Contents

[**Deployd** 2](#_Toc450494286)

[Jhipster 全栈开发 (Angularjs + Sprint boot ) 3](#_Toc450494287)

[Install 3](#_Toc450494288)

[Basic knowledge 5](#_Toc450494289)

[如何实现异步函数？ 5](#_Toc450494290)

[AngularJS Resource：与 RESTful API 交互 7](#_Toc450494291)

[$httpProvider.interceptors 9](#_Toc450494292)

[Angular-translate 12](#_Toc450494293)

[angular-dynamic-locale 17](#_Toc450494294)

[angular-local-storage 17](#_Toc450494295)

[angular-ui-router 18](#_Toc450494296)

[21points dissect 19](#_Toc450494297)

[应用程序状态 19](#_Toc450494298)

[主页面： 22](#_Toc450494299)

[Register 23](#_Toc450494300)

[Activate 24](#_Toc450494301)

[Login 25](#_Toc450494302)

[Reset password request 27](#_Toc450494303)

[Reset password finish 28](#_Toc450494304)

[Setting (设置用户帐户信息) 29](#_Toc450494305)

[Change password 30](#_Toc450494306)

[Logout 31](#_Toc450494307)

[Authenticate 32](#_Toc450494308)

[Jhipster3.1.0 note 53](#_Toc450494309)

[how to use jhipster-jdl 53](#_Toc450494310)

[**angular-fullstack (MongoDB, ExpressJS, AngularJS, NodeJS)** 59](#_Toc450494311)

[**MEAN全栈开发 (MongoDB, ExpressJS, AngularJS, NodeJS)** 60](#_Toc450494312)

**CS/BS 全栈开发 Modern web application development**

Note: could not delete file “ because the path is too long?” (because npm nests dependencies)

**template> npm install –g rimraf**

**template> rimraf node\_modules**

**Deployd**

生成RESTful API based on database

<http://deployd.com/>

1. 创建web app框架

>dpd create sportstore & cd sportstore\public

2) 创建angular框架,写代码，构建

public>yo angular

public>grunt serve:dist

1. 启动web app

sportstore> dpd –p 5500 app.apd

localhost:5500\dist\index.html

# Jhipster 全栈开发 (Angularjs + Sprint boot )

## Install

若需要代理才能上网，需要设置三个地方：

env: HTTP\_PROXY and HTTPS\_PROXY (for git)

cmd>npm config set proxy … && npm config set https-proxy …

Intellij: proxy or gradle.properties

pre-condition: node, git

**set sys env: npm; git\bin**

// install tools for frontend

**>npm install –g yo bower grunt-cli gulp**

若无法下载，则设置proxy

> npm config set proxy <http://165.225.96.34:10015>

> npm config set https-proxy <http://165.225.96.34:10015>

(npm config get proxy, npm config delete proxy)

// install jhipster

> node --version && npm --version && git –version

> yo --version && bower --version && grunt --version

**> npm install –g generator-jhipster**

**>mkdir myApp && cd myApp**

**myApp > yo jhipster**

What is the base name of your application? sample

What is your default Java package name” com.philips.sample

Do you want to use Java 8? Yes

Which type of authentication would you like to use? HTTP Session Authentication (stateful,…)

Which type of database would you like to use? SQL

Which production database would you like to use? PostgreSQL

Which development database would like to use? PostgreSQL

Do you want to use Hibernate 2nd level cache? Yes, with encache (local cache, for a single node)

Do you want to use clustered HTTP sessions? No

Do you want to use WebSockets? No

Would you like to use Maven or Gradle for building the backend? Gradle

Would you like to use Grunt or Gulp.js for building the frontend? Grunt

Would you like to use the Compass CSS Authoring Framework? Yes

pdAmin3 for PostgreSQL

if fails, >npm install & bower install

// run server

myApp > gradlew

access to web exploer: <http://localhost:8080>

Features:

1. Spring boot (jpa, mongodb, cassandra, Spring security, spring mvc REST, Thymeleaf(server-side templates), monitoring), liquibase, Caching(Encache, HazelCast, Hibernate 2nd level cache or Spring Caching abstraction), cloud deployment
2. myApp > yo jhipster:entity foo

myApp > yo jhipster:service user

1. Spring Security includes Ajax endpoints, secured remember-me, audits…

Note:

若gradle无法下载gradle-bin, 然后gradle-bin无法下载spring-boot…

IDE: 若代理，配置proxy于Intellij or Eclipse

命令行: 若需要代理，则设置proxy于gradle.properties)

systemProp.http.proxyHost=165.225.96.34

systemProp.http.proxyPort=10015

systemProp.http.nonProxyHosts=\*.nonproxyrepos.com|localhost

systemProp.https.proxyHost=165.225.96.34

systemProp.https.proxyPort=10015

systemProp.https.nonProxyHosts=\*.nonproxyrepos.com|localhost

若无法连上数据库服务器，启动数据库服务器

若出现下载mongodb失败，是因为测试使用内置mondodb服务器，而下载需要设置代理（目前代码写死，或者可以配置我还没找到），只需在build.gradle注释de.flapdoodle.embed.mongo,这样spring boot就不会自动加载和注入内置mongodb服务器）

若SQL数据库

需要创建，如postgresql

username: postgresql

password: qzlin

创建数据库: resume

配置于src\main\resources\config\application-dev.yml

src\main\resources\config\application-prod.yml

datasource:

url: jdbc:postgresql://localhost:5432/resume

name:

username: postgres

password: qzlin

// run live-reload (via Gulp and Browsersync)

myApp > grunt serve

**myApp> gulp serve**

Features:

angular, bootstrap, bower, karma, grunt 都可以正常使用

Note:

1. bower代理：

若bower\_package无法找到，需要翻墙下载，设置bower代理，然后>bower install

set env for设置bower代理

**HTTP\_PROXY =** [**http://165.225.96.34:10015**](http://165.225.96.34:10015)

**HTTPS\_PROXY =** [**http://165.225.96.34:10015**](http://165.225.96.34:10015)

1. 若sass无法编译，需要安装sass.js, 先安装ruby,然后>gem install sass
2. 若git 别人的sample,>grunt报错unable to find local grunt, 则执行

>npm install grunt –save-dev

>npm install

d) 若port 被占用，可以改port端口，Gruntfile.js -> port: 9000 -> 9100

>yo jhipster:entity author

自动生成如下

//配置文件

.jhipster/Author.json

//后台

src/main/java/com.philips.bookstore/domain/Author

src/main/java/com.philips.bookstore/repository/AuthorRepository

src/main/java/com.philips.bookstore/web/AuthorResource

//数据库

src/main/resources/config/liquibase/master.xml

<include file="classpath:config/liquibase/changelog/20160101093953\_added\_entity\_Author.xml" relativeToChangelogFile="false"/>

src/main/resources/config/liquibase/changelog/20160101093953\_added\_entity\_Author.xml

pgAdmin III -> PostgreSQL 9.5 (localhost:5432) -> bookstore -> author

//前端

src/main/webapp/scripts/app/entities/author/…

src/main/webapp/scripts/components/entities/author/…

src/main/webapp/i18n/en/author.json

src/main/webapp/i18n/fr/author.json

//测试

src/test/java/com.philips.bookstore/web/rest/AthorResourceIntTest.java

src/test/javascript/spec/app/entities/author/author-detail.controller.spec.js

src/test/gatling/simlations/AuthorGatlingTest.scala

Basic knowledge

如何实现异步函数？

使用$q.defer().resolve(data), $q.defer().promise().then(function(data){})

$q: A service that helps you run functions asynchronously, and use their return values (or exceptions) when they are done processing.

Callback-style service

定义服务 （异步函数，用回调方式）

myModule.factory('helloWorld', function($timeout) {

var getMessages = function(callback) {

$timeout(function() {

callback(['Hello', 'world!']);

}, 2000);

};

return { getMessages: getMessages };

});

使用服务 （传入回调函数）

myModule.controller('HelloCtrl', function($scope, helloWorld) {

helloWorld.getMessages(function(messages) {

$scope.messages = messages;

});

});

Promise-style service

定义服务 （异步函数，用promise方式）

myModule.factory('helloWorld', function($q, $timeout) {

var getMessages = function() {

var deferred = $q.defer();

$timeout(function() {

deferred.resolve(['Hello', 'world!']);

}, 2000);

return deferred.promise;

};

return {

getMessages: getMessages

};

});

使用服务 （使用promise的接口）

myModule.controller('HelloCtrl', function($scope, helloWorld) {

helloWorld.getMessages().then(function(messages) {

$scope.messages = messages;

});

});

**若是angular的双向绑定，支持promise作为数据**

**//When Angular encounters a promise inside the view, it automatically sets up a success callback and substitutes the promise for the resulting value once it has been resolved**

myModule.controller('HelloCtrl', function($scope, helloWorld) {

$scope.messages = helloWorld.getMessages();

});

服务层：

angular.module('21pointsApp')

.factory('language', function ($q, $http, **$translate**, LANGUAGES) {

return {

getCurrent: function () {

var language = **$translate.storage().get('NG\_TRANSLATE\_LANG\_KEY');**

if (angular.isUndefined(language)) { language = 'en'; }

**var deferred = $q.defer();**

**deferred.resolve(language);**

**return deferred.promise;**

}

};

})

客户层：

Language.getCurrent().then(function (language) {

**$translate.use(language);**

});

AngularJS Resource：与 RESTful API 交互

Refer: <https://docs.angularjs.org/api/ngResource/service/$resource>

var app = angular.module('helloApp, [**'ngResource'**]);

app.factory('bloodPressure', function (**$resource**, DateUtils) {

return $resource('api/bloodPressures/:id', {}, {

'byMonth': { method: 'GET', isArray: false, url: 'api/bp-by-month/:month'},

'last30Days': { method: 'GET', isArray: false, url: 'api/bp-by-days/30'},

'get': { method: 'GET',

transformResponse: function (data) {

data = angular.fromJson(data);

data.timestamp = DateUtils.convertDateTimeFromServer(data.timestamp);

return data;

}

},

'update': { method:'PUT' }

});

});

$resource提供默认5种actions:

{ 'get': { method: 'GET' },

'save': { method: 'POST' },

'query': { method: 'GET', isArray:true },

'remove': { method: 'DELETE' },

'delete': { method: 'DELETE' }

};

It is important to realize that invoking a $resource object method **immediately returns an empty reference (object or array depending on isArray)**. Once the data is returned from the server the existing reference is populated with the actual data. This is a useful trick since **usually the resource is assigned to a model** which is then rendered by the view. Having an empty object results in no rendering, once the data arrives from the server then the object is populated with the data and the view automatically re-renders itself showing the new data. This means that **in most cases one never has to write a callback function for the action methods**.

资源的操作有两种种调用方式：

通过资源类

HTTP GET "class" actions: Resource.action([parameters], [success], [error])

non-GET "class" actions: Resource.action([parameters], postData, [success], [error])

for example:

//不需要处理回调函数：绑定视图模型,等异步获取数据完，再次渲染视图

$scope.bloodPressure = bloodPressure.get({id : $stateParams.id});

// 需要处理回调函数

$scope.bloodPressures = [];

bloodPressure.query({page: $scope.page, per\_page: 20}, function(result, headers) {

$scope.links = ParseLinks.parse(headers('link'));

for (var i = 0; i < result.length; i++) {

$scope.bloodPressures.push(result[i]);

}

});

bloodPressure.get({id: id}, function (result) {

$scope.bloodPressure = result;

});

$scope.save = function () {

if ($scope.bloodPressure.id != null) {

bloodPressure.update($scope.bloodPressure, onSaveFinished);

} else {

BloodPressure.save($scope.bloodPressure, onSaveFinished);

}

};

// communication between edit resource and view resource

bloodPressure-dialog.controller.js

var onSaveFinished = function (result) {

$scope.$emit('21pointsApp:bloodPressureUpdate', result);

};

bloodPressure-detail.controller.js

$rootScope.$on('21pointsApp:bloodPressureUpdate', function(event, result) {

$scope.bloodPressure = result;

});

通过资源实例

non-GET instance actions: instance.$action([parameters], [success], [error])

Calling these methods invoke an $http with the specified http method, destination and parameters. When the data is returned from the server then the object is an instance of the resource class. The actions save, remove and delete are available on it as methods with the $ prefix. This allows you to easily perform CRUD operations (create, read, update, delete) on server-side data

For example:

var bloodPressures = bloodPressure.query({page: $scope.page, per\_page: 20}, function(result, headers) {

var bloodPressure = bloodPressure[0] ;

bloodPressure**.$save()** ;

bloodPressure**.$get()** ; //重新载入

bloodPressure**.$delete()** ;

};

new BloodPressure(bloodPressure).$save() ;

若需要处理回调函数，均可以在$action().then(function(bloodPressure) {…} ) ;

$httpProvider.interceptors

Refer to: <https://docs.angularjs.org/api/ng/service/$http>

The $http API is based on the deferred/promise APIs exposed by the $q service

For purposes of **global error handling, authentication, or any kind of synchronous or asynchronous pre-processing of request or postprocessing of responses**, it is desirable to be able to intercept requests before they are handed to the server and responses before they are handed over to the application code that initiated these requests.

There are two kinds of interceptors (and two kinds of rejection interceptors):

**request**: interceptors get called with a http **config** object. The function is free to modify the config object or create a new one. The function needs to return the config object directly, or a promise containing the config or a new config object.

**requestError**: interceptor gets called when a previous interceptor threw an error or resolved with a rejection.

**response**: interceptors get called with http **response** object. The function is free to modify the response object or create a new one. The function needs to return the response object directly, or as a promise containing the response or a new response object.

**responseError**: interceptor gets called when a previous interceptor threw an error or resolved with a rejection.

$httpProvider.interceptors.push('errorHandlerInterceptor');

$httpProvider.interceptors.push('authExpiredInterceptor');

$httpProvider.interceptors.push('authInterceptor');

$httpProvider.interceptors.push('notificationInterceptor');

.factory('errorHandlerInterceptor', function ($q, $rootScope) {

return {

**'responseError'**: function (**response**) {

if (!(response.status == 401 && response.data.path.indexOf("/api/account") == 0 )){

$rootScope.$emit('21pointsApp.httpError', response);

}

return **$q.reject(response)**;

}};

});

.factory('authInterceptor', function ($rootScope, $q, $location, localStorageService) {

return {

// Add authorization token to headers

**request**: function (**config**) {

config.headers = config.headers || {};

var token = localStorageService.get('token');

if (token && token.expires && token.expires > new Date().getTime()) {

config.headers['x-auth-token'] = token.token;

}

return **config**;

}};

})

.factory('authExpiredInterceptor', function ($rootScope, $q, $injector, localStorageService) {

return {

**responseError**: function (**response**) {

// token has expired

if (response.status === 401 && (response.data.error == 'invalid\_token' || response.data.error == 'Unauthorized')) {

localStorageService.remove('token');

var Principal = $injector.get('Principal');

if (Principal.isAuthenticated()) {

var Auth = $injector.get('Auth');

Auth.authorize(true);

}

}

return **$q.reject(response)**;

}};

});

.factory('notificationInterceptor', function ($q, AlertService) {

return {

**response**: function(**response**) {

var alertKey = response.headers('X-21pointsApp-alert');

if (angular.isString(alertKey)) {

AlertService.success(alertKey, { param : response.headers('X-21pointsApp-params')});

}

return **response**;

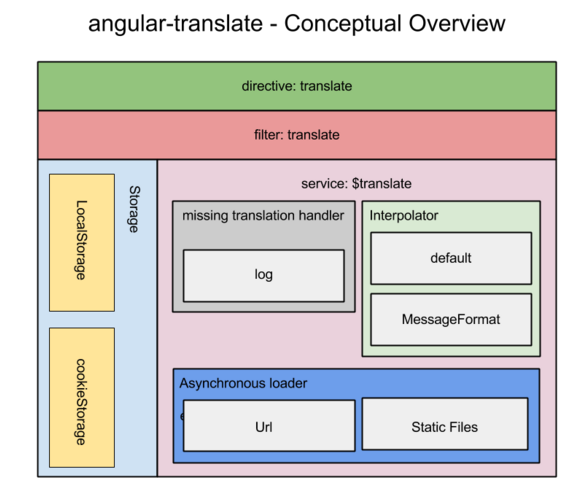
}};

});

Angular-translate

Refer to : <http://angular-translate.github.io/>

>bower install - -save angular-translate



var app = angular.module('myApp', ['**pascalprecht.translate**']);

$translateProvider全局配置$translate Service（因为是单例对象）, 然后使用$translate

$translateProvider.useLoader('$translatePartialLoader', {

urlTemplate: 'i18n/{lang}/{part}.json'

});

$translateProvider.preferredLanguage('en'); //或者采用自动查找浏览器

// try to find out preferred language by yourself, 搜索顺序:

It searches for values in the window.navigator object in the following properties (also in this order):

navigator.languages[0]

navigator.language

navigator.browserLanguage

navigator.systemLanguage

navigator.userLanguage

$translateProvider.determinePreferredLanguage();

// switch the language at runtime

$scope.changeLanguage = function (langKey) {

$translate.use(langKey);

};

// fallback languages: 若指定key在指定的语言中找不到，则从默认语言找

$translateProvider.fallbackLanguage(['en', 'fr']);

//Language Negotiation: 若自动化自动language key,一般是从浏览器获得，而浏览器language key很具体，需要进行映射，如若浏览器使用en\_US，则app应用en

$translateProvider

.translations('en', { /\* ... \*/ })

.translations('de', { /\* ... \*/ })

.registerAvailableLanguageKeys(['en', 'de'], {

'en\_US': 'en',

'en\_UK': 'en',

'de\_DE': 'de',

'de\_CH': 'de'

})

.determinePreferredLanguage();

// Storages: To let you app **remember the language users choose**, angular-translate comes with a support for Storages. Whatever storage you use, angular-translate will save a language key with a specific identifier in it, so it can ask for it next time the user launches the app. angular-translate has built-in support for two storages. localStorage and cookieStorage. Whereas localStorage falls back to cookieStorage if it isn't supported by the browser the user currently uses

>$ bower install angular-translate-storage-cookie

var module = angular.module('AppService', ['pascalprecht.translate','ngCookies']);

$translateProvider.useCookieStorage();

>$ bower install angular-translate-storage-local

$translateProvider.useLocalStorage();

浏览器会在local storage or cookie storage自动记录key-value,

{

**"NG\_TRANSLATE\_LANG\_KEY": en**

}

当然也可以获取：var language = $translate.storage().get('NG\_TRANSLATE\_LANG\_KEY');

// asynchronous loading: load your data from a server you have to use an asynchronous loader, which gets invoked later at runtime when it's needed. angular-translate comes with support for three different asynchronous loaders:

a. Using urlLoader

>$ bower install angular-translate-loader-url

$translateProvider.useUrlLoader('foo/bar.json');

$translateProvider.preferredLanguage('en');

actually requests foo/bar.json?lang=en

b. Using staticFilesLoader

>$ bower install angular-translate-loader-static-files

$translateProvider.useStaticFilesLoader({

prefix: 'locale-',

suffix: '.json'

});

$translateProvider.preferredLanguage('en');

angular-translate will concatenate the given information to {{prefix}}{{langKey}}{{suffix}}. So this will load locale-en.json.

c. Using partialLoader

>$ bower install angular-translate-loader-partial

// Initialize angular-translate

$translateProvider.useLoader('$translatePartialLoader', {

urlTemplate: 'i18n/{lang}/{part}.json'

});

$translateProvider.preferredLanguage('en');

//tell angular-translate which part you want to load when.

