How To Create Single Page Application in minutes! with AngularJs and Yii 2.0

By Hafid Mukhlasin

Table of Contents

How To Create Single Page Application in minutes! with AngularJs 1.3 and Yii 2.0

Introduction

This is a demo and a tutorial showing how to develop an application using Yii 2.0 for creating REST API and then using it from UI built with AngularJS.

Tutorial is available here.
Book is available here

Installing demo

In order to install demo clone this repository:

```
git clone https://github.com/hscstudio/angular1-yii2 angular1-yii2
cd angular1-yii2
```

Then import database angular_spa.sql.

After it's done run the following:

```
cd web-service
composer update --prefer-dist
```

Set database config in web-service\config\db.php.

Set up two hosts in your webserver. One should point to `web-client`, aother to `web-service/web`.
Then set `serviceBase` variable in web-client\app.js to point to web-service URL.

License

Free open source

Changelog

- 0.1 (Alpha - 11/05/2015)
- 0.2 (Beta - 12/05/2015)

How To Create Single Page Application in minutes! with AngularJs 1.3 and Yii 2.0

Table Of Contents

Development

This guide is an OpenSource project so your contribution is very welcome.

In order to get started:

- Clone this repository.
- Check README.md.
- Read everything in `documentation`.
- Send pull requests.

Aside from contributing via pull requests you may submit issues.

Reference

- Official Yii 2.0 guide.
- Yii 2.0 source code.
- Example of AngularJS + Yii 2.0 by githubjeka.

Our Team

- Hafid Mukhlasin - Project Leader / Indonesian Yii developer.
- Alexander Makarov - Editorial / Yii core team.

We'd like to thank our contributors for improving the guide. Thank you!

Introduction

From this guide you'll learn how to create single page application (SPA) in minutes using AngularJs and Yii Framework 2.0. The application will implement a Create Read Update Delete (CRUD) data processing. User interface of this application will be implemented using AngularJs. Data will be provided by API created using Yii Framework 2.0.

Structure

There will be two applications:

- Web Client Application. The one providing UI.
- Web Service Application. The one dealing with data.

> Note: For easier maintenance it is recommended that you develop your RESTful APIs as a separate application which is separated from both website and admin parts of the application.

Technology Behind the Scenes

Since client and service are separated, technology stack used in each case varies.

Web Client Application

For client application we use HTML, JS, and CSS. At least some knowledge of all three is mandatory to follow this tutorial.

- AngularJs 1.3

  AngularJs is a popular JavaScript framework. It does not matter if you do not know too much about it yet. It is relatively easy to undestand if you're familiar with JavaScript and Yii. If you've used jQuery before, forget about it for a while since concepts of AngularJs are different.

- CSS Bootstrap 3

  
  Initially developed by Twitter team, CSS Bootstrap is widely used frontend UI framework. It is a collection of ready to use JavaScript and CSS that allows us to design a beautiful user interface quickly.

Web Service Application

For service part we'll use PHP and MySQL. As a PHP framework we'll use Yii Framework 2.0

Preparations

It's time to start preparing applications. The following are separate steps for client and service applications.

Web Client Application

Create a web root directory (usually it's `web`, `htdocs`, `public_html`, `www` or alike name). Inside add directories named `assets`, `controllers`, `models` and `views`.

The structure we've just created is similar to the structure used by Yii. You can adjust it as you like but for this tutorial we'll stick to Yii convention in order to make it easier to understand.

Let's explain all these directories a bit:

- `assets` contains AngularJs and CSS Bootstrap libraries.
- `controllers` is for AngularJs controllers.
- `models` is for services which deal with RESTful CRUD API we're going to build.
- `views` is for partial pages. Much like views in Yii.

Get CSS Bootstrap

Download a library from http://getbootstrap.com/ and extract it to `assets` directory like the following:

```
assets
    bootstrap
        js
        css
        font
```

Get AngularJs

Download a library from http://angularjs.org/, and extract it into `assets` directory like the following:

```
assets
    bootstrap
    angular
        angular.js
        angular.min.js
        ...
```

Include AngularJs and CSS Bootstrap into HTML

In order to use AngularJs and CSS Bootstrap we need to create an HTML file html file which will use both libraries. Create `index.html` and put it into your web root directory:

```
<!DOCTYPE html>
<html>
<head>
  <!-- CSS -->
  <link rel="stylesheet" href="assets/twitter-bootstrap/css/bootstrap.min.css" />
  <link rel="stylesheet" href="assets/twitter-bootstrap/css/bootstrap-theme.min.css" />
</head>
<body>

  <!-- JS -->
  <script src="assets/angular/angular.min.js"></script>
  <script src="assets/angular/angular-route.min.js"></script>

</body>

</html>
```

Web Service Application

Install Yii 2.0 Basic project template as described in Yii guide. It is preferred to use Composer like the following:

```
composer global require "fxp/composer-asset-plugin:1.0.0"
composer create-project --prefer-dist yiisoft/yii2-app-basic web-service
```

