Testdokument

Überblick

Das Ziel der Teststrategie ist es, sicherzustellen, dass alle Kernfunktionen der Anwendung – vom Datenmodell über die Logik der Views bis hin zur Darstellung der statischen Ressourcen – erwartungsgemäß arbeiten. Dabei wird auf die in Django integrierte Testumgebung gesetzt, um eine automatisierte, reproduzierbare und unabhängige Testausführung zu gewährleisten.

Testebenen

1. Unit-Tests
    - *Fokus:* Einzelne Komponenten (z. B. Model-Methoden und spezifische Logik innerhalb der Views) werden isoliert getestet.
    - *Beispiel:* Überprüfung, dass die __str__-Methode des Movie-Modells den korrekten Titel zurückgibt.
    - *Vorteil:* Schnelle Rückmeldung über das korrekte Funktionieren der kleinsten Bausteine der Anwendung.
2. Integrationstests
    - *Fokus:* Zusammenspiel mehrerer Komponenten. Beispielsweise wird getestet, ob bei einem POST-Request an die Startseite die korrekten Filme gefiltert und in der richtigen Template-Struktur dargestellt werden.
    - *Beispiel:* Überprüfung der Filterfunktionalität, bei der nur Filme eines bestimmten Genres angezeigt werden.
    - *Vorteil:* Sicherstellung, dass die Komponenten nahtlos zusammenarbeiten.
3. Tests von statischen Ressourcen
    - *Fokus:* Überprüfung, ob statische Dateien (wie CSS) korrekt vorhanden und eingebunden sind.
    - *Beispiel:* Überprüfung, ob der Pfad zur CSS-Datei existiert (mittels Patching von os.path.exists), und die entsprechende Rückmeldung durch die View erfolgt.
    - *Vorteil:* Erkennung von Problemen bei der Bereitstellung statischer Inhalte.

Testumgebung und Vorgehensweise

- **Isolierte Testdatenbank:**
  Jeder Test läuft in einer temporären Datenbank, die automatisch erstellt und nach Abschluss der Tests wieder gelöscht wird. So wird sichergestellt, dass Tests unabhängig voneinander ausgeführt werden und keine Daten persistieren.
- **Testclient:**
  Der Django-Testclient simuliert HTTP-Anfragen (GET und POST) an die Anwendung. Dadurch kann das Verhalten der Views in einer realitätsnahen Umgebung überprüft werden.
- **Mocking:**
  Mithilfe von unittest.mock.patch werden externe Abhängigkeiten (z. B. Dateisystemaufrufe) isoliert, sodass Tests deterministisch und ohne externe Einflüsse ablaufen.
- **Klar definierte Testfälle:**
  Jeder Testfall besitzt eine eindeutige ID, eine klare Beschreibung der Vorbedingungen, der auszuführenden Aktion, sowie der erwarteten und tatsächlich beobachteten Ergebnisse. Dies erleichtert die Nachvollziehbarkeit und spätere Erweiterung oder Anpassung der Testfälle.

Testziele

- **Korrekte Darstellung und Funktionalität:**
  Sicherstellen, dass die Hauptansicht die richtigen Inhalte (wie den Titel „Filmempfehlungen“ und gefilterte Filmauflistungen) anzeigt.
- **Validierung der Geschäftslogik:**
  Überprüfung, dass Filterungen (z. B. nach Genre) korrekt funktionieren und keine unerwarteten Filme erscheinen.
- **Fehlerprävention bei statischen Ressourcen:**
  Überprüfung, ob die CSS-Datei vorhanden ist und entsprechend eingebunden wird, um Darstellungsfehler zu vermeiden.
- **Regressionssicherheit:**
  Durch die automatisierte Testausführung (z. B. über Continuous Integration) wird gewährleistet, dass zukünftige Codeänderungen nicht unbeabsichtigt bestehende Funktionen beeinträchtigen.

Werkzeuge

- **Django Testframework:**
  Bietet eine robuste Grundlage für Unit- und Integrationstests.
- **unittest.mock:**
  Ermöglicht das gezielte Simulieren und Überprüfen von Abhängigkeiten und Umgebungsvariablen.
- **Django Test Client:**
  Simuliert HTTP-Anfragen und -Antworten, um die Benutzerinteraktion nachzustellen.

