**Załącznik nr 4**

**SPECYFIKACJA TECHNICZNA  
SYSTEMU TESTGEN**

Platforma do automatyzacji testowania oprogramowania  
z wykorzystaniem sztucznej inteligencji

**SPIS TREŚCI**

[**SPECYFIKACJA TECHNICZNA SYSTEMU TESTGEN** 1](#_Toc211710810)

[**SPIS TREŚCI** 2](#_Toc211710811)

[1. WPROWADZENIE 4](#_Toc211710812)

[1.1. Cel dokumentu 4](#_Toc211710813)

[1.2. Zakres dokumentu 4](#_Toc211710814)

[1.3. Odbiorcy dokumentu 4](#_Toc211710815)

[1.4. Konwencje i oznaczenia 5](#_Toc211710816)

[2. OPIS OGÓLNY SYSTEMU TESTGEN 6](#_Toc211710817)

[2.1. Wizja produktu 6](#_Toc211710818)

[2.1.1. Główne komponenty systemu 6](#_Toc211710819)

[2.2. Cele biznesowe 7](#_Toc211710820)

[2.3. Model biznesowy 7](#_Toc211710821)

[2.3.1. Model SaaS (Software as a Service) 7](#_Toc211710822)

[2.3.2. Model On-Premises 7](#_Toc211710823)

[2.4. Kluczowe innowacje 8](#_Toc211710824)

[2.4.1. Innowacja produktowa 8](#_Toc211710825)

[2.4.2. Wyróżniki konkurencyjne 8](#_Toc211710826)

[2.4.3. Analiza konkurencji 8](#_Toc211710827)

[2.4.4. Przewaga TestGEN 9](#_Toc211710828)

[2.5. Kluczowe cechy do implementacji 10](#_Toc211710829)

[2.5.1. Cechy i funkcjonalności produktu przesądzające o innowacyjności produktu, które MUSZĄ zostać zaimplementowane 10](#_Toc211710830)

[2.5.2. Nowe cechy i funkcjonalności produktu decydujące o przydatności użytkowej produktu jakie system MUSI implementować 12](#_Toc211710831)

[2.5.3. Podział projektu na zadania 13](#_Toc211710832)

[3. WYMAGANIA FUNKCJONALNE 15](#_Toc211710833)

[3.1. Silnik TestSpark - komponenty podstawowe 15](#_Toc211710834)

[3.1.1. Architektura silnika TestSpark 15](#_Toc211710835)

[3.1.2. Technologia AI do automatycznej generacji przypadków testowych 15](#_Toc211710836)

[3.1.3. Podmoduł generujący kod testowy 15](#_Toc211710837)

[3.2. Moduły analizy i generowania testów 16](#_Toc211710838)

[3.2.1. Przegląd modułów 16](#_Toc211710839)

[3.3. Moduły bezpieczeństwa 16](#_Toc211710840)

[3.3.1. Przegląd modułów bezpieczeństwa 16](#_Toc211710841)

[3.4. Interfejs użytkownika 16](#_Toc211710842)

[3.4.1. Wymagania ogólne GUI 16](#_Toc211710843)

[3.4.2. Wymagania dostępności WCAG 2.1 17](#_Toc211710844)

[4. SPECYFIKACJA MODUŁÓW 18](#_Toc211710845)

[4.1. Moduł generowania przypadków testowych z kodu źródłowego 19](#_Toc211710846)

[4.1.1. Opis ogólny modułu 19](#_Toc211710847)

[4.1.2. Funkcjonalność automatyzacji analizy kodu źródłowego 19](#_Toc211710848)

[4.1.3. Funkcjonalność generowania przypadków testowych i kodu testowego (dla back-endu, API) 21](#_Toc211710849)

[4.1.4. Funkcjonalność testów regresji kodu i GUI 25](#_Toc211710850)

[4.1.5. Wykorzystane mechanizmy i technologie 30](#_Toc211710851)

[4.2. Moduł generowania przypadków testowych z dokumentacji i podręczników użytkownika 32](#_Toc211710852)

[4.2.1. Opis ogólny modułu 32](#_Toc211710853)

[4.2.2. Funkcjonalność automatyzacji analizy dokumentacji technicznej i biznesowej 33](#_Toc211710854)

[4.2.3. Funkcjonalność generowania przypadków testowych na podstawie dokumentacji technicznej i dokumentacji biznesowej 37](#_Toc211710855)

[4.2.4. Wykorzystane mechanizmy i technologie 41](#_Toc211710856)

[4.3. Moduł generowania przypadków testowych z UML i GUI 42](#_Toc211710857)

[4.3.1. Opis ogólny modułu 42](#_Toc211710858)

[4.3.2. Funkcjonalność automatyzacji analizy diagramów UML i generowania opisowych przypadków testowych 42](#_Toc211710859)

[4.3.3. Funkcjonalność automatyzacji analizy interfejsów użytkownika (GUI) i generowania przypadków testowych (opisowych lub kodu makra do automatycznego testu) 47](#_Toc211710860)

[4.3.4. Funkcjonalność generowania przypadków testowych i kodu testowego (GUI) 49](#_Toc211710861)

[4.3.5. Funkcjonalność testów regresji GUI 50](#_Toc211710862)

[4.3.6. Wykorzystane mechanizmy i technologie 54](#_Toc211710863)

[4.4. Moduł inteligentnej analizy luk w kodzie oprogramowania i generowania rekomendacji zabezpieczeń 55](#_Toc211710864)

[4.4.1. Opis ogólny modułu i wykorzystane technologie 56](#_Toc211710865)

[4.4.2. Funkcjonalność analizy luk bezpieczeństwa kodu i generowanie przypadków testów bezpieczeństwa 57](#_Toc211710866)

[4.4.3. Funkcjonalność aktualizacji systemu 65](#_Toc211710867)

[4.4.4. Wykorzystane mechanizmy i technologie 67](#_Toc211710868)

[4.5. Moduł analizy luk funkcjonalnych w kodzie i generowania inteligentnych rekomendacji rozszerzeń 68](#_Toc211710869)

[4.5.1. Opis ogólny modułu 68](#_Toc211710870)

[4.5.2. Funkcjonalność analizy zgodności kodu z dokumentacją 68](#_Toc211710871)

[4.5.3. Funkcjonalność podpowiedzi dotyczących błędów 72](#_Toc211710872)

[4.5.4. Funkcjonalność podpowiedzi dotyczących nowych funkcjonalności 76](#_Toc211710873)

[4.5.5. Wykorzystane mechanizmy i technologie 79](#_Toc211710874)

[4.6. Graficzny interfejs użytkownika (GUI) integracja z systemami płatności 80](#_Toc211710875)

[4.6.1. GUI 80](#_Toc211710876)

[5. ARCHITEKTURA SYSTEMU 81](#_Toc211710877)

[5.1. Architektura ogólna 81](#_Toc211710878)

[5.1.1. Model architektoniczny 81](#_Toc211710879)

[5.1.2. Diagram architektury wysokopoziomowej 81](#_Toc211710880)

[5.1.3. Główne warstwy systemu 82](#_Toc211710881)

[5.2. Komponenty backendowe 83](#_Toc211710882)

[5.2.1. Stos technologiczny backend 83](#_Toc211710883)

[5.2.2. Code Analysis Service 84](#_Toc211710884)

[5.2.3. Documentation Analysis Service 85](#_Toc211710885)

[5.2.4. Test Generation Service 85](#_Toc211710886)

[5.3. Komponenty frontendowe 85](#_Toc211710887)

[5.3.1. Web Application (React) 85](#_Toc211710888)

[5.3.2. Mobile Application (Flutter) 87](#_Toc211710889)

[5.4. Integracje zewnętrzne 87](#_Toc211710890)

[5.4.1. Integracja z systemami kontroli wersji 87](#_Toc211710891)

[5.4.2. Integracja z systemami zarządzania testami 87](#_Toc211710892)

[5.4.3. Integracja z systemami CI/CD 88](#_Toc211710893)

[5.4.4. Integracja z systemami płatności 88](#_Toc211710894)

[6. TECHNOLOGIE I NARZĘDZIA 90](#_Toc211710895)

[6.1. Stos technologiczny 90](#_Toc211710896)

[6.1.1. Backend Technologies 90](#_Toc211710897)

[6.1.2. Frontend Technologies 90](#_Toc211710898)

[6.2. Biblioteki i frameworki AI/ML 91](#_Toc211710899)

[6.2.1. Natural Language Processing (NLP) 91](#_Toc211710900)

[6.2.2. Machine Learning Frameworks 92](#_Toc211710901)

[6.2.3. Modele językowe wykorzystywane 92](#_Toc211710902)

[6.3. Narzędzia analizy kodu 93](#_Toc211710903)

[6.3.1. Parsery AST i analizatory statyczne 93](#_Toc211710904)

[6.3.2. Narzędzia SAST (Static Application Security Testing) 93](#_Toc211710905)

[6.3.3. Narzędzia DAST (Dynamic Application Security Testing) 94](#_Toc211710906)

[6.3.4. Narzędzia SCA (Software Composition Analysis) 95](#_Toc211710907)

[6.4. Frameworki testowe wspierane przez TestGEN 95](#_Toc211710908)

[6.4.1. Frameworki testowe backend 95](#_Toc211710909)

[6.4.2. Frameworki testowe frontend/E2E 96](#_Toc211710910)

[6.4.3. API Testing Frameworks 97](#_Toc211710911)

[7. WYMAGANIA NIEFUNKCJONALNE 99](#_Toc211710912)

[7.1. Wydajność 99](#_Toc211710913)

[7.1.1. Wymagania czasowe 99](#_Toc211710914)

[7.1.2. Wymagania throughput 100](#_Toc211710915)

[7.2. Skalowalność 100](#_Toc211710916)

[7.2.1. Skalowanie horyzontalne 100](#_Toc211710917)

[7.2.2. Limity systemu 100](#_Toc211710918)

[7.3. Bezpieczeństwo 101](#_Toc211710919)

[7.3.1. Uwierzytelnianie i autoryzacja 101](#_Toc211710920)

[7.3.2. Szyfrowanie danych 101](#_Toc211710921)

[7.3.3. Bezpieczeństwo API 101](#_Toc211710922)

[7.3.4. Audyt i monitoring bezpieczeństwa 102](#_Toc211710923)

[7.3.5. Zgodność z regulacjami 102](#_Toc211710924)

[7.4. Dostępność (WCAG 2.1) 102](#_Toc211710925)

[7.4.1. Poziom zgodności 102](#_Toc211710926)

[7.4.2. Szczegółowe wymagania WCAG 102](#_Toc211710927)

[7.4.3. Narzędzia do testowania dostępności 104](#_Toc211710928)

[7.4.4. Dodatkowe funkcje dostępności 104](#_Toc211710929)

[7.5. Niezawodność i dostępność (Availability) 104](#_Toc211710930)

[7.5.1. Uptime i SLA 104](#_Toc211710931)

[7.5.2. Backup i disaster recovery 105](#_Toc211710932)

[7.5.3. Monitoring i alerting 105](#_Toc211710933)

[7.6. Utrzymanie i wsparcie 105](#_Toc211710934)

[7.6.1. Poziomy wsparcia 106](#_Toc211710935)

[7.6.2. Dokumentacja 106](#_Toc211710936)

[8. HARMONOGRAM REALIZACJI 107](#_Toc211710937)

[8.1. Zadanie A - Silnik TestSpark i generowanie przypadków testowych 107](#_Toc211710938)

[8.1.1. Szczegóły zadania 107](#_Toc211710939)

[8.1.2. Kluczowe milestone'y 108](#_Toc211710940)

[8.1.3. Kryteria akceptacji Zadania A 109](#_Toc211710941)

[8.2. Zadanie B - Rozbudowa pod kątem audytów bezpieczeństwa 109](#_Toc211710942)

[8.2.1. Szczegóły zadania 109](#_Toc211710943)

[8.2.2. Kluczowe milestone'y 110](#_Toc211710944)

[8.2.3. Kryteria akceptacji Zadania B 110](#_Toc211710945)

[8.3. Zadanie C - GUI, płatności i finalizacja 111](#_Toc211710946)

[8.3.1. Szczegóły zadania 111](#_Toc211710947)

[8.3.2. Kluczowe milestone'y 112](#_Toc211710948)

[8.3.3. Kryteria akceptacji Zadania C 113](#_Toc211710949)

[8.4. Podsumowanie harmonogramu 113](#_Toc211710950)

[9. DOKUMENTACJA TECHNICZNA 115](#_Toc211710951)

[9.1. Dokumentacja API 115](#_Toc211710952)

[9.1.1. Standardy dokumentacji API 115](#_Toc211710953)

[9.1.2. Przykładowa dokumentacja endpointu 115](#_Toc211710954)

[9.1.3. Główne grupy endpointów API 117](#_Toc211710955)

[9.1.4. Webhooks 118](#_Toc211710956)

[9.2. Dokumentacja użytkownika 119](#_Toc211710957)

[9.2.1. Struktura dokumentacji użytkownika 119](#_Toc211710958)

[9.2.2. Przykładowa strona z dokumentacji użytkownika 121](#_Toc211710959)

[9.3. Dokumentacja wdrożeniowa 122](#_Toc211710960)

[9.3.1. Dokumentacja instalacji on-premises 122](#_Toc211710961)

[9.3.2. Przykładowa instrukcja instalacji 123](#_Toc211710962)

[9.3.3. Kubernetes Deployment 125](#_Toc211710963)

[9.4. Materiały szkoleniowe 125](#_Toc211710964)

[9.4.1. Program szkoleń 125](#_Toc211710965)

[9.5. Dane wejściowe dla testów funkcjonalności w trakcie developmentu i wyniki działania 126](#_Toc211710966)

[10. Rozwój graficznego interfejsu użytkownika (GUI) oraz systemami płatności 127](#_Toc211710967)

[10.1. Opracowanie GUI zgodnego z WCAG 2.1 i integracja systemu płatności 127](#_Toc211710968)

[10.1.1. Interfejs UI 127](#_Toc211710969)

[10.1.2. Integracja z modułem płatności 128](#_Toc211710970)

[11. ZAŁĄCZNIKI 130](#_Toc211710971)

[11.1. Słownik pojęć 130](#_Toc211710972)

[10.2. Przykłady kodów źródłowych 132](#_Toc211710973)

[10.2.1. Przykład 1: Kod z podatnością SQL Injection 132](#_Toc211710974)

[10.2.2. Przykład 2: Kod z podatnością XSS 133](#_Toc211710975)

# **1.** WPROWADZENIE

## **1.1.** Cel dokumentu

Niniejszy dokument stanowi kompleksową specyfikację techniczną systemu TestGEN - innowacyjnej platformy do automatyzacji testowania oprogramowania z wykorzystaniem sztucznej inteligencji. Dokument został przygotowany jako podstawowe narzędzie dla zespołów deweloperskich, architektów systemowych oraz analityków biznesowych odpowiedzialnych za implementację systemu.

Specyfikacja zawiera szczegółowe opisy wszystkich komponentów systemu, wymagań funkcjonalnych i niefunkcjonalnych, architektury technicznej oraz metodologii implementacji poszczególnych modułów. Dokument uwzględnia pełną ścieżkę rozwoju produktu od koncepcji po wdrożenie produkcyjne.

## **1.2.** Zakres dokumentu

Dokument obejmuje następujące obszary:

* Opis ogólny systemu TestGEN i jego celów biznesowych
* Szczegółową specyfikację wymagań funkcjonalnych wszystkich modułów
* Architekturę techniczną systemu z uwzględnieniem komponentów AI/ML
* Specyfikację interfejsów użytkownika zgodnych ze standardem WCAG 2.1
* Opis technologii, narzędzi i bibliotek wykorzystywanych w systemie
* Wymagania niefunkcjonalne dotyczące wydajności, skalowalności i bezpieczeństwa
* Harmonogram realizacji projektu z podziałem na etapy
* Wytyczne dotyczące dokumentacji technicznej i użytkownika

## **1.3.** Odbiorcy dokumentu

Dokument jest przeznaczony dla następujących grup odbiorców:

* **Programiści i developerzy** - implementujący poszczególne moduły systemu
* **Architekci systemowi** - projektujący strukturę i komponenty platformy
* **Analitycy biznesowi** - definiujący wymagania i przypadki użycia
* **Testerzy** - weryfikujący poprawność implementacji
* **Specjaliści AI/ML** - implementujący algorytmy uczenia maszynowego
* **DevOps engineers** - odpowiedzialni za wdrożenie i infrastrukturę
* **Kierownicy projektów** - zarządzający realizacją projektu

## **1.4.** Konwencje i oznaczenia

W dokumencie stosowane są następujące konwencje:

**Informacja:** Elementy oznaczone w ten sposób zawierają dodatkowe informacje kontekstowe.

**Uwaga:** Elementy oznaczone w ten sposób wymagają szczególnej uwagi podczas implementacji.

**Ostrzeżenie:** Elementy oznaczone w ten sposób wskazują na krytyczne wymagania lub ograniczenia.

**Wymaganie:** Elementy oznaczone w ten sposób definiują obligatoryjne wymagania systemu.

Fragmenty kodu źródłowego prezentowane są w ramkach:

def example\_function(): """Przykładowa funkcja demonstracyjna""" return "Hello, TestGEN!"

Nazwy klas, metod i zmiennych w tekście: ClassExample, method\_name(), variable\_name

# **2.** OPIS OGÓLNY SYSTEMU TESTGEN

## **2.1.** Wizja produktu

Projekt TestGEN to innowacyjne rozwiązanie dostępne w modelu SaaS oferujące subskrypcję w ramach dwóch pakietów cenowych dostosowane do potrzeb użytkowników, a także w wersji instalacyjnej on-premises dla firm wykorzystujące sztuczną inteligencję (uczenie maszynowe i sieci neuronowe) do automatyzacji procesu testów oprogramowania.

Głównym celem systemu jest automatyzacja analizy dokumentacji technicznej, kodu źródłowego, diagramów UML oraz interfejsów użytkownika (GUI) i generowanie na tej podstawie przypadków testowych oraz kodu testowego dla popularnych frameworków, takich jak JUnit, NUnit czy pytest.

System będzie mógł obsługiwać kod, diagramy UML oraz testy GUI dla aplikacji desktopowych i mobilnych, webowych oraz oprogramowanie dla urządzeń IoT.

### **2.1.1.** Główne komponenty systemu

TestGEN składa się z następujących kluczowych komponentów:

1. **Silnik TestSpark** - autorski silnik AI do automatycznego generowania przypadków testowych i kodów, wykorzystujący modele uczenia maszynowego do analizy kodu i dokumentacji
2. **Moduły analizy**:
   * Moduł analizy kodu źródłowego
   * Moduł analizy dokumentacji technicznej
   * Moduł analizy diagramów UML
   * Moduł analizy interfejsów GUI
3. **Moduły generowania testów**:
   * Generator testów jednostkowych
   * Generator testów integracyjnych
   * Generator testów systemowych
   * Generator testów regresji
4. **Moduły bezpieczeństwa**:
   * Moduł audytu bezpieczeństwa
   * Moduł detekcji podatności (SQL injection, XSS, błędy autoryzacji)
   * Moduł analizy zgodności kodu z dokumentacją
5. **Graficzny interfejs użytkownika (GUI)** - intuicyjny interfejs zgodny z WCAG 2.1
6. **System płatności** - moduł obsługi subskrypcji i rozliczeń

## **2.2.** Cele biznesowe

TestGEN realnie odpowiada na zidentyfikowane potrzeby rynku:

* **Redukcja czasu testów o 60%** - poprzez automatyzację procesu generowania przypadków testowych
* **Automatyczne wykrywanie błędów i podatności (50-70% przyspieszenia)** - dzięki zaawansowanym mechanizmom AI
* **Automatyczne sugerowanie funkcjonalności** - redukujące koszty manualnej analizy
* **Poprawa jakości oprogramowania** - poprzez kompleksową analizę zgodności implementacji z dokumentacją
* **Poprawa bezpieczeństwa oprogramowania –** system automatycznie wyszukuje podatności w kodzie (np. SQL INjection) i sugeruje podejścia do ich usunięia4

## **2.3.** Model biznesowy

### **2.3.1.** Model SaaS (Software as a Service)

System TestGEN będzie dostępny w modelu subskrypcyjnym z dwoma pakietami cenowymi:

|  |  |  |
| --- | --- | --- |
| **Pakiet** | **Funkcjonalności** | **Grupa docelowa** |
| **Pakiet Basic** | * Analiza kodu źródłowego * Generowanie testów jednostkowych * Podstawowa analiza bezpieczeństwa * Limit 100 testów/miesiąc | Małe firmy, freelancerzy, startupy |
| **Pakiet Professional** | * Wszystkie funkcjonalności Basic * Analiza dokumentacji i UML * Zaawansowana analiza bezpieczeństwa * Testy integracyjne i systemowe * Integracja z CI/CD * Unlimited testy | Średnie i duże przedsiębiorstwa |

### **2.3.2.** Model On-Premises

Dla organizacji wymagających pełnej kontroli nad danymi i infrastrukturą, TestGEN będzie dostępny w wersji instalacyjnej on-premises z licencją perpetualną lub roczną.

## **2.4.** Kluczowe innowacje

### **2.4.1.** Innowacja produktowa

TestGEN to uniwersalne narzędzie testerskie, które dzięki zastosowaniu sztucznej inteligencji, uczenia maszynowego oraz technologii analizy kodu umożliwia automatyczne generowanie i uruchamianie testów bez konieczności ich manualnego przygotowywania. System będzie analizował kod źródłowy, dokumentację techniczną, diagramy UML, interfejsy użytkownika (GUI) oraz strukturę API, tworząc spójne, zoptymalizowane scenariusze testowe.

### **2.4.2.** Wyróżniki konkurencyjne

Rozwiązanie wyróżnia się na tle dostępnych narzędzi kilkoma kluczowymi aspektami:

* **Zaawansowana analiza kodu** - TestGEN automatycznie identyfikuje rozbieżności względem dokumentacji, co pozwala wykrywać potencjalne błędy przed wdrożeniem systemu.
* **Automatyczna generacja testów API, GUI i aplikacji** - system generuje testy jednostkowe, systemowe, integracyjne i regresji dla różnorodnych środowisk.
* **Moduł analizy bezpieczeństwa** - wykrywa podatności, takie jak SQL injection, XSS, błędy autoryzacji.
* **Mechanizmy podpowiedzi dotyczące błędów i nowych funkcjonalności** - TestGEN nie tylko wykrywa błędy, ale sugeruje ich optymalne rozwiązania oraz potencjalne ulepszenia aplikacji.
* **Integracja z systemami CI/CD** - automatyzacja testowania w procesie wdrażania, zapewniająca płynne aktualizacje: GitHub Actions, Jenkins, Azure DevOps.
* **Elementy dostępności zgodne z WCAG 2.1** - m.in. możliwość kontrastowania i powiększania czcionki, ułatwiająca korzystanie osobom niedowidzącym.

### **2.4.3.** Analiza konkurencji

TestGEN znacząco różni się od dostępnych rozwiązań zarówno w Polsce, jak i na świecie. W przeciwieństwie do klasycznych narzędzi testowych, takich jak Selenium, JUnit, TestComplete, Ranorex czy Tosca, TestGEN nie ogranicza się jedynie do testowania interfejsu użytkownika czy jednostkowych testów kodu.

|  |  |  |
| --- | --- | --- |
| **Narzędzie** | **Główne funkcje** | **Ograniczenia** |
| **Selenium** | * Automatyzacja testów webowych * WebDriver * Multi-browser support | Brak analizy kodu, brak testowania bezpieczeństwa |
| **JUnit** | * Testy jednostkowe Java * Parametryzacja testów * Integracja CI/CD | Brak analizy dokumentacji, brak analizy bezpieczeństwa |
| **TestComplete** | * Testy UI (desktop, web, mobile) * Keyword-driven testing * Integracja DevOps | Wymaga licencji, brak analizy kodu |
| **GitHub Copilot X** | * Asystent AI * Generowanie testów jednostkowych | Brak analizy systemowej, brak testów bezpieczeństwa |
| **TestGEN** | * Kompleksowa analiza kodu, dokumentacji, UML, GUI * Automatyczne generowanie testów wszystkich typów * Zaawansowana analiza bezpieczeństwa * Wykrywanie luk funkcjonalnych * Inteligentne rekomendacje | - |

### **2.4.4.** Przewaga TestGEN

TestGEN wyróżnia się dzięki zastosowaniu zaawansowanych technik uczenia maszynowego, w tym rozproszonych agentów SLM. System generuje przypadki testowe zarówno dla backendu (API), jak i frontend (GUI), co czyni go bardziej wszechstronnym niż konkurencja.

Dodatkowo analizuje szerokie spektrum danych wejściowych, takich jak:

* Kod źródłowy systemu
* Dokumentacja techniczna i użytkownika (np. diagramy UML)
* Przykładowe dane wejściowe i wyjściowe
* Makiety GUI lub ekrany działającego interfejsu

TestGEN nie tylko generuje przypadki testowe i ew. kod do ich uruchamiania ale także:

* testuje kod
* ale także usprawnia rozwój oprogramowania
  + identyfikując luki
  + sugerując poprawki oraz nowe funkcjonalności
* oferuje automatyzację testowania
* analizuje kodu
* usprawnia testy bezpieczeństwa

## **2.5.** Kluczowe cechy do implementacji

### **2.5.1.** Cechy i funkcjonalności produktu przesądzające o innowacyjności produktu, które MUSZĄ zostać zaimplementowane

(API), jak i frontend (GUI), co czyni go bardziej wszechstronnym niż konkurencja.

#### **2.5.1.1.** Nowość i innowacyjność TestGEN na rynku krajowym i światowym

TestGEN to przełomowe narzędzie do automatyzacji procesu testowania oprogramowania, które znacząco różni się od dostępnych na rynku rozwiązań zarówno w Polsce, jak i na świecie. W przeciwieństwie do klasycznych narzędzi testowych, takich jak Selenium, JUnit, TestComplete, Ranorex czy Tosca, TestGEN nie ogranicza się jedynie do testowania interfejsu użytkownika czy jednostkowych testów kodu. Jego innowacyjność polega na pełnej automatyzacji analizy kodu, dokumentacji technicznej oraz zgodności z wymaganiami biznesowymi, co czyni go unikalnym rozwiązaniem w skali globalnej.

#### **2.5.1.2.** Analiza konkurencyjnych rozwiązań – porównanie funkcjonalności

Na rynku istnieje wiele narzędzi wspierających testowanie oprogramowania, jednak żadne z nich nie oferuje kompleksowej analizy systemu jako całości. Większość dostępnych narzędzi koncentruje się na wybranych aspektach testowania, takich jak testy jednostkowe, testowanie interfejsu użytkownika czy automatyzacja procesów testowych.

#### **2.5.1.3.** Automatyczna analiza kodu

TestGEN wyróżnia się automatyczną analizą kodu, która pozwala na wykrywanie błędów jeszcze przed uruchomieniem testów. Narzędzia takie jak JUnit i Selenium skupiają się na testowaniu kodu poprzez uruchamianie przypadków testowych, ale nie posiadają mechanizmów analizy struktury kodu i jego zgodności z dokumentacją. TestGEN automatycznie identyfikuje niespójności między kodem a dokumentacją, co pozwala na eliminację błędów jeszcze na etapie projektowania.

#### **2.5.1.4.** Generowanie przypadków testowych na podstawie UML

TestGEN jest jedynym narzędziem, które potrafi automatycznie generować przypadki testowe na podstawie diagramów UML. Konkurencyjne rozwiązania, takie jak TestComplete czy Ranorex, wymagają ręcznego definiowania scenariuszy testowych, co jest czasochłonne i podatne na błędy. TestGEN analizuje strukturę aplikacji i generuje kompletne scenariusze testowe, eliminując konieczność ich ręcznego tworzenia.

#### **2.5.1.5.** Moduł analizy bezpieczeństwa

TestGEN posiada zaawansowany moduł analizy bezpieczeństwa, który identyfikuje podatności w kodzie, takie jak SQL injection, XSS czy błędy autoryzacji. Narzędzia takie jak Selenium czy JUnit nie oferują funkcjonalności związanych z testowaniem bezpieczeństwa, co oznacza, że programiści muszą korzystać z dodatkowych narzędzi do analizy podatności. TestGEN integruje testy bezpieczeństwa w jednym systemie, co pozwala na kompleksową ochronę aplikacji.

#### **2.5.1.6.** Inteligentna analiza luk w kodzie i rekomendacje zabezpieczeń

TestGEN wprowadza innowacyjny moduł analizy luk w kodzie, który pozwala na automatyczne wykrywanie podatności oraz generowanie rekomendacji zabezpieczeń w danym zakresie. W przeciwieństwie do innych narzędzi testowych, TestGEN nie tylko wykrywa błędy, ale także sugeruje skuteczne działania naprawcze. System analizuje kod źródłowy, identyfikuje nieprawidłowe filtrowanie danych wejściowych, brak tokenów CSRF, błędy w logice autoryzacyjnej, a następnie proponuje konkretne zmiany w kodzie, eliminujące wykryte zagrożenia.

#### **2.5.1.7.** Inteligentna analiza luk funkcjonalnych i rekomendacje nowych funkcjonalności

TestGEN nie tylko testuje oprogramowanie, ale także aktywnie wspiera jego rozwój, identyfikując niewdrożone funkcjonalności, luki w procesach oraz pominięte elementy. System analizuje kod źródłowy, dokumentację techniczną, diagramy UML oraz przypadki testowe, aby wskazać brakujące funkcje.

#### **2.5.1.8.** Porównywanie kodu, dokumentacji i interfejsów graficznych

TestGEN umożliwia automatyczne wykrywanie zmian wskazując potencjalne błędy regresji. W przeciwieństwie do klasycznych narzędzi testowych, które skupiają się na pojedynczych fragmentach kodu, TestGEN analizuje aplikację identyfikując niespójności między wersjami kodu.

#### **2.5.1.9.** Automatyczne wykrywanie brakujących funkcjonalności

TestGEN nie tylko testuje kod, ale również identyfikuje brakujące funkcjonalności, które mogą zostać wdrożone w celu poprawy spójności i użyteczności aplikacji. System analizuje dokumentację, kod źródłowy oraz przypadki testowe, wskazując np. walidacje, pominięte przypadki brzegowe oraz niedoprecyzowane procesy biznesowe.

Dynamiczna aktualizacja systemu

#### **2.5.1.10.** Dynamiczna aktualizacja systemu

TestGEN nie jest narzędziem statycznym – jego zdolność do ciągłego samodoskonalenia umożliwia analizowanie i przyswajanie nowo odkrytych podatności. System będzie regularnie aktualizowany, aby wykrywać zagrożenia, które wcześniej były nieznane. Dzięki temu właściciele oprogramowania mogą ponownie przeprowadzać analizy, aby zabezpieczyć swoje systemy przed najnowszymi zagrożeniami.

#### **2.5.1.11.** Potwierdzenie skuteczności i przewagi TestGEN

Podczas inkubacji TestGEN przeprowadzono testy skuteczności, które wykazały, że system redukuje czas przygotowania testów do 60% w porównaniu do tradycyjnych metod testowania. Dodatkowo, dzięki zastosowaniu sztucznej inteligencji, TestGEN jest w stanie automatycznie wykrywać błędy w kodzie, co eliminuje konieczność manualnej analizy.

### **2.5.2.** Nowe cechy i funkcjonalności produktu decydujące o przydatności użytkowej produktu jakie system MUSI implementować

#### **2.5.2.1.** Automatyzacja testowania

Automatyzacja testowania na niespotykaną dotąd skalę. TestGEN będzie przełomowym narzędziem, które zrewolucjonizuje testowanie oprogramowania, eliminując ręczne przygotowywanie scenariuszy testowych. System automatycznie generuje testy na podstawie kodu, dokumentacji i GUI, co znacząco przyspieszy proces i zwiększy jego efektywność. Korzyść dla użytkownika: Testerzy oszczędzą czas, skupiając się na analizie wyników, co przyspieszy wdrażanie produktów i poprawi ich jakość.

