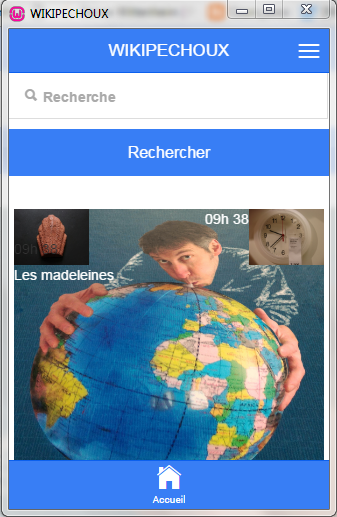
**RAPPORT TECHNIQUE**

**SITE WIKIPECHOUX**



Nathalie Ruff

Table des matières

[**1 Choix des outils . 5**](#_Toc484594759)

[**1.1 L’environnement. 6**](#_Toc484594753)

[**1.2 L’arborescence du projet. 7**](#_Toc484594754)

[**1.3 Notre page d’accueil. 8**](#_Toc484594755)

[La page index.html 8](#_Toc484594756)

[La page menu.html 9](#_Toc484594757)

[La page accueil.html 10](#_Toc484594758)

[**2 La logique d’AngularJS . 11**](#_Toc484594759)

[**2.1 Modèle-Vue-Contrôleur. 11**](#_Toc484594759)

[La vue 11](#_Toc484594760)

[Le contrôleur. 13](#_Toc484594761)

[Le modèle. 14](#_Toc484594762)

[**2.2 Two-Way Data Binding. 14**](#_Toc484594763)

[**2.3 Le routage. 15**](#_Toc484594764)

[App.js. 16](#_Toc484594765)

[**2.4 Les services. 17**](#_Toc484594766)

[**3 La page d’acceuil . 18**](#_Toc484594759)

[La route 18](#_Toc484594767)

[La partie recherche dans l’encyclopédie 18](#_Toc484594767)

[La partie centrale 21](#_Toc484594767)

[**4 La page d’affichage du mot . 23**](#_Toc484594759)

[La route 23](#_Toc484594768)

[**4.1 La partie haute de la page. 23**](#_Toc484594769)

[La page html 24](#_Toc484594770)

[Le contrôleur. 24](#_Toc484594771)

[La recherche des mots « précédent » et « suivant ». 25](#_Toc484594772)

[L’affichage des photos. 27](#_Toc484594773)

[**4.2 Partie basse de la page. 28**](#_Toc484594774)

[Les mots associés. 28](#_Toc484594773)

[La partie du bas ne figure pas pour tous les mots. 29](#_Toc484594773)

[**5 Les madeleines . 30**](#_Toc484594759)

[**5.1 Affichage de toutes les madeleines 30**](#_Toc484594775)

[La route 30](#_Toc484594776)

[La page html 30](#_Toc484594777)

[Le contrôleur 31](#_Toc484594778)

[**5.2 Affichage d’une madeleine. 31**](#_Toc484594779)

[La route 31](#_Toc484594780)

[La page html 31](#_Toc484594781)

[Le contrôleur : 32](#_Toc484594782)

[**6 Le mot aléatoire . 33**](#_Toc484594759)

[La route : 33](#_Toc484594783)

[Le fichier html : 33](#_Toc484594784)

[Le contrôleur : 34](#_Toc484594785)

[**7 Gestion de la connexion . 35**](#_Toc484594759)

[La route 35](#_Toc484594786)

[Le fichier html 35](#_Toc484594787)

[Le contrôleur 37](#_Toc484594788)

[Click sur « Annuler » 37](#_Toc484594788)

[Click sur « Mot de passe perdu » 37](#_Toc484594788)

[Click sur « Se connecter » 37](#_Toc484594788)

[Click sur « Nouveau mot de passe » 39](#_Toc484594788)

[Click sur « Enregistrer le mot de passe » 39](#_Toc484594788)

[**8 La page de gestion . 41**](#_Toc484594759)

[La route 42](#_Toc484594789)

[La page html 42](#_Toc484594790)

[Le contrôleur 43](#_Toc484594791)

[Click sur « Se déconnecter » 43](#_Toc484594791)

[Click sur « gestion de l’encyclopédie » 43](#_Toc484594791)

[Click sur un des items de la liste 44](#_Toc484594791)

[**9 Insertion d’un nouveau mot . 45**](#_Toc484594759)

[La route 45](#_Toc484594792)

[La page html 45](#_Toc484594793)

[Le contrôleur 46](#_Toc484594794)

[Click sur « Rechercher » 47](#_Toc484594791)

[Click sur « Enregistrer » 47](#_Toc484594791)

[**10 Nouvelles photos . 48**](#_Toc484594759)

[La route 49](#_Toc484594795)

[La page html 49](#_Toc484594796)

[Le contrôleur. 50](#_Toc484594797)

[Click sur « Enregistrer » 50](#_Toc484594791)

[**11 Nouvelle vidéo . 51**](#_Toc484594759)

[La route 51](#_Toc484594798)

[La page html 51](#_Toc484594799)

[Le contrôleur 51](#_Toc484594800)

[**12 Modifier *un* mot . 52**](#_Toc484594759)

[La route 52](#_Toc484594801)

[La page html 52](#_Toc484594802)

[Le contrôleur 52](#_Toc484594803)

[**13 Modifier *le* mot . 53**](#_Toc484594759)

[La route 53](#_Toc484594804)

[La page html 53](#_Toc484594805)

[Le contrôleur 54](#_Toc484594806)

[**14 Supprimer un mot . 54**](#_Toc484594759)

[La route 54](#_Toc484594807)

[La page html 54](#_Toc484594808)

[Le contrôleur 55](#_Toc484594809)

[**15 Gestion des madeleines . 56**](#_Toc484594759)

[**16 Gestion des montres . 56**](#_Toc484594759)

[La route 56](#_Toc484594810)

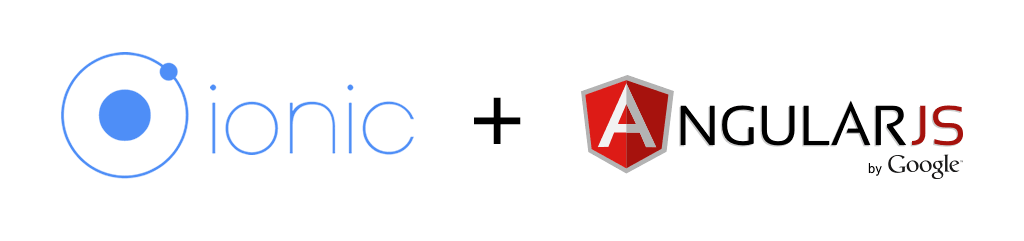
[La page html 56](#_Toc484594811)

[Le contrôleur 57](#_Toc484594812)

[Click sur « Rechercher » 57](#_Toc484594791)

ANNEXE : MCD de la base de données page 59

1. Choix des outils



**Ionic est un framework open source utilisé pour développer des applications mobiles hybrides.**

Depuis quelques années, le développement d'applications hybrides est devenu un enjeu majeur car elles permettent de réduire considérablement les coûts liés au développement.

Et c'est avec l'arrivée des Framework JS, tels qu’AngularJS, que celles-ci ont pu atteindre un niveau de performance presque équivalent à celui des applications natives.

On trouve beaucoup d'avantages à utiliser Ionic Framework pour développer des applications mobiles et web. On s'appuie sur AngularJS, ce qui nous permet d'avoir un développement rapide.

La gestion de la logique et des données est assurée par le framework AngularJS. Ce framework d'applications web fournit aux développeurs une bonne structure d'application et la possibilité d'écrire des applications complètes rapidement.

Angular est un framework JavaScript développé par Google libre et open source qui permet d'améliorer la syntaxe de JavaScript et d'augmenter la productivité de développement. Angular s'appuie sur la logique Modèle - Vue - Controlleur (MVC).

Créé en 2009, il est devenu incontournable pour le développement web.

# L’environnement.



Apache **Cordova** est une plateforme servant à construire des applications natives mobiles en utilisant HTML/CSS/Javascript. Ionic s’appuie sur celui-ci (ainsi que sur **AngularJS**) pour construire ses applications. Cette plateforme est installée automatiquement avec Ionic.

