# Luminus presentation to Clojure DC August 18th 2015

Author: Paul W. Bible

# Luminus

<http://www.luminusweb.net/docs>

Billed as a “micro”-framework

Collection of Clojure packages

Important packages (that I vaguely understand):

* Ring (Server, other component …)
* Compojure (Application structure/interface, routing)
* Selmer (html template engine)
* Timber (Logging)
* Bouncer (form validation)
* Migratus (database migration)
* Yesql (sql processing, pass keyword maps to queries)
* Buddy (password encryption)

# Create a project

lein new luminus test\_app +postgres

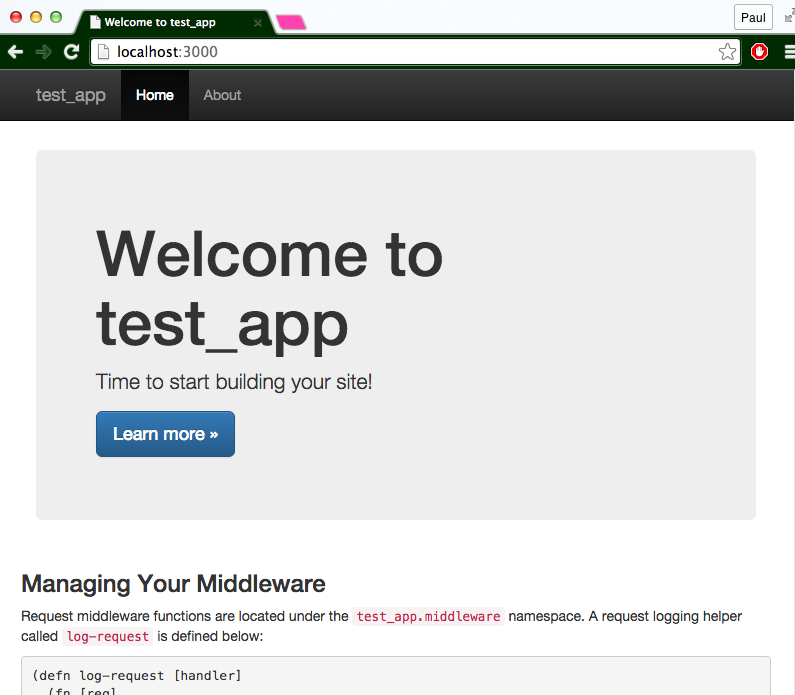
## Test default project:

cd test\_app

lein repl

(start-http-server 3000)

-- Navigate with Browser to localhost:3000 --



# Logging (Timbre)

<https://github.com/ptaoussanis/timbre>

Logging is configured in the application handler

(timbre/merge-config!

{:level (**if** (env :dev) :trace :info)

:appenders {:rotor (rotor/rotor-appender

{:path "test\_app.log" ;; Where your logs are

:max-size (\* 512 1024)

:backlog 10})}})

Call the logger using:

(timbre/info "test\_app is shutting down...")

(timbre/info "shutdown complete!")

# Middleware

Middleware sits between the requests and the application. Lets you inspect the request stream.

Writing a custom middleware function:

(**defn** wrap-logger [handler]

(**fn** [req]

(**do**

(timbre/info (str "target-page: " (str (:path-info req))))

(handler req))))

A middleware wrapper encapsulates the request handler and returns a new request handler.

(defn wrap-myfeature [handler]

(let [new-handler (… initialize …)]

(new-handler req))) ;; Handle the request

# Database Setup

Create a database and application user (postgres)

### Login as admin

psql -U postgres -d postgres -h localhost

### Create a user role for the app

create role testapp login;

\password testapp;

### Create DB/schema and grant privileges to the app role

create schema authorization testapp;

grant all on schema testapp to testapp;

grant all on all tables in schema testapp to testapp;

### Limit app user to the app database schema

Alter user testapp set search\_path = `testapp`;

# Database migrations

Migratus

<https://github.com/yogthos/migratus>

Add migratus to the project.clj file

[migratus "0.8.2"]

Add migratus lein plugin support

:plugins [[lein-environ "1.0.0"]

[migratus-lein "0.1.5"]]

## Configure Migratus

Add to project configuration, tells migratus where to find db

:migratus {:store :database

:migration-dir "migrations"

:migration-table-name "\_migrations"

:db {:classname "org.postgresql.Driver"

:subprotocol "postgresql"

:subname "//localhost/postgres"

:user "testapp"

:password "password"}}

## Create Database scripts (SQL)

Under app\_home/src/module/migrations

Create and up and down SQL scripts.

