Testing

Contents

Clojure

Luminus sets up a default test harness found in the `test` directory of the project.

The database tests run using the `:test` profile. This profile reads the environment variables from the `test-config.edn` file.

The recommended way to run the tests is using the lein-test-refresh plugin. The plugin watches namespaces for changes, and runs tests accordingly. The plugin can be run using the following command out of the box:

```
lein test-refresh
```

A default test will be created for the application handler:

```
(ns myapp.test.handler
  (:require [clojure.test :refer :all]
            [ring.mock.request :refer :all]
            [<app>.handler :refer :all]))

(deftest test-app
  (testing "main route"
    (let [response (app (request :get "/"))]
      (is (= 200 (:status response)))))

  (testing "not-found route"
    (let [response (app (request :get "/invalid"))]
      (is (= 404 (:status response))))))
```

In the tests, a mock request generated using the `ring.mock.request/request` function is generated and passed to the `<app>.handler/app` function. The response is tested against the expected

Build Tool: lein ▼

Topics

Your First Application
REPL Driven Developme
Application Profiles
HTML Templating
Static Assets
ClojureScript
Routing
RESTful Services
Request types
Response types
Websockets
Middleware
Sessions and Cookies
Input Validation
Security
Component Lifecycle
Database Access
Database Migrations
Logging
Internationalization
• **Testing**
Server Tuning
Environment Variables
Deployment
Useful Libraries
Sample Applications
Upgrading
Clojure Resources

Books

response for the route. The example tests simply test that a valid route returns a response with a status `200`, while an invalid one results in a `404` response.

When a relational database profile is used, such as `+postgres`, then a set of tests for the database are added to the project.

```
(ns myapp.test.db.core
  (:require [clojure.test :refer :all]
            [myapp.db.core :refer :all]
            [myapp.db.migrations :as migrations]))

(deftest test-users
  ;; Make sure the user with id 1 doesn't exist.
  ;; You can also use transactions around tests to ensure that.
  (delete-user! {:id "1"})
  (is (= 1 (create-user! {:id         "1"
                          :first_name "Sam"
                          :last_name  "Smith"
                          :email      "sam.smith@example.com"
                          :pass       "pass"})))
  (is (= (get-user {:id "1"})
         [{:id         "1"
           :first_name "Sam"
           :last_name  "Smith"
           :email      "sam.smith@example.com"
           :pass       "pass"
           :admin      nil
           :last_login nil
           :is_active  nil}])))

(use-fixtures :once (fn [f] (migrations/migrate ["migrate"]) (f)))
```

The database connection for the test database should be defined in the `test-config.edn` file.

ClojureScript

Luminus uses doo for running ClojureScript tests. ClojureScript tests are located in `test/cljs`. Doo runs tests in namespaces that have been given as a parameter to 'doo.runner/doo-tests' macro in doo_runner.cljs, that is located in 'test/cljs' directory of the project.

Doo requires a JavaScript environment where to run the tests. This can be a headless environment, such as PhantomJS, or a browser. Prerequisites to run ClojureScript tests depend on the environment. For further information, refer to the documentation provided by doo.

To run tests using PhantomJS, PhantomJS needs to be installed. Once PhantomJS has been installed, tests can be ran using test profile with doo:

```
lein with-profile test doo phantom
```

For running tests in a browser, Javascript test runner Karma and related plugins are needed. These can be set up by following Karma

installation instructions provided by doo. Once all the prerequisites are fulfilled, tests can be ran in a given browser, for example in Safari, with command

```
lein with-profile test doo safari
```

Using doo with the above commands builds ClojureScript with the tests included accordingly to test profile definitions, executes the tests, displays their results and watches for changes in files and runs the tests again when changes happen.

Luminus framework is released under the MIT License - Copyright © 2019