UQAC
Université du Québec
à Chicoutimi

Architecture des logiciels (6GEI311)

Laboratoire 4

Travail présenté à Jérémy Bouchard

Par

Samuel Harvey
(HARS04119907)

Simon Dumas
(DUMS10020109)

Laboratoire 4 - Rapport

Vendredi18 novembre 2022

Table des matières

Explication des tests

1. Test_can_load_tweets

Ce test est plutôt simple, il sert simplement à voir si le programme est capable de charger les données des tweets trouvées à l'aide de la recherche. Pour y parvenir, nous créons d'abord la requête passée à l'API Twitter et puis nous vérifions que la réponse donnée par la requête contient bien le champ 'data'.

2. Test_delete_tweet_db

Ce test sert à voir si nous sommes capables de supprimer des tweets dans la liste de tweets. Pour y parvenir, nous insérons d'abord un tweet dans notre liste, puis nous le supprimons et puis nous testons afin de voir que la liste est bien nulle.

3. Test_pas_bearer_token

Ce test sert à voir si le programme sera en mesure de gérer l'exception générée lorsque nous ne fournissons pas de « bearer_token ». Pour y parvenir, nous confirmons que la réponse donnée par la fonction de la requête correspond bien à "Expecting value: line 1 column 1 (char 0)".

4. Test_bearer_token_invalide

Ce test sert à vérifier que le programme soit en mesure de gérer ce qui arrive lorsque que le « bearer_token » n'a pas une valeur valide. Pour y parvenir, nous confirmons que la réponse donnée par la fonction de la requête correspond bien à "Expecting value: line 1 column 1 (char 0)".

5. Test_can_save_tweets_in_database

Ce test sert à voir si la sauvegarde d'un tweet dans la base de données peut bel et bien se faire. Afin d'y arriver on crée un faux tweet et on le sauvegarde en base de données puis on vérifie si le tweet est bien égal à celui qui se retrouve dans la base de données.

6. Test_pas_de_header

Ce test sert à voir si le programme est capable de gérer ce qui arrive s'il n'y a aucun « header ». Pour arriver à nos fins, on procède par confirmation que la réponse obtenue suite à la requête correspond à "Expecting value: line 1 column 1 (char 0)" qui est la réponse renvoyée.

7. Test_header_invalide

Ce test sert à vérifier que lorsque l'on fournit un « header » invalide, le programme est en mesure de gérer l'exception. Pour arriver au but, nous inscrivons un header invalide et nous vérifions que la réponse obtenue est égale à "Expecting value: line 1 column 1 (char 0)".

8. Test_url_non_valide

Ce test sert à voir si le programme sait gérer une url qui est considéré comme étant invalide. Nous procédons par reconnaissance de la réponse obtenue pour voir si nous obtenons une réponse comprenant le string "Invalid URL".

9. Test_query_a

Ce test permet de vérifier que l'erreur de la recherche de la lettre "a" dans la query est géré par le programme. Nous optons par vérification de la réponse encore une fois pour confirmer que la réponse obtenue est bien équivalente à "There were errors processing your request: Rules must contain at least one positive, non-stopword clause (at position 1)".

10. Test_aucun_resultat

Le test permet de voir si tout est correct lorsqu'il n'y aura aucun résultat trouvé pour le query. Nous avons décidé de s'y prendre en comparant la réponse obtenue avec "'result_count': 0" et comme les deux éléments de comparaison sont identiques, le test passe très bien.

11. Test_query_vide

Ce test sert à déterminer si le programme répondra de la bonne manière lorsque la boîte de « query » sera laissée vide. Sans ne rien changer dans le programme, lorsque nous effectuions une requête vide dans la boîte « query », une erreur apparaissait à l'écran disant que nous ne recevions pas de réponse. En interceptant l'erreur, nous pouvions alors diriger le programme vers la page Display.html et nous pouvions alors afficher un « string » mentionnant l'erreur. Si le tout marche tel que prévu, l'erreur que le test devrait recevoir devrait contenir "Invalid 'query':''. 'query' must be a non-empty string" à json_response['errors'][0]['message'] et c'est ce qui sera intercepté et affiché dans la page Display.html dans le cas de cette erreur.