.state('bloodPressure', {

resolve: {

translatePartialLoader: ['$translate', '$translatePartialLoader', function (**$translate, $translatePartialLoader**) {

**$translatePartialLoader.addPart('bloodPressure');**

**$translatePartialLoader.addPart('global');**

**return $translate.refresh();**

}]

}

})

// Pluralization

MessageFormat: simple variable replacement, SelectFormat, and PluralFormat.

Please refer to:

<http://userguide.icu-project.org/formatparse/messages>, <https://github.com/SlexAxton/messageformat.js>

>$ bower install angular-translate-interpolation-messageformat

$translateProvider.addInterpolation('$translateMessageFormatInterpolation');

our app is configured using the default interpolation, but is also aware that there's another interpolation service that could be used for specific translations

for example:

var app = angular.module('myApp', ['pascalprecht.translate']);

app.config(['$translateProvider', function ($translateProvider) {

$translateProvider.preferredLanguage('en');

$translateProvider.addInterpolation('$translateMessageFormatInterpolation');

$translateProvider.translations('en', {

HEADLINE: 'I\'m a headline',

TEXT: 'I\'m using default interpolation {{ val + val }}',

PLURAL: '{GENDER, select, male{He} female{She} other{They}} liked this.',

});

$translateProvider.translations('de', {

TEXT: 'Ich benutze default interpolation {{ val + val }}',

PLURAL: '{GENDER, select, male{Er fand} female{Sie fand} other{Sie fanden}} es gut.',

});

}]);

<p translate="TEXT" translate-values="{ val: 5 }"></p>

<p translate="PLURAL" translate-values="{ GENDER: 'other' }" translate-interpolation="messageformat"></p>

result in en version:

I'm using default interpolation 10

They liked this.

result in de version:

Ich benutze default interpolation 10

Sie fanden es gut.

Security

$translateProvider.useSanitizeValueStrategy('sanitize');

the following strategies are built-in:

**sanitize**: sanitizes HTML in the translation text using $sanitize 净化

**escape**: escapes HTML in the translation 转义

sanitizeParameters: sanitizes HTML in the values of the interpolation parameters using $sanitize

escapeParameters: escapes HTML in the values of the interpolation parameters

webapp/i18n/en/global.json:

//key: 层次的，可引用内容的

{

"global": {

"foo": {

"foo": "This is my text."

},

"SOME\_NAMESPACE": {

"OK\_TEXT": "OK"

},

"ANOTHER\_NAMESPACE": {

"OK\_TEXT": "@:global.SOME\_NAMESPACE.OK\_TEXT"

}

}

}

**3种使用方式：translate directive, translate filter and $translate service**

Directive:

<h2 **translate=**"21pointsApp.bloodPressure.home.title">Blood Pressures</h2>

<td><span translate="21pointsApp.bloodPressure.timestamp">Timestamp</span></td>

Filter:

<input type="text" class="form-control" ng-model="searchQuery" id="searchQuery" placeholder="{{'entity.action.search' **| translate**}}">

Service:

app.controller('Ctrl', ['$scope', '$translate', function ($scope, $translate) {

**$translate**(['HEADLINE', 'PARAGRAPH', 'NAMESPACE.PARAGRAPH']).then(function (**translations**) {

$scope.headline = translations.HEADLINE;

$scope.paragraph = translations.PARAGRAPH;

$scope.namespaced\_paragraph = translations['NAMESPACE.PARAGRAPH'];

});

}]);

字符串格式化

{

"TRANSLATION\_ID": "{{username}} is logged in."

}

angular.module('myApp').controller('Ctrl', ['$scope', function ($scope) {

$scope.translationData = {

username: 'PascalPrecht'

};

}]);

{{ 'TRANSLATION\_ID' | translate: translationData }}

<ANY **translate**="TRANSLATION\_ID" **translate-values="{**username: translationData.username}"> </ANY>

angular-dynamic-locale

<https://scotch.io/tutorials/internationalization-of-angularjs-applications>

<https://github.com/lgalfaso/angular-dynamic-locale>

angular-translate (it’s used for handling **language translation** stuff)

angular-dynamic-locale (it’s used for **changing angular $locale- which means formatting dates, numbers, currencies, etc.**) libraries.

>bower install --save angular-dynamic-locale

angular.module('myApp').controller('myController', [..., **'tmhDynamicLocale'**,

function(..., **tmhDynamicLocale**) {

**tmhDynamicLocaleProvider.localeLocationPattern('bower\_components/angular-i18n/angular-locale\_{{locale}}.js');**

**tmhDynamicLocaleProvider.useCookieStorage();**

**tmhDynamicLocaleProvider.storageKey('NG\_TRANSLATE\_LANG\_KEY');**

}

])

angular-local-storage

$ bower install angular-local-storage --save

var myApp = angular.module('myApp', [**'LocalStorageModule'**]);

myApp.config(function (**localStorageServiceProvider**) {

//You could set a prefix to avoid overwriting any local storage variables from the rest of your app, Default prefix: ls.<your-key>

localStorageServiceProvider.setPrefix('yourAppName');

//You could change web storage type to localStorage or sessionStorage, Default storage: localStorage

localStorageServiceProvider.setStorageType('sessionStorage');

//Set cookie options (usually in case of fallback), expiry: number of days before cookies expire (0 = does not expire). default: 30; path: the web path the cookie represents. default: '/'

localStorageServiceProvider.setStorageCookie(45, '<path>');

//Send signals for each of the following actions: setItem , default: true; removeItem , default: false

localStorageServiceProvider.setNotify(true, true);

localStorageService.isSupported;

var storageType = localStorageService.getStorageType(); //e.g localStorage

//Directly add/get a value to local storage.If local storage is not supported, use cookies instead.

**localStorageService.set(key, val);**

**localStorageService.get(key);**

localStorageService.keys();

localStorageService.remove(key1, key2, key3, ...);

localStorageService.clearAll();

localStorageService.clearAll(/^\d+$/); // clear numbers

localStorageService.set('property', 'oldValue');

localStorageService.deriveKey('property'); // ls.property

// Return localStorageService.length, ignore keys that not owned.

var lsLength = localStorageService.length();

});

angular-ui-router

<https://github.com/angular-ui/ui-router>

>bower install -- save angular-ui-router

**State:**

**change application view based on application state**, including nested states (for nested views) and multiple named views

**Activating a state:**

* **Click a link containing the ui-sref directive**

ui-sref: create a link, point to a certain state of your application)

* **$state.go()**
* **Navigate to the url associated with the state**

**Controller:**

The controller will not be instantiated if template is not defined.

**Parent.Child state:**

* dot syntax to infer your hierarchy to the $stateProvider
* **parent state must exist**.
* No two states can have the same name.
* When a state is “active”, all of its ancestor states are implicitly active as well
* **Child state will load their templates into their parent’s ui-view** (When a state is activated, its templates are automatically inserted into the ui-view of its parent state’s template)
* Abstract state can have child states but can not get activated itself, abstract state still need their own <ui-view/> for their children to plug into
* **register states in any order and across modules, you can register children before the parent state exists. It will queue them up and once the parent state is registered then the child will be registered.**
* Child states DO inherit the following from parent states: **resolved dependencies, custom data properties,** children of abstract states do inherit the **url** property of their parent as a prefix of their own url

**Resolve:**

use resolve to provide your controller with content or data that is custom to the state, **If** any of these **dependencies are promises, they will be resolved and converted to a value before the controller is instantiated** and the $stateChangeSuccess event is fired.

**data:**

attach custom data to the state object (we recommend using a data property to avoid conflicts)

**onEnter and onExit:**

'onEnter' and 'onExit' callbacks that get called when a state becomes active and inactive respectively. The callbacks also have access to all the resolved dependencies.

**State Change Events:**

//fired when the transition begins.

$rootScope.$on('$**stateChangeStart'**, function(event, toState, toParams, fromState, fromParams){ ... })

//fired once the state transition is complete.

$rootScope.$on('$**stateChangeSuccess'**, function(event, toState, toParams, fromState, fromParams){ ... })

Multiple Named Views: (multiple ui-views per template)

**view name**: is the name used in the view directive inside parent’s html

**state name**: is the state's absolute name, 若没有statename，指index.html

relative names: **viewname -----(statename 默认为当前状态)**

absolute names: **viewname@statename ----(current state will plug into ui-view=”viewname” within statename所指定的html里)**

## 21points dissect

### 应用程序状态

$stateProvider.state('site', {

**'abstract': true,**

views: { //说明navbar子视图插入到url=/,即插入到state=site.home中

'**navbar@**': {

templateUrl: 'scripts/components/navbar/navbar.html',

controller: 'NavbarController'

}

},

resolve: { //controller的依赖项

authorize: ['Auth', function (Auth) { return Auth.authorize(); } ],

translatePartialLoader: ['$translate', '$translatePartialLoader', function ($translate, $translatePartialLoader) { $translatePartialLoader.addPart('global'); }]

}

});

$stateProvider.state('home', {

**parent: 'site',**

**url: '/', //入口/(url) = site.home (state)**

data: { roles: [] },

views: {

'**content@**': { //content子视图插入到url=/index.html 等价于state=site.home

templateUrl: 'scripts/app/main/main.html',

controller: 'MainController'

}

},

resolve: {

mainTranslatePartialLoader: ['$translate', '$translatePartialLoader', function ($translate, $translatePartialLoader) {

$translatePartialLoader.addPart('main');

$translatePartialLoader.addPart('weight');

return $translate.refresh();

}],

points: function(Points) { return Points.thisWeek().$promise; }

}

})

$stateProvider.state('about', {

**parent: 'site',**

**url: '/about', //入口/#/about (url) == site.about (state)**

data: { roles: [] },

views: {

**'content@'**: { //content子视图插入到url=/index.html 等价于state=site.home

templateUrl: 'scripts/app/about/about.html'

}

},

resolve: {

aboutTranslatePartialLoader: ['$translate', '$translatePartialLoader', function ($translate, $translatePartialLoader) {

$translatePartialLoader.addPart('main');

return $translate.refresh();

}]

}

});

$stateProvider.state('register', {

**parent: 'account',**

**url: '/register', //入口/#/register(url) == site.account.register (state)**

data: { roles: [], pageTitle: 'register.title' },

views: {

'content@': { //content子视图插入到url=/index.html等价于state=site.home

templateUrl: 'scripts/app/account/register/register.html',

controller: 'RegisterController'

}

},

resolve: {

translatePartialLoader: ['$translate', '$translatePartialLoader', function ($translate, $translatePartialLoader) {

$translatePartialLoader.addPart('register');

return $translate.refresh();

}]

}

});

切换应用程序状态方式：

**用户输入url, 或者点击ui-sref=”statename”,或者$state.go(‘statename’);**

应用程序状态注册方式angular.module(‘21pointsApp’).config(function($stateProvider){

$stateProvider.state(‘statename1’, {}).state(‘statename2’, {})…

}) ;

**类似依赖注入，所有应用程序状态会被扫描到**

/index.html = ui-view="navbar" + ui-view="content"

components/navbar/navbar.html + navbar.controller.js + navbar.directive.js

若site.home = /, 则content = app/main/main.html + main.controller.js

若site.about = /about, 则content = app/about/about.html

若site.accout.register = /register, 则content = app/account/register/register.html + register.controller.js

若site.account.activate = /activate?key, 则content = app/account/activate/activate.html + activate.controller.js

若site.account.login = /login, 则content = app/account/login/login.html + login.controller.js

若site.account.requestReset = /reset/request,

则content = app/account/reset/request/reset.request.html + reset.request.controller.js

若site.account.finishReset = /reset/finish?key,

则content = app/account/reset/finish/reset.finish.html + reset.finishcontroller.js

若site.account.logout = /logout, 则content = app/main/main.html + logout.controller.js

若site.account.settings = /settings, 则content = app/account/settings/settings.html + settings.controller.js

若site.account.password = /password, 则content = app/account/password/password.html + password.controller.js + password.directive.js

**navbar.html (导航栏 元素显示与隐藏)**

<div class="collapse navbar-collapse" id="navbar-collapse" **ng-switch="isAuthenticated()"**>

<ul class="nav navbar-nav nav-pills navbar-right">

<li ui-sref-active="active"><a ui-sref="home">Home</span></a></li>

*//ng-switch=false,则含ng-switch-when=true的元素隐藏, ng-switch-when=false的元素显示*

<li ui-sref-active="active" **ng-switch-when="true"** class="dropdown pointer">

<a class="dropdown-toggle" data-toggle="dropdown" href="">Entities</a>

<ul class="dropdown-menu">

<li ui-sref-active="active" ><a ui-sref="bloodPressure">bloodPressure</span></a></li>

...

</ul>

</li>

*//has-role, if current user has ROLE\_ADMIN, then show*

<li ng-class="{active: $state.includes('admin')}" ng-switch-when="true" **has-role="ROLE\_ADMIN"** class="dropdown pointer">

<a class="dropdown-toggle" data-toggle="dropdown" href="" id="admin-menu">Administration</a>

<ul class="dropdown-menu">

<li ui-sref-active="active"><a ui-sref="docs">API</a></li>

...

</ul>

</li>

*//ng-switch=true,则含ng-switch-when=true的元素显示, ng-switch-when=false的元素隐藏*

<li ng-class="{active: $state.includes('account')}" class="dropdown pointer">

<a class="dropdown-toggle" data-toggle="dropdown" href="" id="account-menu">Account</a>

<ul class="dropdown-menu">

<li ui-sref-active="active" ng-switch-when="true"><a ui-sref="settings">Settings</a></li>

<li ui-sref-active="active" ng-switch-when="true"><a ui-sref="password">Password</a></li>

<li ui-sref-active="active" ng-switch-when="true"><a href="" ng-click="logout()" id="logout">

Log out</a></li>

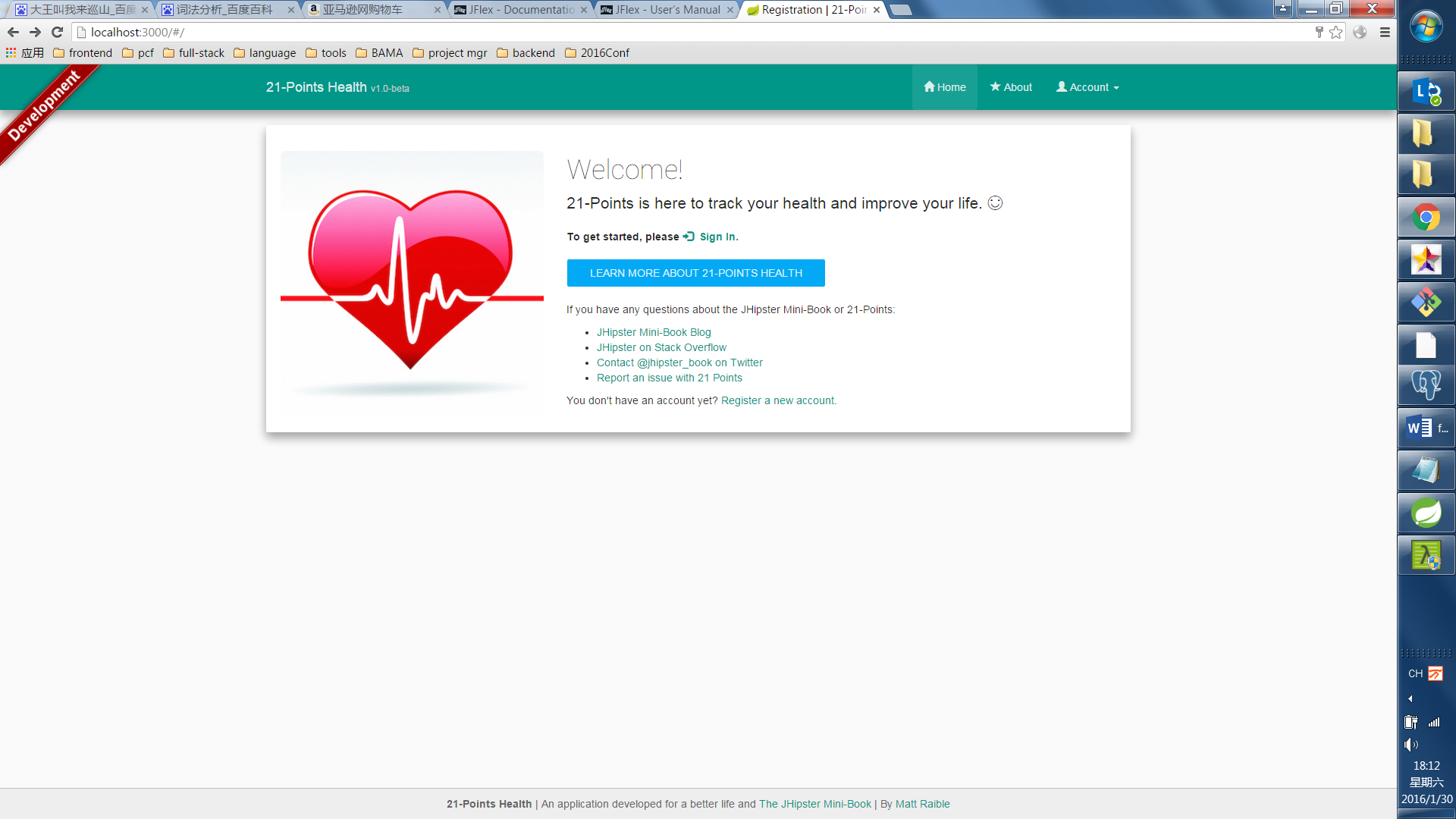
<li ui-sref-active="active" ng-switch-when="false"><a ui-sref="login">Authenticate</a></li>

