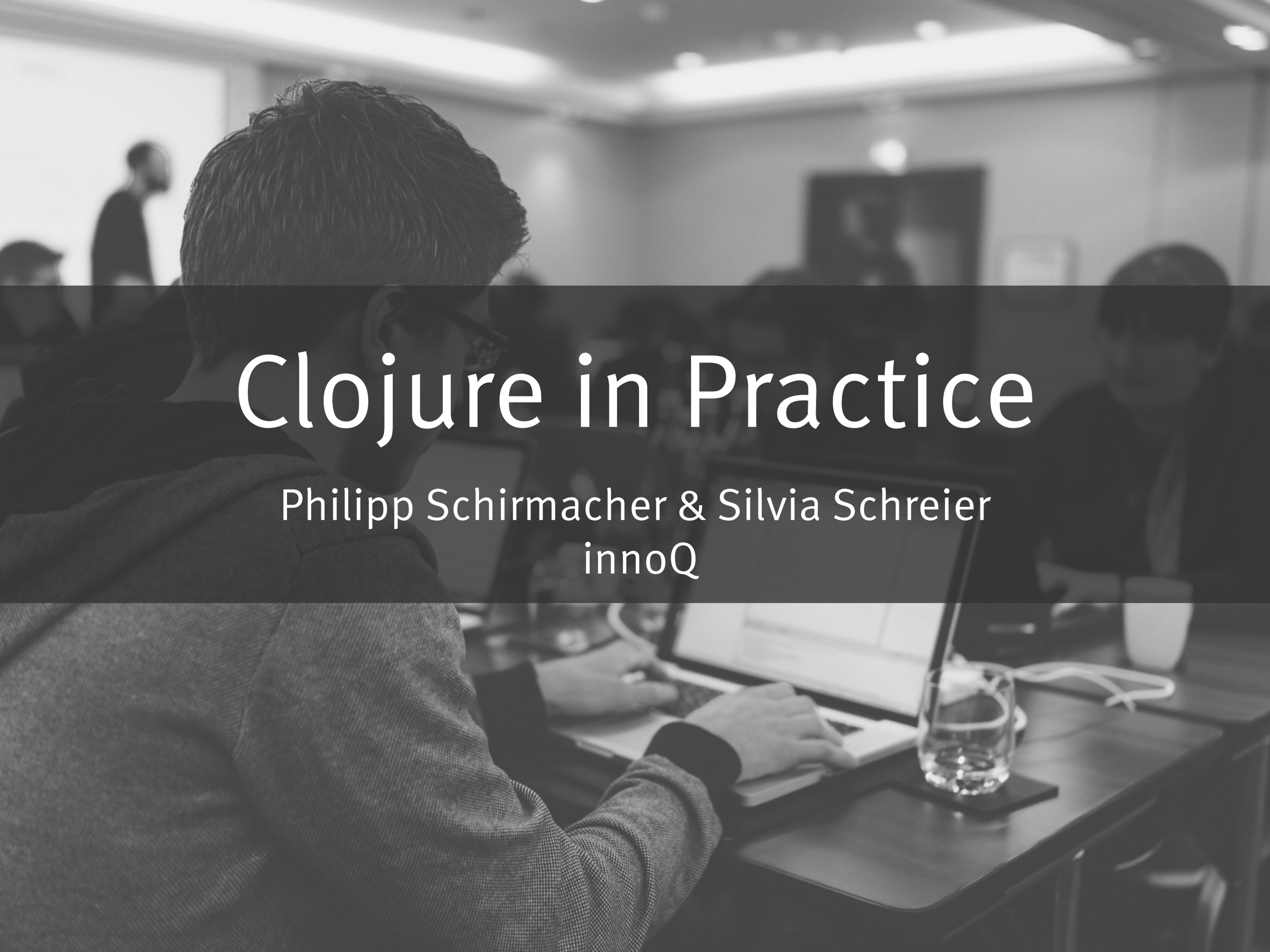


Clojure in Practice

Philipp Schirmacher & Silvia Schreier
innoQ

- How to set up a project?
- How to speak HTTP?
- How to produce HTML?
- How to load & consume HTML?
- How to access a database?
- How to process images?

