La Fringale de Trad'

Dossier de projet

Simon SAUZEDE
simon.sauzede@gmail.com

Dossier de projet

Site vitrine - titrepro.tenbu-o.fr

Sommaire

Introduction

Le site vitrine présenté dans ce dossier a été développé sur une période de 2 mois, du 13 mars 2019 au 10 mai 2019. Il a été réalisé pour l'association La Fringale de Trad', oeuvrant dans l'organisation de bals de musiques traditionnelles, sous la supervision de l'école O'Clock, dispensant des formations en développement web.

J'ai choisi mon stage selon plusieurs critères.

D'abord, très intéressé par l'exercice de la profession sous le statut de freelance, j'ai souhaité travailler pour un organisme qui n'avait aucun rapport avec le développement, afin de pouvoir me projeter dans une relation de client à professionnel dans laquelle le client serait totalement profane en la matière, de sorte que m'incombe entièrement la gestion du projet et l'interprétation du besoin client.

Ensuite, sortant récemment de formation, le projet de stage se devait d'être relativement léger en fonctionnalités, afin de me laisser suffisamment de temps pour pouvoir me former en profondeur sur les nouvelles technologies que j'aborderais, de m'intéresser aux bonnes pratiques lors de l'utilisation de celles-ci, et d'aborder les choses sous un angle plus formateur, mais, par ailleurs, plus chronophage. Je voulais aussi, en parallèle, amorcer le développement d'une librairie personnelle, avec tout ce que cela implique sur la réutilisabilité et le découplage de ses divers composants.

Plus particulièrement, j'ai souhaité me former sur les outils de développement front-end, et développer un site sous la forme d'une Single Page Application. Ainsi, un site vitrine était particulièrement adapté à mes besoins.

En somme, l'idée était de me laisser suffisamment de temps pour apprendre, me documenter, développer "from scratch" les domaines sur lesquels je n'avais pas une bonne maîtrise, et extrapoler sur des problématiques qui ne se présenteraient pas nécessairement durant la période de stage.

Un "petit" site permettait de laisser la part belle à tout ce processus d'apprentissage.

Description du projet

Compétences concernées

Activité type n°1: Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité :

- Maquetter une application
- Réaliser une interface utilisateur web statique et adaptable
- Développer une interface utilisateur web dynamique

Activité type n°2: Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité :

- Créer une base de données
- Développer les composants d'accès aux données
- Développer la partie back-end d'une application web ou web mobile

Présentation du projet

Le projet a consisté en la réalisation d'un site pour La Fringale de Trad', association agissant dans l'animation de bals de musiques traditionnelles.
Le site avait pour objectif de mettre en avant de l'ensemble de leurs activités, et de permettre aux individus intéressés de suivre les performances du groupe de musique que constitue les membres de l'association. Ainsi, le site devait proposer des composants permettant la lecture de musiques, le visionnage de photos, la communication des événements et l'échange avec les visiteurs du site.

Également, l'ensemble du contenu devait être modulable, afin d'anticiper les changements qui pourraient survenir ultérieurement au sein de l'association. Ainsi, une interface d'administration était nécessaire. Les membres de l'association ont souhaité l'enregistrement d'un administrateur unique, c'est-à-dire un unique couple d'identifiant/mot de passe, pour accéder à cette interface. Les membres de l'association n'avaient aucune requête quant aux technologies à employer pour le développement du site, mais souhaitaient en revanche que celui-ci soit hébergé gratuitement. Les charges de prospecter les hébergements et les registres de noms de domaines incombaient au stagiaire.

Le développement du site a été effectué en télétravail, et les échanges avec l'association ont eu lieu par mail et téléphone. Après chaque échange s'en suivait, sous une semaine, une concertation entre les membres de l'association suivi d'une réponse validant ou invalidant les propositions et ajouts sur le site.

Cahier des charges

Objectif

La réalisation du site a pour objectif de permettre la visibilité du groupe de musique La Fringale de trad' sur le net. Celui-ci devra présenter les productions du groupe, les activités passées et à venir, ainsi qu'offrir un moyen de contact aux visiteurs du site. Le groupe devra également avoir la possibilité de mettre à jour les informations du site.

En amont de la phase de développement, le stagiaire devra présenter des maquettes, afin d'établir la mise en page du site en accord avec les membres du groupe. En aval de la phase de développement, il devra assister le groupe dans le choix d'un hébergement, l'achat d'un nom de domaine et la prise en main de l'interface d'administration du site.

Spécificités fonctionnelles

Front:

- Proposer à l'écoute les productions du groupe par le biais d'un lecteur audio.
- Afficher les photos et les affiches de concert par le biais d'un slider.
- Offrir un moyen de veille des évènements en proposant une inscription à la newsletter.

Back:

- Mettre à jour les informations du site et uploader des photos et des fichiers audios, grâce à une interface d'administration.
- Afficher les nombres de visiteurs et de visites sur le site, comptabilisés par mois.
- Tenir informé des prochains événements par le biais d'une newsletter.
- Recevoir les mails saisis par les visiteurs sur le formulaire de contact.

Spécificités techniques

Le framework *Vue.js* a été utilisé afin de permettre la compartimentation des éléments front-end sous forme de *Single File Component,* compilé avec *Webpack*, et d'offrir un affichage réactif. *Vue-resources* a été intégré afin de gérer les appels *AJAX*, et *Vue-x* pour gérer la communication inter-composants via un système de store.

La feuille de style *normalize.css* a été utilisée afin d'écraser les configurations *CSS* divergentes selon les navigateurs. Le préprocesseur *SASS* a été utilisé afin de faciliter la génération et la maintenance des feuilles de style. Les icônes sont gérées avec *Fontawesome*. Le style a été développé sans l'appui d'un framework *CSS*.

Le backend a été conçu avec *Symfony 4*. Le back-office utilise le bundle *EasyAdmin*, et les newsletters sont envoyées grâce au bundle *SwiftMailer*.