#### **2.5.2.2.** Wszechstronność

Wszechstronność – testowanie backendu, frontendu i API. TestGEN zapewni pełne pokrycie testowe dla warstw aplikacji, przekraczając ograniczenia tradycyjnych narzędzi. System wykryje błędy funkcjonalne i logiczne w całej architekturze, eliminując potencjalne problemy na etapie projektowania. Korzyść dla użytkownika: Stabilność i kompatybilność aplikacji będą zwiększone, co pomoże uniknąć kosztownych błędów na późniejszych etapach rozwoju.

#### **2.5.2.3.** Inteligentna analiza kodu i rekomendacje naprawcze

Inteligentna analiza kodu i rekomendacje naprawcze. TestGEN nie tylko wykryje błędy, ale sugeruje skuteczne poprawki, wskazując np. nieprawidłowe filtrowanie danych, brak tokenów CSRF czy błędy autoryzacji i proponując optymalne rozwiązania. Korzyść dla użytkownika: Programiści zaoszczędzą czas, otrzymując konkretne wskazówki zamiast ręcznej analizy kodu, poprawiając zarówno efektywność, jak i bezpieczeństwo aplikacji.

#### **2.5.2.4.** Intuicyjny interfejs

Intuicyjny interfejs – dostępność dla każdego. TestGEN będzie wyposażony w wizualne formularze, podpowiedzi oraz moduł asystenta, eliminując potrzebę znajomości programowania. Dzięki funkcji "przeciągnij i upuść" (lub jej odpowiadającej), użytkownicy łatwo dodadzą pliki i skonfigurują testy. Korzyść dla użytkownika: Nawet osoby bez doświadczenia technicznego będą mogły skutecznie korzystać z TestGEN, co rozszerzy grupę użytkowników narzędzia.

#### **2.5.2.5.** Inteligentna analiza luk funkcjonalnych

Inteligentna analiza luk funkcjonalnych – TestGEN jako doradca biznesowy. System przeanalizuje kod, dokumentację, diagramy UML i przypadki testowe, wskazując niewdrożone funkcjonalności, które poprawią użyteczność aplikacji. Korzyść dla użytkownika: Firmy otrzymają wskazówki dotyczące brakujących funkcji, dzięki czemu ich produkty będą lepiej dostosowane do oczekiwań klientów.

#### **2.5.2.6.** Testowanie bezpieczeństwa w jednym miejscu

Testowanie bezpieczeństwa w jednym miejscu. TestGEN zidentyfikuje SQL injection, XSS i błędy autoryzacyjne, eliminując konieczność stosowania dodatkowych narzędzi do analizy podatności. Korzyść dla użytkownika: Programiści unikną luk bezpieczeństwa na etapie tworzenia aplikacji, minimalizując ryzyko ataków i strat finansowych.

#### **2.5.2.7.** Automatyzacja regresji i analiza wersji kodu

Automatyzacja regresji i analiza wersji kodu. System będzie automatycznie wykrywać zmiany w kodzie, eliminując błędy regresji i wskazując niespójności w projekcie. Korzyść dla użytkownika: Firmy unikną problemów wynikających z niekontrolowanych zmian, poprawiając stabilność aplikacji przy każdej aktualizacji.

#### **2.5.2.8.** Optymalizacja zasobów

Optymalizacja zasobów dzięki inteligentnej kompresji AI. TestGEN dynamicznie skalując obciążenie i minimalizując zużycie energii, zapewni wydajność nawet dla dużych organizacji. Korzyść dla użytkownika: Firmy nie będą musiały inwestować w dodatkowe serwery i zasoby, obniżając koszty infrastruktury IT.

#### **2.5.2.9.** Dynamiczne aktualizacje i samodoskonalenie systemu

Dynamiczne aktualizacje i samodoskonalenie systemu. System będzie regularnie aktualizowany, analizując nowo odkryte podatności, by zapewnić stałą ochronę aplikacji. Korzyść dla użytkownika: Firmy zyskają pewność, że ich oprogramowanie jest odporne na najnowsze zagrożenia, eliminując konieczność ręcznego monitorowania ryzyka.

### **2.5.3.** Podział projektu na zadania

Realizacja projektu składa się z poniższych zadań i głównych artefaktów dostarczanych w ramach tychże zadań. Szczegółowo ramy czasowe zadań zastały ujęte w rozdziale HARMONOGRAM.

**Każde zadanie musi zakończyć się przygotowanym, działającym i przetestowanym zestawem działających elementów systemu wraz z dokumentacją potwierdzającą jego realizację.**

Zadanie 1: opracowanie technologii i kluczowych funkcjonalności

1. Opracowanie mechanizmu analizy kodu przy użyciu metod sztucznej inteligencji i NLP
2. Implementacja modułu wykrywania luk funkcjonalnych oraz brakujących elementów implementacyjnych
3. Automatyczne generowanie przypadków testowych dla różnych środowisk programistycznych
4. Przygotowanie struktury raportowania wykrytych błędów i niezgodności w aplikacji

Zadanie 2: Wdrożenie modułów analizy zgodności kodu oraz testów bezpieczeństwa

1. Moduł analizy kodu pod kątem podatności na błędy związane np. z SQL injection,

XSS, błędami autoryzacji i lukami w logice aplikacji

1. Mechanizmy wykrywania niezgodności pomiędzy dokumentacją techniczną a rzeczywistą implementacją
2. Generowanie rekomendacji dotyczących poprawek kodu i metod zabezpieczeń.

Zadanie 3: Opracowanie interfejsu użytkownika i dostępności funkcji systemu

1. Projektowanie responsywnego GUI z możliwością powiększania czcionki, wysokim kontrastem i obsługą czytników ekranu
2. Wdrożenie elementów interfejsu umożliwiających łatwą konfigurację testów i analizę wyników
3. Zoptymalizowanie architektury systemu w celu poprawy dostępności i łatwości obsługi dla różnych grup użytkowników.

Zadanie 4: Opracowanie dokumentacji wdrożeniowej i materiałów użytkowych (300-400 stron)

1. Szczegółowy opis architektury systemu, modułów testowych oraz metod analizy
2. Instrukcja użytkownika z opisem funkcji TestGEN, sposobem konfiguracji testów oraz interpretacji wyników,
3. Specyfikacja techniczna obejmująca API oraz sposób integracji systemu z istniejącymi
4. procesami testowymi w organizacjach IT.

Zadanie 5: Wdrożenie systemu płatności i modelu komercjalizacji

1. Implementacja modułu płatności dla klientów indywidualnych oraz firm IT
2. Przygotowanie systemu zarządzania subskrypcjami i planami licencyjnymi
3. Specyfikacja techniczna obejmująca API oraz sposób integracji systemu z istniejącymi
4. Wdrożenie pilotażowych instalacji u klientów i analiza wyników ich użytkowania.

# **3.** WYMAGANIA FUNKCJONALNE

## **3.1.** Silnik TestSpark - komponenty podstawowe

Silnik TestSpark stanowi rdzeń systemu TestGEN i jest odpowiedzialny za wszystkie operacje związane z analizą kodu, dokumentacji i generowaniem testów. Poniżej przedstawiono szczegółową specyfikację jego komponentów.

### **3.1.1.** Architektura silnika TestSpark

**REQ-TS-001:** Silnik TestSpark musi być zbudowany jako skalowalna usługa backendowa z możliwością integracji z komponentami TestGEN.

Architektura systemu definiuje:

* **Strukturę danych** - ujednolicone reprezentacje kodu, dokumentacji i artefaktów testowych
* **Logikę przetwarzania kodu źródłowego** - mechanizmy analizy kodu w różnych językach programowania
* **Komponenty AI/ML** - modele uczenia maszynowego i transformery do przewidywania scenariuszy testowych

Zastosowane zostaną narzędzia do statycznej analizy kodu (np. AST parsers i linters), umożliwiające identyfikację funkcji, klas oraz zależności.

### **3.1.2.** Technologia AI do automatycznej generacji przypadków testowych

**REQ-TS-002:** Silnik TestSpark musi wykorzystywać modele uczenia maszynowego i transformery (np. GPT, BERT) do automatycznej generacji przypadków testowych.

Część silnika TestSpark wykorzystująca modele uczenia maszynowego i transformery będzie odpowiedzialna za:

* Przewidywanie scenariuszy testowych na podstawie analizy kodu i dokumentacji
* Klasyfikację scenariuszy testowych według typu (jednostkowe, integracyjne, systemowe)
* Przypisanie odpowiednich danych wejściowych i oczekiwanych rezultatów
* Wydzielanie logiki biznesowej z kodu źródłowego, aby testy odzwierciedlały rzeczywiste działanie systemu

### **3.1.3.** Podmoduł generujący kod testowy

**REQ-TS-003:** System musi automatycznie generować kod testowy gotowy do uruchamiania we frameworkach testowych: JUnit, NUnit, pytest.

Podmoduł generowania kodu testowego powinien automatycznie tworzyć pełne klasy testowe, uwzględniając:

* **Adnotacje** - odpowiednie dekoratory i atrybuty dla danego frameworka
* **Metody testujące** - kompletne metody z asercjami i logiką testową
* **Dane testowe** - fixtures, mock objects, test data generators
* **Setup i teardown** - metody przygotowujące i czyszczące środowisko testowe

## **3.2.** Moduły analizy i generowania testów

### **3.2.1.** Przegląd modułów

System TestGEN składa się z trzech głównych modułów analizy i generowania testów:

1. **Moduł generowania przypadków testowych z kodu źródłowego**
2. **Moduł generowania przypadków testowych z dokumentacji**
3. **Moduł generowania przypadków testowych z UML i GUI**

Każdy z tych modułów będzie szczegółowo opisany w kolejnych rozdziałach specyfikacji.

## **3.3.** Moduły bezpieczeństwa

### **3.3.1.** Przegląd modułów bezpieczeństwa

System TestGEN zawiera dwa kluczowe moduły bezpieczeństwa:

1. **Moduł inteligentnej analizy luk w kodzie oprogramowania** - wykrywa podatności i generuje rekomendacje zabezpieczeń
2. **Moduł analizy luk funkcjonalnych** - identyfikuje braki w implementacji i sugeruje rozszerzenia funkcjonalne

## **3.4.** Interfejs użytkownika

### **3.4.1.** Wymagania ogólne GUI

**REQ-GUI-001:** Interfejs użytkownika musi być zgodny z WCAG 2.1 poziom AA.

**REQ-GUI-002:** System musi zapewniać intuicyjną obsługę dla użytkowników bez doświadczenia programistycznego.

Graficzny interfejs użytkownika (GUI) powinien zawierać:

* **Wizualne formularze** - umożliwiające łatwe wczytywanie plików i konfigurację testów
* **Podpowiedzi kontekstowe** - dymki z informacjami wyświetlane po najechaniu na elementy
* **Moduł asystenta AI** - chatbot pomagający w nawigacji i konfiguracji
* **Funkcję "przeciągnij i upuść"** - dla wygodnego dodawania plików źródłowych

### **3.4.2.** Wymagania dostępności WCAG 2.1

Interfejs musi spełniać następujące wymagania dostępności:

|  |  |  |
| --- | --- | --- |
| **Kryterium WCAG** | **Wymaganie** | **Implementacja** |
| 1.4.3 Kontrast (minimum) | Współczynnik kontrastu min. 4.5:1 | Paleta kolorów z wysokim kontrastem, tryb wysokiego kontrastu |
| 1.4.4 Zmiana rozmiaru tekstu | Możliwość powiększenia tekstu do 200% | Responsywna typografia, kontrolki zoom |
| 2.1.1 Klawiatura | Pełna obsługa klawiatury | Tab navigation, keyboard shortcuts, focus indicators |
| 4.1.2 Nazwa, rola, wartość | Elementy z odpowiednimi atrybutami ARIA | ARIA labels, roles, states dla wszystkich interaktywnych elementów |

# **4.** SPECYFIKACJA MODUŁÓW

**A. Stworzenie autorskiego silnika TestSpark i przygotowanie modeli AI do automatycznego generowania przypadków testowych i kodów do ich wykonywania na podstawie dokumentacji, kodów, diagramów UML i GUI.**

**Termin realizacji: 2 stycznia 2026 - 31 maja 2026**

Celem zadania jest zaprojektowanie i implementacja kluczowego komponentu systemu TestGEN, tj. silnika TestSpark. Silnik ten, wykorzystując zaawansowane algorytmy sztucznej inteligencji, umożliwi pełną automatyzację analizy dokumentacji technicznej, kodu źródłowego, diagramów UML oraz ekranów GUI, generując na tej podstawie przypadki testowe oraz odpowiadający im kod testów. Projekt ma na celu znaczące ograniczenie pracy manualnej w procesie testowania i zwiększenie efektywności oraz pokrycia testowego. Podjęte zostaną następujące kluczowe czynności w ramach realizacji zadania.

1. Prace rozpoczną się od analizy danych wejściowych – kodu źródłowego, dokumentacji użytkownika, diagramów UML oraz zrzutów ekranów GUI. Dane te zostaną sklasyfikowane i zamodelowane zgodnie z ich strukturą oraz przeznaczeniem.

Celem jest opracowanie ujednoliconej reprezentacji danych, umożliwiającej ich dalsze przetwarzanie przez systemy AI.

2. Kolejnym krokiem będzie zaprojektowanie i wdrożenie komponentów ekstrakcji informacji. W tym celu zastosowane zostaną metody NLP (np. NER czy modeli typu transformer), pozwalające na identyfikację kluczowych pojęć, funkcji, warunków brzegowych i wyników opisanych w dokumentacji. W przypadku kodu źródłowego wykorzystane zostaną techniki statycznej analizy, umożliwiające wyodrębnienie struktur logicznych, punktów wejścia i zależności między komponentami. Diagramy UML zostaną przetworzone w celu wydobycia przepływów sterowania i logiki biznesowej, natomiast analiza GUI pozwoli na określenie interaktywnych elementów interfejsu i możliwych ścieżek działania użytkownika.

3. Na podstawie przetworzonych danych, opracowane zostaną mechanizmy automatycznego generowania różnorodnych przypadków testowych. Rozwiązanie będzie uwzględniać zidentyfikowane przepływy biznesowe, zależności między funkcjonalnościami, warunki brzegowe oraz potencjalne ścieżki wykonania. Uwzględnione zostaną scenariusze testowe – np. pozytywne, negatywne i brzegowe – jak również zidentyfikowane przepływy biznesowe, zależności i warunki wykonania. Wygenerowane przypadki testowe będą posiadały ustrukturyzowaną formę, zawierającą kroki, dane wejściowe, oczekiwane rezultaty i warunki wstępne.

4. Ostatni etap obejmuje stworzenie modułu generowania kodu testów. Na podstawie opisów przypadków testowych oraz danych źródłowych, silnik będzie mógł dla wybranych przypadków utworzyć kod dla frameworku testowego (np. JUnit, NUnit, pytest). W tym celu wykorzystane zostaną techniki szablonowe oraz inteligentne rozwiązania mapujące scenariusze testowe na konkretne instrukcje kodu. Generowane testy będą gotowe do uruchomienia i dostosowane do specyfiki interfejsów systemu (API, GUI).

Rezultatem zadania będzie w pełni funkcjonalny silnik TestSpark automatycznie generujący przypadki testowe i kod. Ocena obejmie jego zdolność do wygenerowania co najmniej 10 unikalnych przypadków testowych na podstawie dostarczonych danych (dokumentacja, kod, UML, GUI). Każdy przypadek testowy zawiera jasno określone kroki, dane wejściowe oraz oczekiwane rezultaty. Wygeneruje dla wybranych przyp. szkielet kodu testowego w wybranym frameworku (np. JUnit). Rezultaty zostaną ocenione manualnie pod kątem poprawności i jakości wygenerowanych artefaktów. Sprawdzona będzie ich adekwatność względem danych wejściowych. Dodatkowym wynikiem będzie dokumentacja projektowa opisująca architekturę silnika, zastosowane algorytmy, proces przetwarzania danych oraz wyniki testów prototypu

## **4.1.** Moduł generowania przypadków testowych z kodu źródłowego

### **4.1.1.** Opis ogólny modułu

Moduł generowania przypadków testowych na bazie kodów źródłowych jest odpowiedzialny za automatyczną analizę kodu aplikacji i generowanie na jej podstawie kompleksowych zestawów testów. Moduł wykorzystuje zaawansowane techniki analizy statycznej kodu oraz algorytmy sztucznej inteligencji.

**Technologie:** Realizacja modułu może wykorzystać technologie takie jak AST parsers i linters, SonarQube, Checkstyle i PMD.

### **4.1.2.** Funkcjonalność automatyzacji analizy kodu źródłowego

#### **4.1.2.1.** Opis funkcjonalności

Silnik TestSpark wykorzystuje metody Natural Language Processing (NLP) w celu automatycznej analizy kodu źródłowego testowanego oprogramowania, aby na jego podstawie automatycznie wygenerować przypadki testowe dopasowane do specyfiki danej aplikacji. Jest to pełna automatyczna analiza kodu.

**REQ-MOD-CODE-001:** System musi automatycznie rozpoznawać język programowania i framework aplikacji.

**REQ-MOD-CODE-002:** System musi wspierać co najmniej następujące języki: Java, Python, JavaScript, .NET, PHP, SQL, C#, C++, HTML, Kotlin, Swift.

#### **4.1.2.2.** Szczegółowy przepływ działania

Proces analizy kodu składa się z następujących etapów:

##### Etap A: Wczytanie i parsowanie kodu źródłowego

**Detekcja technologii i struktury projektu:**

System TestGEN automatycznie rozpoznaje:

* **Język programowania** - Java, Python, JavaScript, .NET, PHP, SQL, C#, C++, HTML, Kotlin, Swift itd.
* **Framework** - np. Django, React, Angular, Spring Boot, .NET, Flutter, Android SDK
* **Strukturę katalogów** - pliki źródłowe, testowe, konfiguracyjne, manifesty

Wykorzystuje analizę sygnatur plików (requirements.txt, package.json, pom.xml, build.gradle, csproj, setup.py, itp.).

##### Etap B: Parsowanie kodu i budowa modelu semantycznego

Kod jest analizowany przy użyciu parserów AST (Abstract Syntax Tree) i narzędzi statycznej analizy (np. tree-sitter, astroid, javalang, Esprima, Roslyn). Na tej podstawie system tworzy model semantyczny aplikacji, zawierający:

* Klasy, metody, funkcje, zmienne, parametry, typy zwracane, zależności
* Strukturę kontrolną (if/else, pętle, wyjątki)
* Punkty wejścia aplikacji (routy, endpointy, kontrolery, komponenty UI, akcje użytkownika)

Dzięki temu TestGEN „rozumie" logikę aplikacji, jej funkcje i przepływy danych.

##### Etap C: Analiza logiki biznesowej i przepływów użytkownika

**1. Identyfikacja funkcji aplikacji**

System wykrywa:

* Publiczne metody API, endpointy REST/GraphQL, kontrolery MVC
* Komponenty interfejsu użytkownika (np. Button, Input, Form, View, Page)
* Funkcje reagujące na zdarzenia (onClick, handleSubmit, onChange)
* Klasy biznesowe (np. OrderService, PaymentProcessor, UserManager)

Każdy z tych elementów staje się **kandydatem na przypadek testowy**.

**2. Analiza przepływu danych (data flow)**

System analizuje:

* Zależności między metodami
* Wartości wejściowe/wyjściowe
* Miejsca walidacji danych i obsługi błędów
* Komunikację między modułami (np. API → DB, UI → backend)

Na tej podstawie można tworzyć testy integracyjne i systemowe (np. "Sprawdź, czy dodanie produktu do koszyka zapisuje się w bazie i aktualizuje UI").

**3. Rekonstrukcja ścieżek użytkownika**

Jeśli kod dotyczy aplikacji interaktywnej (np. webowej lub mobilnej), TestGEN rekonstruuje potencjalne **user flows** - np.:

LoginPage

→

Dashboard

→

Settings

→

Logout

#### **4.1.2.3.** Przebiegi funkcjonalności

**Przebieg pozytywny:**

System po przeanalizowaniu dostępnych fragmentów systemu, takich jak kod źródłowy (minimum), dokumentacja, diagramy UML będzie w stanie przygotować opisowe przypadki testowe dla „wyłapanych" elementów systemu (funkcjonalności, funkcje, procedury), a tam gdzie to możliwe kod do ich uruchamiania w popularnych frameworkach.

**Przebieg negatywny:**

Wczytany kod, dokumentacja i inne elementy będą niepoprawne (np. nie będzie kodem w języku programowania) lub niepełne, w związku z czym system nie będzie w stanie go przeanalizować.

### **4.1.3.** Funkcjonalność generowania przypadków testowych i kodu testowego (dla back-endu, API)

#### **4.1.3.1.** Opis funkcjonalności

Na podstawie wczytanych danych w postaci kodu źródłowego system będzie w stanie generować przypadki testowe oraz kod testowy tam gdzie to możliwe dla popularnych frameworków, takich jak JUnit, NUnit czy pytest.

**REQ-MOD-CODE-003:** System musi generować przypadki testowe w formacie opisowym (readable test cases) zawierające: ID, cel, warunki wstępne, kroki, dane wejściowe, oczekiwane rezultaty, typ testu.

**REQ-MOD-CODE-004:** System musi generować kod testowy dla frameworków: JUnit, NUnit, pytest, Mocha, Playwright, Selenium, Appium.

#### **4.1.3.2.** Klasyfikacja testów

Z zebranych informacji TestGEN tworzy pełną specyfikację testową, obejmującą:

* **Testy jednostkowe** - dla każdej funkcji/metody o czystych zależnościach
* **Testy integracyjne** - dla metod wywołujących inne moduły
* **Testy systemowe** - dla endpointów, komponentów UI, przepływów użytkownika
* **Testy regresyjne** - dla istniejących przypadków w repozytorium (jeśli system znajdzie pliki test\_\*)
* **Testy bezpieczeństwa / walidacji** - automatycznie dla miejsc walidacji danych, logowania, ról użytkowników

#### **4.1.3.3.** Generacja przypadków w formie opisowej

Każdy przypadek testowy zawiera:

* ID testu (np. TC\_API\_001)
* Nazwę i opis funkcji
* Warunki wstępne
* Dane wejściowe
* Kroki testowe
* Oczekiwany rezultat
* Typ testu i priorytet

**Przykład:**

**TC\_UI\_003: Logowanie z niepoprawnym hasłem**  
**Cel:** Sprawdzić komunikat o błędzie przy logowaniu z błędnymi danymi.  
**Warunki:** Użytkownik na stronie /login  
**Kroki:**  
1. Wpisz email test@example.com  
2. Wpisz hasło "12345"  
3. Kliknij Zaloguj  
**Oczekiwany rezultat:** pojawia się komunikat "Nieprawidłowy login lub hasło"  
**Typ:** Negatywny, Funkcjonalny

#### **4.1.3.4.** Eksport przypadków

Przypadki mogą być eksportowane do:

* CSV, Excel, PDF
* TestRail, Jira XRay, Zephyr
* YAML/JSON dla generowania testów automatycznych

#### **4.1.3.5.** Kompatybilność z CI/CD

**REQ-MOD-CODE-005:** System musi umożliwiać wybór frameworka CI/CD i generować kod zgodny z: Jenkins, Azure DevOps, GitHub Actions, GitLab CI.

Wymaga implementacji funkcji wyboru dla jakiego CI/CD ma być generowany kod, aby było łatwo go wykorzystać wraz z popularnymi systemami CI/CD oraz narzędziami systemami CI/CD (Jenkins, Azure DevOps, GitHub ) oraz narzędziami testerskimi, takimi jak JUnit, Selenium, NUnit czy TestComplete..

#### **4.1.3.6.** Generacja kodu testowego

Na podstawie wygenerowanych przypadków i kodu źródłowego TestGEN generuje **test suites** w popularnych frameworkach:

|  |  |  |
| --- | --- | --- |
| **Typ aplikacji** | **Framework testowy** | **Przykładowy język** |
| Web | Selenium, Playwright, Cypress | Python / JS / TS |
| API / backend | PyTest, JUnit, Mocha | Python / Java / JS |
| Mobile | Appium, Espresso, XCUITest | Java / Kotlin / Swift |
| Desktop / GUI | SikuliX, PyAutoGUI, WinAppDriver | Python / Java |
| Low-code tests | Robot Framework | dowolny |

##### Tworzenie szkieletów testowych

Każdy przypadek opisowy jest przekształcany w kod testu:

* Automatycznie wypełniane są dane wejściowe i oczekiwane wyniki
* Dodawane są asercje
* Wstawiane są selektory / endpointy z kodu źródłowego

##### Generacja Page Objectów (dla UI)

Dla aplikacji webowych/mobilnych system buduje **Page Object Model (POM)** - pliki zdefiniowane dla każdego widoku.

**Przykład (Python + Selenium):**

class LoginPage: def \_\_init\_\_(self, driver): self.email = driver.find\_element(By.ID, "email") self.password = driver.find\_element(By.ID, "password") self.submit = driver.find\_element(By.ID, "login-btn") def login(self, email, password): self.email.send\_keys(email) self.password.send\_keys(password) self.submit.click()

##### Generacja testu z użyciem POM

def test\_login\_invalid\_password(driver): page = LoginPage(driver) page.login("test@example.com", "12345") assert "Nieprawidłowy login" in driver.page\_source

##### Analiza pokrycia kodu

Po wygenerowaniu testów TestGEN:

* Analizuje, które fragmenty kodu są pokryte testami (coverage mapping)
* Generuje raport (np. w formacie HTML lub JSON)
* Proponuje dodatkowe testy dla niepokrytych ścieżek logicznych

Wykorzystywane są tu narzędzia typu coverage.py, JaCoCo, Istanbul, NYC.

#### **4.1.3.7.** Artefakty końcowe

Po przetworzeniu kodu źródłowego TestGEN generuje:

* **Kompletny zestaw opisowych przypadków testowych** - funkcjonalnych, integracyjnych, regresyjnych, bezpieczeństwa i UI
* **Kod automatycznych testów** - w wybranym frameworku (np. PyTest, JUnit, Playwright, Appium)
* **Struktura projektu testowego** - gotowa do uruchomienia w CI/CD (Jenkins, GitHub Actions, GitLab CI)
* **Raporty pokrycia i analizy ryzyka** - co zostało pokryte testami, a które fragmenty kodu wymagają dopisania scenariuszy

#### **4.1.3.8.** Przykładowy przepływ techniczny

**Przepływ przetwarzania w silniku TestSpark:**

[Źródła kodu]

↓

[Analiza AST + Model semantyczny]

↓

[Identyfikacja funkcji i ścieżek użytkownika]

↓

[Generacja przypadków testowych (opisowych)]

↓

[Generacja kodu testów w wybranym frameworku]

↓

[Analiza pokrycia i raport końcowy]

### **4.1.4.** Funkcjonalność testów regresji kodu i GUI

#### **4.1.4.1.** Opis funkcjonalności

TestGEN umożliwia automatyczne wykrywanie zmian wskazując potencjalne błędy regresji analizując aplikację identyfikując ewentualne niespójności między wersjami kodu lub ekranu użytkownika (porównywanie obrazów).

**REQ-MOD-CODE-006:** System musi porównywać dwie wersje kodu źródłowego i identyfikować zmiany na poziomie semantycznym.

**REQ-MOD-CODE-007:** System musi generować testy regresyjne dla zidentyfikowanych zmian.

**Testy regresji mają sprawdzić, czy wprowadzone zmiany w nowej wersji kodu nie spowodowały błędów w funkcjonalnościach, które wcześniej działały poprawnie.**

#### **4.1.4.2.** Testy regresji kodu - szczegółowy przepływ

System TestGEN ma za zadanie:

* Porównać dwie wersje kodu źródłowego
* Zidentyfikować zmienione, dodane lub usunięte elementy
* Automatycznie dobrać lub wygenerować odpowiednie testy regresyjne
* Uruchomić je (lub przygotować kod do uruchomienia)
* Porównać wyniki i wygenerować raport zmian w zachowaniu aplikacji

##### ETAP 1: Analiza różnic między wersjami kodu

**1. Pobranie obu wersji projektu**

System pobiera:

* **Poprzednią wersję** (np. z repozytorium Git, tag v1.0)
* **Aktualną wersję** (np. z gałęzi develop lub main)

**2. Analiza różnic (diff analysis)**

TestGEN wykonuje **porównanie semantyczne** kodu (nie tylko tekstowe):

* Wykrywa zmiany w klasach, metodach, funkcjach, komponentach UI, endpointach API, plikach konfiguracyjnych, bazach danych
* Oznacza każdy element jako:
  + 🟢 nowy (dodany w nowej wersji)
  + 🔵 zmodyfikowany (zmieniona logika, parametry, typ zwracany, walidacja)
  + 🔴 usunięty (brak w nowej wersji)

**3. Mapowanie zależności**

System analizuje wpływ zmian na resztę kodu:

* Które funkcje/metody zależą od zmienionych modułów
* Które testy jednostkowe/integracyjne były powiązane z tymi fragmentami
* Które pliki interfejsu użytkownika (np. komponenty React, kontrolery) odwołują się do zmienionego kodu

Na tej podstawie powstaje **mapa wpływu (impact map)** - pokazująca, które obszary należy przetestować ponownie.

##### ETAP 2: Wybór i generacja testów regresyjnych

**1. Odtworzenie wcześniejszych przypadków testowych**

TestGEN odnajduje przypadki testowe (ręczne lub automatyczne) wygenerowane wcześniej dla wersji v1.0:

* Testy funkcjonalne
* Testy integracyjne
* Testy UI
* Testy API

**2. Automatyczne klasyfikowanie testów**

System dzieli istniejące testy na kategorie:

* ✅ niezmienione obszary - mogą być ponownie uruchomione bez modyfikacji
* ⚙️ dotknięte zmianami - wymagają aktualizacji danych testowych lub selektorów
* 🆕 nowe funkcjonalności - brak testów, trzeba je wygenerować
* ❌ usunięte funkcje - testy można oznaczyć jako nieaktualne

**3. Generacja nowych testów regresyjnych**

Dla wszystkich fragmentów oznaczonych jako zmodyfikowane lub nowe:

* TestGEN analizuje zmienioną logikę, parametry, walidacje, UI
* Generuje zaktualizowane przypadki testowe (opisowe)
* Tworzy kod testów automatycznych z użyciem istniejących frameworków (PyTest, JUnit, Selenium, Playwright, itp.)

**Przykład:**

Zmieniono endpoint /api/orders - dodano parametr discountCode  
→ TestGEN wygeneruje nowe przypadki:  
- TC\_API\_145: "Utworzenie zamówienia z kodem rabatowym"  
- TC\_API\_146: "Utworzenie zamówienia z nieprawidłowym kodem rabatowym"

##### ETAP 3: Uruchomienie testów regresyjnych

1. TestGEN uruchamia istniejące testy z poprzedniej wersji przeciwko nowej (o ile framework i środowisko na to pozwalają).
2. Dla testów niedziałających z powodu zmian selektorów, API lub parametrów - TestGEN automatycznie proponuje poprawki (np. aktualizację selektorów, parametrów wejściowych).
3. W przypadku zmian w UI - może uruchomić tryb **image-based comparison** (porównanie wyglądu ekranów v1.0 i v1.1).

##### ETAP 4: Porównanie wyników testów

System porównuje wyniki wykonania testów w obu wersjach:

* Które testy przeszły wcześniej i teraz się nie powiodły
* Które nowe testy ujawniły błędy
* Jakie elementy aplikacji zmieniły się wizualnie lub logicznie

Wyniki są prezentowane w **raporcie regresji**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Status v1.0** | **Status v1.1** | **Zmiana** | **Uwagi** |
| TC\_UI\_101 | PASS | FAIL | 🔴 | Błąd walidacji emaila po zmianie regexu |
| TC\_API\_145 | NEW | PASS | 🟢 | Nowy endpoint działa poprawnie |
| TC\_UI\_080 | PASS | PASS | 🟢 | Bez zmian |
| TC\_FUNC\_203 | PASS | SKIP | ⚪ | Funkcja została usunięta |

#### **4.1.4.3.** Testy regresji GUI

Testy regresji GUI mają wykryć, czy zmiany w kodzie lub komponentach frontendu (np. HTML, CSS, React, Vue, Angular, Flutter) **wpłynęły negatywnie na zachowanie, wygląd lub interakcyjność interfejsu użytkownika**.

