

Opracowanie podsystemów sprawdzających zadania dla różnych języków dla Systemu STOS

Scrum: Backlog sprintu

Wersja 1.0.0

Data utworzenia obecnej wersji: 29-04-2025

Data utworzenia dokumentu: 29-04-2025

Mateusz Fydrych
193410

Damian Trowski
193443

Maciej Raciniewski
189774

Spis treści

1	O projekcie i produkcji	2
2	Oszacowanie rozmiaru backlogu produktu	2
3	Założenia i dobór zakresu sprintu	2
4	Cel sprintu	3

1 O projekcie i produkcji

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, aby wykorzystać modularność głównego systemu i umożliwić efektywne skalowanie całego rozwiązania.

W ramach oceny zadania pierwszym krokiem jest kompilacja kodu źródłowego, za które odpowiada kontener do kompilacji danego języka. Jeżeli proces przebiegnie poprawnie (kod źródłowy jest poprawny), skompilowany program trafia do kontenera odpowiadającego za uruchamianie go dla każdego przypadku testowego podanego na wejściu (pliki w postaci 0-n.in). Następnie kontener zapisuje wyniki działania dla każdego przypadku w osobnych plikach (0-n.stdout.out i 0-n.stderr.out) oraz dane takie jak czas działania i zużycie zasobów. Ostatnim etapem jest kontener oceniający, który porównuje wyjście programu z poprawnymi odpowiedziami i dla każdego testu wystawia ocenę.

2 Oszacowanie rozmiaru backlogu produktu

Backlog produktu został oszacowany podczas sesji Planning Poker z udziałem całego zespołu deweloperskiego. Każdy element backlogu został szczegółowo omówiony, a następnie oszacowany w story points według skali M. Cohna (1, 2, 3, 5, 8, 13, 20, 40, 100).

Jeśli każdy z członków zespołu miał różną opinię odnośnie story points dla danego elementu backlogu, proces głosowania był powtarzany aż przynajmniej 2 osoby oszacowały story points na tę samą ilość punktów. Takie podejście pozwalało na stopniowe zbliżanie się do konsensusu i zapewniało, że ostateczne oszacowanie odzwierciedla wspólne zrozumienie złożoności zadania przez zespół.

Wyniki oszacowania:

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	Bardzo wysoki (1)	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

3 Założenia i dobór zakresu sprintu

- **Nazwa Sprintu:** "Sprint 1 - Podstawowa infrastruktura i obsługa języka C"
- **Długość Sprintu:** 2 tygodnie (14 dni, 10 dni roboczych)
- **Data Rozpoczęcia:** 30.04.25r.
- **Data Zakończenia:** 14.05.25r.
- **Zespół Deweloperski:** 3 osoby
- **Pojemność Zespołu:** 3 osoby * średnio 3h/dzień * 10 dni roboczych = 90h
- **Rezerwa na inne prace:** 20% rezerwy na spotkania Scrumowe = 18h
- **Zakładana średnia szybkość zespołu:** 15 SP na sprint

Wybrany element do Sprintu: PB03 Obsługa języka programowania C

Uzasadnienie wyboru zakresu:

Wybrany do sprintu element backlogu produktu, PB03 Obsługa języka programowania C, posiada priorytet 1 'Bardzo wysoki'. Został on wybrany, aby wcześniej dostarczyć kluczową funkcjonalność – wsparcie dla pierwszego języka programowania, co pozwoli na weryfikację podstawowego przepływu pracy w systemie STOS. Spośród elementów jeszcze nieskończonych o tym samym priorytecie (Bardzo wysoki - 1), PB03 ma największą estymację (13 SP), co podkreśla jego istotną rolę w rozwoju produktu na tym etapie.

Estymowana złożoność tego elementu wynosi 13 Story Points. Jest to wartość nieznacznie niższa od wstępnie założonej średniej prędkości zespołu (15 SP na sprint). Taki dobór zakresu uznano za rozsądny, biorąc pod uwagę, że jest to pierwszy sprint projektu. Zapewnia to niezbędny margines bezpieczeństwa dla zespołu, który dopiero wdraża procesy i może napotkać nieprzewidziane wyzwania związane z konfiguracją środowiska czy doprecyzowaniem wymagań.

4 Cel sprintu

Celem sprintu jest stworzenie podstawowej infrastruktury systemu STOS umożliwiającej automatyczne sprawdzanie zadań w języku C. Po zakończeniu sprintu system powinien być w stanie skompilować kod w języku C, uruchomić go z różnymi danymi wejściowymi, porównać wyniki z oczekiwanymi odpowiedziami. Dzięki temu wykładowcy będą mogli zacząć testować system na prostych zadaniach programistycznych w języku C.