<li ui-sref-active="active" ng-switch-when="false"><a ui-sref="register">Register</a></li>

</ul>

</li>

### 主页面：



用户在浏览器输入/, 显示主页面index.html, 含navbar+content+footer

navbar: 状态定义于app/app.js, 视图控制器自定义指令于components/navbar/navbar.html, navbar.controller.js, navbar.directive.js

content: 初始状态为site.home (url=/),状态视图控制器定义于app/main/main.js, main.html, main.controller.js

footer: 直接内嵌于主页面中

home页面主内容main.html=左边div占4列 + 右边div占8列。右边div分未登录的界面内容和已登录的界面内容

<div class=”col-md-8”>

<div ng-switch=”isAuthenticated()”>

<div ng-switch-when=”true”>…</div>

<div ng-switch-when=”false”>…</div>

</div>

</div>

Files:

index.html,

app/app.js

components/navbar/navbar.html

components/navbar/navbar.controller.js

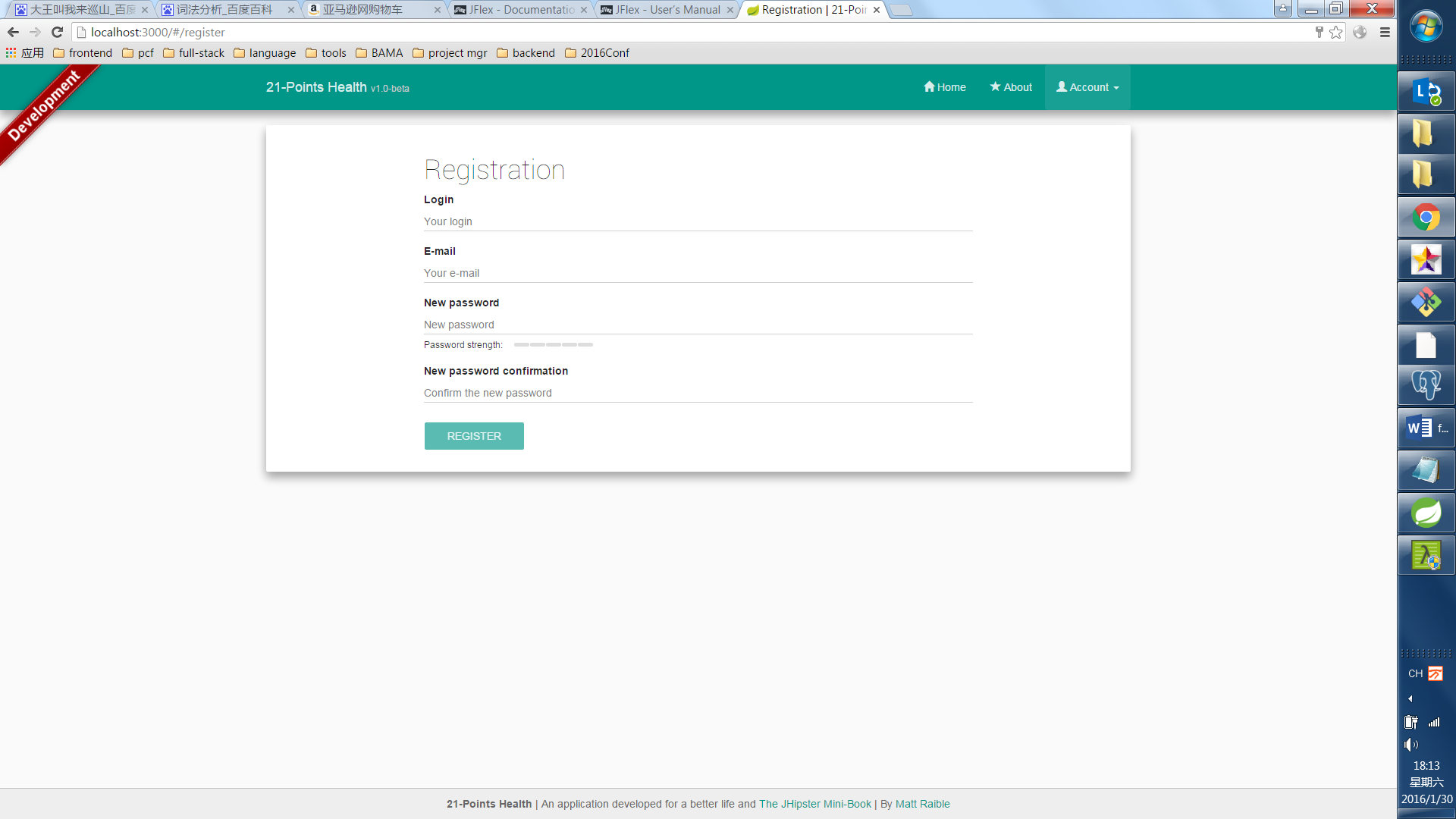
components/navbar/ navbar.directive.js

app/main/main.js

app/main/ main.html

app/main/ main.controller.js

### Register



用户点击链接<a href="#/register"> (in home页面主内容), 或者<a ui-sref="register">(在nav导航栏Account菜单的register), 或者<a href="#/register"> (in login页面)

应用程序状态site.account.register被激活，浏览器路由至/#/register,从而主页面中的content切换为register.html.

register.html界面除了录入用户信息，还得考虑点击register时的反馈和校验信息

**点击register后的反馈：(反馈信息来自后端服务器)**

注册成功：”Registration saved! Please check your email for confirmation.” ng-show=”success”

注册失败：”Registration failed! Please try again later.” ng-show=”error”

注册失败-已存在用户名: ” Login name already registered!” ng-show=”errorUserExists”

注册失败-用户邮箱已被使用: ” E-mail is already in use!” ng-show=”errorEmailExists”

注册失败-两次密码不一致: “The password do not match!” ng-show=”doNotMatch”

**用户输入信息的校验反馈：(反馈信息直接产生于前端)**

<input type="text" class="form-control" ng-model="registerAccount.login" ng-minlength=1 ng-maxlength=50 ng-pattern="/^[a-z0-9]\*$/" required>

对输入有几个要求，每个要求出现错误时均有校验信息

<div ng-show="form.login.$dirty && form.login.$invalid">

<p class="help-block" ng-show="form.login.$error.required">Your login is required. </p>

<p class="help-block" ng-show="form.login.$error.minlength">

Your login is required to be at least 1 character.

</p>

<p class="help-block" ng-show="form.login.$error.maxlength">

Your login cannot be longer than 50 characters.

</p>

<p class="help-block" ng-show="form.login.$error.pattern">

Your login can only contain lower-case letters and digits.

</p>

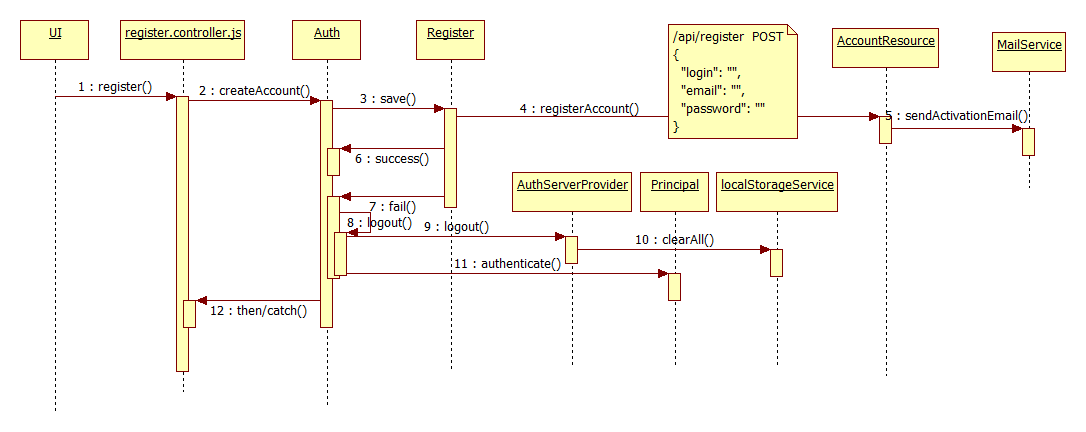
</div>

Files:

app/account/register/register.js

app/account/register/register.html

app/account/register/register.controller.js



用户注册完，会在后台数据库health数据表jhi\_user存入用户信息，同时给该用户生成activation\_key（同时置activated=false），后台服务器给用户邮箱发送邮件，通知用户激活帐户

### Activate

用户收到激活邮件，activationEmail.html内容（后台生成）含链接<a th:href="@{|${baseUrl}/#/activate?key=${user.activationKey}|}"</a>

用户点击链接相当于路由/#/activiate?key，应用程序状态切换到site.account.activate，从而主页面的content切换为activate.html.

注意点：

子状态activate的所有父状态site, site.account都会自动激活，即渲染index.html

activate.html大部分内容都是隐藏，等待activate.controller.js访问后台服务器的结果

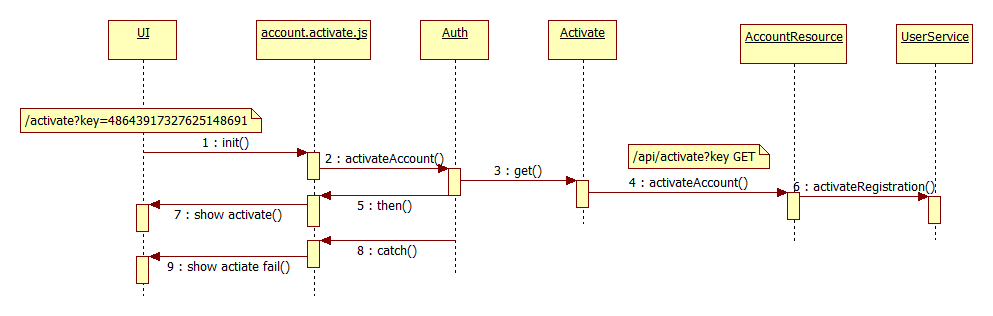
Files:

src/main/resources/mails/activationEmail.html (后台)

app/account/activate/activate.js

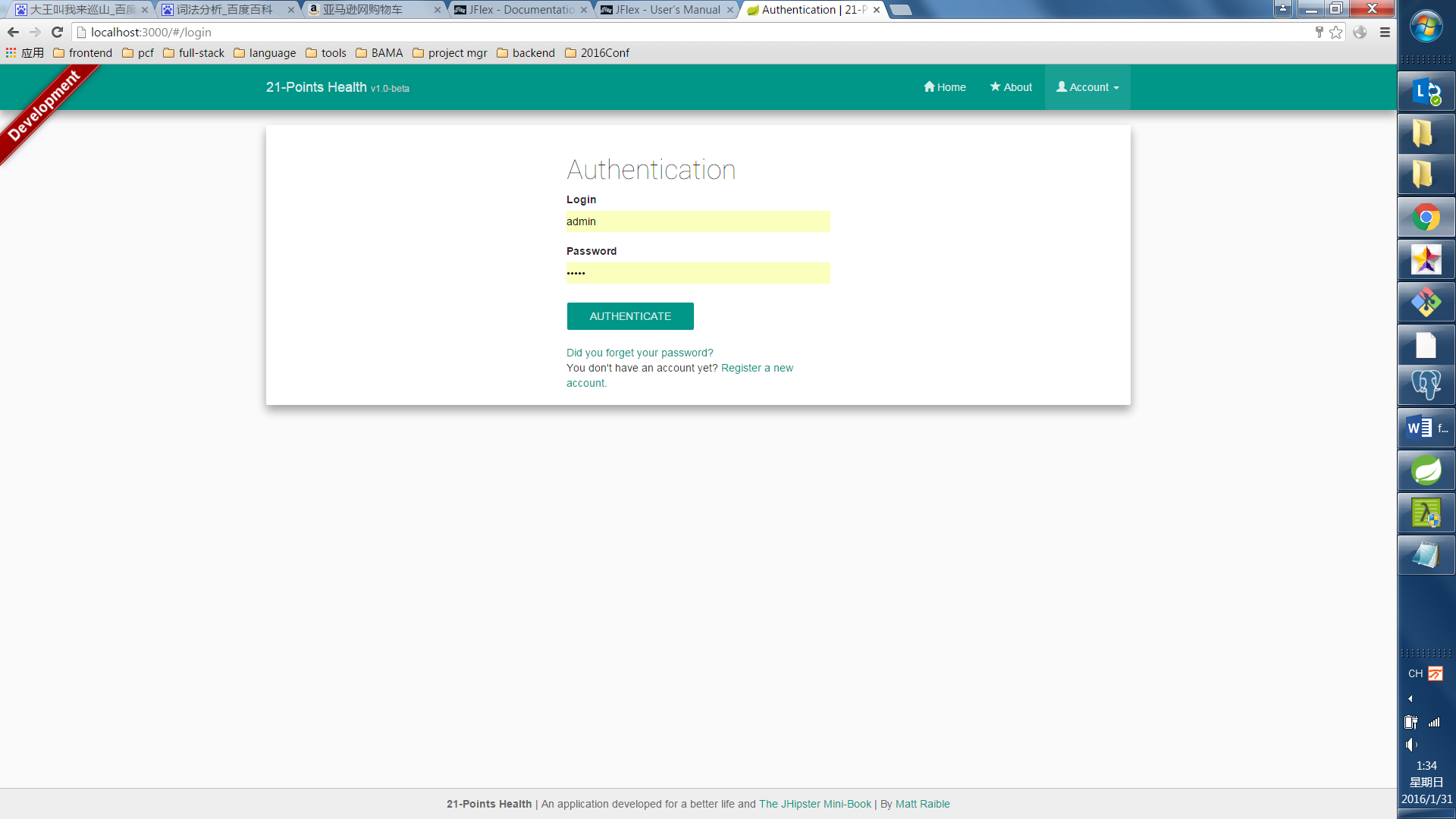
app/account/activate/activate.html

app/account/activate/activate.controller.js



用户激活后，后台通过key查找到用户，置activated=true,置activation\_key=null

### Login



用户点击链接<a ui-sref="login">(在nav导航栏Account菜单的Sign in)

应用程序状态site.account.login被激活，浏览器路由至/#/login,从而主页面中的content切换为login.html.

login.html界面除了录入用户信息，还得考虑点击authenticate后的反馈

点击authenticate后的反馈： (反馈信息来自后端服务器)

认证失败：“Authentication failed! …” ng-show=”authenticationError”

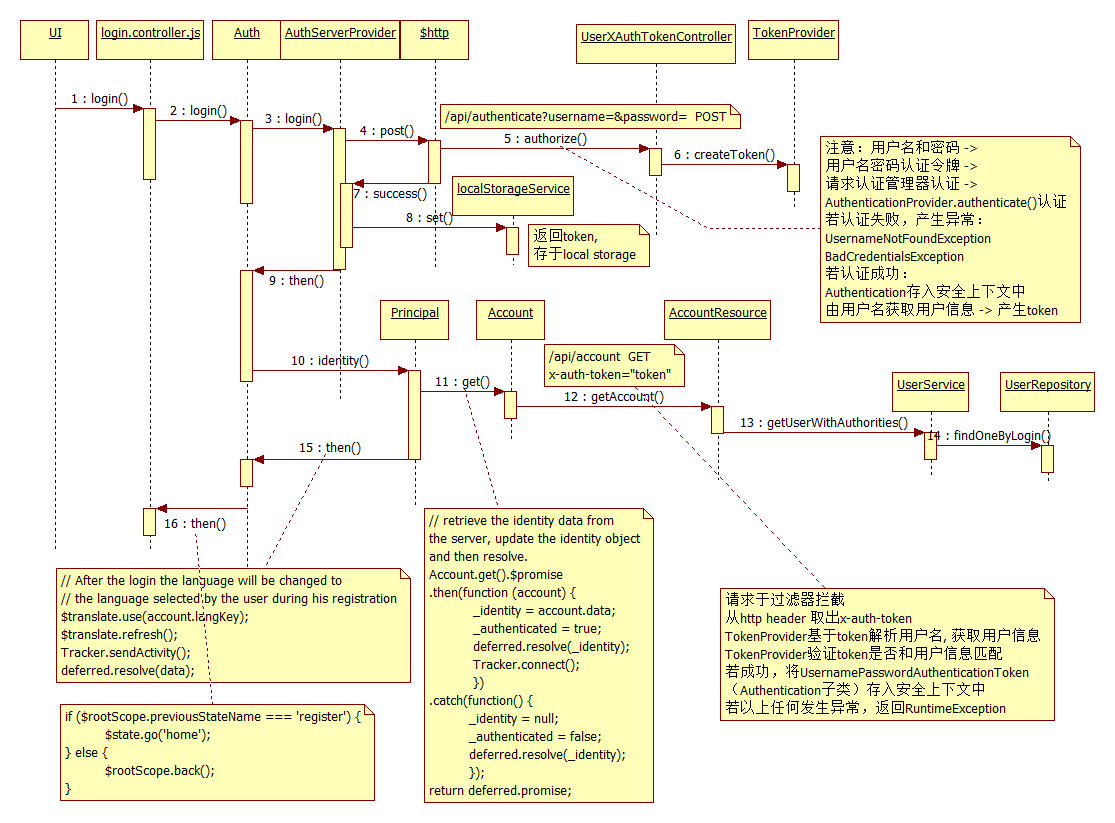
认证成功：**navbar根据isAuthenticated() and has-role="ROLE\_ADMIN"更新导航菜单，路由至home页面, 此时home页面里的内容也会更新为已登录用户的内容。**