TestGEN automatycznie porównuje poprzednią i nową wersję interfejsu, aby:

* Zidentyfikować zmienione elementy (przyciski, formularze, widoki, komunikaty)
* Zaktualizować istniejące testy lub wygenerować nowe
* Przygotować kod automatyzujący: skrypty startowe (np. Selenium / Playwright / Appium / Robot Framework / SikuliX / AutoHotkey / PyAutoGUI) lub krótkie makra - zależnie od docelowego środowiska

##### ETAP 1: Wczytanie i analiza zmian w GUI

**1. Źródła danych wejściowych**

System przyjmuje jednocześnie:

* Kod źródłowy interfejsu (HTML, JSX, TSX, CSS, Vue, XML itp.)
* Dane o strukturze komponentów (np. hierarchia React, routing w Angularze)
* Poprzednią wersję GUI (z repozytorium)
* Aktualną wersję GUI (po zmianach)
* Ewentualnie zrzuty ekranów lub makiety

**2. Porównanie struktur interfejsu**

TestGEN analizuje różnice w strukturze DOM / drzewie komponentów:

* Dodane, usunięte lub zmienione elementy (np. nowe pola formularzy, przyciski, modale)
* Zmienione atrybuty (np. id, class, placeholder, aria-label)
* Zmodyfikowane style (np. kolor, rozmiar, widoczność)
* Nowe logiki interakcji (np. eventy onClick, onSubmit, onChange)

Wynik: **mapa różnic GUI (GUI Diff Map)**, opisująca precyzyjnie, które elementy interfejsu uległy zmianie.

##### ETAP 2: Identyfikacja wpływu zmian na testy

TestGEN analizuje, które testy GUI z poprzedniej wersji (np. testy Selenium lub Playwrighta) odnoszą się do zmienionych elementów.

Dla każdego testu automatycznie określa status:

* ✅ niezmieniony - test może zostać ponownie uruchomiony
* ⚙️ do aktualizacji - zmienił się selektor, etykieta lub zachowanie elementu
* 🆕 do wygenerowania nowy test - dodano nowe funkcjonalności lub widoki
* ❌ do usunięcia - element GUI został wycofany

##### ETAP 3: Generacja przypadków testowych GUI

Na podstawie wykrytych zmian TestGEN tworzy zestaw **nowych lub zaktualizowanych przypadków testowych regresji GUI**.

Każdy przypadek zawiera:

* **Opis testu** (opisowy przypadek testowy) - w języku naturalnym
* **Kroki testowe** (automatycznie wygenerowane z analizy kodu GUI)
* **Oczekiwany wynik**
* **Powiązanie z elementami GUI** (np. data-testid, aria-label, id)

**Przykład:**

**TC\_GUI\_205:** Użytkownik może dodać produkt do koszyka po dodaniu nowego przycisku "Kup teraz"  
  
**Kroki testowe:**  
1. Otwórz widok "Lista produktów"  
2. Kliknij przycisk "Kup teraz" przy dowolnym produkcie  
3. Sprawdź, czy produkt został dodany do koszyka  
4. Zweryfikuj, że wyświetla się komunikat "Produkt dodano do koszyka"  
  
**Oczekiwany wynik:** komunikat jest widoczny, a stan koszyka się zmienia

##### ETAP 4: Generacja kodu testów automatycznych GUI

Na podstawie opisowych przypadków TestGEN generuje kod testów automatycznych w wybranym frameworku.

**Przykład (Playwright, TypeScript):**

test('Użytkownik może dodać produkt do koszyka', async ({ page }) => { await page.goto('/products'); await page.click('button:text("Kup teraz")'); await expect(page.locator('.cart-notification')).toHaveText('Produkt dodano do koszyka'); });

**Przykład (Selenium, Python):**

def test\_add\_to\_cart(driver): driver.get("https://app.example.com/products") driver.find\_element(By.XPATH, "//button[contains(text(), 'Kup teraz')]").click() message = driver.find\_element(By.CLASS\_NAME, "cart-notification").text assert "Produkt dodano do koszyka" in message

##### ETAP 5: Raport regresji GUI

System generuje **raport testów regresji GUI**, który zawiera:

* Zestawienie wszystkich przypadków (nowych, zmienionych, usuniętych)
* Miniatury ekranów "przed" i "po"
* Opis różnic wizualnych i logicznych
* Kod wygenerowanych testów automatycznych
* Rekomendacje napraw (np. "zaktualizować selektor przycisku X")

#### **4.1.4.4.** Przebiegi funkcjonalności

**Przebieg pozytywny:**

System będzie w stanie wyłapać różnice w kodzie i przygotować opisowe przypadków testowych do wykonania testów regresji jak i ewentualny kod do przetestowania danego testu.

**Przebieg negatywny:**

Np. w przypadku wczytania takiego samego kodu system nie będzie w stanie przygotować testów regresji wskazując, że wczytano ten sam kod.

### **4.1.5.** Wykorzystane mechanizmy i technologie

#### **4.1.5.1.** Architektura modułu

Architektura systemu definiująca:

* **Strukturę danych** - ujednolicone reprezentacje kodu w różnych językach
* **Logikę przetwarzania kodu źródłowego** - analiza kodu w różnych językach programowania (np. Java, Python, C#)

Zastosowane zostaną narzędzia do statycznej analizy kodu (np. AST parsers i linters), umożliwiające identyfikację funkcji, klas oraz zależności.

**Przykładowe narzędzia AST:**

* Python: ast, astroid
* JavaScript: Esprima, Acorn, Babel Parser
* Java: JavaParser, Eclipse JDT
* C#: Roslyn
* Multi-language: tree-sitter

#### **4.1.5.2.** Technologia AI do automatycznej generacji przypadków testowych

Część silnika TestSpark wykorzystująca modele uczenia maszynowego i transformery (np. GPT, BERT) do:

* Przewidywania scenariuszy testowych
* Klasyfikacji scenariuszy
* Przypisania odpowiednich danych wejściowych i oczekiwanych rezultatów
* Wydzielania logiki biznesowej z kodu źródłowego, aby testy odzwierciedlały rzeczywiste działanie systemu

#### **4.1.5.3.** Podmoduł generujący kod testowy

Część silnika TestSpark generująca przypadki testowe i kod testowy gotowy do uruchamiania we frameworkach testowych, np. JUnit, pytest czy NUnit.

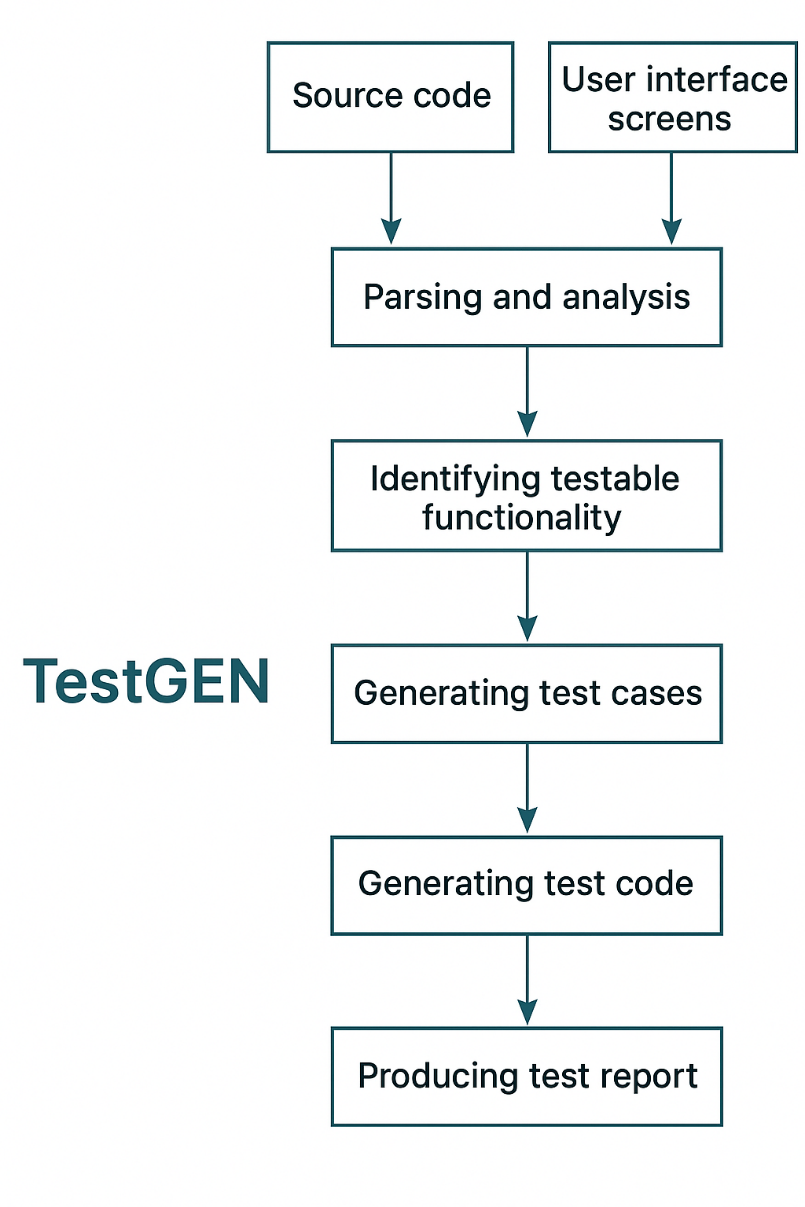
Ta część silnika powinna automatycznie tworzyć pełne klasy testowe, uwzględniając:

* **Adnotacje** - odpowiednie dla danego frameworka (@Test, @BeforeEach, @pytest.fixture, itp.)
* **Metody testujące** - z logiką testową i asercjami
* **Dane testowe** - fixtures, parametryzacja, test data builders

Całość zostanie przygotowana jako skalowalna usługa backendowa z możliwością integracji z komponentami TestGEN.

#### **4.1.5.4.** Diagram działania modułu

Poniższy schemat graficzny przedstawia proces (diagram blokowy lub procesowy TestGEN) pokazujący jak przepływa analiza od kodu po testy automatyczne:



Rysunek 4.1: Schemat przepływu przetwarzania w module analizy kodu źródłowego

## **4.2.** Moduł generowania przypadków testowych z dokumentacji i podręczników użytkownika

### **4.2.1.** Opis ogólny modułu

Moduł generowania przypadków testowych na podstawie dokumentacji i podręczników użytkownika wykorzystuje przetwarzanie języka naturalnego (NLP) oraz techniki uczenia maszynowego do automatycznej ekstrakcji wymagań i scenariuszy testowych z dokumentacji technicznej i biznesowej.

**Technologie:** Python, HuggingFace Transformers, spaCy, TensorFlow, FastAPI, Docker oraz narzędzia CI/CD do automatyzacji testowania i wdrożeń. Mogą zostać wykorzystane modele typu transformer (np. BERT) lub techniki NER (Named Entity Recognition) i klasyfikacji tekstu, reguły heurystyczne.

### **4.2.2.** Funkcjonalność automatyzacji analizy dokumentacji technicznej i biznesowej

#### **4.2.2.1.** Opis funkcjonalności

Celem jest automatyczne wykrywanie rozbieżności pomiędzy wizją biznesową a implementacją techniczną. System porównuje "co miało być zrobione" (w dokumentacji biznesowej) z "co zostało zaprojektowane lub zaimplementowane" (w dokumentacji technicznej).

**REQ-MOD-DOC-001:** System musi automatycznie analizować dokumentację techniczną i biznesową w formacie: PDF, DOCX, TXT, MD, HTML.

**REQ-MOD-DOC-002:** System musi identyfikować rozbieżności między dokumentacją biznesową a techniczną.

Dzięki temu:

* Można wykryć braki funkcjonalne (np. brak opisanej funkcji w kodzie lub projekcie technicznym)
* Można wykryć nadmiar funkcjonalny (np. funkcja w kodzie, której nie ma w wymaganiach biznesowych)
* Można zidentyfikować niespójności logiczne, terminologiczne i procesowe
* Oraz wcześnie ostrzec o potencjalnych błędach projektowych, zanim system przejdzie do testów lub wdrożenia

TestGEN automatycznie identyfikuje rozbieżności względem dokumentacji, co pozwala wykrywać potencjalne błędy przed wdrożeniem systemu. Jest to pełna automatyczna analiza dokumentacji z wymaganiami technicznymi.

#### **4.2.2.2.** Szczegółowy przepływ działania

##### ETAP 1: Wczytanie i klasyfikacja dokumentów

System wczytuje wszystkie źródła wiedzy dotyczące projektu, np.:

* **Dokumentacja biznesowa** - analizy wymagań (BRD, User Stories, Use Case'y, backlogi, makiety UX, opisy procesów)
* **Dokumentacja techniczna** - specyfikacje API, diagramy architektury, pliki konfiguracyjne, opisy modułów, komentarze kodu, instrukcje instalacji
* **Specyfikacje pośrednie** - np. FRD (Functional Requirements Document), SRS (Software Requirements Specification)

Po wczytaniu dokumenty są **zamieniane na jednolitą reprezentację tekstową**, z zachowaniem ich struktury logicznej (nagłówki, tabele, diagramy, sekcje).

System rozpoznaje, które fragmenty opisują:

* Funkcje użytkownika
* Procesy biznesowe
* Komponenty techniczne
* API / endpointy
* Reguły biznesowe
* Ograniczenia, parametry, warunki brzegowe

##### ETAP 2: Budowa modelu semantycznego dokumentacji

System wykorzystuje przetwarzanie języka naturalnego (NLP) i modele semantyczne (LLM) do **wyodrębnienia kluczowych pojęć, relacji i akcji** z obu typów dokumentacji.

Tworzy dwa modele wiedzy:

**Model Biznesowy (Business Knowledge Graph)**

Zawiera:

* Funkcje i cele biznesowe (np. "Użytkownik może dodać produkt do koszyka")
* Role użytkowników (np. "Administrator", "Klient")
* Procesy (np. "Zamówienie", "Zwrot", "Płatność")
* Reguły biznesowe (np. "Rabat tylko dla zamówień powyżej 500 zł")

**Model Techniczny (Technical Knowledge Graph)**

Zawiera:

* Komponenty systemu (np. CartService, PaymentModule, API /orders)
* Metody, endpointy, klasy, bazy danych
* Parametry, konfiguracje, przepływy danych
* Opisy logiki biznesowej w kodzie lub projektach UML

Każdy model jest zbudowany w formie grafu pojęć (knowledge graph), gdzie węzły to pojęcia, a krawędzie - relacje między nimi (np. Użytkownik → dodaje → Produkt).

##### ETAP 3: Porównanie modeli (Business vs. Technical Mapping)

System automatycznie **mapuje pojęcia i funkcje** z modelu biznesowego na odpowiadające im elementy modelu technicznego.

**Przykład mapowania:**

|  |  |  |
| --- | --- | --- |
| **Koncept biznesowy** | **Odpowiednik techniczny** | **Status** |
| "Zaloguj użytkownika" | POST /api/login | ✅ OK |
| "Zresetuj hasło" | brak endpointu | ❌ Brak w dokumentacji technicznej |
| "Generowanie raportu sprzedaży" | ReportService.generate() | ✅ OK |
| "Powiadomienia SMS" | brak komponentu SmsService | ⚠️ Niespójność |
| brak w BRD | AuditLogger.saveEvent() | ⚠️ Nadmiar funkcjonalny |

##### ETAP 4: Analiza rozbieżności

Po porównaniu system klasyfikuje różnice w następujących kategoriach:

**1. Braki funkcjonalne**

Funkcje opisane w dokumentacji biznesowej **nie mają odpowiedników technicznych**.

**Przykład:** BRD mówi o "śledzeniu przesyłki", ale brak modułu TrackingService w projekcie.

**2. Nadmiar funkcjonalny**

W dokumentacji technicznej znajdują się komponenty lub endpointy, które **nie mają pokrycia w wymaganiach biznesowych**.

**Przykład:** w API występuje /internal/exportLogs, którego nie przewidziano w BRD.

**3. Niespójności logiczne**

System wykrywa różnice w logice lub warunkach opisanych w obu dokumentacjach.

**Przykład:**  
BRD: "Zamówienie można anulować do momentu wysyłki."  
Specyfikacja techniczna: "Zamówienie można anulować tylko w ciągu 30 minut od utworzenia."

**4. Różnice terminologiczne**

System wykrywa, że te same pojęcia są różnie nazywane.

**Przykład:** BRD: "Użytkownik Premium" - TechDoc: "AccountType.GOLD".  
To nie błąd, ale wymaga standaryzacji słownika.

**5. Braki integracyjne**

Opisane w BRD przepływy między systemami **nie mają odwzorowania w architekturze technicznej**.

**Przykład:**  
BRD: "System wysyła dane do CRM."  
TechDoc: brak połączenia z API CRM.

##### ETAP 5: Generacja raportu rozbieżności

Po analizie system generuje **Raport Spójności Dokumentacji (Documentation Consistency Report)**.

Zawiera on:

* Listę zidentyfikowanych różnic
* Poziom istotności (np. błąd krytyczny, ostrzeżenie, różnica terminologiczna)
* Odniesienia do konkretnych sekcji w dokumentach
* Rekomendacje działań naprawczych

**Przykład fragmentu raportu:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Typ rozbieżności** | **Opis** | **Wystąpienia** | **Rekomendacja** |
| D-001 | Brak funkcjonalny | Funkcja "Reset hasła" nie występuje w dokumentacji technicznej | BRD §3.2 vs brak w TechDoc | Dodać opis implementacji lub uzupełnić API |
| D-004 | Niespójność logiczna | W BRD limit płatności 5000 zł, w TechDoc 3000 zł | §4.1 vs §2.3 | Ujednolicić limit w obu dokumentach |
| D-007 | Nadmiar funkcjonalny | Endpoint /admin/debug nie występuje w wymaganiach biznesowych | brak w BRD | Sprawdzić zasadność istnienia endpointu |

##### ETAP 6: Uczenie i aktualizacja modeli

System z każdą analizą:

* Uczy się terminologii specyficznej dla organizacji
* Tworzy słownik pojęć projektowych (glosariusz)
* Zapamiętuje reguły mapowania (np. "klient" = "user", "produkt" = "item")

Dzięki temu kolejne analizy są coraz dokładniejsze, a wykrywanie rozbieżności coraz bardziej precyzyjne i mniej podatne na "fałszywe alarmy".

#### **4.2.2.3.** Przebiegi funkcjonalności

**Przebieg pozytywny:**

Użytkownik wczytuje dwa dokumenty - dokumentację techniczną i biznesową. Dokonuje analizy dokumentów pod kątem ich różnicy. Przygotowuje raport wskazujący jakich funkcjonalności/funkcji/procedur nie ma w jednym dokumencie, które zostały wskazane w drugim z krótkim opisem rekomendującym uzupełnienie wybrakowanego dokumentu.

**Przebieg negatywny:**

Gdy wczytano puste dokumenty lub takie same materiały, system nie jest w stanie wskazać różnic.

### **4.2.3.** Funkcjonalność generowania przypadków testowych na podstawie dokumentacji technicznej i dokumentacji biznesowej

#### **4.2.3.1.** Opis funkcjonalności

**REQ-MOD-DOC-003:** System musi automatycznie generować opisowe przypadki testowe na podstawie dokumentacji technicznej i użytkownika.

System ma automatycznie tworzyć opisowe przypadki testowe (test cases) na podstawie treści dokumentacji dostarczonej przez producenta lub integratora oprogramowania.

Celem jest przekształcenie wiedzy zapisanej w dokumentach w zestaw testów, które można wykorzystać w testach akceptacyjnych, integracyjnych, funkcjonalnych, instalacyjnych lub bezpieczeństwa.

#### **4.2.3.2.** Szczegółowy przepływ działania

##### ETAP 1: Wczytanie i klasyfikacja dokumentacji

System przyjmuje różne formaty plików:

* Dokumenty tekstowe (DOCX, PDF, ODT, TXT, MD)
* Dokumenty techniczne (np. UML, Swagger/OpenAPI, README, konfiguracje YAML, JSON)
* Instrukcje użytkownika
* Podręczniki administratora
* Instrukcje instalacyjne
* Specyfikacje wymagań (SRS, BRD, FRD)

**Proces:**

1. Dokumenty są wczytywane i konwertowane do formatu tekstowego (OCR, ekstrakcja z PDF itp.).
2. TestGEN-AI analizuje zawartość i automatycznie klasyfikuje fragmenty na typy:
   * "Wymagania funkcjonalne"
   * "Wymagania niefunkcjonalne"
   * "Instrukcje obsługi / użytkowania"
   * "Instrukcje instalacji"
   * "Opis architektury / integracji"
   * "Opis błędów i wyjątków"
   * "Procedury konfiguracji / bezpieczeństwa"

Ta klasyfikacja tworzy mapę semantyczną dokumentacji - czyli model, w którym każdy fragment jest przypisany do odpowiedniej kategorii testowalnych informacji.

##### ETAP 2: Ekstrakcja informacji testowalnych

Silnik TestSpark analizuje każdy akapit dokumentacji, aby wyłuskać potencjalne punkty testowe.

System rozpoznaje m.in.:

* Akcje użytkownika (np. "użytkownik wybiera opcję", "klika przycisk", "uruchamia moduł")
* Oczekiwane rezultaty (np. "system wyświetla komunikat", "zapisuje dane", "odrzuca żądanie")
* Warunki brzegowe i zależności (np. "jeśli użytkownik nie ma uprawnień", "jeżeli dane są niepoprawne")
* Parametry techniczne i ograniczenia (np. porty, konfiguracje, wersje bibliotek)
* Scenariusze wyjątkowe i błędy (np. "gdy serwer nie odpowiada", "błąd uwierzytelniania")

Zidentyfikowane informacje są następnie ułożone w strukturę przypominającą macierz wymagań testowych (Requirement-to-Test Matrix).

##### ETAP 3: Generacja logicznego modelu testów

System tworzy abstrakcyjny model testowy, który opisuje:

* Obiekty testowane (np. moduły, ekrany, funkcje API)
* Możliwe działania użytkownika lub systemu
* Dane wejściowe i oczekiwane wyniki
* Zależności między przypadkami

Model ten ma postać ontologii testowej - sieci pojęć i relacji między funkcjami, wymaganiami i testami.

**Przykład fragmentu modelu (w postaci logicznej):**

**Funkcja:** Logowanie użytkownika  
→ **Wymaganie:** Po poprawnym wprowadzeniu loginu i hasła użytkownik zostaje zalogowany.  
→ **Scenariusz:** Użytkownik wprowadza poprawne dane.  
→ **Oczekiwany wynik:** Przekierowanie na ekran główny.

##### ETAP 4: Generacja opisowych przypadków testowych

Na podstawie modelu system automatycznie tworzy opisowe przypadki testowe, w czytelnej formie narracyjnej.

Każdy przypadek zawiera:

* **Identyfikator testu** (np. TC\_AUTH\_001)
* **Nazwa testu**
* **Cel testu**
* **Warunki wstępne**
* **Kroki testowe** (zestaw czynności użytkownika lub systemu)
* **Dane wejściowe**
* **Oczekiwany rezultat**
* **Typ testu** (funkcjonalny, instalacyjny, integracyjny itp.)
* **Źródło** (odniesienie do fragmentu dokumentacji)

**Przykład 1 (z dokumentacji użytkowej):**

**TC\_GUI\_05 - Dodanie nowego użytkownika**  
  
**Cel:** Sprawdzić, czy administrator może dodać nowego użytkownika z panelu zarządzania.  
**Warunki wstępne:** Administrator jest zalogowany.  
**Kroki testowe:**  
1. Otwórz zakładkę "Użytkownicy"  
2. Kliknij "Dodaj użytkownika"  
3. Wprowadź imię, nazwisko, e-mail i rolę  
4. Kliknij "Zapisz"  
  
**Oczekiwany rezultat:** Nowy użytkownik pojawia się na liście.  
**Źródło:** Instrukcja użytkownika, rozdział 4.3 "Zarządzanie użytkownikami"

**Przykład 2 (z dokumentacji instalacyjnej):**

**TC\_INST\_02 - Weryfikacja poprawności instalacji bazy danych**  
  
**Cel:** Sprawdzić, czy baza danych została utworzona z prawidłową strukturą.  
**Warunki wstępne:** Środowisko testowe skonfigurowane.  
**Kroki testowe:**  
1. Uruchom skrypt install\_db.sql  
2. Sprawdź, czy utworzone zostały tabele USERS, ROLES, LOGS  
  
**Oczekiwany rezultat:** Wszystkie tabele są obecne i dostępne.  
**Źródło:** Dokumentacja instalacyjna, sekcja 2.1

##### ETAP 5: Weryfikacja spójności i kompletności

System analizuje, czy:

* Każde wymaganie z dokumentacji ma przypisany co najmniej jeden przypadek testowy
* Nie ma przypadków testowych bez źródła w dokumentacji
* Przypadki testowe pokrywają zarówno scenariusze pozytywne, jak i negatywne
* Opisy testów są spójne logicznie i językowo

Wynikiem jest **macierz śladowania (Traceability Matrix)**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Wymaganie** | **Opis** | **Test Case** | **Status pokrycia** |
| FR-001 | Logowanie użytkownika | TC\_AUTH\_001, TC\_AUTH\_002 | ✅ Pokryte |
| FR-002 | Reset hasła | TC\_AUTH\_003 | ⚠️ Częściowo |
| NFR-001 | Czas reakcji < 2s | TC\_PERF\_001 | ✅ Pokryte |

##### ETAP 6: Generacja testów automatycznych (opcjonalna)

Na podstawie opisowych przypadków testowych TestGEN-AI może automatycznie wygenerować szkielety testów automatycznych w wybranych frameworkach, np.:

* **Playwright / Cypress / Selenium** (testy GUI)
* **Pytest / JUnit / NUnit** (testy jednostkowe i integracyjne)
* **Postman / RestAssured** (testy API)
* **Robot Framework** (testy wysokopoziomowe)
* **PowerShell / Bash** (testy instalacyjne)

Każdy kod zawiera odniesienia do przypadków testowych i danych wejściowych.

##### ETAP 7: Raport i eksport wyników

System generuje:

* Raport testów w formacie PDF/HTML/Excel
* Plik eksportu przypadków testowych do narzędzi takich jak Jira, TestRail, Zephyr, Xray, qTest
* Opcjonalny plik JSON/XML z metadanymi testów (dla integracji CI/CD)

#### **4.2.3.3.** Przebiegi funkcjonalności

**Przebieg pozytywny:**

Użytkownik wczytuje dokumentację. System dokonuje analizy dokumentów i generuje kompletny zestaw przypadków testowych wraz z macierzą śladowania.

**Przebieg negatywny:**

Gdy wczytano puste dokumenty lub dokumenty bez testowalnych informacji, system nie jest w stanie wygenerować przypadków testowych i informuje użytkownika o braku odpowiednich danych wejściowych.

### **4.2.4.** Wykorzystane mechanizmy i technologie

#### **4.2.4.1.** Mechanizm importu i analizy tekstów źródłowych

Mechanizm powinien być w stanie zaimportować i analizować teksty źródłowe w różnych formatach (np. PDF, DOC, HTML) przy wykorzystaniu narzędzi do ekstrakcji treści, które umożliwią jednolite przetwarzanie dokumentów, oczyszczanie ich z elementów nieistotnych (np. nagłówki, stopki) oraz strukturyzację danych wejściowych.

**Przykładowe narzędzia:**

* PyPDF2, pdfplumber - ekstrakcja z PDF
* python-docx - przetwarzanie DOCX
* BeautifulSoup - parsowanie HTML
* Tesseract OCR - rozpoznawanie tekstu z obrazów
* Apache Tika - uniwersalny ekstraktor treści

#### **4.2.4.2.** Mechanizm analizy językowej dokumentacji

Mechanizm będzie identyfikował fragmenty zawierające opisy funkcjonalności, kroków użytkownika, warunków brzegowych oraz oczekiwanych rezultatów.

W tym celu zostaną przygotowane modele typu transformer (np. BERT) lub techniki NER (Named Entity Recognition) i klasyfikacji tekstu, które umożliwią wyodrębnienie istotnych encji, takich jak operacje, parametry wejściowe czy komunikaty błędów.

Przewiduje się fine-tuning modeli na domenowych danych, co zwiększy trafność ekstrakcji informacji.

**Przykładowe techniki NLP:**

* Named Entity Recognition (NER) - identyfikacja kluczowych pojęć
* Dependency Parsing - analiza zależności składniowych
* Text Classification - kategoryzacja fragmentów tekstu
* Semantic Similarity - wykrywanie podobieństw semantycznych
* Intent Recognition - rozpoznawanie intencji w tekście

#### **4.2.4.3.** Mechanizm tworzący strukturę przypadku testowego

Dla zidentyfikowanego scenariusza testowego mechanizm powinien być w stanie utworzyć szkielet testu zawierający m.in. dane wejściowe, oczekiwane wyniki, opis kroków oraz typ testu (np. funkcjonalny, walidacyjny).

#### **4.2.4.4.** API integracyjne

Integracja mechanizmów z głównym silnikiem TestSpark oraz przygotowanie API umożliwiającego przesyłanie dokumentacji i odbieranie gotowych przypadków testowych oraz kodu.

**REQ-MOD-DOC-004:** Moduł musi udostępniać RESTful API do integracji z zewnętrznymi systemami zarządzania testami.

## **4.3.** Moduł generowania przypadków testowych z UML i GUI

### **4.3.1.** Opis ogólny modułu

Moduł będzie odpowiedzialny za generowanie przypadków testowych oraz kodu do ich wykonywania na podstawie diagramów UML oraz ekranów użytkownika (GUI). W tym celu będzie automatyzował proces ekstrakcji informacji z wizualnych źródeł i ich przekształcenie w scenariusze testowe gotowe do wykonania w środowiskach testowych.

**Technologie:** Moduł może wykorzystywać mechanizmy takie jak parsery UML (np. PlantUML, GraphML) lub sieci transformer do rozpoznawania diagramów. Dodatkowo może wykorzystywać technologie rozpoznawania obrazu (OCR - np. Tesseract, OpenCV) oraz klasyfikatory wizualne - modele CNN (Convolutional Neural Networks) lub sieci GPT.

### **4.3.2.** Funkcjonalność automatyzacji analizy diagramów UML i generowania opisowych przypadków testowych

#### **4.3.2.1.** Opis funkcjonalności

**REQ-MOD-UML-001:** System musi obsługiwać diagramy UML w formatach: XMI, PlantUML, SVG, PDF, PNG/JPEG.

**REQ-MOD-UML-002:** System musi rozpoznawać następujące typy diagramów UML: sekwencji, przypadków użycia, klas, aktywności, stanów, komponentów.

Silnik TestSpark przetwarza diagramy UML (różne ich typy, np.: sekwencji, przypadków użycia) i na ich podstawie generuje opisowe przypadki testowe oraz gotowe szablony kodu testów.

#### **4.3.2.2.** Szczegółowy przepływ działania

##### 1. Wejście - co system przyjmuje na wejściu

System akceptuje:

* **Pliki UML w formatach strukturalnych** (XMI, PlantUML/text UML, Sparx/Enterprise Architect export)
* **Wektorowe diagramy** (SVG, PDF) zawierające obiekty UML
* **Rasterowe obrazy diagramów** (PNG/JPEG) - wtedy używa OCR + rozpoznawania kształtów

Dzięki temu może działać zarówno na "maszynowych" reprezentacjach UML (łatwe do sparsowania), jak i na obrazach wygenerowanych ręcznie.

##### 2. Faza 1 - wczytanie i normalizacja diagramu

**Kroki działania silnika:**

* **Detekcja formatu** - jeśli to XMI/PlantUML: bezpośrednie parsowanie; jeśli SVG/PDF: parsowanie XML/SVG; jeśli raster (PNG/JPEG): pipeline obraz → OCR na tekst etykiet → wykrywanie kształtów i krawędzi (nodes/connectors) → rekonstrukcja grafu UML.
* **Normalizacja modelu** - wszystkie diagramy konwertowane są do jednego wewnętrznego modelu (np. JSON/UML-IR) z unikalnymi ID elementów, typami (Class/Actor/UseCase/Message/State/Activity/Component), atrybutami i metadanymi (np. linia, numer strony).
* **Wykrycie spójnych flow** - gdy dostarczono kilka diagramów, system łączy elementy powiązane (np. klasa Order z sekwencją PlaceOrder) i buduje wspólny model funkcjonalny.

