Votre mission ? Découvrir Haskell et le mettre en prod

@celine_louvet

fairvio

Contexte et besoin

Le contexte

Début de Fairvioo

- ➔ Faire nos preuves
- ➔ À faible coût

Le besoin

Développement d'une application Web

- ➔ CRUD basique
- ➔ Peu de données au début,
- ➔ Possiblement beaucoup de données à terme

Des contraintes ?

Pas de stack existante

Une seule vraie contrainte : être efficace et viable rapidement et sur le moyen terme

Idéal : prendre du plaisir à développer

Choix de la techno

Totalement subjectif !

Choix de la techno

Choix de la techno : NodeJs ?

Avantages

Pratique pour jeter un POC.

Recrutement facile

Inconvénients

Code souvent peu viable

Interprété

Choix de la techno : Ruby ?

Avantages

Langage “fonctionnel”

Inconvénients

Interprété

Recrutement parfois compliqué

Choix de la techno : PHP ?

Avantages

Frameworks existants

Recrutement facile

Inconvénients

Interprété

Choix de la techno : Java ?

Avantages

Compilé

Écosystème très riche

Inconvénients

Très verbeux

Complexité pour démarrer la stack

Choix de la techno : Scala ?

Avantages

Compilé

Bénéficie de l’ écosystème de Java

Langage fonctionnel

Inconvénients

Tentation de faire du Java

Choix de la techno : Haskell ?

Avantages

Compilé

Langage fonctionnel

Inconvénients

Compliqué

Recrutement ?

Choix de la techno

NodeJs

Ruby

PHP

Scala

Java

Haskell

Choix de la techno

Envie d’un langage compilé

NodeJs

Ruby

PHP

Scala

Java

Haskell

Choix de la techno

Envie d’un langage compilé

~~NodeJs~~

~~Ruby~~

~~PHP~~

Scala

Java

Haskell

Choix de la techno

Envie d’un langage compilé

Envie de fun

~~NodeJs~~

~~Ruby~~

~~PHP~~

Scala

Java

Haskell

Choix de la techno

Envie d’un langage compilé

Envie de fun

~~NodeJs~~

~~Ruby~~

~~PHP~~

Scala

~~Java~~

Haskell

Choix de la techno

Envie d’un langage compilé

Envie de fun

Envie de découverte

~~NodeJs~~

~~Ruby~~

~~PHP~~

Scala

~~Java~~

Haskell

Choix de la techno

Envie d’un langage compilé

Envie de fun

~~NodeJs~~

~~Ruby~~

~~PHP~~

~~Scala~~

~~Java~~

Haskell

Choix de la techno

Envie d’un langage compilé

Envie de fun

~~NodeJs~~

~~Ruby~~

~~PHP~~

~~Scala~~

~~Java~~

Haskell

C’est parti

De quoi j’ai besoin ?

De quoi j’ai besoin

De quoi j’ai besoin

```haskell
module Resources.Types.FormType where

import Data.Aeson                                (FromJSON, ToJSON, parseJSON, toJSON)
import Data.Aeson.Types                          (typeMismatch)
import Data.String                               (fromString)
import Database.PostgreSQL.Simple.FromField (FromField, fromField)
import Web.Internal.HttpApiData                  (FromHttpApiData, parseUrlPiece)

import qualified Data.Text as T                                (Text, toLower)
import qualified Database.PostgreSQL.Simple.ToField as TF (ToField, toField)


data FormType = InMailForm | LandingForm | AdaptativeForm deriving (Eq, Show)

instance FromJSON FormType where
  parseJSON "InMailForm"      = pure InMailForm
  parseJSON "LandingForm"     = pure LandingForm
  parseJSON "AdaptativeForm"  = pure AdaptativeForm
  parseJSON invalid           = typeMismatch "FormType" invalid
```

