Extracted from:

Web Development with Clojure

Build Bulletproof Web Apps with Less Code

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Web Development with Clojure

Build Bulletproof Web Apps with Less Code

Dmitri Sotnikov

edited by Michael Swaine

Web Development with Clojure

Build Bulletproof Web Apps with Less Code

Dmitri Sotnikov

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The team that produced this book includes:

Michael Swaine (editor)
Potomac Indexing, LLC (indexer)
Candace Cunningham (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-937785-64-2
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—January 2014

Setting Up Your Environment

Clojure requires the Java Virtual Machine (JVM) to run, and you will need a working Java Development Kit, version 1.6 or higher.[1] Clojure distribution is provided as a JAR that simply needs to be available on your project's classpath. Clojure applications can be built with the standard Java tools, such as Maven and Ant;[2,3] however, I strongly recommend that you use Leiningen,[4] which is designed specifically for Clojure.

Managing Projects with Leiningen

Leiningen lets you create, build, test, package, and deploy your projects. In other words, it's your one-stop shop for all your project-management-related needs.

Leiningen is the Clojure counterpart of Maven, a popular tool for managing Java dependencies. Leiningen is compatible with Maven, so it has access to large and well-maintained repositories of Java libraries. In addition, Clojure libraries are commonly found in the Clojars repository.[5] This repository is, of course, enabled by default in Leiningen.

With Leiningen, you don't need to worry about manually downloading all the libraries for your project. You can simply specify the top-level dependencies, and they will cause the libraries they depend on to be pulled in automatically.

Installing Leiningen is as simple as downloading the installation script from the official project page and running it.[6]

Let's test this. We'll create a new project by downloading the script and running the following commands:

```
wget https://raw.github.com/technomancy/leiningen/stable/bin/lein
chmod +x lein
mv lein ~/bin
lein new myapp
```

Since we're running lein for the first time, it will need to install itself. Once the install is finished you should see the following output if the command completes successfully:

1. http://www.oracle.com/technetwork/java/javase/downloads/index.html
2. http://maven.apache.org/
3. http://ant.apache.org/
4. http://leiningen.org/
5. https://clojars.org/
6. http://leiningen.org/#install

```
Generating a project called myapp based on the 'default' template.
To see other templates (app, lein plug-in, etc), try `lein help new`.
```

A new folder called myapp has been created, containing a skeleton application. The code for the application can be found in the src folder. There we'll have another folder called myapp containing a single source file named core.clj. This file has the following code inside:

```
(ns myapp.core)

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

Note that the namespace declaration matches the folder structure. Since the core namespace is inside the myapp folder, its name is myapp.core.

What's in the Leiningen Project File

Inside the myapp project folder we have a project.clj file. This file contains the description of our application. With close scrutiny, you'll see that this file is written using standard Clojure syntax and contains the application name, version, URL, license, and dependencies.

```
(defproject myapp "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"]])
```

The project.clj file will allow us to manage many different aspects of our application, as well. For example, we could set the foo function from the myapp.core namespace as the entry point for the application using the :main key:

```
(defproject myapp "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"]]
 ;;this will set foo as the main function
 :main myapp.core/foo)
```

The application can now be run from the command line using lein run. Since the foo function expects an argument, we'll have to pass one in:

```
lein run First
First Hello, World!
```

In the preceding example we created a very simple application that has only a single dependency: the Clojure runtime. If we used this as the base for a web application, then we'd have to write a lot of boilerplate to get it up and running. Let's see how we can use a Leiningen template to create a web-application project with all the boilerplate already set up.

Leiningen Templates

The templates consist of skeleton projects that are instantiated when the name of the template is supplied to the lein script. The templates themselves are simply Clojure projects that use the lein-newnew plug-in.[7] Later on we'll see how we can create such templates ourselves.

For now, we'll use the *compojure-app* template to instantiate our next application.[8] The template name is specified as the argument following the new keyword when running lein, followed by the name of the project. To make a web application instead of the default one as we did a moment ago, we only have to do the following:

```
lein new compojure-app guestbook
```

This will cause Leiningen to use the *compojure-app* template when creating the guestbook application. This type of application needs to start up a web server in order to run. To do that we can run lein ring server instead of lein run.

When we run the application, we'll see the following output in the console and a new browser window will pop up showing the home page.