How to set up a project?

Leiningen

- Alternative to Maven
- Describe Clojure project with generic data structures
- Maven repository compatibility

Leiningen

```
(defproject imagizer "0.1.0-SNAPSHOT"
  :description "yet another clojure demo app"
  :dependencies [[org.clojure/clojure "1.6.0"]
                 [ring "1.3.1"]
                 [compojure "1.1.9"]
                 [hiccup "1.0.5"]
                 [yesql "0.4.0"]
                 [ring/ring-json "0.3.1"]
                 [com.h2database/h2 "1.4.181"]
                 [org.clojure/java.jdbc "0.3.5"]
                 [ragtime "0.3.7"]]
  :plugins [[lein-ring "0.8.12"]
            [lein-cljsbuild "1.0.3"]
            [ragtime/ragtime.lein "0.3.7"]]
  :ring {:handler imagizer.core/webapp})
```

Leiningen

```
$ lein new app imagizer

$ cd imagizer
$ tree
.
├── LICENSE
├── README.md
├── project.clj
├── resources
│   └── public
│       ├── img
│       │   └── icon.svg
│       ├── js
│       │   └── imagizer.js
│       └── stylesheets
│           └── imagizer.css
├── src
│   └── imagizer
│       └── core.clj
└── test
    └── imagizer
        └── core_test.clj
```

Leiningen

```
$ lein help
Leiningen is a tool for working with Clojure projects.

Several tasks are available:
check                Check syntax and warn on reflection.
classpath            Print the classpath of the current project.
clean                Remove all files from project's target-path.
deps                 Download all dependencies.
jar                  Package up all the project's files into a jar file.
ring                 Manage a Ring-based application.
...


$ lein ring server
2014-10-22 10:22:15.237:INFO:oejs.Server:jetty-7.6.13.v20130916
2014-10-22 10:22:15.468:INFO:oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:3000
Started server on port 3000
```

Why Leiningen?

- Easy to extend via plugins
- Plugins can process project definition easily because it's just data

How to speak HTTP?

Ring & Compojure

- Alternative to Spring MVC
- Request & response are data [1]
- Webapp is a function which
  takes a request and
  returns a response

[1] https://github.com/mmcgrana/ring/blob/master/SPEC

Ring & Compojure

```clojure
(def example-request {:uri "/index.html"
                      :request-method :get
                      :headers {"accept" "text/html"}})
```

```clojure
(def example-response {:status 200
                       :headers {"Content-Type" "text/html"}
                       :body "<!DOCTYPE html><html><head>..."})
```

```clojure
(defn hello-world-webapp [req]
  {:status 200
   :body "hello, world!"})
```

Ring & Compojure

```
(defroutes webapp
  (GET "/" [] homepage)
  (GET "/images" [url] (images-page url))
  (GET "/image" [src] (image-page src))
  (POST "/image" [src op] (convert-and-store-image src op))
  (GET "/preview" [src op] (image-preview src op))
  (GET "/result/:uuid" [uuid] (result-page uuid))
  (POST "/result/:uuid/tags" [uuid tag] (add-tag uuid tag))
  (GET "/tags" [] (tags-json))
  (GET "/tags/:tag" [tag] (tag-page tag))
  (GET "/static/:uuid" [uuid] (filtered-file uuid))
  (route/resources "/")
  (route/not-found "oops - not found"))
```

Ring & Compojure

```
(webapp {:request-method :get
         :uri "/"})
> {:status 200
   :headers {"Content-Type" "text/html"}
   :body "<!DOCTYPE html><html>...</html>"}
```

```
(webapp {:request-method :get
         :uri "/unknown"})
> {:status 404
   :headers {"Content-Type" "text/plain"}
   :body "oops - not found"}
```

