1. WPROWADZENIE

1.1 Cel

Test plan kierowany jest do wszystkich członków zespołu projektowego.
Zawiera opis strategii przeprowadzania procesu testowego aplikacji internetowej z obszaru e-commerce, włączając:

- cele i założenia przeprowadzanych testów,
- opis typów testów i poziomów testów,
- strategię wykonywania i zarządzania testami,
- opis cyklu życia defektu,
- identyfikację ewentualnych zagrożeń oraz sposób ich zapobiegania lub łagodzenia.

1.2 Opis projektu

Aplikacja z obszaru e-commerce ma umożliwić użytkownikowi dodawanie produktów do koszyka, edytowanie jego zawartości oraz przechodzenie do etapu finalizacji zamówienia. Aplikacja zostanie dostosowana zarówno do urządzeń desktopowych oraz mobilnych w najpopularniejszych rozdzielczościach oraz do najnowszych wersji systemów operacyjnych.

1.3 Odbiorcy

W skład zespołu QA wchodzą:

- 3 MT (Manual Tester) - 2 junior, 1 mid
- 1 AT (Junior Automation Tester)
- 1 TM (Test Manager)

W skład zespołu developerskiego wchodzą:

- 3 FE (Frontend Developer)
- 2 BE (Backend Developer)
- 1 PO (Product Owner)
- 1 SM (Scrum Master)
- 1 PM (Project Manager)

2. STRATEGIA TESTÓW

2.1 Cel testowania

- dokonywanie oceny produktów pracy, takich jak: wymagania, historyjki użytkownika, czy projekt;

- sprawdzanie na bieżąco czy spełniane są wszystkie wyspecyfikowane wymagania;
- sprawdzenie czy aplikacja jest kompletna i działa zgodnie oczekiwaniami interesariuszy;
- budowanie zaufanie do poziomu jakości testowanej aplikacji;
- zapobieganie i wykrywanie defektów;
- dostarczanie interesariuszom informacji niezbędnych do podejmowania świadomych decyzji (dotyczących zwłaszcza poziomu jakości aplikacji);
- obniżanie poziomu ryzyka związanego z jakością aplikacji.

2.2 Założenia testów

Poniżej przedstawiam minimalne założenia, które muszą być spełnione, aby proces testowy został przeprowadzony, w sposób odpowiedni i dostosowany do wymagań:

- aplikacja jest zaprogramowana i gotowa do procesu testowego;
- zespół Scrum'owy/testerski posiada dostęp do dokumentacji, narzędzi oraz wymagań interesariuszy;
- środowisko testowe jest dostępne;
- wyniki testów będą raportowane codziennie za pomocą odpowiednich narzędzi, które są dostępne;
- skrypty testowe będą opracowywane i zatwierdzane przez odpowiednie osoby;
- zespół testerski będzie wspierał zespół developerski w zakresie przeprowadzania testów;
- wszelkie zależności będą raportowane natychmiastowo.

2.3 Poziomy i typy testów

2.3.1 Testy jednostkowe

Testy jednostkowe skupiają się na modułach (fragmentach kodu), które można przetestować oddzielnie.

Cel: wykrywanie defektów w module, budowanie zaufania do jakości modułu, sprawdzanie zgodności zachowań funkcjonalnych i niefunkcjonalnych modułu z projektem i specyfikacjami, zmniejszanie ryzyka

Zakres: funkcje, klasy, metody, moduły, moduły bazodanowe, konwersje/migracje danych

Testerzy: zespół developerski

Metody: testy automatyczne uruchamiane z poziomu kodu lub systemem CI/CD

Czas: testy jednostkowe są wykonywane niezależnie przez cały czas tworzenia oprogramowania, a wykryte usterki są od razu naprawiane przez programistę.

2.3.2 Testy integracyjne i systemowe

Testy integracyjne skupiają się na interakcjach między modułami lub systemami.
Testy systemowe skupiają się na zachowaniu i możliwościach całej aplikacji.

