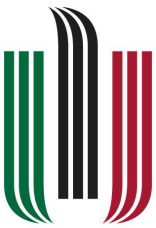
Code Review
Projektu
Lambda Chat
Stan z 21-01-2018
(f7cf3c2863e0e2ac9c25e5722b2a8caaf4cb03e0)

Wojciech Geisler
Paweł Pęczek



AGH
Akademia Górniczo-Hutnicza
im. Stanisława Staszica
w Krakowie

1 Backend

Aplikacja napisana przez Tomka Czajęckiego i Michała Osadnika składa się z dwóch części. Pierwszą, którą poddamy recenzji jest backend, czyli serce ich czatu.

Struktura
Całość backend'u zawiera się w pojedynczym module. Mimo, że jest on tylko jeden nie uważamy, że podział kodu na więcej modułów przyczyniłby się do wzrostu czytelności kodu. Raczej jest to po prostu specyfika projektu.

Uwagi dotyczące kodu
Pierwszym problemem jest brak komentarzy (w aktualnej wersji projektu). Wprawdzie przegląd stanu repozytorium pozwolił stwierdzić, że dokumentacja kodu była istotnie prowadzona, jednak w pewnym momencie została usunięta. Co więcej przegląd usuniętych komentarzy wykazał pewne niezgodności z formatem Haddock.

Nazwa funkcji broadcast nie wskazuje jednoznacznie celu jej istnienia. Być może lepszym pomysłem byłoby nazwać ją broadcastMessage.

Funkcja talk (oraz session):

```
talk :: WS.Connection -> MVar ServerState -> Client -> IO ()
```

wykazuje odstępstwo od przyjętego schematu przekazywania parametrów, które w innych funkcjach wygląda w następujący sposób:

```
someFun :: [...] -> Client -> ServerState -> [...]
```

W funkcji:

```
sendMessageWithVulgarism :: MVar ServerState -> Text -> Text -> IO()
sendMessageWithVulgarism state user msg =
  readMVar state >>= broadcast
(user `mappend`": " `mappend` T.replace "fuck" "f*ck" msg)
```

występuje tylko jeden wulgaryzm do cenzurowania. Wpradzie aplikacja nie ma stanowić gotowego komercyjnego produktu, ale zapewnienie funkcji cenzurowania słów pobranych z listu dowolnej długości wydaje się bardziej przyszłościowe.

W funkcji talk występuje również dość niejasny warunek uznania wiadomości bezpośredniej za niepoprawną:

```
[...] ("direct " `T.isPrefixOf` msg) && (length (words $ T.unpack msg) > 4) [...]
```

Jeśli chodzi o sam moduł Server to w projekcie póki co nie znajdują się testy sprawdzające jego poprawne działanie.

Podsumowanie
Jeśli chodzi o czytelność kodu (nawet mimo braku komentarzy) to stoi ona na bardzo wysokim poziomie. Zagłębienie się w strukturę backend'u nie sprawia problemów, a kod można określić mianem "self descriptive".

2 Frontend

Warstwa dostępowa do aplikacji została wykonana w języku Elm, którego żaden z nas nie zna. Ocena tej części projektu jest więc nieco mniej merytoryczna i bazuje w większości na weryfikacji przestrzegania dobrych praktyk programistycznych.

Assets
Sekcja meta szablonu widoku w HTML powinna zawierać więcej informacji. W tym momencie jest to tylko viewport, kodowanie znaków oraz odnośniki do arkusza .css oraz pliku JS. Oczywiście nie jest to produkt "na rynek", jednak w produkcyjnych aplikacjach web'owych należałoby zwrócić na to uwagę.

Ogólne uwagi dotyczące kodu
Najbardziej istotnym problemem w ocenie tej części pracy jest brak komentarzy. Trzeba jednak przyznać, że większośc kodu ma na tyle czytelną strukturę i nazewnictwo by nie były one koniecnze.

Pozytywne akcenty
Prezentacja treści po stronie przeglądarki wykonywana w klasyczny sposób w porównaniu do rozwiązania zaproponowanego przez twórców projektu wydaje się być strasznie nieefektywna. Łatwość z jaką można w ten sposób uruchomić frontend aplikacji jest wręcz niebywała, a możliwość obejrzenia projektu wykonanego w ten sposób otwiera oczy na zupełnie nowe perspektywy. Bardzo pozytywnie oceniamy zastosowanie języka Elm do prezentacji treści. Jakkolwiek na to nie spojrzeć jest to bardzo interesujący kierunek, w którym rozwija się programowanie funkcyjne.