**WOOZEOSTAGE**

**Christophe BOISSIEUX – Module Chat**

**Gaëtan PERIS – Module OVS**

**Christophe BOISSIEUX & Gaëtan PERIS - Module User**

**(Stage du 18 septembre au 15 octobre 2014)**

Table des matières

[**WoozeoStage Chat** 3](#_Toc401066255)

[Les principaux fichiers : 3](#_Toc401066256)

[Les Entités : 4](#_Toc401066257)

[Mise en place du socket de communication 5](#_Toc401066258)

[**WoozeoStage OVS** 8](#_Toc401066259)

[Les Bundles utilisés : 8](#_Toc401066260)

[Les fichiers JS : 8](#_Toc401066261)

[Les contrôleurs : 9](#_Toc401066262)

[Les Entités : 10](#_Toc401066263)

[Les Formulaires : 10](#_Toc401066264)

[Les Twig : 10](#_Toc401066265)

[**WoozeoStage User** 11](#_Toc401066266)

[Les contrôleurs : 11](#_Toc401066267)

[Les Twig : 11](#_Toc401066268)

WoozeoStage Chat

Le système de chat a été intégré dans Symfony à l’aide de JQuery UI Chatbox ([Source](https://github.com/dexterpu/jquery.ui.chatbox)).

Un certain nombre de modifications ont été apporté afin d’obtenir le rendu souhaité.

Notamment dans le fichier **jquery.ui.chatbox.js**.

Les principaux fichiers :

Le fichier **chat.js** lui a été créé pour gérer tout le fonctionnement de WoozeoStage Chat.

* Dans un premier temps l’utilisateur doit être connecté.
* Lorsque l’on clic sur un utilisateur une chat-box s’ouvre à condition qu’elle n’existe pas déjà. Le nombre de box à l’écran est limité à 5.
* Si on saisit un message lorsque l’on appuie sur la touche « entrée » le message est affiché dans la box et aussitôt enregistré en Base de données. Il est ensuite envoyé à son destinataire via le socket de communication (traité un peu plus loin).
* Si le destinataire du message n’est pas connecté, il le recevra dès sa connexion.
* Lorsqu’une boite est fermée, toutes les boites qui sont situées à sa gauche subissent un déplacement vers la droite.
* On peut également réduire une fenêtre en cliquant sur le bouton de réduction ou sur la barre de titre
* Lorsque l’on reçoit un message il est immédiatement affiché dans la box correspondante.  
  Si la box n’est pas ouverte, elle s’ouvre automatiquement.
* Si on change de page, soit en réactualisant soit en cliquant sur un lien, toutes les informations liées aux box sont enregistrées provisoirement de manières à faciliter le réaffichage des box.
* Lorsque l’on clic sur son profil, on a accès à la l’historique des Chats. Dans un premier temps la liste des personnes avec lesquels on a chatté. Ensuite en cliquant sur une personne on obtient le détail des chats par date.

Le fichier **ChatController.php** est notre controller dans ChatBundle. Il permet de gérer :

* La liste des utilisateurs
* Les ajouts en Base de données via une requête AJAX.
* Les envois de message via le serveur
* Les mises en session des id des box toujours via une requête AJAX
* La récupération des informations nécessaires au réaffichage des box après un changement de page
* La récupération de l’historique des chats.

Les Entités :

L’entité **Messagebox** définit dans le fichier **Messagebox.php** permet de définir les champs et les relations avec l’entité User de FOSUserBundle qui seront par la suite créé dans la table messagebox de la base de données. Les informations pour chaque message étant enregistrées avec l’émetteur du message, le récepteur (destinataire du message), le message et la date de création.

Mise en place du socket de communication

*(node JS étant installé, sinon* [*Installation*](http://fr.openclassrooms.com/informatique/cours/des-applications-ultra-rapides-avec-node-js/installer-node-js)*)*

* On installe **faye** via la console

$ npm install faye

* On crée un fichier **server.js** qui va permettre de lancer le serveur node via la commande console

node web/js/server.js

var http = require('http'),

faye = require('faye');

var server = http.createServer(),

bayeux = new faye.NodeAdapter({mount: '/', timeout: 45});

bayeux.attach(server);

server.listen(3000);

* Coté Symfony2, nous avons une application tout ce qu’il y a de plus standard, qui va « pusher » des messages à Faye par l’intermédiaire de la libraire **nc/faye-client**.