Cel: budowanie zaufania do jakości interfejsów, wykrywanie defektów, sprawdzenie zgodności zachowań funkcjonalnych i niefunkcjonalnych interfejsów i zachowań z projektem i specyfikacjami, zmniejszanie ryzyka, zapobieganie przedostawaniu się defektów na wyższe poziomy testowania, sprawdzenie kompletności systemu i prawidłowości jego działania.

Zakres: interfejsy, infrastruktura, bazy danych, podsystemy, API, aplikacja

Testerzy: zespół QA lub zespół developerski

Metody: jeśli to możliwe to testy automatyczne, jednak mogą to być również testy manualne

Czas: po integracji modułów/systemów

2.3.3 Testy akceptacyjne

Testy akceptacyjne skupiają się na zachowaniu i możliwościach całego systemu/aplikacji.

Cel: budowanie zaufania do aplikacji, sprawdzenie kompletności aplikacji i jej prawidłowego działania, sprawdzenie zgodności zachowania funkcjonalnego i niefunkcjonalnego aplikacji ze specyfikacją.

Zakres: aplikacja podlegająca testowaniu

Testerzy: PO oraz zarząd firmy zlecającej projekt

Metody: testy manualne

Czas: przed wdrożeniem na produkcję

2.3.4 Testy eksploracyjne

Testy eksploracyjne polegają na wykonywaniu testów nieformalnych, to odkrywanie aplikacji na bazie doświadczenia.

Cel: budowanie zaufania do aplikacji, sprawdzenie zgodności zachowania funkcjonalnego i niefunkcjonalnego aplikacji, obserwacja zachowania aplikacji

Zakres: aplikacja podlegająca testowaniu

Testerzy: zespół QA

Metody: testy manualne

Czas: gdy aplikacja jest gotowa

2.3.5 Testy funkcjonalne

Testy funkcjonalne umożliwiają dokonanie oceny funkcji jakie aplikacja realizuje.

Cel: ocena funkcji jakie aplikacja realizuje, budowanie zaufania do aplikacji, sprawdzenie zgodności zachowania funkcjonalnego aplikacji ze specyfikacją.

Zakres: określone funkcje aplikacji

Testerzy: zespół QA

Metody: testy manualne, testy automatyczne

Czas: gdy dana funkcja jest ukończona i gotowa

2.3.6 Testy niefunkcjonalne

Cel: ocena charakterystyki aplikacji takich jak: użyteczność, wydajność i bezpieczeństwo, sprawdzenie zachowania aplikacji przy zwiększonym obciążeniu, sprawdzenie czy aplikacja jest pielęgnowana, czy można łatwo ją zmodyfikować i dostosować do ewentualnych nowych wymagań.

Zakres: określone niefunkcjonalne właściwości aplikacji

Testerzy: zespół QA

Metody: testy głównie automatyczne, ale również manualne

Czas: jak najwcześniej (zgodnie z zasadą “wczesne testowanie”)

2.3.7 Automatyczne testy regresji

Cel: zapobieganie regresji, czyli skutków ubocznych wprowadzonych zmian w kodzie

Zakres: moduł, system, konwersje/migracje danych

Testerzy: zespół QA, zespół developerski

Metody: testy automatyczne

Czas: jak najwcześniej

2.4 Produkty pracy

Proces testowania dostarczy wiele specyficznych dokumentów, raportów i wyników w projekcie, takich jak:

- test plan
- przypadki testowe

- raport końcowy
- raporty defektów
- skrypty testów automatycznych
- Product Backlog

2.5 Oszacowanie nakładu pracy podczas procesu testowego

Działania zespołu testerskiego	Potrzebny czas (MD)
Opracowanie test planu	2 MD
Analiza wymagań	1 MD
Rozmowa z deweloperami i klientem, aby dowiedzieć się jak najwięcej o aplikacji	1 MD
Opracowanie przypadków testowych	3 MD
Wykonanie przypadków testowych	9 MD
Raportowania defektów	2 MD
Retesty	2 MD
Testy regresji	5 MD
Napisanie skryptów testów automatycznych	3 MD
Raport końcowy	1 MD
Analiza raportu podsumowującego oraz posumowanie całego procesu testowego	1 MD

