Contents

[**Deployd** 1](#_Toc443142061)

[Jhipster 全栈开发 (Angularjs + Sprint boot ) 2](#_Toc443142062)

[install 2](#_Toc443142063)

[Basic knowledge 4](#_Toc443142064)

[如何实现异步函数？ 4](#_Toc443142065)

[AngularJS Resource：与 RESTful API 交互 5](#_Toc443142066)

[$httpProvider.interceptors 7](#_Toc443142067)

[Angular-translate 10](#_Toc443142068)

[angular-dynamic-locale 15](#_Toc443142069)

[angular-local-storage 15](#_Toc443142070)

[angular-ui-router 16](#_Toc443142071)

[21points dissect 17](#_Toc443142072)

[应用程序状态 17](#_Toc443142073)

[主页面： 20](#_Toc443142074)

[Register 21](#_Toc443142075)

[Activate 22](#_Toc443142076)

[Login 23](#_Toc443142077)

[Reset password request 25](#_Toc443142078)

[Reset password finish 26](#_Toc443142079)

[Setting (设置用户帐户信息) 27](#_Toc443142080)

[Change password 28](#_Toc443142081)

[Logout 29](#_Toc443142082)

[Authenticate 30](#_Toc443142083)

[**angular-fullstack (MongoDB, ExpressJS, AngularJS, NodeJS)** 36](#_Toc443142084)

[**MEAN全栈开发 (MongoDB, ExpressJS, AngularJS, NodeJS)** 37](#_Toc443142085)

**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

pre-condition: node, git

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

// install tools for frontend

>npm install –g yo bower grunt-cli

若无法下载，则设置proxy

**> npm config set proxy** [**http://165.225.96.34:10015**](http://165.225.96.34:10015)

**> npm config set https-proxy** [**http://165.225.96.34:10015**](http://165.225.96.34:10015)

(npm config get proxy, npm config delete proxy)

// install jhipster

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

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

Template > npm install –g generator-jhipster

Template > 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 live-reload

Template > grunt

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

// run server

Template > 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. Template > yo jhipster:entity foo

Template > yo jhipster:service user

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

Note:

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）

>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**;

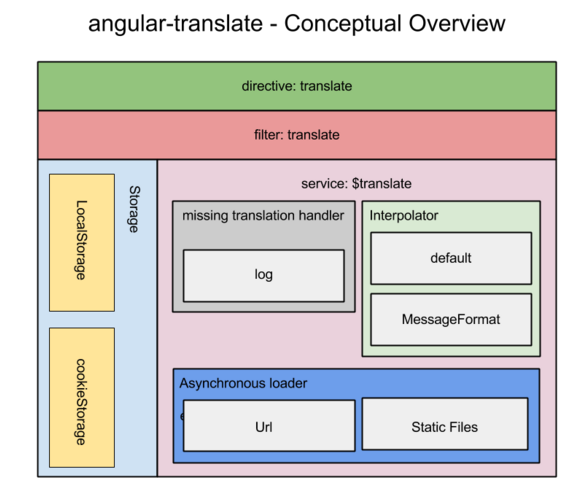
}};

});

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>

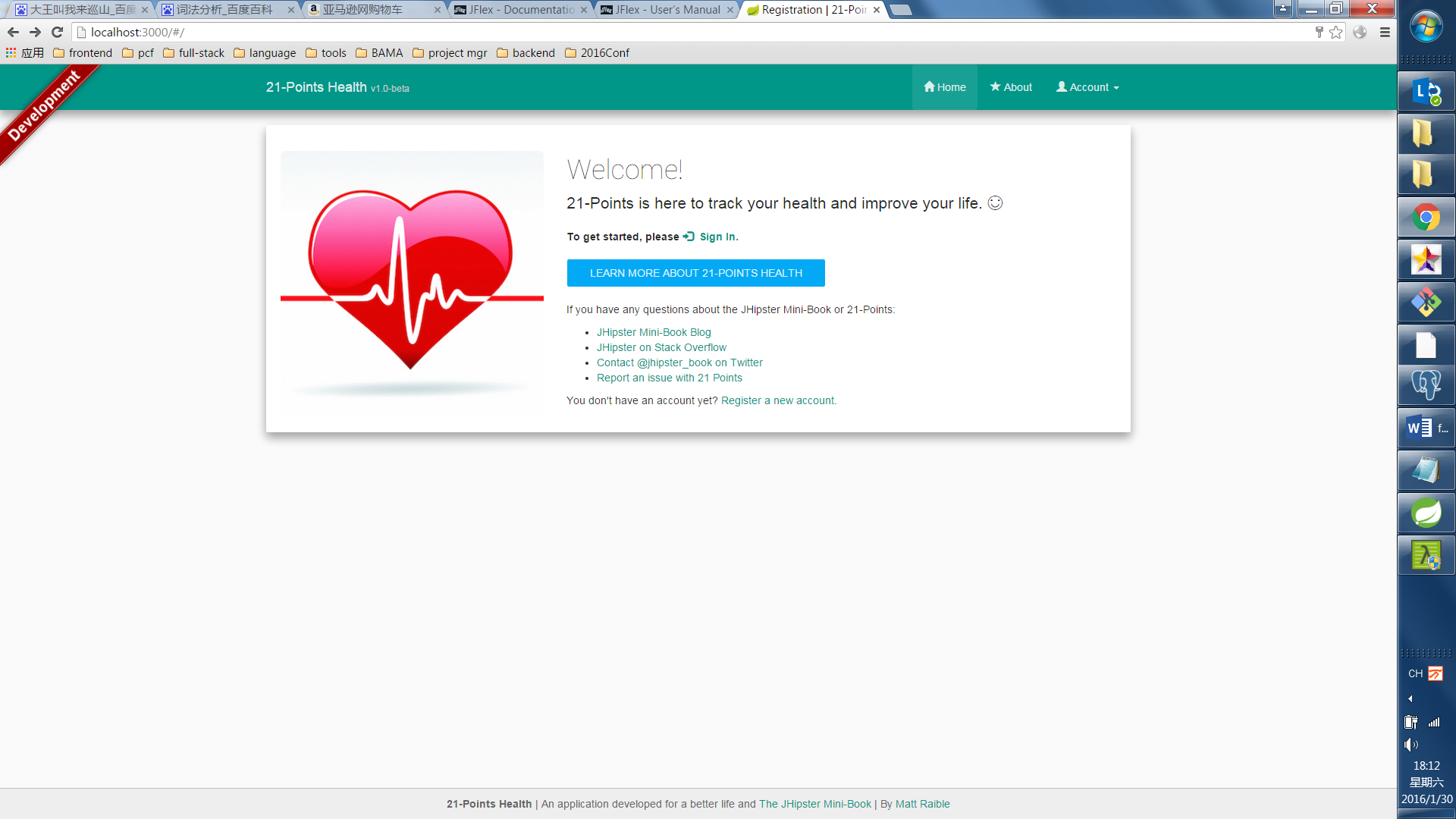
<li ui-sref-active="active" ng-switch-when="false"><a ui-sref="login">Authenticate</a></li>