Avant toute chose il faut s’assurer que vous avez bien installé Node.js pour ceci passez en mode commande et tapez les commandes suivantes :



Vérifiez que Ionic est installé :

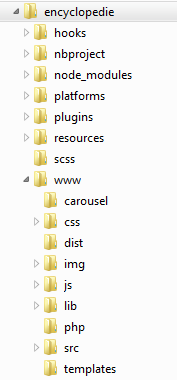


Pour pouvoir construire votre application pour Android il faut que le SDK Android soit installé :



Seuls les utilisateurs de Mac OS pourront développer des applications iOS. À la différence d'Android, il n'est pas possible d'avoir accès au SDK iOS sur d'autres systèmes que iOS.

# L’arborescence du projet.



**Dossier hooks/** : Contient les scripts de "hooks" qui permettent à un moment d’exécuter des tâches. Il n’a pas été utilisé dans le projet.

**Dossier platforms/** : Contient les données de chaque plateforme. Si vous regardez le contenu du répertoire généré pour votre projet, vous trouvez le répertoire android. Et dans ce répertoire, vous avez le code généré pour une application Android comme AndroidManifest.xml, build.gradle... ou encore les répertoires de travail d'une application Android comme assets, src...

**Dossier plugins/** : Contient les plugins Cordova. Toute utilisation d'éléments "avancés" des téléphones sortant du domaine Web (Caméra, Giroscope, Splashscreen) passent par un plugin Cordova. Pour résumer, les plugins permettent de faire le lien avec les fonctions natives du SDK de votre téléphone et le javascript de votre application.

**Dossier scss/** : Contient les scripts Sass (notamment Ionic).

**Le dossier www/** : Contient votre application.

Www est le dossier principal de travail pour les développeurs ioniques. Vous passerez la plupart de votre temps ici. Ionic nous donne la structure de leur dossier par défaut à l'intérieur "www", mais vous pouvez toujours changer pour l’accommoder vos propres besoins. Lorsque vous ouvrez ce dossier, vous trouverez:

* dossier **css** pour nos propres fichiers CSS**.**
* **img** pour nos images**.**
* **js :** Dans ce répertoire, on a l'habitude de stocker les fichiers JavaScript de l'application. Dans votre projet, vous trouverez au minimum les scripts app.js, controller.js et services.js.
* **libs :** Nous pouvons ajouter des librairies JavaScript supplémentaires à notre projet en plus de celles de Ionic. Celles-ci sont stockées dans ce répertoire.
* **Templates :** Ce répertoire contient les fichiers HTML qui vont constituer l'ensemble de l’application. Nous allons définir, dans ce répertoire, les pages correspondant aux différents onglets ou bien au menu ou encore à la page de login...(bref les vues de l’application).
* **index.html.** C'est le point d'entrée de l'application. On importe les librairies Ionic et Cordova dans ce fichier, on y définit aussi les CSS (ceux d'Ionic et nos propres CSS). On importe aussi dedans nos propres scripts JavaScript.

# Notre page d’accueil.

## La page index.html

Outre les imports des librairies la page index.html contient ce qui sera commun à toutes les vues ici la ion-nav-bar (la barre de navigation) et les balises ion-nav-view : il va prendre le code des pages HTML qu’il récupère dans le modèle dans app.js et l’injecter dans cette balise dans l’index.html. Dans chaque nouvelle page, le code html sera contenu dans les balises : <ion-view view-title="WIKIPECHOUX" > <ion-content > ………………….. </ion-content></ion-view>.

Nous avons ajouté une barre de navigation à notre application dans le corps **index.html** en ajoutant l'élément **ion-nav-bar**.

<body ng-app="app" ng-controller="AppCtrl" animation="slide-left-right-ios7" >

<div>

<div>

<ion-nav-bar class="bar-stable">

<ion-nav-back-button></ion-nav-back-button>

</ion-nav-bar>

<ion-nav-view></ion-nav-view>

</div>

</div>

</body>

## La page menu.html

L'un des modèles de navigation les plus courants dans les applications mobiles est un menu latéral. Il s'agit d'un tiroir qui glisse de côté et expose les liens de navigation. Par conception, ils sont hors écran et sont ouverts par une sorte de bouton, souvent l'icône du Burger.

Un menu latéral comprend plusieurs composants :

ionSideMenus , ionSideMenu et ionSideMenuContent .

Afin de facilité la navigation, ici le retour à la page d’accueil, nous avons ajouté la balise <ion-tabs> qui peut contenir plusieurs <ion-tab>. Celui-ci apparait en bas de l’écran.



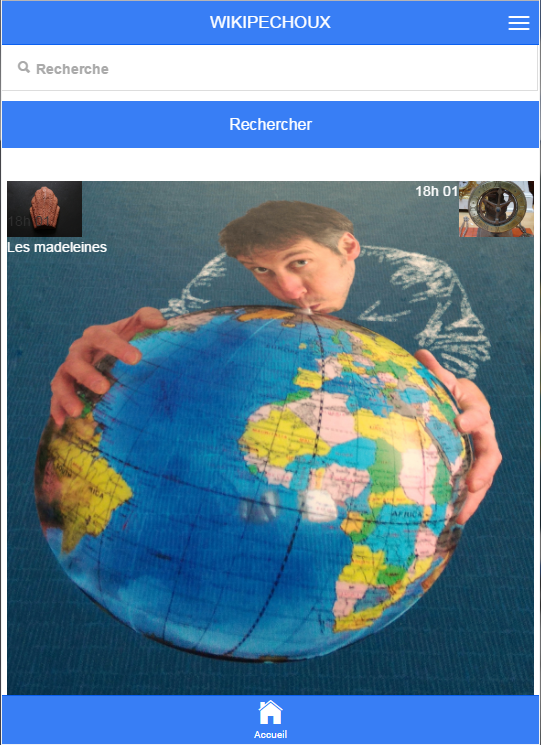
## La page accueil.html

C’est la page visible lors du lancement du site.

Zone de saisie pour la recherche d’un mot.

Menu burger.

Bandeau de titre.



L’horaire de connexion sous forme d’une image.

Lien direct vers l’affichage des madeleines.

« Ion-tabs « pour faciliter la navigation vers la page d’accueil.

1. La logique d’AngularJS.

# Modèle-Vue-Contrôleur.

AngularJS repose sur le design Pattern (« patron de conception » en français) **Modèle-Vue-Contôleur** (MVC). Chaque page est divisée en un Modèle, une Vue et un Contrôleur.

AngularJS permet aussi d’ajouter d’autres outils comme les **services** et les **directives**.

Mais le plus important de tous est bien le Contrôleur puisque c'est celui qui contient la **logique** de la page.

## La vue

C'est ce que voit l'utilisateur final. Elle est générée à partir du **template**. Le template est le **fichier HTML *enrichi*** de certains attributs et balises propres à AngularJS que l'on découvrira au fur et à mesure.

C'est la partie la plus facile à lire et à écrire.

**Le moteur de template :**

AngularJS, intègre nativement un [moteur de template](http://en.wikipedia.org/wiki/Web_template_system). Ceci permet d'utiliser des *raccourcis syntaxiques* pour afficher des variables ou même des entités logiques telles que les boucles et bien d'autres choses encore.

Il existe beaucoup de moteurs de template pour le web, en particulier pour PHP. Twig (celui intégré à Symfony2), par exemple, a une syntaxe très proche d'AngularJS notamment avec les **doubles accolades** .

Voici quelques exemples :

**Le préfixe ng-**

Marque de fabrique d'AngularJS, vous remarquez qu'on le trouve à toutes les balises où une action définie par le Javascript sera exécutée. Ce sont des **directives** AngularJS.

Une directive est un marqueur sur un élément du DOM (Document Object Model : il définit la structure logique d'un document et la manière d'y accéder et de le manipuler) qui dit au compilateur HTML d'Angular d'associer un comportement spécifique au DOM concerné et à ses éléments enfants.

* Dans *index.html* nous trouvons la ligne : <body ng-app="app" ng-controller="AppCtrl">

**La directive ng-app**

Cette directive initialise l'application app : AngularJS sait quel module il va devoir lire pour la page courante. Il génère par la même occasion le $scope de la page.

**La directive ng-controller**

Cette directive permet de dire à AngularJS que le contrôleur de cette section est AppCtrl. Les variables qui seront à l'intérieur de cette balise seront accessibles et modifiables depuis AppCtrl.