3. STRATEGIA REALIZACJI

3.1 Kryteria wejścia i wyjścia

Kryteria wejścia:

Do prawidłowego przeprowadzenia procesu testowego powinny zostać spełnione następujące warunki:

- dostęp do aktualnych wymogów i specyfikacji;
- dostęp do niezbędnych narzędzi;
- dostęp do danych testowych;
- aplikacja powinna być gotowa do przeprowadzenia testów;
- dostęp do odpowiedniego środowiska;

- przed przeprowadzenia testów akceptacyjnych muszą być ukończone wszystkie testy jednostkowe, systemowe oraz testy integracyjne;

Kryteria wyjścia:

Do prawidłowego ukończenia procesu testowego powinny zostać spełnione następujące wymogi:

- wszystkie przypadki testowe zostały wykonane oraz wszystkie skrypty przeprowadzanych przypadków testowych zostały ukończone, wliczając m.in.:
    - wszystkie retesty zakończone
    - defekty są zamknięte
    - brak regresji w aplikacji
    - testy akceptacyjne zostały ukończone i zatwierdzone
- po ukończeniu testów niefunkcjonalnych uzyskanie odpowiednich wskaźników wydajności, użyteczności, niezawodności oraz innych przewidzianych charakterystyk;
- raport końcowy został opracowany i przekazany do interesariuszy.

3.2 Walidacja i zarządzanie defektami

W celu usystematyzowania pracy oraz zadbania o jakość produktu, po wykryciu błędu proponuje się wykonywanie następujących czynności:

- zgłoszenie defektu w odpowiednim narzędziu (JIRA), zaleca się następującą formułę podczas zgłaszania błędu:
    - **Środowisko testowe**

      …

      Dane testowe

      …

      Warunki wstępne

      …

      Kroki do odtworzenia

      …

      Oczekiwany rezultat

      …

      Aktualny rezultat

      …

Występowalność

X/5

Priorytet

1 | 2 | 3 | 4 | 5 | ;

- przekazanie zgłoszenia do zespołu deweloperskiego;
- debugowanie defektu;
- przekazanie do retestu;
- wykonanie retestu przez zespój testerski;
- jeśli błąd nadal występuje, zaleca się ponowne debugowanie danego defektu;
- jeśli błąd został naprawiony, zaleca się testy regresji oraz zamknięcie zgłoszenia.

Harmonogram zgłaszania błędów powinien być zależny od ich priorytetu (zaczynając od tych krytycznych).

3.3 Metryka testów

Metryka	Formuła
Udana ilość testów	$Udana\ ilość\ testów = \left(\frac{Ilość\ poprawnych\ testów}{Liczba\ wszystkich\ wykonanych\ testów}\right) \cdot 100\%$ $\left(\frac{382}{395}\right) \cdot 100\% = 96\%$
Nieudana ilość testów	$Nieudana\ ilość\ testów = \left(\frac{Ilość\ nieudanych\ testów}{Liczba\ wszystkich\ wykonanych\ testów}\right) \cdot 100\%$ $\left(\frac{13}{395}\right) \cdot 100\% = 4\%$
Poprawione defekty	$Poprawione\ defekty = \left(\frac{Ilość\ poprawionych\ defektów}{Całkowita\ liczba\ zgłoszeń\ defektów}\right) \cdot 100\%$ $\left(\frac{551}{801}\right) \cdot 100\% = 68,7\%$
Pokrycie wykonanych testów	$Pokrycie\ wykonanych\ testów = \left(\frac{Ilość\ testów\ wykonanych}{Całkowita\ liczba\ testów\ do\ wykonania}\right) \cdot 100\%$ $\left(\frac{423}{451}\right) \cdot 100\% = 93\%$

4. PROCES ZARZĄDZANIA TESTAMI

4.1 Narzędzia do zarządzania procesem testowym

Głównymi narzędziami wykorzystywanym do zarządzania procesem testowym są:

- Jira - do zarządzania procesem testowym;
- Confluence - do zarządzania dokumentacją;
- TestRail - do zarządzania przypadkami testowymi.

