React – Internationalization

In this lab, you will create a React app with internationalization (i18n).

# Objectives

In this lab, you will

* Use an existing Kraken app named, ‘react-app’
* Add i18n configuration
* Edit React ‘todos.jsx’ page to use translated strings
* Run the app

# Use an Existing Kraken app

1. The lab folder contains an existing Kraken app, ‘react-app’, the solution to the ‘inner-components’ exercise.

# Run the Kraken app to verify correctness

1. Change to the /lab/react-app folder and start the application:

npm install

npm start.

1. Open the browser to localhost:8000 and the page should look like this:



# Configure Kraken to use ‘anemone-lingua’

1. Configure config/config.json to use PayPal’s internationalization module, ‘anemone-lingua’. Add the following as the first element in the middleware section in the file.

**"reactContentLoader"**: {  
 **"priority"**: 100,  
 **"enabled"**: **true**,  
 **"module"**: {  
 **"name"**: **"anemone-lingua"**,  
 **"arguments"**: [  
 {  
 **"contentPath"**: **"path:./locales"**,  
 **"fallback"**: **"en-US"** }  
 ]  
 }  
},

1. The above configures the anemone-lingua module to load the properties files from the ./locales folder.
2. Notice the priority is 100. We require this later.
3. The file structure for the source of the translated strings looks like:



1. The anemone-lingua module creates a hash file from all these properties files for a single language and sends that hash to the browser. The hash file for ‘en-us’ looks something like:

**var *hash*** = {  
 **"errors/404"**: {  
 **header**: **"File not found"**,  
 **description**: **"The URL <code>{url}</code> did not resolve to a route."** },  
  
 **"errors/500"**: { *// info about the error* },  
  
 **"errors/503"**: { *// info about the error* },  
  
 **"todos"**: {  
 **greeting**: **"Unleash the Kraken!"** }  
};

1. Kraken puts the hash in res.locals.messages on the server so it shows up as this.props.messages on the client. The React component accesses the translated strings and displays them on the screen in the render() method using code like:

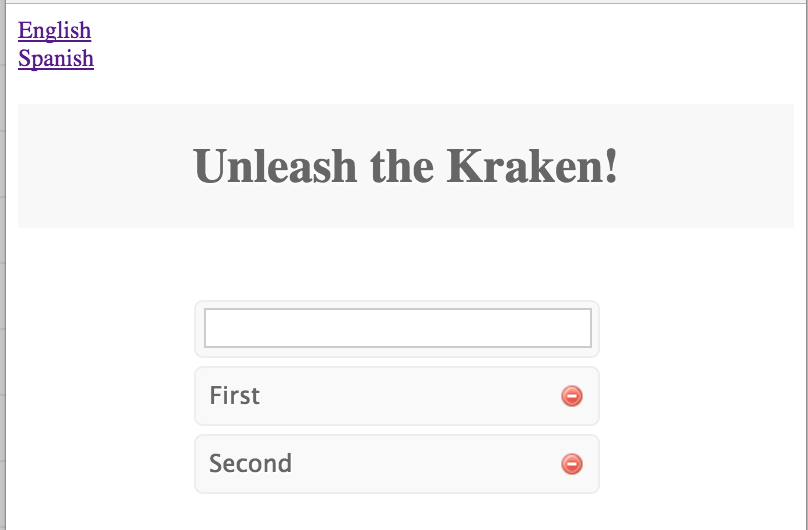
**var** msg = **this**.props.**messages** || {};  
msg = msg.**todos** || { **greeting**: **'Unknown message: greeting'**};  
  
**return** (  
 <**div id="layout"**>  
 <**h1 id="page-title"**>{msg.**greeting**}</**h1**>  
 <**div id="list"**>

1. Edit the TodoList component in /public/views/todos.jsx and insert the above code in the render() method.
2. Install the module and configure the package.json dependency with:

npm install anemone-lingua --save

# Add the Language Links

1. Add the links to the page to set the language to either English or Spanish as shown below:



1. Edit the /public/views/layout.jsx and insert the lines marked in yellow:

<**body**>  
  
<**a href="/setLanguage/en-us"**>English</**a**>  
<**br** />  
<**a href="/setLanguage/es-es"**>Spanish</**a**>  
  
{**this**.props.**children**}  
<**script src='/bundle.js'**></**script**>

1. Now we need to map the above URI, /setLanguage/en-us, to a middleware controller.
2. Edit the file, /controllers/index.js and add the following content IN FRONT OF the current router.get() call.

router.get(**'/setLanguage/:lang'**, **function** (req, res) {  
 res.cookie(**'language'**, req.**params**.**lang**);  
 res.redirect(**'/'**);  
});

1. This creates a cookie called language with the value of the current <language>-<country> code. Notice it then redirects the page to the home page.

# Read the Cookie and Change the Language

1. Something now needs to read the user-selected language and country code from the cookie and allow anemone-lingua to generate the correct translations for the browser.
2. The preferred language comes from the browser as a cookie called language as described above. We must convert this string to a locale object with the properties language and country.
3. Create a middleware component, /lib/language.js, with the following contents:

**'use strict'**;  
  
**module**.exports = **function** () {  
  
 **return function** (req, res, next) {  
 *//Pick up the language cookie.* **var** language = req.cookies.**language**;  
  
 *// the language variable e.g. 'en-us', 'es-es', 'language-country'  
  
 //Set the locality for this response.  
 // anemone-lingua will pick the appropriate bundle* **if** (language) {  
 **var** parts = language.split(**'-'**);  
 **var** locale = {  
 **language**: parts[0],  
 **country**: parts[1]  
 };  
 res.**locals**.**contentLocale** = locale;  
 }  
  
 *// bug in anemone-lingua* res.**locals**.**context** = res.**locals**.**context** || {};  
  
 next();  
 };  
};

1. Edit /config/config.json to load the above middleware component. Insert the following lines in the middleware section of the file.

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

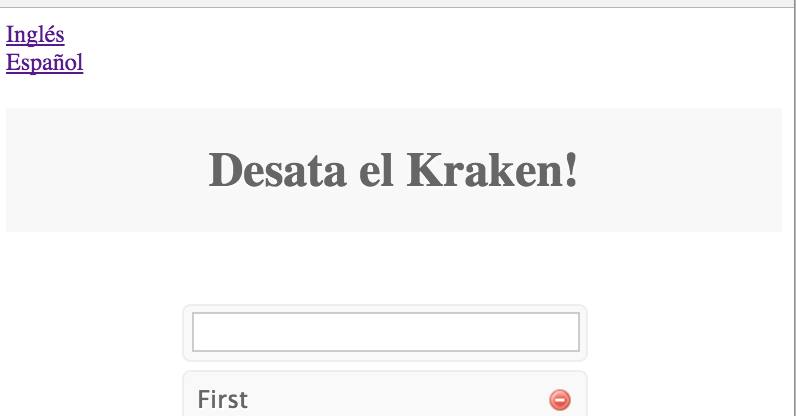
1. Notice the priority is 95, less than the priority of anemone-lingua. Therefore, this runs first, setting the locale later used by anemone-lingua.
2. Restart the application and change languages by clicking on the links.
3. Congratulations!!!!

# Extra Credit

1. Translate the text, ‘English’ and ‘Spanish’, used in the links in the layout.jsx code.
2. Create the layout.properties files for both English and Spanish. The Spanish contents should look like:

english=Inglés  
spanish=Español

1. Change the layout.jsx to load the correct message bundle.
2. Change the layout.jsx to inject the correct key as the text in the links.
3. Restart the application and watch it work…



Congratulations. You have completed this lab.