knowledge15®
April 19–24, 2015 • Mandalay Bay • Las Vegas, NV

Lab Guide

AngularJS

Developing Modern User Interfaces

Tyler Jones & Will Leingang

Default Login / Password:

admin / Knowledge15

itil / Knowledge15

employee / Knowledge15

This

Page

Intentionally

Left

Blank

Introduction

Welcome to using Angular on ServiceNow! With this guide, and a basic understanding of Angular, you learn how to use the Angular framework effectively on top of the ServiceNow platform. This guide is made up of several easy to achieve goals. Each goal helps you understand a fundamental part of an angular application such as how to load angular, how to build a view, and how to communicate with the server.

This lab guide will tell you very specifically what to put in your code. If you copy and paste code from this guide (in digital format) you may have formatting problems. If you hand-type the code in this guide, you may also learn more. If you decide to name your custom application or tables differently than we did, you will want to make sure that you use those names consistently throughout the code you will write.

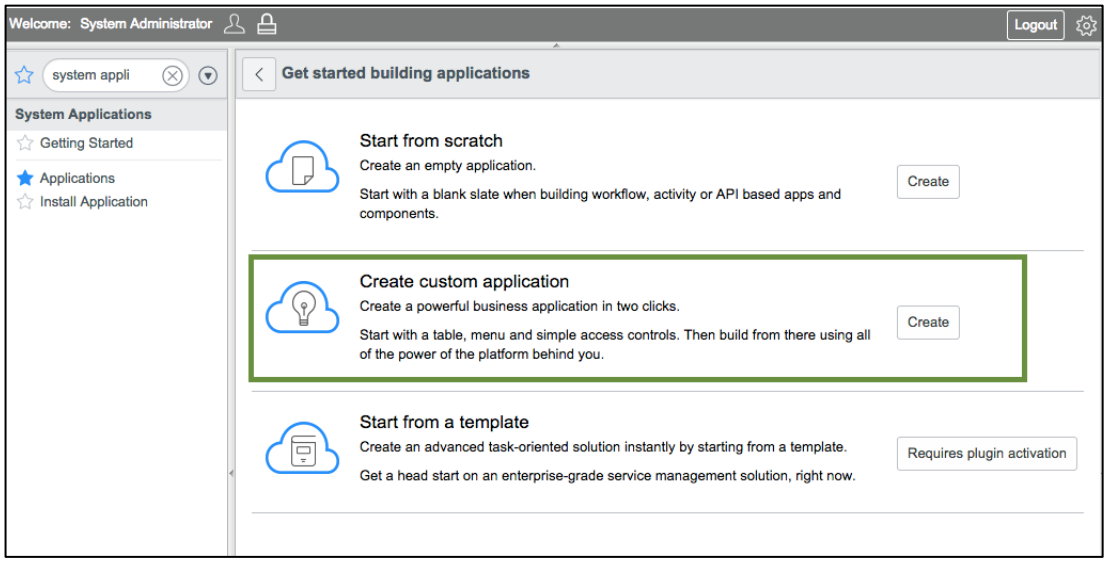
Lab Goal

Like all good guides, we start at the beginning. Before you can build an application you need a place to put it. The first thing we do is create a basic scoped application using the new Application Creator.

Lab 1.1
Create a New App

Create a Base App

1. Go to **System Applications > Applications**.
2. Use **Develop**.
3. Click **New**.
4. Click **Create** next to **Create custom application**.



5. Create the application using the following values:

   - Name: **Angular Todo**
   - Scope: **x_angular_todo**
   - Menu: **Angular Todo**
   - User role: **x_angular_todo_user**
   - Table:
     - Label: **Angular Todo List**
     - Name: **x_angular_todo_list**
     - Module: **Angular Todo List**
     - Extends Table: **task**

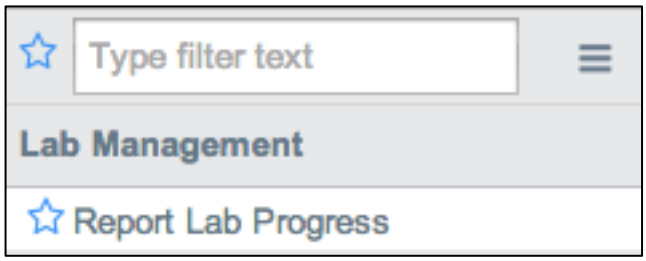
   When you click **Create** it:

   - Creates a table for your todo list
   - Creates a module in the navigator
   - Creates a user role for the application
   - Creates a scoped application
   - Creates an update set to bundle your changes

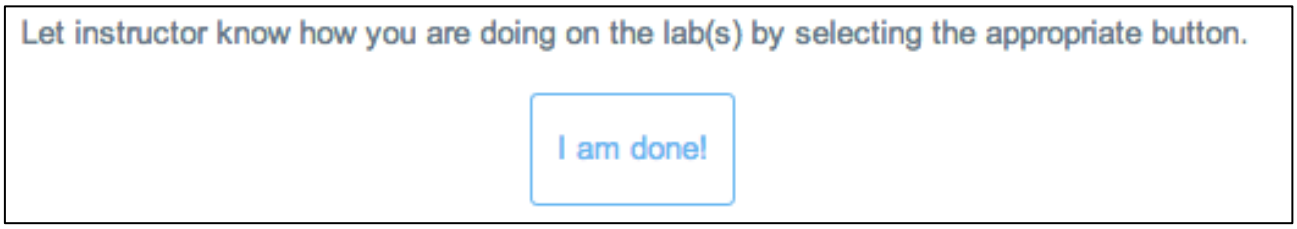
For all further steps make sure **Angular Todo List** is selected as the application and update set in the global configuration setting. After completing these steps, go to the next section to tell us you are done!

Progress Report

1. Navigate to **Lab Management > Report Lab Progress**.



2. Click **I am Done!**



Lab Goal

Lab 2.1 Add Angular to the Instance

At ServiceNow we use Angular for many different components and applications, but we do not have Angular included and running on every page. You are responsible for including and bootstrapping (starting up) Angular on your own for your custom applications. How? By using a UI Script.

Adding Angular via a UI Script

To get started you need to create a UI Script that contains the Angular framework along with any other JavaScript libraries or frameworks you want to use. We use the latest stable version of Angular just to keep things simple. We will take advantage of one of the latest features, one time binding, that make applications snappier.

In case you are not familiar, here is a little background on the terminology and components of the ServiceNow platform you use. You can find more details and usage examples about all of these things at wiki.servicenow.com.

Jelly
Jelly is the xml templating that theServiceNow UI is built on top of. A Jelly template can contain html, jelly expressions, and JavaScript; and it can evaluate server-side JavaScript.

UI Macro
A UI Macro is a bit of Jelly stored in the UI Macro table that can be included on UI Pages and UI Macros. UI Macros differ from UI Pages in that they cannot be accessed as destination pages via a browser such as myinstance.servicenow.com/some_page.do.

UI Page
UI Pages are Jelly templates stored in the UI Pages table. Use UI pages to create custom pages on your ServiceNow instance. UI pages can be accessed via their names. For example: myinstance.servicenow.com/my_ui_page.do where my_ui_page is the actual name of the UI page you create.

UI Script
A UI Script is similar to a UI Macro in that it can contain reusable content, but UI Scripts can contain Client-Side JavaScript only.

Script Include
Create script includes to write JavaScript functions and classes for use on the server. This is where you write the server-side business logic of your angular application.

Add Angular by Creating a UI Script

1. Open **UI Scripts**.
2. Click **New** on the list.
3. Input the following values. This code will load the Angular Framework from Google's public CDN by writing a script tag into any page that includes the UI Script.
   - Script Name: **Angular_1_3_framework**
   - Api Name: **x_angular_todo.Angular_1_3_framework**
   - Script:

```
document.writeln('<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.js"></script>');
```

Lab Goal

This lab explains how to create a new UI Page quickly and how to use this UI script and see Angular working on the UI page.

