P11 – Améliorez un projet existant en Python

Liens :
Application : http://purbeurre.jm-hayons74.fr
Github : https://github.com/jmlm74/P11-Ameliorez-un-projet-existant-en-Python
Trello : https://trello.com/b/8S9Ebfn1/p11-modifiez

Objectifs et contraintes

L'objectif du P11 était en fait multiple :
- Créer un bug, Casser les tests (suite au bug), puis réparer le bug et les tests
- Ajouter une ou plusieurs fonctionnalités à l'application Pur Beurre en simulant les échanges par mail avec le client.

Le Bug et les tests

Le bug renvoyait une erreur 500 sur le site. Cette erreur était bien traitée par le handler correspondant mais ne renvoyait rien de bien précis. Par contre, en passant les tests, l'erreur apparaissait eu grand jour (ligne 59 du fichier P11_07_test_error.out) → *products_app.models.Category.MultipleObjectsReturned: get() returned more than one Category -- it returned 2!* On voit donc clairement qu'un get dans l'ORM renvoie plus de 1 ligne. Une personne de Pur Beurre avait mis un **get** en lieu et place d'un **filter** pour interroger la base de données. Cette modification annulée, les tests se sont de nouveau déroulés sans accroc.

Les fonctionnalités

La suppression des bookmarks

La première des nouvelles fonctionnalités demandées par Pur Beurre était de pouvoir supprimer des bookmarks préalablement enregistrés. Dans l'application lors de l'enregistrement du bookmark, la petite disquette sous le produit passait du vert au rouge et n'était plus cliquable. Le client souhaitait que la disquette rouge reste cliquable et que le fait de cliquer dessus supprime le bookmark. Cela a été fait en réutilisant le module AJAX existant afin de ne pas alourdir le code plus que nécessaire. La suppression proprement dite se fait via une méthode de classe sur le modèle Bookmarks. Cela permet de rester sur le « même moule » que pour la création.

L'internationalisation

La seconde fonctionnalité demandée par Pur Beurre était de disposer d'un site en anglais. La première idée qui vient est d'utiliser l'internationalisation de DJANGO via les modules I18N. Toutefois cela m'a paru un peu lourd à mettre en place et surtout peu évolutif. Toute modification demandait de régénérer un fichier des traductions (.po) puis de le compiler afin d'obtenir un fichier qui devait se trouver à la bonne place… L'ajout d'une page, ou d'une langue demandait un travail assez important qui ne pouvait être réalisé que par un informaticien. J'ai donc opté pour une solution qui me paraissait plus évolutive et surtout plus facile à maintenir.

Dans un premier temps, j'ai créé un modèle (table)« Translation » comprenant 3 attributs (colonnes) :
- Position : Clé qui comporte ce qui doit être affiché. Je suis parti d'un masque « Application-page-champ » ce qui donne par exemple : home_app-index_text1 qui pointera sur le champ text1 de la page index.html de l'application home_app.
- FR : Le texte en français
- UK : Le texte en anglais

Une méthode de classe permet d'obtenir la traduction anglaise ou française d'une position : *def get_translation(cls, position, langage)*

Ensuite, j'ai créé un template tag « dis_play » qui var récupérer directement les traductions depuis les templates de la façon suivante : *{% dis_play "home_app-home-text4" %}* (affiche la traduction du champ text4 de la page home.html de l'application home_app)

La langue est une variable de session positionnée à « FR » si l'utilisateur n'est pas connecté et qui peut être positionné par 2 petits drapeaux en haut à droite de la navbar (visible sur chaque page). Le fait de cliquer sur l'un ou l'autre met à jour la variable de session et renvoi sur la page d'accueil dans la langue souhaitée.

Une difficulté rencontrée a été de traduire les messages d'erreurs des forms standards de Django. Il a fallu pour cela faire une petite fonction qui supprime l'habillage du message pour le récupérer et le considérer comme une position afin d'obtenir sa traduction.

On voit donc que grâce a cette méthode, l'ajour d'une langue se fait rapidement :
- Ajout attribut (colonne) au modèle et traduction
- Ajout du drapeau et ajout de la variable de session (module AJAX) ex : IT pour Italien

À noter que cette méthode ne traite pas les formats date et heure non utilisés dans Pur Beurre et qui nécessiteraient un second développement.

La gestion utilisateur

La dernière modification demandée par Pur Beurre concernait la gestion utilisateur. La V1 ne permettait que de créer un compte et de se connecter en utilisant la même page (connexion/création) en désactivant le bouton de connexion si l'utilisateur était déjà connecté. Cet ajout de fonctionnalité s'est fait sur plusieurs axes :

– Séparation des pages de connexion et de création d'un compte.
Création de 2 pages spécifiques. Ces 2 pages sont des views basées sur des fonctions qui utilisent un seul et unique form spécifique.

– Modification du mot de passe de l'utilisateur connecté.
La modification du mot de passe utilise elle aussi une view basée sur une fonction. Par contre, le form utilisé est un form standard de Django **:PasswordChangeForm.**

– Réinitialisation du mot de passe
La réinitialisation du mot de passe est encore une fois un view basée sur une fonction. Elle utilise un form standard de Django : **PasswordResetForm.** La fonction permet de construire un mail en utilisant un uid (sorte de codage de l'utilisateur en base 64) et un token généré. La combinaison des 2 forme une URL dont la validité est limitée dans le temps. Cette URL est envoyée par mail à l'utilisateur qui cliquant dessus est renvoyé sur une vue standard de Django : **PasswordResetConfirmView** La confirmation du reset du mot de passe renvoi automatiquement sur une URL pointant elle aussi sur une vue standard de Django :**PasswordResetCompleteView.** À noter que chacune de ces vues pointe sur un template Pur Beurre grâce au paramètre Template_name du fichier URLS.PY

L'envoi des mails se fait via L'API de MAILGUN. Les messages, ainsi que le mail changent suivant la langue en cours pour la session.