Ring & Compojure

```
(defproject imagizer "0.1.0-SNAPSHOT"
  :description "yet another clojure demo app"
  :dependencies [[org.clojure/clojure "1.6.0"]
                 [ring "1.3.1"]
                 [compojure "1.1.9"]
                 [hiccup "1.0.5"]
                 [yesql "0.4.0"]
                 [ring/ring-json "0.3.1"]
                 [com.h2database/h2 "1.4.181"]
                 [org.clojure/java.jdbc "0.3.5"]
                 [ragtime "0.3.7"]]
  :plugins [[lein-ring "0.8.12"]
            [lein-cljsbuild "1.0.3"]
            [ragtime/ragtime.lein "0.3.7"]]
  :ring {:handler imagizer.core/webapp})
```

Why Ring & Compojure?

- Simplicity
- Easy to test
- Common request/response format used across libraries

How to produce HTML?

Hiccup

- Alternative to JSPs
- Represent HTML with Clojure data structures
- Transform data structure to HTML string

Hiccup

```
<element/attribute="foo">/>
  <nested>bar</nested>
</element>

[:element]{:attribute "foo"}]
  [:nested "bar"]]
```

Hiccup

```clojure
(def homepage [:html
               [:head [:title "my page"]]
               [:body
                [:h1 "welcome"]
                [:p "just some text"]]])
```

```clojure
(html homepage)
> "<html><head><title>my page</title></head>
   <body>...</body></html>"
```

Hiccup

```
(link-to "http://www.innoq.com" "click here")
> [:a {:href "http://www.innoq.com"} "click here"]
```

```
(form-to [:post "/login"]
         (text-field "Username")
         (password-field "Password")
         (submit-button "Login"))
> [:form {:action "POST" ...}
[:input ...] ...]
```

Hiccup

```clojure
(defn tag-item [tag]
  [:li
   [:a {:href (str "/tags/" tag)} tag]])
```

```clojure
(defn tag-list [tags]
  [:ul.tags (map tag-item tags)])
```

```clojure
(tag-list ["foo" "bar" "baz"])
> [:ul.tags ([:li [:a {:href "/tags/foo"} "foo"]]
  [:li ...] [:li ...])]
```

Why Hiccup?

- Simple, yet powerful
- HTML is just data, so no special language required to manipulate it

What do we have so far?

- Project set up (Leiningen)
- HTTP basics (Ring + Compojure)
- HTML (Hiccup)

What do we have so far?

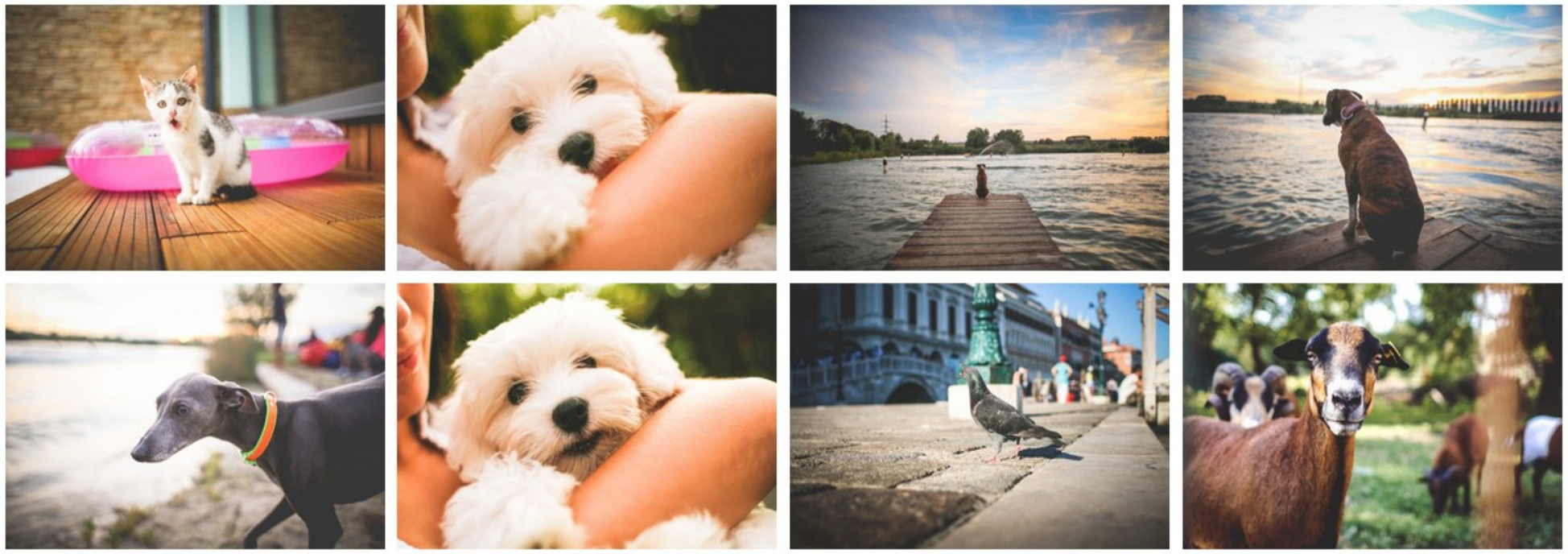
What we want!

