React – AJAX

In this lab, you will modify a React / Kraken app and add AJAX calls to the server.

# Objectives

In this lab, you will

* Examine the server REST interface for the Todos
* Modify the todos controller to use the REST interface
* Modify the todos component to use the todos controller
* Run the app

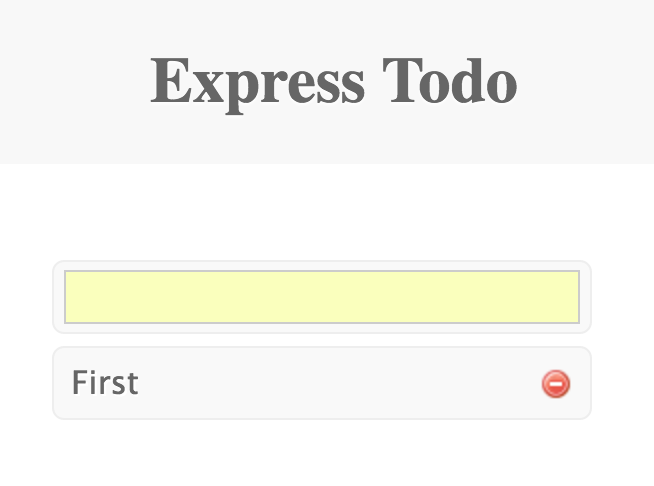
# Run the Kraken app to verify correctness

1. Change to the lab/react-app folder and run

npm install

npm start

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



1. React creates the contents using /public/views/todos.jsx which uses a controller, /public/js/todos-controller.js as shown below.

**var *todos*** = [{**id**: *uuid*(), **content**: **'First'**}];  
  
***module***.exports = {  
 create: **function** (content, callback) {  
 **var** todo = {  
 **id**: *uuid*(),  
 **content**: content  
 };  
 ***todos*** = ***todos***.**concat**(todo);  
 callback( ***todos***);  
 },  
 findAll: **function** ( callback ) {  
 callback( ***todos*** );  
 },  
 update: **function** (item, callback) {  
 **var** tt = ***todos***.map(**function** (todo) {  
 **if** (item.**id** === todo.**id**) {  
 todo.**content** = item.**content**;  
 }  
 **return** todo;  
 });  
 ***todos*** = tt;  
 callback( ***todos*** );  
 },  
 **delete**: **function** (id, callback ) {  
 **var** tt = ***todos***.filter(**function** (todo) {  
 **return** (todo.**id** !== id);  
 });  
 ***todos*** = tt;  
 callback( ***todos*** );  
 }  
};

1. The todo-controller.js contains a CRUD(F) interface for an in-memory list of todo items. Each of the methods uses a callback to return a list of todo items. We will modify this to call the REST interface from the server.
2. The model object in the server (the domain model) in /models/todos.js looks like this:

**'use strict'**;  
  
***module***.exports = **function** *TodosModel*() {  
 **return** {  
 **name**: **'todos'**,  
 **todos**: [ { **id**: **'1'**, **content**: **'First'**}],  
 **\_csrf**: **''** };  
};

1. The todos property is the array of items. The \_csrf property contains the CSRF (Cross Site Request Forgery) required when the browser sends a POST to the server.
2. Examine the REST interface located in /controllers/api/todos as shown below:

***module***.exports = **function** (router) {  
  
 **var** model = **new *TodosModel***();  
  
 router.get(**'/'**, **function** (req, res) {  
  
 model.**\_csrf** = res.**locals**.**\_csrf**;  
  
 res.json(model);  
  
 });  
  
 */\*\*  
 \* Create a new* ***todo*** *\*/* router.post(**'/'**, **function** (req, res) {  
 **var** todo = {  
 **id**: *uuid*(),  
 **content**: req.**body**.**content** };  
 model.**todos**.push(todo);  
 res.json(model);  
 });  
  
 */\*\*  
 \* Delete the* ***todo with the input parameter*** *\*/* router.delete(**'/:id'**, **function** (req, res) {  
 **var** id = req.**params**.**id**;  
  
