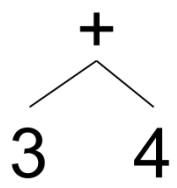
How to write a Clojure based Web Application

This document is meant to be a starting point of web application development with the Clojure language.

The Clojure syntax

Clojure is a dynamic programming language that targets the Java Virtual Machine. Clojure treats code as data and we write it in form of a list. Let's say we want to calculate `(3 + 4)`. What most compilers do is to convert code expressions to an Abstract Syntax Tree (AST). The result of `(3 + 4)` would look like this.



Abstract Syntax Tree (AST)

In Clojure we write that AST directly in form of a list. `(3 + 4)` becomes `(+ 3 4)`. So, if we would transform `((3 + 4) - (5 * 9))` into a Clojure list we would write it that way:
`(- (+ 3 4) (* 5 9))`

Web Application

Let's now create a simple Clojure based Newsletter subscription Web Application. We need a single web page where we present a formula with an email input field.
Before we start we need leiningen. Leiningen is for automating Clojure projects (https://github.com/technomancy/leiningen).

Step 1: Create a project

Switch to your workspace and create a project using leiningen

```
> lein new mybank
```

```
BuffetL:workspace sven$ lein new mybank
Generating a project called mybank based on the 'default' template.
To see other templates (app, lein plugin, etc), try `lein help new`.
BuffetL:workspace sven$
```

I will just create an idea project as well.

Step 2: Add a web server

Inside your newly created project you will find a `project.clj` file where all the dependencies will be declared.

```
BuffetL:mybank sven$ ls -la
total 24
drwxr-xr-x   8 sven  staff  272 May 16 08:30 .
drwxr-xr-x  15 sven  staff  510 May 16 08:30 ..
-rw-r--r--   1 sven  staff  130 May 16 08:30 .gitignore
-rw-r--r--   1 sven  staff  196 May 16 08:30 README.md
drwxr-xr-x   3 sven  staff  102 May 16 08:30 doc
-rw-r--r--   1 sven  staff  266 May 16 08:30 project.clj
drwxr-xr-x   3 sven  staff  102 May 16 08:30 src
drwxr-xr-x   3 sven  staff  102 May 16 08:30 test
BuffetL:mybank sven$ _
```

Here is how `project.clj` looks like after we created it.

```
(defproject mybank "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :license {:name "Eclipse Public License"
            :url "http://www.eclipse.org/legal/epl-v10.html"}
  :dependencies [[org.clojure/clojure "1.5.1"]])
```

Check the version of Clojure if it is the one you want.

Add now *ring* and *compojure* to your project file by adding these two libs to the key `:dependencies`

```
[ring/ring "1.2.0"]
[compojure "1.2.0-SNAPSHOT"]
```

Ring is a Clojure web applications library inspired by Python's WSGI and Ruby's Rack. By abstracting the details of HTTP into a simple, unified API, Ring allows web applications to be constructed of modular components that can be shared among a variety of applications, web servers, and web frameworks. (https://github.com/ring-clojure/ring)

Compojure is a small routing library for Ring that allows web applications to be composed of small, independent parts. (https://github.com/weavejester/compojure)

Leiningen offers a ring-plugin that automates common Ring task like starting a development web server. Add those configurations to your project file.

```
:plugins [[lein-ring "0.8.7"]]
:ring {:handler mybank.core/my_routes
       :auto-reload? true
       :auto-refresh? false}
```

We just declared the `my_routes` as the entry point for our web server in the namespace `core`. Now we have to define `my_routes` in `core.clj`.

That's how `core.clj` looks like before our changes.

```
(ns mybank.core)

(defn foo
  "I don't do a whole lot."
  [x]
  (println x "Hello, World!"))
```

Define `my_routes`. Whenever we request for `/` we call the function `foo`.

```
(defroutes my_routes
  (GET "/" [] (foo "Jim"))
  (route/resources "/"))
```

The `foo` function does not work properly yet because it prints just to the console. We can change that by replacing `println` with `str`.

Here is a simpler version of `foo`.

```
(defn foo [x] (str "Hello, " x))
```

The namespace `route` is still unknown in `core`. We can change that by adding them the key `:use` in `:ns` and tell clojure that we only require `compojure.route`

```
(ns mybank.core
  (:use compojure.core)
  (:require [compojure.route :as route]))
```

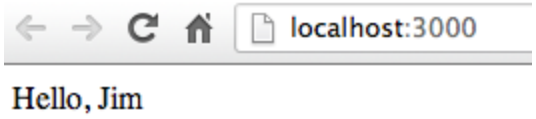
At the last step we have to **resolve our new dependencies** by executing

```
> lein deps
```

and **starting the server**.

```
> lein ring server 3000
```

```
BuffetL:mybank sven$ lein deps
BuffetL:mybank sven$ lein ring server 3000
2013-05-16 09:48:17.728:INFO:oejs.Server:jetty-7.6.1.v20120215
2013-05-16 09:48:17.854:INFO:oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:3000
Started server on port 3000
```



Replace the whole `foo` function call with `(view/index-page)` and

add `[mybank.view :as view]` in `:require`

delete `foo` function

create `mybank.view.clj` file and show it

```
(ns mybank.view)
(defn index-page [] "Bla")
```

update the browser

Include hiccup deps in the project

```
[hiccup "1.0.4"]
```

add this to `:ns` in `view`

```
  (:use hiccup.page hiccup.element)
```

Replace index-page

```
(defn index-page []
  (html5
    [:html
     [:head]
     [:body "bla"]]))
```

```
lein ring server 3000
```

Want to add a REST service? Let's do it.
and `[clojure.data.json :as json]` in `:require` in `core`
add `ring.middleware.params` in `:use` in `core`
add `[org.clojure/data.json "0.2.1"]` in `project`
Add this to `my_routes`

```
(GET "/rest" [] (json/json-str {:email "sven@malvik.de"}))
```

To make it work we have to wrap `my_routes`

```
(def app (wrap-params my_routes))
```

Let's inform the project as well and change.

`:ring {:handler mybank.core/my_routes` to
`:ring {:handler mybank.core/app`

Ok, let the party begin.
Stop the server if it isn't already and call

```
lein deps
lein ring server 3000
```

Everything is ok? Stop the server again and let's move on and deploy our very first clojure webapp.

Stop the server for now.
Now, try `lein run` in the your console and you will get an error saying that you have to declare a main function. `No :main namespace specified in project.clj.`

Let's therefore add the main function in `core.clj`

```
(defn -main [& args] (run-jetty my_routes {:port 8080}))
```

Now we have to tell the project where the main function is located by adding this to the project file `:main mybank.core`

Add `ring.adapter.jetty` to `:use` in `core` as well.

```
(:use compojure.core ring.adapter.jetty)
```

Add `(:gen-class :main true)` to `:ns` in `core.`

Try `lein run`

```
lein clean & lein compile & lein uberjar
java -jar target/mybank-0.1.0-SNAPSHOT-standalone.jar
```

`Vóila!`

Here comes the source:
https://github.com/svenmalvik/SimpleClojureWebApp

For the full code you can visit me on GitHub. The GitHub version also includes HTML templating with hiccup and AngularJS and how to connect to a database:
https://github.com/svenmalvik/ClojureWebAppTemplate

Thank you,
Sven Malvik