

# Opracowanie podsystemów sprawdzających zadania dla różnych języków dla Systemu STOS

## Scrum: Backlog produktu

Wersja 1.0.0

Data utworzenia obecnej wersji: 01-04-2025

Data utworzenia dokumentu: 01-04-2025

Mateusz Fydrych  
193410

Damian Trowski  
193443

Maciej Raciniewski  
189774

## Spis treści

1	O projekcie i produkcji	2
2	Persony użytkowników	2
3	Scenariusze użycia produktu	3
4	Backlog produktu	5
4.1	Elementy backlogu produktu . . . . .	5
4.2	GitHub Project . . . . .	5
5	Kryteria akceptacji	6
6	Definicja ukończenia	7

# 1 O projekcie i produkcie

Projekt dotyczy opracowania podsystemów sprawdzających zadania dla różnych języków programowania dla Systemu STOS. System ten będzie służył do automatycznego sprawdzania zadań programistycznych, co usprawni proces oceniania prac studentów oraz zapewni jednolite kryteria sprawdzania. Poszczególne podsystemy zostaną zrealizowane w formie oddzielnych kontenerów tak by wykorzystać modularność głównego systemu a z drugiej strony by umożliwić efektywne skalowanie całego rozwiązania.

## 2 Persony użytkowników

### Persona 1: Wykładowca



- Jakub Kowalski
- 44 lata
- Doktor inżynier informatyki
- Prowadzi kursy programowania na uczelni
- Doświadczony programista z 15-letnim stażem
- Wykształcenie: doktorat z informatyki, specjalizacja w algorytmach i strukturach danych
- Stanowisko: Adiunkt na wydziale WETI
- Stan cywilny: żonaty, dwoje dzieci (10 i 14 lat)
- Zaawansowanie w korzystaniu z IT: bardzo wysokie w obszarach zawodowych, średnie w nowych narzędziach dydaktycznych
- Zainteresowania: programowanie funkcjonalne, gotowanie, turystyka górska

#### Problemy:

- Przeciążenie pracą przy ręcznym sprawdzaniu zadań
- Niejednolite kryteria oceny różnych osób prowadzących ten sam przedmiot
- Trudność w wykryciu plagiatów
- Brak czasu na indywidualną pomoc studentom z powodu długiego czasu sprawdzania zadań

#### Potrzeby:

- Oszczędność czasu przy sprawdzaniu zadań studentów
- Standaryzacja procesu oceniania
- Możliwość szybkiej weryfikacji dużej ilości prac
- Generowanie raportów z wynikami studentów
- Możliwość analizy logów zawierających błędy wykonania kodu

## Persona 2: Studentka Informatyki



- Maja Nowak
- 20 lat
- Studentka 2 roku informatyki
- Uczy się kilku różnych języków programowania
- Wykształcenie: 1.5 roku studiów informatycznych, ukończone technikum o profilu informatycznym
- Stan rodzinny: singielka, mieszka w akademiku
- Zaawansowanie w korzystaniu z IT: wysokie, sprawnie posługuje się narzędziami programistycznymi
- Zainteresowania: programowanie mobilne, gry komputerowe, siatkówka

### Problemy:

- Stres związany z oddawaniem zadań bez możliwości wcześniejszej weryfikacji
- Zbyt długi czas oczekiwania na ocenę rozwiązania
- Niejasne kryteria oceny błędów w kodzie
- Trudność w diagnozowaniu przyczyn niepowodzenia kodu w środowisku testowym

### Potrzeby:

- Szybka informacja zwrotna dotycząca poprawności rozwiązania
- Jasne kryteria oceny kodu
- Możliwość wielokrotnego testowania rozwiązania przed ostatecznym oddaniem
- Szczegółowe komunikaty o błędach ułatwiające ich naprawę
- Dostęp do systemu z różnych urządzeń

## 3 Scenariusze użycia produktu

### Scenariusz 1: Prowadzenie laboratorium z programowania

Dr Jakub Kowalski przygotowuje się do prowadzenia zajęć laboratoryjnych z programowania. Musi stworzyć zestaw zadań dla 36 studentów, które będzie następnie musiał sprawdzić. Korzystając z Systemu STOS, tworzy nowe zadania i definiuje testy, które będą weryfikować poprawność rozwiązań studentów.

