EXPLICATIONS TESTS POUR PROJET 8

J'ai créé des tests pour chacune de mes applications, je les ai organisés dans un dossier test. En utilisant le package « coverage », cela m'a permis d'évaluer la couverture test de mon projet. J'ai atteint une couverture de 96%. On nous demandait d'atteindre un minimum de 80%, j'ai donc atteint l'objectif demandé.
Nous allons détailler et expliquer comment j'ai procédé pour faire mes tests pour chaque application.

Test Application Favoris :

Concernant cette application, j'ai testé les vues ainsi que les modèles. Nous avons donc 2 fichiers « test_models.py et test_views.py ».
Pour le test sur mon model, j'ai créé des aliments fictifs pour ma base de données ainsi qu'un utilisateur puis je les ai rajoutés à la base de données favoris puis j'ai vérifié la concordance entre mes données de départ et les données dans la base de données favoris. Ce qui m'a permis d'obtenir une couverture à 100% pour les tests sur mon model favoris.
Pour tester mes vues, j'ai tout d'abord intégré des aliments et un utilisateur à ma base de données fictives. Puis si j'ai testé si l'enregistrement dans notre base de données (via reverse et url) on avait bien un code 302, c'est à dire une redirection vers une autre page une fois la sauvegarde effectuée.
Puis j'ai délibérément créé 4 aliments afin de tester la pagination de notre vue, puisque ma vue affichait 3 résultats par page. La couverture pour le test de ma vue est donc de 100%.

Test Application Users :

En ce qui concerne cette application, j'ai testé le module forms.py ainsi que views.py, nous avons donc 2 modules, test_forms.py et test_views.py
Pour tester forms.py, j'ai tout d'abord testé que tous les champs attendus étaient bien présents dans notre modèle ceci pour le formulaire « signup » et « signin » puis j'ai intégré des données afin de vérifier que les formulaires étaient bien valides. Ces tests m'ont permis d'obtenir une couverture de 95%.
Enfin, pour tester les vues de cette application, j'ai d'abord dans ma méthode « setup » intégré un utilisateur fictif afin de procéder à différents tests. J'ai d'abord testé que la création d'un nouvel utilisateur allait bien nous rediriger vers l'home page de notre site puis que cette même création engendrera la présence d'un utilisateur dans notre base.
J'ai également testé que la demande des 2 url « signin » et « signup » allait bien obtenir une réponse status 200. Ainsi qu'une connexion de l'utilisateur sur la page « signin » ou une déconnection grâce à notre vue « signout » nous retourneront une réponse status 302 soit une redirection vers notre home page également.
Puis finalement, que la réponse status pour l'accès à la page « account » retournera également une réponse status 200.
Tous ces tests m'ont permis d'obtenir une couverture de 100%.

Test Application Products :

Pour l'application Products, j'ai du testé, les mêmes modules que pour les applications précédentes et en plus de cela j'ai fait un « mock » afin de tester l'api de OpenfoodFacts.
Afin de tester le module forms.py, j'ai testé que l'on avait bien les champs déclarés, c'est à dire le champ « search » ce test m'a permis d'obtenir une couverture de 100% pour ce module.
Nous avons ensuite testé le module «import_api.py » nous avons donc crée un mock en s'aidant du décorateur patch en lui précisant quelle fonction nous voulons « mocker », ensuite nous introduisons des données fictives afin d'imiter le comportement de notre fonction qui fait appel l'api de Openfoodfacts, puis nous vérifions que nous avons bien un seul produit (celui que nous avons introduit dans le mock) en faisant appel à notre fonction. Pour cela nous avons une couverture de 100%.
Pour tester le module models.py, nous intégrons des produits à notre base de données et nous vérifions la concordance des informations dans nos requêtes avec les produits que nous avons intégré au préalable. La couverture pour ce module est également de 100%.
Puis finalement, nous testons nos vues en intégrant tout d'abord des aliments à notre base de données dans la méthode « setup » puis nous testons tout d'abord les vue « home » et « search » en vérifiant que les url « home » et « search » retournent bien des status code « success » soit 200.
Puis, nous vérifions qu'une recherche sur notre vue « search » retourne bien un status code 302 soit une redirection vers une autre vue.
Puis grâce à l'intégration de nos aliments au départ, nous pouvons vérifier les vues « lire » et « display » que celles-ci retournent bien un status code 302, c'est à dire une redirection vers les résultats en utilisant les affichants les aliments que nous avons intégré au préalable.
Puis nous testons que la vue « lire » retourne bien un code erreur 404 en rentrant un id inconnu de notre base de données. La couverture pour le module views.py est également de 100%