5,6 Backlog sprintu i kryteria akceptacji


PB03: Obsługa języka programowania C (13 SP)

○ Analiza wymagań dla obsługi języka #24
○ Zaprojektowanie kontenera do kompilacji kodu #25
○ Implementacja mechanizmu kompilacji #26
○ Implementacja obsługi różnych standardów języka #27
○ Implementacja mechanizmu uruchamiania skompilowanych programów #28
○ Implementacja mechanizmu przekazywania danych wejściowych/wyjściowych #29
○ Testowanie kontenera dla języka C #30
○ Dokumentacja kontenera dla języka C #31


Rysunek 1: Lista zadań wytwórczych

Analiza wymagań dla obsługi języka #24

 Open |  Parent: Obsługa języka programowania C |  Stos-2025/Projekt_Grupowy-2025 **Public**


 HubGitPL opened 2 hours ago · edited by HubGitPL Edits ▾ ...


- **Oszacowanie:** 4h
- **Opis:** Zespół przeprowadzi sesję analityczną w celu zidentyfikowania wszystkich wymagań funkcjonalnych i нефункциональных związanych z obsługą języka C w systemie.
- **Rezultat:** Po sesji analitycznej każdy członek zespołu będzie znał docelowe wymagania.

[Create sub-issue](#) ▾ 

Rysunek 2: Analiza wymagań dla obsługi języka

Zaprojektowanie kontenera do kompilacji kodu #25

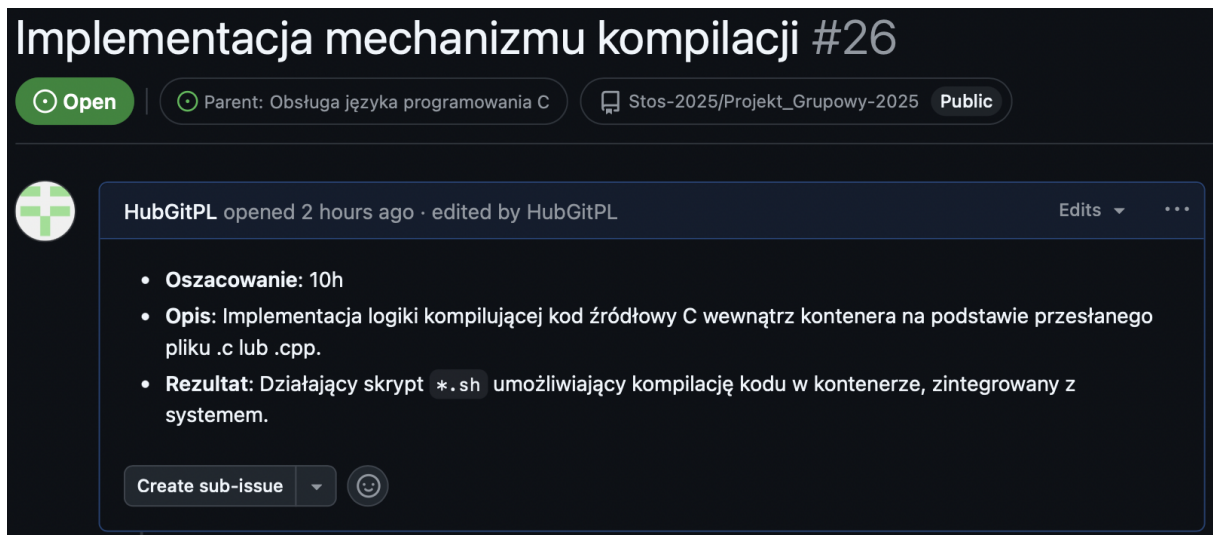
 Open |  Parent: Obsługa języka programowania C |  Stos-2025/Projekt_Grupowy-2025 **Public**

 HubGitPL opened 2 hours ago · edited by HubGitPL Edits ▾ ...

