Narzędzie wspierające tworzenie testów dla aplikacji webowych

Łukasz Kowalewski
Promotor: dr inż. Marcin Adamski

Spis treści

Rozdział 1

Wstęp

W niniejszym wstępie postaram się uzasadnić istotność tematu pracy inżynierskiej. Tytuł pracy "Narzędzie wspierające tworzenie testów dla aplikacji webowych"opisuje funkcję, jaką stworzony w ramach pracy inżynierskiej program ma za zadanie spełnić. Jego rolą jest dostarczenie użytkownikowi narzędzia, które pozwoliłoby mu dostarczać oprogramowania oraz testów lepszej jakości w krótszym czasie. Następne rozdziały postarają się omówić problematykę zagadnienia oraz możliwe rozwiązania. By zrozumieć i tym samym znaleźć motywację, by owe zagadnienia analizować, należałoby uzasadnić potrzebę tworzenia takiego rodzaju oprogramowania.

Testy stanowią kluczowy element każdego zaawansowanego oraz dojrzałego systemu informatycznego bądź aplikacji. Jest to fakt uznany i łatwy do wykazania, zwracając uwagę choćby na istnienie specjalnych oddziałów w korporacjach zajmujących się wyłącznie tworzeniem testów, bądź zauważając wysoki popyt na programistów wyspecjalizowanych w tej dziedzinie. To właśnie takie korporacje najczęściej tworzą zaawansowane narzędzia typu omawianego w powyższej pracy inżynierskiej. W tak dużych projektach wszelkie błędy systemu potrafią odbić się na bardzo wysokich stratach finansowych, one właśnie stanowią główne źródło motywacji, by w ogóle te testy tworzyć. Jednocześnie warto zwrócić uwagę na wprost przeciwny obraz sytuacji dla projektów średnich bądź małych rozmiarów. W tego typu przedsięwzięciach często brakuje źródła motywacji m.in. w postaci potencjalnych wysokich strat finansowych, co skutkuje sytuacją, w której element testów jest pomijany ze względu na wysoki koszt czasowy ich tworzenia.

Głównym celem omawianej aplikacji będzie wyjście naprzeciw potrzebom ułatwienia tworzenia w krótkim czasie dużej ilości prostych testów tak, by małe i średnie projekty możliwie niskim kosztem uzyskały wysokie pokrycie testowe.

1.1 Cel i zakres pracy

Celem niniejszej pracy inżynierskiej jest opracowanie narzędzia, które znacząco usprawni proces tworzenia testów automatycznych w projektach webowych. Z tego względu praca skupia się nie tylko na samym wytworzeniu aplikacji, ale również na omówieniu metod i dobrych praktyk towarzyszących procesowi testowania.

Tworzone oprogramowanie ma rozwiązać przede wszystkim następujące problemy:

- Wysoki próg wejścia w automatyzację testów dla początkujących zespołów – narzędzie powinno dostarczyć przyjazne mechanizmy generowania przykładowych skryptów i scenariuszy testowych.

- Długi czas przygotowywania testów w małych i średnich projektach – planuje się zapewnić funkcje, które przyspieszą proces konfiguracji i pisania testów (np. wstępne generowanie kodu testowego).
- Integracja z istniejącymi technologiami – narzędzie będzie wykorzystywać znane biblioteki (takie jak Selenium lub Playwright) oraz umożliwiać łatwe włączenie do popularnych pipeline'ów CI/CD.

Końcowym rezultatem jest działające oprogramowanie, którego zastosowanie pozwoli na efektywne tworzenie testów automatycznych oraz ich integrację z cyklem wytwarzania oprogramowania. Dokumentacja pracy przybliży dodatkowo zarówno aspekty teoretyczne (przegląd frameworków, omawianie strategii testowania), jak i praktyczne (szczegółowe omówienie kodu, przykłady uruchomienia testów i wdrożenia w środowisku CI/CD).

Zakres opracowania obejmuje zatem:

- Analizę dostępnych narzędzi i bibliotek testowych dla aplikacji webowych,
- Zaprojektowanie i implementację modularnego narzędzia generującego testy,
- Prezentację możliwych sposobów dalszego rozwoju aplikacji, w tym plan rozbudowy funkcjonalności i integracji z innymi platformami.

1.2 Struktura pracy

W dalszych rozdziałach zaprezentowano kluczowe elementy związane z projektowaniem i tworzeniem narzędzia do wspierania testów automatycznych:

- **Rozdział 2** omawia przykłady zastosowań testów automatycznych, przedstawiając ich różnorodność (testy jednostkowe, integracyjne oraz end-to-end).
- **Rozdział 3** zawiera przegląd popularnych technologii i bibliotek wspierających testowanie aplikacji webowych (m.in. frameworki testowe, narzędzia typu record-and-play, rozwiązania CI/CD).
- **Rozdział 4** skupia się na różnych strategiach generowania testów, w tym podejściach opartych na rejestrowaniu akcji i modelach systemu.
- **Rozdział 5** przedstawia architekturę i interfejs tworzonej aplikacji, opisując poszczególne moduły i sposób ich komunikacji.
- **Rozdział 6** zawiera omówienie kluczowych elementów kodu źródłowego narzędzia oraz przybliża zastosowane biblioteki.
- **Rozdział 7** prezentuje praktyczne przykłady użycia aplikacji, zarówno w trybie interaktywnym, jak i zintegrowanym w potoku CI/CD.
- **Rozdział 8** stanowi podsumowanie pracy, w którym zawarto wnioski końcowe dotyczące realizacji założeń oraz wskazano możliwe kierunki rozwoju narzędzia.

Rozdział 2

Przykłady zastosowań testów

Współczesne aplikacje webowe – bez względu na ich skalę – wymagają odpowiedniego poziomu kontroli jakości, a testy automatyczne są jednym z najsolidniejszych sposobów osiągnięcia tego celu. W niniejszym rozdziale przedstawiono typowe rodzaje testów stosowane przy rozwoju oprogramowania, ze szczególnym uwzględnieniem aplikacji internetowych. Rozważymy zarówno proste testy jednostkowe, jak i bardziej rozbudowane testy integracyjne czy kompleksowe testy end-to-end (E2E). Każdy z wymienionych poziomów testowania ma swoją specyfikę oraz obszar zastosowań – dzięki temu w bardziej złożonych projektach można zastosować podejście wielopoziomowe, co pozwala uniknąć błędów na różnych etapach.

Testy te pełnią także niebagatelną rolę w procesie ciągłej integracji i dostarczania (CI/CD). Automatyczne uruchamianie zestawów testowych przed wdrożeniem oprogramowania minimalizuje ryzyko wypuszczenia wadliwej wersji aplikacji. W dalszej części rozdziału pokazane zostaną najważniejsze cechy trzech kluczowych kategorii testów, a także krótki komentarz dotyczący korzyści i wyzwań związanych z ich stosowaniem.

2.1 Testy jednostkowe

Testy jednostkowe (ang. *unit tests*) skupiają się na najmniejszych fragmentach oprogramowania, zazwyczaj na pojedynczych funkcjach, metodach lub klasach. Celem takich testów jest sprawdzenie poprawności działania konkretnego komponentu w oderwaniu od reszty systemu. Dzięki temu programiści mogą szybko wykryć regresje w kodzie w momencie wprowadzania zmian lub nowych funkcjonalności.

Charakterystyka i zalety testów jednostkowych

- **Wczesne wykrywanie błędów:** niewielki zakres testowanych fragmentów kodu umożliwia szybką diagnozę przyczyn problemu.
- **Łatwość uruchamiania:** testy te są zwykle szybkie i mało zasobożerne, dzięki czemu mogą być wykonywane nawet przy każdej kompilacji.
- **Wspieranie refaktoryzacji:** jeśli są dobrze napisane, stanowią swego rodzaju „siatkę bezpieczeństwa” – zmiana kodu pociąga za sobą natychmiastową weryfikację, czy nadal działa on tak, jak powinien.