<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

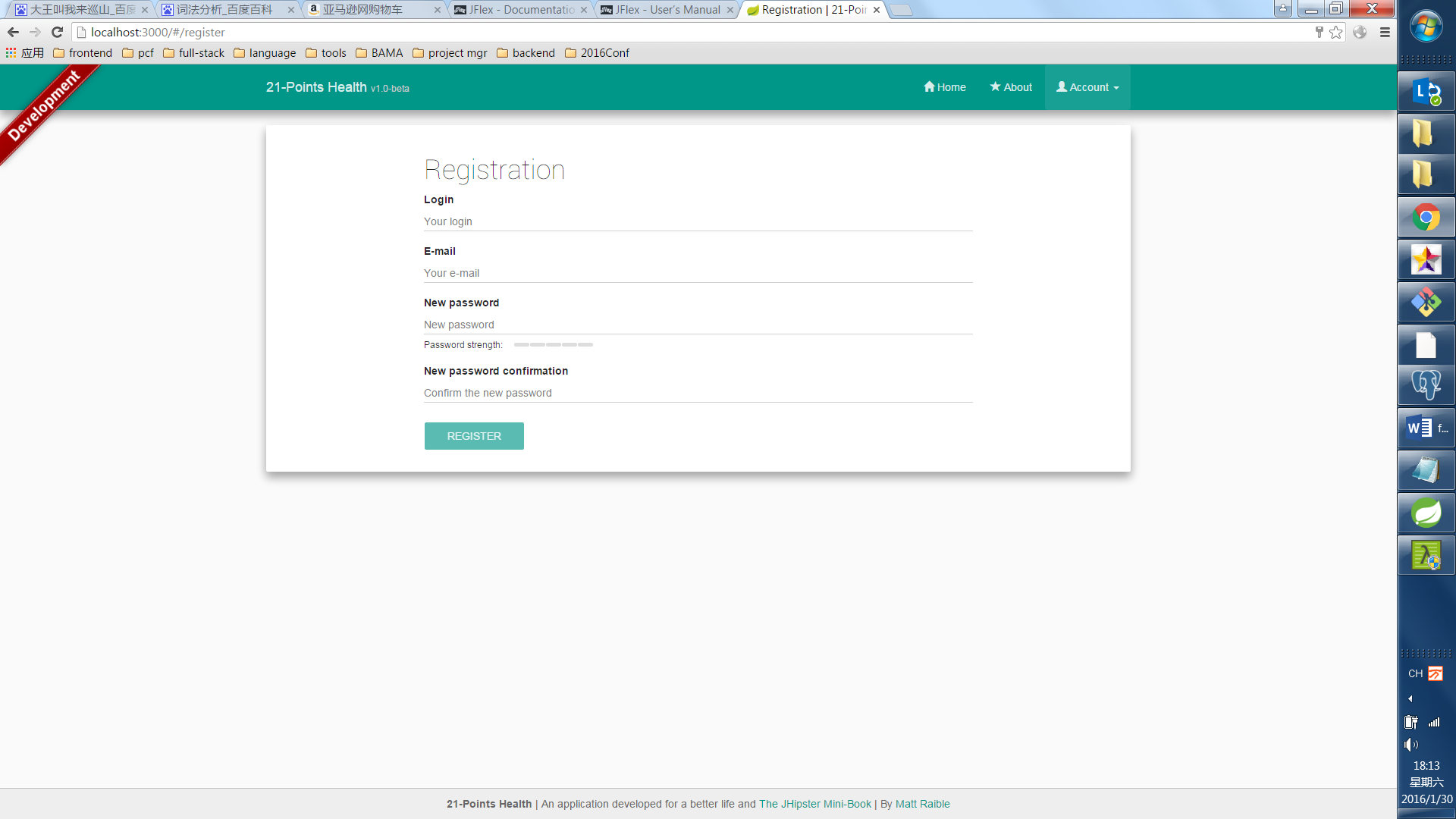
components/navbar/ navbar.directive.js

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>

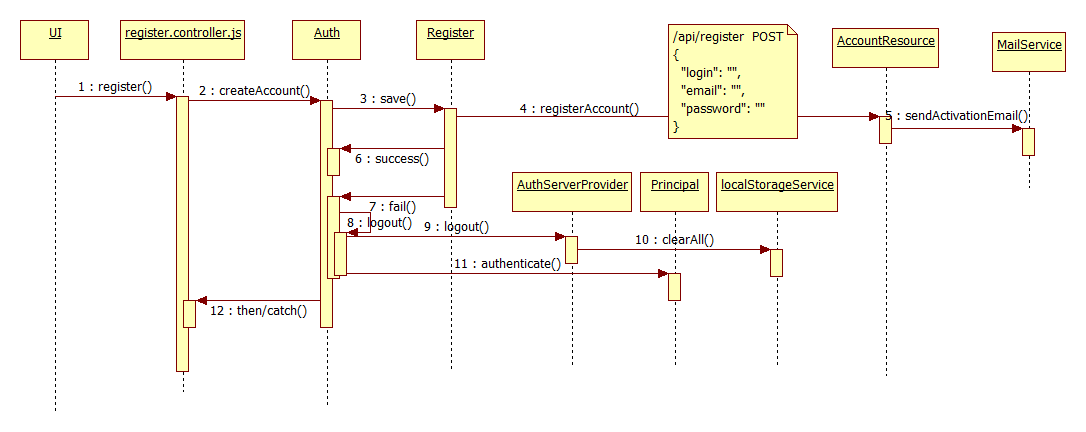
</div>

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

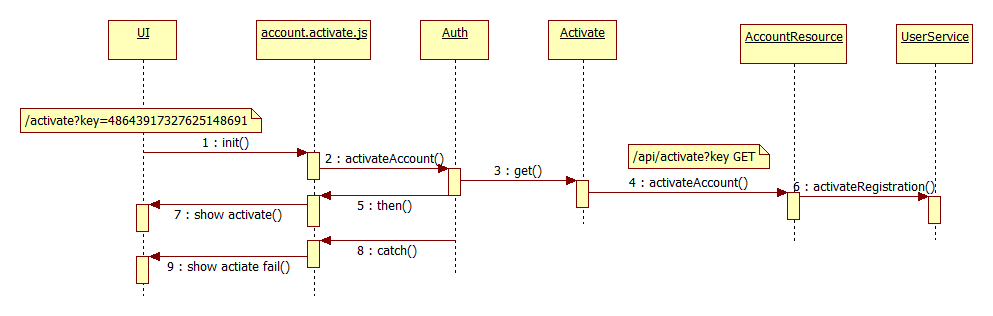
Files:

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

app/account/activate/activate.js

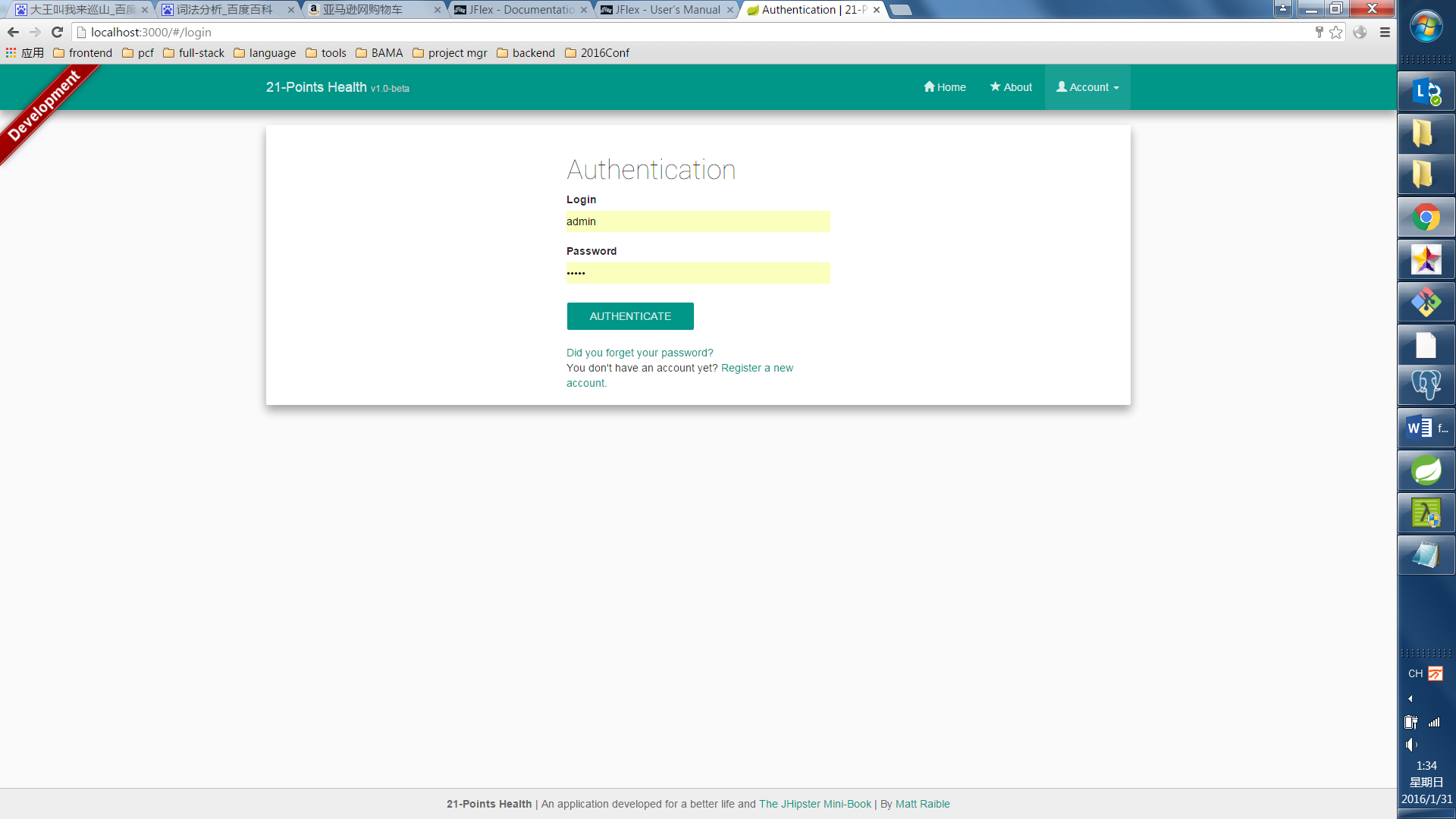
app/account/activate/activate.html

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”

