Haskell web application architecture

Erik Hesselink

August 29, 2013

About me

- Programming since about 8 years old.
- Studied physics and chemistry.
- Worked at small software company.
- Software technology master at Utrecht University.
- Started Silk with 3 others.

About us

- Started in 2009.
- Started with 4 people, now 10.
- Initial funding by one founder.
- Seed investment by Atomico (Skype) and 4 angels (2011).
- Later investment by NEA, Atmico.

About Silk

"Silk is new way to create and consume content."

What does that mean?

Silk - concept

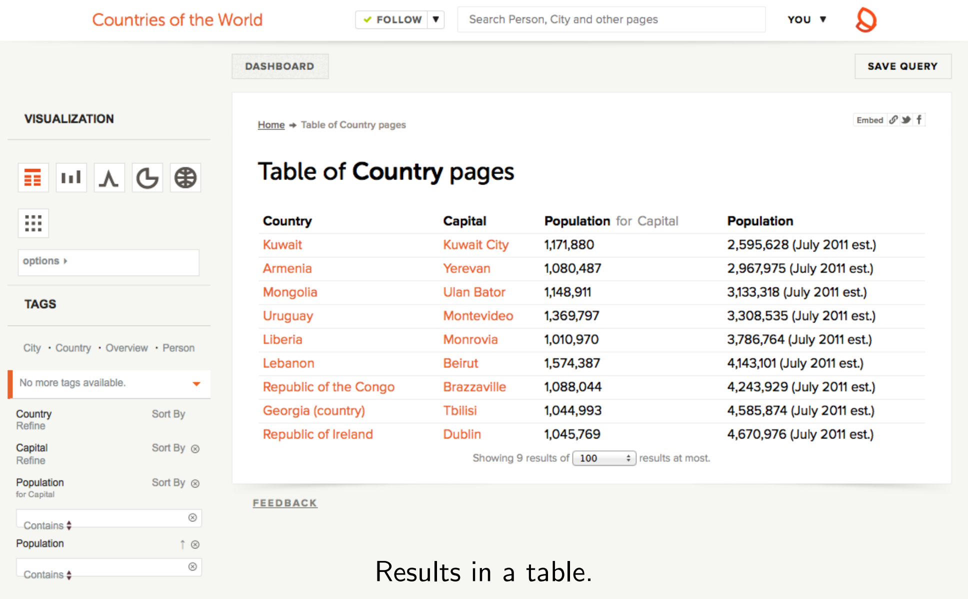
- Site consists of documents (cf. Wikipedia).
- Documents have a name, e.g. ‘Netherlands’.
- Documents contain tags.
    - Value, e.g. tag ‘16,783,092’ as *population*.
    - Link, e.g. tag link to ‘Amsterdam’ as *capital*.
- Tagged links yield a *graph structure*.

Silk - queries

Users can query the total set of information, e.g.:

> *What countries have a population under 5 milion, but a capital with a population over 1 million?*

Results can be shown in different ways: a table, a graph, a map.