Utilisateurs

Le visiteur anonyme est autorisé à consulter la partie vitrine du site. L'administrateur peut se connecter au back office afin de modifier le contenu du site, et il n'existe qu'un seul et unique administrateur.

Feuille de route

Cette section décrit chacune des étapes de développement du projet. Celles-ci sont décrites succinctement et chronologiquement, dans l'idée d'apporter une vision d'ensemble du processus de développement; préalable nécessaire à la bonne compréhension des réalisations majeures de l'application. Ces dernières seront détaillées et argumentées dans une section à part entière.

Maquettage et relation client

Cette étape a consisté en la conceptualisation du site. Puisque le dictionnaire des données et les wireframes (cf annexes) furent aisément réalisés compte-tenu de la simplicité du besoin client, le travail le plus important fut la conception du design du site. Les membres de l'association n'avaient guère de suggestions concernant l'apparence du site, et seuls les termes 'simple' et 'élégant' sont venus caractériser le design attendu. Une première maquette fut réalisée dans ce sens (cf annexes).

Les clients ont trouvé celle-ci trop stylisée, et des instructions plus précises me sont parvenues concernant la page d'accueil, avec une maquette réalisée par leur soin :

- Le layout de la page d'accueil devait s'articuler autour d'une photo, qui occuperait tout l'espace de la page si possible.
- Les boutons de navigations du site devaient être disposés à plusieurs endroits de la photo.
- S'il devait y avoir une image de fond derrière la photo, elle devait rappeler la paille, et avoir une teinte orange claire.

La page d'accueil désirée l'association posait des soucis en terme d'ergonomie et de lisibilité. Afficher du texte par dessus une image figurative et trop contrastée est une mauvaise pratique nuisant à la lisibilité du texte. De plus, une mise en page organisée à partir d'une photographie aux dimensions fixes (par opposition aux images "seamless") a peu de flexibilité pour s'adapter à son contenu lorsque celui-ci varie. Avec les contraintes ainsi définies, je n'ai pas su créer une mise page propre et ergonomique.

Une réponse que j'ai apportée au problème fût la réalisation de maquettes alternatives, respectant une partie des demandes des clients, et proposant une mise en page plus propre et lisible. J'ai également indiqué, parmi ces maquettes, laquelle s'avérait préférable (cf annexe). Cependant, ces maquettes n'ont pas été retenues et nous sommes resté sur la mise en page telle que demandée (cf annexe).

Outillage front et intégration statique

Cette étape a consisté en l'intégration statique du site, sans base de données, moteur de template ou interaction *javascript*. Les points cruciaux de cet exercice furent la configuration d'une pipeline d'assets avec *Webpack* et l'apprentissage du *SASS*.

Node.js étant déjà présent sur mon environnement de développement, il m'a suffit d'initialiser le projet grâce à *Node Package Manager*, puis d'installer *Webpack* comme dépendance du projet.

Le point d'entrée de la compilation pointe sur le fichier `resources/style.scss`. Ce dernier contient toutes les variables et tous les mixins nécessaires aux feuilles de styles. Il inclut par ailleurs les diverses polices de caractère ainsi que chacune des feuilles de style utilisées pour les divers modules de l'application.

```scss
$yellow: #f3bd0b;
$orange: #f3960b;
$brown: rgb(49, 14, 4);

@font-face{
    font-family: "Comfortaa";
    src: url('../fonts/comfortaa/Comfortaa-Regular.ttf') ;
    font-weight: normal;
}
@font-face{
    font-family: "Comfortaa";
    src: url('../fonts/comfortaa/Comfortaa-Bold.ttf') ;
    font-weight: bold;
}

@mixin font-type($type)
{
    font-family: 'Comfortaa';

    @if $type == 'title'
    {
        font-weight: bold;
        font-size: 1em;
    }

    @if $type == 'navigation'
    {
        font-weight: bold;
        font-size: 1.3em;
    }

    @if $type == 'button'
    {
        font-size: 0.8em;
    }
}

@mixin clickable($shadow_offset, $scale)
{
    @if $shadow_offset == ""
    {
        $shadow_offset: 2px;
    }

    @if $scale == ""
    {
        $scale: 1.25;
    }

    transition: transform 0.4s;
    cursor: pointer;

    &:hover {
        transform: scale($scale);
    }
}

@import 'normalize.scss';
@import 'global.scss';
@import 'Modules/section/s-section.scss';
@import 'Modules/player/s-player.scss';
@import 'Modules/photos/s-photos.scss';
@import 'Modules/presentation/s-presentation.scss';
@import 'Modules/shows/s-shows.scss';
@import 'Modules/contact/s-contact.scss';
@import 'Modules/friends/s-friends.scss';
@import 'Modules/aside/s-aside.scss';
```

Le point de sortie de la compilation pointe sur le dossier `public`, qui contient les ressources finales nécessaires au navigateur.
J'ai ensuite entamé la configuration de la pipeline *SASS*, composée de plusieurs loaders et plugins.

Loaders:

- *sass-loader* : compile les fichiers *SASS*.
- *css-loader* : gère les imports CSS pour les compiler en *commonJs*.
- *MiniCssExtractPlugin.loader* : permet de récupérer le *CSS* dans un fichier à part entière, au lieu d'exploiter le fichier *javascript* généré par *webpack*.

Plugins:

- *MiniCssExtractPlugin* : permet d'extraire le *CSS*, récupéré après la compilation du *css-loader*, dans un fichier à part entière. Il suffit pour cela de spécifier dans les options du plugin le point de sortie et le nom du fichier, en plus d'utiliser le loader fourni avec le plugin.
- *FixStyleOnlyEntriesPlugin* : permet d'empêcher *Webpack* de générer un fichier *javascript* lorsque qu'il démarre une pipeline ne concernant que le *CSS*.