12. Test_aucun_params

Ce test sert à déterminer si le programme est en mesure de répondre correctement dans le cas où aucun paramètre n'est entré. Ce test compare la valeur donnée en sortie par la fonction query_twitter_api afin de la comparée avec la valeur attendue qui devrait comprendre « query parameter can not be empty ».

13. Test_expansions_params_invalides
14. Test_tweetfield_params_invalides
15. Test_userfield_params_invalides
16. Test_placefield_params_invalides

Les tests 13,14,15 et 16 sont vraiment similaires et ne seront donc pas expliqués un par un.

Essentiellement, chacun de ses tests entrent, à tour de rôle, un paramètre invalide dans params. Nous pouvons donc ainsi tester pour voir si la valeur retournée par la fonction query_twitter_api indique bien une erreur liée au paramètre invalide fourni. Donc, les valeurs de retour de query_twitter_api devront comprendre un premier « string » lié au paramètre (ex. « expansions ») ainsi que le « string » « is not one of » qui est propre à l'erreur de paramètre invalide.

17. Test_maxresults_params_invalides

Ce test sert à vérifier si le comportement du programme sera le bon dans le cas où nous entrons une valeur invalide au champ « max_results » des paramètres du « query ». Dans ce cas, l'erreur retournée et gérée par le programme devrait contenir les « strings » « max_results » et « is not a valid Int ».

18. Test_maxresultszero_params_invalides

Ce test sert à vérifier si le comportement du programme sera le bon dans le cas où nous entrons un nombre invalide dans le champ « max_results » des paramètres du « query ». Ce test envoie donc la valeur de 0 dans ce champ et test le résultat obtenu afin de voir si celui-ci contient les « strings » suivants : « max_results » et « is not between 10 and 100 ».

19. Test_nexttoken_params_invalides

Ce test sert à vérifier le comportement du programme lorsque le champ « next_token » contient une valeur invalide. Dans ce cas, l'erreur soulevée devrait contenir les « strings » « next_token » et « is not a valid token ».

20. Test_caractere_invalide

Ce test sert à voir se qui se produit lorsqu'un caractère invalide est entré. Ces caractères invalides sont constitués de valeurs telles que « + », « = » ou « < ». L'erreur générée ici devrait contenir les « strings » « There were errors processing your request » et « no viable alternative at character ».

21. Test_query_trop_long

Ce test sert à tester le programme pour voir si celui-ci est en mesure de gérer de très longs « query ». En entrant un très lon query, l'erreur que nous devrions obtenir devrait contenir « Rule length exceeds the max allowable. ».

Modifications effectuées au programme

Afin de gérer les erreurs identifiées, plusieurs mécanismes ont été mis en place. Tout d'abord, nous avons dû ajouter une conversion de type à la variable « headers » qui devait être de type « dict ». Cela nous a permis de tester des valeurs « strings » à la place de valeurs de type « dict », pouvant ainsi générer des erreurs sans faire « planter » le programme. Ensuite, un bon nombre de print() a été ajouter afin de voir les erreurs sur le terminal. Ces « prints » ont été gardés puisqu'ils peuvent toujours être intéressant et sont invisibles au niveau de la page web. Après cela, une très grande partie des erreurs a été retiré grâce au simple fait de transmettre le « string » de l'erreur vers la page Display.html. Cela permet à l'utilisateur de voir qu'il y a une erreur sans que la page ne plante. En utilisant cette page, nous avons aussi directement un lien vers la page de départ, soit la page Search.html, ce qui est plutôt pratique. Notre programme est aussi capable de gérer le fait que l'url présent à droite de « localhost :PORT » soit invalide et ne dirige vers rien de base. Dans ce cas, nous dirigeons le programme vers la page Display.html avec une erreur disant simplement « erreur url… » avec encore et toujours le bouton « Go back » pour nous retourner vers la page Search.html et ainsi remettre l'url de la page à une valeur valide. Finalement, nous avons ajouté un bouton servant simplement à voir tout le contenu de la liste de tweets présents dans la « base de données ».