* Dans *accueil.html* nous avons les lignes :

<ion-list>

<ion-item class="item-icon-right" data-ng-repeat="mot in dico" type="item-text-wrap"

ui-sref="wiki.laDefinition({idMot: mot.id,nom\_mot: mot.nom\_mot,definition: mot.definition, theme: mot.id\_theme})">

<h2>

{{ mot.nom\_mot }} ……………………………

**Les doubles accolades**

On affiche le nom du mot avec une double accolade : {{ mot.nom\_mot }}. Grâce au moteur de template d'AngularJS, cette séquence sera automatiquement ( c'est-à-dire sans que le développeur n'ait à paramétrer quoi que ce soit ! ) repérée et remplacée par la valeur de la variable mot.nom\_mot. C'est magique !

**La boucle ng-repeat**

Nous remarquons également la **directive** ng-repeat. Elle permet de dire à AngularJS qu'il doit répéter l'élément en question (et tous ses éléments enfants) tant qu'il y a des entrées dans le tableau dico... mais pas seulement ! À l'image d'une boucle foreach ($tableau as $entree) de PHP, on dispose de la variable temporaire mot à l'intérieur de la boucle, avec toutes ses propriétés. Ce qui est très pratique pour afficher son nom par exemple.

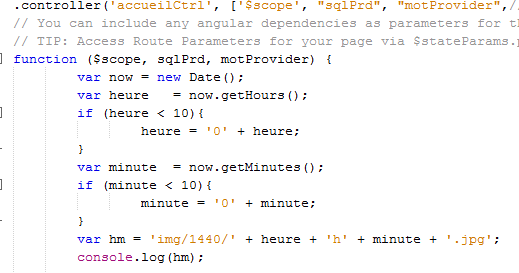
Bref, vous l'aurez compris, ce *moteur de template* va grandement vous faire gagner en clarté et rapidité.

## Le contrôleur.

C'est le cœur du code correspondant à la page. On met dans le contrôleur toute sa **logique**, c'est-à-dire la description des *actions* que la page doit être capable de réaliser.

On verra que tout ce qui ne correspond pas à ces *actions* aura sa place dans les services ou directives. En pratique, ce n'est qu'une *grande fonction* un peu particulière.

Une partie du contrôleur de la page d’accueil :



* **La déclaration du contrôleur :**

On déclare le contrôleur avec la fonction controller().

On donne à ce contrôleur :

Un nom (ici accueilCtrl) qui permettra de l'appeler quand on en aura besoin.

La liste de ses dépendances. Celles-ci peuvent être des services ou des modules.

AngularJS met à notre disposition de nombreux outils qui ont chacun un rôle précis, les services. Chacun commence par un $. Ici, nous n'avons besoin que du service $scope.

Les autres dépendances sont des modules crées pour l’application.

Ensuite apparait la fonction elle-même.

## Le modèle.

Le modèle selon AngularJS, c'est l'**ensemble des scopes de la page**.

On peut se représenter ces familles de scopes sous la forme d'un arbre, comme un arbre généalogique par exemple. La racine de tous les scopes est le $rootScope qui est instancié lorsqu'AngularJS voit la directive ng-app (voir index.html). Vient ensuite le scope du contrôleur instancié par la directive ng-controller. Enfin, certaines directives créent leur propre scope, comme ici ng-repeat.

Nous en verrons le fonctionnement lors des explications de certaines pages spécifiques.

# Two-Way Data Binding.

La plupart des frameworks qui permettent de créer des applications web utilisent cette notion de data-binding. C'est le cas par exemple de Knockout.js, mais c'est également le cas pour AngularJS. Il s'agit d'un moyen de lier la partie vue à la partie logique. En d'autres termes, grâce à cela, les éléments de votre code HTML seront liés à votre contrôleur JavaScript.

Le Two-Way Data Binding, en français, on pourrait le traduire par « Lien bilatéral des données ». Autrement dit, les données de deux domaines distincts sont liées entre elles et dans les deux sens.

Les deux domaines sont ici le **modèle** et la **vue** : lors de la génération de la vue, au chargement de la page, AngularJS crée le modèle, et initialise les variables des scopes selon ce qui est déclaré dans le contrôleur et le template. D'ailleurs, en cas de conflit entre les deux, le contrôleur prendra toujours le dessus sur le template. Puis à partir de ce modèle, il génère enfin la vue.

Ce concept est au centre du framework, et pour cause, il simplifie grandement la tâche du développeur. En effet, il assure via de nombreux services (dont le $scope), que les variables de la vue et du modèle soient **constamment synchronisées**.

# Le routage.

**Son fonctionnement**

Pour comprendre comment fonctionne le routage, il faut revenir au principe de base d'AngularJS : le framework permet de déplacer toute la logique de présentation  **côté client**. Or en déportant aussi le routage, le navigateur du client doit pouvoir être indépendant du serveur pour naviguer d'une page à l'autre.

Le serveur ne guide absolument pas la visite, c'est le client qui fait des demandes asynchrones des objets dont il a besoin.

Ce système va vous permettre de définir comment les écrans sont appelés et pour associer un contrôleur à un template HTML (entre autres).

Ainsi, à chaque page visitée, **le navigateur ne formule plus une requête directement au serveur, mais à AngularJS**. Celui-ci sait alors s'il faut télécharger de nouveaux documents, et comment composer la page demandée.

*Mais comment fait AngularJS pour intercepter ces requêtes ?*

En fait, il utilise une **ressource des URL** justement **réservée aux navigateurs** et qui n'est pas envoyée dans les requêtes : les **ancres**.

En effet, les ancres ne sont traitées **que** par le navigateur, après réception de la réponse du serveur à sa demande. Et, alors que changer d'ancre au sein d'un document **modifie bien l'URL** et l'*état* de la page, cela ne renouvelle en aucun cas une requête auprès du serveur.

Finalement, AngularJS a sa solution toute trouvée c'est la raison pour laquelle tous les liens de l’application seront de la forme : http://www.exemple.com/**#**/contact, le # étant le délimiteur du début de l'ancre.

Une dernière remarque : le fichier index.html doit évidemment être dans le dossier pointé par l'adresse http://www.exemple.com/puisque c'est lui qui lance toute l'application.

## App.js.

C'est le point d'entrée JS de notre application.  C'est dans ce fichier qu'on aura notre gestion de route pour l'application, il appellera également tous nos contrôleurs.

La première ligne du fichier :

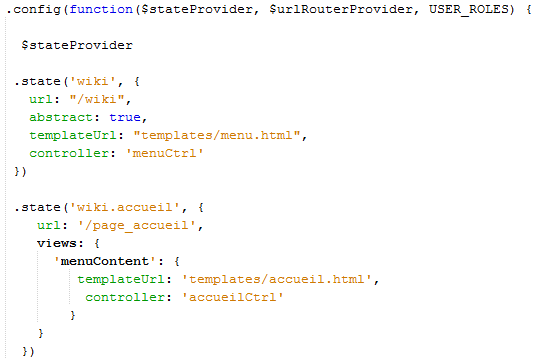
var app = angular.module('app', ['ionic',"ngRoute", 'app.controllers', ……………….

Le module « app » (en lien avec ng-app du fichier index.html) inclus des dépendances.

Ensuite on passe à une fonction « run », pour plus de renseignements voir la documentation.

Puis on passe à la définition de quelques directives détaillées plus loin.

Enfin, les routes. Voici un extrait de ce paramétrage :



abstract correspond à la création d'une route abstraite qui permet de créer aisément un système "d'héritage" sur les routes. Vous vous souvenez, menu.html contient le menu de navigation. Celui-ci sera donc commun à tous les fils. Il est positionné sur l’URL « /wiki »

Le contrôleur associé.

La vue.

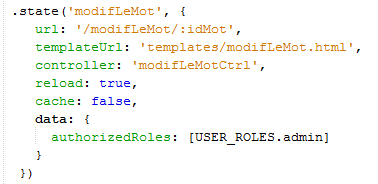
La route.

url de la route.

La seconde route wiki.accueil, route enfant de wiki, la vue contiendra donc le menu précédemment cité.

Le template à afficher sera accueil.html et le contrôleur : accueilCtrl.

Autre cas :



Ici, on se trouve dans la partie sécurisée par login. Ne sont autorisés à visionner cette page que ceux qui ont été identifiés et qui ont un rôle d’administrateur :

data: {

authorizedRoles: [USER\_ROLES.admin]

}

L’url permet de passer un paramètre au contrôleur : url: '/modifLeMot/:idMot', ici idMot est le paramètre passé.

reload: true : La page doit être rechargée.

cache: false : le cache ne doit pas être gardé (scope vidé).

