P13 – D.A. Python OpenClassrooms
Implémentation et bilan

Liens :

Trello : https://trello.com/b/lcZB8Z64/p13-checklistmgr
Github : https://github.com/jmlm74/P13-CheckListMgr
Site : https://www.jm-hayons74.fr
Documents annexes :
– P13 – Dossier conception fonctionnelle
– P13 – Dossier conception techniques
– P13 – Dossier Exploitation

Implémentation

L'application est une application Web en Python utilisant le framework Django. Le projet Django a été découpé en plusieurs applications commençant toutes par app_ dans un souci de clarté et de classement:
- app_home : Partie statique du site –> page d'accueil, légal, contact…
- app_utilities :Partie contenant des views (fonctions) utilitaires utilisées par toutes les applications.
- app_user : Partie gestion des utilisateurs et des sociétés (réservé admin de site).
- app_create_checklist : Partie création des check-lists, des lignes et rubriques.
- app_input_checklist : Partie création des exploitants, matériels et adresses (réservé utilisateurs professionnels).
- app_checklist : Partie saisie des check-lists, génération du pdf et sauvegarde.

Outils externes

De nombreux modules ou autres outils externes ont été utilisés : Framework CSS tel que Bootstrap, Bootstrap modal pour les fenêtres modales (dialog boxes), flaticon et Font Awsome pour les icones, Sortable views pour les tableaux triés, Weasyprint pour la génération des fichiers PDF. Des modules javascripts ont aussi été utilisés pour le DragnDrop et l'integration des photos. Enfin, les APIs d'OpentStreetMap et d'OpenWeatherMap ont été utilisées pour la Homepage. L'envoi des mails (reset mot de passe et Check-List) est effectué via l'API de MAILGUN.

Méthodologie

La méthodologie pour le développement s'est fortement inspiré de la méthode agile bien que l'équipe soit plutôt réduite à sa plus simple expression (moi tout seul et mon mentor une fois par semaine !). J'ai tenté de m'approcher le plus possible de ce que pourrait être un projet en extrême programming mais avant de voir cela, détaillons le mode d'organisation du développement.

Le développement a été effectué suivant le mode suivant :
- Application par application. Chaque application pouvant être validée entièrement indépendamment des autres.
- 3 branches Github ont été créées :
    - Devel : Sert uniquement au développement)
    - CI : Une application terminée et testée la branche devel est mergée à la branche CI. Elle passe ainsi automatiquement par le process d'intégration continue de TRAVIS-CI.
    - PROD : Une fois prête à passer en production, la branche CI est mergée dans la branche PROD. La branche PROD est ainsi automatiquement envoyée sur le site de DOCKER-BUB afin de régénérer un conteneur de production qui sera mis en production automatiquement le lundi matin (ou manuellement si besoin).

La méthodologie n'était pas TDD, toutefois les tests unitaires et de recettes faisait partie intégrantes du développement. Une application n'était considérée comme livrée qu'une fois tous les tests passés et validés. Les commentaires terminés étaient aussi un pré-requis à la livraison. Les tests ont été faits avec l'outil Unittest. Les urls ont été validées dans les 2 sens (path et reverse). Les views ont été testées avec le client de test de

Django (Client()) et RequestFactory pour générer les requests. La validation et non validation des forms ont systématiquement été validées.