Dla każdego zadania określa:

- Język programowania (C, Python, Rust)

- Wymagania funkcjonalne
- Dane testowe wejściowe i oczekiwane wyniki
- Kryteria oceny (poprawność wyników, wydajność, jakość kodu)

System automatycznie obsługuje każdy dostępny język programowania. Uruchamia go, a następnie testuje.

Na zajęciach studenci, w tym Maja Nowak, otrzymują dostęp do zadań. Maja rozwiązuje zadanie w C, po czym przesyła swój kod do systemu. System STOS umieszcza jej kod w izolowanym kontenerze, kompiluje go i uruchamia zestaw testów. Po chwili Maja otrzymuje automatyczny raport z wynikami testów, wskazujący które przypadki testowe przeszły pomyślnie, a które nie.

Maja analizuje wyniki, poprawia swoje rozwiązanie i ponownie przesyła kod do weryfikacji. Ten cykl powtarza się, aż wszystkie testy zostaną zaliczone.

Dr Kowalski po zakończeniu laboratorium otrzymuje automatycznie wygenerowany raport z wynikami wszystkich studentów. Dzięki temu może szybko ocenić postępy grupy i zidentyfikować osoby, które mogą potrzebować dodatkowej pomocy. System oszczędza mu wiele godzin ręcznego sprawdzania rozwiązań i pozwala skupić się na merytorycznej pomocy studentom.

## Scenariusz 2: Organizacja konkursu programistycznego

Dr Jakub Kowalski organizuje ogólnouczelniany konkurs programistyczny na wydziale WETI. W konkursie ma wziąć udział ponad 100 uczestników, którzy będą rozwiązywać 10 zadań o różnym stopniu trudności w ciągu 5 godzin. Ze względu na dużą liczbę uczestników i rozwiązań, dr Kowalski decyduje się wykorzystać system STOS.

Przed konkursem, dr Kowalski korzysta z interfejsu administratora systemu STOS, aby:

- Utworzyć nowy konkurs z określoną datą rozpoczęcia i zakończenia
- Dodać zestaw zadań o zróżnicowanym poziomie trudności
- Zdefiniować precyzyjne testy dla każdego zadania w trzech językach (C, Python, Rust)
- Skonfigurować limity czasowe i pamięciowe dla każdego zadania poprzez intuicyjny panel ustawień
- Zdefiniować system punktacji oparty na poprawności i wydajności rozwiązań

W dniu konkursu, Maja Nowak wraz z innymi uczestnikami logują się do systemu używając swoich indywidualnych kont. Po zalogowaniu, Maja widzi listę dostępnych zadań wraz z ich opisami, limitami czasowymi i przykładowymi testami. Interfejs użytkownika jest przejrzysty i pozwala na łatwe przełączanie się między zadaniami.

Maja wybiera pierwsze zadanie, czyta jego specyfikację i rozpoczyna kodowanie w zintegrowanym edytorze kodu lub w swoim lokalnym środowisku. Po ukończeniu rozwiązania w Pythonie, przesyła kod za pomocą przycisku "Prześlij rozwiązanie".

System natychmiast rozpoczyna proces weryfikacji:

- Umieszcza kod w systemie
- Kompiluje kod z odpowiednimi flagami
- Uruchamia testy jednostkowe i wydajnościowe

Po kilku sekundach, Maja otrzymuje szczegółowy raport zawierający:

- Ogólny status rozwiązania (zaakceptowane/odrzucone)
- Listę testów z informacją, które przeszły, a które nie
- Komunikaty o błędach kompilacji lub wykonania (jeśli wystąpiły)
- Statystyki dotyczące czasu wykonania i zużycia pamięci
- Liczbę zdobytych punktów

Maja zauważa, że jej rozwiązanie nie przechodzi jednego z testów wydajnościowych. Analizuje komunikat błędu, optymalizuje swój algorytm i ponownie przesyła rozwiązanie. Tym razem wszystkie testy przechodzą pomyślnie.

