Dokumentacja Wstępna

Tymoteusz Malec, Jakub Szweda

Politechnika Warszawska, Cyberbezpieczeństwo 24L

22 Marca 2024

Contents

1. Docelowa funkcjonalność projektu

Naszym projektem jest Aplikacja CLI do konwersji formatów znacznikowych. Aplikacja będzie przyjmowała format wejściowy, plik wejściowy, format wyjściowy i nazwę pliku wyjściowego i konwertowała plik wejściowy na docelowy format zapisując go do wyjściowego pliku. Dodatkowo będzie możliwość konwersji na format Pandoc AST by następnie użyć aplikacji Pandoc do dalszej konwersji.

Program będzie składał się z konwertera Github Flavoured Markdown na obiekt reprezentujący Pandoc AST. Następnie konwertery będą brać AST i generować z niego dokument w formacie docelowym. W ten sposób łatwo będzie rozszerzyć funkcjonalność programu o dodatkowe formaty. Dodanie kolejnych formatów będzie polegało na utworzeniu nowej struktury i zaimplementowaniu odpowiedniego traita.

2. Wybrane narzędzia i rozwiązania

W celu realizacji tego projektu wykorzystamy język Rust, aby nie przejmować się bezpieczną dealokacją pamięci. Rust ułatwi nam korzystanie z iteratorów, dla których ten język ma duże wsparcie. Kolejnym argumentem za użyciem Rusta, jest brak potrzeby użycia i testowania kodu dla wielu wersji kodu dla różnych kompilatorów różnych systemów operacyjnych. Kod na różne systemy kompiluje się tym samym kompilatorem - rustc

W projekcie skorzystamy z biblioteki `serde_json` w celu serializacji i deserializacji obiektów. Skorzystamy z niej tylko do serializacji i deserializacji Pandoc AST na format JSON, by można było użyć wygenerowanego AST do dalszej konwersji używając Pandoca oraz by móc deserializować AST uzyskane z aplikacji Pandoc do porównania z AST wygenerowanym przez naszą aplikację. Nie będziemy jednak korzystać z dodatkowych bibliotek do testowania, ponieważ Rust ma wystarczająco dobre mechanizmy wbudowane specjalnie do tego. Również w celu generowania dokumentacji z kodu będziemy korzystać z wbudowanych rozwiązań języka Rust.

3. Github Flavoured Markdown

Format Github Flavoured Markdown ma oficjalną dokumentacje na stronie Github. Zawiera ona ponad 650 przykładów skupiających się na edge-case'ach. W naszych testach będziemy korzystać z tych przykładów by weryfikować poprawność działania programu. Dodatkowo na końcu strony jest omówiony sposób w jaki parsery powinny działać i będziemy realizować naszą wersje według tego opisu. Naszym

celem jest zrealizowanie konwersji w 100% zgodnej z tą dokumentacją. Zakładamy jednak, że jeśli problem okaże się zbyt skomplikowany lub zbyt czasochłonny to możemy pominąć niektóre typy elementów albo rzadkie edge-case.

3.1. Blocks

W dokumentacji Github Flavoured Markdown czytamy, że dokument można postrzegać jako sekwencję wielu bloków. Bloki to elementy strukturalne takie jak nagłówki, tablice, listy. Dzielą się na bloki typu "leaf" - bloki, które nie zawierają już kolejnych bloków oraz bloki typu "container", które mogą zawierać kolejne bloki w sobie. W pierwszej fazie należy parsować dokument linijka po linijce patrząc czy dana linijka jest dalszą częścią poprzednich bloków czy jest linijką kończącą blok i/lub zaczynającą kolejny. Po ukończeniu tej fazy można przystąpić do parsowania zawartości bloków.

3.2. Inline

Inline'y to zawartości bloków. Są to np. tekst, spacja, pogrubienie itp. Z parsowaniem tych elementów powiązane jest wiele zasad, które dokładnie opisane są na wyżej wspomnianej stronie. Szczególnie skomplikowany wydaje się algorytm, który parsuje zagnieżdżone w sobie linki i pogrubienia. Po ukończeniu tego procesu proces parsowania się kończy i otrzymamy gotowe AST.

4. Konwersja na Typst i LaTeX

Spodziewamy się że konwersja z Markdowna na AST będzie o wiele trudniejsza niż konwersja z ustrukturowanego AST na docelowy format, więc pierwsza część zajmie o wiele więcej czasu. W Markdownie ten sam obiekt (nagłówek, lista) można reprezentować na wiele sposobów i często bardzo podobne ciągi znaków używane są do reprezentacji różnych rzeczy zależnie od kontekstu. Posiadając gotowe drzewo będziemy musieli tylko dla każdego z bloków znaleźć sposób na reprezentacje go w docelowym formacie. Konwersja ma zachować wszystkie te same elementy w tej samej kolejości, natomiast podobnie jak w aplikacji Pandoc, rozmiar, położenie, formatowanie może być nieco inne po konwersji co ułatwia problem. Wydaje nam się, że wystarczy przejrzeć wszystkie bloki w drzewie po kolei jeden po drugim i zamieniać je na tekst odpowiedni dla wybranego formatu. Zagnieżdżone struktury nie powinny być problemem.