==========

DOKUMENTACJA TESTÓW I PRZEGLĄD KODU

==========

1. CEL I ZAKRES TESTÓW

Projekt zawiera aplikację Django "flashcards", która umożliwia:
- Tworzenie, edycję i usuwanie fiszek (CRUD).
- Generowanie fiszek z tekstu (z pomocą NLP) i z obrazów (OCR).
- Przeglądanie fiszek, pogrupowanych w kategorie..
- Tryb nauki (wybór fiszki w oparciu o statystyki).

Testy mają na celu:
- Zweryfikowanie, czy każda z kluczowych funkcjonalności działa prawidłowo w różnych scenariuszach (w tym wyjątkach i błędach).
- Wczesne wykrycie regresji lub problemów w logice serwisów (NLP, OCR, Facade), w formularzach i widokach.

Stosowane są testy:
- Jednostkowe (unit tests) – np. testy poszczególnych serwisów, formularzy.
- Integracyjne – np. testy łączące OCR i NLP przez warstwę "FlashcardFacade".
- Czarnoskrzynkowe (black-box) – wysyłające żądania HTTP do aplikacji, sprawdzające od zewnątrz kody odpowiedzi, użyte szablony, komunikaty itp.

2. STRUKTURA TESTÓW

W katalogu `flashcards/tests/` znajdują się następujące pliki,
które pokrywają różne warstwy i funkcjonalności aplikacji:

1. **test_blackbox.py**
   - Zawiera testy **czarnoskrzynkowe** (black-box) interfejsu HTTP, m.in. dodawanie fiszek, wgrywanie plików, panel admina i usuwanie fiszek.
   - Metody testowe w klasie `BlackBoxBasicFlowTest` symulują najpopularniejsze przepływy użytkownika:
     - `test_add_flashcard_valid_data` – dodanie fiszki z poprawnymi danymi
     - `test_add_flashcard_invalid_data` – brak wymaganych pól w formularzu
     - `test_upload_photo_invalid_file_type` – przesłanie pliku, który nie jest obrazem
     - `test_learn_flashcards_nonexistent_category` – nauka w nieistniejącej kategorii
     - i inne (szczegółowy opis w komentarzach metod).

2. **test_decorators.py**
   - Testy jednostkowe dekoratora `log_flashcard_action`, który loguje
     rozpoczęcie i zakończenie akcji.
   - Sprawdzamy, czy dla wywołania metody (np. delete) w panelu admina
     pojawiają się w logu komunikaty "Started action" / "Finished action".

3. **test_forms.py**
   - Testy **jednostkowe** formularza Django (`FlashcardForm`).
   - Weryfikacja walidacji pól `question`, `answer`, obowiązkowej kategorii itd.

4. **test_services_facade.py**
   - Testy **integracyjne** klasy `FlashcardFacade`, łączącej OCR i NLP.
   - Oczekujemy, że `OCRProcessor.extract_text()` i `NLPProcessor.analyze_text()`
     poprawnie stworzą listę obiektów `Flashcard`.

5. **test_services_group.py**
   - Testy serwisu `GroupService` (m.in. generowanie listy kategorii,
     obsługa kategorii "General" dla pustego stringa itp.).
   - Są to testy **jednostkowe**, ponieważ obejmują bezpośrednio logikę
     pobierania danych z modelu `Flashcard`.

6. **test_services_learning.py**
   - Testowanie serwisu `LearningService`, który wybiera następną fiszkę
     w trybie nauki na podstawie statystyk poprawnych/niepoprawnych odpowiedzi.
   - Sprawdza się tutaj losowanie ważone przy pomocy `random.uniform`.

7. **test_services_nlp.py**
   - Testy **mock** dla połączenia z OpenAI (klasa `NLPProcessor`).
   - Weryfikujemy m.in. działanie w sytuacji błędu API (`API Error`) i
     poprawny parsing tekstu w formacie "question: ... answer: ... ---".

