Lab Guide

Killer UX

Brad Tilton & Jake Gillespie

Lab instance: https://LabPrefix###.lab.service-now.com

Default Login / Password:

admin / Knowledge15

itil / Knowledge15

employee / Knowledge15

This

Page

Intentionally

Left

Blank

Lab 1 - Populate the ToDo Page

Description

You will start with a custom, scoped ToDo application built in ServiceNow and a UI Page which will be the single interface to the application. The page already has jquery and bootstrap loaded as well as a few static todos. In this lab you will call load AngularJS, create a module and controller and use an $http.get() call to load the UI Page with ToDos from the ToDo table.

Acceptance Criteria

The user can load the UI Page and see a list of all their outstanding ToDos.

Set Your Application

1. Log into your instance
2. Click the gear icon on the right side of the header and select Cloud Sherpas - ToDo from the application picker.

Add ToDos

1. Go to the ToDo table in your instance.
2. Create a couple of ToDos
    a. Date and User will auto-fill

Configure UI Page

1. Find the UI Page named home_lab.
2. Click on the Endpoint field to see what the starting state is.
3. Add the angular reference
    a. Go to the html field
    b. Add a reference to the AngularJS google cdn towards the end of the html field just before the other script references..

```
<script
src="//ajax.googleapis.com/ajax/libs/angularjs/1.2.16/angular.min.js"></
script>
```

4. Create a module
    a. In the client script field we need to create an angular module.
    b. Add this line:

```
var app = angular.module('todo', []);
```

5. Create a controller
    a. Also in the client script we'll create a controller. All of our subsequent javascript will go into the controller.

b. Add this line:

```
app.controller('ToDoController', [ '$scope', '$http', function($scope,
$http) {
      $scope.url = '/api/now/table/x_clds_todo_app_todo';
      $http.defaults.headers.common.Accept = "application/json";
}]);
```

6. Add a function in the controller to load data into our app.

```
$scope.loadData = function () {
    $http({
        method: 'GET',
        url: $scope.url + '?sysparm_query=complete=false'
    }).
    success(function(data, status) {
        $scope.tasks = data.result;
    }).
    error(function(data, status) {
        $scope.tasks = [{'name':'Error loading ToDos'}];
    });
}
$scope.loadData();
```

7. Now we need to configure the html in the ui page to receive live data from our ToDo app to replace the demo todo we currently have in there.
    a. First, we'll call our app by adding the `ng-app="todo"` attribute to our container-fluid div in line 10.
    b. Next, we'll call the controller by adding `ng-controller="ToDoController"` to the row div in line 11.
    c. Now we need to tell our list element to repeat for every todo using the ngRepeat angular directive. For this, we'll add `ng-repeat="task in tasks"` to the `<li class="list-group-item">` tag.

d. For our last step we need to populate a couple of data points from our ToDos into the app between our repeating <li> tags using angular markup {{}}.
    i. Let's populate the name field from the todo table in our row using angular markup.

```
<div class="col-md-6 col-xs-12">
    <h4>{{task.name}}</h4>
</div>
```

    ii. Now we can populate the date in the next column.

```
<div class="col-md-3 col-xs-5">
    <h4>{{task.date}}</h4>
</div>
```

8. Now we can test the ui page. Refresh the UI Page you opened in the other tab and you should see the original static data replaced with the ToDos you created earlier.

Lab 2 - Add Filtering to the ToDo Page

Description

In this lab you will add active filtering to your ToDo page so you can search your open todos from the top of the page.

Acceptance Criteria

The user will see a search box at the top of the page they can use to search the names of the ToDos in the list.

Add Search

1. The first thing we need to do is to add an input box before our list of todos.
    a. Let's add another before the current `<ul class="list-group">`.

```
<ul class="list-group">
    <li>
        <form>
            <input class="form-control input-lg" type="text"
placeholder="Filter ToDos" />
        </form>
    </li>
</ul>
```

    b. Refresh the ui page. Do you see a search box?
2. Now we're going to use that input to filter our ToDo array.
    a. First, we're going to use the ng-model directive to bind our input box to the filterText property by adding the following to our input element.

```
ng-model="filterText"
```

    b. Next we'll use the filterText property to filter the tasks array by changing the ng-repeat directive we added in lab 1 to reflect the following.

```
ng-repeat="task in tasks | filter:filterText"
```

    c. Refresh the ui page and filter in the Filter ToDos box.