Imagizer

search result

http://picjumbo.com/category/animals/

search



How to load & consume HTML?

clj-http & Hickory

- Alternatives to Jersey HTTP Client and jsoup
- Built on Apache HTTP Client
- Ring-compatible request/response format
- Hiccup-compatible HTML representation

clj-http & Hickory

```
(http/get "http://www.google.de")
> {:cookies {}
   :body "<very long string>"
   :trace-redirects ["http://www.google.de"],
   :request-time 151,
   :status 200,
:headers
{"Server" "gws"
 "Content-Type" "text/html; charset=ISO-8859-1"
 "X-Frame-Options" "SAMEORIGIN"
 "Connection" "close"
 "Alternate-Protocol" "80:quic,p=0.01"
 "Expires" "-1"
 "Date" "Wed, 22 Oct 2014 13:59:12 GMT"
 "X-XSS-Protection" "1; mode=block"
 "Cache-Control" "private, max-age=0"}}
```

clj-http & Hickory

```
(http/get "http://some.rest.api"
          {:headers {"Accept" "application/json"}})
> {:status 200
   :headers {"Content-Type" "application/json"}
   :body "{\"json\" : [\"stuff\"]}"}
```

```
(http/get "http://picjumbo.com/wp-content/uploads/HNCK0619-1300x866.jpg"
          {:as :byte-array})
> {:status 200
   :headers {...}
   :body #<byte[] [B@2f1004fd>}
```

clj-http & Hickory

```
(hickory/as-hiccup
  (hickory/parse "<html><head></head><body>foo</body></html>"))
> [:html {}
   [:head {}]
   [:body {} "foo"]]
```

```
(-> "<html><head></head><body>foo</body></html>"
  hickory/parse
  hickory/as-hiccup)
> [:html {}
   [:head {}]
   [:body {} "foo"]]
```

clj-http & Hickory

```
(->
"<html><head></head><body>foo</body></html>"
  hickory/parse
  hickory/as-hiccup)
> [:html {}
   [:head {}]
   [:body {} "foo"]]

(defn load-html [url]
  (-> url
    http/get
    :body
    hickory/parse
    hickory/as-hiccup))
```

clj-http & Hickory

```
(load-html "http://picjumbo.com/category/animals/")
> [:html {:xmlns "http://www.w3.org/1999/xhtml"}
   [:head {}]
   [:body
...lots of stuff...
[:img {:src "<img url>"}]
...more stuff...]]
```

