**Autorzy**:

Dominik Kołodziej

Marek Matys

**Translator języka Markdown na język HTML**

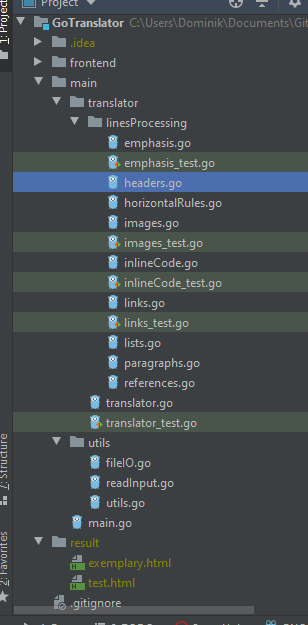
W poniższej dokumentacji wytłuszczoną czcionką są oznaczone nazwy **pakietów,** kursywą są zapisane nazwy *plików*, a z nawiasami napisane są nazwy funkcji().

W projekcie wykorzystany został pakiet **go-sciter** dostarczający większość funkcjonalności aplikacyjnego interfejsu programistycznego (API) Scitera –czyli osadzalnego HTML/CSS/skryptowego silnika do tworzenia nowoczesnych interfejsów użytkownika – do środowiska Golang. Pakiet **go-sciter** dostarcza między innymi takie funkcjonalności jak:

* Wczytywanie plików HTML / kodu HTML jako ciągu znaków (tekstu)
* Manipulacje drzewem DOM / wykonywanie funkcji zwrotnych/obsługę zdarzeń
* Obsługę stanu drzewa DOM/ obsługę atrybutów
* Wczytywanie zasobów niestandardowych
* Sciter Behavior
* Sciter Options
* Sciter Value support
* NativeFunctor (używany w skryptach Scitera)

# **Struktura projektu**

Projekt składa się z 3 głównych katalogów: **frontend**, **main** i results. W pakiecie **frontend** napisany jest front-end aplikacji, w pakiecie **main** właściwy translator, a do folderu results zapisywane są pliki wynikowe (plik HTML przetłumaczony z języka Markdown). Pakiet **main** składa się z 3 elementów: pakietów: **translator** i **utils** oraz pliku *main.go*.



Uruchomienie aplikacji jest równoznaczne z uruchomieniem pliku *main.go*. Jest to, można powiedzieć, swego rodzaju kontener na wszystko. Plik *main.go* składa się z:

* funkcji init() – utworzenie okna programu, czytanie z pliku translatorView.html (katalog frontend), zaczytanie wszystkich przycisków
* funkcji main() – odczytanie pliku za pomocą przycisku „read the file”, przetłumaczenie go z wykorzystaniem funkcji translate() z pakietu translator (patrz niżej ⬇️), zapisanie wyjściowego pliku i ustawienie statusu „Done”
* funkcji closeApplication() – zamyka aplikację
* funkcji setStatus() – ustawia opis statusu tłumaczenia widoczny w oknie aplikacji
* funkcji shouldOpenExplorer() – jeżeli jest zaznaczony checkbox „Open directory” zwraca flagę true, w przeciwnym przypadku – false
* funkcji shouldOpenBrowser() – jeżeli jest zaznaczony checkbox „Open file when finished” zwraca flagę true, w przeciwnym przypadku – false

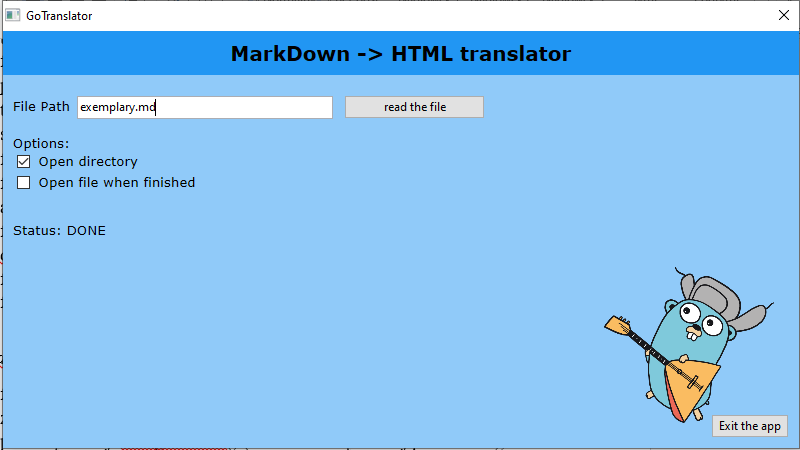
Pakiet **translator** składa się z pakietu **linesProcessing,** oraz plików: *translator.go* i *translator\_test.go.* Plik *translator.go* składa się z:

* funkcji translate() – przyjmująca 2 argumenty - content i fileName - dzieli zawartość pliku .md przekazaną w pierwszym argumencie na linijki za pomocą funkcji stringToLines() (bo takowe się łatwiej przetwarza), następnie wykorzystując pakiet linesProcessing przetwarza przekazane dane i zwraca zawartość pliku .html
* funkcji stringToLines() – dzieli tekst (łańcuch znaków) na linijki
* funkcji linesToString() – odwrotność funkcji stringToLines()
* funkcji wrapInHtml() – wrapuje wszystko co jest w znaczniki html, czyli dorzuca do przetłumaczonej zawartości pliku obowiązkową otoczkę dla plików .html

Pakiet **linesProcessing** odpowiada za wykrywanie poszczególnych elementów języka Markdown i przetłumaczenie tych elementów na język HTML.W skład tego pakietu wchodzą pliki:

* *emphasis.go* – pogrubienia, pochylenia i przekreślenia
* *headers.go* – nagłówki
* *horizontal.go* – poziome linie
* *images.go* – obrazy, ich tytuły i teksty alternatywne
* *inlineCode.go* – teksty zdefiniowane jako kody komputerowe
* *linesProcessing\_test.go* – testy
* *links.go* – odsyłacze (=hiperłącza)
* *lists*.go – listy uporządkowane i listy nieuporządkowane
* *paragraphs.go* – akapity
* *references.go* – referencje (zapis referencji do mapy)

# **Okno graficzne aplikacji**



# **Zasada działania translatora**

Nasz translator działa na zasadzie dopasowywania linii do regexów.

regexLinkClassic := regexp.MustCompile*(*`\[*([*^\[\]()*]*+*)*]\(*([*\w\d.\/:?=#*]*+*)*\)`*)*regexLinkWithTitle := regexp.MustCompile*(*`\[*([*^\[\]()*]*+*)*]\(*([*\w\d.\/\-:?=#*]*+*)*\s"*([*\w\d\s.\/\-:?'=#*]*+*)*"\)`*)*regexLinkReference := regexp.MustCompile*(*`\[*([*^\[\]()*]*+*)*]\[*([*\w\s\d.\/\-\\:?=#*]*+*)*\]`*)*

Powyżej widzimy regexy dla wykrywania linków. Każdy plik z pakietu linesProcessing posiada własne regexy do wykrywania znaczników markdown.

Przeanalizujmy prosty regex dla pogrubień.

