Kontekst testowania

1. Terminologia i definicje
    a. **opinia** - recenzja produktu wystawiona przez konkretnego użytkownika aplikacji
    b. **E2E** - testy end-to-end, bądź też testy systemowe, testujące wszystkie elementy aplikacji
    c. **baza danych** - relacyjna baza danych PostgreSQL służąca w projekcie do persystencji użytkowników, opinii i recenzowanych produktów
    d. **zewnętrzne API** - połączenie z zewnętrznym serwisem odpytywanym o szczegółowe informacje o produktach
    e. **backend** - mikroserwis z API HTTP łączący się z bazą danych przy użyciu prisma ORM oraz z interfejsem użytownika (frontend). zarówno backend jak i frontend napisane są w frameworku remix i posiadają wspólną implementację
    f. **frontend** - aplikacja przeglądarkowa z interfejsem użytkownika do aplikacji OpinionCollector komunikująca się z backendem przy użyciu REST API, również napisana w frameworku remix i posiadająca wspólną implementację z backendem

2. Podprocesy projektu i podprocesy testowe:
    a. testy integracyjne (baza danych zewnętrzne API, komunikacja backend - frontend)
    b. testy E2E (odwzorowanie wszystkich przypadków użycia opisanych w [1])
    c. unit testy (testy wszystkich modułów zawierających implementację logiki biznesowej)
    d. testy manualne (testy efektów wizualnych interfejsów użytkownika i poprawności interakcji użytkownika końcowego z systemem)

3. Elementy testowe
    a. Integracja:
        i. z bazą danych - testowanie persystencji użytkowników, opinii oraz produktów w bazie danych przy użyciu ORM
        ii. z zewnętrznymi API - testowanie połączenia z API udostępniającym informacje o ocenianych produktach
    b. Logika biznesowa
        i. zgodność z wymaganiami funkcjonalnymi - np. dodawanie nowych opinii tekstowych lub liczbowych, sugerowanie nowych produktów, itd.
        ii. poprawność zaimplementowanych funkcjonalności, np. poprawność algorytmu generującego sugestie, agregacji opinii per produkt, itd.
    c. Interfejs użytkownika
        i. poprawne odwzorowanie przypadków użycia (use case) - np. dodawanie nowych opinii
        ii. efekty wizualne - np. filtrowanie listy wyświetlonych produktów
    d. Bezpieczeństwo
        i. implementacji autoryzacji i uwierzytelnienia - poprawność customowej implementacji autoryzacji przy użyciu JWT

        ii. połączenia z bazą danych, zewnętrznymi API - poprawność bezpiecznej implementacji protokołu HTTPS

4. Zakres testów
    a. czas dostępu do bazy danych i zewnętrznych API
    b. stabilność połączeń z zewnętrznym API i bazą danych
    c. poprawność implementacji logiki biznesowej
    d. przystępność interfejsu użytkownika
    e. bezpieczeństwo przechowywanych kluczy i haseł
    f. zgodność integracji:
        i. backend - frontend
        ii. backend - baza danych
        iii. backend - zewnętrzne API
        iv. frontend - serwis uwierzytelniający

5. Założenia i ograniczenia
    a. testowane moduły
        i. część kliencka aplikacji (frontend)
        ii. część serwera aplikacji (backend) wraz z komunikacją z bazą danych
    b. rodzaje przeprowadzanych testów:
        i. testy jednostkowe
        ii. testy e2e
        iii. testy integracyjne
        iv. testy manualne
    c. wszystkie rodzaje testów (oprócz manualnych) są wykonywane:
        i. podczas budowania projektu
        ii. przed mergowaniem do głównej gałęzi
        iii. podczas CI/CD pipeline'u
    d. implementacja projektu wraz z całkowitym możliwym przedziałem czasowym na testowanie aplikacji wynosi 10 tygodni
    e. działanie zewnętrznego serwisu udostępniającego listę opiniowanych produktów jest niezależne od członków zespołu developerskiego, dlatego nie będzie ono testowane
    f. pokrycie kodu wynosi co najmniej 90%
    g. nowe moduły zawierające logikę biznesową muszą posiadać testy jednostkowe przed zmergowaniem do głównej gałęzi projektu
    h. manualne testy części klienckiej aplikacji są wykonywane przed dodaniem nowych funkcjonalności interfejsu użytkownika