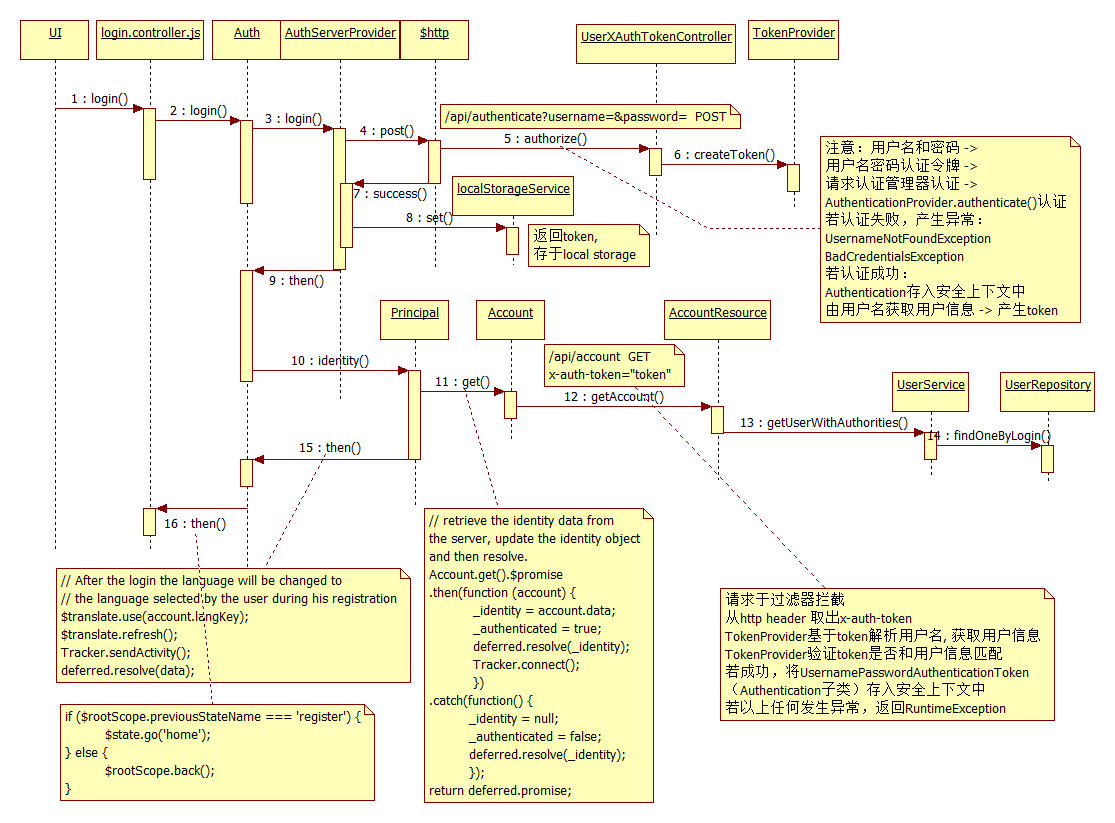
认证成功：navbar根据isAuthenticated() and has-role="ROLE\_ADMIN"更新导航菜单，路由至home页面, 此时home页面里的内容也会更新为已登录用户的内容。

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;