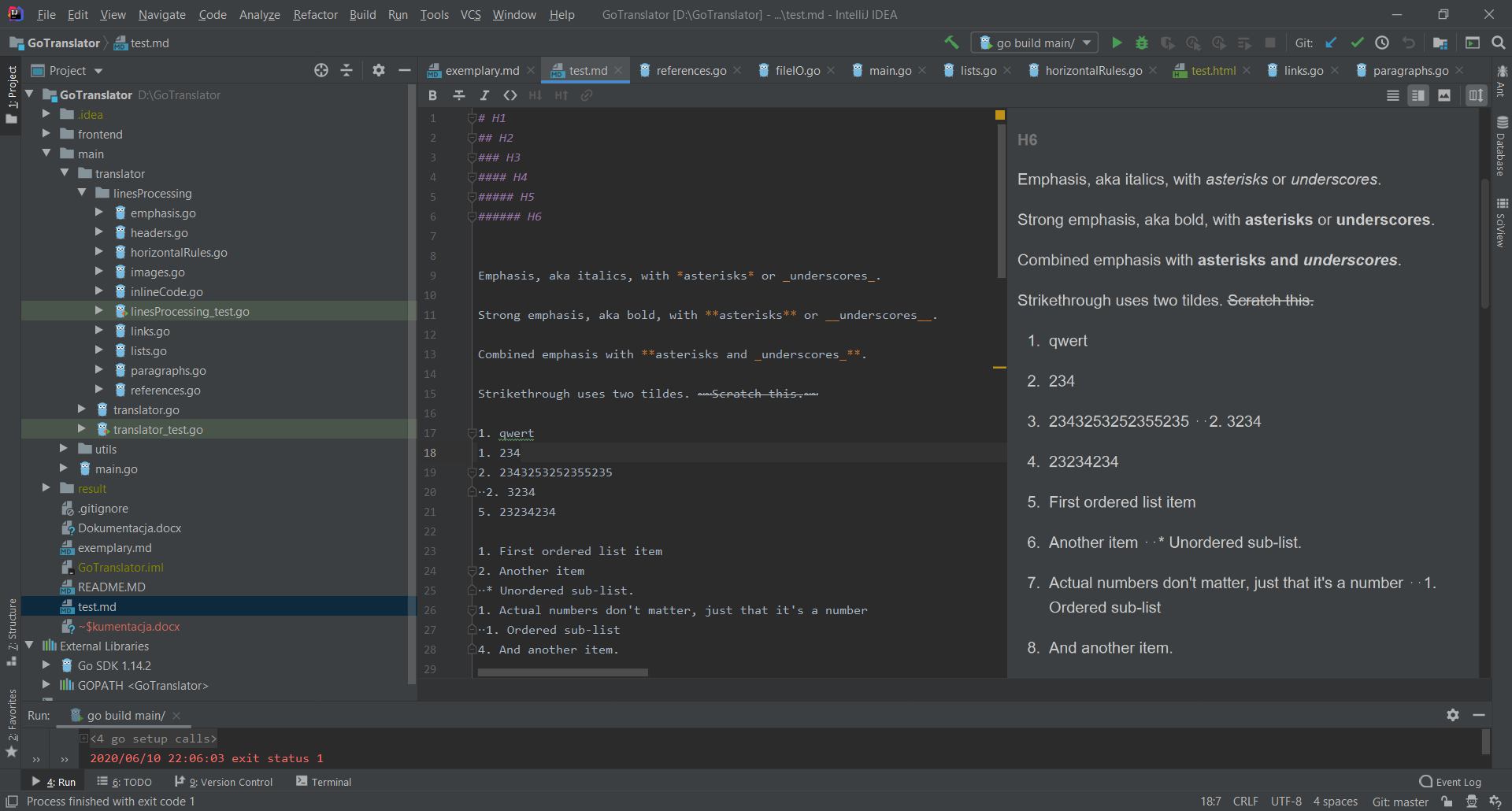
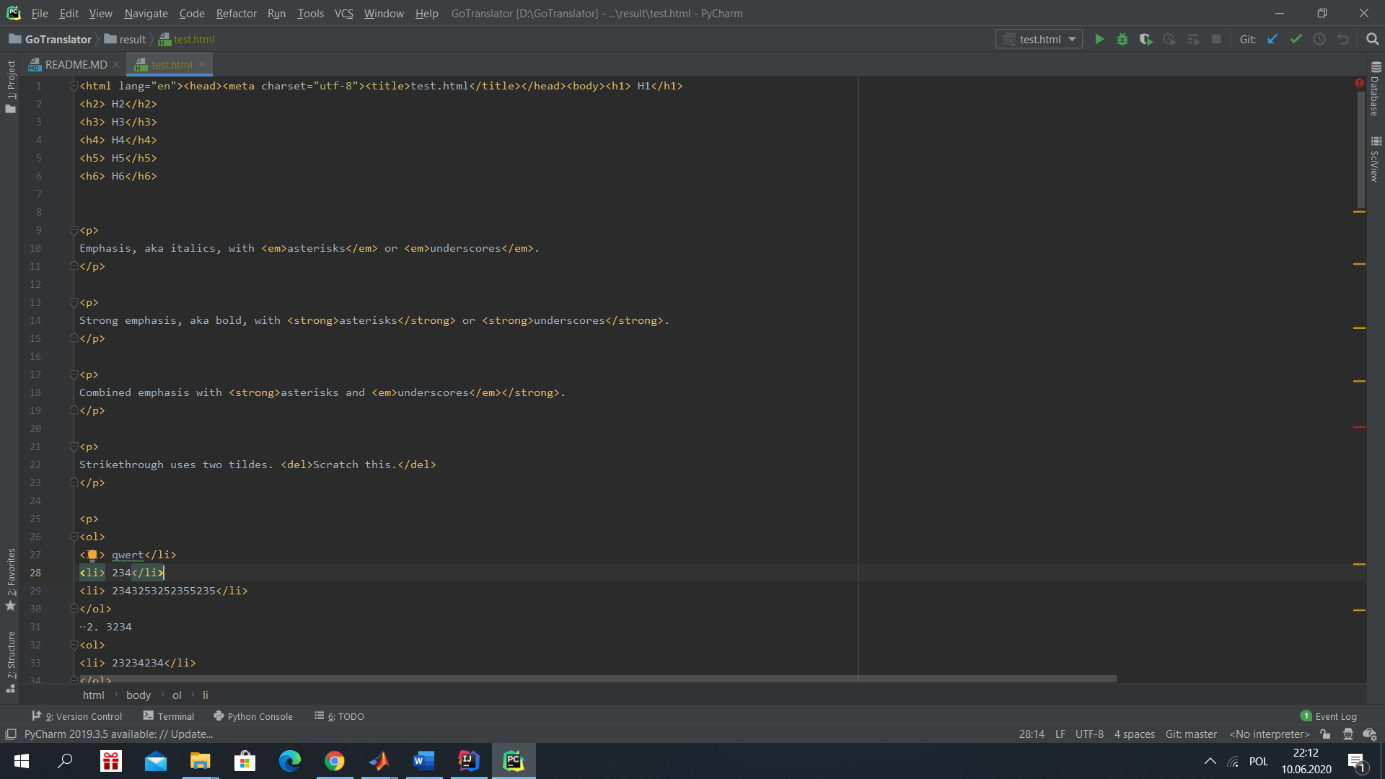