4.2 Proces tworzenia testów

Zalecanym narzędziem do tworzenia przypadków testowych oraz zarządzania nimi jest TestRail. Tworzeniem przypadków testowych zajmować się będzie zespół testerski. W celu zwiększenia efektywności testowania przypadki testowe będą tworzone na podstawie wymagań (po ich szczegółowej analizie), na podstawie doświadczenia oraz zgodnie z zasadami testowania opartego na ryzyku. Każdy tworzony przypadek testowy będzie miał określony adekwatny priorytet.

4.3 Proces wykonywania testów

Po opracowaniu poszczególnych przypadków i procedur testowych zaleca się utworzyć harmonogram wykonywania testów, który określi kolejność ich wykonywania.
Harmonogram powinien przede wszystkim uwzględniać takie czynniki jak: priorytet, zależności, konieczność wykonywania testów potwierdzających (retesty) i regresji.
Na bazie wcześniej przygotowanych przypadków testowych tworzone będą test run'y według harmonogramu.

4.4 Ryzyka i sposoby ich minimalizacji

- Ryzyka produktowe:

Ryzyko	Prawdopodobieństwo	Wpływ	Plan złagodzenia
Wysoka złożoność struktury modułu	średnie	wysoki	• Wzajemne wsparcie zespołu; • Zadbanie o odpowiednią specyfikację struktury modułów.

Duże zależności między systemami: zespół z Polski tworzy forntend , a za warstwę backend'ową odpowiada firma zewnętrzna ze Skandynawii	wysokie	wysoki	• Zadbanie o szczegółową dokumentację i specyfikację produktu; • Zadbanie o dobrą komunikację między zespołami.
Stosowanie innowacyjnej architektury tworzenia oprogramowania	wysokie	wysoki	• Dokładne przeanalizowanie dokumentacji dotyczącej nowej architektury; • Najbardziej złożonymi częściami tworzonego oprogramowania zajmować się będą najlepiej przeszkoleni i doświadczeni deweloperzy i oni również będą zajmować się przeprowadzaniem testów oraz debugowaniem błędów krytycznych.
Napięte harmonogramy	średnie	średni	• Zadbanie o odpowiednią estymację czasu pracy zespołu.

- Ryzyka projektowe

Ryzyko	Prawdopodobieństwo	Wpływ	Plan złagodzenia
Niewystarczające kwalifikacje zespołu testerskiego	wysokie	wysoki	• wsparcie junior'ów przez mid’a oraz przez Test Managera • przeprowadzenie dodatkowych szkoleń
Braki kadrowe	niskie	wysoki	• Wsparcie testerów z innych zespołów (outsourcing)
Niska jakość artefaktów testowych	niskie	średni	• Zadbanie o wysoką jakość artefaktów; • Wparcie Test Managera na każdym etapie procesu testowego;
Błędy lub braki w dokumentacji	średnie	wysoki	• od początku procesu testowego szczegółowa analiza wymagań oraz wszystkich dokumentów; • Rozwiewanie wszelkich wątpliwości “na bieżąco”.
Niedziałające lub niedostępne środowisko testowe	średnie	wysoki	• Zadbanie o dostęp do środowiska testowego; • Wzajemne wspieranie zespołu
Niedostępne narzędzia	niskie	wysoki	• Zadbanie o dostęp do narzędzi poprzez wykupienie odpowiednich licencji.
Problemy w komunikacji w zespole	średnie	średni	• Częste spotkania zespołów lub/i w zespole.