On installe via composer

{

"require": {

"nc/faye-client": "v1.1.0"

}

}

* On crée un fichier **services.yml** pour la définition d’un service client

Dans WS/ChatBundle/Ressources/config

parameters:

WS\_ChatBundle.faye.endpoint: 'http://localhost:3000/'

WS\_ChatBundle.faye.adapter.class: 'Nc\FayeClient\Adapter\CurlAdapter'

WS\_ChatBundle.faye.client.class: 'Nc\FayeClient\Client'

services:

WS\_ChatBundle.faye.adapter:

class: %WS\_ChatBundle.faye.adapter.class%

WS\_ChatBundle.faye.client:

class: %WS\_ChatBundle.faye.client.class%

arguments:

- "@WS\_ChatBundle.faye.adapter"

- "%WS\_ChatBundle.faye.endpoint%"

* Dans le fichier **config.yml** dans app/config

On ajoute une ligne à la rubrique imports :

imports:

- { resource: "@WSChatBundle/Resources/config/services.yml" }

* Utilisation du service pour l’envoi d’un message (exemple depuis un controller)

<?php

// récupération du client

$faye = $this->container->get('acme\_demo.faye.client');

// construction d'un message

$channel = '/messages';

$data = array('text' => 'Lorem ipsum dolor sin amet...');

// envoi du message

$faye->send($channel, $data);

* Appels fichier JS dans **base.html.twig**

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js"></script>

<script type="text/javascript" src="{{ asset('js/client\_angularjs.js') }}"></script>

<script type="text/javascript" src="http://localhost:3000/client.js"></script>

<script src="{{ asset('js/client-app.js') }}"></script>

* Initialisation du client et souscription à un channel

Dans un fichier javascript **client-app.js**

var client = new Faye.Client('http://localhost:3000/');

client.subscribe('/messages', function (message) {

alert('Nouveau message : ' + message.text);

});

Dans woozeochat, ceci a été rajouté dans le fichier chat.js

* Dans une application AngularJS (fichier **client-angularjs.js**)

(voir mise en place <http://afsy.fr/avent/2013/16-symfony-et-angularjs-tips>)

(function() {

'use strict';

angular

.module('demoApp', [])

// initialisation du client et encapsulation dans un service

.service('FayeClient', function() {

return new Faye.Client('http://localhost:3000/');

})

// souscription au channel "/messages"

.run(function($rootScope, FayeClient) {

FayeClient.subscribe('/messages', function(message) {

$rootScope.$broadcast('notification', message);

});

})

// utilisation depuis controller

.controller('DemoCtrl', function($scope) {

$scope.$on('notification', function(event, message) {

alert('Nouveau message : ' + message.text);

});

})

;

})();

Puis Dans un div du template on ajoute **ng-app** qui prend pour valeur le nom du module ici **demoApp**  
<div id="chat\_div" ng-app="demoApp">.

WoozeoStage OVS

Les Bundles utilisés :

* **FriendsOfSymfony User Bundle** : ([installation](https://github.com/FriendsOfSymfony/FOSUserBundle))  
  bundle servant à gérer les utilisateurs.
* **JMS security-extra bundle** et **JMS di-extra Bundle** : ([installation](http://jmsyst.com/bundles/JMSSecurityExtraBundle))  
  Bundle servant à gérer annotation @Secure dans les contrôleurs.
* **FriendsOfSymfony JSRouting Bundle :** [(installation)](https://github.com/FriendsOfSymfony/FOSJsRoutingBundle)Permettre d’utiliser une route Twig dans un fichier Javascript. Il faut rajouter l’annotation options= {‘expose’=true} dans la route.
* **Stfalcon tinymce bundle :** ([installation](https://github.com/stfalcon/TinymceBundle))  
  Intègre un wysiwyg dans les textarea.  
  dans Twig rajouter {{ tinymce\_init() }}  
  Les options du menu sont personnalisables dans ***app/config/config.yml***
* **Egeloen google-map bundle** : ([installation](https://github.com/egeloen/IvoryGoogleMapBundle))  
  Permet l’affichage d’une google map.  
  dans Twig rajouter {{ google\_map(map) }}

Les fichiers JS :

Fichier **calendrier.js**

Associé à la balise <div id=”datepicker”> permet d’afficher le calendrier.

Fichier **adresspicker.js**

Permet de lier une adresse avec la Google-map.