##### 3. Faza 2 - syntaktyczna i semantyczna analiza UML

**Co system wydobywa z poszczególnych typów diagramów:**

* **Diagram sekwencji:** lifelines, komunikaty synchroniczne/asynchroniczne, alternatywne ścieżki (alt), pętle (loop), warunki → bezpośrednie kroki scenariuszy E2E i integracyjnych.
* **Use-case / Actor:** aktorzy i cele → scenariusze akceptacyjne (user stories → acceptance tests).
* **Diagram klas:** atrybuty, metody, relacje (agregacja, kompozycja, dziedziczenie) → testy jednostkowe dla każdej klasy/metody publicznej.
* **Diagram aktywności:** przepływy decyzyjne (decision nodes), akcje równoległe (fork/join) → testy ścieżek, boundary conditions.
* **Diagram stanów:** stany, przejścia, eventy, guardy → testy state machine (np. lifecycle obiektu Order).

Dla każdego wykrytego elementu system tworzy model semantyczny: wymaganie (co), prewarunek (kiedy), postwarunek (jakie zmiany), wyjątki.

##### 4. Faza 3 - identyfikacja punktów testowalnych i ryzyka

System analizuje:

* **Punkty decyzyjne** (if/alt/guards) → generuje warianty testów dla każdej ścieżki
* **Pętle / multiplicities** → generuje testy brzegowe (0, 1, max, >max)
* **Wartości walidacyjne** (typy/regex w atrybutach) → testy walidacji
* **Zależności zewnętrzne** → plan mocków i testów integracyjnych
* **Obsługę błędów i wyjątki** → testy negatywne

Na wyniku obliczana jest miara ryzyka/priorytetu (np. krytyczność funkcji, złożoność ścieżki), aby zasugerować, które testy są priorytetowe do uruchomienia najpierw.

##### 5. Faza 4 - generowanie opisowych przypadków testowych

Dla każdej wykrytej funkcji/flow system generuje zestaw opisowy.

Każdy przypadek testowy zawiera:

* ID (np. TC\_SEQ\_Login\_001)
* Tytuł
* Opis celu
* Warunki wstępne
* Kroki testowe (sekwencja kroków krok→akcja→dane)
* Oczekiwany rezultat
* Typ testu (jednostkowy/integracyjny/systemowy/akceptacyjny/negatywny)
* Priorytet i wiążące elementy UML (odniesienie do ID elementu w modelu)
* Słowa kluczowe (np. "auth", "payment", "boundary")
* Proponowane dane testowe (przykładowe wartości, payloady JSON)
* Wymagane mocki/usługi zewnętrzne

**Przykładowy opisowy TC (z sekwencji logowania):**

**ID:** TC\_SEQ\_Login\_001  
**Tytuł:** Logowanie - prawidłowe dane  
**Opis:** Użytkownik przesyła poprawne dane logowania, system tworzy sesję i przekierowuje do dashboard.  
**Warunki:** Użytkownik istnieje w systemie; konto aktywne.  
**Kroki:**  
1. Otwórz POST /login z payload {"email": "u@test.com", "password": "Haslo1"}  
2. Oczekuj HTTP 200 i token w odpowiedzi  
3. Po użyciu tokena GET /dashboard → status 200 i zawartość dashboard  
  
**Oczekiwane:** Token JWT w odpowiedzi; dashboard dostępny.  
**Typ:** Integracyjny/API. **Priorytet:** Wysoki.  
**Powiązanie UML:** Sequence/LoginFlow message 1..3

##### 6. Faza 5 - generowanie kodu testów (szkielety) i artefaktów

System może wygenerować gotowe szablony testów w wybranych frameworkach. Typowe elementy:

* **Unit tests:** generowane z diagramu klas → testy metod publicznych; generowanie stubów i mocków zależności (np. z Mockito/pytest-mocker).
* **Integration/API tests:** generowane z endpointów wykrytych w sekwencjach → requests + asercje (PyTest, JUnit + RestAssured, Mocha/Chai).
* **E2E UI tests:** z sekwencji/aktivities/use-case → Page Objects + testy Playwright/Selenium/Cypress.
* **State tests:** z diagramów stanów → testy sprawdzające, że na określony event następuje oczekiwane przejście (unit/integration).
* **BDD/acceptance:** wygenerowanie Gherkin .feature.
* **Test harness / CI:** minimalne pliki konfiguracyjne (pytest.ini, playwright.config.js, GitHub Actions workflow do uruchomienia testów).

**Przykłady generowanych artefaktów:**

**Gherkin (z sekwencji logowania):**

Feature: Authentication Scenario: Successful login Given a registered user with email "u@test.com" and password "Haslo1" When the user submits credentials to /login Then the response status is 200 and a JWT token is returned And accessing /dashboard with token returns status 200

**PyTest + Playwright (UI E2E skeleton):**

from playwright.sync\_api import Page

def test\_login\_success(page: Page):

page.goto("https://app.example.com/login")

page.fill("#email", "u@test.com")

page.fill("#password", "Haslo1")

page.click("button[type=submit]")

page.wait\_for\_selector(".dashboard-header")

assert "Witaj" in page.text\_content(".dashboard-header")

**JUnit + RestAssured (API):**

@Test public void loginSuccess() { given() .contentType("application/json") .body("{\"email\":\"u@test.com\",\"password\":\"Haslo1\"}") .when() .post("/login") .then() .statusCode(200) .body("token", notNullValue()); }

**Page Object (python selenium):**

class LoginPage: def \_\_init\_\_(self, driver): self.driver = driver self.email = "#email" self.password = "#password" self.submit = "button[type=submit]" def login(self, email, pwd): self.driver.find\_element\_by\_css\_selector(self.email).send\_keys(email) self.driver.find\_element\_by\_css\_selector(self.password).send\_keys(pwd) self.driver.find\_element\_by\_css\_selector(self.submit).click()

##### 7. Faza 6 - dane testowe, mocki i środowisko

System automatycznie generuje:

* Dane testowe (pozytywne/negatywne, boundary)
* Definicje mocków dla usług zewnętrznych (np. fake auth, fake payment gateway)
* Konfiguracje środowiska testowego (Docker compose skeleton, baza testowa z seedami)
* Skrypty uruchamiania (make, scripts, CI pipeline fragment)

##### 8. Faza 7 - mapowanie (traceability) i raportowanie

Każdy test zostaje powiązany z elementem UML (ID). System eksportuje:

* Macierz śledzenia: UML element ↔ wygenerowane przypadki testowe ↔ wygenerowany kod testu
* Raporty pokrycia logicznego: które sekwencje/transition/branch są testowane, które nie
* Lista ryzyk i rekomendacje: brak testów dla wyjątkowo krytycznych ścieżek, propozycja rozszerzeń

**Przykładowy wiersz CSV macierzy:**

uml\_id,uml\_type,tc\_id,tc\_type,test\_file,priority SEQ\_01,message\_Login,TC\_SEQ\_Login\_001,Integration,tests/api/test\_login.py,High

##### 9. Faza 8 - walidacja, niepewność i human-in-the-loop

System przypisuje score ufności (np. 0-1) dla każdego wygenerowanego testu zależnie od jakości diagramu/kompletności informacji (np. diagram bez guardów → low confidence dla wartości boundary).

Generuje suggested clarifications i zaznacza miejsca gdzie potrzebna jest walidacja człowieka (np. "nie określono, czy login powinien wymuszać złożoność hasła").

Udostępnia interfejs do akceptacji/edycji testów przed eksportem (review UI), ale sam generuje kompletny zestaw bez konieczności od razu zadawania pytań.

**Przykład:** konwersja prostego diagramu sekwencji → testy

Dla prostego diagramu sekwencji:

**User -> LoginPage : submits(email,password)**

**LoginPage -> AuthService : POST /auth**

**AuthService -> DB : verify(user)**

**AuthService --> LoginPage : 200 + token / 401**

**LoginPage -> Dashboard : redirect on 200**

Mogą zostać wygenerowane poniższe testy (przykładowo):

**• TC1: Logowanie poprawne → oczekuj 200 + token → dostęp do dashboard. (E2E + API)**

**• TC2: Logowanie — złe hasło → oczekuj 401 i komunikat.**

**• TC3: Logowanie — brak pól → oczekuj 400 + walidacja.**

**• TC4: Sesja wygasa → token expired → oczekuj 401 przy GET /dashboard.**

Gherkin + PyTest snippets pokazane wcześniej są dobrym przykładem jak to zostanie przekształcone w kod.

##### 10. Artefakty końcowe

Po zakończeniu procesu system dostarcza:

* Plik/zbiór opisowych przypadków testowych (CSV/Excel/JSON/PDF)
* Folder projektu testowego z wygenerowanymi testami (pytest/junit/robot/playwright), PageObjects, mockami, fixtures, configs, README z instrukcjami uruchomienia
* Macierz traceability (UML ↔ test)
* Raport pokrycia logicznego i propozycję minimalnego zestawu regression suite
* Listę pytań/niejasności do weryfikacji (opcjonalnie)

### **4.3.3.** Funkcjonalność automatyzacji analizy interfejsów użytkownika (GUI) i generowania przypadków testowych (opisowych lub kodu makra do automatycznego testu)

#### **4.3.3.1.** Opis funkcjonalności

**REQ-MOD-GUI-001:** System musi analizować zrzuty ekranów w formatach: PNG, JPEG, PDF.

**REQ-MOD-GUI-002:** System musi identyfikować komponenty UI: przyciski, pola tekstowe, listy rozwijane, checkboxy, radiobuttons, menu, modale.

Silnik TestSpark przetwarza ekrany GUI i generuje na ich podstawie przypadki testowe oraz kod automatyzujący testy interfejsu.

#### **4.3.3.2.** Szczegółowy przepływ działania

##### ETAP 1: Wczytanie i wstępna analiza ekranów

Po przesłaniu zrzutów ekranu:

**1. Analiza wizualna interfejsu**

* Rozpoznanie typu widoku (ekran logowania, dashboard, formularz, lista, popup, itp.)
* Identyfikacja głównych komponentów UI: pola tekstowe, przyciski, przełączniki, ikony, menu, listy rozwijane, komunikaty błędów, elementy nawigacji
* Oznaczanie możliwych stanów (np. aktywne/nieaktywne przyciski, pojawienie się komunikatu po błędzie)

**2. Identyfikacja kontekstu biznesowego**

* Na podstawie etykiet, tekstów i nazw elementów (np. "Zaloguj", "Zarejestruj się", "Wyślij", "Koszyk", "Zapisz zmiany") rekonstrukcja celu ekranu i jego powiązanie z innymi
* Dla kilku ekranów - rozpoznanie ścieżki użytkownika (user flow), np.: Ekran 1 (Logowanie) → Ekran 2 (Panel główny) → Ekran 3 (Profil użytkownika)

**3. Identyfikacja punktów interakcji**

* Zaznaczanie potencjalnych punktów akcji: kliknięcia, wprowadzanie tekstu, wybory z list, przełączniki, drag&drop itp.
* Grupowanie ich w sekwencje, które mogą tworzyć pojedynczy przypadek testowy

##### ETAP 2: Synteza funkcjonalności i generowanie przypadków testowych

Na podstawie zidentyfikowanych elementów i flow silnik TestSpark:

**1. Tworzy listę funkcjonalności użytkownika**

Każda funkcja (np. logowanie, filtrowanie wyników, dodanie produktu do koszyka) staje się grupą testów.

**2. Generuje przypadki testowe**

Dla każdej funkcji powstaje zestaw przypadków, w tym:

* **Testy pozytywne** (prawidłowe dane i scenariusze sukcesu)
* **Testy negatywne** (puste pola, błędne dane, nieprawidłowe kolejności akcji)
* **Testy brzegowe** (np. maksymalna długość pól, ograniczenia walidacyjne)
* **Testy UI/UX** (czy przyciski są widoczne, poprawnie podpisane, działają stany hover/disabled)

**3. Tworzenie struktury przypadków w czytelnym formacie**

Każdy przypadek ma:

* ID testu
* Nazwę (np. TC-001: Logowanie z poprawnymi danymi)
* Cel testu
* Warunki wstępne
* Kroki testowe
* Oczekiwany rezultat
* Priorytet

**4. Opcjonalnie - generacja przypadków w formacie Gherkin (BDD)**

Feature: User Login Scenario: Successful login with valid credentials Given user is on login page When user enters email "test@example.com" And user enters password "Haslo123" And user clicks "Login" button Then user should be redirected to dashboard And welcome message should be displayed

### **4.3.4.** Funkcjonalność generowania przypadków testowych i kodu testowego (GUI)

#### **4.3.4.1.** Opis funkcjonalności

**REQ-MOD-GUI-003:** System musi generować kod testowy dla frameworków: Selenium, Playwright, Cypress, Appium, SikuliX.

`

* **Scenariusze testowe w formie opisowej** (np. krok → oczekiwany rezultat), priorytety, dane testowe, warunki wstępne i pożądane kroki odtworzenia błędów
* **Szablony przypadków testowych** (np. Gherkin / Given-When-Then) lub listy kontroli (checklist)
* **Przykładowy kod automatyzujący**: skrypty startowe (np. Selenium / Playwright / Appium / Robot Framework / SikuliX / AutoHotkey / PyAutoGUI), lub krótkie makra (AutoHotkey, VBA) - zależnie od docelowego środowiska
* **Tam gdzie to możliwe także pliki konfiguracyjne lub test-suite skeleton** (np. pytest + selenium, Playwright test, Robot Framework .robot)

**REQ-MOD-GUI-004:** System musi umożliwiać wybór frameworka CI/CD dla generowanego kodu testowego.

**Wymgana jest kompatybilności z istniejącymi frameworkami testerskimi** oraz możliwość wykorzystania systemów CI/CD w zależności od preferencji użytkowników końcowych. Oznacza to **konieczność** implementacji funkcji wybrania dla jakiego CI/CD ma być generowany kod aby było łatwo go wykorzystać wraz z popularnymi systemami CI/CD oraz narzędziami testerskimi, takimi jak JUnit, Selenium, NUnit czy TestComplete (ew. GitHub, Azure DevOps).

#### **4.3.4.2.** Szczegółowy przepływ działania

Użytkownik wczytuje do systemu np. zrzuty ekranów w dobrej rozdzielczości, zawierające cały ekran (lub kolejność ekranów/flow).

**Opcjonalnie może wczytać:**

* Krótki opis scenariuszy biznesowych / co użytkownik ma osiągnąć na danym ekranie
* Dostęp do DOM / id elementów / pliku APK/IPA lub kodu - w celu większej automatyzacji i stabilności (zamiast image-based)

**W wyniku system może wygenerować:**

**1. Przypadek testowy (opisowy)**

**Nazwa:** Logowanie z poprawnymi danymi  
  
**Krok 1:** Otwórz ekran logowania.  
**Krok 2:** Wprowadź email test@example.com.  
**Krok 3:** Wprowadź hasło Haslo123.  
**Krok 4:** Kliknij "Zaloguj".  
  
**Oczekiwany rezultat:** Użytkownik zostaje przekierowany do pulpitu głównego, widoczny jest baner "Witaj".

**2. Fragment skryptu - Selenium (Python)**

from selenium import webdriver from selenium.webdriver.common.by import By from selenium.webdriver.common.keys import Keys driver = webdriver.Chrome() driver.get("https://twoja-aplikacja.example/login") driver.find\_element(By.ID, "email").send\_keys("test@example.com") driver.find\_element(By.ID, "password").send\_keys("Haslo123") driver.find\_element(By.ID, "login-button").click() assert "Witaj" in driver.page\_source driver.quit()

**3. Fragment skryptu - SikuliX (pseudo-skrypt)**

# Sikuli (Jython) click("images/login\_button.png") type("images/email\_field.png", "test@example.com") type("images/password\_field.png", "Haslo123") click("images/submit\_button.png") wait("images/dashboard\_welcome.png", 10)

Z otrzymanych przypadków silnik TestSpark automatycznie wygeneruje:

* **Listę przypadków testowych** (np. w Excel / CSV / PDF) - do importu w narzędziach typu TestRail, Jira XRay, Zephyr
* **Plan testów** - podsumowanie obszarów testowych, priorytetów, i zależności ekranów
* **Mapę przepływu ekranów (user flow diagram)** - pokazującą logiczne przejścia między widokami

### **4.3.5.** Funkcjonalność testów regresji GUI

#### **4.3.5.1.** Opis funkcjonalności

**REQ-MOD-GUI-005:** System musi porównywać dwie wersje interfejsu GUI i identyfikować zmiany wizualne i funkcjonalne.

TestGEN umożliwia automatyczne wykrywanie zmian wskazując potencjalne błędy regresji analizując aplikację identyfikując ewentualne niespójności między wersjami kodu lub ekranu użytkownika (porównywanie obrazów).

#### **4.3.5.2.** Szczegółowy przepływ działania

System po wczytaniu dwóch wersji kodu źródłowego lub dwóch różnych wersji danego ekranu jest w stanie zrobić tzw. diff-a i wskazać elementy, które się od siebie różnią. Następnie jest w stanie przygotować opis przypadku testowego do wykonania testu regresji jak i ewentualny kod do przetestowania GUI.

##### 1. Wejście (input)

System przyjmuje:

* **Para lub zestaw zrzutów ekranów** powiązanych logicznie: v1 (stara) oraz v2 (nowa). Mogą to być pojedyncze ekrany lub sekwencje ekranów (flow).
* **Metadane (opcjonalne):** rozdzielczość, device/emulator, timestamp, identyfikator builda, URL/ścieżka ekranu, informacje o DOM (jeśli dostępne).
* **(Opcjonalnie) pliki referencyjne:** mapowanie elementów UI → unikalne ID/selektory (jeśli dostępne, daje dużo wyższą stabilność testów).

Formaty wejścia: PNG/JPEG/SVG, plus opcjonalne JSON z metadanymi.

##### 2. Wstępne przygotowanie i normalizacja obrazów

**Kroki:**

* **Normalizacja rozdzielczości i skalowania** - dopasowanie DPI/skal do wspólnego układu (by zmniejszyć fałszywe różnice spowodowane skalowaniem)
* **Preprocessing** - filtrowanie szumów, ujednolicenie kolorów, opcjonalne usunięcie dynamicznych treści (np. znaczniki czasu) jeśli są rozpoznane
* **Segmentacja UI** - wykrycie regionów: nagłówek, formularz, lista, przyciski, bannery. Techniki: detekcja konturów, heurystyki layoutu, modele wykrywające komponenty (np. ML do rozpoznawania przycisku/pola)
* **OCR** - rozpoznanie tekstu na ekranie i powiązanie go z regionami (etykiety przycisków/placeholdery/komunikaty). To pozwala robić semantyczne porównania (np. "Zaloguj" → "Sign In")

Wynik: ustrukturyzowana reprezentacja ekranu (lista regionów z bounding box, typem komponentu, ekstraktowanym tekstem).

##### 3. Porównanie wersji - diffing (visual + semantic)

System wykonuje dwie klasy analiz diferencyjnych:

**A. Visual diff (pixel/structural)**

* Obliczanie różnic piksel-po-piksel z adaptacyjnym progiem tolerancji (aby ignorować drobne anty-aliasing/skalowanie)
* Mapowanie regionów (region v1 ↔ najbliższy region v2) za pomocą IoU (intersection over union) i dopasowania kształtu
* Generowanie obrazu diff z podświetleniem zmian (added/removed/modified)

**B. Semantic diff (komponentowy)**

* Porównanie wykrytych komponentów po typie i tekście (np. przycisk z etykietą, pole input, komunikat walidacji)
* Rozpoznawanie zmiany: tekst zmieniony, element przesunięty, element usunięty/dodany, style/czcionka zmiana
* Jeśli dostępny DOM/selektory → porównanie selektorów i atrybutów (np. id, class, aria-\*)

**Wynik diffu**

Dla każdego wykrytego elementu system produkuje rekord:

{ element\_id: optional, bbox\_v1: [...], bbox\_v2: [...], type: "button|input|label|list|image|banner|modal", text\_v1: "Zaloguj", text\_v2: "Zaloguj się", visual\_change: "none|moved|resized|style|added|removed|significant", semantic\_change: "none|text\_changed|type\_changed|present\_absent", confidence: 0.87 }

##### 4. Klasyfikacja zmian (wpływ na funkcjonalność)

Dla każdego wykrytego różnicowego elementu system klasyfikuje wpływ na zachowanie użytkownika:

* **Breaking / Behavioral change** - zmiana, która wpływa na działanie (np. przycisk usunięty, pole input zmienione na readonly, tekst "Usuń konto" zmieniony na "Archiwizuj")
* **UI-only / Cosmetic** - tylko wyglądowe (kolor, font, drobne przesunięcie) - zwykle niższy priorytet regresji
* **Potentially risky** - np. etykieta zmieniona na podobną (może być bug), element przeniesiony (może zmienić discoverability)
* **Unknown** - gdy OCR/segmentacja niska pewność

Reguły klasyfikacji opierają się na heurystykach + regułach (np. usunięcie przycisku submit → Breaking).

##### 5. Generowanie przypadków testów regresji - zasady

**Typy testów generowanych:**

* **Baseline check** - czy ekran renderuje (porównanie podstawowych elementów)
* **Functional regression** - testy, które sprawdzają, czy funkcjonalność związana z zmienionymi elementami działa tak jak wcześniej (np. kliknięcie przycisku powoduje oczekiwane przejście)
* **Negative / Validation** - sprawdzenie walidacji i błędów (zwłaszcza jeśli pola/domyślnie wymagane atrybuty zmieniły się)
* **Visual regression** - testy obrazkowe (pixel-diff) z tolerancjami
* **Accessibility/Label checks** - etykiety aria/tekst alternatywny (jeśli OCR/DOM wskazuje zmiany)
* **Exploratory hints** - sugestie do ręcznego przeglądu (np. "sprawdź nowe CTA w headerze")

Każdy testowy przypadek dostaje: ID, opis, kroki, dane wejściowe, oczekiwany rezultat, typ (functional/visual/negative), priorytet, powiązanie z rekordem diffu i score ufności.

##### 6. Generacja kodu testowego: krok po kroku

1. Wejście pary zrzutów → normalizacja i strukturalizacja (patrz etap 2).
2. Diffing → otrzymanie listy zmian z metadanymi i confidence.
3. Filtracja → usuń zmiany poniżej minimalnego progu (np. confidence < 0.4) albo kosmetyczne jeśli użytkownik chce tylko functional.
4. Mapowanie na scenariusze:

* Dla usuniętych elementów: generuj TC sprawdzający brak elementu i TC alternatywny sprawdzający czy funkcjonalność dostępna inaczej.
* Dla dodanych elementów: TC weryfikujący obecność i TC integracyjny, jeśli element inicjuje akcję.
* Dla tekst zmieniony: TC sprawdzający, że logika nie zależy od etykiety (np. poszukaj selektora bez tekstu) i TC akceptacyjny sprawdzający przekaz.
* Dla przesuniętych/resized: visual regression TC z tolerancją; e2e TC sprawdzający czy klik działa.

1. Generacja kroków testowych — krok → akcja → dane → asercja. Każdy krok powiązany z konkretnym regionem/selektorem lub wzorcem obrazu.
2. Generacja danych testowych — automatyczne przypisanie przykładowych danych lub wykorzystanie istniejącej bazy danych testowej/fixtures.
3. Priorytetyzacja — score = f(change\_type, element\_criticality, coverage\_history). Krytyczne zmiany → wysoki priorytet.
4. Output — zestaw TC w formatach: CSV/Excel, Gherkin .feature, JSON (do integracji), i folder z kodem-skeletonami testów.

System generuje równolegle dwa rodzaje testów (jeśli to możliwe):

**A. DOM-based** (najbardziej stabilne, gdy dostęp do selektorów / DOM jest możliwy)

* Playwright (JS/TS/Python) - preferowany dla nowoczesnych webów, szybkie i multi-browser
* Selenium (Python/Java) - szerokie zastosowanie

**B. Image-based** (jeżeli nie ma DOM lub tylko screenshoty)

* SikuliX (Jython/Java) - dopasowanie obrazów (pattern matching)
* PyAutoGUI / OpenCV - prostsze makra/automatyzacja

**C. Desktop / Native**

* WinAppDriver / pywinauto dla Windows
* Appium dla aplikacji mobilnych

Dla każdego TC system generuje:

* Plik testowy (np. tests/test\_login\_regression.py)
* Page Object / locator map (jeśli DOM-based)
* Obrazy referencyjne + diff images (jeśli image-based)
* Fixtures/mocks (jeśli potrzebne)
* CI job skeleton (np. GitHub Actions YAML)

##### 7. Przykładowe wygenerowane testy

**A. Playwright (Python) - DOM-based test sprawdzający, że zmieniony przycisk działa:**

# tests/test\_regression\_button\_click.py from playwright.sync\_api import Page def test\_cta\_click\_navigates\_to\_new\_page(page: Page): page.goto("https://app.example.com/some-screen") # locator wygenerowany z DOM lub heurystyki (fallback: text) cta = page.locator("button:has-text('Zaloguj się')") # lub "#cta-login" assert cta.is\_visible() cta.click() page.wait\_for\_url("\*\*/dashboard") assert "Witaj" in page.text\_content("h1")

**B. SikuliX (Jython) - image-based test sprawdzający, że przycisk istnieje i działa:**

# sikuli script (Jython) click("images/v2/cta\_login.png") wait("images/v2/dashboard\_header.png", 10) assert exists("images/v2/dashboard\_header.png")

**C. Playwright (visual regression) - pixel diff z tolerancją:**

def test\_visual\_regression\_main\_section(page: Page): page.goto("https://app.example.com/some-screen") screenshot = page.locator("#main-section").screenshot() # porównanie z saved baseline from PIL import Image, ImageChops baseline = Image.open("images/baseline/main\_section\_v1.png") current = Image.open(screenshot\_path) diff = ImageChops.difference(baseline, current) # Count non-zero pixels non\_zero = sum(1 for px in diff.getdata() if px != (0,0,0,0)) assert non\_zero < 500 # threshold wygenerowany automatycznie

**D. Gherkin feature (wyeksportowane):**

Feature: Regression test for Login screen Scenario: CTA "Zaloguj się" still navigates to dashboard Given I open "/some-screen" When I click the "Zaloguj się" button Then I should be on "/dashboard" And I should see "Witaj" in header

#### **4.3.5.3.** Przebiegi funkcjonalności

**Przebieg pozytywny:**

System będzie w stanie wyłapać różnice w GUI i przygotować opisowe przypadków testowych do wykonania testów regresji jak i ewentualny kod do przetestowania GUI.

**Przebieg negatywny:**

Np. w przypadku wczytania identycznych ekranów system nie będzie w stanie przygotować testów regresji wskazując, że wczytano te same materiały.

### **4.3.6.** Wykorzystane mechanizmy i technologie

#### **4.3.6.1.** Mechanizm wczytywania i interpretacji diagramów UML

Mechanizm będzie odpowiedzialny za wczytywanie i interpretację diagramów UML (takich jak diagramy przypadków użycia, aktywności czy sekwencji). Do przetwarzania danych graficznych mogą zostać wykorzystane parsery UML (np. PlantUML, GraphML) lub sieci transformer do rozpoznawania diagramów, które pozwolą na identyfikację elementów kluczowych: aktorów, akcji, zależności, warunków oraz przepływów sterowania. Te informacje zostaną następnie przetworzone do postaci ustrukturyzowanej, możliwej do dalszej analizy i interpretacji.

**Przykładowe narzędzia UML:**

* PlantUML - parsowanie tekstowych diagramów UML
* XMI parsers - przetwarzanie standardowego formatu wymiany UML
* Eclipse UML2 - biblioteka Java do pracy z modelami UML
* Papyrus - narzędzie modelowania z API

#### **4.3.6.2.** Mechanizm tworzenia opisowych przypadków dla GUI i UML

Mechanizm będzie umożliwiał analizę GUI, tj. ekranów użytkownika w postaci zrzutów ekranowych lub struktury interfejsu np. z plików graficznych lub XML.

Na podstawie wczytanych diagramów UML i/lub GUI zostaną przygotowane przepływy użytkownika i działania systemu. W oparciu o te dane zostaną wygenerowane przypadki testowe odzwierciedlające scenariusze biznesowe i operacyjne. Dla nich zostaną utworzone opisy testów, przykładowe dane wejściowe, kroki, warunki początkowe i końcowe oraz oczekiwane rezultaty.

Tak jak w przypadku poprzednich modułów, testy mogą być klasyfikowane np. jako pozytywne, negatywne lub brzegowe.

#### **4.3.6.3.** Mechanizm generowania kodu do uruchomienia testu GUI

Automatyczne generowanie kodu testowego dla wskazanych klas przypadków identyfikując elementy GUI i przypisane im akcje. Zastosowane zostaną algorytmy transformujące logikę przypadków testowych na kod wykonywalny.

**Techniki generacji kodu testowego:**

* Template-based generation - wykorzystanie szablonów kodu
* AST manipulation - manipulacja drzewem składniowym
* Code synthesis z LLM - wykorzystanie modeli językowych
* Pattern matching - dopasowywanie wzorców testowych

**UWAGA:** Niniejszy dokument przekroczył limit znaków dla jednej odpowiedzi. Dokument będzie kontynuowany w kolejnej części, obejmującej pozostałe rozdziały:

* 4.4. Moduł analizy bezpieczeństwa
* 4.5. Moduł analizy luk funkcjonalnych
* Rozdziały 5-10

## **4.4.** Moduł inteligentnej analizy luk w kodzie oprogramowania i generowania rekomendacji zabezpieczeń

**B. Rozbudowa silnika TestSpark pod kątem audytów bezpieczeństwa oraz analizy zgodności kodu z dokumentacją**

**Termin realizacji: 1 czerwca 2026 - 30 września 2026**

W ramach zadania 2 moduł TestSpark, kluczowy komponent systemu TestGEN, zostanie rozszerzony o innowacyjne mechanizmy analizy zgodności kodu i dokumentacji, zaawansowane testowanie bezpieczeństwa, a także możliwość badania istniejącego oprogramowania pod kątem podatności na zagrożenia. Dzięki temu TestGEN stanie się nie tylko narzędziem wspierającym projektowanie nowych systemów, ale również aktywnym audytorem bezpieczeństwa wdrożonych aplikacji. Jednym z kluczowych aspektów rozbudowy TestGEN będzie jego zdolność do ciągłego samodoskonalenia. System będzie mógł regularnie otrzymywać nowo odkryte podatności, analizując dane o cyberzagrożeniach i ucząc się nowych metod wykrywania luk. Dzięki temu będzie

identyfikował zagrożenia, które wcześniej nie były znane.

Zadanie 1: Mechanizm analizy zgodności i wykrywania luk funkcjonalnych

TestGEN będzie automatycznie porównywał kod źródłowy, dokumentację techniczną, podręczniki użytkownika oraz diagramy UML, wykorzystując algorytmy AI do analizy spójności projektu. Zaawansowane techniki przetwarzania języka naturalnego pozwolą na ekstrakcję wymagań i ich zestawienie z implementacją aplikacji. System wykryje brakujące walidacje, niezaimplementowane operacje oraz niespójności między UML a dokumentacją, generując raport z zaleceniami. TestGEN nie tylko oznaczy te niezgodności, ale także zasugeruje brakujące funkcjonalności wynikające z dokumentacji systemu. Dzięki temu testowanie stanie się integralną częścią procesu programistycznego, eliminując błędy przed wdrożeniem, oszczędzając czas i zasoby.