Files:

app/account/login/login.js

app/account/login/login.html

app/account/login/login.controller.js



注意：认证成功后，后台会返回token，从而保存于local storage. 下次与后台交互不需要用户凭证，仅通过token即可

前台每次与后台交互，每次请求都会被认证拦截，见app.js

$httpProvider.interceptors.push('authInterceptor');

.factory('authInterceptor', function ($rootScope, $q, $location, localStorageService) {

return {

// Add authorization token to headers

**request**: function (**config**) {

config.headers = config.headers || {};

var token = localStorageService.get('token');

if (token && token.expires && token.expires > new Date().getTime()) {

config.headers['x-auth-token'] = token.token;

}

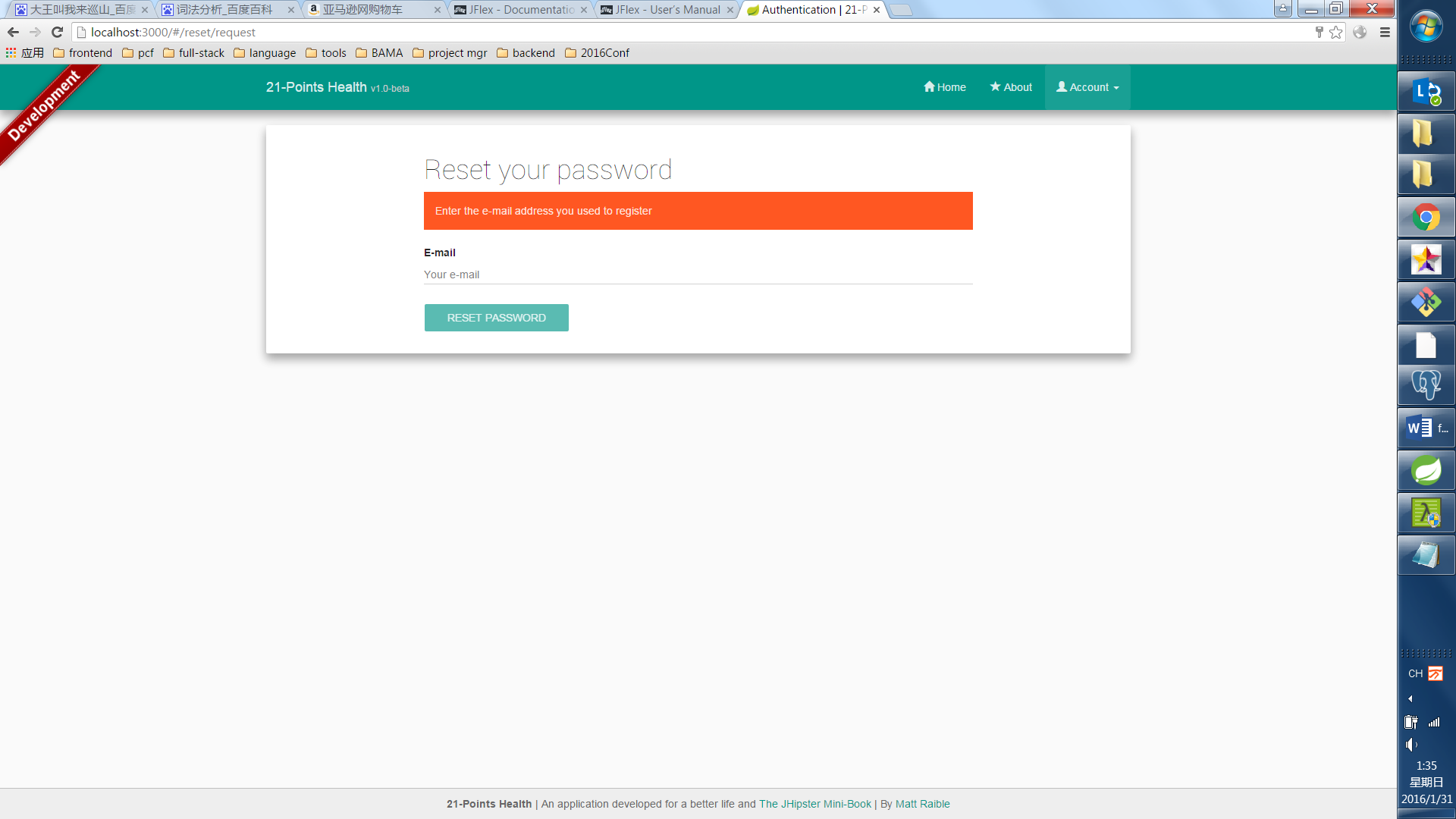
return **config**;

}};

})

**若有token,http请求头会带上token信息，从而在后台，可以基于安全上下文SecurityContext securityContext = SecurityContextHolder.getContext(); 获取http请求的用户具体信息，如用户名，邮箱，权限等**

### Reset password request



用户点击链接<a href="#/reset/request">Did you forget your password?</a> (在登入页面login.html里)

应用程序状态site.account. requestReset被激活，浏览器路由至/#//reset/request,从而主页面中的content切换为reset.request.html.

reset.request.html界面除了录入用户邮箱，还得考虑点击重置密码时的反馈和校验信息

点击reset password后的反馈：(反馈信息来自后端服务器)

成功：”Check your e-mails for details on how to reset your password.” ng-show="success == 'OK'"

失败：”E-Mail address isn't registered! Please check and try again.” ng-show="errorEmailNotExists"

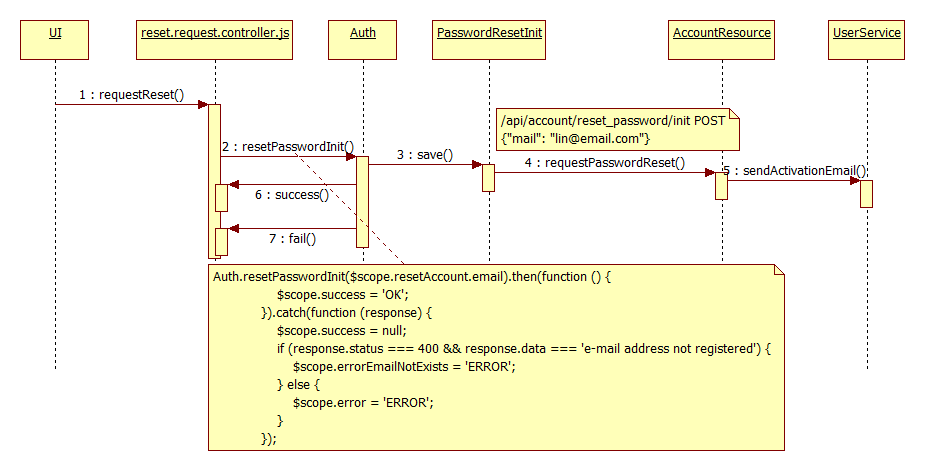
用户输入信息的校验反馈：(反馈信息直接产生于前端)，见register.html

Files:

app/account/reset/request/reset.request.js

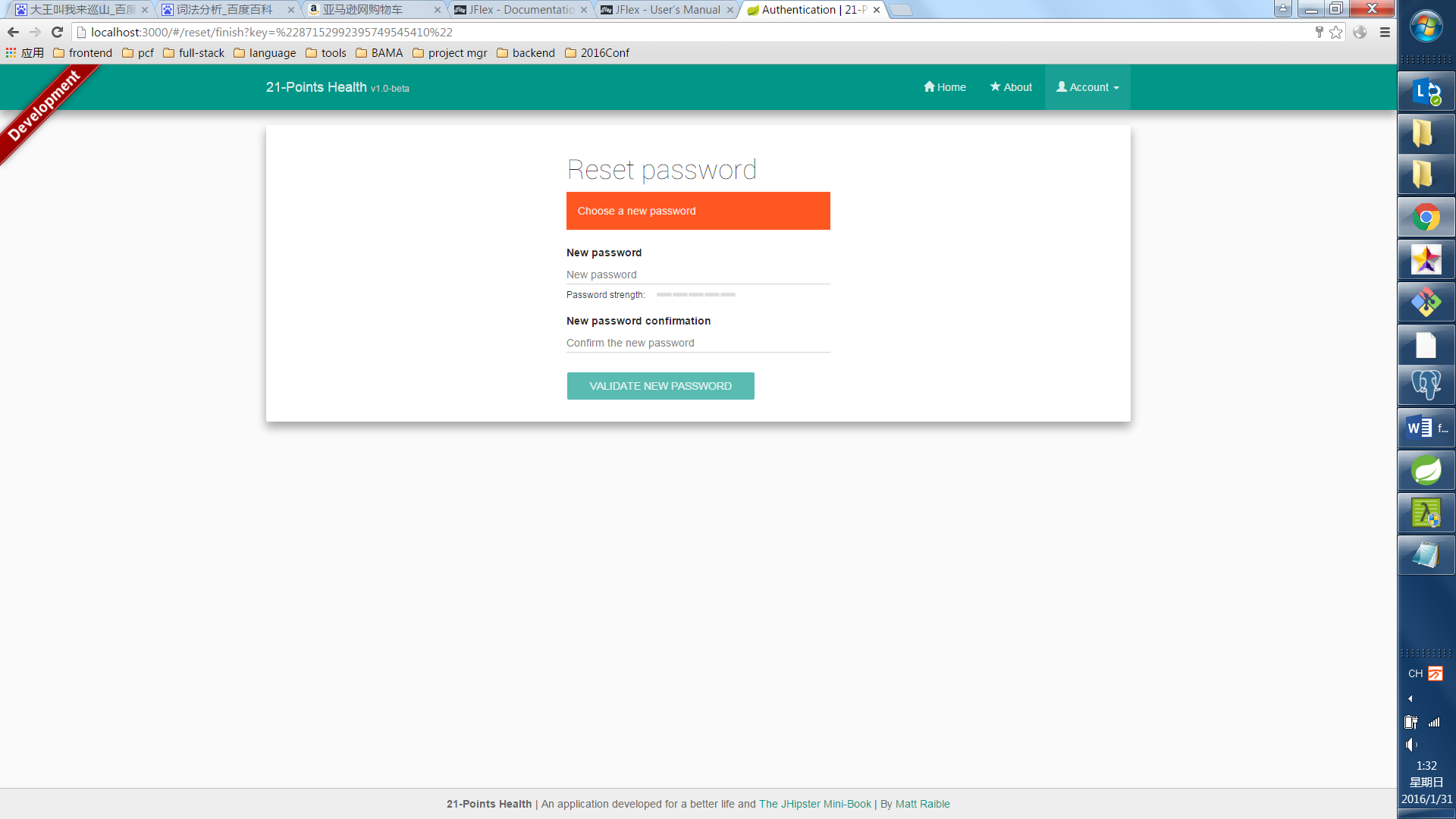
app/account/reset/request/reset.request.html

app/account/reset/request/reset.request.controller.js



后台数据库health数据表jhi\_user存入用户信息，同时给该用户生成reset\_key and reset\_date，后台服务器给用户邮箱发送邮件，通知用户重置密码

### Reset password finish



用户收到邮件，邮件内容passwordResetEmail.html(后台生成)含链接

<a th:href="@{|${baseUrl}/#/reset/finish?key=${user.resetKey}|}"</a>

点击后，路由于/#/reset/finish?key，相当于应用程序状态site.account.finishReset激活，从而主页面的content切换为reset.finish.html.

reset.finish.html界面除了录入用户密码，还得考虑点击验证新密码时的反馈和校验信息

点击validate new password后的反馈：(反馈信息来自后端服务器)

成功：”Your password has been reset.Please <a href="#/login">authenticate</a>” ng-show=”success”

失败：”Your password couldn't be reset...” ng-show=”error”

失败-两次密码不一致: “The password do not match!” ng-show=”doNotMatch”

用户输入信息的校验反馈：(反馈信息直接产生于前端), 见register.html

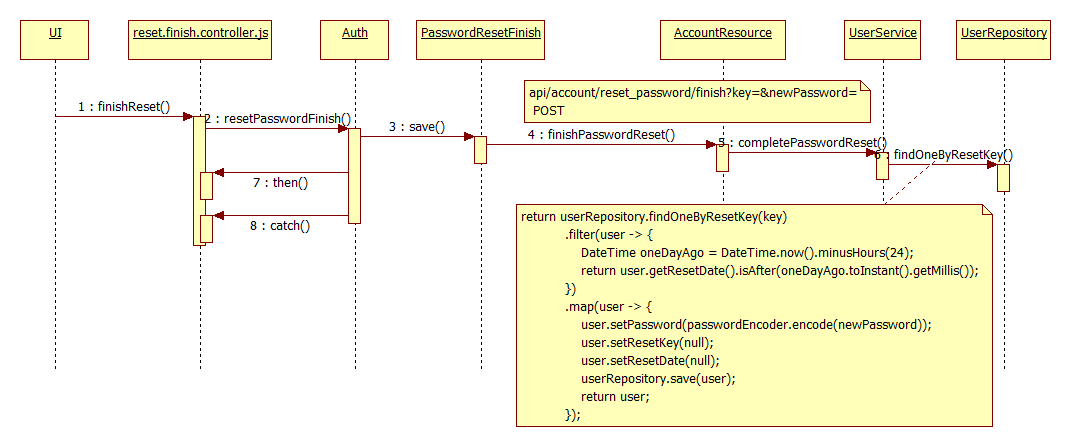
Files:

src/main/resources/mails/passwordResetEmail.html (后台)

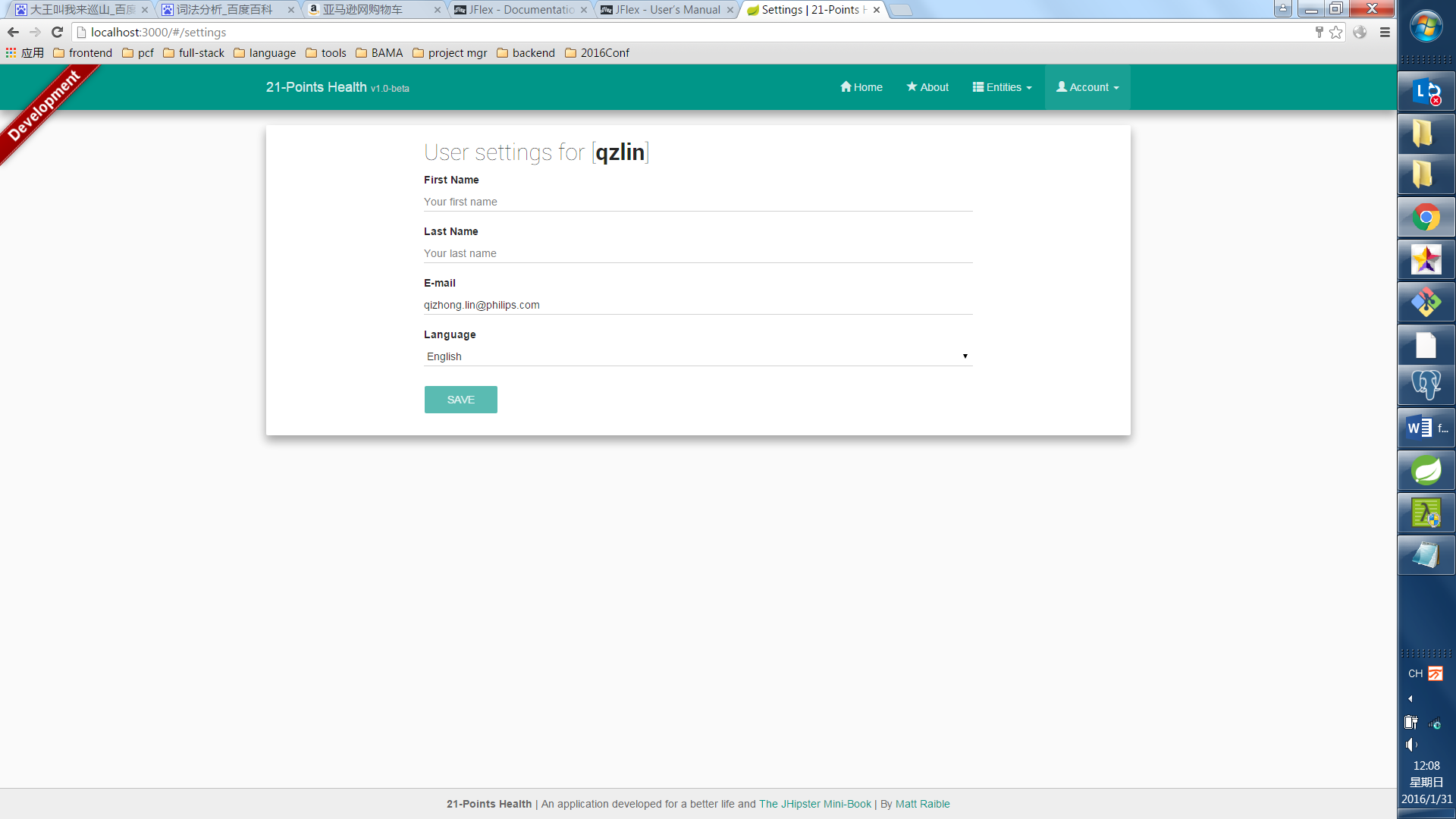
app/account/reset/request/reset.finish.js

app/account/reset/request/reset.finish.html

app/account/reset/request/reset.finish.controller.js



### Setting (设置用户帐户信息)



用户登录后，基于isAuthenticated() and has-role="ROLE\_ADMIN"会更新导航菜单和页面内容， 点击链接<a ui-sref="settings"> (在nav导航栏Account菜单的Settings)