Vous trouverez l'intégralité du fichier de configuration de *Webpack* en annexe.

```
module:
{
  rules:
  [
    {
      test: /\.scss$/,
      use:
      [
        {
            loader: MiniCssExtractPlugin.loader,
        },
        {
            loader: 'css-loader',
            options: {
                url: false,
            }
        },
        {
            loader: 'sass-loader',
        }
      ]
    },
  ]
},
plugins:
[
  new FixStyleOnlyEntriesPlugin(),
  new MiniCssExtractPlugin({
      filename: "css/style.css",
    })
],
```

la fonction `url` - en *commonJs* - via `require`. En effet, les assets qui n'avaient nul besoin d'être compilés, tels que les images ou les polices de caractère, ont été déposés directement dans le dossier `public`, et leur chemin d'importation était construit à partir de ce dossier. Le `require` étant interprété lors de la compilation, *Webpack* tentait d'importer des fichiers absents du dossier resources et plantait la compilation.

Pour finir, la commande *webpack* a été ajoutée à l'entrée `scripts` du `package.json`. Cette commande permet de démarrer la compilation des différentes pipeline.

Grâce à la pipeline ainsi configurée, j'ai pu réaliser l'intégration statique de l'interface visiteur en *HTML* et *CSS*. J'ai veillé à utiliser des balises sémantiques pour structurer mon contenu, et n'ai pas recouru à un framework *CSS*.

Dynamisation de l'interface et recherche de design pattern

Cette étape a consisté en l'installation du framework *Vue.js* afin de compartimenter mon *javascript* sous forme de *Single File Components*. Cette structure m'a permis d'isoler la logique des différents éléments du site, et d'améliorer ainsi leur lisibilité et leur maintenabilité.

Pour ce faire, j'ai configuré une seconde pipeline dont le rôle est de compiler les *Single File Components* à l'aide du loader de *Vue*. Ce loader permet d'interpréter la syntaxe particulière des composants, permettant à la fois de développer le *javascript* dans une balise `<script>` et le template *HTML* dans une balise `<template>`, le tout dans unique fichier à l'extension `.vue`. Un second point d'entrée a été défini sur le fichier `resources/app.js`.

```
import Vue from 'Vue';
import VueResource from 'vue-resource';
import VueX from 'vuex';
Vue.use(VueResource);
Vue.use(VueX);


import appAudio from 'TenbuoModules/audioPlayer/c-audioPlayer.vue';
import appPhotoDisplayer from 'TenbuoModules/photoDisplayer/c-photoDisplayer.vue';
import appSlider from 'TenbuoModules/slider/c-slider.vue';

import hailed from 'TenbuoModules/hailer/d-hailed.js';
import hailer from 'TenbuoModules/hailer/d-hailer.js';
import dial from 'TenbuoModules/dial/d-dial.js';
import display from 'TenbuoModules/photoDisplayer/d-display.js';

import interpolator from 'TenbuoUtils/interpolator.js';

import appNavButton from 'Modules/navButton/c-navButton.vue';
import appSection from 'Modules/section/c-section.vue';
import appPresentation from 'Modules/presentation/c-presentation.vue';
import appShows from 'Modules/shows/c-shows.vue';
import appContact from 'Modules/contact/c-contact.vue';
import appFriends from 'Modules/friends/c-friends.vue';
import appAside from 'Modules/aside/c-aside.vue';

import config from 'Config';


new Vue({
    el: '#app',
    data:
    {
        props: config,
        interpolator: interpolator,
    },
    components:
    {
        'app-nav-button' : appNavButton,
        'app-section' : appSection,
        'app-presentation': appPresentation,
        'app-shows': appShows,
        'app-contact': appContact,
        'app-friends' : appFriends,
        'app-audio' : appAudio,
        'app-photo-displayer' : appPhotoDisplayer,
        'app-aside': appAside,
        'app-slider': appSlider,
    },
    provide: function()
    {
        return {
            interpolator: this.interpolator,
        }
    }
});
```

L'architecture des composants *Vue* est construite de la sorte : une instance racine de *Vue* est liée à un élément de ma page. Cet élément pourra ainsi contenir des composants, qui seront alors des enfants de cette instance. Cette dernière pourra leur fournir des données, appelées props, afin de les configurer. Les composants pourront également intégrer dans leur template d'autres composants, générant ainsi une arborescence d'instances parents-enfants.

Cette arborescence peut devenir très complexe à maintenir lorsqu'elle s'étend sur de trop nombreuses couches. En effet, une modification sur la structure d'un enfant trop bas dans la hiérarchie peut nécessiter la mise à jour de tous les composants entre lui et l'instance racine de *Vue*. Si cette complexité reste toutefois abordable, elle devient plus entravante lorsqu'il s'agit de développer une communication entre siblings ou entre branches distinctes, nécessitant alors de remonter les données jusqu'à la racine, puis de les faire descendre jusqu'au composant concerné.

Afin d'anticiper cette problématique lors de mes futurs projets, j'ai entrepris de me renseigner sur les patrons de conception concernant la communication entre deux composants. Lors de cette recherche, je me suis servi de la documentation officielle de Vue, de stackOverflow et de divers blogs de développeur. L'expression m'ayant apportée les résultats les plus appropriés fut *"component communication design pattern Vue"*.

J'ai découvert un article décrivant les divers patrons de conception existants en matière de communication entre composants, et associant ceux-ci à différents cas d'usages.

https://code.tutsplus.com/tutorials/design-patterns-for-communication-between-vuejs-component--cms-32354

«Choosing the right pattern depends on the project you're involved in or the application you want to build. It depends on the complexity and type of your application. Let's explore the most common scenarios:

- *In simple apps, the props and events will be all you need.*
- *Middle-range apps will require more flexible ways of communication, such as event bus and dependency injection.*
- *For complex, large-scale apps, you will definitely need the power of Vuex as a full-featured state management system.*

And one last thing. You are not required to use any of the explored patterns only because someone else tells you to do so. You are free to choose and use whatever pattern you want, as long as you manage to keep your app working and easy to maintain and scale.»

