Michał Pianka 229979

# Plan Testów

Dokument opisujący zakres, metody, zasoby oraz harmonogram zamierzonych czynności testowych. Określa, między innymi: elementy testowe cechy testowe, zadania testowe, kto będzie te zadania wykonywał, stopień niezależności testerów, środowisko testowe, techniki projektowania testów oraz kryteria wejścia i wyjścia, przesłanki ich użycia, a także ryzyka wymagające ciągłego planowania.

Przykładowa struktura testów:

1. Dziennik zmian

Michał Pianka 19.10.2023 Plan testów i dodanie przypadków testowych

Michał Pianka 09.11.2023 Zmiany w planie testów dodanie przypadków testowych dla API.

1. Wstęp

Dokument ten opisuje plan testów kluczowych funkcjonalności dla systemu Practise Software Testing. Dokument ma za zadania usprawnić proces testowania.

1. Terminologia i definicje

System – aplikacja webowa

API – zbiór definicji i protokołów, które umożliwiają różnym programom komunikację i współpracę ze sobą. API udostępnia endpointy.

Endpoint - jest identyfikowany przez adres URL. Służy do udostępnienia funkcji lub danych.

UI – interfejs użytkownika

Moduł obsługi kont – funkcjonalności systemu związane z zarządzaniem kontem użytkownika.

Body – ciało żądania lub odpowiedzi.

JSON – format pliku tekstowego

1. Zakres dokumentu

Ten dokument zawiera informacje na temat fazy planowania testowania. Opisywany jest sposób testowania dla głównych funkcjonalności systemu tj. obsługa kategorii produktów (wyświetlanie, dodawanie, aktualizacja, usuwanie).

1. Opis testowanego systemu

Testowany system jest sklepem narzędzi ręcznych, które są powszechnie używane w różnych dziedzinach pracy, takich jak budownictwo, naprawy domowe, warsztaty, stolarstwo i inne. System pozwala nam na zakup narzędzi, a także posiada szereg ułatwień wyszukiwania narzędzi poprzez sortowanie, wyszukiwanie po cenie itp.

1. Zakres planowanych testów

Przewiduje się przeprowadzenie testów funkcjonalnych.

Testy będą obejmować funkcje tj. obsługa kategorii produktów.

1. Założenia i ograniczenia

Testerzy mają dostępu do dokumentacji systemu, ale nie do wymagań klienta. Testerzy będą sprawdzać system na podstawie ogólnoprzyjętych standardów tworzenia aplikacji.

1. Interesariusze i komunikacja

W procesie testowania bierze udział 2 osobowy zespół testerów komunikujących się przy pomocy platformy discord.

1. Strategia testowania

Testowanie dynamiczne – testy funkcjonalne

Cel: Sprawdzenie endpointów pod kątem zwracania odpowiednich statusów kodów HTTP, zwracania odpowiednich odpowiedzi oraz sprawdzenie walidacji JSON, aby upewnić się, że działają zgodnie z oczekiwaniami.

Narzędzia: Postman, Selenium

Przewiduje się automatyzację testów UI przy użyciu Selenium.

Klasyfikacja defektów:

Krytyczne: Błędy uniemożliwiające podstawową funkcjonalność.

Ważne: Błędy, które powodują utratę danych, ale nie uniemożliwiają podstawowych funkcji.

Dokumentacyjne: Błędy, które nie wpływają na działanie programu, ale testy dają wynik sprzeczny z dokumentacją.

Kryteria akceptacji: Mniej niż 5% krytycznych defektów i mniej niż 10% ważnych defektów, mniej niż 10% dokumentacyjnych defektów.

1. Artefakty z procesu testowego

Spodziewane artefakty obejmują pełny zestaw przypadków testowych, zarówno manualnych, jak i automatycznych, oraz raporty z testów.

1. Opis środowiska testowego

Architektura: System będzie testowany na architekturze klient-serwer.

Konfiguracja: System zostanie przetestowany na systemie operacyjnym Linux w przeglądarce Google Chrome.

1. Harmonogram testowania

20.10.2023 – 09.11.2023 – implementacja testów API

09.11.2023 – 22.11.2023 – implementacja testów UI

1. Implementacja testów

Testy API zostaną zaimplementowane w Postmanie i przechowywane w kolekcji grupowej na platformie Postman. Żądania zostaną również wyeksportowane do pliku .json.

Testy UI zostaną zaimplementowane w Selenium i przechowywane w repozytorium na platformie Github.