 **var** todos = model.**todos**.filter(**function** (todo) {  
 **return** todo.**id** !== id;  
 });  
  
 model.**todos** = todos;  
 res.json(model);  
 });

1. Notice that each of the REST api calls returns the model object.
2. Let’s refactor the /public/js/todo-controller.js to use AJAX to access the RESTful interface on the server.
3. The delete() method looks like this:

**delete**: **function** (id, callback ) {  
 **var** tt = ***todos***.filter(**function** (todo) {  
 **return** (todo.**id** !== id);  
 });  
 ***todos*** = tt;  
 callback( ***todos*** );  
}

1. The AJAX delete() method should look like this:

**delete**: **function** (id, callback ) {  
 ***$***.post(**'/api/todos/'** + id + **'?\_method=delete'**, { **\_csrf**: ***\_csrf*** }, **function**( model ) {  
 ***\_csrf*** = model.**\_csrf**;  
 callback( model.**todos** );  
 });  
}

1. The AJAX call uses jQuery ($.post()) at the url, /api/todos/:id?\_method=delete . The \_method=delete allows the browser to send a POST and the server to receive a DELETE. The body sends the \_csrf value to the server required for POSTs.
2. The findAll() method should use the following AJAX call:

***$***.get( **'/api/todos'**, **function**( model )

1. The create() method should use:

***$***.post(**'/api/todos'**,{ **content**: content, **\_csrf**: ***\_csrf*** }, **function**( model ) {

1. Notice it sends two properties in the body.
2. The findAll() method should use:

***$***.get( **'/api/todos'**, **function**( model ) {

1. Now, import the jQuery library using:

**var *$*** = ***require***(**'jquery'**);

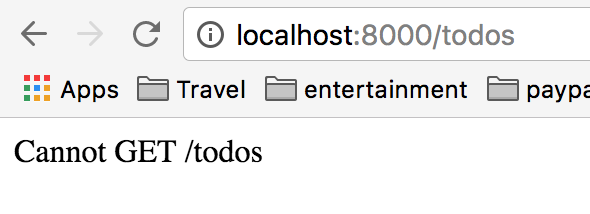
1. Install the jquery library and add to the package.json with:

npm install jquery --save

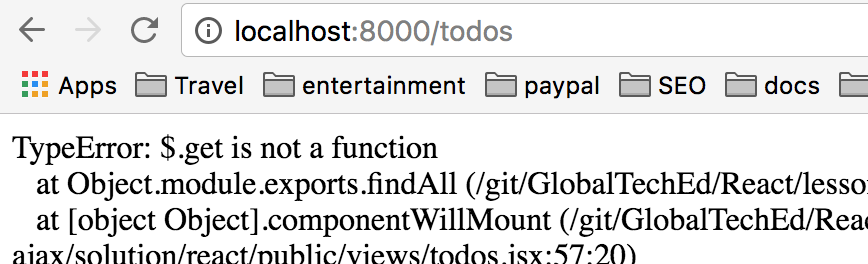
1. Modify the /public/views/todos.jsx to use the controller:

**var *TodoList*** = ***module***.exports = ***React***.**createClass**({  
 getInitialState() {  
 **return** {  
 **todos**: []  
 }  
 },  
  
 componentWillMount() {  
 ***controller***.findAll( **this**.done );  
 },  
  
 done: **function**( todos ) {  
 **this**.setState( { **todos**: todos });  
 },  
  
 render: **function** *render*() {

1. IMPORTANT: The componentWillMount() method is called on both the client AND server side rendering. This won’t work when doing server side rendering.
2. Restart the server and refresh the page. Add / delete some TODO items.
3. Refresh the page and generate an error



1. This error is generated because NODE does not have a route configured.
2. Edit the /controllers/index.js file to change the path to ‘/\*’
3. Restart the server and restart the page. Click on the link to the todos application.
4. Now refresh the page.



1. The error occurs on the server and indicates the $.get(), a jQuery call does not exist.
2. Fix the error by using componentDidMount() instead of componentWillMount().
3. Restart the server and refresh the browser. Add / delete some todo items.

Congratulations. You have completed this lab.