Traduction :

«Le choix du patron dépend du projet sur lequel vous travaillez ou de l'application que vous souhaitez développer. Cela dépend de la complexité et du type de votre application. Explorons les scénarios les plus communs :

- *Dans les applications simples, les props et les événements seront tous ce dont vous avez besoin.*
- *Les application de moyenne envergure nécessiteront des moyens de communication plus flexibles, tel qu'un bus d'événements et l'injection de dépendance.*
- *Pour les applications complexes et de grande envergure, vous aurez définitivement besoin de la puissance de VueX, en tant que système de gestion d'états extrêmement complet.*

Une dernière chose. Vous n'êtes pas obligés d'utiliser l'un des patrons connus uniquement car quelqu'un vous dit de la faire. Vous êtes libre de choisir et d'utiliser n'importe quels patrons que vous voulez, tant que vous arrivez à garder votre application fonctionnelle et facile à maintenir et à étendre.»

Souhaitant découvrir et maîtriser VueX, j'ai décidé de l'intégrer au projet pour gérer l'interaction entre mes composants.
Malgré cela, il demeure le problème de l'accès au store *VueX* dudit composant. Une solution qui a été envisagée était la transmission du store par injection de dépendance depuis la racine. Cependant, j'ai opté pour le développement de directives Vue. Celles-ci sont utilisées sous la forme d'attributs à insérer dans les balises HTML, et induisent un comportement sur l'élément ou le composant ainsi qualifié. Le comportement, par exemple, peut-être la logique d'interaction avec un store référencé par la directive.
Cette solution a été préférée car elle permet un meilleur découplage, puisque la logique d'interaction avec le store est indépendante du composant et se situe dans la directive.

N'ayant, à ce stade, pas encore de backend ni de base de données, j'ai mis en place des données fictives au format *JSON* et ai utilisé *Vue-resources* pour les appels *AJAX*, avec l'intention de simuler une *API*.

Voici les composants et les directives qui ont été réalisées à ce stade du développement, ainsi qu'une succincte description les concernant:

Le lecteur audio

Ce composant fait un appel à l'*API* lors de son initialisation, récupérant ainsi le chemin des différentes ressources audio. A partir de ces données, il crée une piste audio pour chaque fichier, et les référence dans la liste de lecture. Cette dernière permet de sélectionner une piste audio. Les boutons play et pause permettent d'entamer et d'interrompre la lecture de la piste courante. Une timeline permet d'afficher la progression de la piste audio, et il est possible de bouger le curseur de la timeline afin de changer le moment de la musique.

L'Overflow Container

L'idée derrière ce composant était d'avoir une conteneur avec du contenu débordé. Ce contenu devait être parcouru sans scrollbar, mais de manière automatique, ou avec un interval de parcours fixe, sur intéraction de l'utilisateur. Une propriété réactive a été utilisée pour contrôler la valeur du défilement, de sorte que, en interpolant celle-ci successivement jusqu'à une valeur finale, le défilement du contenu s'effectuait de manière continue. Le défilement du contenu était exécuté grâce à la fonction `scrollTo`, et les interpolations successives grâce la fonction `setInterval`.

L'Image Viewer

Ce composant devait être le slider destiné à afficher les photographies de l'association. Celui-ci récupérait le chemin vers les photos par un appel *AJAX* et affichait celles-ci dans l'*Overflow Container*. Une visionneuse d'image était également intégrée, afin d'afficher les photos en grand format sur un clic de l'utilisateur.

Section

Ce composant se présente sous la forme d'un volet déroulant. Le déroulement du volet est basé sur une propriété réactive `height`, que l'on interpole entre 0 et 100. Cette propriété est utilisée afin de générer un style inline qui fera varier la hauteur de l'élément *HTML*. Ce composant a permis de générer les différentes sections du site.

Aside et Aside-btn

Le composant *Aside* constitue la seconde barre de navigation du site, apparaissant lorsqu'une section masque la page d'accueil. Son comportement consiste effectuer une translation horizontale à partir du bord droit de l'écran afin d'apparaître sur l'interface, ou de déclencher l'animation inverse afin de disparaître. *Aside* utilise des composants *Aside-btn* afin de générer les boutons de navigation. Ceux-ci permettent de lancer une animation au survol de la souris, afin d'améliorer le feedback pour l'utilisateur. Pour *Aside*, comme pour *Aside-btn*, les animations fonctionnent là aussi à partir d'une propriété réactive interpolée successivement.

Hailed et Hailer

La directive *Hailer* permet d'écouter un événement de l'élément qu'elle qualifie, puis d'interpeller un autre composant au moment où l'événement est déclenché. La directive *Hailed* doit être positionnée sur un composant. Elle permet de réagir à l'interpellation d'un *Hailer*, ce qui résulte en l'exécution d'une méthode dudit composant. Ces directives ont été utilisées pour déclencher l'apparition des différentes sections sur intéraction avec un bouton de navigation.

Dial

La directive *Dial* permet au composant qu'elle qualifie de s'inscrire sur un store, dans un canal précisé à l'inscription. Elle spécifie alors deux événements : `talk` et `answer`. Lorsque l'événement `talk` est déclenché, les autres composants inscrits sur le canal déclenchent leur événement `answer`. Cette directive est utilisée afin que l'apparition d'une section induise la disparition des autres sections.

Interpolator

Bon nombre de mes composants utilisent une propriété réactive interpolée successivement afin de générer une animation. Ainsi, j'ai développé et dédié un script à ce type d'interpolation : *Interpolator*. Celui-ci est référencé par la racine de *Vue*, et est transmis par injection de dépendance aux composants de l'application. Ce script m'a permis de normaliser le processus d'interpolation pour plus de rapidité dans le développement de mes composants, et une meilleure maintenabilité de ceux-ci.