A la fin de notre table de routage nous rencontrons la ligne :

$urlRouterProvider.otherwise('/wiki/page\_accueil');

Cette ligne nous permet de définir le chemin par défaut de l'application qui est la page accueil.

# Les services.

Les services sont des singletons injectés dans notre application grâce à l’injection de dépendance.

 Il existe des services de base et sont tous préfixés d’un $ mais on peut également en rajouter.

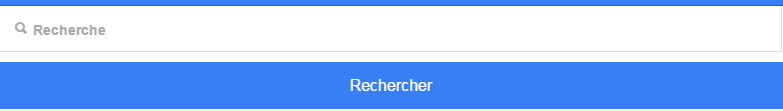
Nous développerons ceux-ci lors de leur utilisation.

1. La page d’accueil.

## La route



* La partie de recherche de mot dans l’encyclopédie.





Ici, on va dire à notre champ input qu’il est relié à la variable *recherche.motCle* de notre scope (grâce à la directive ng-model).



AngularJs : Récupérer ng-model dans le controller accueilCtrl.



: correspond à une directive dans app.js qui permet de gérer l’appui sur la touche entrée. Lors de la validation la fonction rechercher() est exécutée, tout comme le click sur le bouton « Rechercher » (directive ng-click du bouton).

Le code associé dans le contrôleur :



$scope.rechercher = function(){ … } : définit la fonction exécutée lors du click sur le bouton « rechercher ». S’il n’y a pas de mot clé saisi on force la valeur à rechercher à la chaine « AB ».

*motProvider* est un service que j’ai créée pour éviter de surcharger le contrôleur.

La fonction *getlesMots* comprend un paramètre qui est le *motCle* à rechercher dans la base de données.

Des tests ont été effectués car certains mots composés ont été écrits avec des tirets, d’autres avec des espaces (exemple arc-en-ciel).

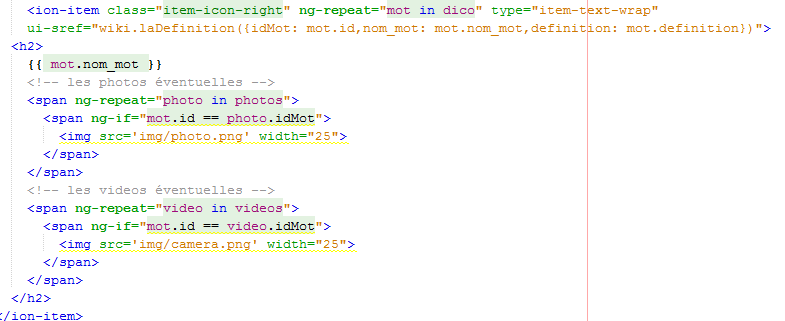
Ainsi la recherche se fera sur les deux possibilités.

$scope.dico = motProvider.getLesMots($scope.recherche.motCle ) : le résultat de la requête sera mis dans la variable $scope.dico.

Lors de la recherche du mot la requête recherche les mots de l’encyclopédie débutant par les lettres saisies.

Le résultat sera affiché sous forme de liste par ordre alphabétique. Si un mot contient des photos ou une vidéo une icône sera affichée à côté du mot. Le click sur le mot affichera une nouvelle page contenant la définition du mot, ses photos, sa vidéo etc.





<ion-item class="item-icon-right" ng-repeat="mot in dico" type="item-text-wrap"

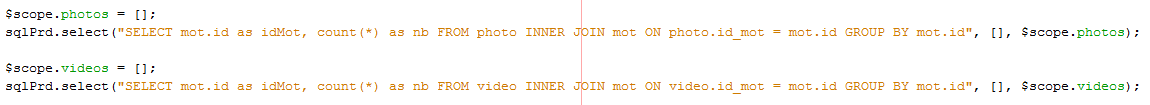
ui-sref="wiki.laDefinition({idMot:mot.id, nom\_mot: mot.nom\_mot,

definition: mot.definition })">

La directive *ng-repeat* permet de boucler sur la variable « dico » qui est le résultat de la requête (voir plus haut), on boucle sur tous les résultats de la requête donc chacun des mots trouvés.

La directive *ui-sref* : cette directive fait le lien avec les routes définies dans app.js, ici la route sera « wiki.definition » avec des paramètres. Ces paramètres seront accessibles dans le contrôleur grâce au service $stateParams.

Si le mot contient des photos et/ou une vidéo une icône sera affichée à côté du mot. Dans le contrôleur on récupère tous les identifiants de tous les mots contenant des photos, idem pour les vidéos.

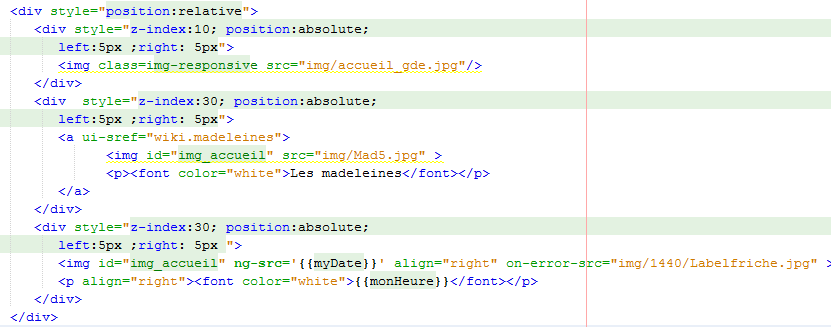


Dans la vue, on boucle sur tous les identifiants des mots afin de vérifier si celui du mot affiché est présent dans la liste préalablement trouvée. Si c’est le cas l’icône est affichée.

En résumé : pour chaque mot récupéré par la requête celui-ci sera affiché et un lien permettra d’afficher ses données dans une autre vue repérée par la route « wiki.definition ».

* La partie centrale.





Il y a 3 éléments dans cette partie :

* 1. L’image de fond appelée accueil.jpg.
  2. L’image miniature de la madeleine avec un lien vers la route « wiki.madeleines » qui est la page affichant les madeleines.
  3. L’affichage de l’heure de connexion avec l’image associée, si elle existe. Une image de substitution est affichée grâce à la directive « on-error-src » qu’il a fallu ajouter dans le fichier app.js.
  4. La page d’affichage du mot.

## La route



Cette page est assez complexe car elle va afficher beaucoup de choses différentes.

# 4.1 La partie haute de la page.

Lien vers le mot suivant dans l’ordre alphabétique avec affichage de ce mot.

Lien vers le mot précédent dans l’ordre alphabétique avec affichage de ce mot.



Affichage de la/ou les photo(s) et de la vidéo associés.

Affichage du mot et de sa définition

## La page html



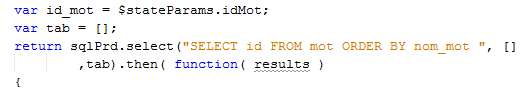
## Le contrôleur.





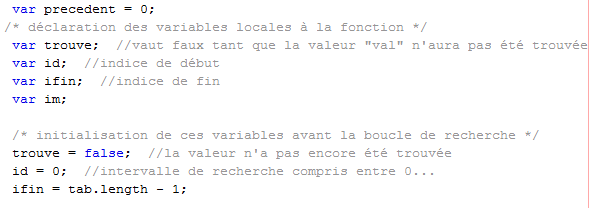
Le nom du mot ainsi que sa définition ont été passés en paramètre. Les caractères ‘\r\n’ représentent un « retour chariot », ces caractères posant problème lors de l’exécution des requêtes ils ont été remplacés par « <br> » dans la base de données. Lors de l’affichage il faut rétablir l’affichage en y réintégrant ce « retour chariot ». D’où la boucle de remplacement.

## La recherche des mots « précédent » et « suivant ».



On met dans un tableau, de nom « tab », les identifiants de tous les mots triés par ordre alphabétique.

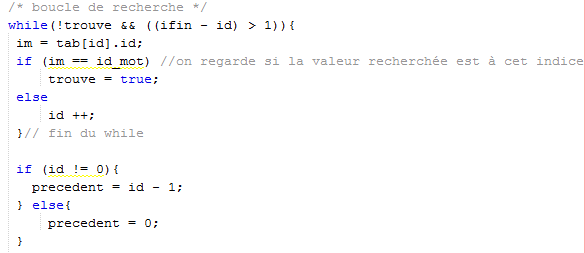
Le principe sera de rechercher l’identifiant du mot courant dans ce tableau afin de trouver l’identifiant du mot précédent ainsi que l’identifiant du mot suivant.