Fichier **adresspickermodif.js**

Permet de lier une adresse avec la Google-map, la carte est centrée sur la longitude et la latitude avec l’adresse entrée en base.

Fichier **adresspickervoir.js**

Permet de centrer la carte avec l’adresse entrée en base ou la latitude et la longitude.

Fichier **gerer.js**

Permet de gérer dynamiquement le nombre d’inscrits à un événement quand tu gères celui-ci.

Fichier **Jquery.ui.addresspicker.js**

Fichier fournit qui va contenir le code de l’API d’adresspicker.

Fichier **listEvenement.js**

Sert à faire une requête Ajax pour trier les événements suivant un critère préalablement sélectionné.

Les contrôleurs :

**AccueilController.php**

Contient la méthode **indexAction** qui permet à l’ouverture du site web de montrer la liste des événements en fonction de la date du jour.

Si l’utilisateur n’est pas connecté, seuls les événements publics sont visibles.

**AdminController.php**

Contient la méthode **listEvenementAction()** qui affiche la liste des événements et qui permet en cas de requête Ajax (voir fichier **listEvenement.js** ) de ne recharger qu’une partie de la page.

**CommentaireController.php**

Permet de gérer les commentaires (CRUD).

Il n’y a pas de méthode supprimer à la place on désactive simplement le commentaire.

**EvenementController.php**

Permet de gérer les commentaires (CRUD).

Il n’y a pas de méthode supprimer à la place on désactive simplement l’événement.

La méthode **gererAction()** permet de gérer la liste des personnes inscrites à l’événement.

**RechercheController.php**

Contient la méthode **rechercheAction()** qui permet de rechercher un événement, suivant 2 critères la ville et/ou le sport.

**SportController.php**

Contient les méthodes d’ajout de sport et liste de sports.

**UserEvenementController.php**

Permet de gérer la participation à un événement (inscription et annulation).

Il contient également la méthode **modifierEvenementAction()** qui permet de mettre tous les participant « *en attente* » sauf le créateur de l’événement, lorsque celui-ci modifie le dit événement.

Les Entités :

**EvenementRepository**

Contient la liste des méthodes pour les requêtes personnalisées comme le tri des événements en fonction du sport ou les méthodes de recherches d’événements en fonction d’une ville et/ou d’un sport. Ainsi que les événements privés de l’utilisateur et de ses amis.

**UserEvenementRepository**

Contient la méthode qui contient la liste des participants triés par noms.

Les Formulaires :

**EvenementGererType**

Il contient une collection de **UserEvenementGererType** afin d’afficher la liste des participants à un événement et gérer celle-ci.

Les Twig :

Evenement**/gerer.html.twig**

Ici on affiche le formulaire pour gérer les participants à un événement. On a séparé le formulaire pour pouvoir traiter le cas des ***userEvenements***. On affiche la liste des participants actif. Dans le cas du créateur de l’événement on y rajoute une ***<div id= « créateur »>***  afin de pouvoir l’empêcher de modifier son statut (voir gerer.js).

Les participants qui ont annulé leur participation et le reste du formulaire est intégré dans une balise Div elle-même cachée par CSS.

WoozeoStage User

Les contrôleurs :

**AmiController.php**

Il contient les méthodes pour envoyer une demande d’ami : **addAction()**.

La méthode pour désactiver une relation entre 2 amis : **desactiverAction()** dans le cas où un utilisateur aurait envie de supprimer un de ses amis.

La méthode pour refuser ou accepter un ami : **gererAction()**.

La méthode **annonceAction()** qui permet de notifier un utilisateur si celui-ci a de nouvelles demandes d’amis ou des demandes acceptées.

**ProfileController.php**

Il s’agit d’une surcharge du **ProfileController** de FOS dans lequel on a ajouté la liste des amis, des demandes d’amis en cours, des événements et des participations.

**UserController.php**

Il contient la méthode **profilAction()** qui permet de visualiser le profil d’un utilisateur (à ne pas confondre avec mon profil). Ce profil contient la liste des événements créés par l’utilisateur et ses participations.

Il faut être connecté et ami avec l’utilisateur pour voir ses événements créés privés. Concernant sa participation à des événements privés, il faut que soit celui ai été créé par l’utilisateur, soit un de ses amis communs.

Les Twig :

Profile/**show.html.twig**

Il s’agit de la surcharge de la vue de FOS dans laquelle on peut voir la liste de ses amis, de ses amis en attente, de ses événements créés et de ses participations.