Zusammenfassung

Die Teststrategie kombiniert Unit- und Integrationstests, um alle wesentlichen Bereiche der Anwendung abzudecken. Dabei werden durch den Einsatz isolierter Testdatenbanken, gezieltes Mocking und die Verwendung des Django-Testclients sichergestellt, dass die Anwendung stabil, fehlerfrei und erweiterbar bleibt. Jeder Testfall ist klar dokumentiert und stellt sicher, dass sowohl funktionale als auch nicht-funktionale Anforderungen erfüllt werden.

Diese strukturierte Herangehensweise bietet eine solide Grundlage für die Qualitätssicherung der Filmempfehlungen-Anwendung und hilft dabei, mögliche Fehler frühzeitig zu erkennen und zu beheben.

Testfallkatalog

Testfall-ID: TC001
Kurzbeschreibung: Überprüfung der __str__-Methode des Movie-Modells
Vorbedingungen:

- Django-Testumgebung ist eingerichtet.
- Alle Migrationen wurden ausgeführt und das Model ist aktuell.

Eingabedaten / Aktion:

- Erstellen eines neuen Movie-Objekts mit z. B. dem Titel „Test Movie“.

Erwartetes Ergebnis:

- Die Methode __str__ gibt den Titel „Test Movie“ zurück.

Tatsächliches Ergebnis:

- (Wird beim Testlauf dokumentiert.)

Testfall-ID: TC002
Kurzbeschreibung: GET-Request an die Startseite (Index.html)
Vorbedingungen:

- Die URL-Konfiguration enthält einen Pfad (z. B. "") mit der richtigen View.
- Django-Testclient ist verfügbar.

Eingabedaten / Aktion:

- Absenden eines GET-Requests an die Startseiten-URL (z. B. /).

Erwartetes Ergebnis:

- Statuscode 200 wird zurückgegeben.
- Die Seite enthält den Text „Filmempfehlungen“.

Tatsächliches Ergebnis:

- (Wird beim Testlauf protokolliert.)

Testfall-ID: TC003
Kurzbeschreibung: POST-Request mit Genre-Filter (Action)
Vorbedingungen:

- Die Testdatenbank enthält Filme unterschiedlicher Genres (mindestens ein Film im Genre „Action“ und einer in „Drama“).

Eingabedaten / Aktion:

- Senden eines POST-Requests an die Startseite mit dem Parameter genre = „Action“.

Erwartetes Ergebnis:

- Es werden ausschließlich Filme des Genres „Action“ angezeigt.
- Filme anderer Genres (z. B. Drama) dürfen nicht in der Ausgabe erscheinen.

Tatsächliches Ergebnis:

- (Wird während des Testlaufs erfasst.)

Testfall-ID: TC004
Kurzbeschreibung: Überprüfung der statischen Datei-Funktion (CSS gefunden)
Vorbedingungen:

- Die CSS-Datei ist im vorgesehenen Verzeichnis vorhanden.
- In der Testumgebung wird os.path.exists mittels Patch simuliert, sodass True zurückgegeben wird.

Eingabedaten / Aktion:

- Absenden eines GET-Requests an den Endpunkt /test_static/.

Erwartetes Ergebnis:

- Die Antwort enthält den Text „Gefunden:“ zusammen mit dem korrekten Dateipfad und der Statuscode 200 wird zurückgegeben.

Tatsächliches Ergebnis:

- (Wird beim Testlauf dokumentiert.)

Testfall-ID: TC005
Kurzbeschreibung: Überprüfung der statischen Datei-Funktion (CSS nicht gefunden)
Vorbedingungen:

- Es wird simuliert, dass die CSS-Datei nicht existiert (Patch von os.path.exists gibt False zurück).

Eingabedaten / Aktion:

- Senden eines GET-Requests an den Endpunkt /test_static/ unter Verwendung des Patches, der False zurückgibt.

Erwartetes Ergebnis:

- Die Antwort enthält den Hinweis „Datei nicht gefunden“ und liefert den Statuscode 200.

Tatsächliches Ergebnis:

- (Wird während der Testausführung aufgezeichnet.)