Po zakończeniu konkursu, system STOS automatycznie generuje i publikuje ranking uczestników. Dr Kowalski ma dostęp do panelu administracyjnego, gdzie może przeglądać:

- Szczegółowe wyniki każdego uczestnika
- Statystyki dotyczące popularności wybranych języków programowania
- Analizę trudności poszczególnych zadań na podstawie liczby poprawnych rozwiązań
- Historię przesłanych rozwiązań wraz z kodem źródłowym

## 4 Backlog produktu

### 4.1 Elementy backlogu produktu

Tabela 1: Elementy backlogu produktu z przypisanymi

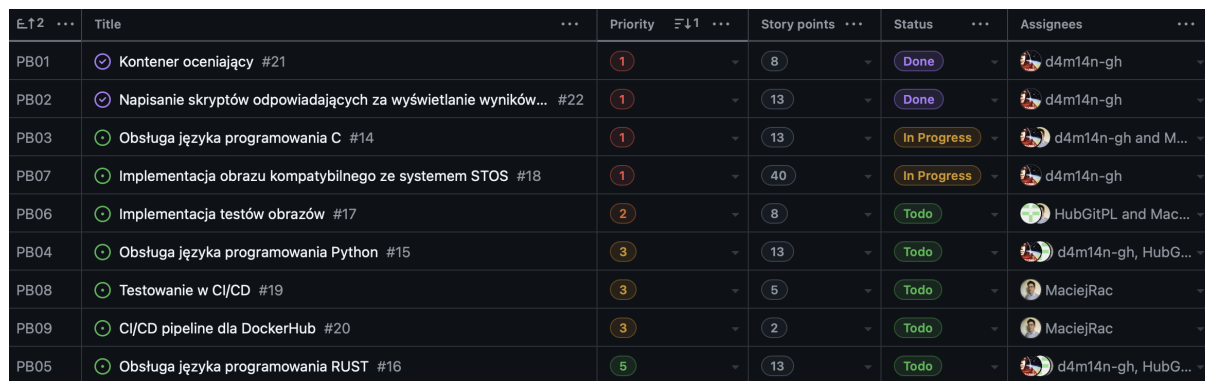
ID	Element backlogu	Priorytet	Story points
PB01	Kontener oceniający	Bardzo wysoki (1)	8
PB02	Napisanie skryptów odpowiadających za wyświetlanie wyników oraz poprawne uruchomienie	Bardzo wysoki (1)	13
PB03	Obsługa języka programowania C	Wysoki (2)	13
PB04	Obsługa języka programowania Python	Średni (3)	13
PB05	Obsługa języka programowania RUST	Średni (3)	13
PB06	Implementacja testów obrazów	Wysoki (2)	8
PB07	Implementacja obrazu kompatybilnego ze systemem STOS	Bardzo Wysoki (1)	40
PB08	Testowanie w CI/CD	Średni (3)	5
PB09	CI/CD pipeline dla DockerHub	Średni (3)	2

#### Wyjaśnienie priorytetów i rozmiarów:

- Priorytet: skala 1-5, gdzie 1 oznacza najwyższy priorytet
- Story points: skala M. Cohna (1, 2, 3, 5, 8, 13, 20, 40, 100), gdzie 100 punktów to najbardziej złożony element.
- Przy szacowaniu rozmiaru danego elementu zespół użył techniki „Planning Poker”. Pozwoliło to na precyzyjne dostosowanie rozmiarów.
- Lista jest posortowana według unikalnego ID (rosnąco)

### 4.2 GitHub Project

Backlog produktu jest na bieżąco aktualizowany w GitHub Project. Narzędzie to pozwala na efektywne zarządzanie elementami backlogu, śledzenie postępów oraz wizualizację całego procesu rozwoju produktu w jednym miejscu. Dzięki integracji z repozytorium kodu, zespół może łatwo łączyć zadania z konkretnymi zmianami w kodzie źródłowym.



ID	Title	Priority	Story points	Status	Assignees
PB01	Kontener oceniający #21	1	8	Done	d4m14n-gh
PB02	Napisanie skryptów odpowiadających za wyświetlanie wyników... #22	1	13	Done	d4m14n-gh
PB03	Obsługa języka programowania C #14	1	13	In Progress	d4m14n-gh and M...
PB07	Implementacja obrazu kompatybilnego ze systemem STOS #18	1	40	In Progress	d4m14n-gh
PB06	Implementacja testów obrazów #17	2	8	Todo	HubGitPL and Mac...
PB04	Obsługa języka programowania Python #15	3	13	Todo	d4m14n-gh, HubG...
PB08	Testowanie w CI/CD #19	3	5	Todo	MaciejRac
PB09	CI/CD pipeline dla DockerHub #20	3	2	Todo	MaciejRac
PB05	Obsługa języka programowania RUST #16	5	13	Todo	d4m14n-gh, HubG...