6. Interesariusze (lista interesariuszy i ich powiązania z procesem testowym).
    a. Członkowie zespołu developerskiego (niezaangażowani w proces testowania produktu)
    b. Project Manager Krzysztof Stępień - ocena stopnia postępu testowania, tworzenie harmonogramu prac, ocena gotowości produktu

Rejestr ryzyka. Identyfikacja ryzyka, którym odpowiadać będzie testowanie wynikłe z planu.

1. Ryzyka projektowe
   a. Nie zaimplementowanie wszystkich funkcjonalności w zadanym czasie
      I. Harmonogram określający kolejne kroki oraz uwzględniający zapas czasowy na ewentualne opóźnienia
   b. Problemy z integracją funkcjonalności rozwiniętych przez różne grupy projektowe
      I. Przeprowadzanie testów integracyjnych oraz end-to-end sprawdzających działanie zarówno nowo dodanych funkcjonalności jak i tych zaimplementowanych wcześniej
   c. Niedobór bądź brak wymaganych umiejętności wśród członków zespołu do odpowiedniej realizacji funkcjonalności/aplikacji
      I. Odpowiednio wczesne rozpoznanie problemu oraz nauka wymaganych technologii/zdobycie potrzebnej wiedzy/reorganizacja zadań

2. Ryzyka produktowe (jakościowe)
   a. Nieuwzględnienie wszystkich możliwych przypadków w testowaniu i wynikające z tego błędy występujące u użytkownika końcowego produktu
      - Możliwość założenia konta z adresem e-mail, który już istnieje w bazie danych
      - Możliwość wystawienia oceny równej 0 gwiazdek (zakres opinii 1-10)
      - Brak możliwości zmiany rynku

      Proponowane rozwiązanie: Testy end-to-end wszystkich funkcjonalności na końcowym produkcie przeprowadzone przez więcej niż jedną osobę - większa szansa wychwycenia ewentualnych mniej zauważalnych wad.
   b. Końcowe oprogramowanie nie spełnia wszystkich założonych funkcjonalności
      - Brak możliwości wystawienia opinii w formie tekstowej
      - Brak możliwości dodania nowych produktów przez administratora
      - Brak możliwości utworzenia nowego konta przez użytkownika niezalogowanego
      - Brak możliwości przeglądania opinii przez użytkowników

      Proponowane rozwiązanie: Lista wszystkich wymaganych funkcjonalności oraz bieżąca analiza zrealizowanych oraz oczekujących na zrealizowanie funkcjonalności przez wszystkich członków zespołu projektowego oraz testowanie każdej zaimplementowanej funkcjonalności przez parę implementującą daną funkcjonalność.
   c. Nieintuicyjny design UI utrudniający użytkownikom korzystanie z programu
      - Opcje do utworzenia nowego konta/logowania umieszczone w trudnym do zlokalizowania miejscu
      - Nieintuicyjny opcja wyboru interesującej użytkownika kategorii produktu zmuszająca go do przeglądania produktów z wszystkich kategorii
      - Nieoczywiste miejsce wyświetlania komentarzy dla danego produktu
      - Niewidoczna opcja sortowania i filtrowania wyników

      Proponowane rozwiązanie: Pozyskanie zewnętrznych opinii w oparciu zarys/prototyp na wczesnym stadium pracy nad UI oraz konsultacje z innymi członkami zespołu.

d. Niewłaściwe uprawnienia przyznane różnym rodzajom użytkowników (użytkownikom zalogowanym, użytkownikom niezalogowanym, administratorom oraz moderatorom)
    - Brak możliwości wykonania przez administratora czynności użytkownika zalogowanego - np. dodanie opinii o produkcie.
    - Użytkownik zalogowany ma możliwość wykonania funkcji administratorskich, np. ma możliwość usunięcia konta innego użytkownika lub też zatwierdzania nowych produktów do bazy danych
    - Użytkownik niezalogowany nie ma możliwości przeglądania opinii wystawionych przez innych użytkowników
    - Administrator ma uprawienia moderatora - odrzucania oraz zatwierdzania opinii dodanych przez użytkowników