Brak wsparcia kadry zarządzającej	średnie	średni	• Szkolenia menadżerskie dla kadry zarządzającej; • Motywowanie kadry zarządzającej do wspierania zespołu.
Bariery językowe	wysokie	średni	• Przeprowadzenie kursów językowych dla pracowników • Wsparcie pracy zespołu przez lektora
Inna strefa czasowa	wysokie	niski	• Elastyczność godzin pracy, odpowiednio do aktualnych potrzeb.
Niski standard pracy w metodyce Scrum	niskie	średni	• Szkolenia dla zespołu ze sposobu wytwarzania oprogramowania w sposób zwinny, w metodyce Agile (framework - Scrum); • Wsparcie Scrum Mastera jako lidera, który służy organizacji i Scrum Teamowi.

4.5 Odpowiedzialność za testy

4.5.1 Zespół testerski

Odpowiedzialności w procesie testowym są zależne od pełnionego stanowiska oraz funkcji w zespole. Inne zadania wchodzą w zakres obowiązków kierownika testów (Test Manager), a inne testera.

Zadania kierownika testów:

- planowanie testów z uwzględnieniem czynników ryzyka,
- sporządzanie i aktualizowanie niniejszego planu testów,
- koordynowanie realizacji strategii testów i planu testów,
- inicjowanie procesów analizy, projektowania, implementacji i wykonywania testów,

- monitorowanie rezultatów testów,
- sprawdzanie statusów kryteriów wyjścia (definicji ukończenia),
- nadzór nad testami oraz zarządzaniem defektami,
- wspieranie całego zespołu testerskiego,
- przygotowanie raportu o postępie testów oraz raportu końcowego.

Zadania testerów:

- przegląd planu testów i pomoc w jego opracowaniu,
- analiza, przegląd i ocena wymagań, historyjek użytkownika, kryteriów akceptacji o specyfikacji pod kątem testowalności,
- projektowanie i implementowanie przypadków testowych i skryptów testowych,
- tworzenie harmonogramu testów,
- wykonywanie testów, ocenianie rezultatów,
- automatyzowanie testów w zależności od potrzeb,
- raportowanie defektów.

4.5.2 Zespół developerski

Do zadań, za które odpowiada zespół deweloperski zalicza się:

- tworzenie dobrej jakości kodu,
- przeprowadzanie testów na poziomie testowania modułowego (jednostkowego),
- przeprowadzanie testów na poziomie integracji modułów,
- naprawa defektów,
- przegląd kodu (code review),
- prowadzeni odpowiedniej dokumentacji.

5. ŚRODOWISKO TESTOWE

Aplikacja działa w wersji mobilnej oraz desktopowej, dlatego środowisko testowe będzie odpowiednio dobrane do tych wymagań:

Przeglądarki:

- Google Chrome
- Safari
- Firefox

Systemy operacyjne:

- macOS

- Windows
- IOS
- Android

Testy przeprowadzane będą na środowisku produkcyjnym.

6. NARZĘDZIA UŻYWANE W CAŁYM PROCESIE TESTOWYM

Podczas procesu testowego zaleca się używanie następujących narzędzi:

Narzędzie	Obszar
Jira	• do zarządzania procesem testowy (Product Backlogiem) • do zarządzania defektami (zgłaszaniam błędów oraz ich śledzeniem)
Confluence	• do zarządzania dokumentacją
Trello	• do zarządzania mniejszymi zadaniami w projekcie
TestRail	• do zarządzania przypadkami testowymi
Selenium IDE	• do nagrywania ekranu oraz parametryzacji poszczególnych akcji
Postman	• do przeprowadzania testów API
Screenpresso	• do zrzutów ekranu i nagrywania ekranu
Developer tools	• jako wsparcie przy wykonywaniu przypadków testowych (głównie niefunkcjonalnych)
Notepad++	• do edycji tekstu
IntelliJ Idea + biblioteki Java (Oracle)	• do tworzenia oraz edycji kodu źródłowego • do debugowania i testowania kodu
Github	• do publikacji oraz śledzenia zmian
Slack/Microsoft Teams	• do komunikacji wewnątrz oraz na zewnątrz zespołu