De quoi j’ai besoin

Stylish

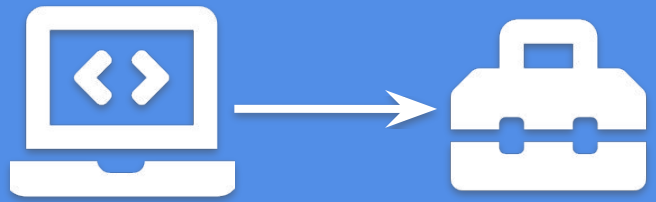
jaspervdj/stylish-haskell

De quoi j’ai besoin

hlint

hackage.haskell.org/package/hlint

De quoi j’ai besoin



De quoi j'ai besoin

Cabal

www.haskell.org/cabal

De quoi j’ai besoin

Stack

www.haskellstack.org

```
dependencies:
- base >= 4.7 && < 5
- aeson
- bytestring
- case-insensitive
- containers
- hslogger
- http-api-data
- http-client
- http-client-tls
- http-types
- mtl
- network-uri
- postgresql-simple
- pwstore-fast
- resource-pool
- scalpel
- scientific
- servant
```

De quoi j’ai besoin

De quoi j'ai besoin

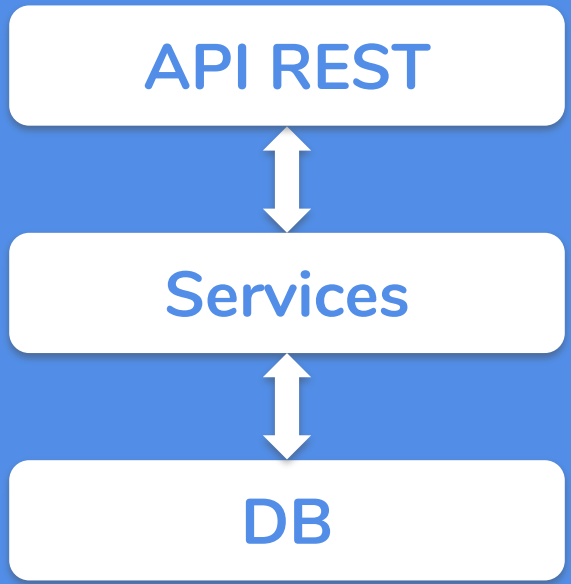
circleci

De quoi j’ai besoin

De quoi j'ai besoin

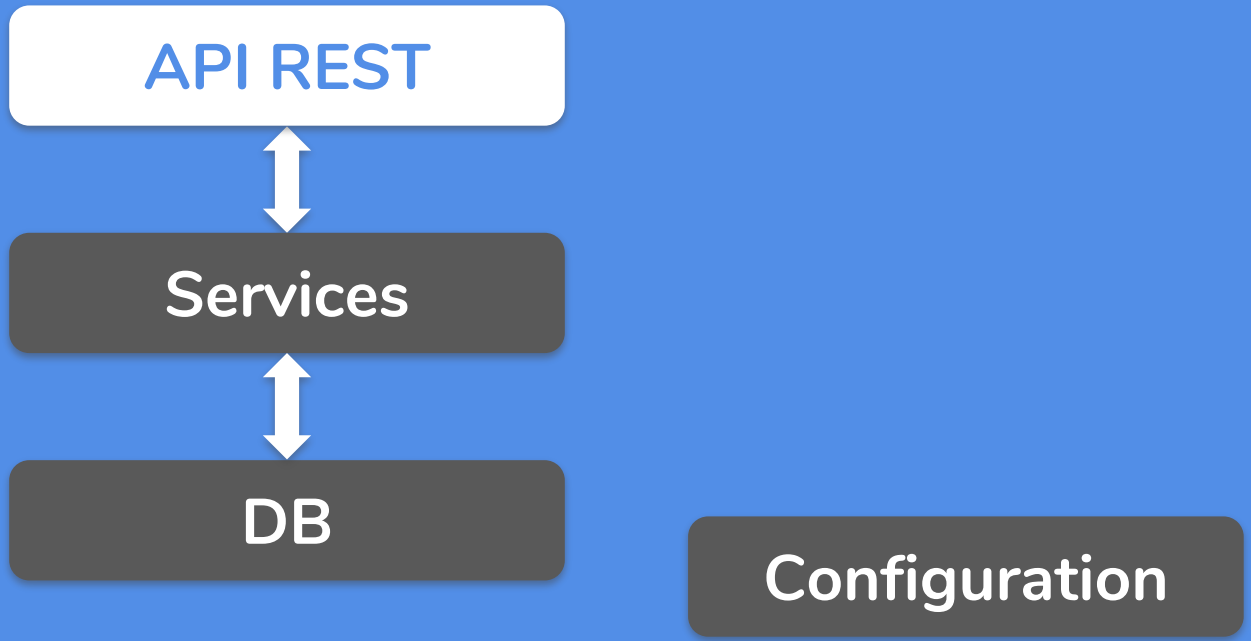
Let’s code

Let’s code



Configuration

Let’s code



API-REST : servant-server

Avantages

Simplicité de description des API

Facilité à mettre en place

API-REST : servant-server

```haskell
-- GET /api/seller
type ListSellerAPI = "seller" :> Get '[JSON] [Seller]


-- POST /api/seller
type CreateSellerAPI = "seller" :> ReqBody '[JSON] Scrap :> Post '[JSON] SellerId


-- GET /api/seller/:id
type GetSellerAPI = "seller" :> Capture "sId" SellerId :> Get '[JSON] Seller
```

```haskell
-- GET /api/seller
type ListSellerAPI = "seller" :> Get '[JSON] [Seller]

listSellers :: Pool Connection -> Handler [Seller]
listSellers = listAllSellers