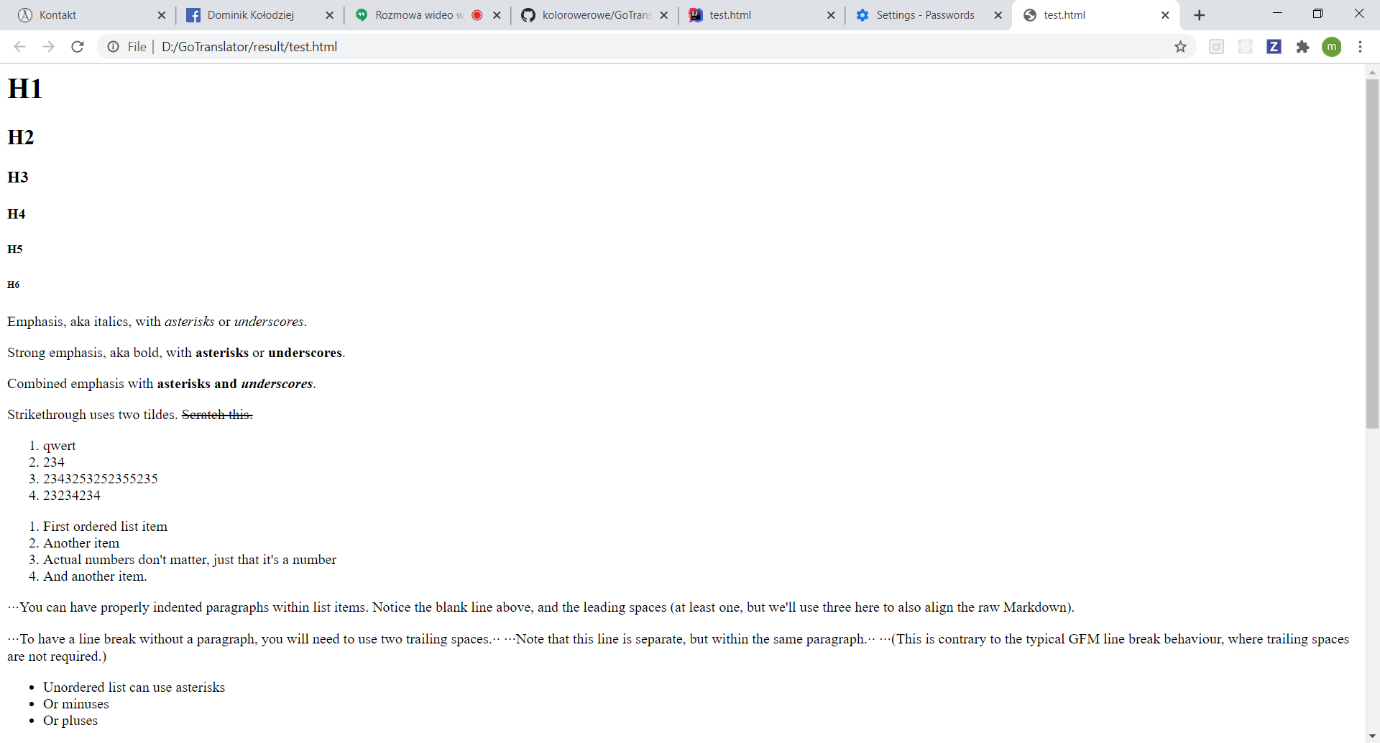
regexStrong := regexp.MustCompile*(*`*(*\_\_.+?\_\_*)*|*(*\\*\\*.+?\\*\\**)*`*)*