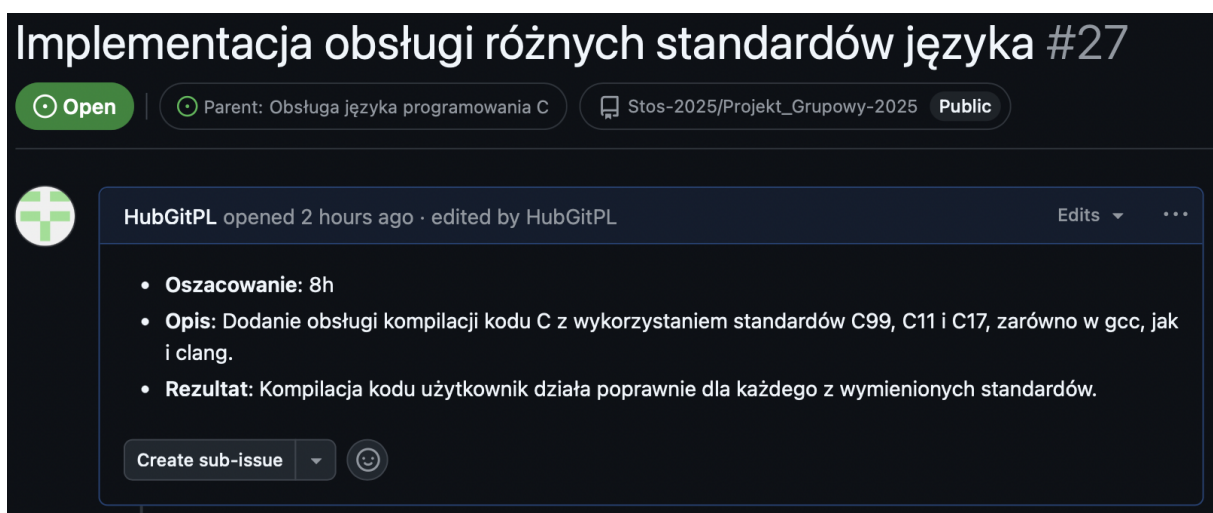
- **Oszacowanie:** 6h
- **Opis:** Opracowanie projektu kontenera Docker zawierającego środowisko do kompilacji kodu C z użyciem wybranych kompilatorów i narzędzi gcc i clang. Z Uwzględnieniem możliwości obsługi różnych standardów.
- **Rezultat:** Plik `Dockerfile` umieszczony w repozytorium.

[Create sub-issue](#) ▾ 

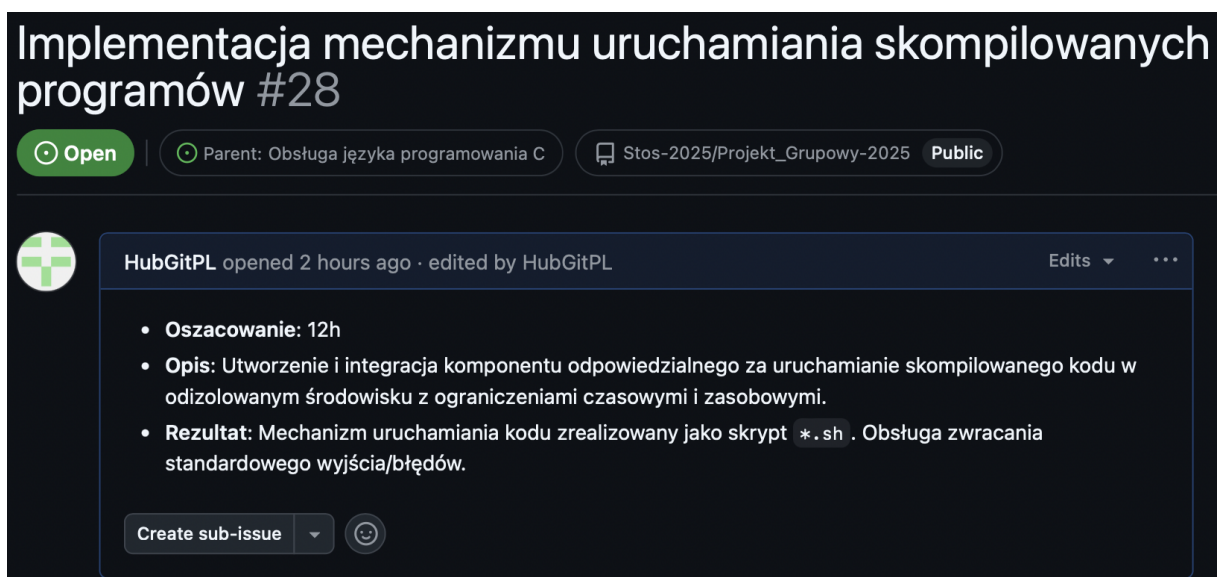
Rysunek 3: Zaprojektowanie kontenera do kompilacji kodu C



Rysunek 4: Implementacja mechanizmu kompilacji





Rysunek 5: Implementacja obsługi różnych standardów języka C





Rysunek 6: Implementacja mechanizmu uruchamiania skompilowanych programów

Implementacja mechanizmu przekazywania danych wejściowych/wyjściowych #29

 Open

 Parent: Obsługa języka programowania C


 Stos-2025/Projekt_Grupowy-2025 **Public**



HubGitPL opened 2 hours ago · edited by HubGitPL Edits ▾ ...


