Settaggio ed esecuzione di test

Organizzazione e scelte implementative

I test di progetto sono essenzialmente divisi in due macro categorie: test di **funzioni specifiche** tramite test di unità e test **django**, usando test di integrazione e di accettazione.
Nella prima categoria rientrano tutti i test strutturati per assicurarsi che le **funzioni di calcolo** e processazione di dati, ad esempio la funzione di calcolo frequenza parole, siano corrette.
Nella seconda categoria invece rientrano test più **specifici** del framework django come ad esempio controlli di correttezza di risoluzione di **url**, controlli di **views** e **models**, la corretta interazione dei diversi moduli tra loro e il corretto funzionamento dell'applicativo riguardo ai requisiti espressi dal cliente.
Nella struttura gerarchica del progetto Django i test sono stati scritti e **inseriti in directory specifiche** chiamate *"tests"* (**una per ogni app**) all'interno della quale si trovano entrambe le categorie specificate in precedenza, ed ogni test deve iniziare con il nome "*test*".
Per il primo tipo di test ci siamo affidati alla libreria '**unittest**', preferendola a '*pytest*' in quanto più semplicistica e direttamente integrata nel linguaggio python.

Scopo

I test sono stati pensati per effettuare un controllo generale di effettiva correttezza del codice scritto ma **senza scendere troppo in profondità** per evitare il rischio di scriverli più per scopo didattico che di effettiva utilità.
Il nostro obiettivo attraverso i test era quello di avere un **rapido modo di controllare** il funzionamente globale dell'applicazione senza perdere tempo ad eseguire check di correttezza su piccole routine esecutive presenti nelle funzioni scritte che possono essere tranquillamente controllate manualmente da chi effettivamente le sta scrivendo.
Invece per il secondo tipo è stata utilizzata la **libreria** dedicata ai test **di django**.
E' presente una **copertura quasi totale** del codice, grazie ai vari test scritti ed effettuati, che aiuteranno gli sviluppatori ad accorgersi di eventuali errori/**bug** nel caso di **modifiche** a **codice scritto in precedenza**.

Configurazione ed esecuzione

Da IDE:

- Per eseguire i test '**unittest**' è necessario eseguire il codice con una variabile di ambiente specifica → DJANGO_SETTINGS_MODULE=cruscanalyzer.settings.
  Per integrare questa direttiva direttamente in **PyCharm** andare in run→ edit configurations→ python tests ed inserire la stringa all'interno del campo '*variabili d'ambiente*'. Questa stringa indica semplicemente all'interprete dove si trova il file **settings.py** del progetto cruscanalyser.
- Per eseguire i restanti test “django” è sufficiente lanciare la porzione di codice dal file corrispondente senza ulteriori settaggi.

Da linea di comando:

- Per lanciare indifferentemente entrambe le categorie di test posizionarsi nella cartella base del progetto, eseguire '**pipenv shell**' per entrare nell'ambiente virtuale del progetto (cruscanalyzer_base_folder) e a seconda del caso d'uso digitare:

    - **'python manage.py test'**
      → *esecuzione di tutti i test esistenti*

    - '**python manage.py test <nome_app_da_testare> '**
      → *esecuzione test app specifica*

      esempio : per lanciare i test associati all'app 'text_management' digitare:
      '**python manage.py test text_management’**

    - '**python manage.py test <path.del.test>** '
      → *esecuzione test specifico*

      esempio : per lanciare i test di verifica indirizzamento degli urls ('test_urls.py') dell'app 'text_management' digitare :

      'python manage.py test
      text_management.tests.text_management.test_urls'
      <comando> *<path del test>*

Entrambe le tipologie a fine esecuzione informeranno sia a linea di comando che tramite notifiche integrate dall'ide di sviluppo il risultato dei test fatti a prescindere dall'esito.