```
lein ring server
guestbook is starting
2013-07-14 18:21:06.603:INFO:oejs.Server:jetty-7.6.1.v20120215
2013-07-14 18:21:06.639:INFO:oejs.AbstractConnector:
StartedSelectChannelConnector@0.0.0.0:3000
Started server on port 3000
```

Now that we know how to create and run our applications, we'll look at our editor options.

You might have noticed that Clojure code can quickly end up having lots of parentheses. Keeping them balanced by hand would quickly turn into an exercise in frustration. Luckily, Clojure editors will do this for us.

In fact, not only do the editors balance the parentheses, but some are even structurally aware. This means the editor knows where one expression ends

7. https://github.com/Raynes/lein-newnew
8. https://github.com/yogthos/compojure-template

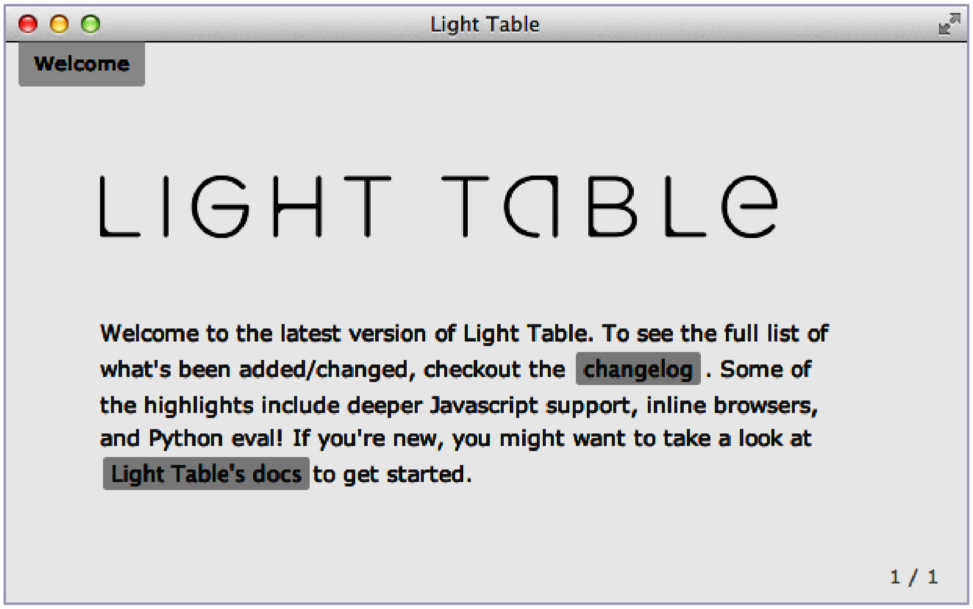
and another begins. Therefore, we can navigate and select code in terms of blocks of logic instead of lines of text.

In this chapter we'll be using Light Table to work with our guestbook application.[9] It's very easy to get up and running and will allow us to quickly dive into writing some code. However, its functionality is somewhat limited and you may find it insufficient for larger projects. Alternative development environments are discussed in Appendix 1, *Alternative IDE Options*, on page ?.

Using Light Table

Light Table does not require any installation and we can simply run the executable after it's downloaded.

Light Table offers a very minimal look. By default it simply shows the editor pane with the welcome message (see the following figure).



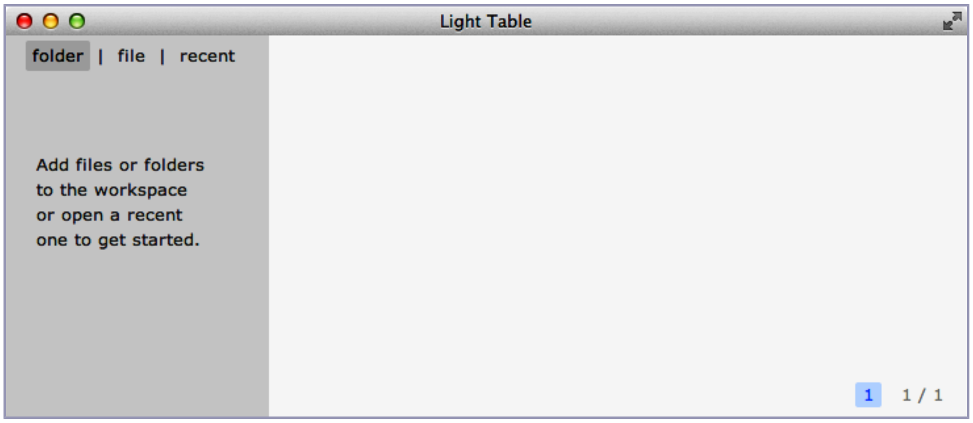