Zadanie 2: Testowanie bezpieczeństwa i zdolność do samodoskonalenia

TestGEN przetestuje tworzone oprogramowanie, jak i pozwoli na analizę istniejących aplikacji. Po wgraniu kodu wdrożonego systemu przeprowadzi audyt bezpieczeństwa, wykrywając podatności i zalecając poprawki. TestGEN będzie mógł regularnie otrzymywać nowe dane o zagrożeniach, które zostaną przeanalizowane i wykorzystane do aktualizacji metod testowania. Metody analizy ryzyka pozwolą na identyfikację podatności, których nie wykrył wcześniej, stale podnosząc poziom ochrony systemów IT.

Zadanie 3: Technologie i sposób realizacji

W rozwoju modułu TestGEN zostaną zastosowane frameworki NLP, systemy analizy kodu, UML oraz mechanizmy analizy bezpieczeństwa, a także modele AI oraz algorytmy wykrywania luk.

### **4.4.1.** Opis ogólny modułu i wykorzystane technologie

Moduł inteligentnej analizy luk w kodzie oprogramowania i generowania rekomendacji zabezpieczeń dla oprogramowania aplikacyjnego oraz wbudowanego stanowi kluczowy komponent systemu TestGEN w zakresie bezpieczeństwa. Jego głównym zadaniem jest automatyczne wykrywanie podatności oraz sugerowanie skutecznych działań naprawczych, zarówno dla nowo tworzonego, jak i już wdrożonego oprogramowania na podstawie kodu źródłowego.

**Technologie:** Realizacja modułu może wykorzystać technologie takie jak AST parsers i linters, SonarQube, Checkstyle i PMD.

Po wgraniu kodu źródłowego do TestGEN, użytkownik uzyskuje szczegółową analizę podatności i zalecenia dotyczące ich eliminacji. Moduł ten realizuje innowacyjne podejście do testów bezpieczeństwa, wykorzystując mechanizmy analizy semantycznej oparte na sztucznej inteligencji.

Realizacja modułu może wykorzystać technologie takie jak AST parsers i linters, SonarQube, Checkstyle i PMD.

**REQ-MOD-SEC-001:** System musi wykrywać co najmniej następujące typy podatności: SQL Injection, XSS, CSRF, błędy autoryzacji, niewłaściwe zarządzanie sesjami, hardcoded secrets.

**REQ-MOD-SEC-002:** System musi generować konkretne rekomendacje naprawcze z przykładami kodu dla każdej wykrytej podatności.

### **4.4.2.** Funkcjonalność analizy luk bezpieczeństwa kodu i generowanie przypadków testów bezpieczeństwa

#### **4.4.2.1.** Opis funkcjonalności

System jest w stanie wygenerować opisy przypadków testowych dla testów bezpieczeństwa i ewentualny kod do ich uruchomienia dla najpopularniejszych podatności.

System analizuje ewentualne luki w kodzie w automatyczny sposób i wykrywa podatności oraz przygotowuje raport rekomendacji zabezpieczeń w danym zakresie jeśli znajdzie jakąś lukę.

W ten sposób TestGEN nie tylko wykrywa błędy, ale także sugeruje skuteczne działania naprawcze. System poprzez analizę kodu źródłowego identyfikuje:

* SQLi
* XSS
* Nieprawidłowe filtrowanie danych wejściowych
* Brak tokenów CSRF
* Błędy w logice autoryzacyjnej
* Proponowanie konkretne zmiany w kodzie, eliminujące wykryte zagrożenia w postaci raportu z rekomendacjami

Wejście — co TestGEN przyjmuje:

* Repozytorium kodu (git clone / archive) lub pojedyncze pliki źródłowe (wielojęzykowe).
* Opcjonalnie: pliki konfiguracyjne środowiska (docker-compose, k8s manifests), pliki IaC (Terraform), schematy DB, test data, środowisko uruchomieniowe (dev/staging).
* (Opcjonalnie) dostęp do running instance (staging) — do testów dynamicznych (DAST).
* Metadane: zakres skanowania, polityka (które klasy luk ignorować), klucze API do narzędzi zewnętrznych.

#### **4.4.2.2.** Architektura modułu bezpieczeństwa

Architektura modułu bezpieczeństwa składa się z następujących głównych komponentów:

* **Preprocessor + Detector technologii** - rozpoznaje język, framework, pliki zależności (package.json, requirements.txt, pom.xml)
* **SAST pipeline (static analysis)** - AST parsing, pattern/rule matching, semgrep/Sonar-like rules, custom rules
* **Taint / Data-flow analyzer** - śledzi przepływ danych (taint analysis) od źródeł wejścia do sinków (DB, eval, innerHTML, exec)
* **Dependency & supply-chain scanner** - sprawdza biblioteki i wersje (CVE, OSS advisories, Snyk/OSV DB)
* **Configuration & IaC scanner** - wykrywa błędne ustawienia (np. brak HTTPS, publiczne credentials)
* **DAST / Runtime tests** - uruchamianie OWASP ZAP, Burp, sqlmap, fuzzery przeciwko instancji testowej (jeśli dostępna)
* **Proof-of-Concept (PoC) generator** - generuje bezpieczne, kontrolowane POC / testy sprawdzające podatność (wykonywane tylko w autoryzowanym środowisku)
* **Fix generator** - sugeruje poprawki kodu (patch/diff), snippet'y i bezpieczne wzorce
* **Prioritizer / Risk engine** - mapuje do CWE/CVSS, ocenia ryzyko, sugeruje kolejność napraw
* **Report & CI integrator** - raport, JIRA tickets, MR suggestions, Git hooks, CI joby

#### **4.4.2.3.** Szczegółowy workflow po wczytaniu kodu źródłowego

##### Krok A - wykrycie technologii i parsowanie

* Rozpoznanie języka i frameworku (np. React/Node, Django, Spring)
* Budowa AST (np. tree-sitter, Python ast, JavaParser)
* Wydobycie funkcji HTTP endpoints, kontrolerów, handlerów, miejsca gdzie przyjmowane są dane od użytkownika, zapytań do bazy i generowania HTML/JS

##### Krok B - reguły SAST + pattern matching

Uruchomienie zestawu reguł typu semantic + regex (np. Semgrep-like):

* Wykrycie String concatenation w zapytaniach do bazy
* innerHTML przypisywanego z danymi użytkownika
* eval() w JavaScript
* Brak walidacji inputów przy wywołaniu API

Dla frameworków natywnych:

* Szukanie braków CSRF tokenów w formularzach (np. brak @csrf\_protect w Django view)
* Brak sprawdzania uprawnień w kontrolerach (np. brak sprawdzenia current\_user.role)

##### Krok C - taint / data-flow analysis

**Oznaczenie źródeł:** request params, headers, cookies, body, form inputs

**Oznaczenie sinków:** zapytania do DB (raw SQL, ORM raw queries), funkcje renderujące HTML (innerHTML, res.send() zawierające user data), systemowe exec/OS calls, LDAP, LDAP injections, deserialization

**Śledzenie przepływu:** czy niesprawdzane dane trafiają do sinka bez sanitization/encoding/parametrization? Jeśli tak → potencjalna podatność

**Rozpoznawanie sanitizers/encoders:**

* Presence of parameterized queries (PreparedStatement, parameterized ORM bindings)
* Output encoding libraries (e.g., escapeHtml)
* Input validation libs

##### Krok D - dependency & config scanning

* Zidentyfikowane biblioteki skanowane względem baz CVE/OSV/Snyk
* Wykrycie podatnych wersji, braków patcha, użycia insecure flags/configs (np. allow\_remote=true)
* Searching for hardcoded secrets (regex for keys), misconfigured CORS (\*), cookies without Secure/HttpOnly, insecure TLS settings

##### Krok E - dynamic checks (jeśli staging dostępny)

* Uruchomienie kontrolowanych testów DAST: OWASP ZAP baseline scan, skan interaktywny
* Próby prostych payloadów SQLi/XSS
* Test CSRF by simulating form posts without token
* Fuzzing endpoints (bounded), testy sesji i role escalation (brute-force role change in REST calls)
* Zbieranie odpowiedzi HTTP, stack traces, headers, cookies

##### Krok F - agregacja dowodów i eliminacja false positives

* Połączenie wyników SAST + taint + DAST → podniesienie confidence gdy taint wykryje przepływ i DAST potwierdzi exploit
* Heurystyki redukujące FP: sprawdzanie użycia ORM safe APIs, wykrycie sanitizers, checking for prepared statements
* Nadanie score ufności i klasyfikacji (Confirmed / Likely / Possible)

#### **4.4.2.4.** Wykrywanie konkretnych rodzajów podatności

##### SQL Injection

**Jak wykrywane:**

Taint analysis: request.params.name → konkatenacja do stringa w execute("SELECT ... " + name) lub format() → sink: db.execute(raw\_sql).

**Testy wygenerowane:**

Testy jednostkowe i integracyjne z payloadami (np. ' OR '1'='1), automatyzowane skrypty uruchamiające sqlmap lub curl z złośliwym parametrem w staging.

**Rekomendowane poprawki:**

Parametryzacja (prepared statements / query parameters) lub korzystanie z ORM bindingów.

**Przykładowa poprawka (Python / psycopg2):**

# vulnerable cursor.execute("SELECT \* FROM users WHERE name = '" + name + "'") # fixed cursor.execute("SELECT \* FROM users WHERE name = %s", (name,))

##### Cross-Site Scripting (XSS)

**Jak wykrywane:**

User input przepływa do HTML bez output encoding (np. res.send("

" + req.body.comment + "

") lub element.innerHTML = userInput w frontend).

**Testy wygenerowane:**

Testy E2E imitujące payloady <script>alert(1)</script> i asercje że w DOM nie pojawia się wykonywalny skrypt; DAST detects reflected/persistent XSS.

**Rekomendacja:**

Output encoding przed renderowaniem, użycie template engines automatycznie escapujących, CSP headers.

**Fix (Node/Express + template engine):**

// vulnerable res.send(`

${userInput}

`); // fixed - using template engine with auto-escaping res.render('template', { data: userInput }); // template engine escapes automatically // or manual escaping const escapeHtml = require('escape-html'); res.send(`

${escapeHtml(userInput)}

`);

##### Brak CSRF (Cross-Site Request Forgery)

**Jak wykrywane:**

Mutujące POST/PUT/DELETE bez CSRF tokenów lub bez weryfikacji SameSite cookie.

**Test:**

Wysłanie żądania POST z sesją użytkownika z innego źródła i oczekiwanie braku akcji - test symuluje CSRF.

**Rekomendacja:**

Dodać CSRF middleware, generować tokeny per session/form, dodać SameSite=strict i wymusić header Origin/Referer checks.

**Patch snippet (Flask-WTF):**

from flask\_wtf import CSRFProtect csrf = CSRFProtect(app) # In templates

Początek formularza

{{ csrf\_token() }}

Dół formularza

**Patch snippet (Python Django):**

# ensure @csrf\_protect on view or have middleware enabled from django.views.decorators.csrf import csrf\_protect @csrf\_protect def my\_view(request): # view logic pass

##### Błędy autoryzacji/logiki (Broken Access Control)

**Jak wykrywane:**

Endpoints performing sensitive actions without verifying the caller's identity/role; direct object references without access checks (GET /user/123/orders where user id from token not validated).

**Testy wygenerowane:**

Role-escalation tests: call endpoints as user A for resources of user B and expect 403. Automated scripts alter JWT sub claims or use different tokens.

**Rekomendacja:**

Centralize authorization checks, use middleware/policies, enforce object-level checks.

**Fix (pseudo):**

# check ownership if resource.owner\_id != current\_user.id: raise Forbidden() # or use decorator @require\_ownership def get\_resource(resource\_id): return Resource.get(resource\_id)

##### Nieprawidłowe filtrowanie / walidacja danych

**Jak wykrywane:**

Absence of schema validation on input; reliance on client-side validation.

**Testy wygenerowane:**

Fuzzing with boundary and malformed inputs; schema-based tests using JSON schema validators.

**Rekomendacja:**

Validate at server-side (schema validation e.g., AJV for Node, pydantic for Python FastAPI).

# Python with Pydantic from pydantic import BaseModel, EmailStr, validator class UserInput(BaseModel): email: EmailStr age: int @validator('age') def age\_must\_be\_positive(cls, v): if v < 0 or v > 150: raise ValueError('Age must be between 0 and 150') return v # Use in endpoint @app.post("/user") def create\_user(user: UserInput): # user is already validated pass

#### **4.4.2.5.** Generowanie przypadków testów bezpieczeństwa

Dla każdej wykrytej podatności TestGEN wygeneruje:

##### a. Opisowy przypadek testowy (readable)

Cel, warunki, kroki, dane, oczekiwane rezultaty, priorytet.

**TC\_SEC\_SQLI\_001:** "POST /search - SQL injection attempt via q parameter"  
  
**Kroki:**  
1. POST body q: "' OR '1'='1"  
2. Oczekuj HTTP 200 or empty results; no stack trace; no DB error leak  
  
**Oczekiwane:** 4xx lub pusta odpowiedź, brak ujawnienia błędów DB  
**Typ:** Security/Negative  
**Priorytet:** Krytyczny

##### b. Automatyczny skrypt testowy

Np. pytest + requests (API), Playwright (jeśli endpoint wywoływany z UI), lub sqlmap wrapper.

import requests def test\_sql\_injection\_protection(): """Test that SQL injection is prevented""" payload = {"q": "' OR '1'='1"} response = requests.post("https://api.example.com/search", json=payload) # Should not return all results assert response.status\_code in [400, 403, 422] # Should not leak database errors assert "sql" not in response.text.lower() assert "syntax error" not in response.text.lower()

##### c. DAST script / ZAP policy

Konfiguracja skanu ataku dla konkretnych endpointów.

##### d. Unit / Integration tests

Checks for input validation and sanitizers.

##### e. PoC (bezpieczny, kontrolowany)

Krótkie, jasno oznaczone, z limitem (sleep, timeouts) i wymaganiem explicit consent do uruchomienia.

##### f. Regression tests

Dodane do repo tests/security/ i CI, aby zapobiec regresji.

Przykład opisowego TC + pytest (API SQLi), opisowy TC:

**ID: TC\_SEC\_API\_SQLI\_001**

**Cel: Zweryfikować, że parametr `username` w POST /users/search nie jest podatny na SQLi.**

**Warunki: staging instance z test DB**

**Kroki:**

**1. POST /users/search { "username": "' OR '1'='1" }**

**2. Sprawdź status odpowiedzi i zawartość (nie powinno zwrócić wszystkich użytkowników ani błędu DB)**

**Oczekiwane: 400 lub 200 z pustym/odfiltrowanym wynikiem; brak stack trace.**

#### **4.4.2.6.** Rekomendacje naprawcze

Dla każdej klasy podatności system generuje:

* **Krótkie wyjaśnienie** (co się stało i dlaczego jest to niebezpieczne)
* **Konkretne sugestie kodowe** z fragmentami poprawionego kodu (dla wykrytego języka)
* **Alternatywy projektowe** (np. użycie ORM zamiast ręcznych query, wprowadzenie centralnego middleware autoryzacji)
* **Lista testów regresyjnych** które należy dodać po wprowadzeniu poprawki
* **Priorytety i estimated effort** (szacunek trudności naprawy)
* **Mapowanie do standardów** (OWASP Top10, powiązane CWE), co ułatwia komunikację z audytorami

**Przykłady rekomendacji:**

|  |  |  |
| --- | --- | --- |
| **Podatność** | **Rekomendacja** | **Priorytet** |
| SQLi | Użyj PreparedStatement/parametrów; waliduj i castuj typy; ogranicz przywileje DB | Krytyczny |
| XSS | Encoduj kontekstowo; CSP; HttpOnly cookie, sanitycznie renderuj JSON w responses | Wysoki |
| CSRF | Dodać CSRF tokens, wymuszanie SameSite i referer/origin checks | Wysoki |
| AuthZ | Przenieść checks do warstwy service + centralny policy enforcer (RBAC/ABAC) | Krytyczny |

#### **4.4.2.7.** Raport końcowy i traceability

System generuje raport końcowy zawierający:

* **Executive summary** (najwyższe ryzyka, CI build id, data)
* **Szczegółowa lista znalezionych problemów** (ID, severity, CWE/OWASP tag, plik:linia, trace of dataflow)
* **Zalecane zmiany i automatyczne patch snippets**
* **Wygenerowane testy** (ścieżki plików) i instrukcje uruchomienia
* **SBOM + lista zależności z CVE**
* **Macierz traceability:** vuln ↔ test ↔ patch ↔ CI job
* **Lista pytań / niepewności** gdzie wymagana jest manualna weryfikacja (human-in-loop)

#### **4.4.2.8.** Integracja w CI/CD i workflow naprawczy

System generuje joby do CI (GitHub Actions/GitLab CI/Jenkins) uruchamiające:

* SAST
* Dependency scan
* Generated security tests (DAST/pytest)
* Report aggregation

Przy wykryciu krytycznego problemu:

* Automatyczne otwarcie ticketu (JIRA) z opisem, patch snippet, i linkiem do PR suggestion

Po wprowadzeniu patcha:

* TestGEN automatycznie uruchamia regression tests (w tym wygenerowane security tests) i oznacza status FIXED / NOT FIXED

#### **4.4.2.9.** Human-in-the-loop, scoring i ograniczenia

* Każda automatyczna sugestia zawiera score ufności. Niskie score'y trafiają do reviewerów bezpieczeństwa.
* Dla krytycznych zmian (np. auth bypass), wymagana ręczna walidacja i code review.
* Ograniczenia: analiza może generować false positives (zwłaszcza przy dynamicznych językach i metaprogramowaniu) — dlatego rekomenduję połączenie: SAST + DAST + IAST + manual review.
* Nie generuję publicznego exploitu — POC są defensywne i zakładają środowisko testowe.

#### **4.4.2.10.** Przykład end-to-end: wykrycie SQLi i ścieżka naprawy

**Scenariusz:**  
1. TestGEN przetwarza repo → znajduje searchController.js gdzie query budowane jest "... WHERE name LIKE '%" + q + "%'"  
2. Taint analysis: req.query.q → buildQuery → db.exec. Confidence: wysoka  
3. Generuje: TC\_SEC\_SQLI\_001 (opis + pytest), DAST run z payload, i rekomendowany patch (parameterized queries)  
4. Generuje PR template z propozycją zmiany i testem regresyjnym  
5. Po merge - CI uruchamia testy i potwierdza fix

// before const sql = `SELECT \* FROM products WHERE name LIKE '%${q}%'`; db.query(sql, ...); // after const sql = `SELECT \* FROM products WHERE name ILIKE $1`; db.query(sql, [`%${q}%`]);

### **4.4.3.** Funkcjonalność aktualizacji systemu

#### **4.4.3.1.** Opis funkcjonalności

**REQ-MOD-SEC-003:** System musi umożliwiać regularne aktualizacje bazy podatności (co najmniej raz w miesiącu).

TestGEN nie jest narzędziem statycznym - jego zdolność do ciągłego samodoskonalenia umożliwia analizowanie i przyswajanie nowo odkrytych podatności. System będzie regularnie aktualizowany, aby wykrywać zagrożenia, które wcześniej były nieznane. Dzięki temu właściciele oprogramowania mogą ponownie przeprowadzać analizy, aby zabezpieczyć swoje systemy przed najnowszymi zagrożeniami.

#### **4.4.3.2.** Security Knowledge Layer

System będzie aktualizowany o warstwę wiedzy bezpieczeństwa (Security Knowledge Layer) - dynamiczną bazę reguł, CVE, exploit patternów, heurystyk i przykładów napraw.

TestGEN będzie zasilany kilkoma klasami źródeł danych, które aktualizują jego wiedzę o zagrożeniach:

|  |  |  |
| --- | --- | --- |
| **Typ źródła** | **Przykłady** | **Zastosowanie** |
| Bazy CVE/NVD | cve.mitre.org, nvd.nist.gov | Identyfikacja nowych podatnych bibliotek, frameworków, API |
| Bazy podatności open source (SCA) | Snyk, GitHub Security Advisories, OSV.dev, PyUp, npm audit, Maven Central advisories | Automatyczne wykrycie, że np. django==4.1.2 ma CVE |
| Projekty OWASP | OWASP Top 10, ASVS, API Security Top 10 | Aktualizacja modeli ryzyka i nowych kategorii zagrożeń |
| Bug bounty / exploit feeds | HackerOne, Exploit-DB, CISA KEV, ZeroDay Initiative | Uczenie się nowych wektorów ataków, np. łańcuchów XSS → RCE |
| Rządowe / cert feedy | CERT, CISA, ENISA | Wykrywanie trendów i alertów sektorowych |
| Społeczność i GitHub commits | Pull requests naprawiające CVE w popularnych projektach | Uczenie się skutecznych wzorców napraw |

#### **4.4.3.3.** Pipeline aktualizacyjny

##### Etap 1: Pobranie i ekstrakcja danych

* Pobranie nowych rekordów CVE, advisories i changelogi bibliotek
* Dla każdej podatności wyodrębniane są:
  + Opis (CVE summary)
  + Dotknięte wersje
  + Typ błędu (np. SQLi, path traversal, XSS)
  + Przykładowe fragmenty kodu przed/po naprawie
  + Język i framework

##### Etap 2: Klasyfikacja semantyczna (LLM + parsery)

* LLM analizuje tekst CVE i klasyfikuje go wg:
  + Kategorii (OWASP/CWE)
  + Typu źródła błędu (input validation, unsafe deserialization, logic flaw)
  + Dotkniętych technologii (np. React, Flask, Spring Boot)
* Wynik: nowy "wzorzec podatności" zapisany w formacie semgrep / AST rule

##### Etap 3: Aktualizacja repozytorium reguł i testów

System dodaje nową regułę detekcji do repozytorium SAST/DAST.

**Przykład - dla CVE-2025-XXXX w Django generuje regułę:**

rules: - id: django-missing-csrf-token pattern: | def $FUNC(...): ... pattern-not-inside: | @csrf\_protect def $FUNC(...): ... message: "View function missing CSRF protection" severity: WARNING languages: [python] metadata: cve: CVE-2025-XXXX owasp: A01:2021

Jednocześnie generuje przykładowe testy jednostkowe i integracyjne, aby wykryć błędne użycie (np. brak tokenu CSRF).

##### Etap 4: Weryfikacja i testowanie nowych reguł

* Nowa reguła jest testowana na zestawie przykładowych projektów (pozytywne i negatywne przypadki)
* Ocena false positive rate
* Jeśli FPR < 5%, reguła jest dodawana do produkcji

##### Etap 5: Deploy i notyfikacja użytkowników

* Aktualizacja jest wdrażana do systemu TestGEN
* Użytkownicy otrzymują powiadomienie o nowych możliwościach detekcji
* Sugestia ponownego skanowania projektów dla wykrycia nowych zagrożeń

### **4.4.4.** Wykorzystane mechanizmy i technologie

#### **4.4.4.1.** Architektura modułu

* **Mechanizm identyfikujący niskopoziomowe wzorce podatności** w kodzie źródłowym
* **Warstwa semantycznej analizy kodu**, wykorzystująca autorskie modele AI i techniki NLP, które umożliwią rozpoznanie kontekstu funkcjonalnego oraz przepływu danych w systemie

#### **4.4.4.2.** Mechanizm wykrywania podatności i generowania rekomendacji

Na podstawie wyników statycznej i semantycznej analizy kodu, system identyfikuje nieprawidłowe filtrowanie danych wejściowych, brak tokenów CSRF, błędy w logice autoryzacyjnej, a następnie zasugeruje konkretne zmiany w kodzie, mające na celu eliminację wykrytych zagrożeń.

**Przykładowe narzędzia SAST:**

* Semgrep - pattern-based static analysis
* Bandit - Python security linter
* ESLint security plugins - JavaScript/TypeScript
* SpotBugs + Find Security Bugs - Java
* Brakeman - Ruby on Rails security scanner

#### **4.4.4.3.** Mechanizm aktualizacji

W celu samodoskonalenia system będzie mógł być regularnie aktualizowany, np. raz w miesiącu, aby wykrywać zagrożenia, które wcześniej były nieznane.

## **4.5.** Moduł analizy luk funkcjonalnych w kodzie i generowania inteligentnych rekomendacji rozszerzeń

### **4.5.1.** Opis ogólny modułu

Moduł analizy luk funkcjonalnych w kodzie i generowania inteligentnych rekomendacji rozszerzeń dla oprogramowania aplikacyjnego oraz wbudowanego ma za zadanie nie tylko identyfikację niespójności w testowanym oprogramowaniu, ale także aktywny udział w jego rozwoju poprzez proponowanie brakujących funkcji, które powinny zostać wdrożone dla poprawy spójności i użyteczności biznesowej.

**Technologie:** Wykorzystane zostaną technologie NLP, techniki mapowania semantycznego oraz analizy ontologicznej.

**REQ-MOD-FUNC-001:** System musi identyfikować rozbieżności między kodem a dokumentacją biznesową i techniczną.

**REQ-MOD-FUNC-002:** System musi generować rekomendacje nowych funkcjonalności na podstawie analizy wieloźródłowej.

Moduł wykorzysta zintegrowaną analizę z wielu źródeł, np. kodu źródłowego, dokumentacji technicznej, diagramów UML oraz przypadków testowych, aby zidentyfikować niewdrożone funkcjonalności, luki w procesach oraz pominięte elementy, które nie zostały uwzględnione na etapie implementacji.

### **4.5.2.** Funkcjonalność analizy zgodności kodu z dokumentacją

#### **4.5.2.1.** Opis funkcjonalności

Analiza struktury kodu źródłowego i jego zgodności z dokumentacją - TestGEN automatycznie identyfikuje niespójności między kodem a dokumentacją. Jeśli do TestGEN wgra się kod oprogramowania oraz dokumentację, to system powinien sprawdzić, czy w kodzie są odwołania do funkcji.

Audyt bezpieczeństwa kodu (identyfikacja podatności) pod kątem analizy zgodności z dokumentacją np. czy w kodzie są funkcje/procedury o nazwach wspomnianych w dokumentacji.

* Porównywanie dokumentacji technicznej z implementacją kodu
* Generowanie raportów wskazujących brakujące funkcjonalności i zalecenia dotyczące poprawy jakości oprogramowania

#### **4.5.2.2.** Szczegółowy przepływ działania

**Przykładowy przepływ:**

[Wczytanie dokumentacji i kodu]

↓

[Analiza dokumentacji → ekstrakcja wymagań funkcjonalnych]

↓

[Analiza kodu → ekstrakcja struktury i funkcji]

↓

[Porównanie semantyczne LLM]

↓

[Identyfikacja braków i niespójności]

↓

[Raport + rekomendacje + testy automatyczne]

Celem modułu Analizy Zgodności Kod–Dokumentacja w systemie TestGEN jest:

* sprawdzenie, czy implementacja odpowiada temu, co opisano w dokumentacji technicznej lub projektowej,
* wykrycie brakujących lub nadmiarowych funkcji,
* identyfikacja potencjalnych luk bezpieczeństwa lub błędów w logice, wynikających z rozbieżności między dokumentacją a kodem.

Proces można podzielić na sześć logicznych etapów:

##### Etap 1: Wczytanie danych wejściowych

Użytkownik (np. QA Engineer lub Software Architect) wgrywa do TestGEN:

* Dokumentację techniczną lub projektową (np. pliki .md, .pdf, .docx, .txt, .html)
* Kod źródłowy projektu (np. repozytorium ZIP, link do Git, folder z .py, .js, .java itp.)
* Opcjonalnie: diagramy UML lub pliki OpenAPI / Swagger, jeśli istnieją

System rozpoznaje język programowania, strukturę projektu i format dokumentacji.

##### Etap 2: Ekstrakcja informacji z dokumentacji

Moduł LLM Document Analyzer analizuje treść dokumentacji i wyodrębnia z niej:

* Listę funkcji, klas, modułów i API, które powinny istnieć
* Opisy ich parametrów, typów danych, efektów działania, wyjątków, ról użytkowników, poziomów autoryzacji
* Powiązania między komponentami (np. "Moduł UserService korzysta z AuthManager")

**Wynikiem tego etapu może być struktura danych w formacie:**

{ "functions\_expected": [ { "name": "authenticate\_user", "description": "Waliduje dane logowania użytkownika i zwraca token JWT", "params": ["username", "password"], "returns": "JWT token", "security": ["check\_password\_hash", "CSRF token required"] }, { "name": "reset\_password", "description": "Wysyła e-mail resetujący hasło użytkownika", "params": ["user\_email"], "returns": "status\_code" } ] }

##### Etap 3: Analiza kodu źródłowego

Moduł Code Analyzer dokonuje analizy kodu w następujący sposób:

* Parsuje składnię i strukturę AST (Abstract Syntax Tree) dla każdego pliku źródłowego
* Ekstraktuje listę funkcji, metod, klas, endpointów API i procedur SQL wraz z ich sygnaturami, komentarzami i atrybutami bezpieczeństwa
* Tworzy lokalną reprezentację kodu

{ "functions\_found": [ {"name": "authenticate\_user", "params": ["username", "password"], "returns": "token"}, {"name": "reset\_password\_v2", "params": ["email"], "returns": "bool"}, {"name": "get\_user", "params": ["id"], "returns": "UserObject"} ] }

* Wyszukuje odwołania do funkcji, które pojawiają się w dokumentacji - zarówno pod względem nazwy, jak i semantyki (LLM potrafi rozpoznać funkcje o podobnym znaczeniu mimo różnic w nazwie, np. reset\_password vs reset\_user\_password)

##### Etap 4: Porównanie kodu z dokumentacją

LLM łączy obie struktury (z dokumentacji i kodu) i przeprowadza semantyczne porównanie:

* **Zgodność nazw funkcji:** Czy wszystkie funkcje wymienione w dokumentacji rzeczywiście istnieją w kodzie?
* **Zgodność parametrów:** Czy funkcje mają takie same parametry, typy i nazwy jak opisano?
* **Zgodność logiki działania (opis vs implementacja):** Czy funkcja wykonuje operacje zgodne z opisem (np. w dokumentacji: "sprawdza uprawnienia", a w kodzie brak sprawdzania autoryzacji)? Czy kod implementuje wszystkie kroki opisane w dokumentacji?
* **Zgodność komponentów:** Czy struktura klas, modułów i zależności (np. UserService → DatabaseManager) jest zgodna z projektowaną architekturą?
* **Zgodność API:** Czy endpointy REST/GraphQL zgodne są z opisanymi w dokumentacji (metoda, ścieżka, schemat odpowiedzi)?
* **Zgodność bezpieczeństwa:** Czy funkcje, które powinny uwzględniać np. CSRF, XSS czy walidację wejścia, rzeczywiście mają to zaimplementowane w kodzie?

