Pike - Perfectly Incomplex Konsole Editor

Autorzy

- Maksym Bieńkowski
- Jędrzej Grabski

Opis projektu

Pike jest prostym, intuicyjnym i łatwym do skonfigurowania edytorem tekstowym dla terminala, zaprojektowanym jako alternatywa dla edytorów takich jak `nano`. Projekt ma na celu dostarczenie lekkiego narzędzia do edycji tekstu z podstawowymi funkcjami, takimi jak:

- Obsługa pracy na wielu buforach
- Wyszukiwanie w tekście.
- Obsługa cofania i powtarzania zmian.
- Konfiguracja skrótów klawiszowych.

Dokumentacja

Dokumentacja użytkownika w języku angielskim dostępna jest w repozytorium w pliku `docs/usage.md` Instrukcje instalacji dostępne są w `README`. W repozytorium zawarty jest też drobny opis modułów projektu oraz instrukcje dotyczące pracy z kodem w `docs/development.md`. Znacząca większość funkcji i modułów zawiera obszerne doc dommenty, z których przy pomocy `cargo doc` można wygenerować całkiem solidną dokumentację wnętrzności edytora.

Statystyki kodu

Liczba testów: 78, w tym testy snapshotów UI bazowane na insta Pokrycie testami: 82%, obliczone przy pomocy tarpaulin Spędzone godziny: estymujemy na łącznie około 120

Wykaz lini kodu wygenerowany przez narzędzie cloc

Language	Files	Blank	Comment	Code	Total
Rust	9	553	291	2864	3708

Napotkane problemy

Początkowo motywem był brak odpowiedniego zarządzania pamięcią i myślenie w uprzednio wyuczonym schemacie. Reguły Rustowego kompilatora wymagają od programisty mądrego planowania struktury zależności w swoim kodzie i z tym borykaliśmy się najczęściej. Podjęte decyzje architekturalne okazywały się czasami błędne, w związku z czym refaktoring dużych części kodu i przerzucanie odpowiedzialności między modułami były zjawiskiem raczej częstszym niż rzadszym. Wraz z biegiem projektu problemy jednak zaczęły pojawiać się coraz rzadziej i pracowało się nad nim coraz płynniej, więc zdecydowanie uczyniliśmy postępy.

Ostatnim błędem, o którym warto wspomnieć, jest wyznaczenie sobie mało realistycznych wymagań dotyczących funkcjonalności projektu. Byliśmy podekscytowani, aby nad nim popracować i mieliśmy dużo ciekawych pomysłów i inspiracji z używanych na codzień narzędzi, jednak w tak obłożonym semestrze musieliśmy zrezygnować z implementacji kilku funkcjonalności, takich jak find and replace, swapfile, fuzzy search oraz nawigacji katalogów w formie drzewa, co prawdopodobnie okazałoby się dobrym tematem projektu na ZPR samo w sobie.

Wnioski

Projekt ten był świetną okazją, żeby zaznajomić się z Rustem i poznać zarówno jego rozległe możliwości, jak i natknąć się na wiele przeszkód. Zabranie się za coś tak sporego bez wcześniejszego styku z tym językiem okazało się dość dużym wyzwaniem, jednak sprawiło nam zdecydowanie wiele frajdy. Sam w sobie Rust okazał się bardzo przyjemnym do pracy językiem i zgodziliśmy się, że prędzej sięgniemy po niego, niż po C++ w prawie każdym przypadku. Mimo, że zarządzanie pamięcią wymagało początkowo delikatnej zmiany paradygmatu w porównaniu z innymi językami, z którymi zdarzyło się nam pracować, z pewnością uniknęliśmy w ten sposób wielu problemów, które objawiłyby się kilkaset linii kodu dalej. Ogromną pomocą była obszerna dokumentacja, przede wszystkim książka `The Rust Programming Language` i `docs.rs`. Wykorzystane przez nas biblioteki, przede wszystkim Ratatui i scribe były bardzo solidnie udokumentowane i przyjemnie się z nimi pracowało.

Mimo niespełnienia części początkowych wymagań uważamy, że dostarczyliśmy solidny i funkcjonalny program z dużymi możliwościami rozszerzenia. Obiecujemy sobie, że w przeciwieństwie do innych projektów semestralnych, które mieliśmy rozszerzać po oddaniu i poszły w niepamięć, z tym będzie inaczej, w końcu mamy już plan dalszej pracy i research przeprowadzony w pierwszej fazie projektu.

Wielką zaletą tego projektu jest to, że nie musi zostać odłożony na półkę po oddaniu, a ten dokument piszę w taki sposób: