFER. Python tutorial : virtualenv and why you should use virtual environments. URL : <https://www.youtube.com/watch?v=N5vscPTWK0k>.
- [3] Jonathan RIGGIO. Enoncé du projet : Helbimmo. URL : https://ecampus2122.helb-prigogine.be/pluginfile.php/174447/mod_resource/content/1/Projet%20Programmation%20Web%20II%202122.pdf.
- [4] Ajax (informatique), November 2020. Page Version ID : 176227094. URL : [https://fr.wikipedia.org/w/index.php?title=Ajax_\(informatique\)&oldid=176227094](https://fr.wikipedia.org/w/index.php?title=Ajax_(informatique)&oldid=176227094).
- [5] Learn Django. Django Search Tutorial. URL : <https://learndjango.com/tutorials/django-search-tutorial>.
- [6] annakata. Adding a parameter to the URL with JavaScript - Stack Overflow. URL : <https://stackoverflow.com/questions/486896/adding-a-parameter-to-the-url-with-javascript/487049#487049>.
- [7] Jessy ADAM. Projet helbimmo - rapport final.
- [8] Steve Jalim. How to get a single widget to set 2 fields in Django? - Stack Overflow. URL : <https://stackoverflow.com/questions/2856790/how-to-get-a-single-widget-to-set-2-fields-in-django/2857189#2857189>.