Lab 3.1 Add Angular to a UI Page

Create a UI Page That Uses Angular

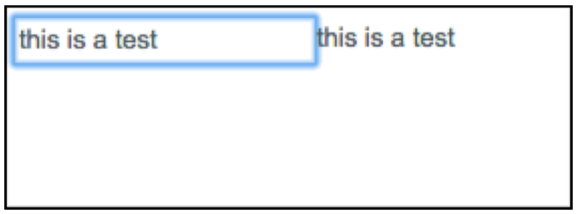
1. Go to **UI Pages**.
2. Click **New**.
3. Create a new UI Page with the following:
   - Name: **angular_todo**
   - HTML:

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<script language="javascript" src="x_angular_todo.Angular_1_3_framework.jsdbx" />
<div ng-app="">
    <input type="text" ng-model="myVar"/>
    {{myVar}}
</div>
</j:jelly>
```

Verification

Since this is a scoped application, the actual URL to the UI page is prefixed with your application name. The UI page URL is shown next to the Endpoint label on your form. In this case it will be: **x_angular_todo_angular_todo.do**

1. Open a new tab and go to **[instancename]/x_angular_todo_angular_todo.do**



If everything worked you should see a blank page with a single input field. Whatever you type in that field should automatically print right next to it. Goal 1 accomplished! This demonstrates a very basic Angular feature, using a UI Script to import Angular, and a UI Page to bring it all together.

When you have reached this point, please tell us you are ready to proceed.

Progress Report

1. Navigate to **Lab Management > Report Lab Progress**.

2. Click **I am Done!**

Lab Goal

Now start building the app. The goal of our app is to collect to-do lists and manage them.

Lab 4.1
Create Our First Angular App

Create the App

1. Go to **UI Scripts**.
2. Click **New**.
3. Create a UI Script with the following values:
   - Name: **app_angular_todo**
   - Script:

```
angular.module('todo', []);
angular.module('todo').controller('TodoListCtrl', function($scope) {
    $scope.todos = [];

    $scope.add = function(newTodo) {
        $scope.todos.push(newTodo);
    };
});
```

Update the UI Page to Use the App

We want to update the page to use this app. First, add the new UI script to this page:

1. Go to **UI Pages**.
2. Go to **angular_todo**.
3. Change the page so that it has the following script:

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <script src="x_angular_todo.angular_1_3_framework.jsdbx"></script>
    <script src="x_angular_todo.app_angular_todo.jsdbx"></script>
    <div ng-app="">
        <input type="text" ng-model="myVar"/>
        {{myVar}}
    </div>
</j:jelly>
```

Add the Angular App to the UI Page

1. While still on the **UI Page** titled **angular_todo**, add the Angular App:

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <script src="x_angular_todo.angular_1_3_framework.jsdbx"></script>
    <script src="x_angular_todo.app_angular_todo.jsdbx"></script>
    <div ng-app="todo">
        <input type="text" ng-model="myVar"/>
        {{myVar}}
    </div>
</j:jelly>
```

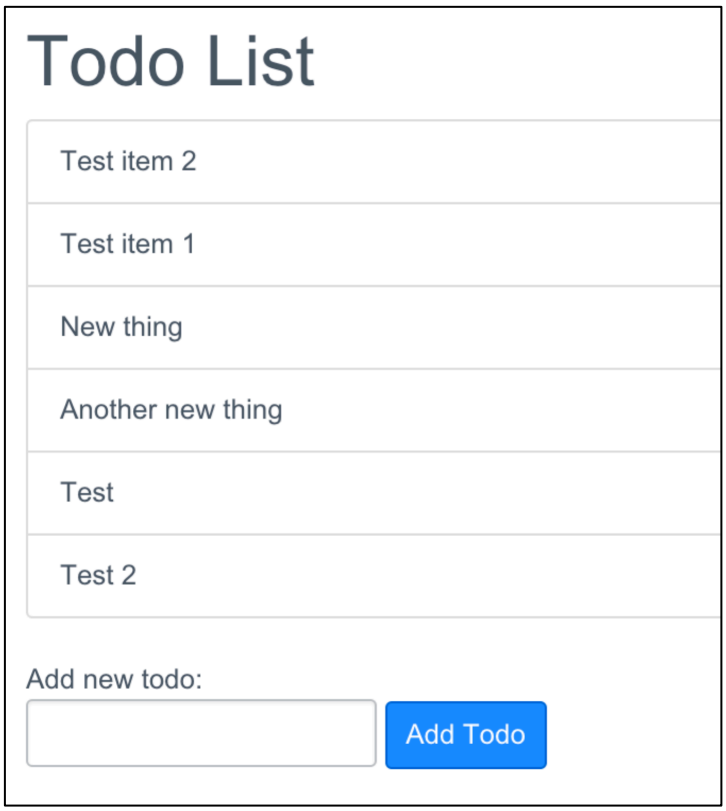
Add Markup for the Controller

1. Add the template logic for the controller:

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <script src="x_angular_todo.angular_1_3_framework.jsdbx"></script>
    <script src="x_angular_todo.app_angular_todo.jsdbx"></script>
    <div ng-app="todo">
        <div class="col-md-6" ng-controller="TodoListCtrl">
            <h1>Todo List</h1>
            <ul class="list-group">
                <li class="list-group-item" ng-repeat="todoItem in todos" ng-bind="todoItem"></li>
            </ul>
            <div>
                <label>Add new todo:
                    <input class="form-control" ng-model="newTodo"/>
                </label>
                <button ng-click="add(newTodo)" type="button" class="btn btn-primary">Add Todo</button>
            </div>
        </div>
    </div>
</j:jelly>
```