应用程序状态site.account.settings被激活，浏览器路由至/#/settings,从而主页面中的content切换为settings.html.

settings.html界面除了修改用户信息，还得考虑点击save后的反馈和校验信息

成功： “Settings saved!” ng-show="success"

失败：”An error has occurred! Settings could not be saved.” ng-show="error"

失败-email: “E-mail is already in use!...” ng-show="errorEmailExists"

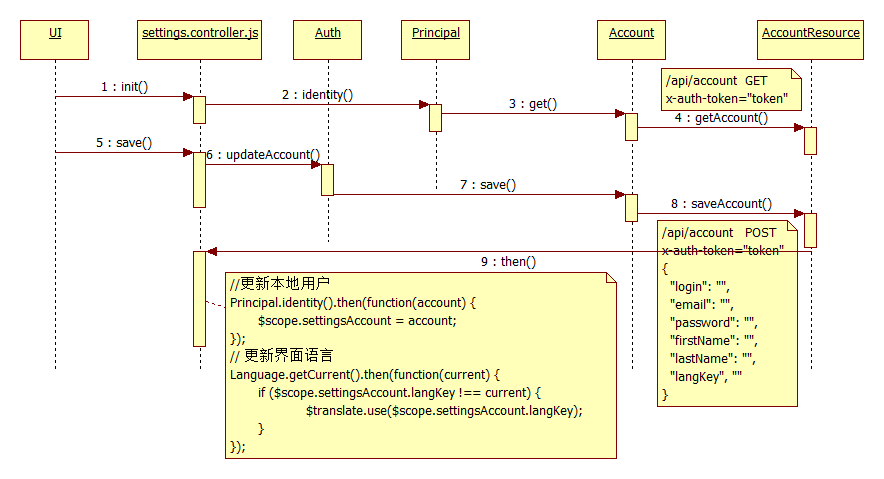
用户输入信息的校验反馈：(反馈信息直接产生于前端), 见register.html

Files:

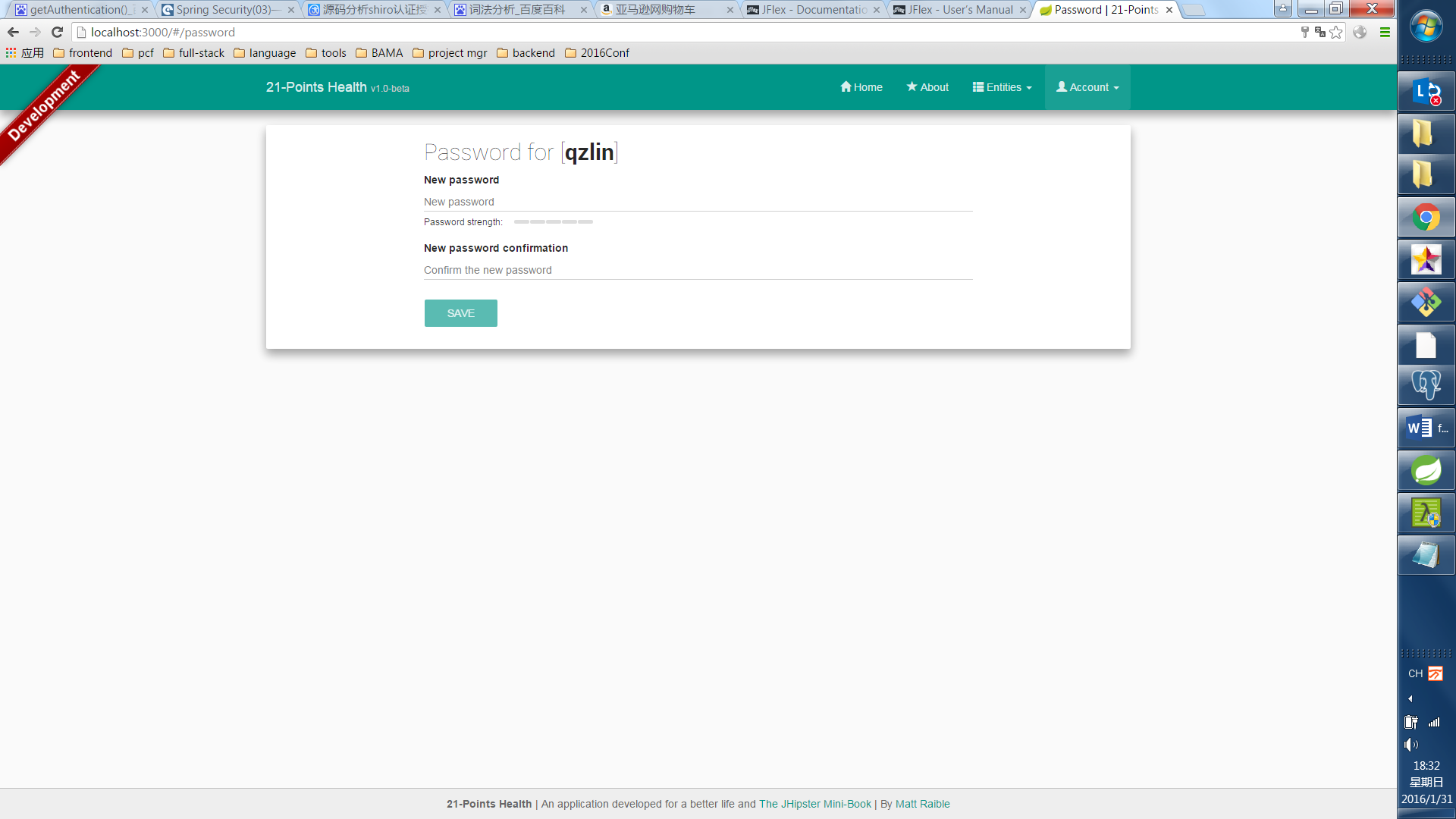
app/account/settings/settings.js

app/account/settings/settings.html

app/account/settings/settings.controller.js



### Change password



用户登录后，基于isAuthenticated() and has-role="ROLE\_ADMIN"会更新导航菜单和页面内容， 点击链接<a ui-sref="password"> (在nav导航栏Account菜单的Password)

应用程序状态site.account.password被激活，浏览器路由至/#/password,从而主页面中的content切换为password.html.

password.html界面除了修改用户信息，还得考虑点击save后的反馈和校验信息

成功： “Password changed!” ng-show="success"

失败：”An error has occurred!The password could not be changed.” ng-show="error"

失败-两次密码不一致: “The password …do not match!” ng-show=" doNotMatch"

用户输入信息的校验反馈：(反馈信息直接产生于前端), 见register.html

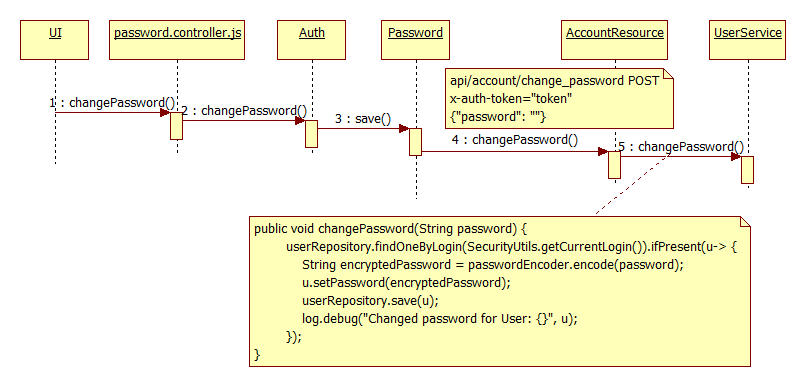
Files:

app/account/password/password.js

app/account/password/password.html

app/account/password/password.controller.js

app/account/password/password.directive.js



用户输入密码时，会被监视密码的强度，密码强度组件passwordStrengthBar实现在password.directive.js，视图模板见template, 链接函数里监视输入密码，基于算法计算密码强度(特征：长度，大写，小写，数字，特殊符号等)，然后映射为视图模板指示条的颜色

### Logout

用户登录后，基于isAuthenticated() and has-role="ROLE\_ADMIN"会更新导航菜单和页面内容， 点击链接<a href="" ng-click="logout()"> (在nav导航栏Account菜单的Log out)

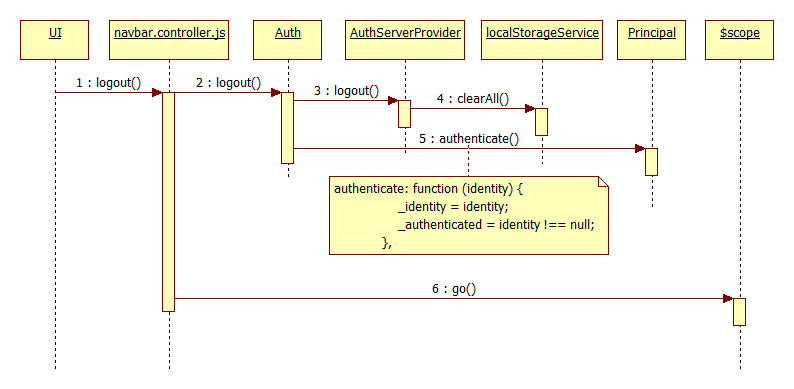
清除local storage，清空认证，浏览器路由至/#/home

Files:

app/account/logout/logout.js

app/account/logout/logout.controller.js

components/navbar/navbar.controller.js



### Authenticate

第一次，基于用户名和密码认证

/api/authenticate?username=&password= POST

web/rest/UserXAuthTokenController.java

public Token authorize(@RequestParam String username, @RequestParam String password) {

// UsernamePasswordAuthenticationToken是Authentication具体实现类，表示用户名和密码(框架自带)

UsernamePasswordAuthenticationToken token = new

UsernamePasswordAuthenticationToken(username, password);

//请求认证，若认证失败，产生异常：UsernameNotFoundException 和 //BadCredentialsException

Authentication authentication = this.authenticationManager.authenticate(token);

//若认证成功，则在安全上下文中保存Authentication

SecurityContextHolder.getContext().setAuthentication(authentication);

UserDetails details = this.userDetailsService.loadUserByUsername(username);

return tokenProvider.createToken(details);

}

security/AuthenticationProvider.java

security/UserDetailsService.java

public class AuthenticationProvider implements security.authentication.AuthenticationProvider {

private PasswordEncoder passwordEncoder;

private UserDetailsService userDetailsService;

public AuthenticationProvider(UserDetailsService userDetailsService, PasswordEncoder passwordEncoder) {

this.userDetailsService = userDetailsService;

this.passwordEncoder = passwordEncoder;

}

@Override

public Authentication authenticate(Authentication authentication) throws AuthenticationException {

UsernamePasswordAuthenticationToken token =

(UsernamePasswordAuthenticationToken) authentication;

String login = token.getName();

UserDetails user = userDetailsService.loadUserByUsername(login);

if (user == null) { throw new UsernameNotFoundException("User does not exists"); }

String password = user.getPassword();

String tokenPassword = (String) token.getCredentials();

if (!passwordEncoder.matches(tokenPassword, password)) {

throw new BadCredentialsException("Invalid username/password");

}

return new UsernamePasswordAuthenticationToken(user, password, user.getAuthorities());

}

}

@Component("userDetailsService")

public class UserDetailsService implements org.springframework.security.core.userdetails.UserDetailsService {

@Inject private UserRepository userRepository;

@Override @Transactional

public UserDetails loadUserByUsername(final String login) {

String lowercaseLogin = login.toLowerCase();

Optional<User> userFromDatabase = userRepository.findOneByLogin(lowercaseLogin);

return userFromDatabase.map(user -> {

if (!user.getActivated()) {

throw new UserNotActivatedException("User " + lowercaseLogin + " was not activated");

}

List<GrantedAuthority> grantedAuthorities = user.getAuthorities().stream()

.map(authority -> new SimpleGrantedAuthority(authority.getName()))

.collect(Collectors.toList());

return new org.springframework.security.core.userdetails.User(lowercaseLogin,

user.getPassword(),grantedAuthorities);

}).orElseThrow(() -> new UsernameNotFoundException("User " + lowercaseLogin + " was not found in the database"));

}

}

登录后，每次请求基于token验证

config/SecurityConfiguration.java

@Configuration @EnableWebSecurity

@EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true)

public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

@Inject private UserDetailsService userDetailsService;

@Inject private TokenProvider tokenProvider;

@Override protected void configure(HttpSecurity http) throws Exception {

http

.exceptionHandling()

.authenticationEntryPoint(authenticationEntryPoint)

.and()

.sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS)

.and()

.apply(securityConfigurerAdapter());

}

private XAuthTokenConfigurer securityConfigurerAdapter() {

return new XAuthTokenConfigurer(userDetailsService, tokenProvider);

}

}

config/XAuthConfiguration.java

application.yml

security/xauth/XAuthTokenConfigurer.java

security/xauth/XAuthTokenFilter.java

security/xauth/TokenProvider.java

application.yml

authentication:

xauth:

secret: myXAuthSecret

tokenValidityInSeconds: 1800 # Token is valid 30 minutes

//Configures x-auth-token security.

@Configuration

public class XAuthConfiguration implements EnvironmentAware {

private RelaxedPropertyResolver propertyResolver;

@Override public void setEnvironment(Environment environment) {

this.propertyResolver = new RelaxedPropertyResolver(environment, "authentication.xauth.");

}

@Bean public TokenProvider tokenProvider(){

String secret = propertyResolver.getProperty("secret", String.class, "mySecretXAuthSecret");

int validityInSeconds = propertyResolver.getProperty("tokenValidityInSeconds", Integer.class, 3600);

return new TokenProvider(secret, validityInSeconds);

}

}

public class XAuthTokenConfigurer extends SecurityConfigurerAdapter<DefaultSecurityFilterChain, HttpSecurity> {

private TokenProvider tokenProvider;

private UserDetailsService detailsService;

public XAuthTokenConfigurer(UserDetailsService detailsService, TokenProvider tokenProvider) {

this.detailsService = detailsService;

this.tokenProvider = tokenProvider;

}

@Override public void configure(HttpSecurity http) throws Exception {

XAuthTokenFilter customFilter = new XAuthTokenFilter(detailsService, tokenProvider);

http.addFilterBefore(customFilter, UsernamePasswordAuthenticationFilter.class);

}

}

public class XAuthTokenFilter extends GenericFilterBean {

private final static String XAUTH\_TOKEN\_HEADER\_NAME = "x-auth-token";

private UserDetailsService detailsService;

private TokenProvider tokenProvider;

public XAuthTokenFilter(UserDetailsService detailsService, TokenProvider tokenProvider) {

this.detailsService = detailsService;

this.tokenProvider = tokenProvider;

}

@Override public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {

try {

HttpServletRequest httpServletRequest = (HttpServletRequest) servletRequest;

String authToken = httpServletRequest.getHeader(XAUTH\_TOKEN\_HEADER\_NAME);

if (StringUtils.hasText(authToken)) {

String username = this.tokenProvider.getUserNameFromToken(authToken);

UserDetails details = this.detailsService.loadUserByUsername(username);

if (this.tokenProvider.validateToken(authToken, details)) {

UsernamePasswordAuthenticationToken token = new UsernamePasswordAuthenticationToken(details, details.getPassword(), details.getAuthorities());

SecurityContextHolder.getContext().setAuthentication(token);

}

}

filterChain.doFilter(servletRequest, servletResponse);

} catch (Exception ex) {

throw new RuntimeException(ex);

}

}

}

public class TokenProvider {

private final String secretKey;

private final int tokenValidity;

public TokenProvider(String secretKey, int tokenValidity) {

this.secretKey = secretKey;

this.tokenValidity = tokenValidity;

}

public Token createToken(UserDetails userDetails) {

long expires = System.currentTimeMillis() + 1000L \* tokenValidity;

String token = userDetails.getUsername() + ":" + expires + ":" + computeSignature(userDetails, expires);

return new Token(token, expires);

}

public String computeSignature(UserDetails userDetails, long expires) {

StringBuilder signatureBuilder = new StringBuilder();

signatureBuilder.append(userDetails.getUsername()).append(":");

signatureBuilder.append(expires).append(":");

signatureBuilder.append(userDetails.getPassword()).append(":");

signatureBuilder.append(secretKey);

MessageDigest digest;

try { digest = MessageDigest.getInstance("MD5");

} catch (NoSuchAlgorithmException e) {

throw new IllegalStateException("No MD5 algorithm available!");

}

return new String(Hex.encode(digest.digest(signatureBuilder.toString().getBytes())));

}

public String getUserNameFromToken(String authToken) {

if (null == authToken) { return null; }

String[] parts = authToken.split(":");

return parts[0];

}

public boolean validateToken(String authToken, UserDetails userDetails) {

String[] parts = authToken.split(":");

long expires = Long.parseLong(parts[1]);

String signature = parts[2];

String signatureToMatch = computeSignature(userDetails, expires);

return expires >= System.currentTimeMillis() && signature.equals(signatureToMatch);

}

}

从而后台可以基于authentication(含用户名密码的认证或token的认证)从安全上下文获取用户信息

security/SecurityUtils.java

public final class SecurityUtils {

private SecurityUtils() {}

public static String getCurrentLogin() {

SecurityContext securityContext = SecurityContextHolder.getContext();

Authentication authentication = securityContext.getAuthentication();

UserDetails springSecurityUser = null;

String userName = null;

if(authentication != null) {

if (authentication.getPrincipal() instanceof UserDetails) {

springSecurityUser = (UserDetails) authentication.getPrincipal();

userName = springSecurityUser.getUsername();

} else if (authentication.getPrincipal() instanceof String) {

userName = (String) authentication.getPrincipal();

}

}

return userName;

}

//Check if a user is authenticated.

public static boolean isAuthenticated() {

SecurityContext securityContext = SecurityContextHolder.getContext();

Collection<? extends; GrantedAuthority> authorities = securityContext.getAuthentication().getAuthorities();

if (authorities != null) {

for (GrantedAuthority authority : authorities) {

if (authority.getAuthority().equals(AuthoritiesConstants.ANONYMOUS)) {

return false;

}

}

}

return true;

}

//If the current user has a specific security role.

public static boolean isUserInRole(String role) {

SecurityContext securityContext = SecurityContextHolder.getContext();

Authentication authentication = securityContext.getAuthentication();

if(authentication != null) {

if (authentication.getPrincipal() instanceof UserDetails) {

UserDetails springSecurityUser = (UserDetails) authentication.getPrincipal();

return springSecurityUser.getAuthorities().contains(new SimpleGrantedAuthority(role));

} }

return false;

}

}

21-point system: to track healthy.