}

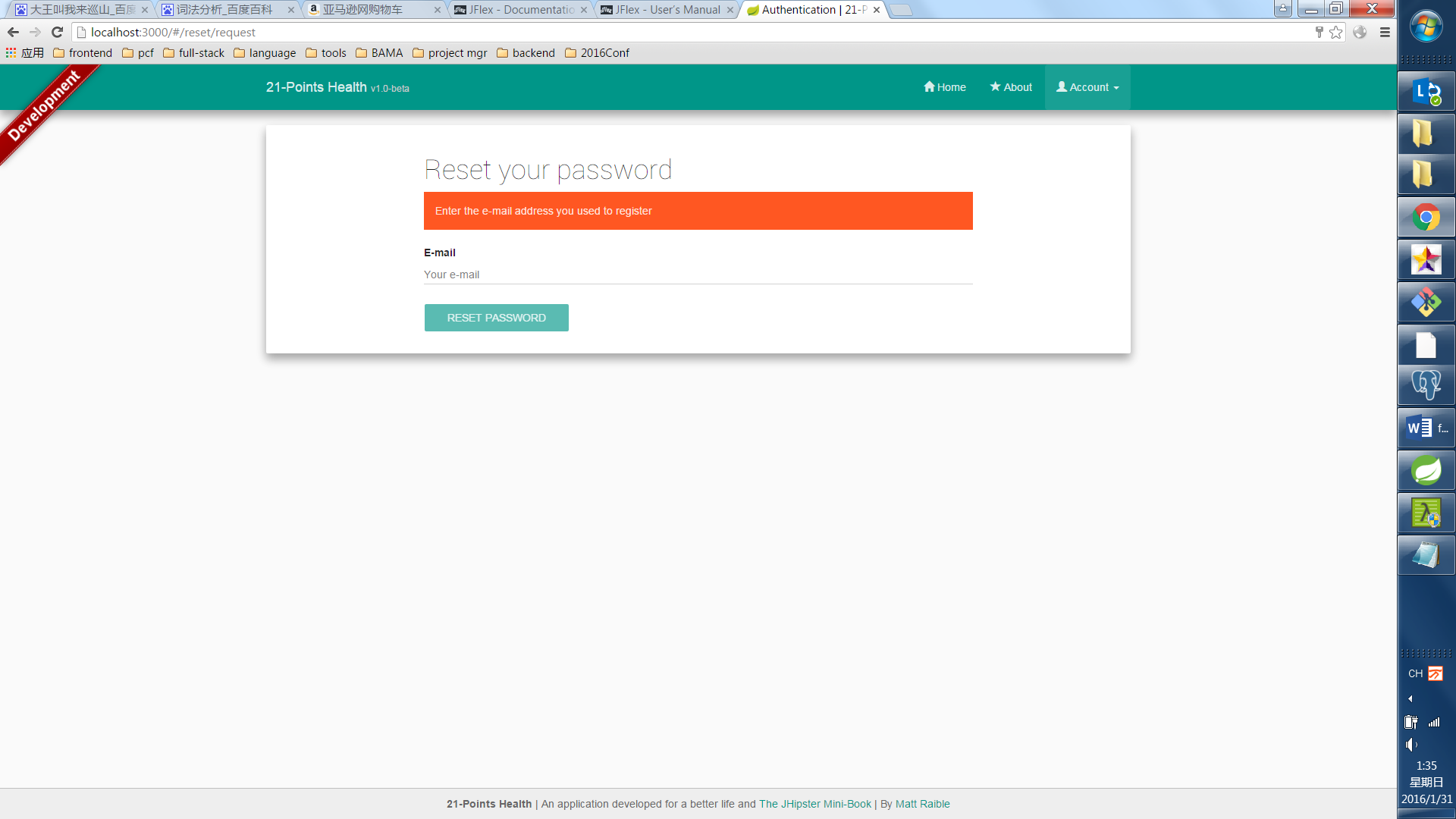
return **config**;

}};

})

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"

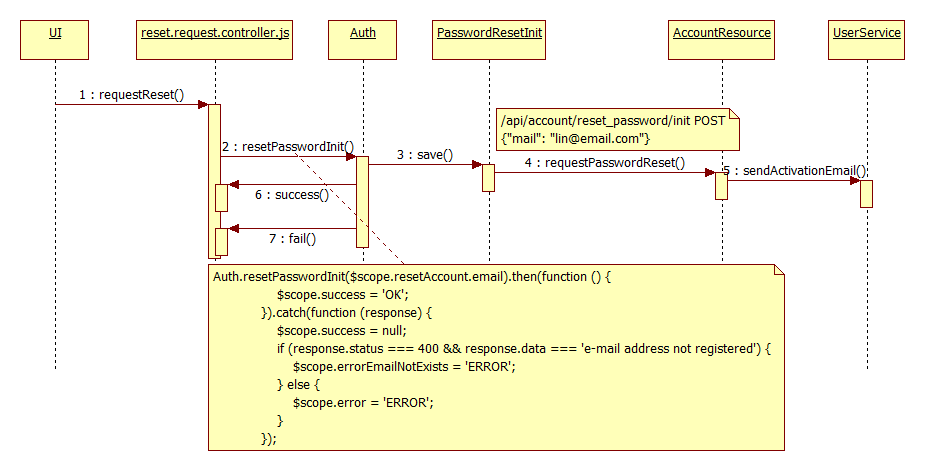
用户输入信息的校验反馈：(反馈信息直接产生于前端)，见register.html

Files:

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

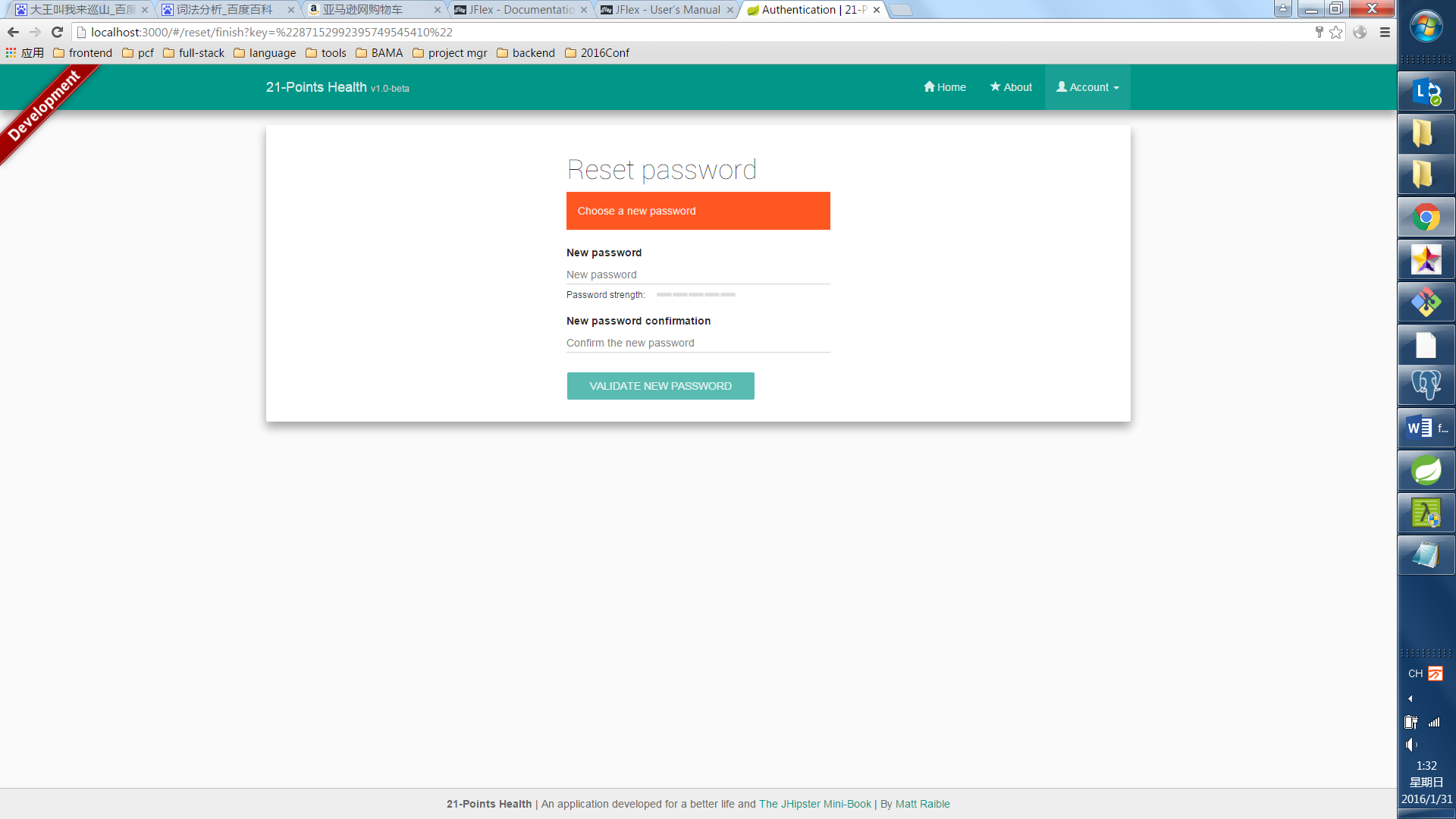
app/account/reset/request/reset.request.html

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

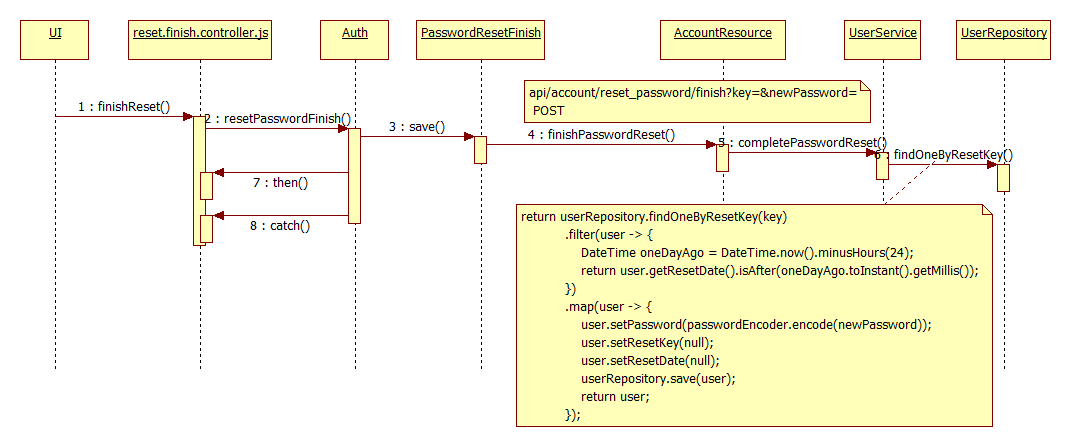
Files:

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

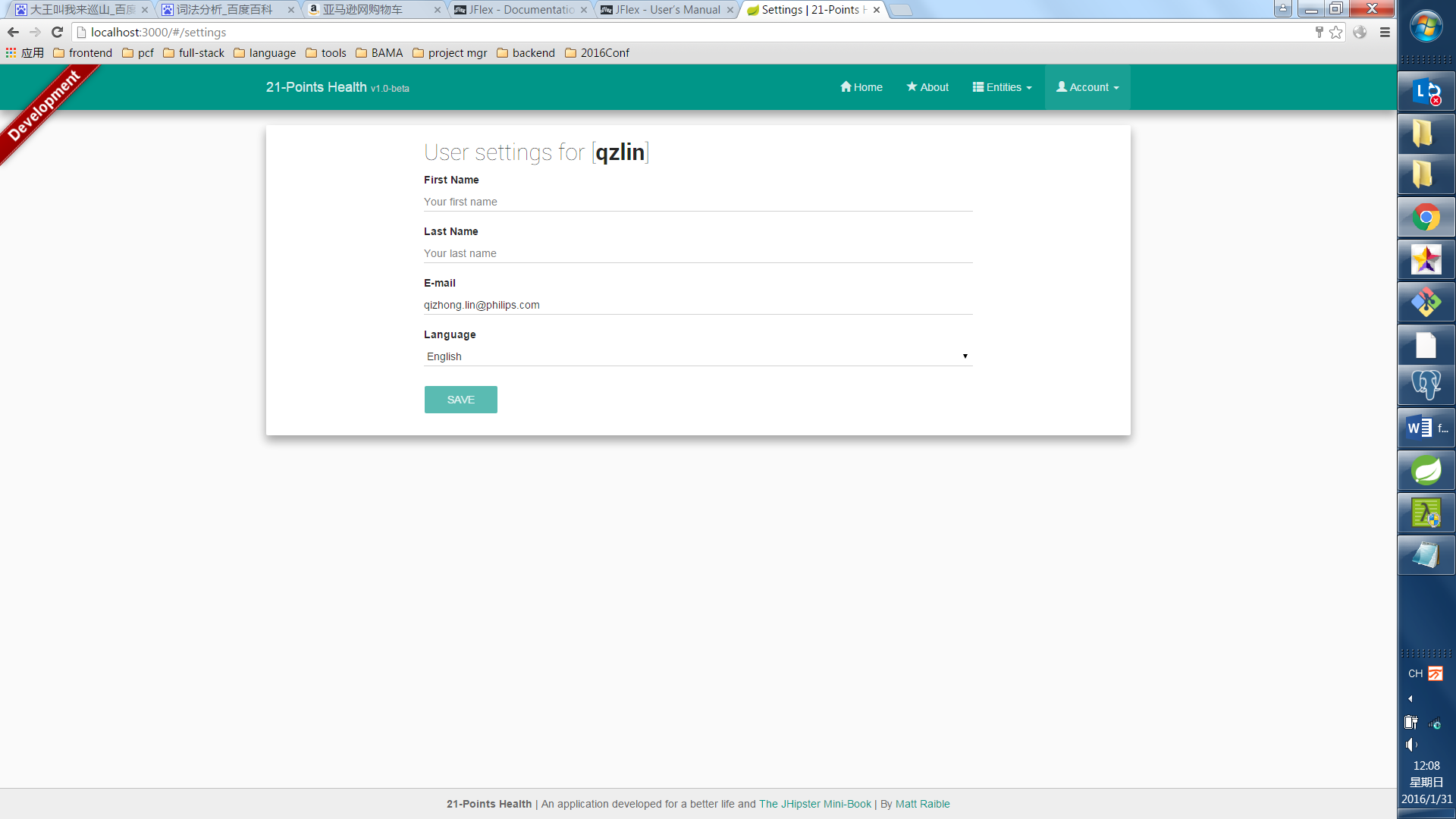
app/account/reset/request/reset.finish.js

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"

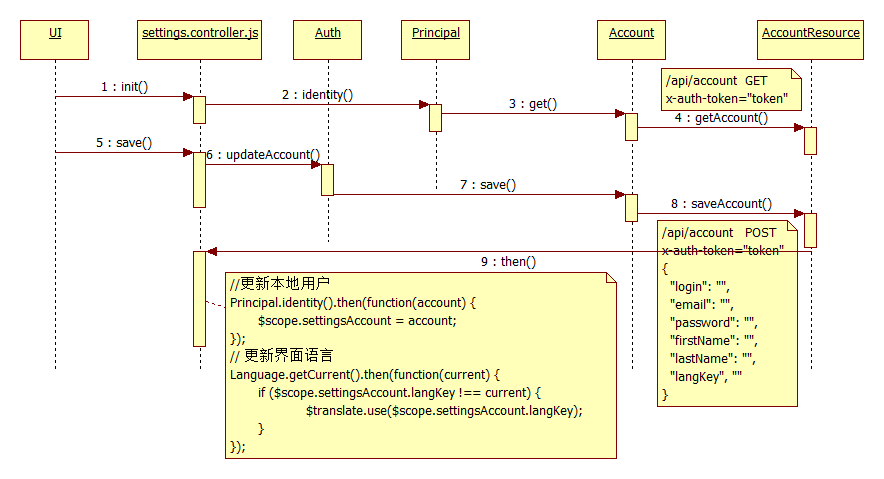
用户输入信息的校验反馈：(反馈信息直接产生于前端), 见register.html

Files:

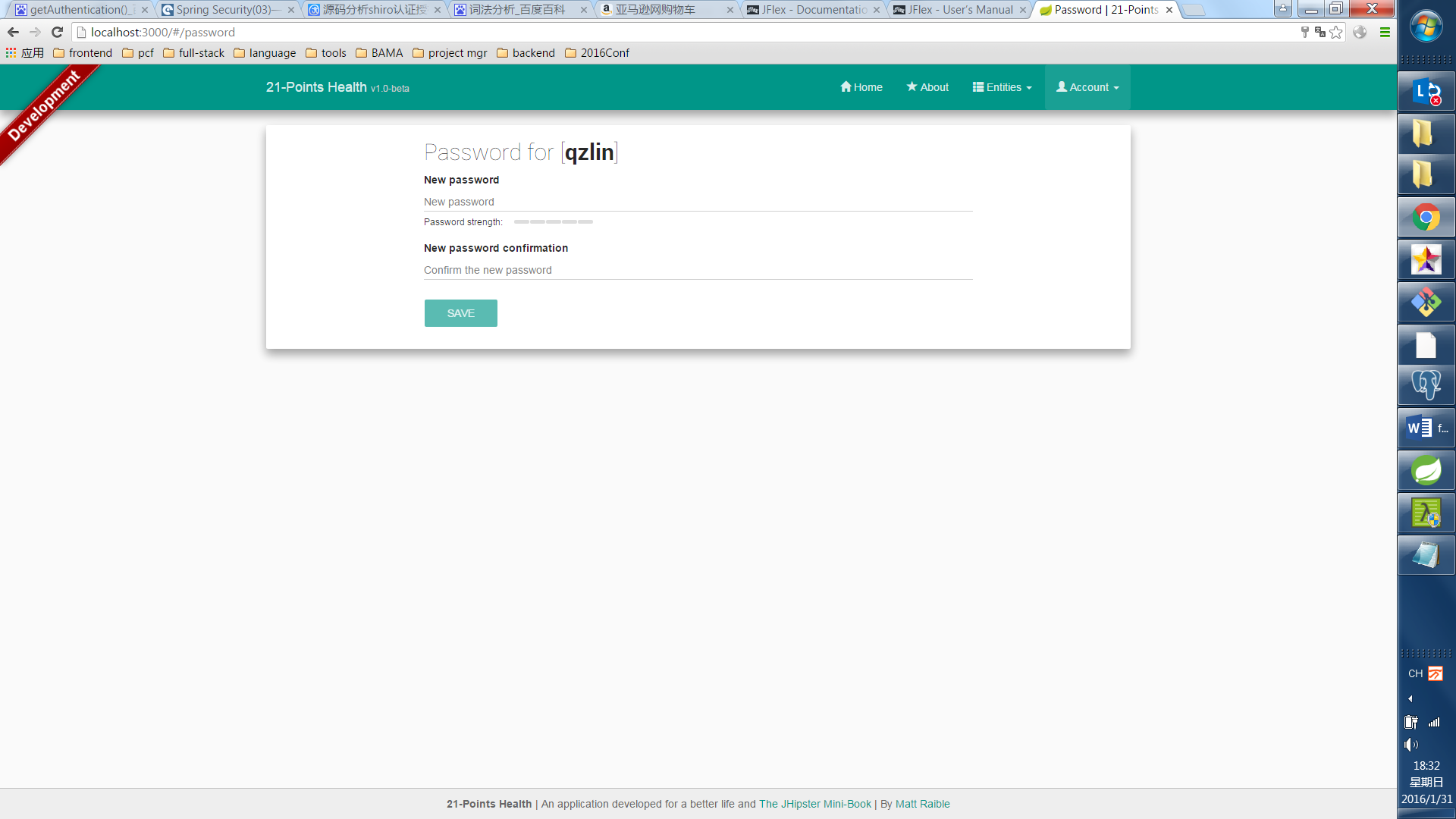
app/account/settings/settings.js

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

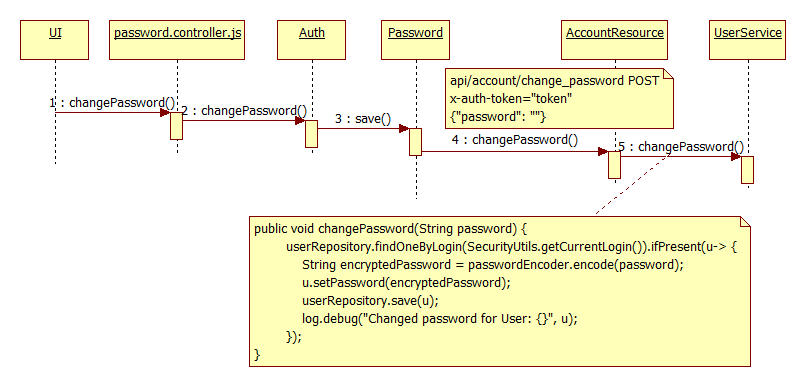
Files:

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)

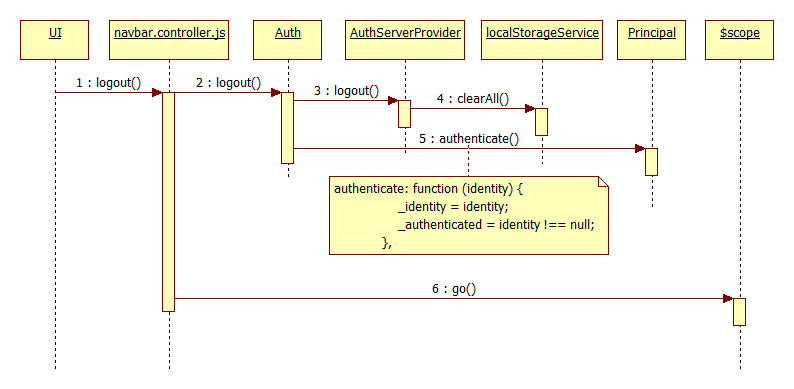
清除local storage，清空认证，浏览器路由至/#/home

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;

}

}

**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