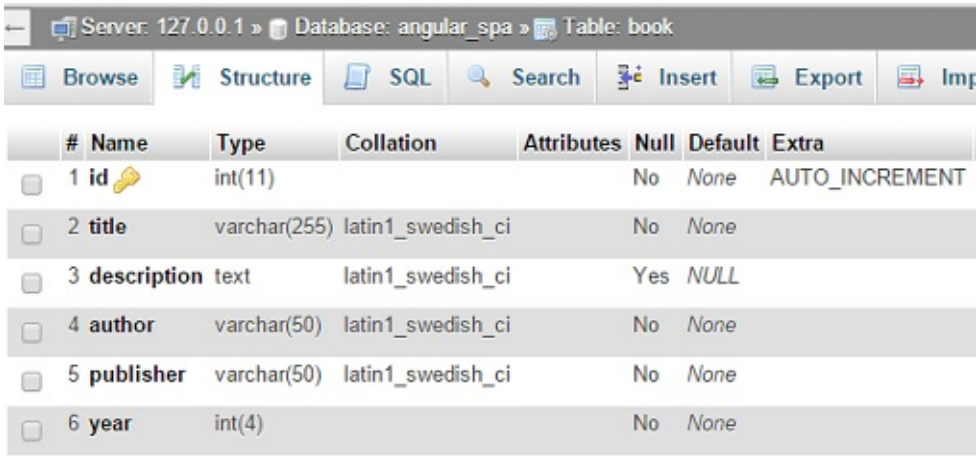
Alternatively you can do it manually downloading and extracting archive as described at the same guide page.

Creating Web Service

Now that prepartions done we'll start with service part of our application using high performance Yii 2.0 PHP framework. It's a great fit since it has built in functionality that allows kickstarting with RESTful APIs easily. Basics are already done for you ensuring many details to be set up correctly while more advanced things could be implemented as needed.

Create Database Structure

We'll use database named `angular_spa` so create it. Then add `book` table with following structure:



You can use angular_spa.sql in order to import it.

Insert some data in the table, you can create more tables but for the sake of simplicity, in this tutorial we'll use only `book`.

Configure Database Connection

Open config/db.php in the root of Yii web service application. Modify `db` configuration to specify your MySQL settings:

```
return [
    'class' => 'yii\db\Connection',
    'dsn' => 'mysql:host=localhost;dbname=angular_spa',
    'username' => 'root', // specify your username here
    'password' => '', // specify your password here
    'charset' => 'utf8',
];
```

Create Models

Use Gii to generate model class for database tables (we have only one so far). Refer to Yii's guide for details on how to do it.

You should get models/Book.php with the following content:

```
namespace app\models;
use Yii;

class Book extends \yii\db\ActiveRecord
{
    public static function tableName()
    {
        return 'book';
    }

    public function rules()
    {
        return [
            [['title', 'author', 'publisher', 'year'], 'required'],
            [['id', 'year'], 'integer'],
            [['title'], 'string', 'max' => 255],
            [['description'], 'string'],
            [['author','publisher'], 'string', 'max' => 50]
        ];
    }
}
```

Set up Yii RESTful Application

If you are not familiar with REST support in Yii 2.0, you can get and idea about it by reading corresponding guide section.

Create REST Controller

There's no ability to generate REST CRUD in Yii's Gii code generator but doing it without it is easy and fun.

First, create BookController.php in the `controllers` directory.

```
namespace app\controllers;

use yii\rest\ActiveController;

class BookController extends ActiveController
{
    // adjust the model class to match your model
    public $modelClass = 'app\models\Book';

    public function behaviors()
    {
        return
        \yii\helpers\ArrayHelper::merge(parent::behaviors(), [
            'corsFilter' => [
                'class' => \yii\filters\Cors::className(),
            ],
        ]);
```

```
    }
}
```

The controller class extends from `yii\rest\ActiveController`. By specifying `modelClass` as `app\models\Book`, the controller knows which model can be used for fetching and manipulating data.

We're adding CORS behavior in order to grant access to third party code (AJAX calls from external domain).

If you have more models, create alike controllers for each one.

Configuring URL Rules

Modify application configuration, `urlManager` component in web.php in the `config` directory:

```
'urlManager' => [
    'enablePrettyUrl' => true,
    'enableStrictParsing' => true,
    'showScriptName' => false,
    'rules' => [
        ['class' => 'yii\rest\UrlRule', 'controller' => 'book'],
    ],
]
```

The above configuration mainly adds a URL rule for the book controller so that the book data can be accessed and manipulated with pretty URLs and meaningful HTTP verbs. If you have more controllers, specify them as array:

```
    'rules' => [
        ['class' => 'yii\rest\UrlRule', 'controller' => ['book','user','employee','etc']]
    ],
```

> Note: If you are using Apache as a web server, you need to add a .htaccess file to your web root. In case of nginx you don't need to do it.