Rules: 3 points per day

If you eat healthy, you get a point

If you exercise, you get a point

If you don’t drink alcohol, you get a point

Ionic = HTML5 UI for native application, offers a library of mobile-optimized HTML, CSS and Javascript components, gestures and tools for building highly interactive apps, build with Sass and optimized for AngularJS. Provides a native look and feel.

JHipster: a goldmine of information and lessons from several years of developer experience, Yeoman generator, Yeorman expects you to be in the directory you want to create your project in, rather than creating the directory for you

Liquibase: as source control for your database, it will help create new fields as you add them to your entities. It will also refactor your database, for example creating tables and dropping columns. It also has the ability to undo changes to your database, either automatically or with custom SQL

Liquibase will create your database schema for you and help you update your database when the need arises. It provides an easy-to-use workflow to adding new properties to your JHipster-generated entities using its diff feature.

PostgreSQL

#create user health with password ‘health’ ;

#drop database health;

#create database health ;

#grant all privileges on database health to health ;

src/main/resources/config/application-dev.yml

database:

dataSourceClassName: org.postgresql.ds.PGSimpleDataSource

url: jdbc:postgresql://localhost/health

username: health

password: health

GitHub:

$git init

$git add –A

$git commit –m “Initial check in of …”

#git push origin master

Continuous integration with Jenkins, deploy.

Generating entities

for each entity:

database table

liquibase change set

JPA entity class

spring data JpaRepository interface

spring MVC RestController class

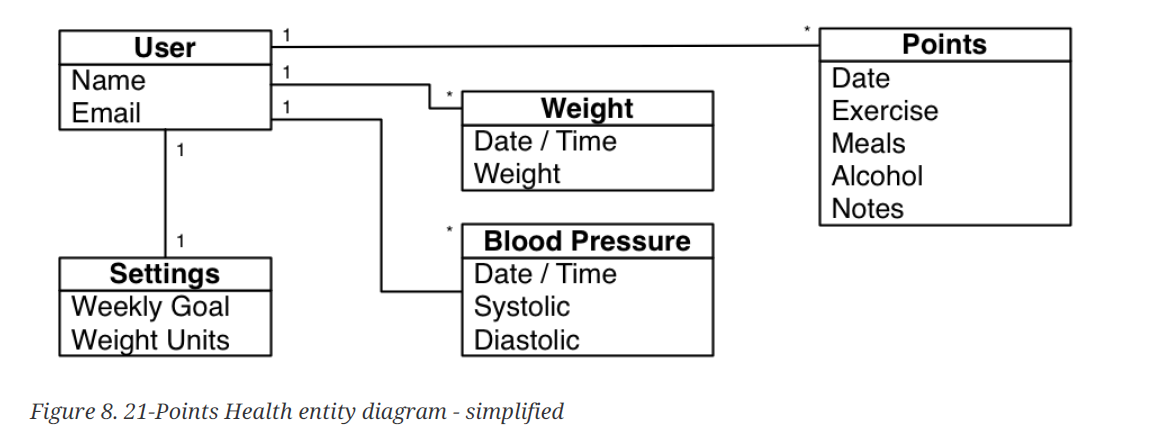
AngularJS router, controller and service and

HTML page

If you have entities with relationships, it will generate the necessary schema to support them (with foreign keys), and the Javascript and HTML code to manage them

Two methods of code generation for entity

* entity sub-generator
* UML editors (Modelio, UML Designer, GenMyModel and Visual Paradiam)



The most important thing to remember when generating entities with JHipster is that you must generate the entity that owns the relationship first

>yo jhipster:entity points (many-to-one relationship with User)

>yo jhipster:entity weight (many-to-one …)

>yo jhipster:entity bloodpressure (many-to-one …)

>yo jhipster:entity preferences (one-to-one relationship with User)

(please refer to page 27)

Bootstrap: web component frameworks

Building the UI and business logic

**Material design theme for Bootstrap**

>bower install bootstrap-material-design –save

<script>

$.material.init() ;

</script>

Display the user’s username instead of user’s ID

src/main/webapp/scripts/app/entities/goal/goal-dialog.html

<select class=”form-control” id=”field\_user” name=”user” ng-model=”goal.user.id” ng-option=”user.id as user.login for user in users>

UI mockup: OmniGraffle + Bootstrap stencil

UI code (HTML, Javascript, and CSS) is my favorite. I like that you can see changes immediately and make progress quickly – especially when you’re using dual monitors with Browsersync.

When developing UIs, I tend to make them work before writing tests. It’s usually a very visual activity and, with the aid of Browsersync, there’s rarely a delay before you see your changes. I like to write units tests for my Angular controllers and directives using Jasmine and I like to write integration tests with Protrator.

AngularUI Router

AngularUI Router is a routing component for AngularJS, It’s organized around states, to which you can attach routes and other behaviors. Some of its best features include support for nested/named views, tab history, and loading data before rendering a view.

AnguarUI Router allows you to look up data before displaying a page. It does this with its resolve property.

Angular’s $resource

Angular ships with a $resource factory that allows you to interact with a RESTful API with only a few lines of code. If your API is implemented so that you can perform CRUD (create, read, update, delete) at a specific endpoint, you can use $resource to wrap that endpoint, and voila, you have a service!

.factory(‘Product’, function($resource){

return $resource(API\_HOSTNAME + ‘/api/products/:id’) ;

}) ;

This gives you a number of actions(or methods) by default, you can easily override these methods or add new ones.

{

‘query’: {method: ‘GET’, isArray: true},

‘get’ : {method: ‘GET’},

‘save’: {method: ‘POST’},

‘remove’: {method: ‘DELETE’},

‘delete’: {method:’DELETE’}

}

Product.get{id: $stateParams.id}

Internationalization (i18n): because the word has 18 letters between “I” and “n”

Bootstrap 3

Grid system has four tiers of classes: xs (phone), sm (tablet), md (desktop), and lg (larger desktop)

Show and hide elements based on browser size:

.visible - [xs | sm | md | lg]

.hidden – [xs | sm | md | lg]

Add Bootstrap to your web application, changes are it’ll quickly start to look better. Typography, margins and padding will look better by default. However your forms might look funny, because Bootstrap requires a few classes on your form elements to make them look good.

Components: Bootstrap ships with a number of components included. Some require Javascript, some only require HTML5 markup and CSS classes to work. Web developers have always liked components in their frameworks. A framework that offers easy-to-use components often allows developers to write less code. Some popular Bootstrap components include: dropdowns, button groups, button dropdowns, navbar, breadcrumbs, pagination, alerts, progress bars and panels.

**Icons: Font icons are just fonts, but they contain symbols and glyphs instead of text. You can style, scale and load them quickly because of their small size.**

Sass: stands for “Syntactically Awesome StyleSheets” – variables, nesting, mixins, inheritance

$primary-color: #333

nav {

ul { list-style: none; }

li { display: inline-block ;}

}

@mixin border-radius($radius) {

-webkit-border-radius: $radius ;

-moz-border-radius: $radius ;

-ms-border-radius: $radius ;

border-radius: $radius ;

}

.box { @include border-radius(10px) ; }

Elasticsearch: add searchability to your entities. JHipster’s Elasticsearch support requires choosing Java 8+ and a SQL database. Spring Boot uses and configures Spring Data Elasticsearch. When using JHipster’s entity sub-generator. It automatically indexes the entity and creates and endpoint to support searching its properties. Search superpowers are also added to the AngularJS UI, so you can search in your entity’s list screen.

Continuous integration and deployment: Jenkins

Configure a continuous-integration(CI) server to build/test/deploy whenever I checked in changes to Git. I chose Jenkins for my CI server and used the following:

download Jenkins.war

java –jar Jenkins.war –httpPort=9000

JHipster leverages Spring MVC and its @RestController annotation to create a REST API. Its endpoints publish JSON to and consume JSON from clients. By separating the business logic and data persistence from the client, you can provide data to many different clients (HTML5, iOS, Android, TVs, watches, IoT devices, etc.)

**Spring Initializr: a configurable service for generating Spring projects.**

Security

With dependency spring-boot-starter-security, you get HTTP Basic authentication out of the box. By default, a user is created with username user and the password is printed in the logs when the application starts. T**o override the generated password, you can define a security.user.password.**

The most basic Spring Security Java configuration creates a servlet Filter, which is responsible for all the security (protecting URLs, validating credentials, redirecting to login, etc.).

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

auth.inMemoryAuthentication()

.withUser("user").password("password").roles("USER");

}

}

There’s not much code, but it provides many features:

• Requires authentication to every URL in your application.

• Generates a login form for you.

• Allows user:password to authenticate with form-based authentication.

• Allows the user to logout.

• Prevents CSRF attacks.

• Protects against session fixation.

• Security-header integration.

◦ HTTP Strict Transport Security for secure requests.

◦ X-Content-Type-Options integration.

◦ Cache control.

◦ X-XSS-Protection integration.

◦ X-Frame-Options integration to help prevent clickjacking.

• Integrates with HttpServletRequest API methods: getRemoteUser(), getUserPrinciple(),

isUserInRole(role), login(username, password), and logout()

three authentication options:

• HTTP Session Authentication — Uses the HTTP session, so it is a stateful mechanism.

Recommended for small applications.

• OAuth2 Authentication — A stateless security mechanism. You might prefer it if you want to scale

your application across several machines.

• Token-based authentication — Like OAuth2, a stateless security mechanism. This is specific to

JHipster, not provided by Spring Security.

用户名，密码，角色等见于src/main/resources/config/liquibase/…

RDBMS: transactions - ACID (atomicity, consistency, isolation, durability), were not designed to cope with the scale and agility challenges that face modern applications, nor were they built to take advantage of the cheap storage and processing power available today

NoSQL: distributed across several machines, with some latency, it guarantees only that all instances will eventually be consistent.

Cassandra: is a distributed storae system for managing structured data that is designed to scale to a very large size across many commodity servers.

日期操作函数

LocalDate today = new LocalDate() ;

LocalDate thisMonday = today.withDayOfWeek(DateTimeConstants.MONDAY) ;

LocalDate lastMonday = thisMonday.minusWeeks(1) ;

LocalDate previousDate = today.minusDay(days) ;

DateTime daysAgo = previousDate.toDateTimeAtCurrentTime() ;

DateTime rightNow = today.toDateTimeAtCurrentTime() ;

自定义资源

.factory(‘Preferences’, function($resource) {

return $resource(‘api/preferences/:id’, {}, {

‘user’: {method: ‘GET’, isArray: false, url: ‘/api/my-preferences’},

…

}) ;

}) ;

.controller(‘MainController’, function($scope, Preferences) {

Preferences.user(function(data) {

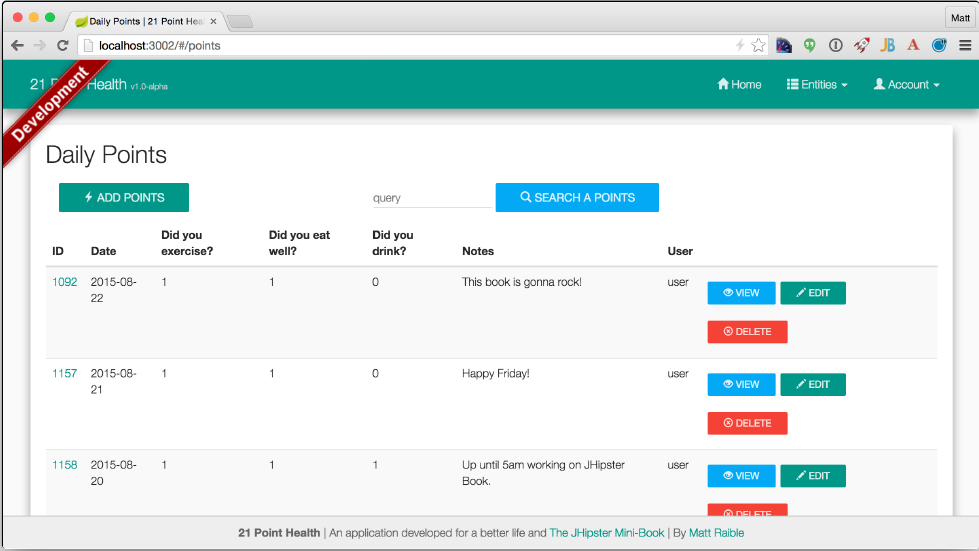
$scope.preferences = data ;

}) ;

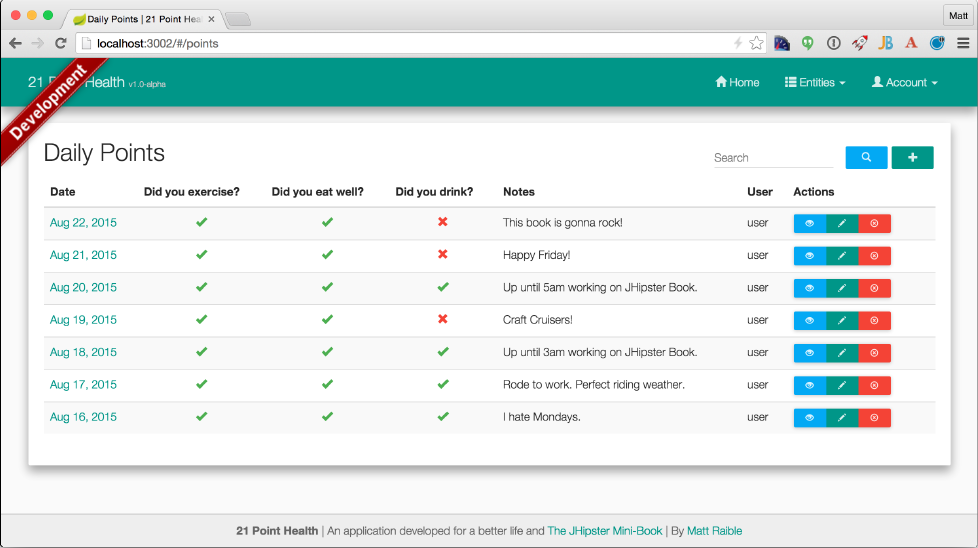
}) ;

实体集合

表格显示， 无限滚动，搜索，CRUD操作



is changed to



script/app/entities/points/points.js

script/app/entities/points/points.html

script/app/entities/points/points.controller.js

$stateProvider.state('points', {

parent: 'entity', url: '/points',

data: {roles: ['ROLE\_USER'], pageTitle: '…'},

views: {

'content@': {

templateUrl: 'scripts/app/entities/points/points.html',

controller: 'PointsController'

}},

resolve: {

translatePartialLoader: **function** ($translate, $translatePartialLoader) {

$translatePartialLoader.addPart('points');

$translatePartialLoader.addPart('global');

**return** $translate.refresh();

}]

}

})

模型数据 （来自控制器）

.controller(‘PointsController’, function($scope, Points, PointsSearch, ParseLinks) {

$scope.pointsList = [] ;

$scope.page = 1 ;

$scope.loadAll() ;

}



<div class=*"row"*>

<div class=*"col-sm-7"*><h2>Points</h2></div>

<div class=*"col-sm-5 text-right"*>

<form name=*"searchForm"* class=*"form-inline"*>

<div class=*"form-group p-r"*>

<input type=*"text"* class=*"form-control"* **ng-model=*"searchQuery*"**>

</div>

<button class=*"btn btn-info btn-sm"* **ng-click=*"search()"***tooltip=*"…"*><i class=*"glyphicon glyphicon-search"*></i>

</button>

<button class=*"btn btn-primary btn-sm"* ui-sref=*"points.new"* tooltip=*"…"*><span class=*"glyphicon glyphicon-plus"*></span>

</button>

</form>

</div>

</div>

解析：

class=”text-right” 表示该div右对齐

<form class=”form-inline”> 表示内联表单

class=”p-r” 表示padding-right: 10px （见main.css）

控制器

$scope.search = **function**(){

PointsSearch.query({query: $scope.searchQuery}, **function**(result) {

$scope.pointsList = result;

}, **function**(response) {

**if**(response.status === 404) {$scope.loadAll();}

});

};

components/entities/points/points.search.service.js

.factory('PointsSearch', function ($resource) {

return $resource('api/\_search/points/:query', {}, {

'query': { method: 'GET', isArray: true}

});

});

src/main/java/org/jhipster/health/web/rest/PointsResource.java

@RequestMapping(value = "/\_search/points/{query}",

method = RequestMethod.GET, produces = MediaType.APPLICATION\_JSON\_VALUE)

@Timed

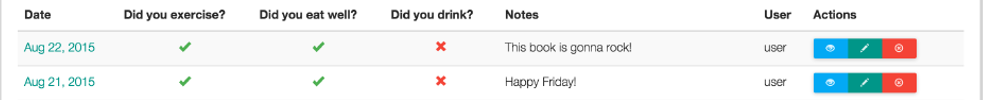
// 默认按String字段搜索，搜索内容可以只是String字段内容一部分

public List<Points> search(@PathVariable String query) {

return StreamSupport.stream(pointsSearchRepository.search(

queryString(query)).spliterator(), false).collect(Collectors.toList());

}



<div class=*"table-responsive"*>

<table class=*"table table-striped"*>

<thead><tr><th>Date</th><th>Exercise</th><th>Meals</th><th>Alcohol</th>

<th>Notes</th><th>user</th><th class=*"col-md-2"*>Actions</th></tr>

</thead>

<tbody infinite-scroll=*"loadPage(page + 1)"* infinite-scroll-disabled=*"links['last'] == page"*>