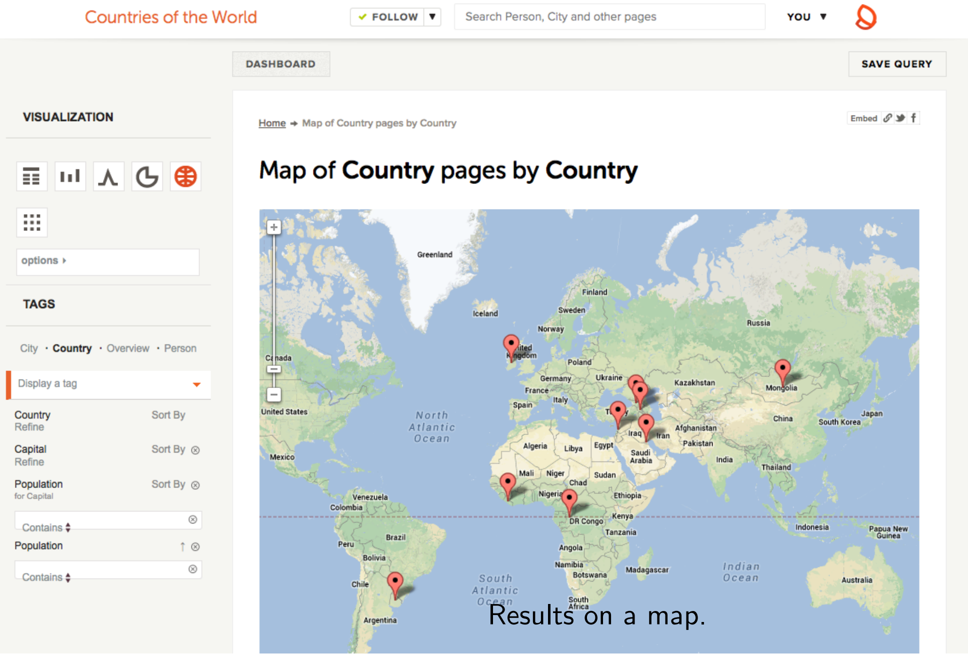
Home → Table of Country pages

Table of Country pages

Country	Capital	Population for Capital	Population
Kuwait	Kuwait City	1,171,880	2,595,628 (July 2011 est.)
Armenia	Yerevan	1,080,487	2,967,975 (July 2011 est.)
Mongolia	Ulan Bator	1,148,911	3,133,318 (July 2011 est.)
Uruguay	Montevideo	1,369,797	3,308,535 (July 2011 est.)
Liberia	Monrovia	1,010,970	3,786,764 (July 2011 est.)
Lebanon	Beirut	1,574,387	4,143,101 (July 2011 est.)
Republic of the Congo	Brazzaville	1,088,044	4,243,929 (July 2011 est.)
Georgia (country)	Tbilisi	1,044,993	4,585,874 (July 2011 est.)
Republic of Ireland	Dublin	1,045,769	4,670,976 (July 2011 est.)

Showing 9 results of 100 results at most.

Results in a table.



Countries of the World

✓ FOLLOW ▼

Search Person, City and other pages

YOU ▼

DASHBOARD

SAVE QUERY

VISUALIZATION

options ›

TAGS

City · **Country** · Overview · Person

Display a tag

Country Sort By
Refine

Capital Sort By ⊗
Refine

Population Sort By ⊗
for Capital

Contains ⊗

Population ↑ ⊗

Contains ⊗

Home → Map of Country pages by Country

Embed

Map of Country pages by Country

Results on a map.

Editor

- Users can create their own content.
- Inline WYSIWIG editor.
- Tagging made easy.
  - Suggestions.
  - Instant gratification.
- Can embed components, like query results.
- Charts and maps can be embedded externally.

Architecture

- Web-facing server (Haskell) with HTTP (REST) interface.
- Talks to other servers (Haskell) through HTTP.
- Fat client (Javascript).
- Talks to server API through AJAX.
- Web site (Haskell) also uses API.

Architecture - diagram

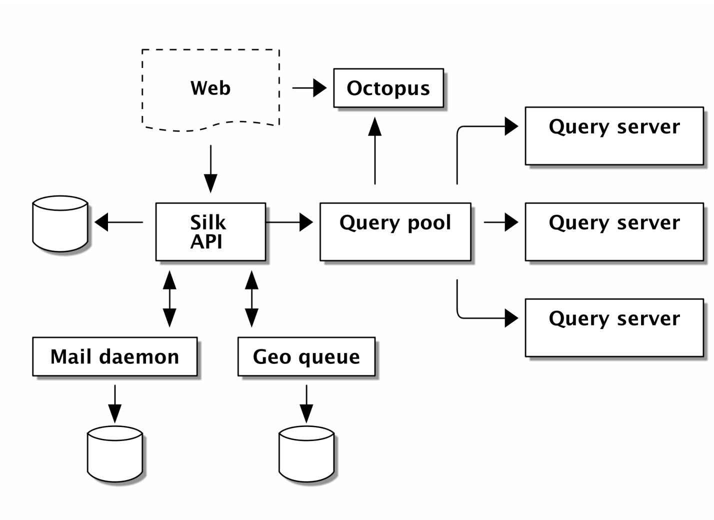


Figure: Silk architecture

Server-side technology

- Documents stored on Amazon S3.
- Relational database storing users, permissions, sessions etc.
- Graph database stores tag structure and processes queries.
- Embeds cached in Varnish.

Architecture lessons

- Split app in separate processes.
  - Easier to see what's going on.
  - Easier to scale.
  - Easier to test.
  - Harder to coordinate.
- Interface with HTTP APIs.
- Seperate machine per process.
- Continuous integration.
- Binary deployment.

Server packages

A service *Foo* results in four packages:

- The server (`foo-server`).
- The domain/API (`foo-api`).
- The client (`foo-client`).
- The shared types (`foo-types`).

Packages: server

The `server` package handles:

- Configuration (command line, file, runtime).
- Starting HTTP server.

Packages: api

The `api` package contains the actual handlers and domain logic.

- Can be run from GHCi.
- Reusable from e.g. command line program.
- Used to generate client libraries, documentation.

Sometimes wrapped in with `server`.

Packages: client

The `client` package communicates with the `server` through HTTP.

- Generated from api description (ideally).
- Doesn't hide all http details.

Packages: types

The `types` package contains the types shared between `server` and `client`.

- Contains all public types.
- Includes needed instances, e.g. (de)serialization.
- Small utility functions (e.g. smart constructors) but no more.

Types and client can be released, server and api are private.