Verification

1. Load the page at **x_angular_todo_angular_todo.do**. You should be able to add todos to your list. If that works, then great!



Lab Goal

Now that the application works, leverage ServiceNow and store this data in the instance. We already created our table **x_angular_todo_list** to store the data.

For this lab, use GlideAjax to handle the data. If you are familiar with GlideAjax, you know that it is XML-based. Work around this issue to use JSON naturally with Angular.

Lab 4.2
Persist the Data in Our App

Add A Script Include for the Backend

1. Go to **Script Includes**.
2. Click **New**.
3. Create a **Script Include** with the following:
   - Name: **TodoAppPersistence**
   - Client callable: **true**
   - Script:

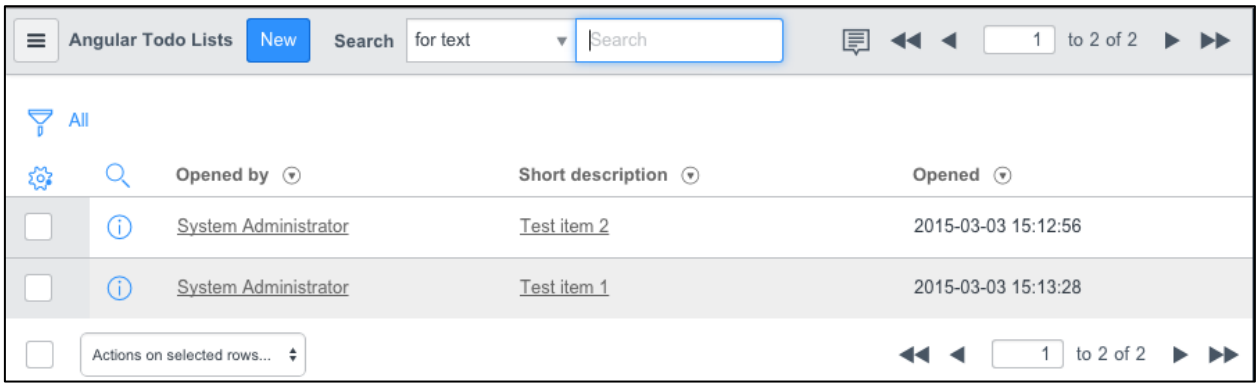
```
var TodoAppPersistence = Class.create();
TodoAppPersistence.prototype = Object.extendsObject(global.AbstractAjaxProcessor, {
    getList: function() {
        var todos = new GlideRecord('x_angular_todo_list');
        todos.addQuery('opened_by', gs.getUserID());
        todos.orderBy('opened_at');
        todos.query();

        var list = [];
        while (todos.next()) {
            list.push(todos.short_description + '');
        }

        return new global.JSON().encodeArray(list);
    },

    type: 'TodoAppPersistence'
});
```

4. Confirm that there is data to pull. Create some records in the table, and be sure to fill in the **opened_by** and **short_description** fields.



5. Now set up the app to receive the data. To want GlideAjax to work with JSON use the adapter for that working with Angular. Make a dependency to it in the application.