8. **test_services_ocr.py**
   - Testy działania OCR z użyciem biblioteki `pytesseract` (klasa `OCRProcessor`).
   - Sprawdzamy prawidłowość zwracanej zawartości przy różnych wyjątkach,
     np. `FileNotFoundError` czy `IntegrityError` podczas zapisu w bazie.

9. **test_services_renderers.py**
   - Testy jednostkowe rendererów: `TextRenderer` (formatowanie tekstowe)
     i `AudioRenderer` (generowanie pliku MP3 przy pomocy `gTTS`).

10. **test_urls.py**
    - Testy **czarnoskrzynkowe** sprawdzające, czy poszczególne ścieżki (URL)
    kierują do właściwych klas widoków (np. `/add/` do `AddFlashcardView`).

11. **test_views_* (pozostałe pliki)**
    - Zestaw testów **integracyjnych** sprawdzających konkretne widoki Django.
    - Każdy plik dotyczy innego aspektu:
    - `test_views_add.py` – widok dodawania fiszki.
    - `test_views_edit.py` – widok edycji fiszki.
    - `test_views_delete.py` – widok usuwania.
    - `test_views_list.py` – lista fiszek z paginacją.
    - `test_views_learn.py` – tryb nauki (wybór kolejnych fiszek).
    - `test_views_ocr.py` – przetwarzanie obrazu i generowanie fiszek OCR.
    - `test_views_save.py` – zapisywanie fiszek z pamięci sesji do bazy.
    - `test_views_generate_from_text.py` – obsługa generowania fiszek z tekstu.

Wszystkie te pliki łącznie dają pokrycie różnymi testami jednostkowymi, integracyjnymi i czarnoskrzynkowymi,
odzwierciedlając główne elementy logiki aplikacji i widoków.

3. SPOSÓB URUCHAMIANIA TESTÓW

Testy można uruchomić z poziomu konsoli, stojąc w katalogu głównym, gdzie znajduje się "manage.py":

python manage.py test flashcards

Django automatycznie wykryje wszystkie pliki zaczynające się od "test_" w katalogu "flashcards/tests/".

4. PRZEGLĄD KODU I NAJWAŻNIEJSZE ELEMENTY

- flashcards/decorators.py
  * "log_flashcard_action" – dekorator logujący początek i koniec określonej akcji, np. usunięcie fiszki w panelu admina.

- flashcards/forms.py
  * "FlashcardForm" – formularz Django używany w widokach do tworzenia/edycji fiszki. Wymaga podania pola "question" i "answer", a także wybrania lub stworzenia kategorii.

- flashcards/models.py
  * "Flashcard" – główny model, zawiera pytanie, odpowiedź, kategorię, statystyki poprawnych/niepoprawnych odpowiedzi, itp.
  * "update_statistics" – aktualizuje licznik correct_answers/incorrect_answers.

- flashcards/services/*
  * "flashcard_facade.py" – łączy OCR z NLP (klasa "FlashcardFacade").
  * "group_service.py" – logika dzielenia fiszek na kategorie, w tym kategoria "General".
  * "learning_service.py" – wybiera następną fiszkę do nauki (losowanie ważone).
  * "nlp.py" – "NLPProcessor" woła API OpenAI (z użyciem "openai.chat.completions") do generowania fiszek.
  * "ocr.py" – "OCRProcessor" używa "pytesseract" do rozpoznawania tekstu na obrazie.
  * "renderers.py" – generowanie widoku fiszki w formie tekstu ("TextRenderer") lub audio ("AudioRenderer" przez gTTS).

- flashcards/views/flashcards_views.py
  * Zawiera klasy widoków Django (CreateView, UpdateView, TemplateView itp.):
    - "AddFlashcardView", "EditFlashcardView", "DeleteFlashcardView" (operacje na modelu).
    - "FlashcardListView" (lista z paginacją).
    - "DisplayFlashcardView" + "FlashcardAudioView" (wyświetlanie fiszki w formie tekstu/audio).
    - "LearnFlashcardsView" (tryb nauki).
    - "UploadPhotoView", "OCRResultsView" (wysyłanie obrazu do OCR).
    - "GenerateFromTextView" (obsługa NLP).
    - "SaveFlashcardsView" (zapisywanie fiszek z sesji w bazie).