-- POST /api/seller
type CreateSellerAPI = "seller" :> ReqBody '[JSON] Scrap :> Post '[JSON] SellerId

createSeller :: Pool Connection -> Scrap -> Handler SellerId
createSeller pool scrap = maybeCreateSeller pool scrap >>= maybeNotCreated


-- GET /api/seller/:id
type GetSellerAPI = "seller" :> Capture "sId" SellerId :> Get '[JSON] Seller

getSeller :: Pool Connection -> SellerId -> Handler Seller
getSeller pool sId = maybeGetSeller pool sId >>= maybeNotFound
```

API-REST : servant-server

```haskell
type SellerAPI = ListSellerAPI :<|> CreateSellerAPI :<|> GetSellerAPI

sellerServer :: Pool Connection -> Server SellerAPI
sellerServer pool =
       listSellers pool
  :<|> createSeller pool
  :<|> getSeller pool
```

API-REST : servant-server

Avantages

```
type SellerAdminAPI = SA.Auth '[SA.JWT] SessionUser :> NonProtectedSellerAdminAPI

type NonProtectedSellerAdminAPI = UpdateSellerAPI :<|> DeleteSellerAPI
```

- Facilité à mettre en place (client et serveur)

API-REST : servant-server

Avantage

Simplicité

Facilité à r erveur)

```haskell
-- POST /smtp/email
type WelcomeAPI =
       Header "api-key" ApiKey
  :> "smtp" :> "email"
  :> ReqBody '[JSON] WelcomeEmail
  :> Post '[JSON] Response

data WelcomeEmail = WelcomeEmail
  { templateId :: Int
  , sender     :: Address
  , to         :: [Address]
  , params     :: Params
  } deriving (Show, Generic, ToJSON)
```