    Proponowane rozwiązanie: Sprawdzenie osobno dla każdego rodzaju użytkownika (niezalogowanego, zalogowanego, administratora oraz moderatora) zarówno dostępności danej funkcji jak i także braku możliwości wykonania danej czynności (sprawdzając zarówno możliwość skorzystania z danej funkcji jak i upewniając się o braku uprawnień niwelujemy ryzyko pominięcia danego aspektu w przypadku któregoś rodzaju użytkownika).
e. Istnienie luk bezpieczeństwa
    - Brak zabezpieczenia hasła
    - Dopuszczenie korzystania ze słabych haseł (np. 5-literowych)
    - Możliwość edycji opinii przez innych użytkowników

    Proponowane rozwiązanie: Stworzenie listy potencjalnych słabych punktów i zapobieganie ich powstawaniu oraz dostosowane przypadków testowych tak, aby uwzględniały weryfikację działania funkcjonalności/aplikacji również w aspekcie bezpieczeństwa.

Strategia testowa. Opisuje podejście testowe do konkretnego projektu.

1. Produkty testowania
    a. Produkty planowania
        i. Poniższy dokument
        ii. Readme.md
        iii. Tablica Kanban
    b. Produkty monitorowania i nadzoru testów
        i. Raporty generowane przez Cypress, Vitest
        ii. CI/CD
        iii. Tablica Kanban
    c. Produkty analizy
        i. Raport pokrycia kodu
        ii. Scenariusze testowania
    d. Produkty projektowania testów
        i. Pliki z testami
        ii. Scenariusze testowania
        iii. Stworzenie środowiska testowego
    e. Produkty implementacji testów

        i. Procedury testowe i kolejność ich wykonywania
        ii. Zestawy testowe
        iii. Harmonogram wykonywania testów
    f. Produkty wykonywania testów
        i. Raporty generowane przez narzędzia testowe
        ii. Raport o wykazanych przez testy defektach
    g. Produkty ukończenia testów
        i. Lista poprawek eliminujących wykryte w testach defekty
2. Narzędzia

    Manualne sprawdzenie funkcjonalności wprowadzone jest jako jeden z etapów kontroli jakości przy integracji funkcjonalności z główną gałęzią.

    Scenariusze testowe dokumentowane będą w wybranym edytorze tekstu (Google Docs).

    a. Testowanie manualne aplikacji.
    b. Testowanie automatyczne
        i. Testy automatyczne jednostkowe
            1. Biblioteka Vitest wraz z React Testing Library
                a. Vitest to środowisko uruchomieniowe testów, odpowiedzialne za uruchamianie, automatyzację i sterowanie środowiskiem testowym (tj. implementacją przeglądarki internetowej).
                    1. React Testing Library zapewnia narzędzia pozwalające na manipulację zawartością treści DOM w przeglądarce oraz budowanie asercji.
                b. Testy automatyczne end-to-end (E2E)
                c. Biblioteka Cypress wraz z Cypress Testing Library
                d. Cypress to narzędzie pozwalające na tworzenie i uruchamianie testów polegających na wykonywaniu czynności na stronie w sposób automatyczny poprzez udostępniane metody pozwalające na interakcje bezpośrednio z uruchomioną instancją przeglądarki internetowej
    c. Jarzmo testowe
        i. happy-dom - implementacja przeglądarki zrealizowana w języku JavaScript, kompatybilna z biblioteką Vitest, umożliwiająca uruchamianie testów jednostkowych
        ii. pełne implementacje przeglądarek internetowych (w przypadku testów end-to-end)
        iii. okrojona, pozbawiona interfejsu graficznego implementacja przeglądarki internetowej oparta o projekt chromium (w przypadku testów end-to-end)
    d. Zarządzanie danymi
        i. W celu pokrycia testami funkcjonalności związanych z danymi dynamicznymi, zastosowano narzędzie do generacji danych losowych, aby odtworzyć model interakcji z użytkownikiem.
3. Techniki projektowania testów
    a. Czarnoskrzynkowe
        i. Testy end-to-end

        ii. Integracyjne
    b. Białoskrzynkowe
        i. Testy jednostkowe

Scenariusze przypadków testowych tworzone są w oparciu o doświadczenie, w szczególności za pomocą techniki zgadywania błędów.