*id* : permet de boucler dans le tableau « tab ».

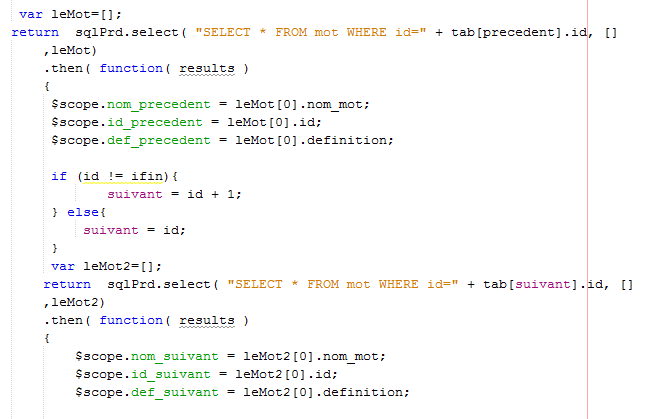
*ifin* : dernier indice du tableau « tab ».

*precedent* : indice du mot précédent dans le tableau « tab ».



Si l’identifiant courant n’est pas le premier par ordre alphabétique (id != 0), le précédent se trouvera à l’indice précédent dans le tableau sinon on reste sur le premier !

Tant qu’on a pas trouvé l’indice du mot dans le tableau et que nous ne sommes pas à la fin du tableau, on continue à parcourir le tableau.



Si l’indice du mot courant est le dernier, le suivant reste sur le dernier sinon on passe à id + 1.

Même principe de requête pour les renseignements sur le mot suivant.

On recherche le mot dans la table (requête sql) et on met le résultat dans le scope.

Contournement du mode asynchrone, on désire attendre que la première requête soit exécutée avant de passer à la suivante, d’où les requêtes imbriquées.

return sqlPrd.select( "SELECT \* FROM mot WHERE id=" + tab[precedent].id, []

,leMot)

.then( function( results )

{ …………………….

return sqlPrd.select( "SELECT \* FROM mot WHERE id=" + tab[suivant].id,

[] ,leMot2)

.then( function( results )

{

………………….

});

});

## L’affichage des photos.



Dans le service crée, j’ai ajouté une fonction getPhotos avec l’identifiant du mot en paramètre. La requête va rechercher les photos correspondant à l’identifiant de la photo. D’où la boucle « ng-repeat » dans la vue.

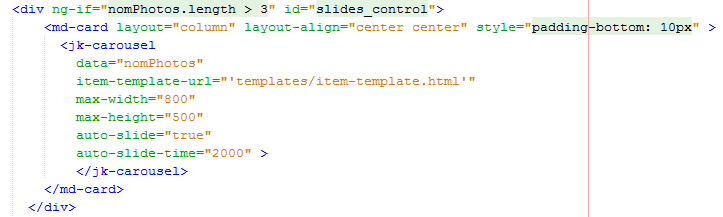
Le principe sera le même pour la vidéo.

L’affichage des photos :

Si le nombre de photos est inférieur à 4 :



Pour l’affichage des photos celui-ci sera différents si le nombre de photo est supérieur à 3.



Afin de pouvoir utiliser le carrousel il a fallu que j’intègre les liens vers les fichiers du service et de la directive dans index.html.



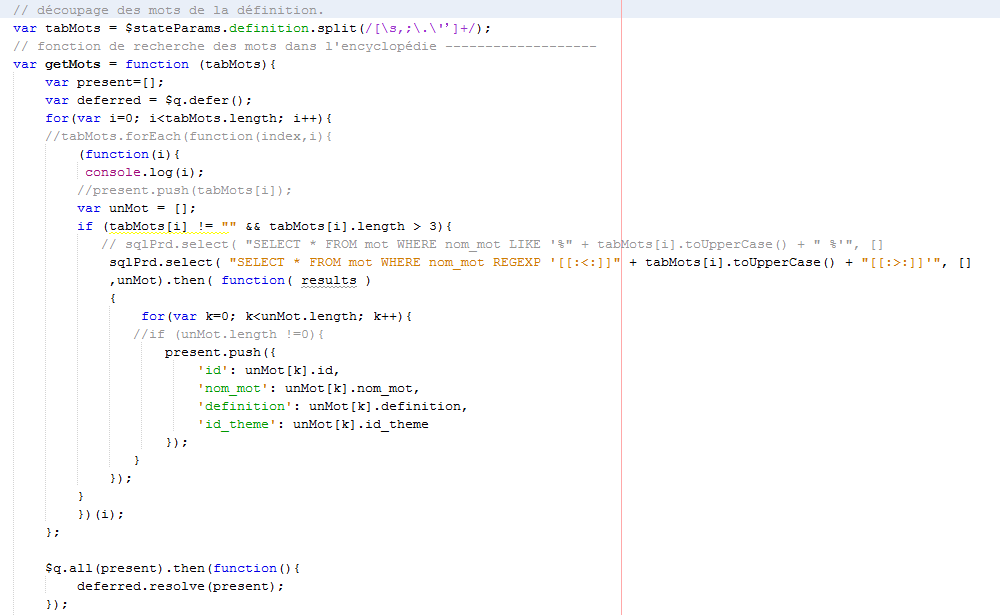
Il fallait également modifier le fichier « item-template.html » du répertoire « carrousel » afin d’y intégrer le chemin exact de l’emplacement des photos à afficher :



# 4.2 Partie basse de la page.



* **Les mots associés** : tous les mots de la définition ont été enregistrés dans un tableau. Chaque élément du tableau dont le nombre de lettres est supérieur à 4 sera recherché dans l’encyclopédie, s’il y a correspondance le mot est affiché avec un lien vers la page de définition de ce mot.



La première ligne permet de découper la définition et mettre le résultat dans un tableau. Les caractères de séparation peuvent être : un espace, deux points ( :), etc.

Pour chaque mot figurant dans le tableau, il faut faire une requête de recherche.

Le problème est que la boucle « for » de parcours du tableau s’effectue plus rapidement que la requête de recherche !!!

Il a alors fallu requérir au système de « promises » proposé par AngularJS, donc utilisation d’une fonction. Chaque mot trouvé sera mis dans un nouveau tableau nommé « present ».

L’expression de recherche REGEXP :

*"SELECT \* FROM mot WHERE nom\_mot REGEXP '[[:<:]]" + tabMots[i].toUpperCase() + "[[:>:]]'"*

On utilise [[:<:]]mot[[:>:]] pour vérifier qu’on matche réellement le mot « mot » et non le mot « motel ».

* **La partie du bas ne figure pas pour tous les mots.**

M Péchoux a créé une annexe avec des précisions sur certains mots de l’encyclopédie. Comme les mots étaient les mêmes que ceux figurant dans l’encyclopédie il a fallu créer une nouvelle table nommée « annexe ».



Toujours le même principe. Le mot est recherché, par requête dans la base de données.

* 1. Les madeleines.

# 5.1 Affichage de toutes les madeleines

La route :



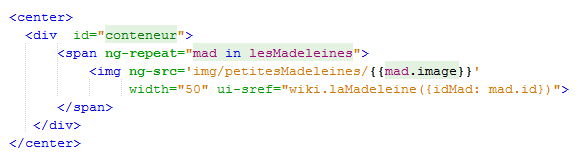
## La page html

Pour le moment les madeleines sont affichées comme sur un plateau.



Elles sont affichées dans un « conteneur » défini dans le fichier style.css.





L’identifiant de la madeleine est passé en paramètre.

Les madeleines sont enregistrées sous deux formats, l’un, les petites madeleines dans le répertoire « img/petitesMadeleines » l’autre, pour les madeleines en grand format dans « img/madeleines ».

## Le contrôleur

Le contrôleur ne présente pas de difficultés il ne fait qu’exécuter le requête de recherche des madeleines dans la base de données.



Le click sur une madeleine nous amène à la page laMadeleine.html :



# 5.2 Affichage d’une madeleine.

La route, avec son paramètre, l’identifiant de la madeleine :



La page html : La madeleine est affichée avec son histoire.





Cette page comporte le lien vers la page des madeleines 

Puis l’affichage du champ « libelle » de la table de base de données, ainsi que de la photo dont le nom se trouve dans le champ « image » de la table.