API-REST : servant-server

Inconvénients

Messages d’erreur incompréhensibles

```
type ReviewsForSellerAuthAPI = SA.Auth '[SA.JWT] SessionUser :> NonProtectedAuthAPI

type NonProtectedAuthAPI =
       ListReviewsAPI
  :<|> GetReviewAPI
  :<|> GetFullReviewAPI
  :<|> ReadReviewAPI
  :<|> ListAnswerAPI

reviewsForSellerAuthServer :: Pool Connection -> Server ReviewsForSellerAuthAPI
reviewsForSellerAuthServer pool (SAS.Authenticated user) =
       listReviews pool user
  :<|> getReview pool user
  :<|> getFullReview pool user
  :<|> readReview pool user
  :<|> listAnswer pool user
reviewsForSellerAuthServer _ _ = SAS.throwAll err401
```

```
type ReviewsForSellerAuthAPI = SA.Auth '[SA.JWT] SessionUser :> NonProtectedAuthAPI

type NonProtectedAuthAPI =
       ListReviewsAPI
  :<|> GetReviewAPI
  :<|> GetFullReviewAPI
  :<|> ReadReviewAPI
  :<|> ListAnswerAPI
  :<|> CreateAnswerAPI

reviewsForSellerAuthServer :: Pool Connection -> Server ReviewsForSellerAuthAPI
reviewsForSellerAuthServer pool (SAS.Authenticated user) =
       listReviews pool user
  :<|> getReview pool user
  :<|> getFullReview pool user
  :<|> readReview pool user
  :<|> listAnswer pool user
reviewsForSellerAuthServer _ _ = SAS.throwAll err401
```

API-REST : servant-server

```
    • Couldn't match type ‘ReviewId -> Handler [Answer]’
                     with ‘(Data.Text.Internal.Text -> Handler [Answer])
                            :<|> (Data.Text.Internal.Text -> CreationAnswer -> Handler Answer)’
      Expected type: Servant.Server.Internal.ServerT
                       ReviewsForSellerAuthAPI Handler
        Actual type: SAS.AuthResult SessionUser
                     -> Handler [Review]
                        :<|> ((ReviewId -> Handler Review)
                              :<|> ((ReviewId -> Handler FullReview)
                                    :<|> ((ReviewId -> Handler NoContent)
                                          :<|> (ReviewId -> Handler [Answer]))))
    • The equation(s) for ‘reviewsForSellerAuthServer’ have two arguments,
      but its type ‘Pool Connection -> Server ReviewsForSellerAuthAPI’
      has only one
   |
48 | reviewsForSellerAuthServer pool (SAS.Authenticated user) =
   | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^...
```