4. Kryteria zakończenia testów
    a. Osiągnięcie odpowiedniego procentowego poziomu pokrycia (określonego w kontekście testowania, sekcji 5.: Założenia i ograniczenia)
    b. Przeprowadzenie testów na wszystkich uwzględnianych w procesie testowania komponentach
    c. Niska gęstość defektów zapewniona przez pozytywny rezultat wszystkich zaprojektowanych testów
5. Metryki
    a. Procentowe pokrycie testowe
    b. Liczba wykrytych i usuniętych defektów
    c. Postęp projektu uwidoczniony na tablicy Kanban
6. Wymagania co do danych testowych
    a. Głównym wymaganiem danych testowych jest odpowiedni dobór do warunków i założeń testowych. Przykładowo, testując formularz logowania, pożądane dane do przypadków weryfikujących poprawność badanego scenariusza powinny spełniać warunki testu tj. testy poprawnego logowania powinny wykorzystywać dane spełniające warunki walidacji, testy odrzucenia logowania zaś wykorzystywać dane niespełniające walidacji.
7. Wymagania co do środowiska testowego.
    a. Baza danych wypełniona przykładowymi danymi spełniającymi wymagania narzucone przez warunki testowe (patrz: punkt 6).
    b. Wymagania techniczne pokrywające się z wymaganiami pełnej wersji aplikacji.
    c. Środowisko testowe możliwie blisko odzwierciedlające specyfikę środowiska produkcyjnego.
    d. Zaślepienie elementów niepodlegających badanemu scenariuszowi testowemu w sposób odzwierciedlający ich funkcjonowanie w pełnej wersji aplikacji.
8. Testy regresji i retesty.
    a. Pełen zestaw testów wykonywany jest każdorazowo po dołączeniu nowej funkcjonalności, tym samym uwzględniając regresję i retesty w procesie testowania.
    b. Integralność procesu będzie zachowana poprzez narzędzia CI/CD oraz bibliotekę Husky, która wykona proces testowy przy każdym commicie.
9. Kryteria zawieszenia i wznowienia testów oraz osoby odpowiedzialne za decyzję odnośnie ich przeprowadzenia (np. warunki pełnej akceptacji, warunkowej akceptacji lub odrzucenia).
    a. Warunki pełnej akceptacji
        i. Testy spełniają warunki zakończenia (patrz: punkt 4) oraz kod zaakceptowany w procesie code review.
    b. Warunkowa akceptacja
        i. Testy spełniają warunki zakończenia, kod niezaakceptowany w procesie code review.

c. Warunki odrzucania
    i. Testy nie spełniają warunków zakończenia.

Personel

1. Role, zadania i odpowiedzialności (opisują, jakie role występują w procesie testowym oraz kto i za co jest odpowiedzialny).
    a. Rozróżniamy dwie główne role: kierownika testów oraz testera, którym przypisujemy następujące zadania i odpowiedzialności:
        i. Kierownik testów:
            1. jest to jeden z liderów zespołu developerskiego
            2. planuje cele testów oraz terminy ich wykonania
            3. na bieżąco aktualizuje plan testów dostosowując go do wydajności czasowej zespołu
            4. koordynuje pracę testerów, pomaga w ich integracji
            5. sprawdza jakość i pokrycie testów przez testerów (wprowadza miary oceniające jakość testów oraz ich pokrycie)
            6. monitoruje i przygotowuje raporty dotyczące postępu zespołu testującego
            7. podejmuje główną decyzję dotyczącą stosowanych bibliotek/środowisk do testów
            8. służy pomocą i radą - jest mentorem zespołu
            9. reprezentuje zespół