Welcome to the latest version of Light Table. To see the full list of what's been added/changed, checkout the changelog . Some of the highlights include deeper Javascript support, inline browsers, and Python eval! If you're new, you might want to take a look at Light Table's docs to get started.

Figure 1—Light Table workspace

We'll add the *workspace* pane from the menu by selecting *View -> Workspace* or pressing Ctrl-T on Windows/Linux or Cmd-T on OS X.

From there we can open the guestbook project by navigating to the Folder tab on the top left, as the following figure shows.

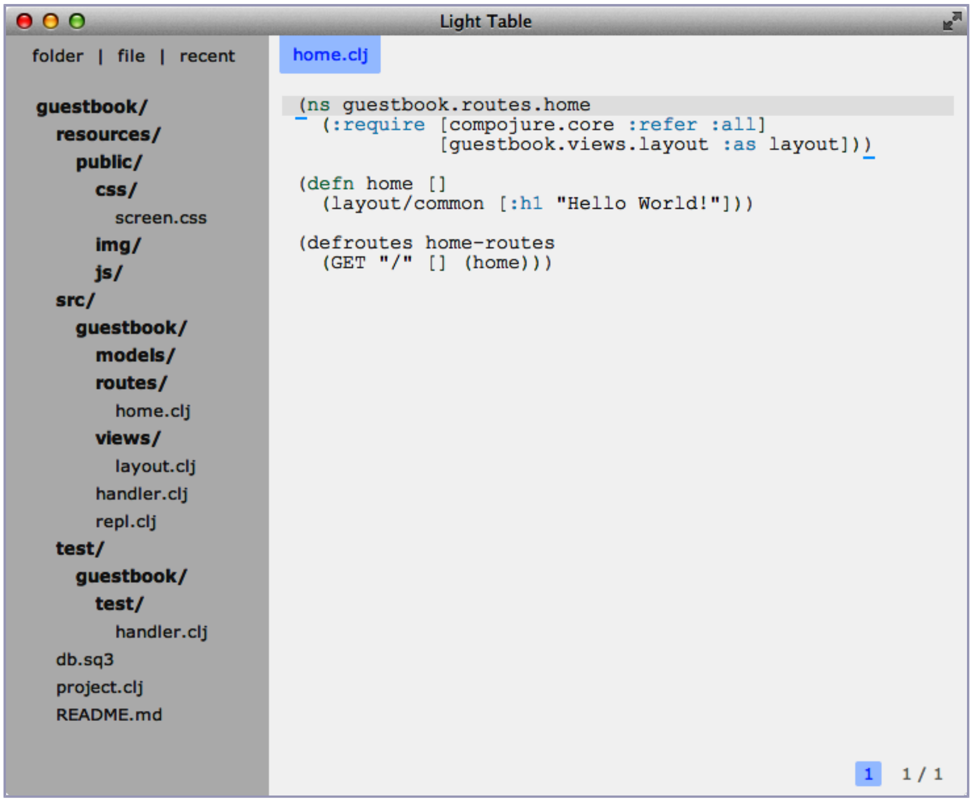
9. http://www.lighttable.com/



Add files or folders to the workspace or open a recent one to get started.

Figure 2—Opening a project

Once the project is selected we can navigate the project tree and select files we wish to edit (see the following figure).



```
guestbook/
  resources/
    public/
      css/
        screen.css
      img/
      js/
  src/
    guestbook/
      models/
      routes/
        home.clj
      views/
        layout.clj
      handler.clj
      repl.clj
  test/
    guestbook/
      test/
        handler.clj
  db.sq3
  project.clj
  README.md
```

```
(ns guestbook.routes.home
  (:require [compojure.core :refer :all]
            [guestbook.views.layout :as layout]))

(defn home []
  (layout/common [:h1 "Hello World!"]))

(defroutes home-routes
  (GET "/" [] (home)))
```

Figure 3—Light Table project

Now that we have our development environment set up, we can finally look at adding some functionality to our guestbook application.