Miejsce w cyklu życia aplikacji

Zazwyczaj testy jednostkowe tworzone są równolegle z implementacją funkcji bądź metod, często w metodyce *Test-Driven Development* (TDD). Nawet jeśli programiści nie stosują formalnie TDD, to i tak przeważnie piszą testy tuż po implementacji kluczowych funkcji. Pozwala to na bieżąco weryfikować stabilność kodu i ogranicza ryzyko dalszych problemów w bardziej złożonych warstwach systemu.

Znaczenie dla jakości oprogramowania

Choć testy jednostkowe nie gwarantują wykrycia wszystkich błędów (szczególnie tych na poziomie integracji czy funkcjonalności kompleksowej), stanowią one fundament rzetelnego procesu testowego. Dzięki nim łatwiej utrzymać wysoką jakość aplikacji w długofalowym rozwoju, ograniczając liczbę niespodziewanych błędów w kluczowych komponentach kodu.

2.2 Testy integracyjne

Testy integracyjne (ang. *integration tests*) służą do weryfikacji współdziałania między różnymi komponentami lub modułami systemu. W odróżnieniu od testów jednostkowych, które koncentrują się na pojedynczych funkcjach, testy integracyjne sprawdzają poprawność przepływu danych oraz komunikacji na wyższym poziomie.

Główny cel i zakres

Podstawowym celem testów integracyjnych jest upewnienie się, że wszystkie części aplikacji współpracują ze sobą w sposób zgodny z założeniami. W projektach webowych może to obejmować m.in. komunikację serwera z bazą danych, przepływ danych między mikrousługami, a także integrację z zewnętrznymi API. Zakres takiego testu jest zatem szerszy niż w przypadku testów jednostkowych, ponieważ obejmuje wiele komponentów, a niekiedy także usługi firm trzecich.

Przykłady zastosowań w aplikacjach webowych

- **Sprawdzenie warstwy API i bazy danych:** testy integracyjne mogą weryfikować, czy zapytania HTTP wysyłane przez front-end trafiają do warstwy serwerowej i są poprawnie przetwarzane, a także czy baza danych zwraca oczekiwane wyniki.
- **Integracja z usługami zewnętrznymi:** w przypadku aplikacji korzystających z serwisów płatności czy map, testy integracyjne pozwalają sprawdzić, czy dane usługi są właściwie wywoływane i czy zwracają poprawne odpowiedzi.
- **Weryfikacja logiki po stronie serwera:** testy mogą obejmować zaszyte w kodzie warunki, reguły biznesowe czy sekwencje operacji, które są wykonywane przy interakcji między kilkoma modułami.

Korzyści i wyzwania

Zaletą testów integracyjnych jest to, że pozwalają wykryć problemy w miejscach, gdzie komponenty łączą się i współpracują. Dzięki temu można uniknąć sytuacji, w której pojedyncze moduły działają poprawnie samodzielnie, ale po zestawieniu razem pojawiają się nieoczekiwane błędy.
Wyzwania to z kolei konieczność skonfigurowania odpowiednich środowisk testowych i danych. Wymaga to często stawiania testowych baz danych lub stubów i mocków dla zewnętrznych API. Przy dużych projektach może być to stosunkowo pracochłonne i kosztowne w utrzymaniu.

2.3 Testy end-to-end (E2E)

Testy end-to-end (ang. *end-to-end tests*) to najbardziej rozbudowany rodzaj testów funkcjonalnych, w których symuluje się rzeczywiste zachowanie użytkownika końcowego (lub interakcję system–system) w całym przekroju aplikacji. Obejmują one wszystkie warstwy oprogramowania, od interfejsu użytkownika po bazę danych i usługi zewnętrzne.

