Tutorial: Microservices in Haskell

Alexander Abushkevich

March 2015

REST and DB access

alexanderaa.github.io/haskell-microservices

Agenda

- Microservices - definition and introductory notes
- Quick overview - request/response cycle and associated type conversions
- Focus on a variety of libraries, which help to (de)serialize JSON/XML/. . .

Agenda

- Microservices - definition and introductory notes
- Quick overview - request/response cycle and associated type conversions
- Focus on a variety of libraries, which help to (de)serialize JSON/XML/...

- Focus on a variety of libraries, which help to (de)serialize DB data
- REST in Haskell
- Putting all pieces together - compile and run the resulting microservice

Microservices - definition

```
    val range f t =
        let
            fun range' acc f t =
                if (f >= t)
                    then acc
                    else (range' (f :: acc) (f+1) t)
        in
            range' [] f t
        end
```

Yesod

Applicative functors

```
Prelude> import Control.Applicative
Prelude Control.Applicative> :t (<$>)
(<$>) :: Functor f => (a -> b) -> f a -> f b
```

Declare data types

Message, which contains text s

```haskell
data MId           = MId Integer                deriving (I
data EmailAddress  = EmailAddress String        deriving (I
data MText         = MText String               deriving (I
data MStatus = MNew
             | MRead
             | MDeleted
                 deriving (Eq, Ord, Show)

data Message = Message { mId      :: MId
                       , mText    :: MessageText
                       , mStatus  :: MessageStatus
                       , mDated   :: UTCTime }
                           deriving  (Eq, Ord, Show, Typea
```

Data.Aeson

- “Aeson is a fast Haskell library for working with JSON data”

Data.Aeson

- “Aeson is a fast Haskell library for working with JSON data”
- https://hackage.haskell.org/package/aeson

Data.Aeson

- "Aeson is a fast Haskell library for working with JSON data"
- https://hackage.haskell.org/package/aeson

Modules

- Data.Aeson
  - Data.Aeson.Encode <- encode JSON
  - Data.Aeson.Parser <- correctly parse JSON string (UTF)
  - Data.Aeson.TH <- derives ToJSON, FromJSON
  - Data.Aeson.Types <- data types

Data.Aeson

- Provides two typeclasses, ToJSON and FromJSON

Data.Aeson

- Provides two typeclasses, ToJSON and FromJSON
-

Aeson conversions

```haskell
instance AE.FromJSON Message where
    parseJSON (AE.Object v) =
        Message <$> (v AE..: "id"       )
                <*> (v AE..: "message"  )
                <*> (v AE..: "status"   )
                <*> (v AE..: "dated"    )
```

Add simple quickcheck test

```
prop_OptionJSON :: Option -> Bool
prop_OptionJSON o = (((AE.decode . AE.encode) (Just o)) ==
```

PostgreSQL-simple

Questions?

References

http://silkapp.github.io/rest/tutorial.html