##### Etap 5: Wykrywanie niespójności i luk

Na tym etapie TestGEN identyfikuje różne typy problemów:

|  |  |  |
| --- | --- | --- |
| **Typ niespójności** | **Opis przykładowy** | **Priorytet** |
| 🔴 Brak funkcji opisanej w dokumentacji | Dokumentacja wspomina reset\_password, ale funkcja nie istnieje | Krytyczny |
| 🟠 Funkcja istnieje, ale różni się sygnaturą | reset\_password\_v2(email) zamiast reset\_password(user\_email) | Wysoki |
| 🟡 Brak weryfikacji bezpieczeństwa | Dokumentacja zakłada walidację tokenu, ale kod jej nie zawiera | Wysoki |
| 🟢 Funkcja nadmiarowa | Kod zawiera funkcję, której nie ma w dokumentacji | Średni |
| 🔵 Rozbieżność logiczna | Dokumentacja opisuje zapis do DB, ale kod tylko zwraca dane | Średni |

##### Etap 6: Generowanie raportu zgodności

System generuje raport w formacie HTML / PDF / JSON:

**Raport zgodności kodu z dokumentacją**  
  
**Projekt:** AuthService v2.3  
**Data analizy:** 2025-10-13  
**Analizowane pliki:** 27  
**Język:** Python  
  
**Podsumowanie:**  
✅ 14 funkcji zgodnych z dokumentacją  
❌ 3 funkcje brakujące  
⚠️ 2 funkcje rozbieżne logicznie  
➕ 4 funkcje nadmiarowe

**Przykład niespójności:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Dokumentacja** | **Kod** | **Typ problemu** | **Rekomendacja** |
| reset\_password(user\_email) | brak | Funkcja brakująca | Dodać implementację resetowania hasła |
| authenticate\_user() - wymaga walidacji CSRF | brak walidacji tokena | Luka bezpieczeństwa | Dodać csrf\_protect() |
| get\_user\_by\_id(id) | get\_user(id) | Rozbieżność semantyczna | Ujednolicić nazwy i opis |

System w raporcie niespójności wskazuje rekomendacje:

* Propozycje implementacji brakujących funkcji (np. szkielety kodu w Pythonie/JavaScript)
* Sugestie poprawy dokumentacji, jeśli implementacja jest poprawna, ale dokumentacja nieaktualna
* Zalecenia dot. bezpieczeństwa, np. dodanie walidacji danych wejściowych, obsługi błędów, lub tokenów CSRF
* Automatyczne generowanie testów jednostkowych / integracyjnych sprawdzających zgodność implementacji z opisem dokumentacji

### **4.5.3.** Funkcjonalność podpowiedzi dotyczących błędów

#### **4.5.3.1.** Opis funkcjonalności

Moduł Analizy błędów kodu i podpowiedzi napraw ma pomóc w:

* Szybkim wykrywaniu błędów składniowych, logicznych i semantycznych
* Wykrywaniu niezgodności z dokumentacją techniczną lub standardami projektu
* Sugerowaniu konkretnych poprawek (z gotowym kodem lub fragmentem do wklejenia)
* Edukacyjnym wyjaśnianiu, dlaczego dany błąd powstał i jak go unikać w przyszłości

**Przykładowy przebieg:**

[Wczytanie kodu + dokumentacji]

↓

[Analiza składniowa → błędy formalne]

↓

[Analiza semantyczna → błędy logiczne]

↓

[Porównanie z dokumentacją → brakujące funkcje / niezgodności]

↓

[Generacja raportu i sugestii poprawek]

↓

[Integracja z CI/CD lub IDE → automatyczne komentarze i poprawki]

#### **4.5.3.2.** Szczegółowy przepływ działania

##### 1. Wejście systemu

System TestGEN przyjmuje jako dane wejściowe:

* Kod źródłowy projektu (np. pojedyncze pliki .py, .js, .java, .cpp lub cały folder repozytorium)
* Dokumentację techniczną (opcjonalnie, aby lepiej zrozumieć kontekst biznesowy i standardy kodowania)
* Reguły stylu lub konfigurację lintera, jeśli projekt korzysta z takich (np. .pylintrc, .eslintrc, .editorconfig)

##### 2. Analiza składniowa (syntactic analysis)

**Cel:** Wykrycie błędów formalnych w kodzie, takich jak:

* Brak średnika, nawiasu, cudzysłowu
* Źle zagnieżdżone bloki
* Literówki w nazwach zmiennych/funkcji
* Niezamknięte komentarze
* Błędne formatowanie (np. mieszanie tabów i spacji)

**Mechanizm:**

* TestGEN tworzy drzewo składniowe AST (Abstract Syntax Tree) dla każdego pliku
* Jeśli parser napotka błędy, LLM analizuje komunikat kompilatora/interpretera i fragment kodu, w którym wystąpił błąd

**Silnik TestSpark (LLM):**

* Rozumie kontekst języka
* Wykrywa, gdzie brakuje elementu składniowego (np. średnika)
* Sugeruje poprawkę, podając fragment poprawionego kodu

**Przykład:**

Kod:

function add(a, b) { return a + b } console.log(add(3,4))

Raport:

**Błąd:** Brak średnika po instrukcji 'return a + b'.  
  
**Sugestia naprawy:**

function add(a, b) { return a + b; }

##### 3. Analiza semantyczna (logiczna)

**Cel:** Wykrycie błędów nie w składni, ale w sensie działania kodu, np.:

* Użycie niezdefiniowanej zmiennej
* Błędne przekazanie parametrów do funkcji
* Błędy typów danych (np. dodanie liczby do łańcucha znaków)
* Błędne użycie API, brak obsługi wyjątków

**Mechanizm:**

* System wykonuje statyczną analizę kodu (data flow) - śledzi zależności między zmiennymi, wywołaniami funkcji i zwracanymi wartościami
* LLM ocenia poprawność logiki na podstawie kontekstu:
  + np. "funkcja divide(a, b) nie sprawdza, czy b != 0"
  + "instrukcja await użyta poza funkcją async"
* Na tej podstawie generuje opisowy raport oraz sugestię poprawki

**Przykład:**

Kod:

def divide(a, b): return a / b

Raport TestGEN:

**Błąd logiczny:** Funkcja divide nie sprawdza, czy b != 0.  
  
**Sugestia naprawy:**

def divide(a, b): if b == 0: raise ValueError("Division by zero") return a / b

##### 4. Analiza zgodności z dokumentacją i konwencjami

Jeśli system ma dostęp do dokumentacji (np. pliku API Spec, README, opisów klas):

* LLM porównuje deklaracje funkcji i parametrów w kodzie z tymi opisanymi w dokumentacji
* Wykrywa:
  + Brakujące lub nadmiarowe parametry
  + Niezgodność typów zwracanych danych
  + Brak docstringów lub komentarzy wymaganych przez standard projektu
  + Nieprzestrzeganie naming convention (np. camelCase zamiast snake\_case)
* Na tej podstawie tworzy raport zgodności i rekomendacje

**Przykład:**

Dokumentacja:

Funkcja get\_user powinna zwracać obiekt typu User z polem email

Kod:

def get\_user(id): return id

Raport TestGEN:

**Niezgodność z dokumentacją:**  
- Funkcja `get\_user` powinna zwracać obiekt `User`, ale zwraca wartość typu int.  
  
**Sugestia:**

return User(id=id, email=...)

##### 5. Generowanie raportu błędów i sugestii

Po zakończeniu analizy, TestGEN generuje kompletny raport błędów i poprawek w formatach:

* HTML / PDF (czytelny raport dla dewelopera)
* JSON (dla integracji z CI/CD)
* Markdown (do commit message lub pull requestu)

**Struktura raportu:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Plik** | **Linia** | **Typ błędu** | **Opis** | **Sugestia naprawy** | **Poziom** |
| 001 | main.js | 12 | Syntax | Brak średnika | Dodaj ; po return | Niski |
| 002 | auth.py | 44 | Logic | Brak obsługi dzielenia przez zero | Dodaj if b == 0: | Wysoki |
| 003 | user\_service.py | 17 | Conformance | Zwracany typ niezgodny z dokumentacją | Zwróć obiekt User | Średni |

### **4.5.4.** Funkcjonalność podpowiedzi dotyczących nowych funkcjonalności

#### **4.5.4.1.** Opis funkcjonalności

TestGEN nie tylko testuje oprogramowanie, ale także aktywnie wspiera jego rozwój, identyfikując niewdrożone funkcjonalności, luki w procesach oraz pominięte elementy.

System analizuje dostępne materiały takie jak kod źródłowy, dokumentację techniczną, diagramy UML, oraz przypadki testowe, aby wskazać brakujące funkcje i sugeruje potencjalne ulepszenia aplikacji po analizie dokumentacji.

Celem modułu jest proaktywne wspieranie rozwoju systemu IT poprzez analizę kodu, dokumentacji i istniejących testów, aby:

* Wykrywać brakujące lub niezaimplementowane funkcjonalności
* Sugerować ulepszenia na poziomie produktu, architektury lub logiki aplikacji
* Usprawniać proces projektowania i wdrażania zmian poprzez automatyczne raporty z rekomendacjami

#### **4.5.4.2.** Szczegółowy przepływ działania

##### 1. Wczytanie i przygotowanie danych wejściowych

System przyjmuje zestaw danych źródłowych:

* **Kod źródłowy** (z różnych modułów aplikacji)
* **Dokumentację techniczną i funkcjonalną** (np. pliki Markdown, Confluence export, PDF)
* **Diagramy UML** (np. use case, class, activity, sequence)
* **Zestaw przypadków testowych** (np. w formacie Gherkin, test plans, test suites)

Na tym etapie TestGEN dokonuje:

* Ujednolicenia formatu danych
* Ekstrakcji kluczowych informacji (np. nazw klas, funkcji, endpointów, komponentów UI, opisów funkcji biznesowych)
* Stworzenia semantycznego modelu aplikacji, łączącego kod i dokumentację

##### 2. Analiza zgodności funkcjonalnej (feature mapping)

Silnik TestSpark (LLM) analizuje powiązania między:

* Funkcjami wymienionymi w dokumentacji a rzeczywistymi metodami w kodzie
* Przypadkami użycia (UML) a ich implementacjami
* Planowanymi funkcjonalnościami (np. "powiadomienia push") a ich śladami w repozytorium kodu

System wykrywa:

* Funkcje opisane w dokumentacji, ale brakujące w kodzie
* Funkcje zaimplementowane w kodzie, ale nieopisane w dokumentacji
* Przerwane lub niekompletne przepływy (np. brak implementacji obsługi błędów, walidacji, logowania)

##### 3. Analiza heurystyczna i semantyczna

Na podstawie wzorców projektowych i wiedzy o typowych komponentach aplikacji (np. e-commerce, CRM, ERP, SaaS), TestGEN:

* Wykrywa obszary wymagające rozwoju, np.:
  + Brak systemu powiadomień
  + Brak historii zmian
  + Brak walidacji danych
  + Brak mechanizmów backupu
* Porównuje projekt z bazą wiedzy o najlepszych praktykach (np. OWASP, ISO/IEC 25010, Clean Architecture, UX heuristics)
* Analizuje użycie API i zależności między modułami, szukając niezaadresowanych wymagań biznesowych (np. brak logiki dla rzadkich scenariuszy użytkownika)

##### 4. Generowanie raportu z rekomendacjami

System tworzy czytelny raport, który może zawierać sekcje:

**🔹 "Brakujące funkcjonalności"**

Lista funkcjonalności opisanych w dokumentacji, ale **niewystępujących w kodzie**.

Każda pozycja zawiera:

* Nazwę funkcji lub modułu
* Odniesienie do fragmentu dokumentacji
* Sugerowane miejsce w kodzie, gdzie można ją zaimplementować

**🔹 "Potencjalne ulepszenia"**

Na podstawie analizy kodu i znanych wzorców system proponuje:

* Nowe funkcje, które zwiększyłyby wartość aplikacji (np. logowanie działań użytkownika, cache danych, dark mode)
* Usprawnienia architektury (np. wydzielenie modułu autoryzacji, poprawa bezpieczeństwa API)
* Rekomendacje UX/UI, jeśli analizowano ekrany użytkownika

**🔹 "Uzasadnienie"**

Dla każdej propozycji system generuje opis:

* Dlaczego dana funkcjonalność jest potrzebna
* Jakie problemy rozwiązuje
* Jakie ryzyka eliminuje
* Na jakie standardy lub praktyki się powołuje

##### 5. Możliwość automatycznego wdrożenia rekomendacji

Na życzenie użytkownika TestGEN może:

* Wygenerować szkielet kodu dla nowych funkcjonalności (np. puste metody, endpointy, klasy)
* Dodać nowe przypadki testowe w formacie Gherkin lub unittest do przyszłej implementacji
* Zintegrować raport z systemami zarządzania zadaniami (np. Jira, Azure DevOps), tworząc automatyczne backlogi z rekomendacjami

**Przykładowy raport rekomendacji:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Kategoria** | **Funkcjonalność** | **Status** | **Rekomendacja** | **Uzasadnienie** |
| Dokumentacja vs kod | Eksport raportów CSV | Brak implementacji | Dodać moduł export\_reports() | Funkcja opisana w dokumentacji §4.2, brak w repozytorium |
| Usprawnienie architektury | Obsługa błędów logowania | Częściowo zaimplementowana | Uzupełnić o komunikaty UX | Brak pełnej obsługi błędów w UI |
| Nowa funkcjonalność | Historia zmian w profilu | Niezaimplementowana | Dodać tabelę "user\_audit\_log" | Poprawia zgodność z RODO i kontrolę operacji użytkownika |

### **4.5.5.** Wykorzystane mechanizmy i technologie

#### **4.5.5.1.** Parser wieloźródłowy

Parser umożliwi przetwarzanie danych wejściowych w różnych formatach (np. Markdown, PDF, DOCX, UML). Informacje te zostaną ujednolicone poprzez rozwiązanie semantyczne i technologie NLP, co pozwoli na ich analizę w jednolitym kontekście.

**Przykładowe narzędzia parsowania:**

* Apache Tika - uniwersalny ekstraktor treści
* Pandoc - konwerter formatów dokumentów
* python-docx, PyPDF2 - specjalistyczne parsery
* XMI parsers - dla diagramów UML

#### **4.5.5.2.** Mechanizm detekcji niespójności

Umożliwia zestawienie funkcji opisanych w dokumentacji z rzeczywistą implementacją oraz istniejącymi testami. Wykorzystane zostaną techniki mapowania semantycznego oraz analizy ontologicznej, które pozwolą na precyzyjne wykrycie brakujących lub niedoprecyzowanych funkcjonalności.

#### **4.5.5.3.** Mechanizm rekomendowania nowych funkcjonalności

Automatyczne sugerowanie elementów, które powinny zostać wdrożone, takie jak:

* Brakujące walidacje
* Pominięte przypadki brzegowe
* Niepełne ścieżki alternatywne
* Niedoprecyzowane procesy biznesowe

Sugestie zostaną przedstawione w formie czytelnych raportów oraz artefaktów testowych, które mogą zostać bezpośrednio wykorzystane przez programistów.

## **4.6.** Graficzny interfejs użytkownika (GUI) integracja z systemami płatności

### **4.6.1.** GUI

**C. Rozwój graficznego interfejsu użytkownika (GUI) oraz systemami płatności**

**Termin realizacji: 1 października 2026 - 31 grudnia 2026**

W ramach tego zadania TestGEN osiągnie swoją finalną wersję, obejmującą pełnoprawny graficzny interfejs użytkownika (GUI), zgodny ze standardem WCAG 2.1 a także integrację systemu płatności oraz opracowanie dokumentacji wdrożeniowej.

Szczegóły zostały opisane w rozdziale **10. Rozwój graficznego interfejsu użytkownika (GUI) oraz systemami płatności.**

# **5.** ARCHITEKTURA SYSTEMU

## **5.1.** Architektura ogólna

### **5.1.1.** Model architektoniczny

System TestGEN jest zbudowany w oparciu o architekturę mikroserwisową, która zapewnia:

* **Skalowalność** - możliwość niezależnego skalowania poszczególnych komponentów
* **Niezawodność** - izolacja awarii w ramach pojedynczych serwisów
* **Elastyczność technologiczną** - możliwość wykorzystania różnych technologii dla różnych modułów
* **Łatwość wdrażania** - niezależne deployment poszczególnych serwisów

**REQ-ARCH-001:** System musi być zbudowany w architekturze mikroserwisowej z wykorzystaniem konteneryzacji (Docker).

**REQ-ARCH-002:** System musi wspierać zarówno deployment w chmurze (SaaS) jak i on-premises.

### **5.1.2.** Diagram architektury wysokopoziomowej

**Architektura wysokopoziomowa systemu TestGEN:**

┌─────────────────────────────────────────────────────────────┐

│ WARSTWA PREZENTACJI │

│ ┌──────────────┐ ┌──────────────┐ ┌──────────────┐ │

│ │ Web UI │ │ Mobile App │ │ CLI Tool │ │

│ │ (React) │ │ (Flutter) │ │ (Python) │ │

│ └──────────────┘ └──────────────┘ └──────────────┘ │

└─────────────────────────────────────────────────────────────┘

↓

┌─────────────────────────────────────────────────────────────┐

│ API GATEWAY │

│ (Kong / AWS API Gateway) │

│ Authentication, Rate Limiting, Load Balancing │

└─────────────────────────────────────────────────────────────┘

↓

┌─────────────────────────────────────────────────────────────┐

│ WARSTWA USŁUG BIZNESOWYCH │

│ ┌────────────┐ ┌────────────┐ ┌────────────┐ │

│ │ Code │ │ Doc │ │ UML/GUI │ │

│ │ Analysis │ │ Analysis │ │ Analysis │ │

│ │ Service │ │ Service │ │ Service │ │

│ └────────────┘ └────────────┘ └────────────┘ │

│ ┌────────────┐ ┌────────────┐ ┌────────────┐ │

│ │ Security │ │ Func Gap │ │ Test Gen │ │

│ │ Analysis │ │ Analysis │ │ Service │ │

│ └────────────┘ └────────────┘ └────────────┘ │

└─────────────────────────────────────────────────────────────┘

↓

┌─────────────────────────────────────────────────────────────┐

│ SILNIK TESTSPARK (CORE) │

│ ┌──────────────────────────────────────────────────────┐ │

│ │ AI/ML Engine (LLM, NLP, AST Analysis) │ │

│ │ - Transformers (BERT, GPT) │ │

│ │ - Code Analysis (tree-sitter, AST parsers) │ │

│ │ - Pattern Recognition │ │

│ └──────────────────────────────────────────────────────┘ │

└─────────────────────────────────────────────────────────────┘

↓

┌─────────────────────────────────────────────────────────────┐

│ WARSTWA DANYCH I STORAGE │

│ ┌────────────┐ ┌────────────┐ ┌────────────┐ │

│ │PostgreSQL │ │ MongoDB │ │ Redis │ │

│ │(Metadata) │ │(Documents) │ │(Cache) │ │

│ └────────────┘ └────────────┘ └────────────┘ │

│ ┌────────────┐ ┌────────────┐ │

│ │ S3/Blob │ │ Elastic │ │

│ │ Storage │ │ Search │ │

│ └────────────┘ └────────────┘ │

└─────────────────────────────────────────────────────────────┘

↓

┌─────────────────────────────────────────────────────────────┐

│ INTEGRACJE ZEWNĘTRZNE │

│ ┌────────────┐ ┌────────────┐ ┌────────────┐ │

│ │ GitHub │ │ Jira │ │ CI/CD │ │

│ │ GitLab │ │ TestRail │ │ (Jenkins, │ │

│ │ │ │ │ │ Actions) │ │

│ └────────────┘ └────────────┘ └────────────┘ │

└─────────────────────────────────────────────────────────────┘

### **5.1.3.** Główne warstwy systemu

#### Warstwa prezentacji

Odpowiedzialna za interakcję z użytkownikami poprzez:

* **Web UI** - aplikacja webowa React z Material-UI
* **Mobile App** - aplikacja mobilna Flutter (iOS/Android)
* **CLI Tool** - narzędzie linii poleceń Python dla integracji CI/CD

#### API Gateway

Centralny punkt dostępu do wszystkich serwisów, zapewniający:

* Uwierzytelnianie i autoryzację (OAuth 2.0 / JWT)
* Rate limiting i throttling
* Load balancing
* Request routing
* API versioning

#### Warstwa usług biznesowych

Mikroserwisy realizujące poszczególne funkcjonalności systemu:

* **Code Analysis Service** - analiza kodu źródłowego
* **Documentation Analysis Service** - analiza dokumentacji
* **UML/GUI Analysis Service** - analiza diagramów i interfejsów
* **Security Analysis Service** - analiza bezpieczeństwa
* **Functional Gap Analysis Service** - analiza luk funkcjonalnych
* **Test Generation Service** - generowanie testów

#### Silnik TestSpark (Core)

Centralny komponent AI/ML zawierający:

* Modele transformerów (BERT, GPT) do analizy języka naturalnego
* Parsery AST dla różnych języków programowania
* Mechanizmy pattern recognition
* Taint analysis engine
* Knowledge graphs

#### Warstwa danych

Rozproszona warstwa przechowywania danych:

* **PostgreSQL** - metadata, konfiguracja, użytkownicy
* **MongoDB** - dokumenty, raporty, przypadki testowe
* **Redis** - cache, sesje, kolejki zadań
* **S3/Blob Storage** - pliki źródłowe, artefakty
* **Elasticsearch** - full-text search, logi, analityka

#### Integracje zewnętrzne

Konektory do systemów zewnętrznych:

* Systemy kontroli wersji (GitHub, GitLab, Bitbucket)
* Systemy zarządzania testami (Jira, TestRail, Zephyr)
* Systemy CI/CD (Jenkins, GitHub Actions, Azure DevOps)
* Systemy płatności (Stripe, PayPal)

## **5.2.** Komponenty backendowe

### **5.2.1.** Stos technologiczny backend

|  |  |  |
| --- | --- | --- |
| **Komponent** | **Technologia** | **Uzasadnienie** |
| API Gateway | Kong / AWS API Gateway | Wydajność, skalowalność, bogata funkcjonalność |
| Mikroserwisy | Python (FastAPI) / Node.js (Express) | Wydajność, ekosystem AI/ML, async support |
| Message Queue | RabbitMQ / AWS SQS | Asynchroniczne przetwarzanie, kolejkowanie zadań |
| Task Scheduler | Celery / Apache Airflow | Distributed task execution, scheduling |
| Konteneryzacja | Docker + Kubernetes | Orchestration, scaling, deployment |

### **5.2.2.** Code Analysis Service

#### Odpowiedzialności

* Przyjmowanie kodu źródłowego od użytkowników
* Detekcja języka programowania i frameworka
* Parsowanie kodu do AST
* Ekstrakcja metadanych (klasy, metody, zależności)
* Delegowanie analizy do silnika TestSpark
* Zwracanie wyników analizy

#### API Endpoints

POST /api/v1/code/analyze - Request Body: { "code": "...", "language": "python", "framework": "django" } - Response: { "analysis\_id": "uuid", "status": "processing" } GET /api/v1/code/analysis/{analysis\_id} - Response: { "status": "completed", "results": {...} } POST /api/v1/code/compare - Request Body: { "code\_v1": "...", "code\_v2": "..." } - Response: { "diff": [...], "regression\_tests": [...] }

### **5.2.3.** Documentation Analysis Service

#### Odpowiedzialności

* Przyjmowanie dokumentacji w różnych formatach
* Ekstrakcja tekstu (OCR dla PDF/obrazów)
* Klasyfikacja sekcji dokumentacji
* Analiza NLP i ekstrakcja wymagań
* Porównywanie z kodem/dokumentacją techniczną
* Generowanie przypadków testowych

### **5.2.4.** Test Generation Service

#### Odpowiedzialności

* Agregacja wyników z różnych serwisów analizy
* Generowanie opisowych przypadków testowych
* Generowanie kodu testów w wybranych frameworkach
* Tworzenie struktury projektów testowych
* Generowanie raportów
* Eksport do systemów zewnętrznych

#### API Endpoints

POST /api/v1/tests/generate - Request Body: { "analysis\_ids": ["uuid1", "uuid2"], "framework": "pytest", "test\_types": ["unit", "integration"] } - Response: { "test\_suite\_id": "uuid", "status": "generating" } GET /api/v1/tests/suite/{suite\_id} - Response: { "status": "completed", "test\_cases": [...], "code\_files": [...], "download\_url": "..." } POST /api/v1/tests/export - Request Body: { "suite\_id": "uuid", "format": "jira", "config": {...} } - Response: { "export\_id": "uuid", "status": "success" }

## **5.3.** Komponenty frontendowe

### **5.3.1.** Web Application (React)

#### Architektura frontendowa

* **Framework:** React 18+ z TypeScript
* **State Management:** Redux Toolkit / Zustand
* **UI Library:** Material-UI (MUI) v5+
* **Routing:** React Router v6
* **API Client:** Axios / React Query
* **Build Tool:** Vite

#### Główne moduły UI

|  |  |  |
| --- | --- | --- |
| **Moduł** | **Funkcjonalność** | **Komponenty kluczowe** |
| Dashboard | Przegląd projektów, statystyki | ProjectList, StatsWidget, RecentActivity |
| Code Upload | Wgrywanie kodu, konfiguracja analizy | FileUploader, LanguageSelector, AnalysisConfig |
| Documentation Upload | Wgrywanie dokumentacji | DocUploader, FormatSelector, PreviewPanel |
| Analysis Results | Wyniki analizy, wizualizacje | ResultsViewer, CodeHighlighter, DiffViewer |
| Test Cases | Przeglądanie i edycja testów | TestCaseList, TestCaseEditor, TestRunner |
| Reports | Raporty, wykresy, eksport | ReportGenerator, ChartComponents, ExportDialog |

#### Wymagania dostępności (WCAG 2.1)

**REQ-UI-001:** Wszystkie komponenty UI muszą być zgodne z WCAG 2.1 poziom AA.

Implementacja wymagań WCAG:

* **Kontrast kolorów:** Współczynnik minimum 4.5:1 dla tekstu normalnego, 3:1 dla dużego tekstu
* **Nawigacja klawiaturą:** Wszystkie funkcje dostępne bez myszy, widoczne fokus indicators
* **Atrybuty ARIA:** Wszystkie interaktywne elementy z odpowiednimi rolami, labels i states
* **Responsywność:** Działanie na urządzeniach od 320px do 4K
* **Skalowanie tekstu:** Możliwość powiększenia do 200% bez utraty funkcjonalności
* **Tryb wysokiego kontrastu:** Dedykowany motyw dla użytkowników słabowidzących

### **5.3.2.** Mobile Application (Flutter)

#### Architektura mobilna

* **Framework:** Flutter 3.x+
* **State Management:** Riverpod / BLoC
* **Local Storage:** Hive / SQLite
* **HTTP Client:** Dio
* **Platforms:** iOS 13+, Android 8+

#### Kluczowe funkcjonalności mobilne

* Skanowanie kodu QR do szybkiego dostępu do projektów
* Push notifications dla statusu analiz
* Offline mode z synchronizacją
* Biometryczna autentykacja (Face ID, Touch ID, Fingerprint)
* Udostępnianie raportów przez natywne API

## **5.4.** Integracje zewnętrzne

### **5.4.1.** Integracja z systemami kontroli wersji

#### GitHub Integration

# GitHub App / OAuth App - Webhooks: push, pull\_request, release - API: Repos, Contents, Pull Requests, Comments - Actions: Automatic analysis on PR creation # Example webhook payload handling POST /webhooks/github { "action": "opened", "pull\_request": { "head": { "sha": "..." }, "base": { "sha": "..." } } } → Trigger regression analysis → Post results as PR comment

#### GitLab Integration

# GitLab CI/CD Integration - Webhooks: Push, Merge Request - API: Projects, Repository Files, Merge Requests - CI/CD: Custom pipeline jobs # .gitlab-ci.yml example testgen\_analysis: stage: test script: - testgen-cli analyze --project-id $CI\_PROJECT\_ID - testgen-cli generate-tests --output ./tests artifacts: paths: - tests/

### **5.4.2.** Integracja z systemami zarządzania testami

#### Jira / Xray Integration

# Jira REST API Integration POST /api/v1/integrations/jira/export { "test\_suite\_id": "uuid", "jira\_project\_key": "PROJ", "issue\_type": "Test", "mapping": { "test\_case\_id": "custom\_field\_10001", "priority": "priority", "status": "status" } } # Automatic test case creation - Create Jira issues for each test case - Link to requirements (Epic, Story) - Update test execution results - Sync bidirectionally

#### TestRail Integration

# TestRail API Integration POST /api/v1/integrations/testrail/sync { "test\_suite\_id": "uuid", "testrail\_project\_id": 123, "section\_id": 456 } # Features: - Create test cases in TestRail - Map test case fields - Create test runs - Update test results - Generate reports

### **5.4.3.** Integracja z systemami CI/CD

#### Jenkins Integration

# Jenkins Pipeline Integration pipeline { agent any stages { stage('TestGEN Analysis') { steps { script { // Trigger TestGEN analysis sh 'testgen-cli analyze --code ./src' // Generate tests sh 'testgen-cli generate --framework junit' // Run generated tests sh 'mvn test' } } } stage('Security Scan') { steps { sh 'testgen-cli security-scan --report security-report.json' } } } post { always { // Publish TestGEN reports publishHTML(target: [ reportDir: 'testgen-reports', reportFiles: 'index.html', reportName: 'TestGEN Report' ]) } } }

#### GitHub Actions Integration

# .github/workflows/testgen.yml name: TestGEN Analysis on: pull\_request: branches: [ main, develop ] push: branches: [ main ] jobs: analyze: runs-on: ubuntu-latest steps: - uses: actions/checkout@v3 - name: TestGEN Analysis uses: testgen/analyze-action@v1 with: api-key: ${{ secrets.TESTGEN\_API\_KEY }} project-id: ${{ secrets.TESTGEN\_PROJECT\_ID }} analysis-type: 'full' - name: Generate Tests uses: testgen/generate-tests-action@v1 with: framework: 'pytest' output-path: './tests/generated' - name: Run Generated Tests run: | pip install -r requirements.txt pytest tests/generated/ - name: Upload Results uses: actions/upload-artifact@v3 with: name: testgen-results path: testgen-reports/

#### Azure DevOps Integration

# azure-pipelines.yml trigger: - main - develop pool: vmImage: 'ubuntu-latest' steps: - task: UsePythonVersion@0 inputs: versionSpec: '3.10' - script: | pip install testgen-cli testgen-cli login --api-key $(TESTGEN\_API\_KEY) displayName: 'Setup TestGEN CLI' - script: | testgen-cli analyze --code $(Build.SourcesDirectory) testgen-cli generate --framework pytest displayName: 'Analyze and Generate Tests' - script: | pytest tests/generated/ --junitxml=test-results.xml displayName: 'Run Generated Tests' - task: PublishTestResults@2 inputs: testResultsFormat: 'JUnit' testResultsFiles: '\*\*/test-results.xml' displayName: 'Publish Test Results'

### **5.4.4.** Integracja z systemami płatności

#### Stripe Integration

**REQ-PAYMENT-001:** System musi obsługiwać płatności cykliczne (subskrypcje) poprzez Stripe.

# Stripe Subscription Management POST /api/v1/subscriptions/create { "plan\_id": "basic|professional", "payment\_method": "pm\_card\_xxx", "customer\_email": "user@example.com" } # Webhook handling POST /webhooks/stripe Events: - invoice.payment\_succeeded - invoice.payment\_failed - customer.subscription.deleted - customer.subscription.updated # Subscription tiers Basic Plan: - Price: $49/month - Features: Limited analysis, 100 tests/month - Stripe Price ID: price\_xxx Professional Plan: - Price: $199/month - Features: Unlimited analysis, CI/CD integration - Stripe Price ID: price\_yyy

# **6.** TECHNOLOGIE I NARZĘDZIA

## **6.1.** Stos technologiczny

### **6.1.1.** Backend Technologies

|  |  |  |  |
| --- | --- | --- | --- |
| **Kategoria** | **Technologia** | **Wersja** | **Zastosowanie** |
| Języki programowania | Python Node.js Go | 3.10+ 18 LTS 1.20+ | Mikroserwisy AI/ML API Gateway, real-time Performance-critical services |
| Web Frameworks | FastAPI Express.js | 0.104+ 4.18+ | REST API WebSocket, streaming |
| Bazy danych | PostgreSQL MongoDB Redis | 15+ 6.0+ 7.0+ | Relacyjne dane Dokumenty, logi Cache, kolejki |
| Message Queue | RabbitMQ Apache Kafka | 3.12+ 3.5+ | Task queue Event streaming |
| Search Engine | Elasticsearch | 8.x | Full-text search, analytics |
| Container | Docker Kubernetes | 24.x+ 1.27+ | Konteneryzacja Orkiestracja |

### **6.1.2.** Frontend Technologies

|  |  |  |  |
| --- | --- | --- | --- |
| **Kategoria** | **Technologia** | **Wersja** | **Zastosowanie** |
| Framework | React Flutter | 18+ 3.x | Web UI Mobile app |
| Język | TypeScript Dart | 5.0+ 3.0+ | Type-safe web Mobile development |
| UI Library | Material-UI TailwindCSS | 5.x 3.x | Component library Utility-first CSS |
| State Management | Redux Toolkit Riverpod | 1.9+ 2.x | Web state Flutter state |
| Build Tools | Vite Webpack | 4.x 5.x | Fast dev server Production bundling |

## **6.2.** Biblioteki i frameworki AI/ML

### **6.2.1.** Natural Language Processing (NLP)

|  |  |  |
| --- | --- | --- |
| **Biblioteka** | **Wersja** | **Zastosowanie w TestGEN** |
| Transformers (HuggingFace) | 4.30+ | Modele BERT, GPT do analizy dokumentacji i kodu |
| spaCy | 3.6+ | NER, dependency parsing, text classification |
| NLTK | 3.8+ | Tokenization, POS tagging, text preprocessing |
| LangChain | 0.0.300+ | LLM orchestration, prompt engineering |
| Sentence Transformers | 2.2+ | Semantic similarity, embeddings |

### **6.2.2.** Machine Learning Frameworks

|  |  |  |
| --- | --- | --- |
| **Framework** | **Wersja** | **Zastosowanie** |
| TensorFlow | 2.13+ | Deep learning models, neural networks |
| PyTorch | 2.0+ | Custom model training, fine-tuning |
| scikit-learn | 1.3+ | Classical ML algorithms, preprocessing |
| XGBoost | 2.0+ | Classification, ranking models |

### **6.2.3.** Modele językowe wykorzystywane

**Modele bazowe (foundation models):**

* **GPT-4 / GPT-3.5-turbo (OpenAI)** - generowanie kodu testów, analiza semantyczna
* **CodeLlama (Meta)** - specjalizowany model do analizy i generowania kodu
* **BERT-base / RoBERTa** - klasyfikacja tekstu, NER w dokumentacji
* **CodeBERT (Microsoft)** - rozumienie kodu źródłowego
* **T5 / FLAN-T5** - text-to-text generation, parafrazowanie

**Fine-tuning:** Modele bazowe będą dodatkowo fine-tunowane na domenowych danych (test cases, security vulnerabilities, code patterns) aby zwiększyć dokładność i specjalizację.