API-REST : servant-server

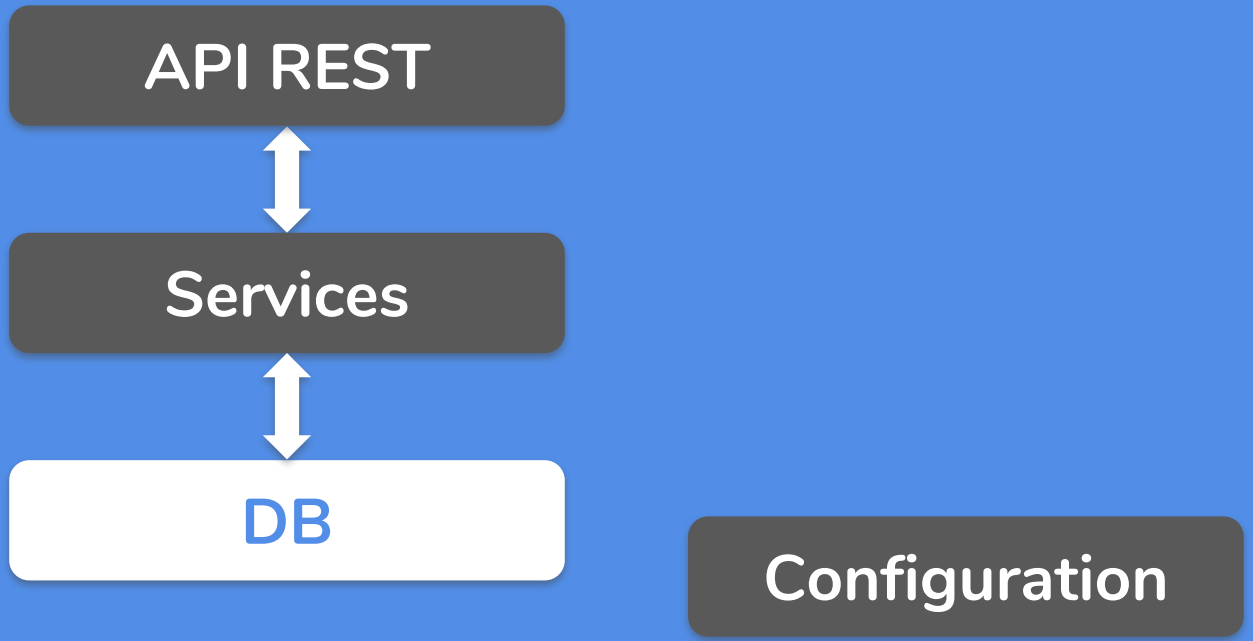
Inco

Mes

```
-- POST /api/auth/seller/review/:id/answer
type CreateAnswerAPI =
       "auth" :> "seller"
  :> "review"
  :> Capture "rId" ReviewId :> "answer"
  :> ReqBody '[JSON] CreationAnswer
  :> PostCreated '[JSON] Answer

createAnswer = undefined
```

Let’s code



DB : postgresql-simple

Avantages

S’apparente à un connecteur JDBC

Équivalent aux PreparedStatements en Java

DB : postgresql-simple

```
insert :: Query
insert = "INSERT INTO USERS (id, lastname, firstname, email, password) VALUES (?, ?, ?, ?, ?)"

fetchById :: Query
fetchById = "SELECT id, lastname, firstname, email FROM USERS WHERE id=?"

update :: Query
update = "UPDATE USERS SET lastname=?, firstname=?, email=?, password=? WHERE id=?"

delete :: Query
delete  = "DELETE FROM USERS WHERE id=?"
```

DB : postgresql-simple

Inconvénients

Les erreurs SQL ne sont levées qu'au Runtime

Un code pas très explicite

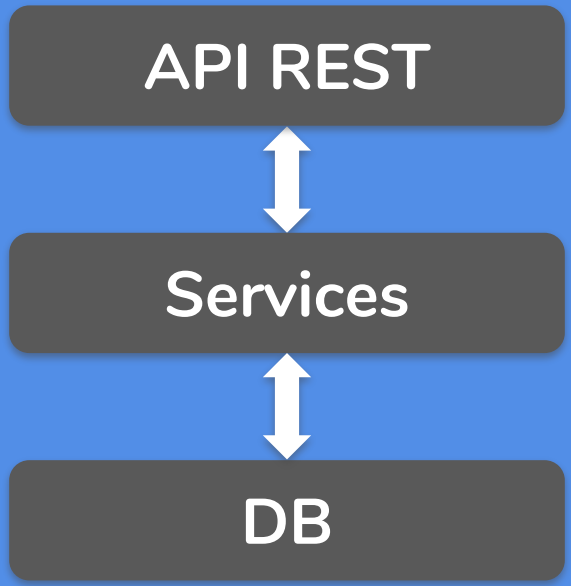
DB : postgresql-simple

```haskell
listAll :: FromRow a => Query -> Pool Connection -> IO [a]
listAll sqlQuery pool = withResource pool performSelect
  where
    performSelect connection = query_ connection sqlQuery

listById :: FromRow a => Query -> Pool Connection -> Text -> IO [a]
listById sqlQuery pool entityId = withResource pool performSelect
  where
    performSelect connection = query connection sqlQuery (Only entityId)
```