Enable JSON Input

To let the API accept input data in JSON format, configure the `parsers` property of the `request` application component to use the `yii\web\JsonParser`:

```
'request' => [
    'parsers' => [
        'application/json' => 'yii\web\JsonParser',
    ]
]
```

> Info: The above configuration is optional. Without it, the API would only recognize `application/x-www-form-urlencoded` and `multipart/form-data` input formats.

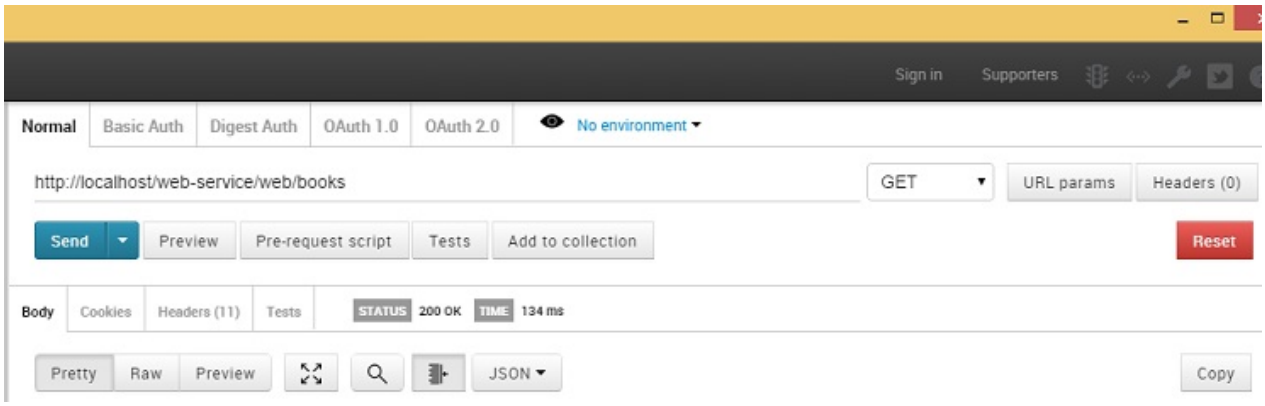
Summary

Not it's time to check what we've created so far with that little effort. We already have RESTful API for getting and managing book data:

```
GET /books: list all books page by page;
HEAD /books: show the overview information of book listing;
POST /books: create a new book;
GET /books/123: return the details of the book 123;
HEAD /books/123: show the overview information of book 123;
PATCH /books/123 and PUT /books/123: update the book 123;
DELETE /books/123: delete the book 123;
OPTIONS /books: show the supported verbs regarding endpoint /books;
OPTIONS /books/123: show the supported verbs regarding endpoint /books/123.
```

> Info: Yii will automatically pluralize controller names for use in endpoints. You can configure this using the `yii\rest\UrlRule::$pluralize`.

You may also access your API via Web browser by entering http://localhost/web-service/web/books. However, you may need some browser plugins to send specific request headers. For example, Postman Chrome Extension:



Creating Web Client

It is time to create a web client or, in other words, the user interface for the application service we've created previously. We will use AngularJs as JavaScript framework and CSS bootstrap to style our UI.

Setup Application Entry Script

Entry script is a script that handles all application requests. In our case it will be index.html file contained in the web directory.

Define Angular App

Add `ng-app` attribute to `html` tag. As an example, we'll use `spaApp` as the value.

```
<!DOCTYPE html>
<!-- define angular app -->
<html ng-app="spaApp">
<head>
  <!-- CSS -->
```

Define Default Angular Controller

Add `ng-controller` attribute to `body` tag. Controller name is `index`.

```
</head>
<!-- define angular controller -->
<body ng-controller="index">
```

Create Main Menu

Add the following content inside `body` tag:

```
<body ng-controller="index">
  <nav class="navbar navbar-default">
    <div class="container">
      <div class="navbar-header">
        <a class="navbar-brand" href="#/">Single Page Application</a>
      </div>
      <ul class="nav navbar-nav navbar-right">
        <li><a href="#/"><i class="glyphicon glyphicon-home"></i> Home</a></li>
        <li><a href="#/site/about"><i class="glyphicon glyphicon-tag"></i> About</a></li>
        <li><a href="#/site/contact"><i class="glyphicon glyphicon-envelope"></i> Contact
      </ul>
    </div>
  </nav>
  <div id="main" class="container">
```

```
    <!-- angular templating -->
        <!-- this is where content will be injected -->
    <div ng-view></div>
  </div>

  <footer class="text-center">
    <p>Yii 2.0.3 + AngularJs 1.3.15</p>
  </footer>
```

`Navbar` is used for menu, `main` is page container. `Footer` is, obviously, a footer.

Important thing here is an `id` of in div with `container` class:

```
<div id="main" class="container">
  <!-- angular templating -->
    <!-- this is where content will be injected -->
  <div ng-view></div>
</div>
```

Dynamic content from other files or page views will be placed into `<div ng-view></div>`.