        ii. Tester:
            1. może być to członek zespołu developerskiego, który tworzy funkcjonalności w projekcie
            2. tworzy testy jednostkowe oraz integracyjne dla stworzonej przez siebie (lub developera, w wypadku, gdy nie jest członkiem zespołu developerskiego) funkcjonalności
            3. uczestniczy w opracowywaniu planów testów
            4. analizuje dane funkcjonalności pod kątem ich testowalności (wymagania, scenariusze, kryteria akceptacji, przypadki brzegowe)
            5. przygotowuje dane testowe
            6. członek zespołu developerskiego ma za zadanie obowiązkowo pisać testy jednostkowe oraz integracyjne do swoich funkcjonalności
            7. tworzy testy E2E
            8. korzysta z narzędzi ustalonych przez kierownika testów
            9. przegląda testy innych członków zespołu pod kątem ich jakości
            10. dokumentuje odchylenia od oczekiwanych rezultatów po wykonaniu testów oraz informuje zespół developerów o konieczności poprawy funkcjonalności
            11. ewaluuje charakterystyki niefunkcjonalne, przede wszystkim: wydajność, niezawodność

    b. W strukturze naszego personelu występują **pary testujące**, są to zespoły 2-osobowe, złożone z dwóch testerów, którzy wykonują swoje zadania i odpowiedzialności w ramach przypisanych parze funkcjonalności aplikacji.
2. Potrzeby związane z zatrudnieniem (jeśli np. brakuje w zespole osób określonych umiejętnościach).
    a. w zespole developersko-testowym każdy członek potrafi na podstawie dostarczonych materiałów poprawnie testować wytwarzane oprogramowanie, w związku z czym, nie są spodziewane żadne braki kadrowe;
    b. dodatkowo przez wzgląd na naukowo-badawczy charakter projektu nie byłoby możliwe zatrudnienie do zespołu żadnej osoby zewnętrznej;
3. Potrzeby związane ze szkoleniami.
    a. braki w umiejętnościach i posługiwaniu się danym językiem programowania/biblioteką lub narzędziem do testowania będą uzupełniane poprzez współpracę i coaching wewnątrz zespołu;

Czynności testowe

1. Pierwszą czynnością jest stworzenie planu testów, w którym dokładnie zdefiniujemy:
    a. Zakres oraz cele testów
    b. Strategie testowania
    c. zasoby ludzkie i inne
    d. miary do monitorowania i nadzorowania testów
    e. czynności testowe
    f. dokumentacji testów
    g. harmonogram

   Cały plan testów jest poniższym dokumentem.
2. Stworzenie scenariuszy testowych w których zostaną dla każdej funkcjonalności (zgodnie z wymaganiami funkcjonalnymi dostępnymi https://tulodz.sharepoint.com/:w:/s/22-SALUS-PiKGX/EbvN_pCcWkxLuIhHPhwqCSMBmWNichhAw_S2HPBWI45kHg?e=evWeFc) nadany identyfikator scenariusza, nazwa, opis, wymagane narzędzia oraz powiązania z innymi artefaktami.
3. Stworzenie przypadków testowych dla opisanych wcześniej scenariuszy testowych. Każdy przypadek powinien zawierać identyfikator, cel, priorytet, opisane warunki wstępne, dane testowe, oczekiwany wynik, warunki wyjściowe oraz dokładny opis przebiegu testu.
4. Implementacja testów dla każdego stworzonego przypadku używając biblioteki React Testing Librarty oraz środowiska testowego Vitest. Należy przed implementacją testów przygotować dane testowe (np. użytkownik zalogowany, produkty) i czy zostały one dobrze załadowane do Vitest.
5. Wykonanie testów i analiza ich wyników. Sprawdzamy czy otrzymane wyniki są zgodne z oczekiwanymi (zapisanymi w danym przypadku testowym), jeżeli nie, należy przeanalizować anomalie w celu ustalenia ich przyczyny, stworzenie raportu oraz naprawienie anomalii. Dla każdorazowego wykonania testów, należy utworzyć raport w którym trzeba określić identyfikator raportu, datę wykonania oraz tabele testów (Scenariusz, Przypadki testowe, Oczekiwane wyniki, Aktualny wynik, Status, Komentarz).

