Wprowadzenie do testów

Testowanie projektu ma na celu sprawdzić:

1. Spełnienie wymagań funkcjonalnych.

2. Spełnienie wymagań niefunkcjonalnych.

3. Czy istnieją elementy, które mogą skutkować nieoczekiwany wynik.

4. Czy projekt jest odporny na dane poza założeniami.

5. Czy projekt nie zawiera ukrytych błędów.

Testowany obiekt

Obiektem testowym jest aplikacja do kupowania biletów do kina napisana w języku *python* z użyciem bazy danych *SQL*.

Plan testów

Proces testowania aplikacji będzie podzielony na kilka etapów. Rozpoczniemy od testów automatycznych, które skupią się na przetestowaniu najmniejszych części kodu. Następnie skoncentrujemy się na testach systemowych, które będą kluczowe i najważniejsze, ponieważ sprawdzą one funkcjonowanie systemu jako całości oraz pokryją jak najwięcej funkcji. Testy systemowe obejmą wszystkie warunki testowe i uwzględnią różne scenariusze użycia.

Proces testowania będzie kontynuowany przez cały czas trwania projektu. Wszystkie awarie i błędy będą zgłaszane na bieżąco osobom odpowiadającym za implementację odpowiednich części projektu. Każdy problem będzie zgłoszony w formie raportu, gdzie zostanie szczególnie opisany błąd oraz oczekiwany wynik wraz z zagrożeniami, do których ten błąd może prowadzić.

Za testowanie aplikacji będzie odpowiedzialny tester projektu oraz programiści, którzy będą wstępnie weryfikować poprawność zaimplementowanych funkcjonalności.

Narzędzia do testowania

Głównym narzędziem jest framework *pytest*, który posiada różne narzędzia do testowania aplikacji webowych oraz docker do przygotowania różnych danych wejściowych dla testów.

Opis testów automatycznych

test_addMovie.py

test_addMovieActors.py

test_addMovieDetails.py

test_addMovieScreening.py

test_bookTicket.py

test_raport.py

Każdy test jest przeznaczony do sprawdzenia funkcji działających bezpośrednio z bazą danych. Testy głównie sprawdzają poprawność integracji oraz przetwarzania danych w bazie danych. Przed rozpoczęciem testowania uruchamiamy obraz bazy danych za pomocą środowiska wirtualizacji docker, gdzie skrypt wpisuje odpowiedni dane dla testow. Takie podejście umożliwia szybkie przygotowanie dowolnych danych testowych oraz zapewnia izolację każdego testu dla uniknięcia błędów testowych. Przypadki testowe sprawdzają przykładowe dane, które doprowadzają do uzyskania oczekiwanych poprawnych oraz niepoprawnych wyników. Każdy test powinien zakończyć się powodzeniem lub zrozumiałym komunikatem o nie akceptacji.

Testy systemowe

Głównym celem testów systemowych jest sprawdzenie całej aplikacji ze strony użytkownika oraz rozpatrzyć nieoczekiwane przypadki użycia.

1. Sprawdzenie poprawności działania nawigacji pomiędzy oknami w aplikacji
    a. Czy kliknięcie na odpowiednią kartę powoduje wyświetlenie właściwej zawartości.
    b. Czy odpowiednie okna otwierają się i zamykają w odpowiednich momentach.
    c. Czy każdy przycisk działa lub nie działa.
    d. Czy listy wyświetlają odpowiednią zawartość.
2. Testowanie każdej funkcji wyświetlającej dane
    a. Poprawność dodawania danych.
    b. Poprawność wyświetlenia danych.
    c. Poprawność dodawania uzupełnień.
    d. Poprawność zachowania danych.
3. Testy integracyjne
    a. Testowanie współdziałania różnych funkcji.
    b. Sprawdzenie możliwych defektów pomiędzy funkcjami.
    c. Sprawdzenie dużej liczby przypadków użycia.
4. Testowanie wymagań niefunkcyjnych
    a. Sprawdzenie otwartości na różnych urządzeniach.
    b. Symulacja awarii oraz wykorzystanie kopii zapasowych do odtwarzania danych.
    c. Zwiększenie liczby obiektów w aplikacji dla testowania skalowalności.

Najważniejsze odnalezione błędy

1. Możliwość wpisania niepoprawnych danych np. ujemną cenę za bilet lub ocenę filmu poza przedziałem.
2. Możliwość robienia duplikatów premier filmowych w tym samym dniu.
3. Problem z uruchomieniem aplikacji na Windows.
4. Problem ze zrozumieniem w jakiej formie trzeba wpisać dane ze strony użytkownika oraz administratora.

Wszystkie odnalezione błędy zostały naprawione oraz ponownie przetestowane

Końcowy raport

Wszystkie zaimplementowane testy skutecznie sprawdziły poprawność działania aplikacji. Dzięki nim mamy pewność w spełnieniu wymagań projektowych. Testy systemowe oraz automatyczne miały duże znaczenie podczas tworzenia aplikacji i znacznie skróciły czas lokalizacji błędów. Znaleziona liczba błędów jest oczekiwana co wskazuje na duży poziom ufności aplikacji. Większość testów została zaimplementowana zgodnie z planem, chociaż było spędzono więcej czasu na sprawdzanie logiki biznesowej, niż było zaplanowane.