Pogrubienia w markdown są oznaczane między dwoma podłogami ( \_\_pogrub\_\_ ) lub pomiędzy dwoma gwiazdkami ( \*\*pogrub\*\* ). Dzięki takiemy wykryciu potrafimy zastąpić dane znaczniki markdown na znaczniki html:

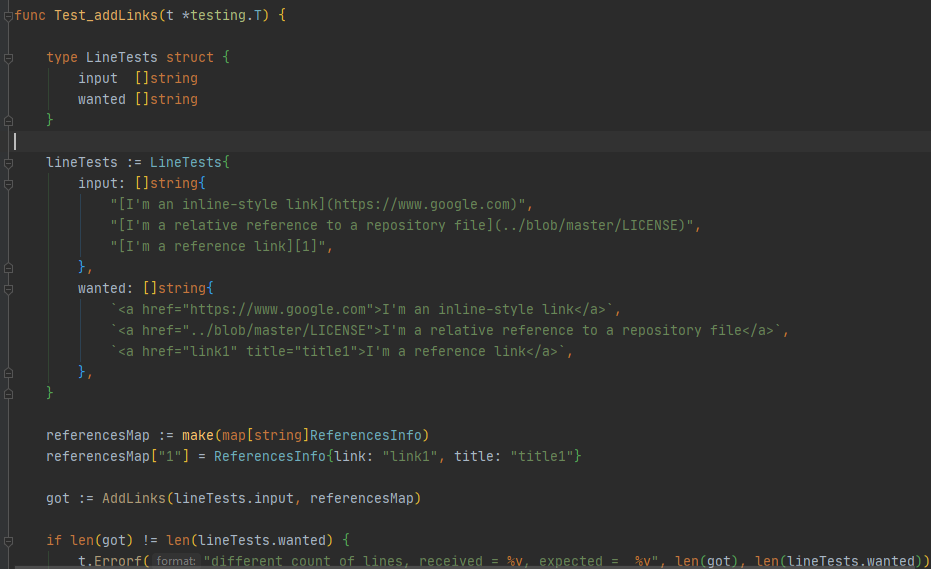
return `<strong>` + utils.CutStringFromBoth*(*text, 2, 2*)* + `</strong>`

# **Działanie translatora**

Poniżej zrzuty ekranu przedstawiające działanie naszego programu: źródłowy plik MD, wyjściowy plik HTML i wyrenderowana strona w przeglądarce:



# **Testy**



*Przykładowy test*

Gdzie „input” oznacza linie w pliku markdown. Przetwarzamy je funkcją AddLinks i chcemy by były zgodne z liniami „wanted”



*Logi z uruchomienia testów*