Requirements:

* Names must have 14 digit prefix.
* \*\*\* Separate multiple SQL statements with “--;;”. (Specific to Migratus)

Users-up ( 00000000000100-users.up.sql ):

**CREATE** **TABLE** users(

user\_id SERIAL **NOT** **NULL** **PRIMARY** **KEY**,

username **VARCHAR**(30) **NOT** **NULL**,

email **VARCHAR**(60),

password **VARCHAR**(256),

about\_me **VARCHAR**(256),

created\_at **TIMESTAMP** **NOT** **NULL** **DEFAULT** (now() AT **TIME** **ZONE** 'utc'),

updated\_at **TIMESTAMP** **NOT** **NULL** **DEFAULT** (now() AT **TIME** **ZONE** 'utc'));

--;;

**CREATE** **OR** REPLACE FUNCTION update\_updated\_at()

RETURNS TRIGGER **AS** $$

**BEGIN**

NEW.updated\_at = now() AT **TIME** **ZONE** 'utc';

RETURN NEW;

**END**

$$ language 'plpgsql';

--;;

**CREATE** TRIGGER update\_user\_updated\_at BEFORE **UPDATE**

**ON** users FOR EACH ROW EXECUTE **PROCEDURE**

update\_updated\_at();

Users-down ( 00000000000100-users.down.sql ):

**DROP** TRIGGER update\_user\_updated\_at **ON** users;

--;;

**DROP** FUNCTION update\_updated\_at();

--;;

**DROP** **TABLE** users;

## Use Migratus

lein migratus migrate Process all “up” scripts

lein migratus up 100 Process the 100 up script

lein migratus down 100 Process the 100 down script

# Database queries (Yesql)

<https://github.com/krisajenkins/yesql>

Under app\_home/src/module/models

## Add yesql query file (users.sql)

pure SQL with special tags “—name:” etc

-- name: insert-user<! 🡨 \*\*\*\* Define query,

-- inserts a new user into the users table

-- Expects :username, :email, and :password

**INSERT** **INTO** users (username, email, password)

**VALUES** (:username, :email, :password)

## Add yesql clojure wrapper

… defqueries macro (Yesql)

(defqueries "app/models/users.sql"

{:connection db-spec})

(**defn** add-user!

"Saves a user to the database"

[user]

(**let** [new-user (**->>** (hashers/encrypt (:password user))

(assoc user :password)

(insert-user<!))] 🡨 \*\*\*\* Call SQL from code with variable

(dissoc new-user :pass)))

# Adding Pages (routes)

Controls what is seen (and done) at *App.domain /feature/request*

## Create the route manager

(**ns** hipstr.routes.albums

(:require [compojure.core :refer :all]

[hipstr.layout :as layout]

[hipstr.models.album-model :as album]))

;; other functions …

(defroutes album-routes

(*GET* "/albums/recently-added" []

(album/get-recently-added))) ;; get-recently-added defined through Yesql

## Add to application handler

(**ns** hipstr.handler

(:require [compojure.core :refer [defroutes routes wrap-routes]]

[hipstr.routes.home :refer [home-routes]]

[hipstr.routes.test-routes :refer [test-routes]]

[hipstr.routes.album :refer [album-routes]] ;; 🡨 added album routes

…

(**def** app

(**->** (routes

(wrap-routes home-routes middleware/wrap-csrf)

album-routes ;; 🡨 album-routes added to application

test-routes

base-routes)

middleware/wrap-base))

## Create html Template (Selmer)

<https://github.com/yogthos/Selmer>

Create a template under: app\_home/resources/templates/albums/recently-added.html

--- recently-added.html ---

{% extends "base.html" %}

{% block content %}

<h1>Recently Added</h1>

<o1 class=*"albums"*>

{% for a in albums %}

<li>

<div class=*"artist"*><a href=*"/albums/{{a.artist}}"*>{{a.artist}}</a></div>

<div class=*"album-name"*>{{a.album\_name}}</div>

<div class=*"release-date"*>{{a.release\_date}}</div>

</li>

{% endfor %}

</o1>

{% endblock %}

---

This is a selmer template.

Boiler plate inherited from “base.html”.

Selmer recognizes the {% expr %} and {{expr}} commands and renders the pages.

## Modify route manager to render with Selmer

Create a function that passes SQL rows to the html renderer

(**defn** recently-added-page

"Renders out the recently added page."

[]

(layout/render "albums/recently-added.html"

{:albums (album/get-recently-added)}))

Then modify the route to use the function

(defroutes album-routes

(*GET* "/albums/recently-added" [] (recently-added-page)))

# Resources

Luminus:

http://www.luminusweb.net/

Changes for Luminus 2.0

<http://yogthos.net/posts/2015-02-28-Luminus-2.0.html>

Book website:

<https://www.packtpub.com/application-development/clojure-web-development-essentials>

Other Book:

<https://pragprog.com/book/dswdcloj/web-development-with-clojure>