Restructuration de l'arborescence

N'étant absolument pas familier des outils de développement front-end, je n'avais pas réfléchi à l'arborescence de mon projet, et, avec l'ajout de nouveaux assets au fur et à mesure du développement, il m'a fallu restructurer celle-ci pour répondre à plusieurs problématiques.

Tout d'abord, je n'avais fait aucune distinction entre les ressources spécifiques au projet et celles qui seraient intégrées à ma librairie en vu d'être réutilisées pour d'autres applications web. Ensuite, il m'a fallu prendre en compte l'intégration prochaine de Symfony au projet, afin de gérer le backend de l'application. Enfin, la configuration de mes composants, via les props, devenait plus envahissante au fil du développement et il me fallait trouver un moyen propre de séparer la configuration de l'algorithmique.

Développer des composants suffisamment génériques nécessite de leur laisser une certaine liberté de configuration. Cela se traduit par le passage de nombreuses props, qui doivent auparavant être connues de l'instance racine de *Vue*. De plus, l'assignation des props s'effectuant dans les templates, le code *HTML* devient surchargé et difficile à comprendre.

Afin de désengorger l'instance racine de *Vue*, j'ai délocalisé la valeur des props dans un fichier `config.js` que j'ai importé dans celle-ci. Pour éviter le passage en masse des props dans les templates, j'ai utilisé la directive `v-bind` permettant de passer toute la configuration du composant sous la forme d'un objet.

Voici une représentation de l'arborescence restructurée à ce stade du développement, avant l'intégration de Symfony au projet :

```
.
+-- public                      # Dossier contenant les assets finaux - point de sortie des pipelines
|   |
|   +-- css
|   +-- fonts
|   +-- images
|   +-- js
|   +-- musics
|   +-- webfonts
|   |
|   +-- index.html              # Page unique du site
|
+-- resources
|   |
|   +-- modules                 # Modules spécifiques du projet
|   |   |
|   |   +-- module1             # Les modules sont composés du style et du composant Vue
|   |   |   |
|   |   |   +-- component.vue
|   |   |   +-- style.scss
|   |   |
|   |   +-- module2
|   |   +-- module3
|   |
|   +-- tenbuoLib               # Librairie personnelle, non gérée via npm
|   |   |
|   |   +-- modules             # Modules composés de composants, de stores et de directives.
|   |   +-- utils               # Scripts utilitaires, ex: Interpolator.js
|   |
|   +-- app.js                  # Point d'entrée de Webpack
|   +-- config.js               # Fichier contenant la configuration des composants du projet
|   +-- style.scss              # Point d'entrée de Webpack
|   +-- normalize.scss
|
+-- package.json.lock
+-- package.json
+-- webpack.config.js
```

Au vu de la nouvelle organisation des fichiers, j'ai mis en place des alias *Webpack,* dans un soucis de clarté lors des imports :

```
resolve:
{
    alias:
    {
        Vue: path.resolve('./node_modules/vue/dist/vue.js'),
        Config: path.resolve('./resources/config.js'),

        TenbuoModules: path.resolve('./resources/tenbuoLib/modules'),
        TenbuoUtils: path.resolve('./resources/tenbuoLib/utils'),
        TenbuoStyle: path.resolve('./resources/tenbuoLib/style'),
        Modules: path.resolve('./resources/modules')
    }
}
```

Back office

J'ai intégré *Symfony* au projet, et créé les entités *Doctrine* responsables de la lecture et de l'écriture des données dans la base. Ceci a été réalisé grâce à l'utilitaire en ligne de commande de *Symfony*, et avec l'appui de mon dictionnaire des données (cf annexes). Une fois les entités créées, il m'a suffit de générer et d'exécuter une migration afin de modifier la structure de la base de données en accord avec le modèle précédemment établi.

Doctrine est un *ORM* (Object Relational Mapper) permettant d'effectuer la lecture et l'écriture des données dans une base de données. Pour cela, il repose sur son composant *DBAL*, permettant de faire l'interface avec la base. *DBAL* permet malgré tout d'utiliser des requêtes *SQL* traditionnelles pour interagir avec la base de données, mais propose également un système de query builder, moyen alternatif de générer des requêtes *SQL* offrant, entre autres, une protection contre les injections.

Le bundle *EasyAdmin* m'a permis de générer automatiquement toute l'interface d'administration de l'application. Il a suffit pour cela de configurer le bundle afin de lui indiquer quelles entités devaient être ouvertes à l'administration, et quelles actions du CRUD étaient autorisées sur celles-ci.

Authentification

Il n'existe qu'un seul utilisateur autorisé à s'authentifier, l'administrateur du site, et l'authentification lui permet d'accéder à l'interface d'administration. Plusieurs éléments interviennent dans la gestion de l'authentification :

Le formulaire d'authentification :

J'ai utilisé le formulaire recommandé par la documentation de *Symfony*. Celui-ci est, entre autre, composé de deux champs cachés, l'un transmettant un jeton *CSRF* à la soumission du formulaire, l'autre indiquant la route sur laquelle rediriger l'utilisateur après authentification.

Le fichier de configuration du système d'authentification :

Le fichier `security.yaml` spécifie l'algorithme utilisé pour l'encryptage des mots de passe, l'entité à considérer pour gérer les utilisateurs, les différents pare-feux utilisés, et les modalités d'accès aux routes de l'application.