- **Oszacowanie:** 8h
- **Opis:** Implementacja sposobu przekazywania danych wejściowych do programu oraz zbierania wyników ze standardowego wyjścia i błędów.
- **Rezultat:** Możliwość podania danych wejściowych (stdin) do programu, zapis wyników do plików `stdout.txt` i `stderr.txt`. Funkcjonalność przetestowana z przynajmniej dwoma scenariuszami wejścia/wyjścia.


Create sub-issue ▾





Rysunek 7: Implementacja mechanizmu przekazywania danych wejściowych/wyjściowych

Testowanie kontenera dla języka C #30

 Open

 Parent: Obsługa języka programowania C


 Stos-2025/Projekt_Grupowy-2025 **Public**



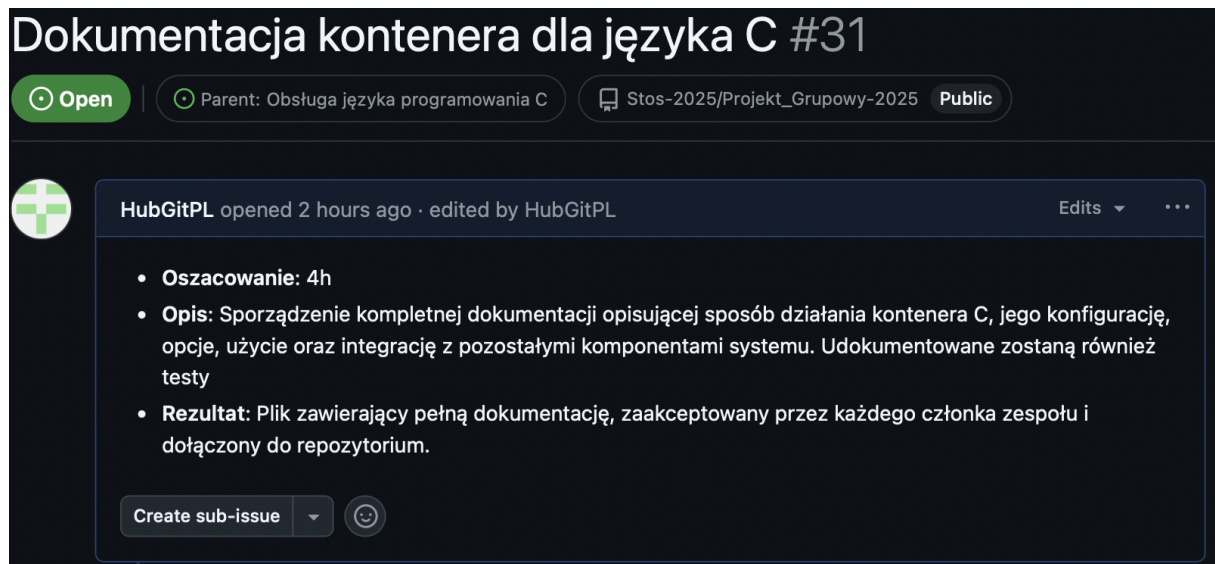
HubGitPL opened 2 hours ago · edited by HubGitPL Edits ▾ ...

- **Oszacowanie:** 10h
- **Opis:** Przeprowadzenie testów jednostkowych, integracyjnych oraz penetracyjnych kontenera odpowiedzialnego za język C. Testy sprawdzają poprawność kompilacji, uruchamiania, izolacji i odporności na złośliwy kod. Testy mają być skupione na skrajnych scenariuszach.
- **Rezultat:** Wszystkie testy zakończone pomyślnie

Create sub-issue ▾



Rysunek 8: Testowanie kontenera dla języka C



Rysunek 9: Dokumentacja kontenera dla języka C

Zadania wytwórcze:

1. Analiza wymagań dla obsługi języka C - 4h
2. Zaprojektowanie kontenera do kompilacji kodu C - 6h
3. Implementacja mechanizmu kompilacji - 10h
4. Implementacja obsługi różnych standardów języka C - 8h
5. Implementacja mechanizmu uruchamiania skompilowanych programów - 12h
6. Implementacja mechanizmu przekazywania danych wejściowych/wyjściowych - 8h
7. Testowanie kontenera dla języka C - 10h
8. Dokumentacja kontenera dla języka C - 4h

Szacowany łączny czas zadań wytwórczych: 62h

Szacowany łączny czas pracy wraz z rezerwą: 80h

7 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
- Dokumentacja elementu została napisana i zaakceptowana przez zespół
- Obraz Docker został opublikowany w DockerHub