# Przypadki testowe

|  |  |  |
| --- | --- | --- |
| **ID** | **Nazwa testu** | **Kategoria** |
| 1 | GetCategoryByIdPositive | Basic positive test |
| 2 | GetCategoryByIdNegative | Negative testing with valid input |
| 3 | GetCategoryByIdNegativeValid | Negative testing with invalid input |
| 4 | PostCategoryPositive | Basic positive test |
| 5 | PostCategoryNegative | Negative testing with invalid input |
| 6 | PostCategoryNegative2 | Negative testing with valid input |
| 7 | PutCategoryByIdNegative | Negative testing with valid input |
| 8 | PutCategoryByIdPositive | Basic positive test |
| 9 | PutCategoryByIdNegativeInvalid | Negative testing with invalid input |
| 10 | PutCategoryByIdNegativeInvalid | Negative testing with valid input |
| 11 | DeleteCategoryNegativeInvalid | Security, authorization, and permissions test |
| 12 | DeleteCategoryNegativeValid | Negative testing with valid input |
| 13 | DeleteCategoryNegativeInvalid | Negative testing with invalid input |
| 14 | DeleteCategoryPositive | Basic positive test |
| 15 | GetProductByIdPositive | Basic positive test |
| 16 | GetProductByIdNegativeValid | Negative testing with valid input |
| 17 | GetProductByIdNegativeInvalid | Negative testing with invalid input |
| 18 | PostProductPositive | Basic positive test |
| 19 | PostProductNegativeUnauthorized | Negative test with no authorization |
| 20 | PostProductNegativeForbidden | Negative test with wrong authorization |
| 21 | PostProductNegativeInvalid | Negative test invalid input |
| 22 | PutProductPositive | Basic positive test |
| 23 | PutProductInvalid | Negative test invalid input |
| 24 | PutProductUnauthorized | Negative test no authorization |
| 25 | DeleteProductPositive | Basic positive test |
| 26 | DeleteProductInvalid | Negative test with wrong id |
| 27 | DeleteProductUnauthorized | Negative test no authorization |

Test nr 1 ma za zadanie sprawdzić poprawność wyświetlania kategorii o podanym Id. Priorytet krytyczny. Aby zrealizować test należy wysłać żądanie GET na odpowiedni endpoint wraz z Id kategorii znajdującym się w bazie danych. Po wysłaniu żądania oczekuje odpowiedzi serwera, który w body prześle w formacie JSON informacje o kategoriach. Oraz zwróci status kod 200.

Test nr 2 ma zadanie sprawdzić, co się stanie po spróbowaniu wyświetlenia kategorii o Id nie znajdującym się w bazie. Priorytet Ważne. Aby zrealizować test należy wysłać żądanie GET na odpowiedni endpoint wraz z Id kategorii nie znajdującym się w bazie danych. Po wysłaniu żądania oczekuje odpowiedzi serwera, który w body prześle w formacie JSON informacje o niepowodzeniu. Oraz zwróci status kod 404.

Test nr 3 ma zadanie sprawdzić, co się stanie po dodaniu ciała żądania do endpointa. Priorytet Dokumentacyjny. Aby zrealizować test należy wysłać żądanie GET na odpowiedni endpoint wraz z Id kategorii znajdującym się w bazie danych lecz z dodatkową wiadomością w ciele żądania w formacie JSON. Po wysłaniu żądania oczekuje odpowiedzi serwera, który w body prześle w formacie JSON informacje o błędnym żądaniu. Oraz zwróci status kod 400.

Test nr 4 ma zadanie sprawdzić poprawność dodawania nowej kategorii. Priorytet krytyczny. Aby zrealizować test należy wysłać żądanie POST na odpowiedni endpoint z odpowiednim ciałem żądania (name,slug). Po wysłaniu żądania oczekuje odpowiedzi serwera, który w body prześle w formacie JSON informacje o stworzeniu kategorii. Oraz zwróci status kod 201.

Test nr 5 ma za zadanie sprawdzić czy kategoria zostanie dodane pomimo dodania do ciała żądania nieznanej wartości. Priorytet krytyczny. Aby zrealizować test należy wysłać żądanie POST na odpowiedni endpoint dodając do ciała żądania dodatkową wartość. Po wysłaniu żądania oczekuje odpowiedzi serwera, który w body prześle w formacie JSON informacje o błędnym żądaniu. Oraz zwróci status kod 422.

Test nr 6 ma za zadanie sprawdzić można stworzyć kategorię o już użytej nazwie. Priorytet krytyczny. Aby zrealizować test należy wysłać żądanie POST na odpowiedni endpoint, a w ciele żądania wysłać już zarezerwowaną nazwę. Po wysłaniu żądania oczekuje odpowiedzi serwera, który w body prześle w formacie JSON informacje o błędnym żądaniu ze względu na duplikację nazwy kategorii. Oraz zwróci status kod 422.