Testy w "test_views_*.py" sprawdzają typowe scenariusze użycia widoków (np. puste pole, poprawne dane, brak pliku, itp.).

5. ZAKRES I PRZYKŁADOWE SCENARIUSZE

W testach sprawdzane są następujące, najważniejsze scenariusze i funkcjonalności:

1) **Dodawanie Nowej Fiszki (CRUD)**
  - Zarówno poprawne dane (tzw. happy path),
    jak i brak wymaganych pól (pole question/answer puste).
  - Obsługa nowej kategorii i wyboru istniejącej.

2) **Wyświetlanie Listy Fiszek (z Paginacją)**
  - Dzielimy kategorie (5 na stronę) oraz fiszki w obrębie kategorii (5 na stronę).
  - Testujemy różne kombinacje parametrów GET, np. `page_categories=2`,
    `page_flashcards_Category_0=3`.

3) **Generowanie Fiszek z Tekstu (NLP)**
  - Użytkownik wprowadza tekst, wywoływana jest analiza (`NLPProcessor`),
      a wynik wyświetlany na stronie "flashcards/generate_from_text.html".
  - Sprawdzany brak tekstu (pusty string) i ewentualny błąd "NLP Failure".

4) **Wgrywanie Obrazu (OCR)**
  - Poprawny plik obrazu (zwraca listę wykrytych fiszek).
  - Przesłanie pliku niebędącego obrazem (`text/plain`).
  - Brak pliku w polu `FILES`.
  - Obsługa wyjątków (np. `Exception("OCR Failure")`).

5) **Tryb Nauki**
  - Użytkownik odwiedza `/learn/<category>/` i otrzymuje losowo wybraną fiszkę.
  - Testy sprawdzają brak fiszek w danej kategorii (ostrzeżenie i redirect),
      zgłaszanie odpowiedzi poprawnej/niepoprawnej (inkrementacja statystyk).

6) **Usuwanie Fiszki**
  - Próba usunięcia istniejącej fiszki (potwierdzenie w `confirm_delete.html`).
  - Usuwanie nieistniejącej fiszki (kod 404).

7) **Obsługa Panelu Admina**
  - Logowanie jako superuser, wyświetlenie listy fiszek w `/admin/`.
  - Usuwanie poprzez "delete_selected" w panelu admina, z dekoratorem logującym akcję.

8) **Renderowanie Audio/Tekstu**
  - Testy `AudioRenderer` i `TextRenderer` sprawdzają generowany output
      (np. plik dźwiękowy przez `gTTS`, formatowanie "Question: ...\nAnswer: ...").

9) **Routing i URL**
  - Poprzez plik `test_urls.py` sprawdzamy, czy np. `"/add/"` rozwiązuje się
      na `AddFlashcardView`, `"/delete/<pk>/"` na `DeleteFlashcardView`, itd.

Powyższe scenariusze pokrywają kluczowe funkcjonalności,
zapewniając pewność, że najważniejsze ścieżki w aplikacji Django działają
poprawnie i reagują w oczekiwany sposób (odpowiednie kody HTTP, szablony,
komunikaty, stan bazy).

6. UWAGI KOŃCOWE

- Wszystkie testy można uruchomić poprzez:
  python manage.py test flashcards

- W razie używania "pytest", można skonfigurować "pytest-django" i wywołać:
  pytest

- Pliki testowe są podzielone według funkcjonalności (views, forms, services),
  co ułatwia nawigację i utrzymanie. W razie problemów lub nowej logiki
  dodajemy nowe testy w odpowiadających plikach.

- Dla dokładnych statystyk pokrycia warto użyć "coverage.py":
  coverage run manage.py test flashcards
  coverage report -m

- Wszelkie testowe pliki HTML/szablony znajdują się w "flashcards/templates/flashcards/"
  i można w nich wstawiać debugowe wypisywanie błędów formularza.