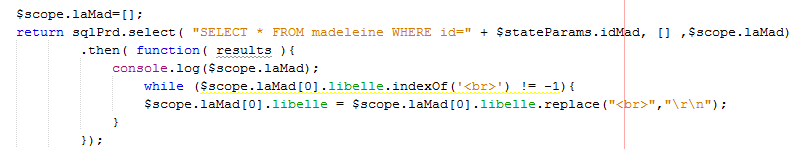
## Le contrôleur :

Le click sur le bouton : « les madeleines » dont le lien est l’exécution de la fonction « lesMadeleines() ». On ne fait que nous renvoyer sur la route wiki.madeleines.



L’affichage de la définition et de la photo :

On exécute la requête de recherche avec le paramètre idMad que nous atteingnons par la commande $stateParams.idMad.



Lorsque la requête est exécutée il faut réinsérer les « retour chariot » dans le texte (voir affichage de la définition du mot, le principe est le même).

* 1. Le mot aléatoire.

Un des choix, après l’affichage du menu burger est « Mot aléatoire » :



Lors du click sur ce choix, la page de définition d’un mot va s’afficher (comme lors de l’affichage d’un mot par le menu de la page d’accueil) mais le mot affiché ici sera choisi de manière aléatoire dans la base de données.

## La route :

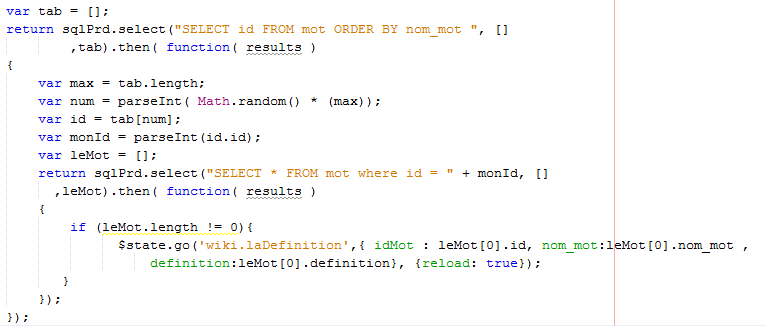


## Le fichier html :



C’est la page de définition du mot ….

## Le contrôleur :



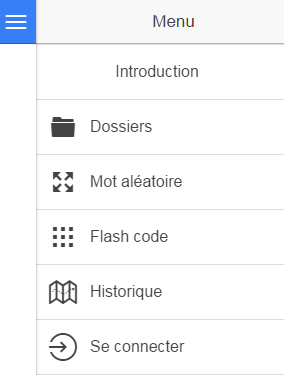
On commence par mettre dans un tableau « tab » tous les identifiants de tous les mots de l’encyclopédie.

Grâce à la fonction Math.random() on va trouver un indice de manière aléatoire.

Il faut effectuer une recherche du mot avec l’identifiant trouvé dans le tableau à l’indice aléatoire et appeler la route wiki.laDefinition avec les paramètres adéquates.

* 1. Gestion de la connexion.

Un des choix, après l’affichage du menu burger est « se connecter » :

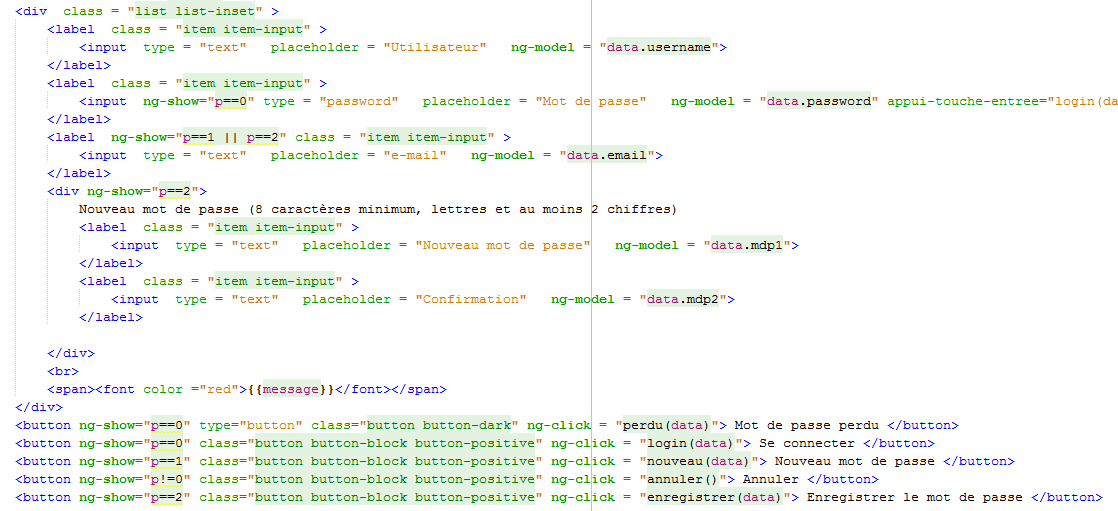


## La route



## Le fichier html

Toutes les valeurs sont enregistrées sous un nom générique data.



A l’ouverture de la page, on demande le nom de l’utilisateur et un mot de passe, deux boutons « mot de passe perdu » et « se connecter » tout ceci paramétré par la directive





Si on clique sur le bouton « mot de passe perdu », le paramètre « p » passe à la valeur 1 et on demande à l’utilisateur son adresse mail à la place de son mot de passe, deux autres boutons vont s’afficher « Nouveau mot de passe » et « annuler ».

Lors du click sur le bouton « nouveau mot de passe » un test de correspondance entre le nom de l’utilisateur est effectué, si’il y a erreur un message apparait sinon on peut saisir un nouveau mot de passe, le paramètre « p » passe à la valeur 2.



S’affichent alors deux zones de texte supplémentaires pour la saisie du nouveau ot de passe, deux boutons « annuler » et « enregistrer le mot de passe ».



Des contrôles sont effectués sur le mot de passe, s’il est correct il est enregistré dans la base de données.

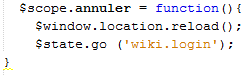
## Le contrôleur

Des variables sont initialisées.



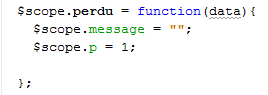
* **Click sur « annuler » :**

On recharge la page, sinon les anciennes données sont gardées !



Retour à la page de login.

* **Click sur « mot de passe perdu »** :



Le message d’erreur est réinitialisé.

La variable p passe à 1 pour la gestion des affichages.

* **Click sur « Se connecter »** :



La variable « data » passée en paramètre a deux valeurs : username et password. Une service AuthService a été ajouté au projet. Celui-ci s’occupe de vérifier la concordance entre le nom de l’utilisateur et son mot de passe.

Dans le fichier service.js la fonction login :



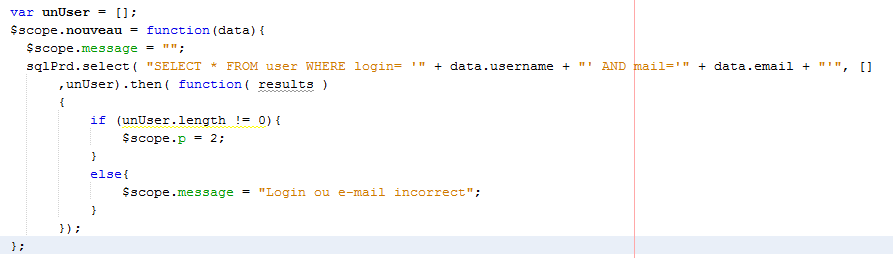
Tout est géré, dans le service, avec des « promises » (voir plus haut, affichage des mots).

On commence par une requête de recherche dans la base de données, sur le nom de l’utilisateur. Il va falloir comparer le mot de passe saisi avec celui, crypté, de la base de données. Le contrôle se fera dans le fichier php « password.php ». Si tout se passe bien on est connecté, c’est-à-dire retour au contrôleur et affichage de la page gestion :



S’il y a erreur un popup s’affiche avec un message d’erreur.

* **Click sur « Nouveau mot de passe »** :



Arrivé ici on a saisi un nom d’utilisateur ainsi qu’une adresse mail. On exécute une requête de recherche du nom d’utilisateur et adresse mail dans la base de données, dans la table « user ».

Si l’utilisateur est trouvé la variable « p » passe à 2, il est autorisé à saisir un nouveau mot de passe. Sinon un message d’erreur est affiché.

* **Click sur « enregistrer le mot de passe »** :



Le 1er mot de passe saisi est testé.