clj-http & Hickory

```
(doc tree-seq)
-------------------------
clojure.core/tree-seq
([branch? children root])
  Returns a lazy sequence of the nodes in a tree, via a depth-first walk.
  branch? must be a fn of one arg that returns true if passed a node
  that can have children (but may not).  children must be a fn of one
  arg that returns a sequence of the children. Will only be called on
  nodes for which branch? returns true. Root is the root node of the
  tree.
```

```
(defn all-elements [hiccup-html]
  (let [might-have-children? vector?
        children (fn [node]
                   (drop 2 node))]
    (tree-seq might-have-children? children hiccup-html)))
```

```
(defn find-images [url]
  (let [html (load-html url)]
    (filter (fn [elem]
              (and (vector? elem) (= :img (first elem))))
            (all-elements html))))
```

Why clj-http & Hickory?

- Well-known request/response format
- Well-known HTML representation
- Once again: easy to process because it's just data

How to access a database?

yesql & ragtime

- yesql: alternative to Spring JdbcTemplate
- Don't build a DSL around a DSL
- Parses SQL queries into Clojure functions
- ragtime: migrating structured data
- Common interface for migrations like Ring for HTTP

yesql & ragtime

src/db/all_image_tags.sql

```
-- name: all-img-tags
-- finds all existing image tags
SELECT DISTINCT Tag FROM Image_Tag;
```

```
(defqueries "db/all_image_tags.sql")
```

```
(all-img-tags db-spec)
> [{:tag "animal"}
   {:tag "blur"}
   {:tag "funny"}]
```

yesql & ragtime

src/db/add_tag.sql

```
-- name: add-tag!
INSERT INTO Image_Tag(Image, Tag) VALUES (:file, :tag);
```

```
(defqueries "db/add_tag.sql")

(add-tag! db-spec uuid tag)
```

yesql & ragtime

migrations/20141010-add-image-tag.up.sql

```
CREATE TABLE Image_Tag (Tag varchar(200), Image varchar(40));
```

migrations/20141010-add-image-tag.down.sql

```
DROP TABLE Image_Tag;
```

yesql & ragtime

project.clj

```
:plugins [[lein-ring "0.8.12"]
          [ragtime/ragtime.lein "0.3.7"]]

:ragtime {:migrations ragtime.sql.files/migrations
          :database "jdbc:h2:./db/data"}
```

```
> lein ragtime migrate
```

Why yesql & ragtime?

- No DSL for a DSL
- Simple
- Easy data processing

How to process images?

Java interop

- Using the rich JVM ecosystem
- Im4java: Java interface for ImageMagick
- It is easy to create objects, access attributes and call methods
- Bean/Java-Object ↔ Map

Java interop

```
(def uuid (java.util.UUID/randomUUID))

uuid
> #uuid "8ce47fde-8c01-4396-8dea-4ec0d0ef88d5"

(.length (.toString uuid))
> 36

(-> uuid
  (.toString)
  (.length))
> 36

(-> uuid
  (.toString)
  (count))
> 36
```

Java interop

```
(ns imagizer.core
  (:import [org.im4java.core Info]))
```

```
(new Info "source.jpg")
(Info. "source.jpg")
```

```
(let [img-info (Info. "source.jpg")]
  (.getImageWidth img-info))
```

Why Java interop?

- Good built-in support
- Utilizing whole ecosystem
- But be careful with mutable state
- Use conversion to map instead

Summary

	Java	Clojure
Configuration	Maven	Leiningen
HTTP server	SpringMVC	Ring + Compojure
HTML templating	JSP	Hiccup
HTTP client	Jersey HTTP Client	clj-http
HTML parsing	Jsoup	Hickory
JSON handling	Jackson	data.json
Database access	Spring JdbcTemplate	yesql
Migrations	Liquibase	ragtime
JVM Interop	-	built-in

There is more to discover!

Thank you! Questions?

Silvia Schreier
silvia.schreier@innoq.com
@aivlis_s

Philipp Schirmacher
philipp.schirmacher@innoq.com
@pschirmacher

innoQ

http://www.innoq.com