Lab 3 - Add the Ability to Create New ToDos

Description

In this lab you will add html and javascript to your page to create new todos in ServiceNow and show them on the page without reloading the page.

Acceptance Criteria

A user should be able to add a ToDo to the ToDo table from the ui page and see their addition reflected in the page without refreshing.

Lab Steps

1. First we need to add an input box and submit button to the html portion of our page so we can add a ToDo.
    a. Let's add a form and an input text box after the second tag.

```
<form>
    <div class="input-group">
        <input class="form-control input-lg" type="text"
placeholder="Add ToDo" />
    </div>
</form>
```

    b. Save and refresh the page.
    c. That looks ok, but people like buttons. Let's add an Add button on the line after the input element.

```
<span class="input-group-btn">
    <input class="btn btn-default btn-lg" type="submit">Add</input>
</span>
```

    d. Refresh the page.
2. Now we need to add a function to our controller to take the input and post it to the ToDo table using the REST API.

a. Let's add the addTask function to our scope.

```
$scope.addTask = function(todo) {
    $http({
        method: 'POST',
        url: $scope.url,
        headers: {'Content-Type': 'application/json'},
        data: todo
    }).
    success(function(data, status, headers, config) {

    }).
    error(function(data, status, headers, config) {

    });
}
```

b. Now that we've posted to the data, we need to add it back to our page's scope, so let's add the following to the success() function.

```
$scope.tasks.push(data.result);
```

and this to our error() function:

```
$scope.tasks.push([{'name':'Error adding ToDo'}]);
```

3. Now that we have our html elements and angular function in place we need to link them together.
    a. First, lets use the ng-model directive to bind our input box to the todo.name property using the method from Lab 2.2a.

    ```
    ng-model="todo.name"
    ```

    b. Next, we need to add the ng-submit directive to our form element to call our addTask function we added in step 2.

    ```
    ng-submit="addTask(todo)"
    ```

    c. Save those changes and refresh your ui page.
4. Try adding a ToDo. Does it show up on the page and did it add it to the ToDo table on the back end?
5. There is one more thing we can do to make this more user friendly.
    a. When you submitted the ToDo the value in the input box stayed. It would be nice if that box cleared after you submitted the todo. This is easy to do by adding a line to both the success and error functions inside our addTask() function.

    ```
    $scope.todo.name = '';
    ```

b. Refresh your ui page and add another ToDo. Did the add todo function work?

Lab 4 - Add the ability to complete or delete the todos

Description

In this lab you will add html and javascript to your page so that you can complete or delete ToDos from the page.

Acceptance Criteria

A user should be able to complete or delete their ToDos from the page without refreshing.

Lab Steps

1. First we need to add a function in our controller to either delete or update the record to make it completed using the REST API.
    a. For this function, we need to target a specific ToDo and we'll also need to tell our function whether we're updating or deleting the ToDo. The idx argument is going to determine which element in the array we're targeting. Let's accept a couple of arguments and then retrieve the correct ToDo from our tasks array.

```
$scope.removeTask = function(idx, action) {
   var removedTask = $scope.tasks[idx];
}
```

    b. Then the last thing we need to do is post to the ToDo and either update or delete the record depending on the action we pass. Add the following to the removeTask function.

```
$http({
    method: action,
    url: $scope.url + '/' + removedTask.sys_id,
    headers: {'Content-Type': 'application/json'},
    data: {'complete':true}
}).
success(function(data, status, headers, config) {
    $scope.tasks.splice(idx, 1);
});
```

    c. Notice that we also removed the ToDo from the scope through the success promise in this function. Try adding an error promise here as well.
2. Now we need to add the html that will complete or delete our ToDos.
    a. You may have noticed that we had an empty quarter of the width with the following tags.

```
<div class="col-md-3 col-xs-7">
</div>
```

b. Now let's add our two buttons there.

```
<div class="pull-right btn-group-lg btn-group">
   <button type="button" class="btn btn-success btn-lr">
      <span class="glyphicon glyphicon-ok" aria-hidden="true" />
   </button>
   <button type="button" class="btn btn-danger">
      <span class="glyphicon glyphicon-remove" aria-hidden="true" />
   </button>
</div>
```

c. Save and refresh your ui page. Do the buttons show up to the right of the name and date?

d. If they do, let's add some functionality to them. Angular has an ng-click directive we can use.

   i. For the complete task button, we can add the attribute to the button element.

      ```
      ng-click="removeTask($index,'PUT')"
      ```

   ii. For the delete task button, let's add this attribute to that button element.