<tr ng-repeat=*"points in pointsList"*>

<td class=*"text-nowrap"*>

<a ui-sref=*"points.detail({id:points.id})"*>…</a></td>

<td class=*"text-center"*>

<i class=*"glyphicon"* ng-class=*"{'glyphicon-ok text-success': points.exercise,'glyphicon-remove text-danger':!points.exercise}"*></i></td>

…

<td><div class=*"truncate"* popover=*" {{points.notes}}"*> …</div></td>

<td>{{points.user.login}}</td>

<td><div class=*"btn-group m-n"*>

<button type=*"submit"* ui-sref=*"points.detail({id:points.id})"*><span class=*"glyphicon glyphicon-eye-open"*></span></button>

<button type=*"submit"* ui-sref=*"points.edit({id:points.id})"*<span class=*"glyphicon glyphicon-pencil"*></span></button>

<button type=*"submit"* ng-click=*"delete(points.id)"*><span class=*"glyphicon glyphicon-remove-circle"*></span></button></div></td>

…

解析：

<div class=*"table-responsive"*>

<table class=*"table table-striped"*>

将任何.table元素包裹在.table-responsive元素内，即可创建响应表格，其会在小屏幕设备上(小于768px)水平滚动，当屏幕大于768px宽度时，水平滚动条消失

ng-class：可以基于逻辑选择性添加class, 若存在points.meals，则添加类*glyphicon-ok text-success,* 若不存在，则添加类*glyphicon-remove text-danger*

截断字符串显示

*.truncate* {

width: *180px*;

white-space: *nowrap*;

overflow: *hidden*;

text-overflow: *ellipsis*;

cursor: *pointer*;

}

搜索当前用户的Points

public interface PointsRepository extends JpaRepository<Points,Long> {

@Query("select points from Points points where points.user.login = ?#{principal.username} order by points.date desc")

Page<Points> findAllForCurrentUser(Pageable pageable);

}

List<Points> points = pointsRepository.findAllByDateBetween(startOfWeek, endOfWeek) ;

List<Points> points\_user = points.stream().filter(p -> p.getUser().getLogin().equals(SecurityUtils.getCurrentLogin())

.collect(Collectors.toList()) ;

分页加载（无限滚动）

<tbody infinite-scroll="loadPage(page + 1)" infinite-scroll-disabled="links['last'] == page">

$scope.loadPage = function(page) { $scope.page = page ; $scope.loadAll() ; }

$scope.loadAll = function() {

Points.query({page: $scoe.page, per\_page: 20}, function(result, headers) {

$scope.links = ParseLinks.parse(headers(‘link’)) ;

for (var i = 0 ; i < result.length ; ++ i) { $scope.pointsList.push(result[i]) ; }} ; } ;

ngInfiniteScroll

<http://sroze.github.io/ngInfiniteScroll/index.html>

angular.module('myApplication', ['infinite-scroll']);

<ANY infinite-scroll='{expression}'

[infinite-scroll-distance='{number}']

[infinite-scroll-disabled='{boolean}']

[infinite-scroll-immediate-check='{boolean}']

[infinite-scroll-listen-for-event='{string}']>

</ANY>

points.html

<tbody infinite-scroll="points.loadPage(points.page + 1)" infinite-scroll-disabled="points.links['last'] == points.page">

<tr ng-repeat="item in points.pointsList">

ngInfiniteScroll will call points.loadPage(points.page + 1) any time the bottom of the element approaches the bottom of the browser window. when infinite-scroll-disabled=“true”, indicates that the infininite scroll expression should not be evaluated even if all other conditions are met. This is usually used to throttle or pause the infinite scroll,

**points.controller.js**

vm.pointsList = [];

vm.page = 1;

vm.links = {};

function loadPage(page) {

vm.page = page;

vm.loadAll();

};

function loadAll() {

Points.query({page: vm.page, per\_page: 20}, function (result, headers) {

vm.links = ParseLinks.parse(headers('link'));

for (var i = 0; i < result.length; i++) {

vm.pointsList.push(result[i]);

}

});

};

//line:9306 headersGetter of angular.js

**headers** = function(name) {

if (!headersObj) headersObj = parseHeaders(headers);

if (name) {

var value = headersObj[lowercase(name)];

if (value === void 0) { value = null; }

return value;

}

return headersObj;

};

hearder = **headers(‘link’)** ;

则header = "</api/points?page=1&per\_page=20>; rel="last",</api/points?page=1&per\_page=20>; rel="first""

vm.links = **ParseLinks.parse(headers('link'))**; //解析header，last = page(第一个api), first=page

则vm.links= { last:1, first: 1}

//Server: /api/points/ GET

返回ResponseEntity(T body, MultiValueMap<String, String> headers, HttpStatus statusCode)

其中HttpHeaders headers = {

“X-Total-Count”: 3,

“Link”: ’</api/points?page=1&per\_page=20>; rel="prev",</api/points?page=1&per\_page=20>; rel="last",</api/points?page=1&per\_page=20>; rel="first"’

}

实体删除确认框 （视图

<div **class="modal fade"** id="deletePointsConfirmation">

<div **class="modal-dialog"**>

<div **class="modal-content"**>

<form name="deleteForm" ng-submit="confirmDelete(points.id)">

<div **class="modal-header"**>

<button type="button" **class="close" data-dismiss="modal" ng-click="clear()"**>&times;</button>

<h4 **class="modal-title"** translate="entity.delete.title">Confirm delete operation</h4>

</div>

<div **class="modal-body"**>

<p>Areyou sure you want to delete this Points?</p>

</div>

<div **class="modal-footer"**>

<button type="button" class="btn btn-default" **data-dismiss="modal" ng-click="clear()"**>>Cancel</button>

<**button type="submit"** ng-disabled="deleteForm.$invalid" class="btn btn-danger">Delete</button>

</div>

</form>

点击实体集合页面的删除操作，弹出删除确认模态对话框

$scope.delete = function (id) {

Points.get({id: id}, function(result) {

$scope.points = result;

$('#deletePointsConfirmation').modal('show');

});

};

模态对话框确认提交后，删除数据（前后台），关闭对话框

$scope.confirmDelete = function (id) {

Points.delete({id: id}, function () {

$scope.reset();

$('#deletePointsConfirmation').modal('hide');

$scope.clear();

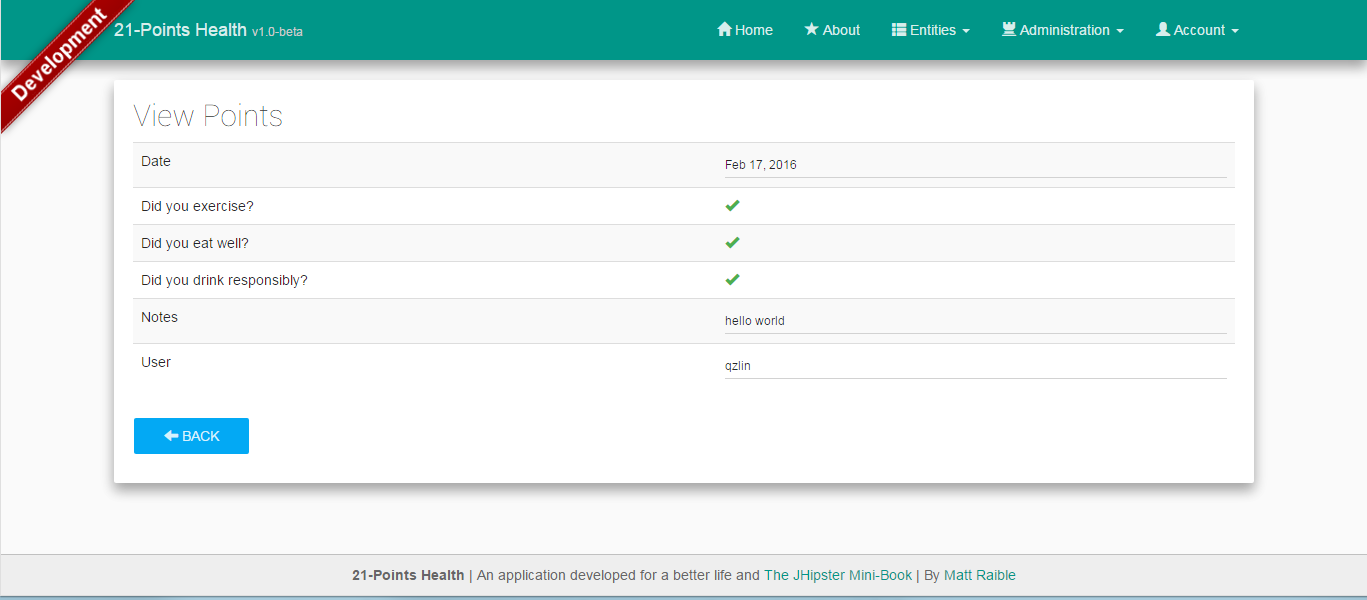
});

};

实体详细视图

scripts/app/entities/points/points-detail.html

scripts/app/entities/points/points-detail.controller.js



<button ui-sref=*"points.detail({id:points.id})"*>…</button>： 点击链接会进入应用程序状态entity.points.detail = url: '/points/{id:int}', 从而Points.get({id : $stateParams.id})能接受到，在渲染视图时，entity已经获得数据，传给控制器，最终填充视图

.state('points.detail', {

parent: 'entity',

url: '/points/{id:int}',

data: {roles: ['ROLE\_USER'],

pageTitle: '21pointsApp.points.detail.title'},

views: {'content@': {templateUrl: 'scripts/app/entities/points/points-detail.html', controller: 'PointsDetailController'}},

resolve: {translatePartialLoader: function(..){…},

entity: **function**($stateParams, Points) {

**return** Points.get({id : $stateParams.id});

}}

})

.controller('PointsDetailController', function ($scope, $rootScope, $stateParams, entity, Points, User) {

$scope.points = entity;

$scope.load = function (id) {

Points.get({id: id}, function(result) { $scope.points = result; });

};

//接收新增或修改视图的数据，从而更新该详细视图的数据

$rootScope.$on('21pointsApp:pointsUpdate', function(event, result) {

$scope.points = result;

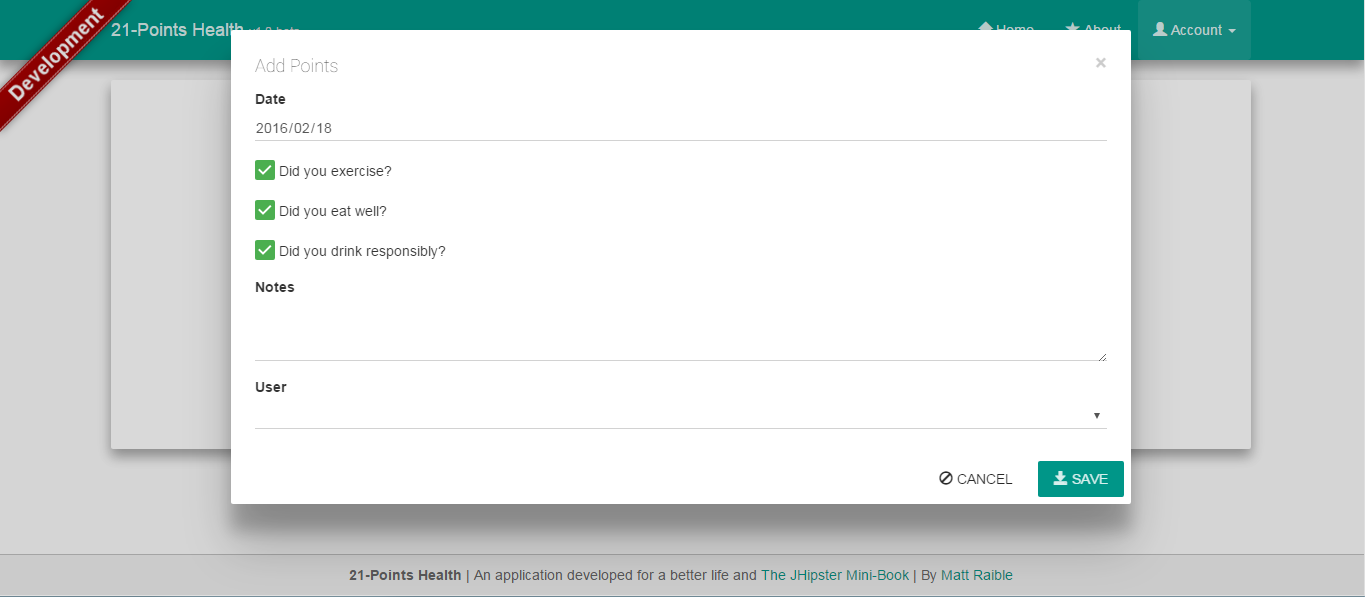
});

});

实体新增/编辑视图

scripts/entities/points/points-dialog.html

scripts/entities/points/points-dialog.controller.js



<button ui-sref=*"points.new"*>…</button>： 点击链接, 首先触发应用程序状态entity.points.new = url: '/new' 的事件onEnter()

.state('points.new', {

parent: 'points', url: '/new',

data: { roles: ['ROLE\_USER'] },

onEnter: function($stateParams, $state, $modal) {

$modal.open({

templateUrl: 'scripts/app/entities/points/points-dialog.html',

controller: 'PointsDialogController',

size: 'lg',

resolve: {

entity: function () {

return {date: null, exercise: null, meals: null, alcohol: null, notes: null, id: null};

}}

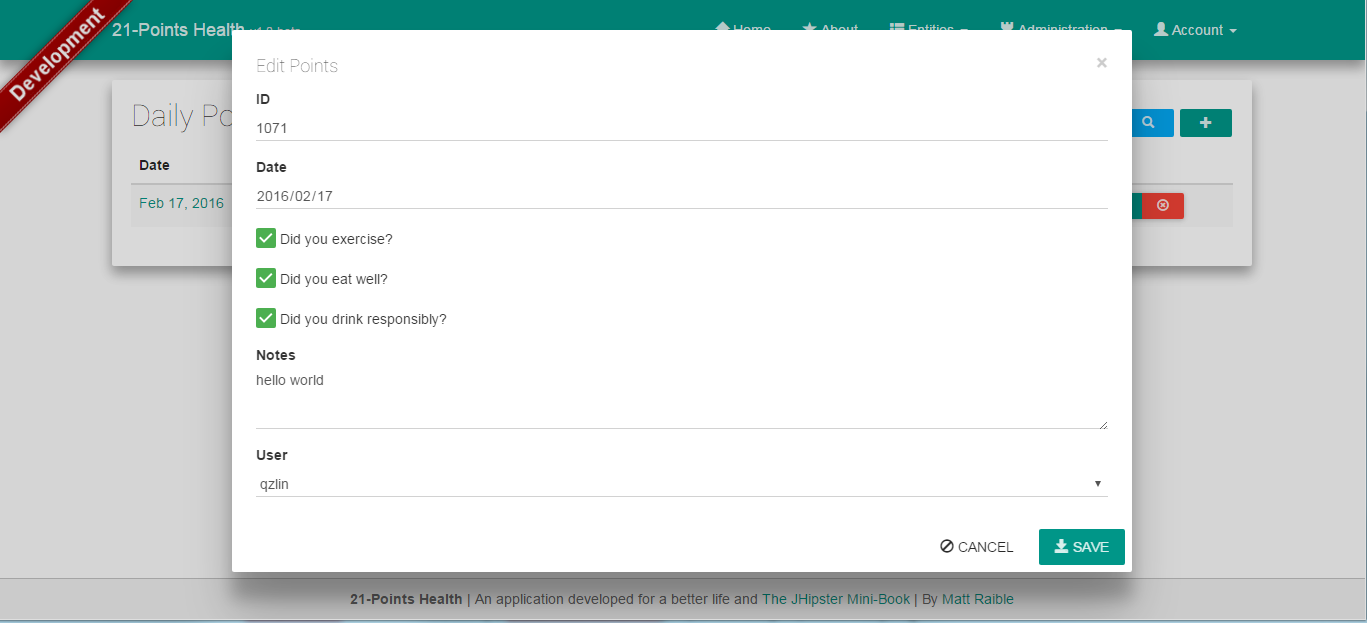
}).result.then(function(result) {

$state.go('points', null, { reload: true });

}, function() { $state.go('points'); })

}

})



<button ui-sref=*"points.edit({id:points.id})"*>…</button>： 点击链接, 首先触发应用程序状态entity.points.edit = url: '/points/{id:int}' 的事件onEnter()

.state('points.edit', {

parent: entity, url: '/{id}/edit',

data: { roles: ['ROLE\_USER'] },

onEnter: function($stateParams, $state, $modal) {

$modal.open({

templateUrl: 'scripts/app/entities/points/points-dialog.html',

controller: 'PointsDialogController',

size: 'lg',

resolve: {

entity: function(Points) {

return Points.get({id : $stateParams.id});}

}

}).result.then(function(result) {

$state.go('points', null, { reload: true });

}, function() { $state.go('^'); })

}

});

$modal.open({…})打开模态框，获取entity（若是新增，新建entity）,填充编辑视图（模态框），

由是否存在entity（判断points.id）来显示Add/Edit视图

当前用户是管理员，选择将实体关联于指定用户，若非管理员，关联用户为当前用户

每个输入框都需要校验反馈，隐藏或显示于输入框之下

<form name="editForm" role="form" novalidate ng-submit="save()" show-validation>

**<div class="modal-header">**

**<button type="button" class="close" data-dismiss="modal" ng-click="clear()">&times;</button>**

<h4 class="modal-title" ng-show="points.id"> Edit Points</h4>

<h4 class="modal-title" ng-hide="points.id">Add Points</h4>

</div>

**<div class="modal-body">**

<div class="form-group" ng-show="points.id">

<label for="id" translate="global.field.id">ID</label>

<input type="text" class="form-control" id="id" name="id" ng-model="points.id" readonly>

</div>

<div class="form-group">

<label for="field\_date">Date</label>

<input type="date" class="form-control" name="date" id="field\_date" ng-model="points.date"

required>

<div ng-show="editForm.date.$invalid">

<p class="help-block" ng-show="editForm.date.$error.required">This field is required. </p>

