KrakenJS

In this lab, you will create a simple “todo” application using the Kraken web application framework.

# Objectives

In this lab, you will learn to

* create a model, view, and controller for "todo" items in Kraken,
* implement behavior for each route, and
* create an internationalized title.

# Install Tools

Before starting the lab, you will need a MongoDB server running on your lab machine for the database backend in our application. You will also need to install the generator-kraken module that will be used to generate a skeleton Kraken application.

## Install MongoDB

1. Go to the MongoDB homepage at http://www.mongodb.org and install the latest version for your operating system.

Be sure to follow all of the installation instructions. Mongo doesn't have a one-step install process. In particular, ensure that you create the database directory: /data/db on Mac and \data\db on Windows.

1. Once installed, start the server with the mongod command line utility.

## Install generator-kraken module

1. If you did not install Yeoman from the previous lab, you should do this now by opening a command prompt in the lesson directory and issuing the command npm install -g yo. This will download & globally install the latest version of yo.
2. Open a command prompt in the lesson directory and issue the command npm install -g generator-kraken. It may be necessary to prefix the npm command with sudo if you are using a Mac or Linux platform.

# Lab Steps

Now that the tools are installed, we can begin the lab.

## Scaffold a new web application with Kraken

1. Open a terminal in the lab directory, then issue the command yo kraken.

If you are using a Windows machine and it cannot find the 'yo' command, you probably do not have admin permissions on your machine. Ask your instructor for a workaround.

1. Enter todo for the app name, Todo Application for the description, and whatever you want for the author.
2. Take the defaults for the remaining prompts.

This will scaffold a web application that uses Express and Kraken in the directory todo.

## Start & view the web application

1. Change into the todo directory and issue the command npm start; this will start the server process.

After the application starts, it will display the hostname and port number on the console. Ignore any deprecation warnings. They won't affect your results.

1. Open http://localhost:8000 in a web browser.

You should see something similar to the following:



Once you see this, move on to the next step.

## Add MongoDB connectivity

1. Open package.json in your editor and add "mongoose": "3.8.12" to the dependencies section.
2. Copy the lib directory from the lab directory of this lesson into the todo directory, which contains your application.
3. Open index.js in your editor and add the following require statement at the top: var db = require('./lib/db');.
4. Add the line db.config(config.get('databaseConfig')); in the function at options.onconfig of index.js, right after the comment "any config setup/overrides here."

Make sure next(null); is the last statement in the function.

1. Open config/config.json and add the following section to this file immediately before the middleware property:

"databaseConfig": {  
 "host": "localhost",  
 "database": "test"  
 }

1. Stop the server and run npm install to install the mongoose dependency for MongoDB connectivity.
2. Start the server again with npm start and you should now see a message in the console that says "db connection open" indicating the MongoDB connection is working.

## Generate a controller and dependencies

1. Stop the server and run yo kraken:controller todos which will generate a model, view, controller, and I18N bundle in their respective places.

Accept the default at the prompt.

## Edit model

1. Open todo/models/todos.js and replace it with the contents of impl/model-todos.js. You can also just replace the file, but this will give you a chance to review the code.

This defines a very simple "todo" model with mongoose that only contains one String field.

## Edit controller

1. Open todo/controllers/todos/index.js and replace it with the contents of impl/controller-todos.js.

This controller defines typical create, read, update, delete routes using that should be familiar to most web developers.

### Copy CSS styles and graphics

1. Copy lab/css/app.less over the existing todo/public/css/app.less. This is a simple stylesheet and delete button similar to what was used in the Express lab.
2. Create the directory todo/public/png.
3. Copy lab/png/delete.png to todo/public/png/delete.png.

## Edit views

1. Open the file todo/public/templates/layouts/master.dust and add the following line as the last line in the head element:

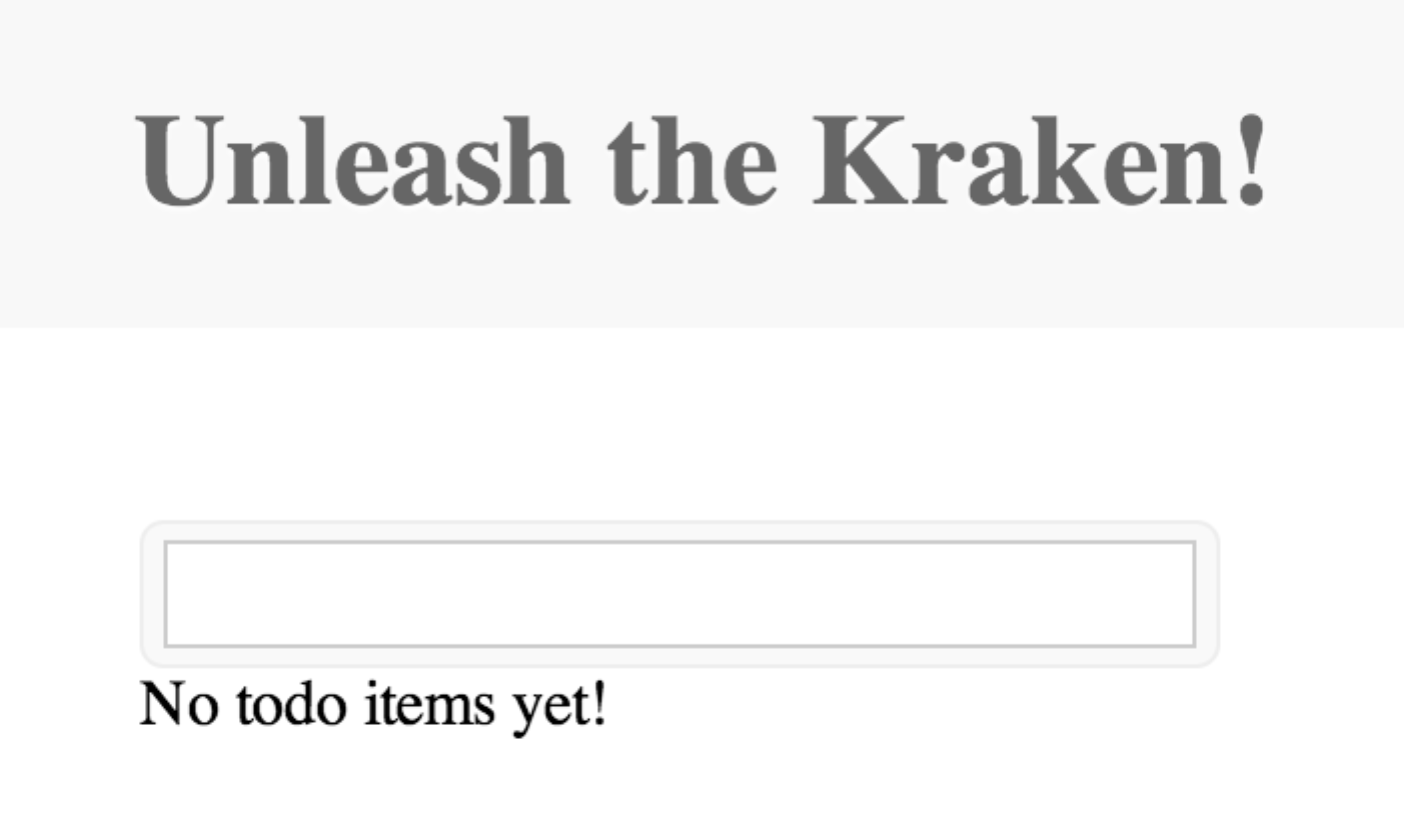
<link rel="stylesheet" href="/css/app.css">

1. Open todo/public/templates/todos/index.dust and replace it with the contents of impl/todos.dust.

This Dust template first contains a form for creating new "todo" items. Then it checks for the existence of a todos object to render the list of all "todos". If the todos object does not exist, it checks for the update\_todo object that is used to render the item to be updated. If neither exists, there are no "todo" items in the database yet.

## Run the web application

1. Start the web application again with npm start, return to the browser and view the URL http://localhost:8000/todos. You should see the following screen:



Verify that you are able to create, edit, and delete "todo" items and then move on to the next step.

# Internationalize the title

For the final part of the lab, we will take a brief look at how you can internationalize strings with Kraken.

## Create setLanguage controller

1. Create a file called controllers/setLanguage/index.js and paste the following implementation:

'use strict';  
  
module.exports = function (server) {  
 server.get('/:lang', function (req, res) {  
 res.cookie('language', req.param('lang'));  
 res.redirect('/todos');  
 });  
};

This route will set a cookie called language depending on the named parameter. We will add the language choices in the Dust template soon.

## Add language middleware

1. Open config/config.json and add the following to the middleware object:

"language": {  
 "priority": 95,  
 "enabled": true,  
 "module": {  
 "name": "path:./lib/language"  
 }  
},

Open todo/lib/language.js and examine what is happening. This middleware checks for the req.cookies.language cookie and if it exists, res.locals.context.locality is set with its value. This will cause Kraken to use the appropriate language bundle when rendering the template.

## Add a Spanish language bundle

1. Create locales/ES/es/todos/index.properties and paste the following key/value: greeting=Desata el Kraken!
2. Edit locales/US/en/todos/index.properties and change the greeting key to Unleash the Kraken!

## Modify template to switch languages

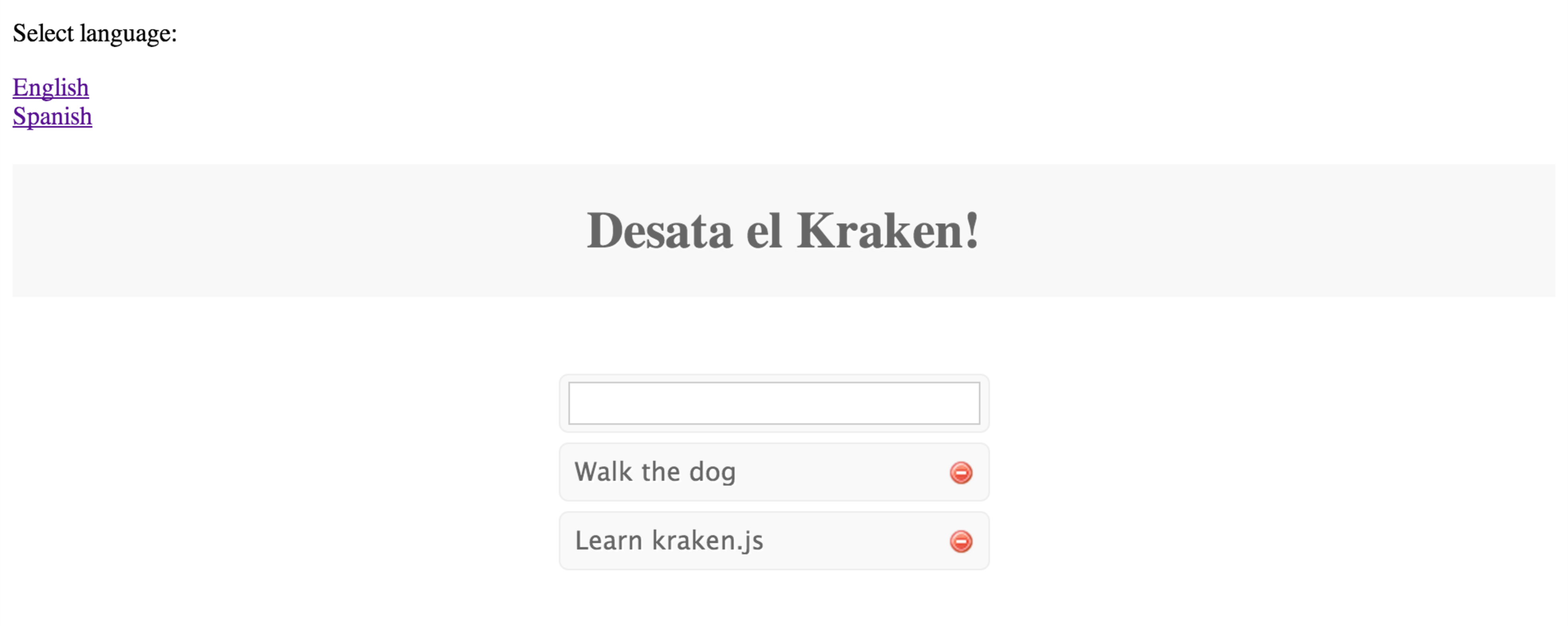
1. Open public/templates/todos/index.dust and make the following modification:

…  
{<body}  
 <p>Select language:</p>  
 <p><a href="/setLanguage/en-us" alt="English">English</a><br/>  
 <a href="/setLanguage/es-es" alt="Spanish">Spanish</a></p>  
 <h1 id="page-title">{@pre type="content" key="greeting"/}</h1>  
 <div id="list">  
…

This will create two links from which the setLanguage controller can be called and the appropriate cookie set. The title has been modified to use the greeting key from the message bundle instead of a hard coded greeting.

## Run the web application

1. Start the web application again with npm start, return to the browser and view the URL http://localhost:8000/todos. You should now see two links available and see something similar to the following if you select Spanish:



Once everything is working properly, this lab is now complete!