Runtime

Let’s code



Configuration

Configuration

Lecture des variables d’environnement fournie nativement

Configuration

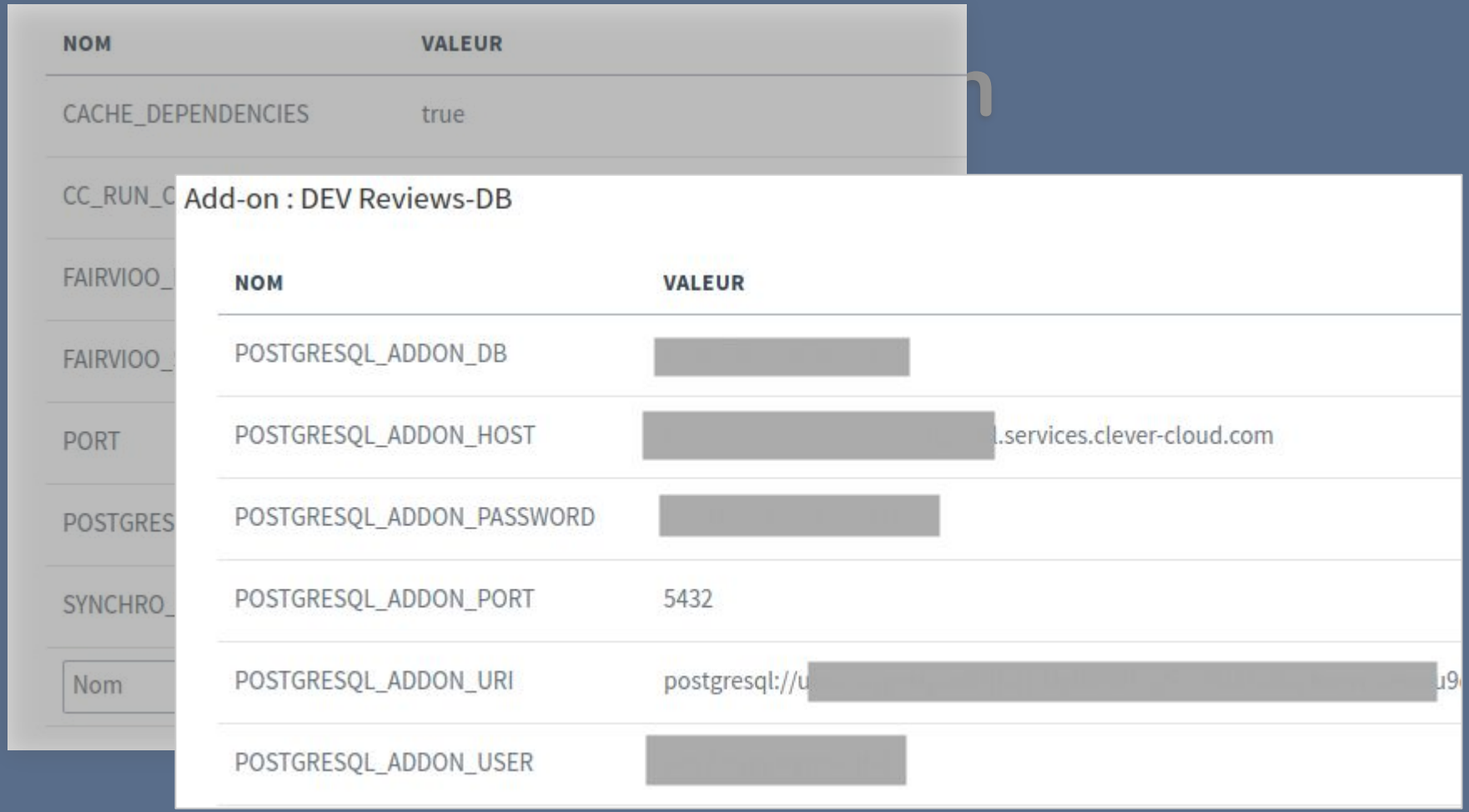
```
readConnectInfo :: IO ConnectInfo
readConnectInfo = ConnectInfo
    <$> lookupEnvVar "POSTGRESQL_ADDON_HOST" connectHost
    <*> (read <$> lookupEnvVar "POSTGRESQL_ADDON_PORT" (show . connectPort))
    <*> lookupEnvVar "POSTGRESQL_ADDON_USER" connectUser
    <*> lookupEnvVar "POSTGRESQL_ADDON_PASSWORD" connectPassword
    <*> lookupEnvVar "POSTGRESQL_ADDON_DB" connectDatabase

lookupEnvVar :: String -> (ConnectInfo -> String) -> IO String
lookupEnvVar name defaultAccessor = fromMaybe defaultValue <$> lookupEnv name
  where
    defaultValue = defaultAccessor defaultConnectInfo
```