Create Main Module and Sub Module

The main module is intended to control other scripts such as sub module. We name it app.js and place it into the webroot of the web client:

```
'use strict';
// adjust to the your url of web service
var serviceBase = 'http://127.0.0.1/web-service/web/'
// declare app level module which depends on views, and components
var spaApp = angular.module('spaApp', [
  'ngRoute',
  'spaApp.site',
]);
// sub module declaration
var spaApp_site = angular.module('spaApp.site', ['ngRoute'])

spaApp.config(['$routeProvider', function($routeProvider) {
  // config default route
  $routeProvider.otherwise({redirectTo: '/site/index'});
}]);
```

Default route is `/site/index`. This route will handled by `spaApp.site` sub module.

Create Sub Module Definition

After creating `spaApp.site` sub module we need to define what that sub module does. Create a file site.js in `controllers` directory.

```
'use strict';
spaApp_site.config(['$routeProvider', function($routeProvider) {
  $routeProvider
    .when('/site/index', {
        templateUrl: 'views/site/index.html',
        controller: 'index'
    })
    .when('/site/about', {
        templateUrl: 'views/site/about.html',
        controller: 'about'
    })
    .when('/site/contact', {
        templateUrl: 'views/site/contact.html',
        controller: 'contact'
    })
    .otherwise({
        redirectTo: '/site/index'
    });
}])
.controller('index', ['$scope', '$http', function($scope,$http) {
    // create a message to display in our view
    $scope.message = 'Everyone come and see how good I look!';
}])
.controller('about', ['$scope', '$http', function($scope,$http) {
    // create a message to display in our view
    $scope.message = 'Look! I am an about page.';
}])
.controller('contact', ['$scope', '$http', function($scope,$http) {
    // create a message to display in our view
    $scope.message = 'Contact us! JK. This is just a demo.';
}]);
```

This file is sub module to handle site views. It is very similar to Yii's `SiteController`:

```
spaApp_site.config(['$routeProvider', function($routeProvider) {
  $routeProvider
    .when('/site/index', {
        templateUrl: 'views/site/index.html',
        controller: 'index'
    })
    ...
    ...
    .otherwise({
        redirectTo: '/site/index'
    });
}])
```

This is routing configuration of this sub module only. Every route has may `templateUrl` and `controller`.

- `templateUrl` is an external HTML file that is used as partial content. Quite similar to Yii views.
- `controller` is a name of controller that prepares template data such as variables. It is simlar to what Yii controller does.

`.otherwise` tells the application what to do if no route matches.

```
.controller('index', ['$scope', '$http', function($scope,$http) {
    // create a message to display in our view
    $scope.message = 'Everyone come and see how good I look!';
}])
```

`$scope` is a scope that can be handled by the angular app in this case is all the tags under the tag which is marked with ng - app

$scope.message, message is variabel in file templateUrl, let say views/site/index.html, point to

Include Main Module and Sub Module

After creating main module `app.js` and sub module `site.js`, we must include it in an entry script of app index.html:

```
  <script src="assets/angular/angular-animate.min.js"></script>
  <!-- Include this js -->
  <script src="app.js"></script>
  <script src="controllers/site.js"></script>
</body>
```

Create Template File

Create a template file to be used by controller in views directory. The file name would be site/index.html:

```
<div class="jumbotron text-center">
    <h1>Home Page</h1>

    <p>{{ message }}</p>
</div>
```

Create site/contact.html:

```
<div class="jumbotron text-center">
    <h1>Contact Page</h1>

    <p>{{ message }}</p>
</div>
```

Create site/about.html:

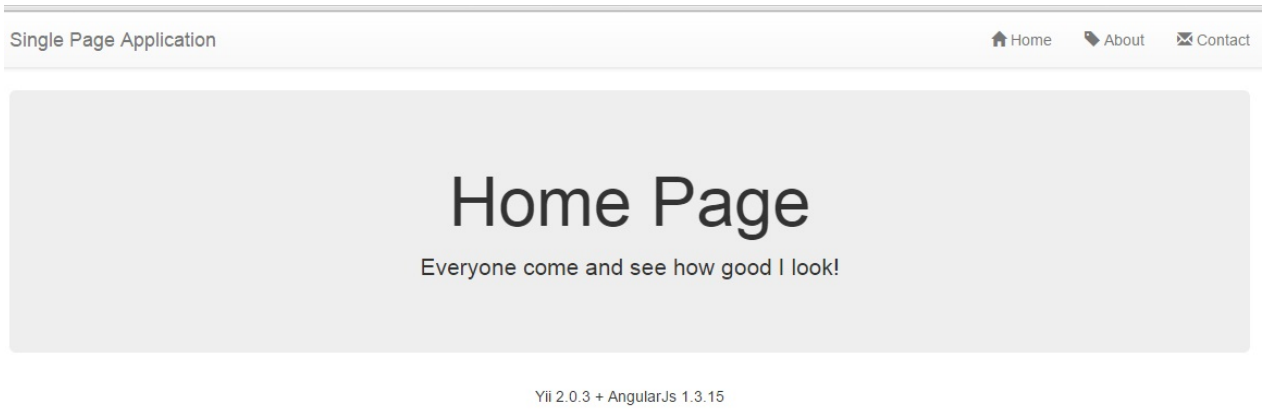
```
<div class="jumbotron text-center">
    <h1>About Page</h1>

    <p>{{ message }}</p>
```

```
</div>
```

These views are simple placeholders for now.

