PureScript Test Task

Jonn Mostovoy
jonn.mostovoy@iohk.io

November 2016

1 Setting

In this test task we will be working with types that capture the concept time tracking and reporting on time spent on issues. Turns out that time tracking can be captured with types that inhabit the following multiparameter typeclasses:

```
class Track e t b a where
    extract
        :: (Foldable b, Monoid (t (b a)), Traversable t)
        => e -> t (b a) -> t (b a)

class Report r t d  where
    produce
        :: (Monoid (t d), Traversable t)
        => (t d -> r)
```

First typeclass captures the idea of tracking events of type e, putting those into a universe of activity categories t with payload data a embedded into a foldable structure b and merging it with *mappend* with the previously tracked events. For example, if we receive a feed of events from a VoIP meeting software about which participant starts speaking at a particular time — $(Text, POSIXTime)$, and we want to record time periods for each meeting separately into a structure $[(Text, NominalDiffTime)]$, we could define an instance of this typeclass and provide implementation for *extract* that works for this use case:

```
instance Track [(Text, POSIXTime)] [] [] (Text, NominalDiffTime)
  where
    extract xs ds = ...
```

An example of using such construction would be the following:

```
> let x = [("Jonn",1479081309s),("Charles",1479081429s),
           ("Arseniy",1479081666s),("Jonn",1479082000s),
           ("George",1479083307s),("Meeting Ended",1479085307s)]
> extract x ([] :: [[(Text, NominalDiffTime)]])
```

```
[[("George",2000s),("Jonn",1307s),("Arseniy",334s),
  ("Charles",237s),("Jonn",120s)]]
```

Second typeclass captures the idea of producing reports out of the tracked events. For instance, if we want to produce report on who have spoken how much during the meetings, we can model the report with type $Map\ Text\ NominalDiffTime$ and implement report generation by defining the following instance:

```
type Tau = NominalDiffTime
instance Report (Map Text Tau) [] (Text, Tau) where
    produce xs = fromListWith (+) xs
```

Let's see how to use *produce*, continuing our example:

```
> let y = extract x ([] :: [[(Text, NominalDiffTime)]])
> (produce . (concatMap id)) y
fromList [("Arseniy",334s),("Charles",237s),("George",2000s),
          ("Jonn",1427s)]
```

2 Basic Task

Using 'aeson', 'servant', 'argonaut', and 'purescript-bridge' implement a program that displays values of concrete tracking and reporting types in browser.
Concrete tracking type is $[[(Text, NominalDiffTime)]]$.
Concrete reporting type is $Map\ Text\ NominalDiffTime$.
How your approach would have been different if you would have had to work with a sum type? A record?

3 Advanced Task

Using the same set of libraries, bridge as many members of the typeclasses defined in the first section as possible from Haskell types to PureScript types and provide a way to display values of those types in browser.

4 Relevant Code

Relevant code can be found here — `https://github.com/input-output-hk/purescript-test-task` in the `src/Sample.hs` file. NB! Don't fork this repository while working on your submission.