6. Stworzenie raportu zbiorczego, który zawiera podsumowanie, odchylanie od planu, ocenę poziomu spełnienia wymagań określonych w planie testowym, czynniki opóźnień jeżeli wystąpiły, ryzyka rezydualne, raport z aktualnego stanu testów (opisane w punkcie 5), reużywalne produkty procesu testowego oraz wnioski całego procesu testowego.

Harmonogram

1. **Kamień milowy**
    - Zadanie: Przygotowanie Planu Testów
    - Termin: 17.10.2022
2. **Kamień milowy**
    - Zadanie: Zaprojektowanie scenariuszy testowych
        i. Warunek wstępny: zidentyfikowanie cech aplikacji, nadających się do przetestowania
    - Termin: 31.10.2022
3. **Kamień milowy**
    - Zadanie: Przygotowanie Przypadków Testowych
    - Termin: 14.11.2022
4. **Kamień milowy**
    - Zadanie:
        i. Implementacja testów
        ii. Wykonywanie testów
        iii. Monitorowanie i nadzorowanie testów
        iv. Ukończenie testów

      Efekt finalny przedstawić w formie raportu z wykonanych testów
    - Termin: 28.11.2022

Komunikacja

W tym projekcie zespół testerów składa się głównie z developerów, którzy zajmują się również testowaniem zaimplementowanych przez siebie funkcjonalności. Oprócz tego jest też kilku testerów, którzy nie są developerami.

Informacjami, które będą przekazywane pomiędzy testerami a developerami są:

- ➢ statusy implementacji funkcjonalności - to czy dana funkcjonalność została w pełni zaimplementowana przez developera
- ➢ statusy wykonanych testów przez testera oraz w przypadku niepowodzenia testu, informację dla developera, co należy poprawić

Główny kanał komunikacji, za pomocą którego te informacje będą przekazywane, to **tablica zadań** w repozytorium projektu na portalu GitLab. Drugorzędnym w tej kwestii kanałem komunikacji jest **arkusz excel** z wypisanymi funkcjonalnościami i przyporządkowanymi do nich zespołami. Jednakże ten kanał komunikacji dotyczy bardziej

developerów, gdyż jego głównym zadaniem jest przyporządkowanie funkcjonalności do zespołu. Developerzy po zakończeniu pracy nad zadaniem powinni zmienić jego status na ***DONE***.

Jest to znak dla testerów, że zadanie jest gotowe i można przystąpić do testowania. Jeżeli testy wykażą nieprawidłowości, status zadania powinien zostać zmieniony na ***FAILED***, a w opisie powinien znaleźć się raport z testu oraz opis sposobu odtworzenia błędu. Takie zadanie trafia z powrotem do programisty, a po naprawieniu błędu do testera. Jeżeli tester nie wykryje żadnych nieprawidłowości to zadanie otrzymuje status ***VERIFIED***.

Wykorzystywanie tablicy zadań w repozytorium projektu na portalu GitLab oraz arkusza excel pozwala na monitorowanie postępu prac. W łatwy sposób każdy członek zespołu może zweryfikować postępy, a tym samym zaobserwować ile zadań zostało zweryfikowanych, a w ilu zadaniach wykryto nieprawidłowości.

Dodatkowym kanałem komunikacji dla całej grupy projektowej, pozwalający w szybki i sprawny sposób poinformować testera czy developera o postępie procesu wykonywania czy weryfikowania zadania jest specjalnie stworzona konwersacja grupowa w aplikacji Messenger. Wymienione wyżej kanały komunikacji są jednak priorytetowe. To one są jasną i przejrzystą formą weryfikacji postępu testowania, a także przechowują informacje o testach w łatwo dostępny sposób.