L'event listener Encrypt.php :

`Encrypt.php` permet de gérer le cryptage du mot de passe en écoutant les différents événements déclenchés lors du cycle de vie des entités *Doctrine.*

Api

J'ai souhaité dédier une *API* à la récupération des informations de la base de données depuis mes composants via des appels *AJAX*. En réalité, mes requêtes à la base de données étant relativement simples, j'ai utilisé une unique route à paramètre variable pour récupérer les informations : `/api/{type}`. Un boucle `switch` sur le paramètre `type` me permet de déterminer quelles données doit retourner la route. Celle-ci est uniquement accessible en `GET`, et est protégée des requêtes qui sont pas des *XmlHttpRequest*.

```php
class APIController extends AbstractController
{
    public function getDatas(Request $request, RegistryInterface $registry, $type)
    {
        if($request->isXmlHttpRequest())
        {
            if($request->getMethod() === "GET")
            {
                switch($type)
                {
                    case "musics":
                        $repo = new MusicRepository($registry);
                        $datas = $repo->findBy(array(), array('id' => 'desc'));
                        break;
                    case "photos":
                        $repo = new PhotoRepository($registry);
                        $datas = $repo->findBy(array(), array('id' => 'desc'));
                        break;
                    case "posters":
                        $repo = new PosterRepository($registry);
                        $datas = $repo->findBy(array(), array('id' => 'desc'));
                        break;
                    case "members":
                        $repo = new MemberRepository($registry);
                        $datas = $repo->findAll();
                        break;
                    case "friends":
                        $repo = new FriendLinkRepository($registry);
                        $datas = $repo->findAll();
                        break;
                    case "presentation":
                        $repo = new PresentationRepository($registry);
                        $datas = $repo->findAll();
                        break;
                    case "dances":
                        $repo = new DanceRepository($registry);
                        $datas = $repo->findAllDESC();
                        break;
                    case "mail":
                        $repo = new AdminRepository($registry);
                        $datas = ["mail" => $repo->findEffectiveAdmin()->getMail()];
                        break;
                    default:
                        return new Response("['no routes match']");
                }

                $json = json_encode($datas);
                return new Response($json);
            }
        }
        else
        {
            return $this->redirectToRoute('home');
        }
    }
}
```

Les divers `Repository` sont des objets fournis par *Doctrine* pour chaque entité, permettant de récupérer les données liées à chacune d'entre elles dans la base. Ils permettent également de définir des méthodes pour exécuter des requêtes *SQL* personnalisées.

Les données sont sérialisées et retournées au format *JSON*. Pour permettre la sérialisation des entités, il m'a fallu leur implémenter l'interface `JsonSerializable` et définir la méthode `jsonSerialize`, cette dernière permettant de décrire le processus de sérialisation des données en *JSON*. La serialisation s'opère ensuite grâce l'utilisation de la fonction `json_encode`, native de *php*.

Mailing

L'envoi d'une newsletter et la réception de mails de contact nécessitaient un mécanisme permettant la communication entre le serveur web et les serveurs SMTP. J'ai utilisé le bundle *SwiftMailer* pour ce faire.

Celui-ci permet d'effectuer une connexion à un serveur SMTP, en lui fournissant au préalable une adresse mail et le mot de passe associé. A partir de là, *SwiftMailer* est capable de configurer des mails et de les envoyer via la connexion décrite précédemment.

Adaptabilité

J'ai entrepris d'étudier l'adaptabilité du site face aux différents formats d'écrans. Cette préoccupation est intervenue trop tard dans le processus de développement, de sorte que mes feuilles de style étaient devenues difficilement maintenables, et que chaque ajustement entrainaient des effets de bords nécessitant d'autres modifications. De plus, les clients ayant formulés leurs demandes sur le modèle desktop du site, il m'a été relativement difficile de penser la mise en page en mobile first. J'en retiens aujourd'hui que l'adaptabilité doit être une préoccupation qui doit intervenir dès les premiers instants de la conception de la mise en page.

La mise en page du site a été configurée autour de l'image centrale de l'écran d'accueil, et ce fut le principal obstacle à l'adaptabilité du site. En effet, ladite image était configurée pour occuper toute la hauteur de l'appareil d'affichage, de sorte qu'une diminution de la largeur de l'écran implique un débordement du contenu à droite. Pour palier à ce débordement, les appareils mobile semblent diminuer les dimensions virtuelles de l'appareil d'affichage afin de conserver une totale visibilité du contenu pour l'utilisateur.

Ainsi, au lieu d'avoir un débordement du contenu nécessitant un scrolling latéral, il en résultait en différence entre la taille physique et virtuelle d'affichage, désorganisant intégralement le contenu de la page.

Pour résoudre le problème, il m'a fallu utiliser les media queries, en basant leur exécution sur les proportions de l'image centrale, et, ce faisant, basculer sur une mise page plus stable.

Cependant, en mode paysage, les proportions étaient telles que, même sur les petits écrans, le layout ne s'adaptait pas. A la précédente media querie, j'ai ajouté une condition sur les appareils d'une largeur inférieure à 769 pixels. J'ai également procédé à quelques ajustements ponctuels selon l'orientation de l'écran (portrait ou paysage) et la hauteur de l'écran.

Voici un extrait concernant les media queries gérant la mise en page globale du site :

```scss
@media (max-aspect-ratio: 3612/2672), (max-device-width : 769px)
{
    #background{
        display: none;
    }

    #foreground{
        width: 100%;
        height: auto;
        min-height: 100vh;

        background-image: url($photo_url);
        box-sizing: border-box;
        border:none;
        overflow: hidden;
    }

    header{
        display: block;
        width: 100%;
        height: auto;
        position: relative;
        top: 0;
        right: 0;

        margin: 0 auto 0 auto;
        padding: 0 0 1em 0;

        .logo{
            display: block;
            max-width: 100vw;
            width: unset;
            height: unset;
            max-height: 20vh;
            margin: 0 auto;
        }
    }

    nav{
        width: 80%;
        height: 20em;
        align-content: center;
        align-items: center;

        position: relative;
        top: 0;
        left: 0;

        margin: 1em auto 1em auto;
    }


    .photo{
        max-width: 100%;
        max-height: 300px;
        width: unset;
        height: unset;
        margin: 0 auto 0 auto;
        border: 5px double $orange;
    }

    @media (orientation: portrait)
    {
        header .logo{
            max-height: 25vh;
        }
    }
}
```

Refontes/Fix

Cette étape a consisté en la résolution de bug mineurs et la refontes de certains composants.