      ```
      ng-click="removeTask($index,'DELETE')"
      ```

e. Let's go ahead and save and test that functionality. When you hit the complete and delete buttons do the tasks go away. If you refresh are they still there? What if you go check the ToDo table?

Lab 5 - Add the ability to edit existing todos

Description

In this lab you will add html and javascript to your page so that you can edit ToDos from the page.

Acceptance Criteria

A user should be able to edit their ToDos from the page without refreshing.

Lab Steps

1. First we're going to add some html elements.
    a. First we're going to add a form with an input element we can use to edit the name of the todo. Add the form right after the task name h4

```
<div class="col-md-6 col-xs-12">
    <h4>{{task.name}}</h4>
    <form class="ng-pristine ng-valid">
        <input class="form-control input-lg" type="text" />
    </form>
</div>
```

    b. If you save and refresh you'll see our page looks a little wonky. We really don't want the label and input field to be there at the same time, so we're going to use the ng-show directive to show and hide these based on a new property we're creating called showNameEdit.

```
<h4 ng-show="!showNameEdit">{{task.name}}</h4>
<form class="ng-pristine ng-valid" ng-show="showNameEdit">
   <input class="form-control input-lg" type="text" />
</form>
```

    c. Now let's add bind the input box to the label by adding the following ng-show directive to the input element.

```
ng-model="task.name"
```

    d. Lastly, let's use ng-click, ng-submit and ng-blur to set showNameEdit so we can show and hide the label and input.

```
<h4 ng-show="!showNameEdit" ng-click="showNameEdit =
true">{{task.name}}</h4>
<form class="ng-pristine ng-valid" ng-show="showNameEdit" ng-
submit="showNameEdit = false">
```

```
    <input focus-me="showNameEdit" ng-blur="showNameEdit = false"
class="form-control input-lg" ng-model="task.name" type="text" />
</form>
```

e. You should now be able to click on the label and make the input box appear. Then when you click into the input box and hit enter or click out of it it should go away and your changes should be reflected in the label.

2. We have the page scope updating now, but we also need to update to update the ServiceNow database
   a. Now let's add another function to our controller for updating the the current task.

```
$scope.updateTask = function(idx) {
    var updatedTask = $scope.tasks[idx];
}
```

   b. We also need to post the task using the REST API by adding the following after the updatedTask line.

```
$http({
    method: 'PATCH',
    url: $scope.url + '/' + updatedTask.sys_id,
    headers: {'Content-Type': 'application/json'},
    data: {'name': updatedTask.name, 'date': updatedTask.date}
}).success(function(data, status, headers, config) {});
```

3. Now that we have the function created let's call it from our ng-blur and ng-submit directives.

```
<h4 ng-show="!showNameEdit" ng-click="showNameEdit = true">{{task.name}}</h4>
<form class="ng-pristine ng-valid" ng-show="showNameEdit" ng-
submit="updateTask($index); showNameEdit = false">
    <input ng-blur="updateTask($index); showNameEdit = false" class="form-
control input-lg" ng-model="task.name" type="text" />
</form>
```

4. One last thing is we're going to add a custom directive that already exists in your ui page client script to the input field. This directive will automatically add the focus to the input element when it is shown.
   a. Uncomment the code at the bottom of the client script field on your ui page.
   b. Now add the focusMe directive to the input field.

```
<input focus-me="showNameEdit" ng-blur="updateTask($index); showNameEdit
= false" class="form-control input-lg" ng-model="task.name" type="text"
/>
```

5. Let’s go ahead and test the page to make sure that you can edit tasks and post to the DB.

Challenge

How would you use the same logic and functionality to be able to edit the date field as well as the name field?

Challenge Labs

In order to complete the challenge labs, you will have to use what you've learned in the previous labs as well as put your googling skills to the test.

User should be able to order the ToDos

1. Need a field where the user can select what field to order by
2. Bind that field to a property
3. Use orderBy in the filter in the ngRepeat directive

User should be able to see completed ToDos

1. Need some sort of button that will show completed ToDos
2. Write or modify a function that can be called that will use JSONv2 to get all of the user's completed ToDos.
3. Duplicate the existing html to show a completed todo list below the existing todo list.
4. Call the function from your button using ngClick and use ngShow to show or hide the completed list based on what's been clicked.

Bonus: Add Tagging to the ToDos

Make use of Fuji's tagging in your ToDo app.