Test nr 7 ma za zadanie sprawdzić można zaktualizować kategorię o już użytą nazwę. Priorytet krytyczny. Aby zrealizować test należy wysłać żądanie PUT na odpowiedni endpoint, a w ciele żądania wysłać już zarezerwowaną nazwę. Po wysłaniu żądania oczekuje odpowiedzi serwera, który w body prześle w formacie JSON informacje o błędnym żądaniu ze względu na duplikację nazwy kategorii. Oraz zwróci status kod 422.

Test nr 8 ma za zadanie sprawdzić można zaktualizować kategorię. Priorytet krytyczny. Aby zrealizować test należy wysłać żądanie PUT na odpowiedni endpoint, a w ciele żądania wysłać niezarezerwowaną nazwę. Po wysłaniu żądania oczekuje odpowiedzi serwera, który w body prześle w formacie JSON informacje o powodzeniu. Oraz zwróci status kod 200.

Test nr 9 i 10 (różnią się poprawnością JSONA) ma za zadanie sprawdzić czy można zaktualizować nieistniejącą kategorię. Priorytet dokumentacyjny. Aby zrealizować test należy wysłać żądanie PUT na endpoint z nieistniejącym id categorii. Po wysłaniu żądania oczekuje odpowiedzi serwera, który w body prześle w formacie JSON informacje o błędnym żądaniu ze względu na próbę zaktualizowania nieistniejącej kategorii. Oraz zwróci status kod 404.

Test 11 ma za zadanie sprawdzić czy można usunąć kategorię nie mając do tego uprawnień. Priorytet krytyczny. Aby zrealizować test należy wysłać żądanie DELETE na odpowiedni endpoint bez podania tokenu uwierzytelnienia. Po wysłaniu żądania oczekuje odpowiedzi serwera, który w body prześle w formacie JSON informacje o zabronieniu wykonania czynności. Oraz zwróci status kod 403.

Test 12 ma za zadanie sprawdzić czy można usunąć kategorię w użyciu(prawdopodobnie powiązana kaskadą). Priorytet ważne. Aby zrealizować test należy wysłać żądanie DELETE na odpowiedni endpoint podając Id kategorii powiązanej z inną encją. Po wysłaniu żądania oczekuje odpowiedzi serwera, który w body prześle w formacie JSON informacje o konflikcie. Oraz zwróci status kod 409.

Test 13 ma za zadanie sprawdzić czy można usunąć kategorię nieistniejącą. Priorytet dokumentacyjny. Aby zrealizować test należy wysłać żądanie DELETE na odpowiedni endpoint podając Id kategorii nieistniejącej. Po wysłaniu żądania oczekuje odpowiedzi serwera, który w body prześle w formacie JSON informacje o niepoprawnym id. Oraz zwróci status kod 422.

Test 14 ma za zadanie sprawdzić czy można usunąć kategorię. Priorytet krytyczny. Aby zrealizować test należy wysłać żądanie DELETE na odpowiedni endpoint podając Id istniejącej kategorii wraz z tokenem admina. Po wysłaniu żądania oczekuje odpowiedzi serwera, który w body prześle w formacie JSON informacje o poprawnym usunięciu kategorii. Oraz zwróci status kod 204.

Test nr 15 ma za zadanie sprawdzić poprawność wyświetlania produktu o podanym Id. Priorytet krytyczny. Aby zrealizować test należy wysłać zadanie GET na odpowiedni endpoint wraz z poprawnym (znajdującym się w bazie) id produktu. Po wysłaniu żądania oczekuje odpowiedzi serwera, który w body prześle w formacie JSON informację o produkcie (kod 200)

Test nr 16 ma za zadanie sprawdzić odpowiedź serwera na podanie produktu o niewłaściwym Id. Priorytet wysoki. Aby zrealizować test należy wysłać zadanie GET na odpowiedni endpoint wraz z nieznajdującym się w bazie id produktu. Po wysłaniu żądania oczekuje odpowiedzi serwera, “Requested item not found” (kod 404)

Test nr 17 ma za zadanie sprawdzić odpowiedź serwera na podanie produktu o niewłaściwym Id. Priorytet wysoki. Aby zrealizować test należy wysłać zadanie GET na odpowiedni endpoint wraz z niepoprawnym id produktu. Po wysłaniu żądania oczekuje odpowiedzi serwera, “Requested item not found” (kod 404)

Test nr 18 ma za zadanie sprawdzić poprawność dodawania produktu. Priorytet krytyczny. Aby zrealizować test należy wysłać zadanie POST na odpowiedni endpoint wraz z poprawnym JSONem produktu I autoryzacja. Po wysłaniu żądania oczekuje odpowiedzi serwera, który w body prześle w formacie JSON informację o produkcie (kod 200)