Na czym polega koncepcja testów E2E w aplikacjach webowych

Główną ideą testów E2E jest sprawdzenie, czy cała aplikacja działa poprawnie, gdy rozpatrujemy ją jako kompletny produkt. Taki test może obejmować np.:

1. Uruchomienie przeglądarki i wejście na stronę logowania.
2. Wprowadzenie danych uwierzytelniających i przejście do właściwej części aplikacji.
3. Skorzystanie z określonych funkcji (np. wyszukiwanie produktu, dodawanie do koszyka, finalizacja zamówienia).
4. Weryfikację rezultatów w bazie danych czy też w komunikatach wyświetlanych użytkownikowi.

Cały przepływ jest obserwowany oraz weryfikowany pod kątem spełnienia wymagań biznesowych i oczekiwań użytkownika.

Kiedy i dlaczego warto je stosować

- **Pełne pokrycie scenariuszy biznesowych:** testy E2E sprawdzają logikę aplikacji od początku do końca, co pozwala mieć pewność, że kluczowe ścieżki są wolne od krytycznych błędów.
- **Symulacja realnych warunków:** testy odzwierciedlają działania prawdziwego użytkownika, dzięki czemu można wychwycić problemy wynikające z kolejności czynności czy specyfiki interfejsu.
- **Wysoka wiarygodność:** przejście całego procesu „na żywo” daje dobre potwierdzenie, że aplikacja działa w sposób oczekiwany.

Z drugiej strony testy E2E są zwykle **najwolniejsze** i **najtrudniejsze w utrzymaniu**, ponieważ wymagają uruchomienia wszystkich niezbędnych elementów systemu (np. front-end, back-end, baza danych, zewnętrzne usługi). Dlatego stosuje się je najczęściej do krytycznych ścieżek użytkownika oraz do ostatecznej weryfikacji przed wdrożeniem.

Przykładowe narzędzia

Do najpopularniejszych rozwiązań wspierających testy end-to-end w aplikacjach webowych należą:

- **Selenium WebDriver** – klasyczne narzędzie automatyzujące przeglądarkę, szeroko wspierane przez społeczność.
- **Cypress** – nowoczesny framework skupiający się na testach front-endu z szybkim sprzężeniem zwrotnym.
- **Playwright** – rozwijane przez Microsoft narzędzie oferujące bogate możliwości w zakresie automatyzacji przeglądarek (Chromium, Firefox, WebKit).
- **TestCafe** – stosunkowo proste rozwiązanie pozwalające pisać testy E2E w JavaScripcie i TypeScripcie, bez konieczności instalacji dodatkowych sterowników przeglądarki.

Wybór konkretnego narzędzia zależy od technologii użytych w danym projekcie, wymagań zespołu oraz preferencji co do języka programowania.

Rozdział 3

Dostępne technologie

Rozdział poświęcony krótkiej analizie i porównaniu dostępnych narzędzi czy bibliotek wspierających tworzenie i uruchamianie testów dla aplikacji webowych.

3.1 Frameworki testowe

Omów pokrótce kilka popularnych frameworków (np. *Jest*, *Mocha*, *JUnit*, *TestNG*, *PyTest*, *Playwright*, *Selenium*), zwracając uwagę na to, w jaki sposób mogą być zastosowane do testów webowych.

3.2 Biblioteki wspomagające generowanie testów

Wspomnij np. o rozwiązaniach, które pozwalają w jakiś sposób wspierać generowanie testów: *record-and-play*, narzędzia do analizy statycznej, itp. Możesz też nawiązać do *Selenium* i *Playwright codegen*.

3.3 Narzędzia CI/CD

Zasygnalizuj, że w projektach o większej skali testy automatyczne integruje się z pipeline'ami CI/CD (np. GitLab CI, GitHub Actions, Jenkins).

Rozdział 4

Strategie generacji testów

Przedstaw wybrane podejścia do automatycznego lub półautomatycznego generowania testów.

4.1 Generowanie testów na podstawie rejestrowania akcji

Opis: rejestrowanie czynności użytkownika w przeglądarce (np. za pomocą *Selenium IDE*, *Playwright Codegen*), a następnie generowanie skryptów testowych.

4.2 Generowanie testów na podstawie modeli

Możesz omówić krótko podejście *Model-Based Testing*, gdzie testy są generowane z modelu zachowań systemu, stanów i przejść.