Mon composant *Image Viewer* posait un certain nombre de problèmes qui nécessitaient de le repenser.
Le défilement des images par à-coups semblait présenter une inexactitude dans sa valeur de déplacement, celle-ci s'accumulant au fil des intéractions de l'utilisateur. Ainsi, alors que les images affichées devaient être parfaitement centrée dans la fenêtre de visibilité du composant, celles-ci se positionnaient à terme sur les bords de ladite fenêtre.
Également, la visionneuse d'image grand format intégrée au composant n'interagissait qu'avec les images de celui-ci. Souhaitant que les images de la page d'accueil puissent également être affichées en grand, il me fallait en faire un composant à part entière.
J'ai alors refondu l'*Image Viewer* en deux nouveaux composants : le *Slider* et le *Photo Displayer*.

Mon lecteur audio était sujet à un bug concernant les événements *Drag* sur le navigateur Mozilla Firefox, que je me suis occupé de résoudre. Celui-ci est abordé ultérieurement dans la suite du dossier.

Sur *iOS5*, le navigateur présentait des soucis d'affichage. Il s'avérait que celui-ci n'interprétait pas l'*ES6* et, en particulier, la syntaxe des fonctions fléchées. J'ai donc utilisé *babel-loader* lors de la compilation des fichiers *javascript*.

Déploiement

Afin de pouvoir mettre le site en production, il m'a fallu modifier certaines options dans la configuration de *Webpack*. J'ai donc développé une fonction `prod` chargée d'ajuster les paramètres de compilation.

Cette fonction passe le mode de *Webpack* à `'prod'`. Egalement, elle permet de changer la version de *Vue* utilisée dans les fichiers compilés afin, d'une part, d'avoir une version minifiée, et d'autre part, d'éviter l'affichage de messages informatifs dans la console du navigateur. Enfin, j'ai ajouté un nouveau plugin chargé de minifier le *CSS*.

```
var configModifiers =
{
    "prod": function()
    {
        configDefault.mode = "production";
        configDefault.resolve.alias.Vue = path.resolve('./node_modules/vue/dist/vue.min.js');
        configDefault.plugins.push(new OptimizeCssAssetsPlugin({assetNameRegExp: /\.css$/g }));
    }
};
```

La commande `webpack --env='prod'` permet de passer la chaîne de caractère `'prod'` en paramètre de la fonction chargée de retourner la configuration de *Webpack*. Grâce à cela, j'ai pu exécuter la fonction définie ci-dessus afin d'opérer les changements de configuration pour la mise en production.

```
module.exports = function(e)
{
    if(e !== null && e !== undefined)
    {
        configModifiers[e]();
    }
    return configDefault;
}
```

Compteur de visiteurs

En fin de stage, l'association a souhaitée que je réalise une interface permettant de consulter le nombre de visites qui a eu lieu sur le site. Celle-ci permet de visualiser le nombre de visiteurs (différenciés par adresse Ip) et de visites (nombre totale des visites de tous les visiteurs) calculés par mois. Pour cela il m'a suffit de créer deux entités, Month et Visit, représentant respectivement un mois de l'année et une adresse ip associée à un nombre de visite.

Réalisations importantes

Interpolator

L'*Interpolator* est un objet *javascript* que j'ai développé afin factoriser et de normaliser les processus d'interpolation à l'origine des animations de mes composants.

Il aurait été possible, en lieu et place d'animations générées en *javascript*, d'utiliser les transitions *CSS*. Cela aurait apporté plus de simplicité et de performance à ces animations, mais je n'aurais cependant pas eu la main mise sur l'algorithme gérant la transition. Par conséquent, j'ai souhaité concevoir un système d'interpolation personnalisable en *javascript*, et, de surcroît, enrichir ma librairie.

L'animation est effectuée par itération successive d'un algorithme d'interpolation, grâce à la fonction `setInterval`. Chaque itération change la valeur courante d'interpolation passée en paramètre, et le composant réagit à cette modification. Le composant doit également savoir si un processus d'interpolation est en cours, afin d'adapter son comportement en fonction. Les paramètres en entrée du processus doivent donc être passés par référence afin que les composants puissent garder un oeil sur l'interpolation en cours. Voici l'objet *Interpolator* :

```
export default {
    startVals: [],
    currentTimes : [],

    start: function(datas, interval)
    {
        if(datas.id)
        {
            return;
        }
        datas.id = setInterval(this.process.bind(this, datas, interval), interval);
        this.startVals[datas.id] = datas.val;
        this.currentTimes[datas.id] = 0;
    },

    process: function(datas, interval)
    {
        var t = this.currentTimes[datas.id]/datas.duration;
        var factor = this.algo[datas.algo](t);

        datas.val = (1-factor) * this.startVals[datas.id] + factor * datas.endVal;

        this.currentTimes[datas.id] += interval;

        // Checking if duration is over
        if(this.currentTimes[datas.id] >= datas.duration)
        {
            this.stop(datas);
        }
    },
```

```
    stop: function(datas, achieve = true)
    {
        if(achieve)
        {
            datas.val = datas.endVal;
        }
        clearInterval(datas.id);
        datas.id= null;
    },

    algo:
    {
        "linear": function(t)
        {
            return t;
        },
        "EIO_cos": function(t)
        {
            return -0.5 * Math.cos(Math.PI * t) + 0.5;
        }
    }
}
```

L'objet `datas` en entrée de l'interpolation doit être constitué de :

- `id` : L'identifiant du processus d'interpolations successives retourné par la fonction setInterval. Nous devons garder en mémoire cet identifiant afin que les composants adaptent leur comportement selon qu'un processus soit en cours (l'identifiant sera alors un entier) ou non (l'identifiant sera alors null).
- `val` : La valeur courante d'interpolation. Cette valeur sera actualisée à chaque itération de l'interpolation, de sorte que les composants puissent réagir à chaque modification de celle-ci, et créer ainsi une animation.
- `endVal` : La valeur finale à atteindre à l'issue des interpolations successives.
- `duration` : La durée du processus.
- `algo` : Le nom de l'algorithme utilisé pour l'interpolation.