Add GlideAjax to the Angular App

1. Go to UI Scripts and **open app_angular_todo**.

2. Add a dependency **to sn.glideAjax**:

```
angular.module('todo', ['sn.glideAjax']);
angular.module('todo').controller('TodoListCtrl', function($scope, GlideAjax) {
    $scope.todos = [];

    $scope.add = function(newTodo) {
        $scope.todos.push(newTodo);
    };
});
```

Add GlideAjax to the UI Page

1. Go to **UI Pages**.
2. Open the UI page **angular_todo**.
3. Modify the page to include **GlideAjax:**

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <script src="x_angular_todo.angular_1_3_framework.jsdbx"></script>
    <script src="x_angular_todo.app_angular_todo.jsdbx"></script>
    <script src="service.sn-glideAjax.jsdbx"></script>
    <div ng-app="todo">
        <div class="col-md-6" ng-controller="TodoListCtrl">
            <h1>Todo List</h1>
            <ul class="list-group">
                <li class="list-group-item" ng-repeat="todoItem in todos" ng-bind="todoItem"></li>
            </ul>
            <div>
                <label>Add new todo:
                    <input class="form-control" ng-model="newTodo"/>
                </label>
                <button ng-click="add(newTodo)" type="button" class="btn btn-primary">Add Todo</button>
            </div>
        </div>
    </div>
</j:jelly>
```

Now there is a backend and a library. Query for the **Todos** and populate the model array.

Allow the App to Add Todos

1. Go to **UI Scripts**.
2. Go to **app_angular_todo**.
3. Change the script to make the following changes:

```
angular.module('todo', ['sn.glideAjax']);
angular.module('todo').controller('TodoListCtrl', function($scope, GlideAjax) {
    $scope.todos = [];
    load();

    $scope.add = function(newTodo) {
        $scope.todos.push(newTodo);
    };

    function load() {
        var ga = new GlideAjax('TodoAppPersistence');
        ga.addParam('sysparm_name', 'getList');
        ga.setScope('x_angular_todo');
        ga.getJSON(function(response) {
            if (!response)
                return;

            $scope.todos = response;
        });
    }

});
```

Verification

1. Now view the UI Page at **x_angular_todo_angular_todo.do.** When the page and app load, you should see a **todo** for each of the records created.

Don't forget to Report Lab Progress!

Progress Report

1. Navigate to **Lab Management> Report Lab Progress**.
2. Click **I am Done!**

Lab Goal

This lab explains how to store new **Todos** on the server.

Lab 4.3 Persist New Todos

Add the Backend to Add New Todos

1. Go to **Script Includes**.
2. Open **TodoAppPersistence**.
3. Add the following to the **Script Include**:

```
var TodoAppPersistence = Class.create();
TodoAppPersistence.prototype = Object.extendsObject(global.AbstractAjaxProcessor, {
    getList: function() {
        var todos = new GlideRecord('x_angular_todo_list');
        todos.addQuery('opened_by', gs.getUserID());
        todos.orderBy('opened_at');
        todos.query();

        var list = [];
        while (todos.next()) {
            list.push(todos.short_description + '');
        }

        return new global.JSON().encodeArray(list);
    },


    add: function() {
    var todos = new GlideRecord('x_angular_todo_list');
        todos.newRecord();
        todos.short_description = this.getParameter('short_description');
        todos.opened_by = gs.getUserID();
        todos.insert();
    },

    type: 'TodoAppPersistence'
});
```

Add the Ability to Persist New Todos into the App

1. Go to **UI Scripts**.
2. Open **app_angular_todo**.
3. Add the following to the app to call the backend:

```
angular.module('todo', ['sn.glideAjax']);
angular.module('todo').controller('TodoListCtrl', function($scope, GlideAjax) {
    $scope.todos = [];
    load();

    $scope.add = function(newTodo) {
        $scope.todos.push(newTodo);

        var ga = new GlideAjax('TodoAppPersistence');
        ga.addParam('sysparm_name', 'add');
        ga.addParam('short_description', newTodo);
        ga.setScope('x_angular_todo');
        ga.getJSON(function(response) {
            // don't need to do anything here
        });
    };

    function load() {
        var ga = new GlideAjax('TodoAppPersistence');
        ga.addParam('sysparm_name', 'getList');
        ga.setScope('x_angular_todo');
        ga.getJSON(function(response) {
            if (!response)
                return;