Configuration

Fourni nativement par Haskell

```
source workspace/fairvioo-reviews-server/scripts/localhost.sh
  && PORT=8080 POSTGRESQL_ADDON_USER="${PG_USER}" POSTGRESQL_ADDON_DB="${PG_DB}"
  && fairvioo-review-server-exe
```



NOM	VALEUR
CACHE_DEPENDENCIES	true
CC_RUN_C	
FAIRVIOO_	
FAIRVIOO_	
PORT	
POSTGRES	
SYNCHRO_	
Nom	

Add-on : DEV Reviews-DB

NOM	VALEUR
POSTGRESQL_ADDON_DB	[redacted]
POSTGRESQL_ADDON_HOST	[redacted].services.clever-cloud.com
POSTGRESQL_ADDON_PASSWORD	[redacted]
POSTGRESQL_ADDON_PORT	5432
POSTGRESQL_ADDON_URI	postgresql://u[redacted]
POSTGRESQL_ADDON_USER	[redacted]

Et les tests ?

Et les tests ?

HSpec : équivalent à Chai.js (Should)

HUnit : équivalent à JUnit

QuickCheck : générateur de tests

Et les tests ?

HSpec : équivalent à Chai.js (Should)

HUnit

QuickC

```
isSubscriptionComplete :: User -> Bool
isSubscriptionComplete user = state == Complete
  where
    state = subscriptionState $ userToFullUser user
```

Et les tests ?

```haskell
main :: IO ()
main = hspec $ do
  describe "Resources.Types.User.isSubscriptionComplete" $ do
    it "returns false if subscription is None" $ do
      let user = userWithSubscription None
      isSubscriptionComplete user `shouldBe` False

    it "returns false if subscription is Sent" $ do
      let user = userWithSubscription Sent
      isSubscriptionComplete user `shouldBe` False

    it "returns true if subscription is Complete" $ do
      let user = userWithSubscription Complete
      isSubscriptionComplete user `shouldBe` True
```

Et si on mettais en Prod ?

Et si on mettait en Prod ?

Nouvelle instance sur Clever (PostgreSQL & Haskell)

Redirection DNS

➔ **Push Git via Pull Request**

Et les performances ?

Des tutos à conseiller

Des tutos à conseiller

Vidéos de Frédéric Menou :

www.youtube.com/user/fredericmenou

Les points durs

Les points durs - Haskell

Énorme coût d’entrée

Les points durs - Haskell

Énorme coût d’entrée

N’est pas un langage orienté objet

Les points durs - Haskell

Énorme coût d’entrée

N’est pas un langage orienté objet

Complexité à écrire un code simple à lire

Les points durs - Haskell

Énorme coût d'entrée

N'é

Co

```haskell
createSeller :: Pool Connection -> Scrap -> Handler SellerId
createSeller pool scrap = do
  maybeCreated <- maybeCreateSeller pool scrap
  maybeNotCreated maybeCreated
```

Les points durs - Haskell

Énorme coût d'entrée

```
createSeller :: Pool Connection -> Scrap -> Handler SellerId
createSeller pool scrap = maybeCreateSeller pool scrap >>= maybeNotCreated
```

Complexité à écrire un code simple à lire

Les points durs - Haskell

Énorme coût d’entrée

N’es

```haskell
createSeller :: Scrap -> PoolHandler SellerId
createSeller = maybeCreateSeller >=> maybeNotCreated
```

Complexité à écrire un code simple à lire

Les points durs - Documentation

Complexe à comprendre

Les points durs - Documentation

Complexe à comprendre

Parfois difficile à trouver

Les points durs - Authentification

Via Servant

première version avec BasicAuth

module expérimental pour JWT

Les points positifs

Les points positifs

Programmation fonctionnelle pure

Les points positifs

Programmation fonctionnelle pure

Compilateur qui aide beaucoup

Les points positifs

Programmation fonctionnelle pure

Compilateur qui aide beaucoup

Code lisible par un non-developpeur

Verdict ?

Difficile de revenir à un autre langage

Questions ?