</div>

<div class="form-group" has-role="ROLE\_ADMIN">

<label for="field\_user">user</label>

<select class="form-control" id="field\_user" name="user" ng-model="points.user.id" ng-options="user.id as user.login for user in users"></select>

</div>

</div>

**<div class="modal-footer">**

**<button type="button" class="btn btn-default" data-dismiss="modal" ng-click="clear()">**

**Cancel</button>**

**<button type="submit" ng-disabled="editForm.$invalid || editForm.$submitted" class="btn btn-primary">Save</button>**

</div>

</form>

解析：

模态框分modal-header, modal-body, modal-footer, 关闭图标，取消和确定按钮

确定保存后，发出信号（带结果），详细视图绑定该信号，可以更新数据

控制器

controller('PointsDialogController',

function ($scope, $stateParams, $modalInstance, entity, Points, User) {

// defaults for new entries

if (!entity.id) {

entity.date = new Date();

…

}

$scope.points = entity;

$scope.users = User.query();

$scope.save = function () {

if ($scope.points.id != null) { Points.update($scope.points, onSaveFinished); }

else { Points.save($scope.points, onSaveFinished); }

};

$scope.clear = function () { $modalInstance.dismiss('cancel'); };

var onSaveFinished = function (result) {

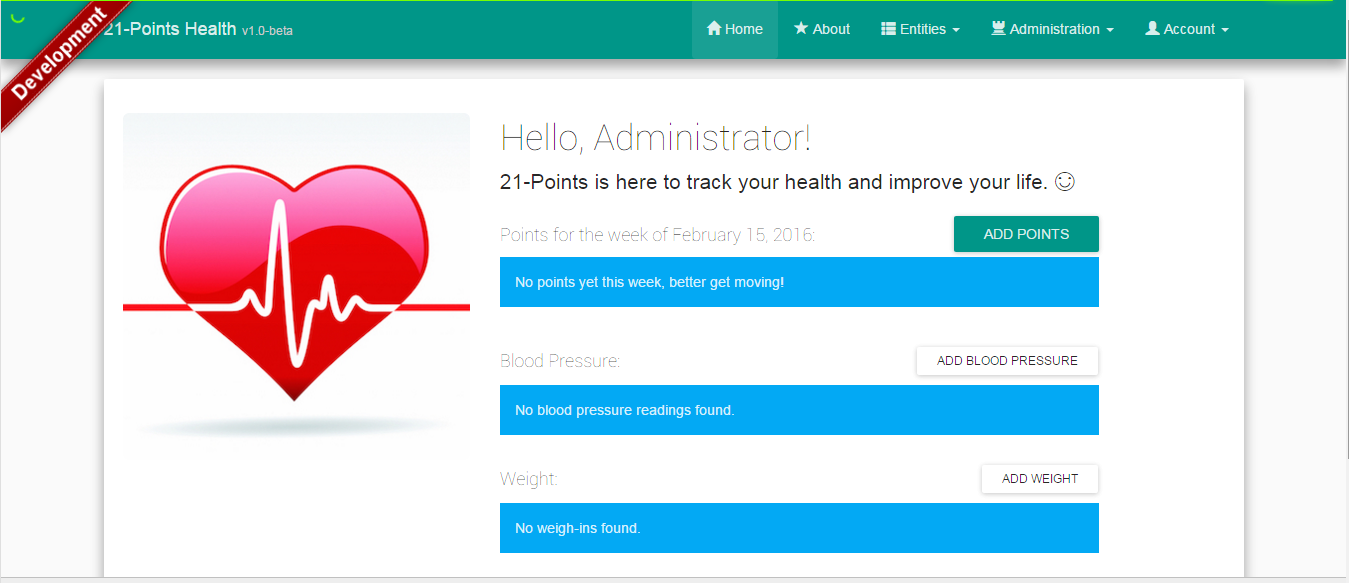
**$scope.$emit('21pointsApp:pointsUpdate', result);**

$modalInstance.close(result);

};

});

别的页面也可以入口添加实体，因为CRUD操作实体都是通过对话框，因此可以复用对话框和控制器



scripts/app/main/main.js

.state(**'points.add'**, {

**parent: 'home', url: 'add/points',**

data: { roles: ['ROLE\_USER'] },

onEnter: function($stateParams, $state, $modal) {

$modal.open({

**templateUrl: 'scripts/app/entities/points/points-dialog.html',**

**controller: 'PointsDialogController',**

size: 'lg',

resolve: {

entity: function () {

return {date: null, exercise: null, meals: null, alcohol: null, notes: null, id: null};

}

}

}).result.then(function(result) { **$state.go('home', null, { reload: true });**},

function() { **$state.go('home');** })

}

})

注意到没：如果在Points列表页面入口，则弹出对话框，关闭应回到入口页面；如果在首页(Main)页面入口，也同样弹出对话框，但关闭应回到入口页面（Main）

## Jhipster3.1.0 note

### how to use jhipster-jdl

model/jhipster-jdl.jh

entity Resume {

infoEmail String required,

infoPhone String required,

infoGithub String,

infoLinkedin String,

profileBasic String maxlength(1024),

profileTechniqueDomain String maxlength(1024),

profileSoftwareSystem String maxlength(1024),

profileMultibranchExperience String maxlength(1024),

profilePreferredPosition String maxlength(1024)

}

relationship OneToOne {

Resume{user(login)} to User

}

entity ResumeExperienceProjectAccomplish {

accomplish String maxlength(1024)

}

entity ResumeExperienceProject {

index Integer required,

name String required,

startTime ZonedDateTime,

endTime ZonedDateTime,

introduction String required maxlength(1024)

responsiility String required maxlength(1024)

platform String required maxlength(1024)

}

relationship OneToMany {

ResumeExperienceProject{accomplish} to ResumeExperienceProjectAccomplish

}

entity ResumeExperience {

index Integer required,

position String required,

company String required,

startTime ZonedDateTime required

endTime ZonedDateTime

}

relationship OneToMany {

ResumeExperience{project} to ResumeExperienceProject

}

relationship OneToMany {

Resume{experience} to ResumeExperience

}

entity ResumeSkill {

skill String maxlength(1024)

}

relationship OneToMany {

Resume{skill} to ResumeSkill

}

entity ResumePaper {

paper String maxlength(1024)

}

relationship OneToMany {

Resume{paper} to ResumePaper

}

entity ResumeEducation {

major String required,

university String required,

startTime ZonedDateTime required,

endTime ZonedDateTime required

}

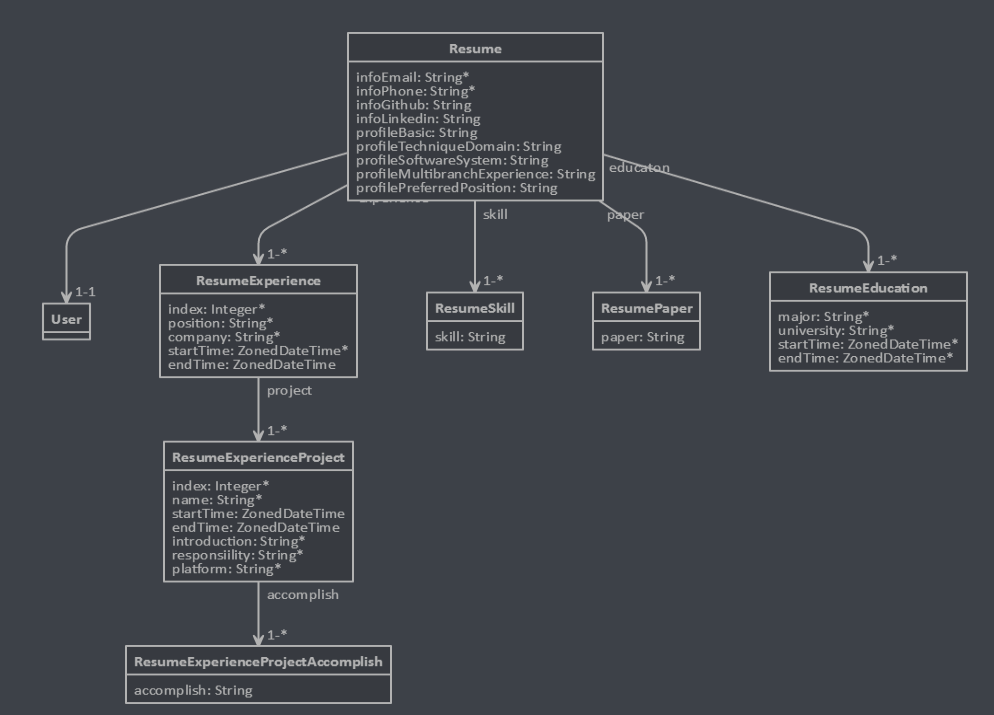
relationship OneToMany {

Resume{educaton} to ResumeEducation

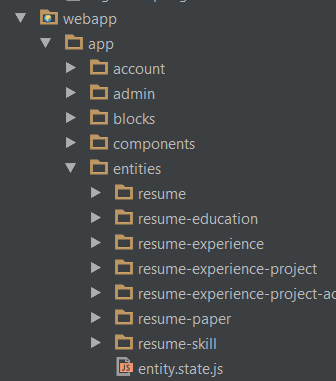
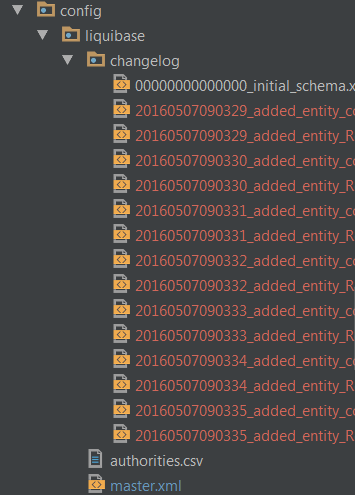
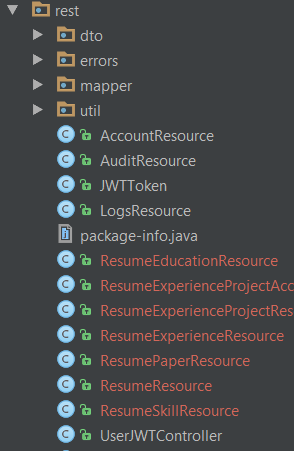
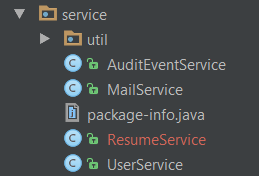
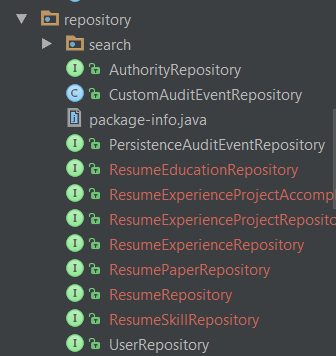
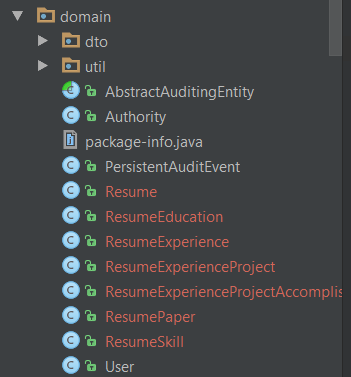
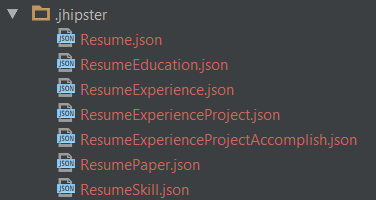
}

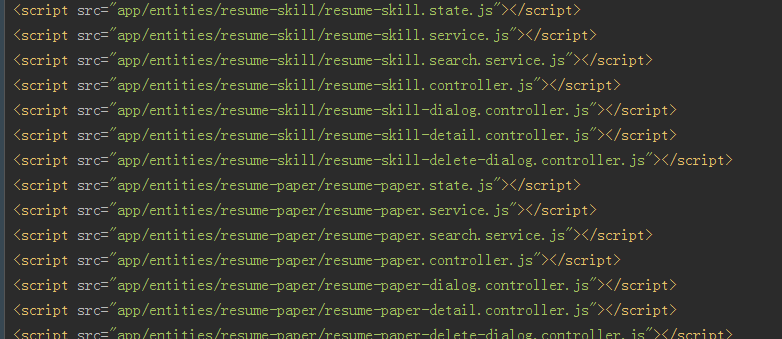
service Resume with serviceClass

<http://jhipster.github.io/jdl-studio/>



>yo jhipster:import-jdl yourJdlFilePath.jh





And test …

若关系

relationship ManyToOne {

Blog{user(login)} to User

}

则类

public class Blog {

private Long id;

private String name;

private String handle;

@ManyToOne

private User user;

}

则数据库

Blog(id:Long, name:String, handle:String, user\_id:Long)

注意Blog对象含User对象，但User对象不含Blog对象

若关系

relationship OneToMany {

Resume{skill} to ResumeSkill{resume}

}

public class Resume {

@OneToMany(mappedBy = "resume")

@JsonIgnore

@Cache(usage = CacheConcurrencyStrategy.NONSTRICT\_READ\_WRITE)

private Set<ResumeSkill> skills = new HashSet<>();

｝

public class ResumeSkill {

@ManyToOne

private Resume resume;

}

注意：数据表关系，懒加载方式，若提前加载，需要标注事务

Why should I use transactions to fetch lazy JPA relationships?

By default JPA uses lazy initialization of one-to-many and many-to-many entity relationships. If you use this default configuration, you will probably see the dreaded LazyInitializationException: it means you have tried to use an un-initialized relationship outside of a transaction.

As the generated Service class has by default the @Transactional annotation, all of its methods are transactional. This means that you can fetch all the required lazy relationship inside those business methods, without any LazyInitializationException.

Tip: use @Transactional(readOnly = true) on a method if you are not modifying any data. This is a nice performance optimization (Hibernate won't need to flush its 1st level cache, as we are not modifying anything), as well as a quality enhancement with some JDBC drivers (Oracle won't allow you to send INSERT/UPDATE/DELETE statements)

注意到Resume对象含有ResumeSkill对象的集合，ResumeSkill对象含有Resume对象，若json序列化，会产生无限循环，所以Resume对象的集合属性全部@JsonIgnore。以上是jHipster默认，当然若要获取含有所有属性的Resume,将@JsonIgnore标注于子对象所含的父对象即如下：

public class Resume {

@OneToMany(mappedBy = "resume")

@Cache(usage = CacheConcurrencyStrategy.NONSTRICT\_READ\_WRITE)

private Set<ResumeSkill> skills = new HashSet<>();

｝

public class ResumeSkill {

@JsonIgnore

@ManyToOne

private Resume resume;

}

ResumeService.java

@Transactional(readOnly = true)

public List<Resume> findAll() {

log.debug("Request to get all Resumes");

List<Resume> resumes = resumeRepository.findAll();

resumes.forEach(resume -> {

resume.getExperiences().forEach(experience -> {

experience.getProjects().forEach(project -> {

project.getAccomplishes().size();

});

});

resume.getSkills().size();

resume.getPapers().size();

resume.getEducatons().size();

});

return resumes;

}

### Front-end: resume

Pre: Angular + Bootstrap

Index.html

<script src="bower\_components/bootstrap-sass/assets/javascripts/bootstrap.js"></script>

注意：由于index.html是由gulpfile.js注入生成，

<!-- build:js app/vendor.js -->

<!-- bower:js -->

<!-- endbower -->

<!-- endbuild -->

<script src="bower\_components/bootstrap-sass/assets/javascripts/bootstrap.js"></script>

<!-- build:js app/app.js -->

<!-- inject:js -->

<!-- endinject -->

<!-- endbuild -->

当中会被覆盖，故将此<script>放入之中

请复习gulpfile.js

<script src="app/entities/resume/scrollTo.directive.js"></script> for <…scroll-to=>

(function(){

'user strict';

angular.module('resumeApp')

.directive('scrollTo', scrollTo);

scrollTo.$inject = ['$anchorScroll'];

function scrollTo($anchorScroll){

return {

restrict: 'A',

link: function (scope, element, attrs) {

element.bind('click', function (event) {

event.preventDefault();

event.stopPropagation();

var location = attrs.scrollTo;

$anchorScroll(location);

});

}

};

}

})();

resume.controller.js

activate() ;

function activate() {

$(function(){

var body = $('body');

body.scrollspy({target: ".bs-docs-sidebar"});

$(window).on("load", function(){ $('body').scrollspy("refresh");});

$(".sidebar").click(function (a) { a.preventDefault(); });

setTimeout(function () {

var sidebar = $(".bs-docs-sidebar");

sidebar.affix({

offset: {

top: function () {

var c = sidebar.offset().top, d = parseInt(sidebar.children(0).css("margin-top"), 10), e = $(".bs-docs-nav").height();

return this.top = c - d - e;

},

bottom: function () {

return this.bottom = $(".bs-docs-footer").outerHeight(!0);

}}})

}, 100) ;

});

}

**angular-fullstack (MongoDB, ExpressJS, AngularJS, NodeJS)**

// install meanjs

project-parent-directory> **mkdir template && cd template**

template > **node --version && npm --version && git --version**

template > **yo --version && bower --version && grunt --version**

template> **npm install –g generator-angular-fullstack**

template> yo angular-fullstack

Note: 若出现error, ‘rm’ is not internal command, do in the following way:

install git -> right clicked inside project folder -> git bash here -> yo meanjs

// run mongod

>mongod –dbpath ./data

**MEAN全栈开发 (MongoDB, ExpressJS, AngularJS, NodeJS)**

// install meanjs

project-parent-directory> **mkdir template && cd template**

template > **node --version && npm --version && git --version**

template > **yo --version && bower --version && grunt --version**

template> **npm install –g generator-meanjs**

template> yo meanjs

Note: 若出现error, ‘rm’ is not internal command, do in the following way:

install git -> right clicked inside project folder -> git bash here -> yo meanjs