Test your application

http://localhost/web-client



Modify app.js

Add global JavaScript varable `serviceBase` that refers to your Yii 2.0 web service. Then add a sub module called `spaApp.book`:

```
'use strict';
var serviceBase = 'http://127.0.0.1/web-service/web/'
// Declare app level module which depends on views, and components
var spaApp = angular.module('spaApp', [
  'ngRoute',
  'spaApp.site',
  'spaApp.book',
]);
var spaApp_site = angular.module('spaApp.site', ['ngRoute'])
var spaApp_book = angular.module('spaApp.book', ['ngRoute']);

spaApp.config(['$routeProvider', function($routeProvider) {
  $routeProvider.otherwise({redirectTo: '/site/index'});
}]);
```

Create book.js in `models` directory

`book.js` will handle CRUD data provided by REST service is is pretty much what models are doing in Yii.

```
'use strict';
spaApp_book.factory("services", ['$http','$location','$route',
    function($http,$location,$route) {
    var obj = {};
    obj.getBooks = function(){
        return $http.get(serviceBase + 'books');
```

```
    }
    obj.createBook = function (book) {
        return $http.post( serviceBase + 'books', book )
            .then( successHandler )
            .catch( errorHandler );
        function successHandler( result ) {
            $location.path('/book/index');
        }
        function errorHandler( result ){
            alert("Error data")
            $location.path('/book/create')
        }
    };
    obj.getBook = function(bookID){
        return $http.get(serviceBase + 'books/' + bookID);
    }

    obj.updateBook = function (book) {
        return $http.put(serviceBase + 'books/' + book.id, book )
            .then( successHandler )
            .catch( errorHandler );
        function successHandler( result ) {
            $location.path('/book/index');
        }
        function errorHandler( result ){
            alert("Error data")
            $location.path('/book/update/' + book.id)
        }
    };
    obj.deleteBook = function (bookID) {
        return $http.delete(serviceBase + 'books/' + bookID)
            .then( successHandler )
            .catch( errorHandler );
        function successHandler( result ) {
            $route.reload();
        }
        function errorHandler( result ){
            alert("Error data")
            $route.reload();
        }
    };
    return obj;
}]);
```

There are multiple functions such as `obj.getBooks`, `obj.createBook`, etc. which are passing data to RESTful endpoints. For example, the following will get a list of the books using GET. See this guide.

```
obj.getBooks = function(){
        return $http.get(serviceBase + 'books');
    }
```

Create a book using POST:

```
obj.createBook = function (book) {
        return $http.post( serviceBase + 'books', book )
```

Update a book using PUT:

```
obj.updateBook = function (book) {
        return $http.put(serviceBase + 'books/' + book.id, book )
```

Create a controller for `Site` Sub Module

Create book.js in `controllers` directory. It will handle book views like Yii controller does:

```
'use strict';
spaApp_book.config(['$routeProvider', function($routeProvider) {
  $routeProvider
    .when('/book/index', {
        templateUrl: 'views/book/index.html',
        controller: 'index'
    })
    .when('/book/create', {
        templateUrl: 'views/book/create.html',
        controller: 'create',
        resolve: {
            book: function(services, $route){
                return services.getBooks();
            }
        }
    })
    .when('/book/update/:bookId', {
        templateUrl: 'views/book/update.html',
        controller: 'update',
        resolve: {
          book: function(services, $route){
            var bookId = $route.current.params.bookId;
            return services.getBook(bookId);
          }
        }
    })
    .when('/book/delete/:bookId', {
        templateUrl: 'views/book/index.html',
        controller: 'delete',
    })
    .otherwise({
        redirectTo: '/book/index'
    });
}]);

spaApp_book.controller('index', ['$scope', '$http', 'services',
    function($scope,$http,services) {
    $scope.message = 'Everyone come and see how good I look!';
    services.getBooks().then(function(data){
        $scope.books = data.data;
    });
    $scope.deleteBook = function(bookID) {
        if(confirm("Are you sure to delete book number: " + bookID)==true && bookID>0){
            services.deleteBook(bookID);
```

```
            $route.reload();
        }
    };
}])
.controller('create', ['$scope', '$http', 'services','$location','book',
    function($scope,$http,services,$location,book) {
    $scope.message = 'Look! I am an about page.';
    $scope.createBook = function(book) {
        var results = services.createBook(book);
    }
}])
.controller('update', ['$scope', '$http', '$routeParams', 'services','$location','book',
    function($scope,$http,$routeParams,services,$location,book) {
    $scope.message = 'Contact us! JK. This is just a demo.';
    var original = book.data;
    $scope.book = angular.copy(original);
    $scope.isClean = function() {
        return angular.equals(original, $scope.book);
    }
    $scope.updateBook = function(book) {
        var results = services.updateBook(book);
    }
}]);
```