Test nr 19 ma za zadanie sprawdzić poprawność dodawania produktu. Priorytet krytyczny. Aby zrealizować test należy wysłać zadanie POST na odpowiedni endpoint wraz z poprawnym JSONem produktu I bez autoryzacji. Po wysłaniu żądania oczekuje odpowiedzi serwera (kod 401)

Test nr 20 ma za zadanie sprawdzić poprawność dodawania produktu. Priorytet krytyczny. Aby zrealizować test należy wysłać zadanie POST na odpowiedni endpoint wraz z poprawnym JSONem produktu I z tokenem należącym do zwykłego użytkownika. Po wysłaniu żądania oczekuje negatywnej odpowiedzi serwera (kod 403)

Test nr 21 ma za zadanie sprawdzić poprawność dodawania produktu. Priorytet krytyczny. Aby zrealizować test należy wysłać zadanie POST na odpowiedni endpoint wraz z niepoprawnym JSONem produktu I autoryzacja. Po wysłaniu żądania oczekuje negatywnej odpowiedzi serwera (kod 422)

Test nr 22 ma za zadanie sprawdzić poprawnosc modyfikacji produktu. Priorytet krytyczny. Aby zreallizować test należy wysłac żądanie PUT na odpowiedni endpoint I podać odpowiedni id produktu, w “body” którego znajduje się odpowiedni JSON, z odpowiednią autoryzacją. Po wysłaniu żądania oczekuje odpowiedzi serwera, (200)

Test nr 23 ma za zadanie sprawdzić poprawnosc modyfikacji produktu. Priorytet krytyczny. Aby zreallizować test należy wysłac żądanie PUT na odpowiedni endpoint I podać odpowiedni id produktu, w “body” którego znajduje się nieodpowiedni JSON, z odpowiednią autoryzacją. Po wysłaniu żądania oczekuje odpowiedzi serwera, (400)

Test nr 24 ma za zadanie sprawdzić zabezpieczenie modyfikacji produktu. Priorytet krytyczny. Aby zreallizować test należy wysłac żądanie PUT na odpowiedni endpoint I podać odpowiedni id produktu, w “body” którego znajduje się odpowiedni JSON, ze złą autoryzacją. Po wysłaniu żądania oczekuje odpowiedzi serwera, (403)

Test nr 25 ma za zadanie sprawdzić poprawnosc usuwania produktu. Priorytet krytyczny. Aby zreallizować test należy wysłac żądanie PUT na odpowiedni endpoint I podać odpowiedni id produktu, zodpowiednią autoryzacją. Po wysłaniu żądania oczekuje odpowiedzi serwera, (200)

Test nr 26 ma za zadanie sprawdzić poprawnosc usuwania produktu. Priorytet krytyczny. Aby zreallizować test należy wysłac żądanie PUT na odpowiedni endpoint I podać nieodpowiedni id produktu, zodpowiednią autoryzacją. Po wysłaniu żądania oczekuje odpowiedzi serwera, (404)

Test nr 27 ma za zadanie sprawdzić zabezpieczenia usuwania produktu. Priorytet krytyczny. Aby zreallizować test należy wysłac żądanie PUT na odpowiedni endpoint I podać odpowiedni id produktu, bez odpowiedniej autoryzacji. Po wysłaniu żądania oczekuje odpowiedzi serwera, (403)

# Raport z wykonania testów

Sumaryczny dokument przedstawiający działania testowe i ich rezultaty; zawiera także ocenę testowanych elementów pod względem zgodności z kryteriami wyjścia. Przykładowa struktura:

1. Podsumowanie (co było testowane, ograniczenia testów).
2. Odchylenia od planu.
3. Ocena kompletności testów (poziom spełnienia wymagań określonych na początku projektu).
4. Czynniki opóźniające postęp oraz sposoby radzenia sobie z nimi.
5. Wartości miar w projekcie.
6. Ryzyka rezydualne (ryzyka pozostałe w programie i odkryte po ukończeniu testowania).
7. Produkty procesu testowego.
8. Raport z aktualnego stanu testów.
9. Wnioski.

Raportu z aktualnego stanu testów to dokument zawierający podsumowanie aktywności testowych i osiągniętych wyników, aby raportować posęp testowych w stosunku do założeń. Struktura:

1. Identyfikator raportu.
2. Data wykonania.
3. Tabela testów (Scenariusz, Przypadki testowe, Oczekiwane wyniki, Aktualny wynik, Status, Komentarz).