Types of a server

A server uses a newtyped monad transformer:

```haskell
newtype Foo a =
  Foo {unFoo :: ReaderT FooConfig (ServerPartT IO) a}
```

And a simple runner:

```haskell
runFoo :: Config → Foo a → ServerPartT IO a
runFoo cfg foo = runReaderT (unFoo foo) cfg
```

Why a newtype?

Why a newtype? The alternatives:

- Using the literal types.
    - Verbose.
    - Lots of change everywhere.
- Using a type synonym.
    - Cannot have custom type class instances.
    - Less type safety.
- Using type class contexts.
    - Verbose.
    - Lots of change everywhere.
    - But more granularity.

Derive all the things!

With `GeneralizedNewtypeDeriving`

```
newtype Foo a = …
  deriving (Functor, Applicative, Monad
           , MonadIO, ServerMonad, …
           )
```

And occasionally `StandaloneDeriving`:

```
deriving instance Monad (f (Fix f)) ⇒ Monad (Fix f)
```

MonadBaseControl still problematic due to associated type synonym.

Create you own class

Sometimes it's useful to create your own copy of a standard class.

```
class ConfigReader m where
  askConfig   :: m Config
  localConfig :: (Config → Config) → m a → m a
instance ConfigReader Foo where
  askConfig     = Foo ∘ ask     ∘ unFoo
  localConfig f = Foo ∘ local f ∘ unFoo
```

This way you can stack it later with another reader.

Create you own class - 2

Of course this is just the beginning of the boilerplate...

```
instance ConfigReader m ⇒ ConfigReader (ReaderT r m) where
  askConfig     = lift askConfig
  localConfig f = mapReaderT ∘ localConfig
instance ConfigReader m ⇒ ConfigReader (StateT s m) where
  ...
```

API resource

DSL for our REST API describes resources.

```
site :: Resource Root WithSite Site
site = mkResource
  { identifier      = "site"
  , multiGet        = Just listing
  , singleGetBy     = [("uri", byId)]
  , singleUpdateBy  = [("uri", update)]
  , singleDelete    = Just delete
  , singleActions   = [("query", query)
                      ,("wipe",  wipe)
                      ]
  }
```

API artifacts

- Run to get API server.
- Can also generate clients . . .
  - Haskell
  - Javascript
  - Ruby
- . . . and documentation.

API resource - 2

$site :: Resource\ Root\ WithSite\ Site$

- Context the resource runs in (`Root`).
- Context subresources run in (`WithSite`).
- Type the resource describes (`Site`).

API resource - 3

```
site = mkResource
    { identifier    = "site"
    , multiGet      = Just listing
    , singleGetBy   = [("uri", byId)]
   ...
```

- Uris will begin with `site`.
- Listing by GETting `site/`.
- Single item by GETting `site/uri/<uri>`.

API resource - 4

```
...
 , singleUpdateBy = [("uri", update)]
 , singleDelete   = Just delete
 , singleActions  = [("query", query)
                    ,("wipe",  wipe)
                    ]
 }
```

- Update item by PUTting `site/uri/<uri>`.
- Delete item by DELETEing `site/uri/<uri>`.
- Special actions by POSTing to `site/query` and `site/wipe`.

API tree

Combine to create nested resources.

```
silk :: Router
silk = api → user
           → site → page → autosave
                         → version
                  → tag
```

API endpoint

```
byId  :: Handler Root Site
byId  = mkGetter (readId ∘ xmlJsonO) $ λu →
  do repo ← queryRepository u ‘orThrow‘ NotFound
     readableFor (Repo.uri repo)
     return repo
```

- Handler contains *input and output dictionary*.
- Handler action runs in context.
  - `Root` for getters.
  - `WithSite` for actions.
- Inputs and outputs described to capture dictionaries.
  - `Read` for the identifier.
  - XML and JSON serialization for the output.
- Can throw predefined exceptions, or define its own (serializable).

Using the client

```
 {-# LANGUAGE OverloadedStrings #-}
import Silk.Client
import qualified Silk.Client.Site as Site
getSite :: String → String   → IO Site
getSite    username password = run "api.silkapp.com" $
  do signin username password
     Site.byUri "world.silkapp.com"


newtype ApiT m a =
  ApiT {unApiT :: StateT ApiState
    (ReaderT ApiInfo (ResourceT m)) a}
```

Contact

Interested?

- Check out Silk at `http://silkapp.com`.
- Email me at `erik@silkapp.com`.
- Follow us on twitter: @silkapp.

Contact

Interested?

- Check out Silk at `http://silkapp.com`.
- Email me at `erik@silkapp.com`.
- Follow us on twitter: @silkapp.

Questions?

Contact

Interested?

- Check out Silk at `http://silkapp.com`.
- Email me at `erik@silkapp.com`.
- Follow us on twitter: @silkapp.

Thank you.