/^[a-z0-9]{8,}$/i.test(str1): entre le début (^) et la fin ($) de la chaine de caractères on vérifie qu’il n’y a que des lettres et des chiffres ([a-z0-9]),qu’il y a au moins 8 caractères ({8,}) que ce soit des majuscules ou des minuscules (i).

str1.replace(/[a-z]/ig,'').length>1 : en remplaçant les lettres par un espace quel que soit la casse (i) et en globalité (g : trouver toutes les correspondances plutôt que de s’arrêter à la première). Si la longueur résultante est >1, donc au moins égale à 2 c’est qu’il y a 2 chiffres.

On teste si les deux chaines sont égales :



Ensuite il faut exécuter le codage du mot de passe saisi, ceci se fait dans le fichier password.php.

Si tout ceci se passe convenablement il faut enregistrer ce nouveau mot de passe dans la base de données.





Après l’appel du fichier php, on récupère dans une variable, ici nommée « d » le retour de la fonction.



Dans la variable « d.nouveau » il y aura donc le mot de passe encodé qu’il faudra intégrer dans la base de données.

Ensuite, on fait une requête de recherche de l’utilisateur avec son mot de passe afin de se positionner sur le bon enregistrement. Si tout se passe bien, le mot de passe est remplacé par celui encodé en php et le tout est enregistré dans la table « user » par une requête de modification (update).

Si l’enregistrement s’est bien déroulé la variable « p » est remise à 0 et un message est affiché confirmant l’enregistrement du nouveau mot de passe.

* 1. La page de gestion.

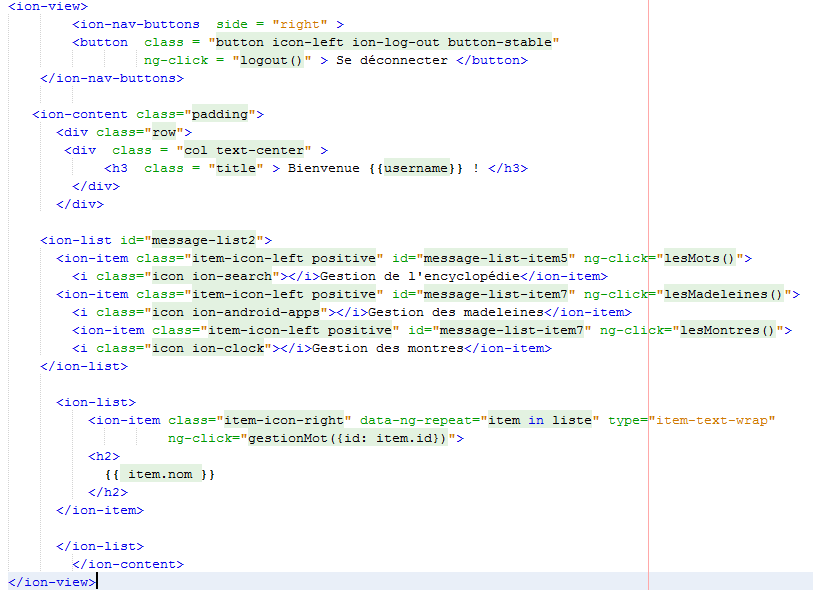
## La route



Seul un utilisateur autorisé peut visionner cette page.

## La page html





A tout moment on peut se déconnecter grâce au bouton en haut de l’écran à droite qui appelle le fonction « logout() ».

Un menu différent sera affiché selon que l’on clique sur « Gestion de l’encyclopédie », « Gestion des madeleines » ou « Gestion des montres ». Tout ceci est géré dans le contrôleur et mis dans une liste.

## Le contrôleur

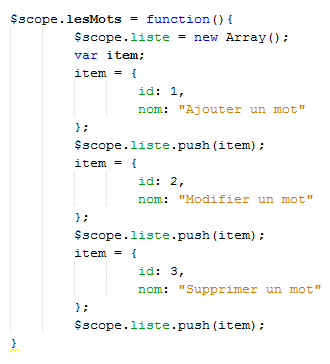
* **Click sur « se déconnecter »** :



On fait appel au service « AuthService » qui s’occupe de tout. Retour à la page d’accueil.

* **Click sur « Gestion de l’encyclopédie » :**

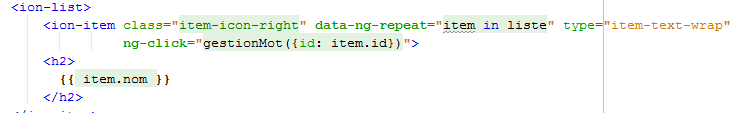
Exécution de la fonction « lesMots() » :



On ajoute des  « item » dans un tableau nommé « liste », chaque « item » sera composé d’un identifiant nommé « id » ainsi que d’un nom « nom ». C’est ceci qui nous permettra d’afficher les bons items et de faire les bons liens.

Dans le fichier html nous avions :

Boucle sur chaque item de la liste.

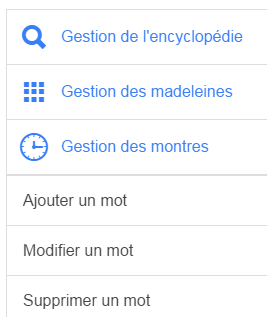


Affichage du nom de l’item.

Lien vers la fonction gestionMot() du contrôleur avec l’id de l’item comme paramètre.

Le principe reste le même pour la gestion des madeleines et celle des montres.

Résultat après le click sur « Gestion de l’encyclopédie » :



* **Click sur un des items de la liste.** Cela nous envoie à chaque fois sur une nouvelle page.



* 1. Insertion d’un nouveau mot.

## La route



Seul un utilisateur autorisé peut visionner cette page.

## La page html



Selon un thème, on saisit un mot. Celui-ci est recherché dans la base de données, s’il est trouvé il s’affiche, sinon il y a possibilité d’en saisir sa définition. Si le thème n’est pas précisé ce sera le thème « encyclopédie » qui sera choisi par défaut.



Affichage des thèmes dans une liste déroulante (select).



Saisie du mot et appel de la fonction « rechercher() » pour voir si le mot existe déjà.



Si le mot n’existe pas (dico.length == 0) on peut saisir sa définition dans la variable « recherche.definition ». Et valider par le bouton qui appellera la fonction « enregistrer() ».



Si le mot, ou un mot avoisinant existe, il sera affiché dans une liste.

## Le contrôleur



Chargement de la liste des thèmes.

* **Click sur « Rechercher ».**



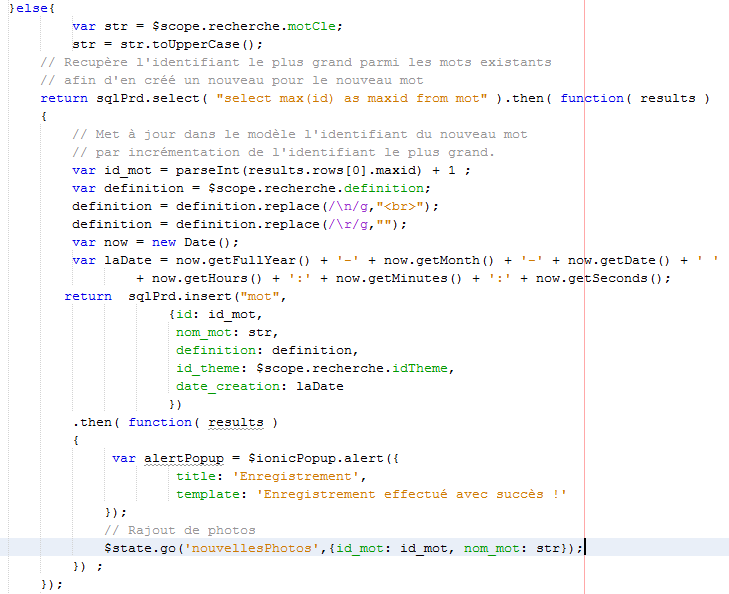
Si le thème n’est pas saisi on en impose un par défaut.

On recherche les mots dans la base de données avec notre service « motProvider ».

* **Click sur « Enregistrer ».**



Si le mot n’est pas saisi ou qu’il n’y a pas de définition il n’y aura pas d’enregistrement !



S’il y a un mot et une définition nous allons pouvoir l’enregistrer dans la base de données.

On commence par rechercher l’identifiant le plus grand dans la table « mot », une fois trouvé, on enlève de la définition les caractères de « retour chariot » pour les remplacer par <br> (l’opération inverse est faite lors de l’affichage du mot).