Tout cela entre dans les bonnes pratiques de l'Xtreme programming que nous allons détailler une par une :
– Client sur site : Je n'ai pas à proprement parlé de client pour ce projet. Je suis mon propre client. Étant donné cet état de fait, j'ai donc un peu joué le rôle de product owner.
– Planning poker : Le simple fait de découper le projet en applications de petites tailles et mises dans le tableau Kanban (cf lien Trello) couple au fait que seul développeur. Le fait de faire des points réguliers et fréquents avec moi-même ne pose pas de problème. De plus, les sessions de mentorat peuvent aussi être assimilées au point hebdomadaire. Enfin le mode de développement explique plus haut entre complètement dans un mode de planning poker/sprints.
– Intégration continue : Là encore, le fait de découper le projet en petites entités a facilité l'intégration continue. C'est une habitude que j'aie prise assez tôt dans le parcours et que je fais en utilisant Travis CI.
Petites livraisons : C'est le type même de développement du projet. C'est aussi mon mode de fonctionnement : Un gros problème est en fait une multitude de petits problèmes plus faciles à résoudre !
Rythme soutenable : Un rythme soutenable pour l'un va être infernal pour une autre personne : Difficile d'être juge et partie la-dessus ! Mais je suis encore vivant !
– Tests Unitaires/fonctionnels : Les tests sont une phase essentielle lors du développement d'un projet. Le projet n'a pas été développé en TDD mais les tests ont très tôt tenu la place qui est la leur dans le projet. Chaqque application livrée n'etait considérée comme terminé que si les tests étaient suffisants et concluant (cf Travis).
– Conception simple : Comme dit plus haut le fait de découper le projet en petites applications entre dans le concept de l'Extreme Programming. De plus, Django, sans l'imposer facilite grandement cette tâche.
– Refactoring : Là encore, une livraison ne vaut que si le code a été revu et commenté.
– Appropriation collective du projet : Facile (ou difficile) à faire quand on est seul...
– Convention de nommage : Python, Django et même l'EDI ont tendance à faciliter cela. Un peu d'effort au début et cela devient naturel au fur et à mesure.
– Travail en binôme : Moi et moi-même avons vraiment beaucoup travaillé de concert.

Production

Le serveur de production est un serveur Linux Debian hébergé par Digital-Ocean. Comme expliqué ci-dessus, à chaque push de l'application sur la branche PROD, le serveur DOCKER récupère les sources et régénère un conteneur contenant l'application et le serveur WSGI Gunicorn. Ce conteneur peut soit être automatiquement déployé sur le serveur à la mise à jour automatique (le lundi matin) soit manuellement. Le serveur de pages statiques est un serveur NGINX et le serveur de base de données est un serveur POSTGRESQL. Chaque serveur fait l'objet d'un conteneur DOCKER spécifique (configurations et données dans des volumes Docker permanents).

Difficultés

J'ai dû faire face à quelques difficultés de plusieurs ordres.

Techniques :

- Comment faire telle ou telle chose d'une manière que je trouve intéressante et moderne : Boites de dialogues plutôt que d'avoir une « grosse page », Tableaux triés avec pagination, dragndrop…
- Avoir un rendu sympathique sur grand écran mais aussi sur tablette (le smartphone fonctionne mais c'est tout de même assez moche, car de nombreuses saisies sont à effectuer.).
- DragnDrop qui ne fonctionne pas sur écran tactile –> résolu mais une mauvaise surprise !
- Utilisation d'une fonctionnalité Python 3.8 (opérateur morse :=) que ne fonctionnait pas sur mon serveur de test (sans Docker) car Debian et Python 3.7.

Organisationnels :

- Prendre garde à ne pas entrer dans un effet « Tunnel » sur les fonctionnalités…

Bilan

Le but était pour moi de réaliser un site avec 2 types d'utilisateurs et une gestion assez simple des droits (quelque chose qui se rapproche de ce qui peut exister au sein d'une entreprise).
Je pense avoir pu réaliser une application fonctionnelle et disposant d'une interface UX assez moderne. Par contre, pour tout ce qui est des couleurs et polices de caractères, je suis resté très « standard ».
En effet, dès que cela me paraissait possible, j'ai pris un soin particulier de l'expérience utilisateur : Dialog-boxes, drag'ndrop, curseurs pour les boutons radio…
De nombreuses fonctionnalités pourront être ajoutées ultérieurement et l'organisation du projet permettra ces futurs ajouts facilement (applications de taille réduite…)