## **6.3.** Narzędzia analizy kodu

### **6.3.1.** Parsery AST i analizatory statyczne

|  |  |  |
| --- | --- | --- |
| **Język** | **Narzędzie** | **Funkcjonalność** |
| Multi-language | tree-sitter | Uniwersalny parser dla wielu języków, inkrementalne parsowanie |
| Python | ast, astroid Bandit Pylint | AST parsing Security linting Code quality |
| JavaScript/TS | Esprima, Babel ESLint TypeScript Compiler API | AST parsing Linting Type checking |
| Java | JavaParser SpotBugs PMD | AST parsing Bug detection Code quality |
| C# | Roslyn SonarAnalyzer | AST parsing, semantic analysis Code quality, security |
| C/C++ | Clang cppcheck | AST parsing Static analysis |

### **6.3.2.** Narzędzia SAST (Static Application Security Testing)

|  |  |  |
| --- | --- | --- |
| **Narzędzie** | **Typ** | **Zastosowanie w TestGEN** |
| Semgrep | Open-source | Pattern-based code analysis, custom rules |
| SonarQube | Commercial/Open | Code quality, security vulnerabilities, code smells |
| Bandit | Open-source | Python security issues detection |
| Brakeman | Open-source | Ruby on Rails security scanner |
| FindSecBugs | Open-source | Java security vulnerabilities |

### **6.3.3.** Narzędzia DAST (Dynamic Application Security Testing)

|  |  |  |
| --- | --- | --- |
| **Narzędzie** | **Typ** | **Zastosowanie** |
| OWASP ZAP | Open-source | Web app security scanning, API testing |
| Burp Suite | Commercial | Advanced web security testing, manual testing tools |
| sqlmap | Open-source | Automated SQL injection detection and exploitation |
| Nikto | Open-source | Web server scanning |

### **6.3.4.** Narzędzia SCA (Software Composition Analysis)

|  |  |
| --- | --- |
| **Narzędzie** | **Zastosowanie** |
| Snyk | Dependency vulnerability scanning, license compliance |
| OWASP Dependency-Check | Open-source dependency vulnerability scanner |
| npm audit / pip-audit | Language-specific dependency scanners |
| GitHub Dependabot | Automated dependency updates and security alerts |

## **6.4.** Frameworki testowe wspierane przez TestGEN

### **6.4.1.** Frameworki testowe backend

|  |  |  |  |
| --- | --- | --- | --- |
| **Framework** | **Język** | **Typ testów** | **Wspierane w TestGEN** |
| pytest | Python | Unit, Integration, E2E | ✅ Pełne wsparcie |
| unittest | Python | Unit | ✅ Pełne wsparcie |
| JUnit 5 | Java | Unit, Integration | ✅ Pełne wsparcie |
| TestNG | Java | Unit, Integration, E2E | ✅ Pełne wsparcie |
| NUnit | C# | Unit, Integration | ✅ Pełne wsparcie |
| xUnit | C# | Unit, Integration | ✅ Pełne wsparcie |
| Jest | JavaScript/TS | Unit, Integration | ✅ Pełne wsparcie |
| Mocha + Chai | JavaScript | Unit, Integration | ✅ Pełne wsparcie |
| RSpec | Ruby | Unit, Integration | ⚠️ Ograniczone wsparcie |
| PHPUnit | PHP | Unit, Integration | ⚠️ Ograniczone wsparcie |

### **6.4.2.** Frameworki testowe frontend/E2E

|  |  |  |
| --- | --- | --- |
| **Framework** | **Typ** | **Wspierane w TestGEN** |
| Selenium WebDriver | Web E2E | ✅ Pełne wsparcie (Python, Java, JS) |
| Playwright | Web E2E | ✅ Pełne wsparcie (Python, TS, JS) |
| Cypress | Web E2E | ✅ Pełne wsparcie (JavaScript) |
| Puppeteer | Web E2E | ✅ Pełne wsparcie (JavaScript) |
| TestCafe | Web E2E | ⚠️ Ograniczone wsparcie |
| Appium | Mobile E2E | ✅ Pełne wsparcie (iOS, Android) |
| Espresso | Android E2E | ✅ Pełne wsparcie |
| XCUITest | iOS E2E | ✅ Pełne wsparcie |
| Robot Framework | Keyword-driven | ✅ Pełne wsparcie |

### **6.4.3.** API Testing Frameworks

|  |  |  |
| --- | --- | --- |
| **Framework** | **Język** | **Wspierane w TestGEN** |
| REST Assured | Java | ✅ Pełne wsparcie |
| requests + pytest | Python | ✅ Pełne wsparcie |
| SuperTest | JavaScript/TS | ✅ Pełne wsparcie |
| Postman/Newman | JSON collections | ✅ Eksport kolekcji |
| Karate DSL | Gherkin-style | ⚠️ Ograniczone wsparcie |

# **7.** WYMAGANIA NIEFUNKCJONALNE

## **7.1.** Wydajność

### **7.1.1.** Wymagania czasowe

|  |  |  |
| --- | --- | --- |
| **Operacja** | **Wymaganie** | **Cel** |
| Logowanie użytkownika | < 2s | < 1s |
| Wczytanie dashboard | < 3s | < 2s |
| Upload pliku (do 100MB) | < 30s | < 15s |
| Analiza małego projektu (< 10k LOC) | < 5 min | < 2 min |
| Analiza średniego projektu (10-100k LOC) | < 30 min | < 15 min |
| Analiza dużego projektu (> 100k LOC) | < 2h | < 1h |
| Generowanie testów (100 przypadków) | < 10 min | < 5 min |
| Eksport raportu (PDF/HTML) | < 30s | < 15s |

**REQ-PERF-001:** System musi przetwarzać co najmniej 1000 LOC na sekundę w trybie analizy standardowej.

**REQ-PERF-002:** Czas odpowiedzi API nie może przekraczać 200ms dla 95% requestów (P95).

### **7.1.2.** Wymagania throughput

**REQ-PERF-003:** System musi obsługiwać minimum 100 równoczesnych użytkowników bez degradacji wydajności.

**REQ-PERF-004:** System musi przetwarzać minimum 50 analiz kodu równocześnie.

## **7.2.** Skalowalność

### **7.2.1.** Skalowanie horyzontalne

**REQ-SCALE-001:** Wszystkie mikroserwisy muszą wspierać skalowanie horyzontalne (stateless design).

**REQ-SCALE-002:** System musi automatycznie skalować się w górę/dół w oparciu o obciążenie (auto-scaling).

**Parametry auto-scalingu:**

* CPU utilization > 70% → scale up
* CPU utilization < 30% (przez 10 min) → scale down
* Queue depth > 100 → scale up workers
* Memory utilization > 80% → scale up

### **7.2.2.** Limity systemu

|  |  |  |
| --- | --- | --- |
| **Zasób** | **Limit na Plan Basic** | **Limit na Plan Professional** |
| Użytkownicy | 5 | Unlimited |
| Projekty | 10 | Unlimited |
| Analiz/miesiąc | 100 | Unlimited |
| Wygenerowane testy/miesiąc | 1,000 | Unlimited |
| Storage | 10 GB | 100 GB |
| Rozmiar pojedynczego pliku | 100 MB | 500 MB |
| API Calls/dzień | 1,000 | 10,000 |

## **7.3.** Bezpieczeństwo

### **7.3.1.** Uwierzytelnianie i autoryzacja

**REQ-SEC-001:** System musi wykorzystywać OAuth 2.0 / OpenID Connect do uwierzytelniania użytkowników.

**REQ-SEC-002:** Hasła muszą być hashowane z wykorzystaniem bcrypt lub Argon2 (minimum 12 rounds).

**REQ-SEC-003:** System musi wspierać Multi-Factor Authentication (MFA) poprzez TOTP lub SMS.

**REQ-SEC-004:** Tokeny JWT muszą wygasać po maksymalnie 1 godzinie, z możliwością odświeżenia poprzez refresh token (ważny do 7 dni).

### **7.3.2.** Szyfrowanie danych

**REQ-SEC-005:** Wszystkie połączenia muszą wykorzystywać TLS 1.3 (minimum TLS 1.2).

**REQ-SEC-006:** Dane wrażliwe w bazie danych muszą być szyfrowane w spoczynku (encryption at rest) z wykorzystaniem AES-256.

**REQ-SEC-007:** Kod źródłowy przesłany przez użytkowników musi być szyfrowany podczas przechowywania.

### **7.3.3.** Bezpieczeństwo API

**REQ-SEC-008:** System musi implementować rate limiting: 100 requests/minute dla użytkownika Basic, 1000 requests/minute dla Professional.

**REQ-SEC-009:** API musi być zabezpieczone przed atakami OWASP Top 10 (SQL Injection, XSS, CSRF, etc.).

**REQ-SEC-010:** System musi implementować CORS policy z whitelistą dozwolonych origin.

### **7.3.4.** Audyt i monitoring bezpieczeństwa

**REQ-SEC-011:** Wszystkie operacje wrażliwe muszą być logowane (audit log): logowanie, zmiana hasła, dostęp do danych, zmiany uprawnień.

**REQ-SEC-012:** System musi przechowywać logi audytowe przez minimum 1 rok.

**REQ-SEC-013:** System musi alertować administratorów o podejrzanych aktywnościach (brute force, unauthorized access attempts).

### **7.3.5.** Zgodność z regulacjami

|  |  |  |
| --- | --- | --- |
| **Regulacja** | **Wymagania** | **Status implementacji** |
| RODO/GDPR | * Prawo do usunięcia danych * Prawo do przenoszenia danych * Zgoda na przetwarzanie * Data breach notification (72h) | Wymagane |
| SOC 2 Type II | * Security controls * Availability guarantees * Confidentiality measures | Planowane (rok 2) |
| ISO 27001 | * ISMS implementation * Risk management * Security policies | Planowane (rok 2) |

## **7.4.** Dostępność (WCAG 2.1)

### **7.4.1.** Poziom zgodności

**REQ-A11Y-001:** Interfejs użytkownika musi być zgodny z WCAG 2.1 poziom AA.

### **7.4.2.** Szczegółowe wymagania WCAG

|  |  |  |  |
| --- | --- | --- | --- |
| **Kryterium WCAG** | **Poziom** | **Wymaganie** | **Implementacja** |
| 1.1.1 Treść nietekstowa | A | Wszystkie obrazy muszą mieć alt text | Automatyczna walidacja w CI |
| 1.3.1 Informacje i relacje | A | Semantyczny HTML, ARIA labels | React components z wbudowanym ARIA |
| 1.4.3 Kontrast (minimum) | AA | Kontrast min. 4.5:1 (tekst normalny), 3:1 (duży tekst) | Color system z automatyczną walidacją |
| 1.4.4 Zmiana rozmiaru tekstu | AA | Możliwość powiększenia do 200% | Responsywna typografia (rem units) |
| 1.4.11 Kontrast elementów nietekstowych | AA | Kontrast min. 3:1 dla UI components | Dedykowany color palette |
| 2.1.1 Klawiatura | A | Pełna obsługa klawiatury | Tab navigation, keyboard shortcuts |
| 2.1.2 Brak pułapki klawiatury | A | Możliwość opuszczenia każdego elementu | Focus management |
| 2.4.3 Kolejność fokusa | A | Logiczna kolejność tabulacji | Careful DOM structure |
| 2.4.7 Widoczny fokus | AA | Wyraźny wskaźnik fokusa | Custom focus styles (outline, ring) |
| 3.1.1 Język strony | A | Określenie języka dokumentu | lang="pl" attribute |
| 3.2.3 Spójna nawigacja | AA | Konsystentna struktura nawigacji | Shared layout components |
| 4.1.2 Nazwa, rola, wartość | A | Prawidłowe ARIA attributes | Automated accessibility testing |

### **7.4.3.** Narzędzia do testowania dostępności

**Wykorzystywane narzędzia:**

* **axe-core** - Automatyczne testy dostępności w CI/CD
* **WAVE** - Browser extension do manualnej weryfikacji
* **Lighthouse** - Audyty dostępności w Chrome DevTools
* **NVDA / JAWS** - Testy z czytnikami ekranu
* **Keyboard navigation testing** - Manualne testy nawigacji

### **7.4.4.** Dodatkowe funkcje dostępności

* **Tryb wysokiego kontrastu** - Dedykowany motyw dla użytkowników słabowidzących
* **Powiększanie czcionki** - Kontrolki do zmiany rozmiaru tekstu (Small / Medium / Large / Extra Large)
* **Skip links** - Linki "Przejdź do treści" na początku strony
* **Keyboard shortcuts** - Skróty klawiszowe do najważniejszych funkcji
* **Screen reader announcements** - Live regions (aria-live) dla dynamicznych zmian
* **Reduced motion** - Respektowanie preferencji prefers-reduced-motion

## **7.5.** Niezawodność i dostępność (Availability)

### **7.5.1.** Uptime i SLA

**REQ-AVAIL-001:** System musi zapewniać dostępność na poziomie 99.5% (SLA dla Plan Basic).

**REQ-AVAIL-002:** System musi zapewniać dostępność na poziomie 99.9% (SLA dla Plan Professional).

|  |  |  |
| --- | --- | --- |
| **SLA** | **Dopuszczalny downtime/rok** | **Dopuszczalny downtime/miesiąc** |
| 99.5% | 43.8 godzin | 3.65 godzin |
| 99.9% | 8.76 godzin | 43.8 minut |
| 99.99% (cel długoterminowy) | 52.6 minut | 4.38 minut |

### **7.5.2.** Backup i disaster recovery

**REQ-AVAIL-003:** System musi wykonywać automatyczne backupy baz danych co 24 godziny.

**REQ-AVAIL-004:** Backupy muszą być przechowywane przez minimum 30 dni.

**REQ-AVAIL-005:** RPO (Recovery Point Objective) musi wynosić maksymalnie 1 godzinę.

**REQ-AVAIL-006:** RTO (Recovery Time Objective) musi wynosić maksymalnie 4 godziny.

### **7.5.3.** Monitoring i alerting

**Metryki monitorowane 24/7:**

* **Uptime** - Dostępność wszystkich serwisów
* **Response time** - Czasy odpowiedzi API (P50, P95, P99)
* **Error rate** - Procent błędnych requestów (4xx, 5xx)
* **CPU & Memory utilization** - Zużycie zasobów
* **Database performance** - Query time, connections, locks
* **Queue depth** - Długość kolejek zadań
* **Storage usage** - Zużycie dysku i obiektowego storage

**Alerty (PagerDuty / OpsGenie):**

* Critical: Uptime < 99%, Error rate > 5%, API response P95 > 5s
* Warning: Uptime < 99.5%, Error rate > 2%, CPU > 80%
* Info: Unusual traffic patterns, Queue depth growing

## **7.6.** Utrzymanie i wsparcie

### **7.6.1.** Poziomy wsparcia

|  |  |  |  |
| --- | --- | --- | --- |
| **Plan** | **Kanały wsparcia** | **Czas odpowiedzi** | **SLA rozwiązania** |
| Basic | Email, Dokumentacja | 48h roboczych | 5 dni roboczych |
| Professional | Email, Chat, Phone | 4h roboczych | 24h roboczych (Critical) 3 dni (Normal) |
| Enterprise (on-premises) | Email, Chat, Phone, Dedicated Support | 1h (24/7) | 4h (Critical) 24h (High) 3 dni (Normal) |

### **7.6.2.** Dokumentacja

**REQ-DOC-001:** System musi posiadać kompletną dokumentację API (OpenAPI/Swagger).

**REQ-DOC-002:** System musi posiadać dokumentację użytkownika w języku polskim i angielskim.

**REQ-DOC-003:** System musi posiadać dokumentację wdrożeniową dla instalacji on-premises.

**Komponenty dokumentacji:**

* **User Guide** (300+ stron) - Instrukcja użytkownika end-to-end
* **API Reference** - Kompletna dokumentacja API z przykładami
* **Administrator Guide** - Instalacja, konfiguracja, zarządzanie
* **Developer Guide** - Integracje, webhooks, CLI
* **Architecture Documentation** - Diagramy, decyzje architektoniczne
* **Troubleshooting Guide** - Najczęstsze problemy i rozwiązania
* **Release Notes** - Changelog dla każdej wersji

# **8.** HARMONOGRAM REALIZACJI

## **8.1.** Zadanie A - Silnik TestSpark i generowanie przypadków testowych

### **8.1.1.** Szczegóły zadania

**Termin realizacji:** 2 stycznia 2026 - 31 maja 2026 (5 miesięcy)

**Cel:** Zaprojektowanie i implementacja kluczowego komponentu systemu TestGEN, tj. silnika TestSpark.

Celem zadania jest zaprojektowanie i implementacja kluczowego komponentu systemu TestGEN, tj. silnika TestSpark. Silnik ten, wykorzystując zaawansowane algorytmy sztucznej inteligencji, umożliwi pełną automatyzację analizy dokumentacji technicznej, kodu źródłowego, diagramów UML oraz ekranów GUI, generując na tej podstawie przypadki testowe oraz odpowiadający im kod testów. Projekt ma na celu znaczące ograniczenie pracy manualnej w procesie testowania i zwiększenie efektywności oraz pokrycia testowego. Podjęte zostaną następujące kluczowe czynności w ramach realizacji zadania.

1. Prace rozpoczną się od analizy danych wejściowych – kodu źródłowego, dokumentacji użytkownika, diagramów UML oraz zrzutów ekranów GUI. Dane te zostaną sklasyfikowane i zamodelowane zgodnie z ich strukturą oraz przeznaczeniem.

Celem jest opracowanie ujednoliconej reprezentacji danych, umożliwiającej ich dalsze przetwarzanie przez systemy AI.

2. Kolejnym krokiem będzie zaprojektowanie i wdrożenie komponentów ekstrakcji informacji. W tym celu zastosowane zostaną metody NLP (np. NER czy modeli typu transformer), pozwalające na identyfikację kluczowych pojęć, funkcji, warunków brzegowych i wyników opisanych w dokumentacji. W przypadku kodu źródłowego wykorzystane zostaną techniki statycznej analizy, umożliwiające wyodrębnienie struktur logicznych, punktów wejścia i zależności między komponentami. Diagramy UML zostaną przetworzone w celu wydobycia przepływów sterowania i logiki biznesowej, natomiast analiza GUI pozwoli na określenie interaktywnych elementów interfejsu i możliwych ścieżek działania użytkownika.

3. Na podstawie przetworzonych danych, opracowane zostaną mechanizmy automatycznego generowania różnorodnych przypadków testowych. Rozwiązanie będzie uwzględniać zidentyfikowane przepływy biznesowe, zależności między funkcjonalnościami, warunki brzegowe oraz potencjalne ścieżki wykonania. Uwzględnione zostaną scenariusze testowe – np. pozytywne, negatywne i brzegowe – jak również zidentyfikowane przepływy biznesowe, zależności i warunki wykonania. Wygenerowane przypadki testowe będą posiadały ustrukturyzowaną formę, zawierającą kroki, dane wejściowe, oczekiwane rezultaty i warunki wstępne.

4. Ostatni etap obejmuje stworzenie modułu generowania kodu testów. Na podstawie opisów przypadków testowych oraz danych źródłowych, silnik będzie mógł dla wybranych przypadków utworzyć kod dla frameworku testowego (np. JUnit, NUnit, pytest). W tym celu wykorzystane zostaną techniki szablonowe oraz inteligentne rozwiązania mapujące scenariusze testowe na konkretne instrukcje kodu. Generowane testy będą gotowe do uruchomienia i dostosowane do specyfiki interfejsów systemu (API, GUI).

Rezultatem zadania będzie w pełni funkcjonalny silnik TestSpark automatycznie generujący przypadki testowe i kod. Ocena obejmie jego zdolność do wygenerowania co najmniej 10 unikalnych przypadków testowych na podstawie dostarczonych danych (dokumentacja, kod, UML, GUI). Każdy przypadek testowy zawiera jasno określone kroki, dane wejściowe oraz oczekiwane rezultaty. Wygeneruje dla wybranych przyp. szkielet kodu testowego w wybranym frameworku (np. JUnit). Rezultaty zostaną ocenione manualnie pod kątem poprawności i jakości wygenerowanych artefaktów. Sprawdzona będzie ich adekwatność względem danych wejściowych. Dodatkowym wynikiem będzie dokumentacja projektowa opisująca architekturę silnika, zastosowane algorytmy, proces przetwarzania danych oraz wyniki testów prototypu

### **8.1.2.** Kluczowe milestone'y

|  |  |  |
| --- | --- | --- |
| **Milestone** | **Data** | **Deliverables** |
| M1.1: Projekt architektury | 31.01.2026 | * Diagram architektury silnika TestSpark * Specyfikacja API wewnętrznych * Wybór modeli AI/ML |
| M1.2: Moduł analizy kodu | 28.02.2026 | * Parser AST dla 5+ języków * Ekstraktor metadanych * Prototyp analizy |
| M1.3: Moduł analizy dokumentacji | 31.03.2026 | * NLP pipeline * Ekstrakcja wymagań * Klasyfikator dokumentów |
| M1.4: Moduł analizy UML/GUI | 30.04.2026 | * Parser UML (XMI, PlantUML) * Analiza wizualna GUI (OCR + CNN) * Rekonstrukcja user flows |
| M1.5: Generator testów | 31.05.2026 | * Generowanie opisowych test cases * Generowanie kodu (pytest, JUnit, Selenium) * Integracja wszystkich modułów * Dokumentacja techniczna |

### **8.1.3.** Kryteria akceptacji Zadania A

**Kryteria sukcesu:**

* Silnik TestSpark generuje minimum 10 unikalnych przypadków testowych na podstawie dostarczonych danych
* Każdy przypadek testowy zawiera jasno określone kroki, dane wejściowe oraz oczekiwane rezultaty
* System generuje szkielet kodu testowego w wybranym frameworku (np. JUnit, pytest)
* Rezultaty są weryfikowane manualnie pod kątem poprawności i jakości
* Sprawdzona adekwatność względem danych wejściowych
* Dokumentacja projektowa opisująca architekturę, algorytmy i wyniki testów

## **8.2.** Zadanie B - Rozbudowa pod kątem audytów bezpieczeństwa

### **8.2.1.** Szczegóły zadania

**Termin realizacji:** 1 czerwca 2026 - 30 września 2026 (4 miesiące)

**Cel:** Rozszerzenie silnika TestSpark o mechanizmy analizy zgodności kodu i dokumentacji, zaawansowane testowanie bezpieczeństwa oraz możliwość badania istniejącego oprogramowania pod kątem podatności.

W ramach zadania 2 moduł TestSpark, kluczowy komponent systemu TestGEN, zostanie rozszerzony o innowacyjne mechanizmy analizy zgodności kodu i dokumentacji, zaawansowane testowanie bezpieczeństwa, a także możliwość badania istniejącego oprogramowania pod kątem podatności na zagrożenia. Dzięki temu TestGEN stanie się nie tylko narzędziem wspierającym projektowanie nowych systemów, ale również aktywnym audytorem bezpieczeństwa wdrożonych aplikacji. Jednym z kluczowych aspektów rozbudowy TestGEN będzie jego zdolność do ciągłego samodoskonalenia. System będzie mógł regularnie otrzymywać nowo odkryte podatności, analizując dane o cyberzagrożeniach i ucząc się nowych metod wykrywania luk. Dzięki temu będzie identyfikował zagrożenia, które wcześniej nie były znane.

1. Zadanie 1: Mechanizm analizy zgodności i wykrywania luk funkcjonalnych

TestGEN będzie automatycznie porównywał kod źródłowy, dokumentację techniczną, podręczniki użytkownika oraz diagramy UML, wykorzystując algorytmy AI do analizy spójności projektu. Zaawansowane techniki przetwarzania języka naturalnego pozwolą na ekstrakcję wymagań i ich zestawienie z implementacją aplikacji. System wykryje brakujące walidacje, niezaimplementowane operacje oraz niespójności między UML a dokumentacją, generując raport z zaleceniami. TestGEN nie tylko oznaczy te niezgodności, ale także zasugeruje brakujące funkcjonalności wynikające z dokumentacji systemu. Dzięki temu testowanie stanie się integralną częścią procesu programistycznego, eliminując błędy przed wdrożeniem, oszczędzając czas i zasoby.

2. Zadanie 2: Testowanie bezpieczeństwa i zdolność do samodoskonalenia

TestGEN przetestuje tworzone oprogramowanie, jak i pozwoli na analizę istniejących aplikacji. Po wgraniu kodu wdrożonego systemu przeprowadzi audyt bezpieczeństwa, wykrywając podatności i zalecając poprawki. TestGEN będzie mógł regularnie otrzymywać nowe dane o zagrożeniach, które zostaną przeanalizowane i wykorzystane do aktualizacji metod testowania. Metody analizy ryzyka pozwolą na identyfikację podatności, których nie wykrył wcześniej, stale podnosząc poziom ochrony systemów IT.

3. Zadanie 3: Technologie i sposób realizacji

W rozwoju modułu TestGEN zostaną zastosowane frameworki NLP, systemy analizy kodu, UML oraz mechanizmy analizy bezpieczeństwa, a także modele AI oraz algorytmy wykrywania luk.

### **8.2.2.** Kluczowe milestone'y

|  |  |  |
| --- | --- | --- |
| **Milestone** | **Data** | **Deliverables** |
| M2.1: Moduł analizy zgodności | 30.06.2026 | * Porównywanie kodu z dokumentacją * Detekcja luk funkcjonalnych * Raport rozbieżności |
| M2.2: Moduł bezpieczeństwa SAST | 31.07.2026 | * Detekcja SQL Injection, XSS, CSRF * Taint analysis * Pattern matching rules |
| M2.3: Moduł bezpieczeństwa DAST | 31.08.2026 | * Integracja z OWASP ZAP * Automatyczne testy penetracyjne * PoC generation |
| M2.4: System aktualizacji | 30.09.2026 | * Pipeline aktualizacji CVE * Auto-update detection rules * Dokumentacja modułów bezpieczeństwa |

### **8.2.3.** Kryteria akceptacji Zadania B

**Kryteria sukcesu:**

* System wykrywa minimum 5 typów podatności (SQLi, XSS, CSRF, AuthZ, Hardcoded Secrets)
* Dla każdej podatności generuje konkretną rekomendację naprawczą z przykładem kodu
* System identyfikuje rozbieżności między kodem a dokumentacją z accuracy > 80%
* Pipeline aktualizacji CVE działa automatycznie i dodaje nowe reguły co miesiąc
* Dokumentacja modułów bezpieczeństwa (100+ stron)

## **8.3.** Zadanie C - GUI, płatności i finalizacja

### **8.3.1.** Szczegóły zadania

**Termin realizacji:** 1 października 2026 - 31 grudnia 2026 (3 miesiące)

**Cel:** Stworzenie finalnej wersji systemu TestGEN z graficznym interfejsem użytkownika zgodnym z WCAG 2.1 oraz integracją systemu płatności.

W ramach tego zadania TestGEN osiągnie swoją finalną wersję, obejmującą pełnoprawny graficzny interfejs użytkownika (GUI), zgodny ze standardem WCAG 2.1 a także integrację systemu płatności oraz opracowanie dokumentacji wdrożeniowej.

Zadanie 1 – Opracowanie GUI zgodnego z WCAG 2.1 i integracja systemu płatności

Na tym etapie zostanie stworzony intuicyjny, dostępny i ergonomiczny interfejs graficzny TestGEN, który będzie spełniał wymagania WCAG 2.1 (Web Content Accessibility Guidelines), zapewniając wygodne korzystanie z systemu zarówno dla osób bez doświadczenia w programowaniu, jak i dla użytkowników z różnymi potrzebami dostępności.

Interfejs będzie zawierał kluczowe elementy dostępności zgodnie z WCAG 2.1, w tym:

* Kontrastowanie czcionki i elementów graficznych – zapewniające czytelność dla
* osób słabowidzących.
* Możliwość powiększania czcionki – użytkownicy będą mogli dostosować rozmiar
* tekstu bez utraty czytelności interfejsu.
* Obsługa czytników ekranu – wszystkie kluczowe elementy będą oznaczone alternatywnymi
* opisami tekstowymi.
* Pełna nawigacja klawiaturą – interfejs będzie w pełni funkcjonalny bez konieczności
* używania myszy.
* Eliminacja migających i dynamicznych elementów – zgodnie z wytycznymi WCAG,
* aby uniknąć problemów z dostępnością dla osób z epilepsją czy nadwrażliwością
* na ruch.
* Zgodność z technologiami asystującymi – aplikacja będzie współpracować z systemami
* wspomagającymi osoby z niepełnosprawnościami.

Dodatkowo zostanie wdrożony system płatności, który umożliwi użytkownikom zakup abonamentu na korzystanie z TestGEN i będzie wspierał różne metody płatności.

Zadanie 2 – Wytworzenie dokumentacji wdrożeniowej

Dokumentacja wdrożeniowa będzie zawierać:

* Opis architektury systemu – szczegółowe informacje o strukturze modułów, przepływie danych i integracji.
* Instrukcję użytkownika – przewodnik po GUI, konfiguracji i funkcjonalnościach TestGEN.
* Specyfikację techniczną – dokumentację API, formaty danych, wymagania systemowe i sposoby integracji.
* Dokumentację modułów AI – opis działania algorytmów sztucznej inteligencji.
* Procedury testowania – opis scenariuszy testowych i metod weryfikacji zgodności z WCAG 2.1.
* Zasady wdrażania i skalowania – instrukcje instalacji, konfiguracji oraz aktualizacji systemu.
* Opis modułu identyfikacji luk funkcjonalnych – mechanizmy analizy brakujących funkcji.
* Instrukcję integracji z systemami płatności – zasady wdrożenia mechanizmu zakupowego.

Plan dalszego rozwoju i aktualizacji – strategię wdrażania nowych wersji TestGEN.

Rezultatem zadania będzie finalna wersja systemu TestGEN z graficznym interfejsem użytkownika (GUI) zgodnym z WCAG 2.1, umożliwiającym intuicyjną obsługę dla wszystkich użytkowników. System zostanie zintegrowany z mechanizmem płatności, pozwalającym na zakup abonamentu. Dokumentacja wdrożeniowa zawierać będzie opis architektury, specyfikację techniczną, procedury testowania oraz zasady wdrażania i skalowania systemu. Jej jakość zostanie zweryfikowana pod kątem spójności i zgodności ze standardami technicznymi.