Il faut récupérer la date du jour et la mettre au format défini dans la base de données.

Puis on insère le mot dans la table par une requête d’insertion (insert), s’il y a problème un message popup est affiché.

Si tout se passe bien on peut passer à l’étape suivant c’est-à-dire aller à la page de saisie des photos avec comme paramètres l’identifiant du mot ainsi que son nom.

* 1. Nouvelles photos.

## La route



## La page html





La zone de sélection de fichiers, « multiple » pour pouvoir sélectionner plusieurs fichiers.

La taille max sera de 2Mo.

Partie récupérée sur Internet.



Le nom du mot est affiché grâce au paramètre de la page précédente.

On peut ici ajouter une vidéo par le click sur le bouton adéquat qui appelle la page d’ajout de vidéo.

## Le contrôleur.



Le click sur le bouton « Nouvelle vidéo » exécute la fonction « ajoutVideos » qui ne fait que nous diriger vers la page d’ajout avec des paramètres.



Le fait de télécharger les photos entraine l’exécution du fichier « upload.php », celui-ci récupère le nom du (des) fichier(s) à télécharger et les place dans le bon répertoire.

* **Click sur « Enregistrer ».**



On boucle sur tous les fichiers se trouvant dans la queue des fichiers téléchargés, pour chacun des fichiers on l’enregistre dans la base de données dans la table « photos ». L’enregistrement est délégué au service motProvider.

Un message popup est affiché lorsque les photos ont été enregistrées. Puis on réaffiche la page de modification du mot l’identifiant du mot en paramètre.

* 1. Nouvelle vidéo.

## La route



## La page html



Les différences avec la saisie de nouvelles photos sont :

* La taille du fichier ne sera pas limitée à 2Mo.
* On ne pourra sélectionner qu’un seul fichier par mot.

## Le contrôleur

Celui-ci sera identique à « nouvelles photos » sauf pour le nombre de fichiers à télécharger.



Si on sélectionne un second fichier celui-ci remplacera celui existant dans la queue de téléchargement.

* 1. Modifier **un** mot.

## La route



## La page html



Le code est le même que dans la page d’accueil pour afficher les mots sauf qu’ici la requête de recherche intègre le thème du mot.

## Le contrôleur



La différence avec la page d’accueil est le lien sur le click sur le mot. Ici nous allons vers la route « modifierLeMot » avec l’identifiant du mot en paramètre.

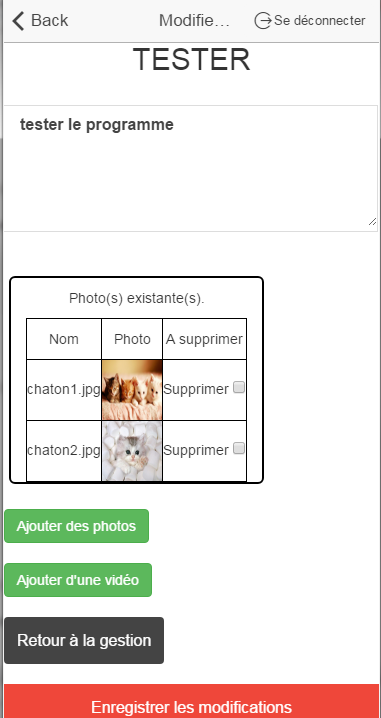
* 1. Modifier **le** mot.

## La route



## La page html

L’écran sera le même que pour « nouveau mot », on commence par rechercher le mot que l’on désire modifier puis cliquer sur ce mot afin d’avoir les données à modifier.



Il est possible de modifier la définition du mot.

Il est possible de supprimer les photos déjà existantes en cochant la case « supprimer » et enregistrer les modifications.

Il est possible d’ajouter de nouvelles photos, une nouvelle vidéo s’il n’y en a déjà une.

S’il y a déjà une vidéo correspondant au mot le bouton ne s’affichera que si on supprime la vidéo existante.

## Le contrôleur



La différence de code avec l’ajout d’un nouveau mot se situe dans l’enregistrement des données dans la table. Ici c’est une requête de mise à jour « update » qui est exécutée.

* 1. Supprimer un mot.

## La route

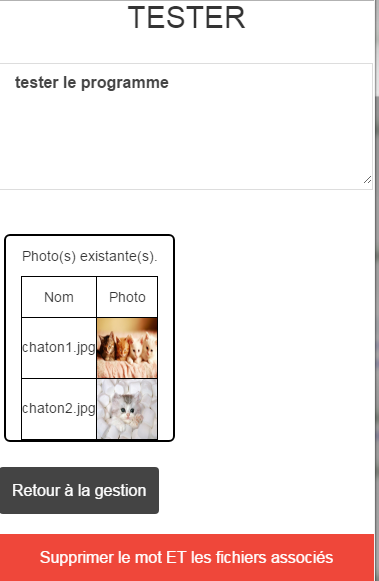




## La page html

La première page affichée sera la même que lors de la modification d’un mot. La seule différence sera le lien lors du click sur le mot. Ici on ira vers la route « supprLeMot » avec l’identifiant du mot en paramètre.

Cette seconde page nous permettra de valider cette suppression.



La suppression entrainera, bien entendu, la suppression en cascade des photos et vidéo des tables « photos » et « vidéos ».

Pour pouvoir effectuer cette suppression en cascade il faut s’assurer que les tables soient de type InnoDB.

## Le contrôleur

Le contrôleur de la seconde page.



Il suffit de supprimer le mot, les photos et la vidéo associés sont supprimés en cascade.

* 1. Gestion des madeleines.

La gestion des madeleines est quasi identique à celle des mots.

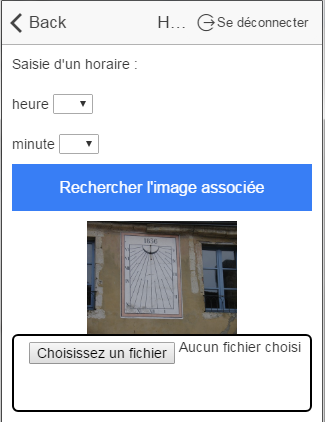
Le fichier uploadMad.php diffère dans l’enregistrement des madeleines car il faut les enregistrer deux fois, une fois en format normal et une fois en miniature.

* 1. Gestion des montres.

## La route



## La page html

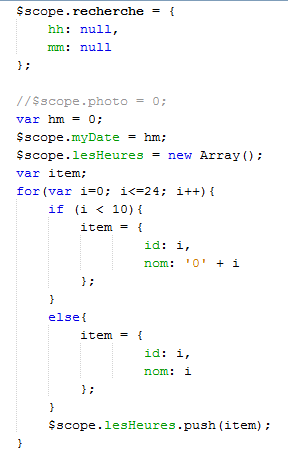


Les horaires sont saisis à partir de liste déroulantes. On peut rechercher si une image est déjà associée à l’horaire. Si ce n’est pas le cas l’image est le cadran solaire sans aiguille !

Il est alors possible de choisir le fichier correspondant à l’horaire.



## Le contrôleur



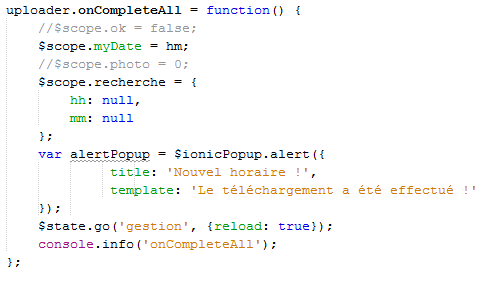
Remplissage du tableau qui va permettre de remplir la liste déroulante des heures.

Les noms des fichiers devant être au format - -h- -, deux caractères pour l’heure et deux pour les minutes avec le caractère h entre les deux.

Il faudra alors ajouter le caractère ‘0’ pour les nombres à 1 chiffre.

Vider les champs de recherche des heure et minute.

Le travail est identique pour les minutes.



Lorsque le téléchargement est terminé, un message popup est affiché et la page de gestion est rechargée.

* **Click sur « Rechercher ».**



On récupère l’heure et les minutes, la variable « hh » va contenir le chemin vers l’image à afficher. La variable « $scope.myDate » va faire le lien avec la page html.



Si le fichier n’existe pas une image par défaut sera affichée grâce à la directive on-error-src, voir page d’accueil.

Annexe : MCD de la base de données.