4.3 Analiza statyczna i tworzenie testów

Wspomnij, że niektóre narzędzia potrafią analizować kod aplikacji i np. na podstawie ścieżek wywołań generować scenariusze testowe.

Rozdział 5

Architektura oraz interfejs aplikacji

W tym rozdziale opiszesz konstrukcję swojego narzędzia, jego moduły, sposób komunikacji itp.

5.1 Ogólny zarys architektury

Zwięźle opisz główne moduły aplikacji (np. moduł generowania testów, moduł interfejsu graficznego/CLI, moduł zarządzania danymi). Wskaż, w jaki sposób te moduły się ze sobą komunikują.

5.2 Interfejs użytkownika

Zaprezentuj, jak wygląda interfejs (graficzny lub konsolowy) Twojej aplikacji:

- główne ekrany/formularze,
- sposób nawigacji,
- kluczowe funkcje (np. generowanie pliku testowego, zapis ustawień).

Możesz dołączyć zrzuty ekranu i omówić je.

Rozdział 6

Omówienie kodu źródłowego aplikacji

Przedstaw strukturę plików oraz folderów oraz wyjaśnij najważniejsze fragmenty implementacji. Wskaż, które biblioteki z rozdziału o „Dostępnych technologiach" zostały wykorzystane i w jaki sposób.

6.1 Struktura projektu

Opisz, w jaki sposób podzieliłeś/łaś projekt na moduły, pakiety, foldery:

- logika generowania testów,
- definicja modelu danych (np. dane o testach, konfiguracje),
- klasy pomagające w tworzeniu raportów lub integracji z CI.

6.2 Kluczowe fragmenty kodu

Zademonstruj przykłady najważniejszych funkcji/metod, np.:

- kod odpowiedzialny za rejestrowanie interakcji,
- generowanie plików testowych,
- konwersję danych do konkretnych frameworków.

Przedstaw je w formie listingów z krótkim komentarzem.

Rozdział 7

Przykłady użyć aplikacji

W tym rozdziale możesz zaprezentować, jak Twoje narzędzie działa w praktyce.

7.1 Uruchamianie narzędzia w trybie interaktywnym

Pokaż krok po kroku, jak użytkownik wypełnia formularz czy wybiera opcje w CLI, aby wygenerować testy. Zademonstruj końcowe pliki testowe.

7.2 Integracja z procesem CI/CD

Wyjaśnij, jak można włączyć wygenerowane testy do automatycznego pipeline'u, np. w GitLab CI, Jenkinsie, itp. Przedstaw przykładowy fragment konfiguracji.

Rozdział 8

Podsumowanie

Na koniec podsumuj swoją pracę, oceń jej efekty i zaproponuj kierunki rozwoju.

8.1 Wnioski końcowe

Opisz, w jakim stopniu udało się osiągnąć założone cele, co było największym wyzwaniem, co dało najwięcej satysfakcji/rezultatów.

8.2 Możliwości dalszego rozwoju

Opisz, jakie jeszcze funkcjonalności można by dodać do narzędzia, jakie usprawnienia byłyby przydatne, jakie biblioteki lub technologie mogłyby zostać wykorzystane w przyszłości.

Bibliografia

[1] Roman, A., & Zmitrow, K. (2024). *Testowanie oprogramowania w praktyce: studium przypadków 2.0*.

[2] Osherove, R. (2024). *Testy jednostkowe: świat niezawodnych aplikacji*.

[3] Roman, A., & Zmitrow, K. (2024). *Testowanie oprogramowania w praktyce: studium przypadków*.

[4] Roman, A. (2024). *Testowanie i jakość oprogramowania: modele, techniki, narzędzia*.

[5] CircleCI. (n.d.). What is End-to-End Testing? Pozyskano z `https://circleci.com/blog/what-is-end-to-end-testing/`

[6] Microsoft Playwright. (n.d.). Introduction to Playwright. Pozyskano z `https://playwright.dev/docs/intro`