            $scope.todos = response;
        });
    }
});
```

Verification

Now load the app at **x_angular_todo_angular_todo.do**. You should be able to do the following:

- Load the app
- Add new **Todos**
- Refresh the page and have your **Todos** persisted.

Lab Goal

The app is small because there is only a little markup. But at some point, you are going to want to use partials and pull them dynamically. This lab explains how to add a dependency to pull templates stored in UI Macros.

Lab 5.1 Patterns to Help the App Scale

Add a Service to Pull UI Macros as Angular Templates

1. Go to **UI Pages**.
2. Open the page **angular_todo**.
3. Add the following dependency to the page:

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <script src="x_angular_todo.angular_1_3_framework.jsdbx"></script>
    <script src="x_angular_todo.app_angular_todo.jsdbx"></script>
    <script src="service.sn-glideAjax.jsdbx"></script>
    <script src="service.uiMacroTemplate.jsdbx"></script>
    <div ng-app="todo">
        <div class="col-md-6" ng-controller="TodoListCtrl">
            <h1>Todo List</h1>
            <ul class="list-group">
                <li class="list-group-item" ng-repeat="todoItem in todos" ng-bind="todoItem"></li>
            </ul>
            <div>
                <label>Add new todo:
                    <input class="form-control" ng-model="newTodo"/>
                </label>
                <button ng-click="add(newTodo)" type="button" class="btn btn-primary">Add Todo</button>1.
            </div>
        </div>
    </div>
</j:jelly>
```

Add the Dependency to the App

1. Go to **UI Scripts**.
2. Open **app_angular_todo**.
3. Change the first line to update the todo app dependencies:

```
angular.module('todo', ['sn.glideAjax', 'sn.uiMacro']);
```

Use the Dependency in the App

To use the partial, convert the TodoList controller into a directive.

1. Go to UI Scripts.
2. Go to **app_angular_todo**.
3. Change the app into a directive by changing it like below:

```
angular.module('todo', ['sn.glideAjax', 'sn.uiMacro']);
angular.module('todo').directive('todoList', function(uiMacroTemplate) {
    return {
        restrict: 'E',
        templateUrl: uiMacroTemplate('x_angular_todo_todo_list'),
        controller: function($scope, GlideAjax) {
            $scope.todos = [];
            load();

            $scope.add = function(newTodo) {
                $scope.todos.push(newTodo);

                var ga = new GlideAjax('TodoAppPersistence');
                ga.addParam('sysparm_name', 'add');
                ga.addParam('short_description', newTodo);
                ga.setScope('x_angular_todo');
                ga.getJSON(function(response) {
                    alert(response);
                });
            };

            function load() {
                var ga = new GlideAjax('TodoAppPersistence');
                ga.addParam('sysparm_name', 'getList');
                ga.setScope('x_angular_todo');
                ga.getJSON(function(response) {
                    if (!response)
                        return;

                    $scope.todos = response;
                });
            }
        }
    }
});
```

Note how we use the new dependency. When **uiMacroTemplate** is called, we pass in the name of the UI macro. When the macro is passed into Angular's templateUrl, we can pull the partial on demand for the new directive.

Create the Partial as a UI Macro

Create the partial in the **UI Macros** table. Note that we can cut and paste this from the UI page, but remove the **ng-controller** attributes.

1. Go to **UI Macros**.
2. Click **New**.
3. Create a new **UI Macro** with the following values:

   Name: **todo_list**
   Script:

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<div class="col-md-6">
<h1>Todo List</h1>
<ul class="list-group">
<li class="list-group-item" ng-repeat="todoItem in todos" ng-bind="todoItem"></li>
</ul>
<div>
<label>Add new todo:
<input class="form-control" ng-model="newTodo"/>
</label>
<button ng-click="add(newTodo)" type="button" class="btn btn-primary">Add Todo</button>
</div>
</div>
</j:jelly>
```

Use the Directive on the Page

We have the partial and the directive, Now modify the UI Page to use the new directive. Since most of the HTML is now in the partial, update our UI Page.

1. Go to **UI Pages**.
2. Open **angular_todo**.
3. Update the page, removing the markup and replacing it with the directive:

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <script src="x_angular_todo.angular_1_3_framework.jsdbx"></script>
    <script src="x_angular_todo.app_angular_todo.jsdbx"></script>
    <script src="service.sn-glideAjax.jsdbx"></script>
    <script src="service.uiMacroTemplate.jsdbx"></script>
    <div ng-app="todo">
        <todo-list></todo-list>
    </div>
</j:jelly>
```