Create Template File for Book Sub Module

Create template file that is pointed by controller in `book` sub module in views directory. The name is book/index.html:

```
<div>
    <h1>BOOK CRUD</h1>
    <p>{{ message }}</p>
    <div ng-show="books.length > 0">
    <a class="btn btn-primary" href="#/book/create">
        <i class="glyphicon glyphicon-plus"></i> Create
    </a>
    <table class="table table-striped table-hover">
    <thead>
    <th>Title</th>
    <th>Author</th>
    <th>Publisher</th>
    <th>Year</th>
    <th style="width:80px;">Action </th>
    </thead>
    <tbody>
        <tr ng-repeat="data in books">
            <td>{{data.title}}</td>
            <td>{{data.author}}</td>
            <td>{{data.publisher}}</td>
            <td>{{data.year}}</td>
            <td>
            <a class="btn btn-primary btn-xs" href="#/book/update/{{data.id}}">
                <i class="glyphicon glyphicon-pencil"></i>
            </a>
            <a class="btn btn-danger btn-xs" ng-click="deleteBook(data.id)">
```

```
                <i class="glyphicon glyphicon-trash"></i>
            </a>
            </td>
        </tr>
    </tbody>
    </table>
    </div>
    <div ng-show="books.length == 0">
        Empty
    </div>
</div>
```

Create book/create.html:

```
<div>
    <h1>BOOK CRUD</h1>

    <p>{{ message }}</p>
    <form role="form" name="myForm">
        <div class= "form-group" ng-class="{error: myForm.title.$invalid}">
            <label> Title </label>
            <div>
            <input name="title" ng-model="book.title" type= "text" class= "form-control"
            <span ng-show="myForm.title.$dirty && myForm.title.$invalid" class="help-inli
            </div>
        </div>
        <div class= "form-group">
            <label> Description </label>
            <div>
            <textarea name="description" ng-model="book.description" class= "form-control
            </div>
        </div>
        <div class= "form-group" ng-class="{error: myForm.author.$invalid}">
            <label> Author </label>
            <div>
            <input name="author" ng-model="book.author" type= "text" class= "form-control
            <span ng-show="myForm.author.$dirty && myForm.author.$invalid" class="help-in
            </div>
        </div>
        <div class= "form-group" ng-class="{error: myForm.publisher.$invalid}">
            <label> Publisher </label>
            <div>
            <input name="publisher" ng-model="book.publisher" type= "text" class= "form-c
            <span ng-show="myForm.publisher.$dirty && myForm.publisher.$invalid" class="h
            </div>
        </div>
        <div class= "form-group" ng-class="{error: myForm.year.$invalid}">
            <label> Year </label>
            <div>
            <input name="year" ng-model="book.year" type= "text" class= "form-control" pl
            <span ng-show="myForm.year.$dirty && myForm.year.$invalid" class="help-inline
            </div>
        </div>

        <a href="#/book/index" class="btn btn-default">Cancel</a>
        <button ng-click="createBook(book);"
                ng-disabled="myForm.$invalid"
```

```
                type="submit" class="btn btn-default">Submit</button>
    </form>
</div>
```

Create book/update.html:

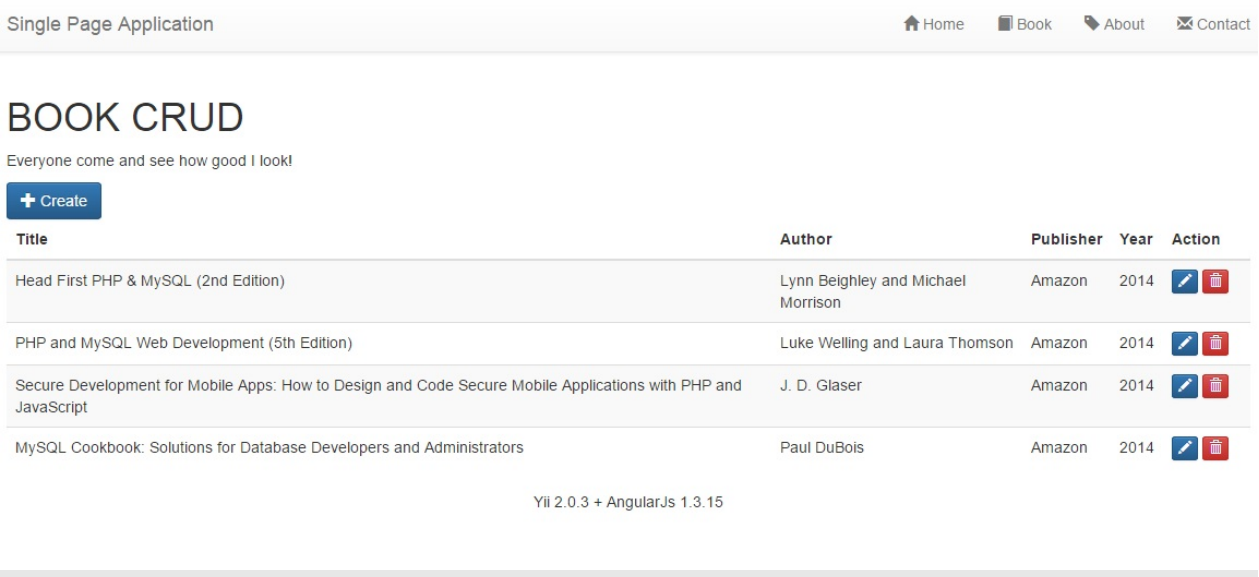
```
<div>
    <h1>BOOK CRUD</h1>

    <p>{{ message }}</p>
    <form role="form" name="myForm">
        <div class= "form-group" ng-class="{error: myForm.title.$invalid}">
            <label> Title </label>
            <div>
            <input name="title" ng-model="book.title" type= "text" class= "form-control"
            <span ng-show="myForm.title.$dirty && myForm.title.$invalid" class="help-inli
            </div>
        </div>
        <div class= "form-group">
            <label> Description </label>
            <div>
            <textarea name="description" ng-model="book.description" class= "form-control
            </div>
        </div>
        <div class= "form-group" ng-class="{error: myForm.author.$invalid}">
            <label> Author </label>
            <div>
            <input name="author" ng-model="book.author" type= "text" class= "form-control
            <span ng-show="myForm.author.$dirty && myForm.author.$invalid" class="help-in
            </div>
        </div>
        <div class= "form-group" ng-class="{error: myForm.publisher.$invalid}">
            <label> Publisher </label>
            <div>
            <input name="publisher" ng-model="book.publisher" type= "text" class= "form-c
            <span ng-show="myForm.publisher.$dirty && myForm.publisher.$invalid" class="h
            </div>
        </div>
        <div class= "form-group" ng-class="{error: myForm.year.$invalid}">
            <label> Year </label>
            <div>
            <input name="year" ng-model="book.year" type= "text" class= "form-control" pl
            <span ng-show="myForm.year.$dirty && myForm.year.$invalid" class="help-inline
            </div>
        </div>

        <a href="#/book/index" class="btn btn-default">Cancel</a>
        <button ng-click="updateBook(book);"
                ng-disabled="isClean() || myForm.$invalid"
                type="submit" class="btn btn-default">Submit</button>
    </form>
</div>
```

Modify Main Menu

Don't forget to add link to book CRUD.

```
<li><a href="#/book/index"><i class="glyphicon glyphicon-book"></i> Book</a></li>
```

Test it



Back To Index
01. Introduction
02. Preparation
03. Create Web Service
04. Create Web Client
05. Customization
06. Conclusion

Customization

This page still editing. You're very welcome to contribute.

Miscellaneous

For installation `assets` (javascript and css), we can use bower. Bower is a package manager for the web. It's quite with composer in PHP.

> You should drop all folders and files inside assets folder before using bower, because bower will download libraries for You by special structure.

Installation

Before installation bower, we must install:

- nodeJs & npm Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. NPM is package manager for javascript
- Git a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Installation Bower

Bower is a command line utility. Install it with npm.

```
npm install -g bower
```

Usage

Create file bower.json in folder web-client

```
{
  "name": "Angular1-Yii2",
  "version": "1.0.0",
  "homepage": "https://github.com/hscstudio/angular1-yii2",
  "description": "AngularJS 1.3 and Yii Framework 2.0",
  "dependencies": {
    "angular": "~1.3.0",
    "bootstrap": "~3.1.1",
    "angular-route": "~1.3.0"
  }
}
```

And then create file .bowerrc, this file contains configuration of bower, add parameter directory to specify target folder installation.

```
{
  "directory": "assets"
}
```

In command line, do this

```
cd web-client2
bower install
```

After installation finished, You can see folder assets have contained library angular, bootstrap, etc.

Include in Index

Because structure folder that downloaded by bower is different, we should adjust links js and css in file index.html.

```
  <!-- CSS -->
  <link rel="stylesheet" href="assets/bootstrap/dist/css/bootstrap.min.css" />
  <link rel="stylesheet" href="assets/bootstrap/dist/css/bootstrap-theme.min.css" />
```

```
  <!-- JS -->
  <script src="assets/angular/angular.min.js"></script>
  <script src="assets/angular-route/angular-route.min.js"></script>
```

Enhance User Interface

Angular Animation

There are several types of animation techniques that we can apply in our angularjs application. But in this tutorial I will discuss about the animation at the turn of the page. To do that, we need the ngAnimate module to enable animations throughout the application.