Rysunek 1: Wizualizacja elementów backlogu produktu w GitHub Project

## 5 Kryteria akceptacji

### **PB01: Kontener oceniający**

- System poprawnie izoluje wykonywany kod
- Mechanizmy zabezpieczające przed złośliwym kodem
- Ograniczenie czasowe i zasobowe dla uruchamianych programów
- Poprawne raportowanie wyników wykonania
- Obsługa różnych języków programowania
- Testy penetracyjne testujące podatność na ataki poprzez sprawdzanie złośliwego kodu

### **PB02: Napisanie skryptów odpowiadających za wyświetlanie wyników oraz poprawne uruchomienie**

- Skrypty poprawnie wyświetlają wyniki testów
- Jednolity format wyświetlania dla wszystkich języków programowania
- Obsługa błędów i wyjątków podczas uruchomienia
- Czytelny interfejs użytkownika
- Możliwość konfiguracji sposobu wyświetlania

### **PB03: Obsługa języka programowania C**

- System poprawnie kompiluje kod C
- System poprawnie uruchamia skompilowany program
- System zwraca standardowe wyjście oraz błędy z programu
- Obsługa różnych standardów języka C

### **PB04: Obsługa języka programowania Python**

- System poprawnie uruchamia skrypty Python
- Wsparcie dla różnych wersji Pythona (3.7+)
- Możliwość instalacji dodatkowych bibliotek poprzez pip
- Poprawna obsługa standardowego wejścia/wyjścia

### **PB05: Obsługa języka programowania RUST**

- System poprawnie kompiluje kod Rust
- System poprawnie uruchamia skompilowany program
- Wsparcie dla różnych wersji kompilatora Rust
- Obsługa testów jednostkowych napisanych w Rust

### **PB06: Implementacja testów obrazów**

- Automatyczne testy sprawdzające poprawność działania obrazów Docker
- Testy weryfikujące izolację kontenerów
- Testy wydajnościowe obrazów
- Testy kompatybilności z różnymi wersjami Dockera
- Raportowanie wyników testów w czytelnym formacie

### **PB07: Implementacja obrazu kompatybilnego ze systemem STOS**

- Obraz Docker zgodny ze specyfikacją systemu STOS

- Implementacja interfejsu komunikacyjnego z głównym systemem
- Obsługa standardowych operacji (kompilacja, uruchomienie, testowanie)
- Mechanizmy zabezpieczające przed złośliwym kodem
- Optymalizacja rozmiaru i wydajności obrazu

#### **PB08: Testowanie w CI/CD**

- Konfiguracja automatycznych testów w środowisku CI/CD (GitHub Actions)
- Automatyczne testy dla każdego pull requesta
- Testy integracyjne z systemem STOS
- Generowanie raportów z pokrycia kodu testami
- Automatyczne powiadomienia o wynikach testów

#### **PB09: CI/CD pipeline dla DockerHub**

- Automatyczne budowanie obrazów Docker po zatwierdzeniu zmian
- Publikowanie obrazów w repozytorium DockerHub
- Wersjonowanie obrazów zgodne z semantycznym wersjonowaniem
- Automatyczne testy obrazów przed publikacją
- Dokumentacja procesu CI/CD

### **Ogólne kryteria akceptacji**

- System umożliwia sprawdzanie zadań w co najmniej 3 różnych językach programowania
- System działa stabilnie
- System jest odporny na próby nadużyć (próby złośliwego kodu)
- System jest skalowalny - możliwe jest dodanie obsługi nowych języków programowania
- Poszczególne podsystemy zrealizowane jako oddzielne kontenery umożliwiają łatwą rozbudowę i skalowanie całego rozwiązania
- Przechodzi wszystkie testy akceptacyjne
- Dokumentacja jest kompletna i zrozumiała dla użytkowników

## **6 Definicja ukończenia**

Element backlogu produktu można uznać za ukończony, gdy:

- Kod implementujący element został napisany i umieszczony w repozytorium GitHub
- Przeprowadzono testy jednostkowe, integracyjne i penetracyjne, które zakończyły się pomyślnie
- Kod przeszedł proces przeglądu (code review) przez co najmniej jednego członka zespołu
- Funkcjonalność została zintegrowana z resztą systemu
- Dokumentacja elementu została napisana i zaakceptowana przez zespół
- Funkcjonalność została przetestowana na różnych środowiskach (Linux, Windows, MacOS)
- Obrazy (Docker) zostały opublikowane w DockerHub
- Element spełnia wszystkie kryteria akceptacji zdefiniowane dla niego