### **8.3.2.** Kluczowe milestone'y

|  |  |  |
| --- | --- | --- |
| **Milestone** | **Data** | **Deliverables** |
| M3.1: GUI - Core components | 31.10.2026 | * Dashboard, Project management * File upload & analysis config * WCAG 2.1 compliant components |
| M3.2: GUI - Results & Reports | 15.11.2026 | * Results viewer * Test case management * Report generation & export |
| M3.3: Payment system | 30.11.2026 | * Stripe integration * Subscription management * Billing portal |
| M3.4: Dokumentacja końcowa | 31.12.2026 | * User Guide (300-400 stron) * API Documentation * Admin Guide * Deployment Guide (on-premises) |

### **8.3.3.** Kryteria akceptacji Zadania C

**Kryteria sukcesu:**

* GUI spełnia wszystkie wymagania WCAG 2.1 poziom AA (automated tests + manual audit)
* System płatności obsługuje subskrypcje Basic i Professional
* Dokumentacja użytkownika zawiera 300-400 stron
* System przechodzi testy E2E dla wszystkich głównych scenariuszy
* Performance testing: spełnia wszystkie wymagania z rozdziału 7.1
* Security audit: brak krytycznych podatności

## **8.4.** Podsumowanie harmonogramu

**Timeline projektu TestGEN (2026):**

Q1 2026 Q2 2026 Q3 2026 Q4 2026

│ │ │ │

├──────────────────────┼──────────────────────┼──────────────────────┼────────►

│ │ │ │

│ Zadanie A: │ │ │

│ Silnik TestSpark │ │ │

│ Jan ──────────────────────────────► Maj │ │

│ │ │ │

│ │ Zadanie B: │ │

│ │ Bezpieczeństwo │ │

│ │ Jun ──────────────────────────► Sep │

│ │ │ │

│ │ │ Zadanie C: │

│ │ │ GUI & Finalizacja │

│ │ │ Oct ──────────────────────► Dec

│ │ │ │

└──────────────────────┴──────────────────────┴──────────────────────┴────────►

**Całkowity czas realizacji:** 12 miesięcy (styczeń 2026 - grudzień 2026)

Specyfikacja Techniczna Systemu TestGEN - Wersja 1.0 - Część 2

© 2025 - Dokument poufny - Przeznaczony wyłącznie dla zespołu deweloperskiego

# **9.** DOKUMENTACJA TECHNICZNA

## **9.1.** Dokumentacja API

### **9.1.1.** Standardy dokumentacji API

**REQ-API-DOC-001:** Dokumentacja API musi być zgodna ze standardem OpenAPI 3.0 (Swagger).

**REQ-API-DOC-002:** Każdy endpoint API musi być udokumentowany z przykładami request/response.

Dokumentacja API będzie zawierać:

* **Opis endpointu** - szczegółowy opis funkcjonalności
* **Parametry** - wszystkie parametry (path, query, body) z typami i opisami
* **Request body schema** - schemat JSON z przykładami
* **Response schemas** - wszystkie możliwe kody odpowiedzi z przykładami
* **Error codes** - lista możliwych błędów i ich znaczenie
* **Authentication** - wymagania dotyczące autentykacji
* **Rate limiting** - limity wywołań dla danego endpointu
* **Przykłady użycia** - kompletne przykłady w różnych językach (curl, Python, JavaScript)

### **9.1.2.** Przykładowa dokumentacja endpointu

**POST** /api/v1/code/analyze

**Opis:** Inicjuje analizę kodu źródłowego. System wykryje język programowania, framework i przeprowadzi pełną analizę statyczną generując przypadki testowe.

**Autentykacja:** Bearer Token (JWT)

**Rate Limit:** 10 requests/minute (Basic), 100 requests/minute (Professional)

#### Request Body

{ "code": "string (required)", "language": "string (optional)", "framework": "string (optional)", "analysis\_options": { "include\_security": "boolean (default: true)", "include\_performance": "boolean (default: false)", "test\_types": ["unit", "integration", "e2e"] }, "metadata": { "project\_name": "string", "version": "string" } }

#### Response 202 Accepted

{ "analysis\_id": "550e8400-e29b-41d4-a716-446655440000", "status": "processing", "created\_at": "2026-10-15T10:30:00Z", "estimated\_completion": "2026-10-15T10:35:00Z", "status\_url": "/api/v1/code/analysis/550e8400-e29b-41d4-a716-446655440000" }

#### Response 400 Bad Request

{ "error": "INVALID\_REQUEST", "message": "Code parameter is required", "details": { "field": "code", "constraint": "not\_empty" } }

#### Response 401 Unauthorized

{ "error": "UNAUTHORIZED", "message": "Invalid or expired token" }

#### Response 429 Too Many Requests

{ "error": "RATE\_LIMIT\_EXCEEDED", "message": "Rate limit exceeded. Try again in 30 seconds", "retry\_after": 30 }

#### Przykłady użycia

**cURL:**

curl -X POST https://api.testgen.io/v1/code/analyze \ -H "Authorization: Bearer YOUR\_API\_KEY" \ -H "Content-Type: application/json" \ -d '{ "code": "def add(a, b):\n return a + b", "language": "python", "analysis\_options": { "include\_security": true, "test\_types": ["unit"] } }'

**Python:**

import requests url = "https://api.testgen.io/v1/code/analyze" headers = { "Authorization": "Bearer YOUR\_API\_KEY", "Content-Type": "application/json" } data = { "code": "def add(a, b):\n return a + b", "language": "python", "analysis\_options": { "include\_security": True, "test\_types": ["unit"] } } response = requests.post(url, json=data, headers=headers) result = response.json() print(f"Analysis ID: {result['analysis\_id']}")

**JavaScript:**

const axios = require('axios'); const analyzeCode = async () => { try { const response = await axios.post( 'https://api.testgen.io/v1/code/analyze', { code: 'def add(a, b):\n return a + b', language: 'python', analysis\_options: { include\_security: true, test\_types: ['unit'] } }, { headers: { 'Authorization': 'Bearer YOUR\_API\_KEY', 'Content-Type': 'application/json' } } ); console.log('Analysis ID:', response.data.analysis\_id); } catch (error) { console.error('Error:', error.response.data); } }; analyzeCode();

### **9.1.3.** Główne grupy endpointów API

|  |  |  |
| --- | --- | --- |
| **Grupa** | **Endpointy** | **Opis** |
| Authentication | /auth/login /auth/logout /auth/refresh /auth/register | Autentykacja i zarządzanie sesją |
| Projects | /projects /projects/{id} /projects/{id}/settings | Zarządzanie projektami |
| Code Analysis | /code/analyze /code/analysis/{id} /code/compare | Analiza kodu źródłowego |
| Documentation | /docs/analyze /docs/analysis/{id} /docs/compare | Analiza dokumentacji |
| UML/GUI | /uml/analyze /gui/analyze /gui/compare | Analiza diagramów i GUI |
| Security | /security/scan /security/vulnerabilities/{id} /security/recommendations/{id} | Skanowanie bezpieczeństwa |
| Test Generation | /tests/generate /tests/suite/{id} /tests/export | Generowanie testów |
| Reports | /reports/generate /reports/{id} /reports/{id}/download | Raporty i eksport |
| Integrations | /integrations/github /integrations/jira /integrations/testrail | Integracje zewnętrzne |
| Subscriptions | /subscriptions /subscriptions/upgrade /billing/invoices | Zarządzanie subskrypcjami |

### **9.1.4.** Webhooks

System TestGEN wspiera webhooks do asynchronicznej komunikacji o zdarzeniach:

|  |  |  |
| --- | --- | --- |
| **Event Type** | **Opis** | **Payload** |
| analysis.completed | Analiza zakończona | analysis\_id, status, results\_url |
| analysis.failed | Analiza nieudana | analysis\_id, error\_code, error\_message |
| tests.generated | Testy wygenerowane | suite\_id, test\_count, download\_url |
| vulnerability.found | Wykryto podatność | vulnerability\_id, severity, type, location |
| subscription.updated | Subskrypcja zaktualizowana | subscription\_id, plan, status |

#### Przykład webhook payload:

{ "event": "analysis.completed", "timestamp": "2026-10-15T10:35:00Z", "data": { "analysis\_id": "550e8400-e29b-41d4-a716-446655440000", "project\_id": "proj\_123", "status": "completed", "results\_url": "https://api.testgen.io/v1/code/analysis/550e8400-e29b-41d4-a716-446655440000", "summary": { "test\_cases\_generated": 45, "vulnerabilities\_found": 3, "code\_coverage\_estimate": "78%" } } }

## **9.2.** Dokumentacja użytkownika

### **9.2.1.** Struktura dokumentacji użytkownika

**REQ-USER-DOC-001:** Dokumentacja użytkownika musi zawierać 300-400 stron i być dostępna w językach: polski, angielski.

**Zakres dokumentacji użytkownika:**

#### Część I: Wprowadzenie (30-40 stron)

* **Rozdział 1: O systemie TestGEN**
  + Czym jest TestGEN
  + Kluczowe funkcjonalności
  + Korzyści dla użytkowników
  + Wymagania systemowe
* **Rozdział 2: Szybki start**
  + Rejestracja i pierwsze logowanie
  + Konfiguracja pierwszego projektu
  + Pierwsza analiza kodu
  + Generowanie pierwszych testów

#### Część II: Praca z systemem (120-150 stron)

* **Rozdział 3: Zarządzanie projektami**
  + Tworzenie i konfiguracja projektów
  + Zarządzanie członkami zespołu
  + Ustawienia projektu
  + Archiwizacja i usuwanie projektów
* **Rozdział 4: Analiza kodu źródłowego**
  + Wgrywanie kodu (plik, repozytorium, ZIP)
  + Konfiguracja analizy
  + Interpretacja wyników
  + Testy regresji kodu
  + Przykłady dla różnych języków programowania
* **Rozdział 5: Analiza dokumentacji**
  + Wspierane formaty dokumentacji
  + Wgrywanie dokumentacji
  + Porównywanie dokumentacji z kodem
  + Generowanie testów z dokumentacji
* **Rozdział 6: Analiza UML i GUI**
  + Wgrywanie diagramów UML
  + Analiza interfejsów użytkownika
  + Generowanie testów UI
  + Testy regresji GUI
* **Rozdział 7: Analiza bezpieczeństwa**
  + Skanowanie podatności
  + Interpretacja wyników bezpieczeństwa
  + Rekomendacje naprawcze
  + Generowanie testów bezpieczeństwa
* **Rozdział 8: Generowanie testów**
  + Wybór frameworka testowego
  + Konfiguracja generowania
  + Przeglądanie i edycja przypadków testowych
  + Eksport testów
* **Rozdział 9: Raporty i eksport**
  + Generowanie raportów
  + Dostosowywanie raportów
  + Eksport do różnych formatów
  + Integracja z systemami zewnętrznymi

#### Część III: Zaawansowane funkcje (80-100 stron)

* **Rozdział 10: Integracje**
  + Integracja z GitHub/GitLab
  + Integracja z Jira/TestRail
  + Integracja z CI/CD (Jenkins, GitHub Actions, Azure DevOps)
  + Webhooks i API
* **Rozdział 11: Praca z CLI**
  + Instalacja testgen-cli
  + Podstawowe komendy
  + Automatyzacja z CLI
  + Przykłady użycia w skryptach
* **Rozdział 12: Dostępność i personalizacja**
  + Ustawienia dostępności (WCAG)
  + Personalizacja interfejsu
  + Skróty klawiszowe
  + Motywy kolorystyczne

#### Część IV: Administracja i zarządzanie (40-60 stron)

* **Rozdział 13: Zarządzanie kontem i subskrypcją**
  + Ustawienia konta
  + Zmiana planu subskrypcji
  + Zarządzanie płatnościami
  + Faktury i historia płatności
* **Rozdział 14: Bezpieczeństwo i prywatność**
  + Zarządzanie hasłem i MFA
  + Tokeny API
  + Logi audytowe
  + RODO i prywatność danych

#### Część V: Rozwiązywanie problemów (30-50 stron)

* **Rozdział 15: FAQ**
  + Najczęściej zadawane pytania
  + Typowe problemy i rozwiązania
* **Rozdział 16: Troubleshooting**
  + Problemy z analizą kodu
  + Problemy z generowaniem testów
  + Problemy z integracjami
  + Problemy z wydajnością
* **Rozdział 17: Wsparcie techniczne**
  + Jak uzyskać pomoc
  + Kanały wsparcia
  + Zgłaszanie błędów
  + Propozycje nowych funkcji

### **9.2.2.** Przykładowa strona z dokumentacji użytkownika

#### Rozdział 4.2: Wgrywanie kodu źródłowego

**System TestGEN oferuje trzy sposoby wgrywania kodu do analizy:**

**Metoda 1: Wgrywanie pojedynczych plików**

1. Na stronie głównej projektu kliknij przycisk **"Nowa analiza"**
2. Wybierz opcję **"Wgraj pliki"**
3. Przeciągnij pliki do obszaru przesyłania lub kliknij **"Wybierz pliki"**
4. Wybierz język programowania (opcjonalnie - system wykryje automatycznie)
5. Kliknij **"Rozpocznij analizę"**

**Wskazówka:** Możesz wgrać do 100 plików jednocześnie. Maksymalny rozmiar pojedynczego pliku to 100 MB (Plan Basic) lub 500 MB (Plan Professional).

**Metoda 2: Wgrywanie archiwum ZIP**

1. Spakuj swój projekt do pliku ZIP
2. Na stronie głównej projektu kliknij **"Nowa analiza"**
3. Wybierz opcję **"Wgraj archiwum ZIP"**
4. Wybierz plik ZIP z dysku
5. System automatycznie rozpozna strukturę projektu
6. Kliknij **"Rozpocznij analizę"**

**Metoda 3: Integracja z repozytorium Git**

1. W ustawieniach projektu przejdź do zakładki **"Integracje"**
2. Wybierz **"Połącz z GitHub"** lub **"Połącz z GitLab"**
3. Autoryzuj dostęp do repozytorium
4. Wybierz repozytorium z listy
5. Wybierz branch do analizy
6. Włącz automatyczną analizę przy każdym push (opcjonalnie)

**Najlepsza praktyka:** Dla projektów rozwijanych aktywnie zalecamy integrację z Git i włączenie automatycznej analizy. System będzie automatycznie analizował każdą zmianę i generował testy regresji.

## **9.3.** Dokumentacja wdrożeniowa

### **9.3.1.** Dokumentacja instalacji on-premises

**REQ-DEPLOY-DOC-001:** Dokumentacja wdrożeniowa musi zawierać szczegółowe instrukcje instalacji on-premises.

**Struktura dokumentacji wdrożeniowej (150-200 stron):**

#### Część I: Przygotowanie środowiska

* **Wymagania systemowe**
  + Wymagania sprzętowe (CPU, RAM, Disk)
  + Systemy operacyjne (Linux, Windows Server)
  + Wymagane oprogramowanie (Docker, Kubernetes, PostgreSQL, etc.)
  + Wymagania sieciowe (porty, firewall)
* **Architektura deployment**
  + Single-node vs. Multi-node
  + High Availability setup
  + Load balancing
  + Disaster recovery

#### Część II: Instalacja krok po kroku

* **Instalacja Docker i Kubernetes**
* **Konfiguracja baz danych**
  + PostgreSQL setup
  + MongoDB setup
  + Redis setup
  + Inicjalizacja schematów
* **Deployment aplikacji**
  + Pobieranie obrazów Docker
  + Konfiguracja zmiennych środowiskowych
  + Deployment mikroserwisów
  + Konfiguracja API Gateway
* **Konfiguracja SSL/TLS**
  + Generowanie certyfikatów
  + Konfiguracja HTTPS
  + Reverse proxy setup

#### Część III: Konfiguracja i dostrajanie

* **Konfiguracja aplikacji**
* **Integracja z systemami autentykacji (LDAP, Active Directory, OAuth)**
* **Konfiguracja backupu**
* **Monitoring i logging**
* **Performance tuning**

#### Część IV: Utrzymanie i administracja

* **Procedury aktualizacji**
* **Backup i restore**
* **Skalowanie systemu**
* **Troubleshooting**

### **9.3.2.** Przykładowa instrukcja instalacji

#### Instalacja TestGEN on-premises - Quick Start

**Wymagania minimalne:**

* OS: Ubuntu 22.04 LTS / CentOS 8 / Windows Server 2019+
* CPU: 8 cores
* RAM: 32 GB
* Disk: 500 GB SSD
* Docker 24.x+
* Kubernetes 1.27+ (opcjonalnie)

**Krok 1: Przygotowanie środowiska**

# Aktualizacja systemu sudo apt update && sudo apt upgrade -y # Instalacja Docker curl -fsSL https://get.docker.com -o get-docker.sh sudo sh get-docker.sh # Instalacja Docker Compose sudo apt install docker-compose-plugin # Weryfikacja instalacji docker --version docker compose version

**Krok 2: Pobranie konfiguracji TestGEN**

# Pobranie repozytorium instalacyjnego git clone https://github.com/testgen/on-premises.git cd on-premises # Skopiowanie przykładowej konfiguracji cp .env.example .env # Edycja konfiguracji nano .env

**Krok 3: Konfiguracja zmiennych środowiskowych**

# .env file POSTGRES\_HOST=localhost POSTGRES\_PORT=5432 POSTGRES\_DB=testgen POSTGRES\_USER=testgen POSTGRES\_PASSWORD=CHANGE\_ME\_STRONG\_PASSWORD MONGODB\_HOST=localhost MONGODB\_PORT=27017 MONGODB\_DB=testgen MONGODB\_USER=testgen MONGODB\_PASSWORD=CHANGE\_ME\_STRONG\_PASSWORD REDIS\_HOST=localhost REDIS\_PORT=6379 REDIS\_PASSWORD=CHANGE\_ME\_STRONG\_PASSWORD JWT\_SECRET=CHANGE\_ME\_RANDOM\_STRING\_64\_CHARS API\_BASE\_URL=https://your-domain.com # Licencja LICENSE\_KEY=YOUR\_LICENSE\_KEY\_HERE

**Krok 4: Uruchomienie baz danych**

# Uruchomienie PostgreSQL, MongoDB i Redis docker compose up -d postgres mongodb redis # Weryfikacja docker compose ps # Inicjalizacja schematów bazy danych docker compose run --rm init-db

**Krok 5: Deployment aplikacji**

# Uruchomienie wszystkich serwisów docker compose up -d # Weryfikacja statusu docker compose ps # Sprawdzenie logów docker compose logs -f api-gateway

**Krok 6: Weryfikacja instalacji**

# Test health check curl http://localhost:8080/health # Oczekiwany output: { "status": "healthy", "services": { "api\_gateway": "ok", "postgres": "ok", "mongodb": "ok", "redis": "ok" } }

**Krok 7: Konfiguracja HTTPS (produkcja)**

# Instalacja Certbot dla Let's Encrypt sudo apt install certbot # Generowanie certyfikatu sudo certbot certonly --standalone -d your-domain.com # Konfiguracja Nginx jako reverse proxy sudo nano /etc/nginx/sites-available/testgen # Restart Nginx sudo systemctl restart nginx

**Krok 8: Utworzenie pierwszego użytkownika administratora**

# Utworzenie admina docker compose exec api python manage.py create-admin \ --email admin@your-company.com \ --password CHANGE\_ME # Login w przeglądarce https://your-domain.com

**Instalacja zakończona!** System TestGEN jest gotowy do użycia. Zaloguj się używając powyższych danych i rozpocznij konfigurację pierwszego projektu.

### **9.3.3.** Kubernetes Deployment

Dla większych wdrożeń zalecamy użycie Kubernetes:

# Przykładowy Kubernetes deployment apiVersion: apps/v1 kind: Deployment metadata: name: testgen-api namespace: testgen spec: replicas: 3 selector: matchLabels: app: testgen-api template: metadata: labels: app: testgen-api spec: containers: - name: api image: testgen/api:latest ports: - containerPort: 8000 env: - name: POSTGRES\_HOST valueFrom: secretKeyRef: name: testgen-secrets key: postgres-host - name: JWT\_SECRET valueFrom: secretKeyRef: name: testgen-secrets key: jwt-secret resources: requests: memory: "2Gi" cpu: "1000m" limits: memory: "4Gi" cpu: "2000m" livenessProbe: httpGet: path: /health port: 8000 initialDelaySeconds: 30 periodSeconds: 10 readinessProbe: httpGet: path: /ready port: 8000 initialDelaySeconds: 10 periodSeconds: 5 --- apiVersion: v1 kind: Service metadata: name: testgen-api-service namespace: testgen spec: selector: app: testgen-api ports: - protocol: TCP port: 80 targetPort: 8000 type: LoadBalancer

## **9.4.** Materiały szkoleniowe

### **9.4.1.** Program szkoleń

System TestGEN będzie wspierany kompleksowym programem szkoleń:

|  |  |  |  |
| --- | --- | --- | --- |
| **Szkolenie** | **Grupa docelowa** | **Czas trwania** | **Format** |
| TestGEN dla użytkowników | Testerzy, QA Engineers | 1 dzień | Online/Stacjonarnie |
| TestGEN Advanced | Senior Testerzy, Lead QA | 2 dni | Online/Stacjonarnie |
| TestGEN dla deweloperów | Programiści | 1 dzień | Online/Stacjonarnie |
| TestGEN API i integracje | DevOps, Architects | 1 dzień | Online |
| TestGEN Administration | System Administrators | 2 dni | Stacjonarnie |

## **9.5.** Dane wejściowe dla testów funkcjonalności w trakcie developmentu i wyniki działania

Muszą zostać przygotowane i dostarczone wszelkie dane wejściowe i wyjściowe dla każdej wydevelopowanej funkcjonalności które posłużyły do jej testowania. W szczególności powinny to być:

* przykładowe kody źródłowe
* dokumentacja (techniczna, biznesowa podręczniki użytkownika
* GUI, diagramy UML

Wykorzystane scenariusze weryfikacji poprawności działania systemu, w tym testy integracyjne, wydajnościowe i bezpieczeństwa.

Dodatkowo należy przygotować dane wyjściowe i raport testów potwierdzających dostarczenie funkcjonalności i jej poprawne działanie.

# **10.** Rozwój graficznego interfejsu użytkownika (GUI) oraz systemami płatności

W ramach tego zadania TestGEN osiągnie swoją finalną wersję, obejmującą pełnoprawny graficzny interfejs użytkownika (GUI), zgodny ze standardem WCAG 2.1 a także integrację systemu płatności oraz opracowanie dokumentacji wdrożeniowej.

## **10.1.** Opracowanie GUI zgodnego z WCAG 2.1 i integracja systemu płatności

Na tym etapie zostanie stworzony intuicyjny, dostępny i ergonomiczny interfejs graficzny TestGEN, który będzie spełniał wymagania WCAG 2.1 (Web Content Accessibility Guidelines), zapewniając wygodne korzystanie z systemu zarówno dla osób bez doświadczenia w programowaniu, jak i dla użytkowników z różnymi potrzebami dostępności.

### **10.1.1.** Interfejs UI

Interfejs będzie zawierał kluczowe elementy dostępności zgodnie z WCAG 2.1, w tym:

* Kontrastowanie czcionki i elementów graficznych – zapewniające czytelność dla
* osób słabowidzących.
* Możliwość powiększania czcionki – użytkownicy będą mogli dostosować rozmiar
* tekstu bez utraty czytelności interfejsu.
* Obsługa czytników ekranu – wszystkie kluczowe elementy będą oznaczone alternatywnymi
* opisami tekstowymi.
* Pełna nawigacja klawiaturą – interfejs będzie w pełni funkcjonalny bez konieczności
* używania myszy.
* Eliminacja migających i dynamicznych elementów – zgodnie z wytycznymi WCAG,
* aby uniknąć problemów z dostępnością dla osób z epilepsją czy nadwrażliwością
* na ruch.
* Zgodność z technologiami asystującymi – aplikacja będzie współpracować z systemami
* wspomagającymi osoby z niepełnosprawnościami.

#### **4.5.4.1.** Graficzny Interfejs Użytkownika (GUI)

Opis ogólny i wykorzystane technologie:

Opracowanie finalnego graficznego interfejsu użytkownika (GUI) systemu TestGEN, zapewniającego intuicyjną i zgodną ze standardem WCAG 2.1 obsługę zarówno dla doświadczonych testerów, jak i osób bez technicznego doświadczenia.

Projekt GUI uwzględni kluczowe wymagania dostępności, w tym:

* Kontrastowanie czcionek i elementów graficznych dla lepszej czytelności,
* Możliwość powiększania tekstu bez utraty funkcjonalności interfejsu,
* Obsługę czytników ekranu, zapewniającą dostępność dla osób niewidomych,
* Pełną nawigację klawiaturą, eliminującą konieczność użycia myszy,
* Alternatywne teksty dla grafik i ikon,
* Unikanie migających elementów zgodnie z wytycznymi WCAG 2.1

GUI systemu TestGEN będzie powinno być intuicyjne zapewniające dostępność dla każdego poprzez wizualne formularze, podpowiedzi oraz moduł asystenta, eliminując potrzebę znajomości programowania.

Dzięki funkcji "przeciągnij i upuść" (lub jej odpowiadającej), użytkownicy łatwo dodadzą pliki i skonfigurują testy. Dzięki temu nawet osoby bez doświadczenia technicznego będą mogły skutecznie korzystać z TestGEN.

* (WCAG 2.1)
* Dodanie UML
* Dodanie dokumentacji
* Dodanie grafik GUI (np. makiet)
* Dodanie kodu
* Wizualne formularze (?)
* Podpowiedzi (dymek z podpowiedzią jak najedziemy na element systemu, np. przycisk)
* Moduł asystenta (np. okienko do darmowego LLMa)

### **10.1.2.** Integracja z modułem płatności

Dodatkowo zostanie wdrożony system płatności, który umożliwi użytkownikom zakup abonamentu na korzystanie z TestGEN i będzie wspierał różne metody płatności.

Opis ogólny i wykorzystane technologie

#### **10.1.2.1.** Integracja

Zintegrowanie wszystkich modułów systemu TestGEN, tworząc jeden spójny i w pełni funkcjonalny system.

Proces obejmie połączenie mechanizmów analizy kodu, modułów testowania bezpieczeństwa oraz algorytmów sztucznej inteligencji, aby zapewnić kompleksowe wsparcie w testowaniu

oprogramowania.

#### **10.1.2.2.** Wdrożenie systemu płatności

Umożliwi zakup abonamentu na korzystanie z TestGEN. Moduł ten będzie w pełni zintegrowany z interfejsem użytkownika i dostosowany do różnych metod płatności, zapewniając wygodną

realizację transakcji.

#### **10.1.2.3.** Testy integracyjne

Weryfikacja poprawność współdziałania wszystkich komponentów, a optymalizacja wydajności zagwarantuje stabilność działania przy dużym wolumenie danych. Finalna wersja TestGEN będzie gotowa do komercyjnego wdrożenia i użytkowania przez klientów.

# **11.** ZAŁĄCZNIKI

## **11.1.** Słownik pojęć

|  |  |
| --- | --- |
| **Termin** | **Definicja** |
| AST (Abstract Syntax Tree) | Drzewo składniowe - reprezentacja struktury kodu źródłowego w formie drzewa, używana do analizy statycznej |
| BERT | Bidirectional Encoder Representations from Transformers - model języka naturalnego używany do analizy tekstów |
| CI/CD | Continuous Integration / Continuous Deployment - metodologia ciągłej integracji i wdrażania kodu |
| CSRF | Cross-Site Request Forgery - typ ataku webowego polegający na wykonaniu nieautoryzowanych akcji w imieniu użytkownika |
| DAST | Dynamic Application Security Testing - dynamiczne testowanie bezpieczeństwa aplikacji |
| GUI | Graphical User Interface - graficzny interfejs użytkownika |
| JWT | JSON Web Token - standard tokenu służącego do autoryzacji |
| LLM | Large Language Model - duży model języka naturalnego (np. GPT, Claude) |
| NER | Named Entity Recognition - rozpoznawanie nazwanych encji w tekście |
| NLP | Natural Language Processing - przetwarzanie języka naturalnego |
| OAuth 2.0 | Protokół autoryzacji umożliwiający aplikacjom dostęp do zasobów użytkownika |
| OCR | Optical Character Recognition - rozpoznawanie tekstu z obrazów |
| PoC | Proof of Concept - dowód koncepcji, prototyp demonstrujący możliwość realizacji |
| RODO/GDPR | Rozporządzenie o Ochronie Danych Osobowych / General Data Protection Regulation |
| RPO | Recovery Point Objective - maksymalny akceptowalny okres utraty danych |
| RTO | Recovery Time Objective - maksymalny akceptowalny czas przywracania systemu |
| SaaS | Software as a Service - model dostarczania oprogramowania jako usługi w chmurze |
| SAST | Static Application Security Testing - statyczne testowanie bezpieczeństwa aplikacji |
| SCA | Software Composition Analysis - analiza składu oprogramowania (biblioteki, zależności) |
| SQL Injection | Typ ataku polegający na wstrzykiwaniu złośliwego kodu SQL do zapytań bazodanowych |
| Taint Analysis | Analiza przepływu danych śledząca niebezpieczne dane od źródła do miejsca użycia |
| TestSpark | Autorski silnik AI systemu TestGEN odpowiedzialny za analizę i generowanie testów |
| UML | Unified Modeling Language - język modelowania graficznego systemów informatycznych |
| WCAG | Web Content Accessibility Guidelines - wytyczne dostępności treści internetowych |
| XSS | Cross-Site Scripting - typ ataku polegający na wstrzyknięciu złośliwego skryptu do strony webowej |

## **10.2.** Przykłady kodów źródłowych

### **10.2.1.** Przykład 1: Kod z podatnością SQL Injection

**Kod podatny (Python):**

import sqlite3 def get\_user(username): conn = sqlite3.connect('users.db') cursor = conn.cursor() # PODATNOŚĆ: SQL Injection query = f"SELECT \* FROM users WHERE username = '{username}'" cursor.execute(query) user = cursor.fetchone() conn.close() return user # Atak: get\_user("admin' OR '1'='1") # Zwróci wszystkich użytkowników!

**Kod bezpieczny (Python):**

import sqlite3 def get\_user(username): conn = sqlite3.connect('users.db') cursor = conn.cursor() # BEZPIECZNE: Parametryzowane zapytanie query = "SELECT \* FROM users WHERE username = ?" cursor.execute(query, (username,)) user = cursor.fetchone() conn.close() return user

**Testy wygenerowane przez TestGEN:**

import pytest from app import get\_user def test\_sql\_injection\_protection(): """Test that SQL injection attempts are prevented""" # Próba ataku SQL injection malicious\_input = "admin' OR '1'='1" result = get\_user(malicious\_input) # Powinno zwrócić None (brak użytkownika) zamiast wszystkich użytkowników assert result is None, "SQL injection vulnerability detected!" def test\_get\_user\_valid(): """Test getting user with valid username""" result = get\_user("john\_doe") assert result is not None assert result[1] == "john\_doe" # username column def test\_get\_user\_invalid(): """Test getting user with non-existent username""" result = get\_user("nonexistent\_user\_12345") assert result is None

### **10.2.2.** Przykład 2: Kod z podatnością XSS

**Kod podatny (JavaScript/React):**

function UserComment({ comment }) { // PODATNOŚĆ: XSS - dangerouslySetInnerHTML return (

); } // Atak: comment.text = ""

**Kod bezpieczny (JavaScript/React):**

import DOMPurify from 'dompurify'; function UserComment({ comment }) { // BEZPIECZNE: Sanityzacja HTML const sanitizedText = DOMPurify.sanitize(comment.text); return (

); } // Lub jeszcze lepiej - bez innerHTML: function UserCommentSafe({ comment }) { return (

{comment.text}

); }

**Testy E2E wygenerowane przez TestGEN (Playwright):**

import { test, expect } from '@playwright/test'; test('XSS protection in comments', async ({ page }) => { await page.goto('http://localhost:3000/comments'); // Próba wstrzyknięcia XSS const xssPayload = ''; await page.fill('#comment-input', xssPayload); await page.click('#submit-comment'); // Sprawdź, że skrypt nie został wykonany page.on('dialog', dialog => { throw new Error('XSS vulnerability: alert was triggered!'); }); // Sprawdź, że tekst jest wyświetlony jako plain text, nie jako HTML const commentText = await page.textContent('.comment'); expect(commentText).toContain('