Add module angular-animate in bower.json

```
{
  ...
  ...
  "dependencies": {
    ...
    ...
    "angular-animate": "~1.3.0"
  }
}
```

And then do

```
bower update
```

Create style.css for define animation, for example:

```
.animate.ng-leave      {

}
.animate.ng-enter         {
    -webkit-animation:scaleUp 0.5s both ease-in;
    -moz-animation:scaleUp 0.5s both ease-in;
    animation:scaleUp 0.5s both ease-in;
}

/* scale up */
@keyframes scaleUp {
    from         { opacity: 0.3; transform: scale(0.8); }
}
@-moz-keyframes scaleUp {
    from         { opacity: 0.3; -moz-transform: scale(0.8); }
}
@-webkit-keyframes scaleUp {
    from         { opacity: 0.3; -webkit-transform: scale(0.8); }
}
```

- ng-enter : will attach when entering view
- ng-leave : when attach when leaving view

Include css animation in index.html:

```
<link rel="stylesheet" href="style.css" />
```

Add class animate in ng-view

```
<div id="main" class="container">
    <!-- angular templating -->
    <!-- this is where content will be injected -->
    <div ng-view class="animate"></div>
</div>
```

Include module ngAnimate in app.js:

```
var spaApp = angular.module('spaApp', [
  'ngRoute',
  'spaApp.site',
  'spaApp.book',
  'ngAnimate' // add module ngAnimate
]);
```

For further informations, read this:

- https://docs.angularjs.org/tutorial/step_12
- https://docs.angularjs.org/guide/animations
- https://docs.angularjs.org/api/ngAnimate

Flash Message

Angular Lazy Loader

https://oclazyload.readme.io/v1.0/docs/getting-started

When we add some module, we must add include script for sub module in main. By use module ocLazyLoad we can make lazy load in angular, include only when needed.

- Download ocLazyLoad.js (you can install it with bower install oclazyload or npm install oclazyload) and add the file to your project.
- Add the module oc.lazyLoad to your application:

```
var spaApp = angular.module('spaApp', [
'ngRoute',
'ngAnimate',
'spaApp.site',
'spaApp.book',
'oc.lazyLoad', // add this module lazyLoader
]);
```

- Load on demand:

```
spaApp.controller("MyCtrl", function($ocLazyLoad) {
$ocLazyLoad.load('testModule.js');
});
```

  With $ocLazyLoad you can load angular modules, but if you want to load any component (controllers / services / filters / ...) without defining a new module it's entirely possible (just make sure that you define this component within an existing module).

There are multiple ways to use $ocLazyLoad to load your files, just choose the one that you prefer.

Also don't forget that if you want to get started and the docs are not enough, see the examples in the 'examples' folder!

Customize RESTful API

Versioning

Customize URL

Error Handling

Authorization

Handling File Upload

Deploying Application

Before we deploy our application, we need do some things.

Web Client

Read official guide for application production click here.

Disabling Debug Data

By default AngularJS attaches information about binding and scopes to DOM nodes, and adds CSS classes to data-bound elements, but we can disable this in production for a significant performance boost with:

```
spaApp.config(['$compileProvider', function ($compileProvider) {
  $compileProvider.debugInfoEnabled(false);
}]);
```

Strict DI Mode

Using strict di mode in your production application will throw errors when a injectable function is not annotated explicitly. Strict di mode is intended to help you make sure that your code will work when minified. However, it also will force you to make sure that your injectable functions are explicitly annotated which will improve angular's performance when injecting dependencies in your injectable functions because it doesn't have to dynamically discover a function's dependencies. It is recommended to automate the explicit annotation via a tool like ng-annotate when you deploy to production (and enable strict di mode)

To enable strict di mode, you have two options:

```
<div ng-app="spaApp" ng-strict-di>
  <!-- your app here -->
</div>
```

or

```
angular.bootstrap(document, ['spaApp'], {
  strictDi: true
});
```

Web Service

Same with AngularJs, in production environments, we should disable debug mode. It may have a significant and adverse performance effect, besides that the debug mode may expose sensitive information to end users.

Modify file web/index.php in web-service, set `Yii_DEBUG` const to be `false`, and then `YII_ENV` to be `prod`.

```
defined('YII_DEBUG') or define('YII_DEBUG', false);
defined('YII_ENV') or define('YII_ENV', 'prod');
```

Conclusion

Yii 2.0 makes it much easier and faster to implement a web service as well as teaching us how to do it properly. The API created could be used easily from AngularJs or any other client. Overall, AngularJS and Yii 2.0 are a good match.

Happy coding!

Jakarta, Indonesia May, 12 2015

Hafid Mukhlasin

http://www.hafidmukhlasin.com