Deux tableaux internes à l'*Interpolator* permettent de garder en mémoire certaines valeurs nécessaires au processus d'interpolation successives.

- `startVals` : ce tableau liste toutes les valeurs de départ de chaque processus d'interpolation, chaque valeur étant enregistrée à un index correspondant à l'identifiant du processus concerné. La valeur de départ est déterminée comme étant la valeur courante à la première itération du `setInterval`.
- `currentTimes` : comme précédemment, à chaque index correspondant à l'id du processus concerné, est enregistré le temps écoulé depuis le lancement du processus.

L'*Interpolator* est composé de trois méthodes :

- `start` permet de déclencher le processus. C'est cette méthode qui est utilisée par les composants pour démarrer les interpolations successives. Elle nécessite en entrée les données d'interpolations décrites précedemment, ainsi que le pas du `setInterval` en millisecondes.

- `process` correspond au traitement à effectuer à chaque itération de l'interpolation. Il récupère la proportion du temps écoulé par rapport à la durée maximale de l'animation. Cette valeur est ensuite passée dans l'algorithme d'interpolation afin de récupérer la proportion de la valeur finale que nous souhaitons obtenir. Cette méthode est passée en paramètre de la fonction `setInterval`, ce qui génère les interpolations successives.

- `stop` permet d'interrompre le processus. Cette méthode est exécutée lorsque la durée courante de l'animation a atteinte la durée maximale précisée dans l'objet en entrée. Elle peut être également exécutée depuis un composant, si, par exemple, celui-ci a besoin de déclencher une nouvelle animation alors que la précédente n'est pas terminée. La paramètre `achieve` permet de déterminer si, lors d'un arrêt forcé depuis un composant, la valeur finale doit être assignée à la valeur courante.

Afin de rendre l'*Interpolator* accessible par tous mes composants, j'ai utilisé le système d'injection de dépendances de *Vue*. Il faut tout d'abord que l'instance racine de *Vue* fournisse ladite dépendance grâce à la méthode `provide`. Ensuite, dans les composants concernés, il suffit d'injecter la dépendance via la propriété `inject`.

Dans l'instance racine de Vue :

```
provide: function()
{
    return {
        interpolator: this.interpolator,
        calculation: this.calculation
    }
}
```

Dans un composant :

```
inject:['interpolator'],
```

Test et jeux d'essai

Afin de tester le fonctionnement de l'*Interpolator*, j'ai installé *Vue Devtools*, une extension permettant aux outils de développement du navigateur de visualiser l'état interne des composants et de les modifier.
J'ai réalisé ces tests à travers l'étude de l'animation de défilement du slider photo.

Données en entrée :

J'ai configuré les paramètres internes du composant de la sorte :
- pas du défilement: 50%
- algorithme d'interpolation: 'EIO_cos'
- durée du défilement: 10 secondes

Données attendues :

L'animation devait présenter une certaine accélération en début de course et une décélération en fin de course. Au clic sur le bouton de défilement, le contenu devait défiler de droite à gauche, sur une distance correspondant à la moitié de la largeur de la fenêtre de visibilité. L'animation devait durer 10 secondes.

Données en sortie :

Les données en sortie ont été conformes aux données attendues. La durée de l'animation était suffisamment longue pour ne pas laisser de doute sur l'exactitude de la mesure de celle-ci. Grâce à l'extension Vue Devtools, j'ai constaté que la valeur courante du défilement avait atteinte la valeur attendue. De plus, le slider s'était décalé d'exactement deux photographies, soit exactement la moitié de sa largeur. J'ai constaté à l'oeil nu que le défilement avait accéléré au début, puis décéléré à la fin, même si aucune mesure précise ne me permet de l'attester numériquement.

Lecteur audio

Avant d'entamer la description détaillées des composants réalisés avec Vue, je souhaite expliquer ici les termes et les concepts qui seront utilisés par la suite.

- Les props: ces propriétés sont spécifiquement utilisées pour configurer un composant. Elles reçoivent leur valeur d'un composant parent ou de l'instance racine de *Vue*.
- Les propriétés internes: ce sont les propriétés qui servent à gérer l'état interne du composant. Celles-ci sont réactives, de sorte que l'affichage et les propriétés calculées basées sur les propriétés internes sont réévalués lorsque la valeur de ces dernières change. Les propriétés internes sont définies dans la fonction `datas` du composant.
- Les propriétés calculées: ce sont des propriétés construites en surcouche d'une ou plusieurs propriétés internes ou props du composant. Elles agissent par défaut comme des getters, et comme des setters si on leur indique à la fois des entrées `get` et `set`.
- Les observateurs: ce sont des fonctions utilisées pour observer une propriété interne du composant. Ainsi, à chaque fois que la propriété est modifiée, l'observateur sera exécuté.
- Les hooks: Les composants proposent plusieurs hooks afin d'exécuter des instructions durant leur cycle de vie. Le hook `mounted` est appelé une fois le composant entièrement compilé. Seul celui-ci a été utilisé dans mes composants.
- Les directives: elles se présentent sous la forme d'attributs spécifiques aux balises *HTML* interprétées par *Vue*. Elles permettent d'induire un comportement sur un élément du *DOM* ou un composant. Les plus courantes sont les directives `v-bind`, permettant d'insérer une valeur dans un attribut natif à partir d'une propriété interne, `v-if`, soumettant l'affichage d'un élément à une condition, `v-for`, itérant l'affichage d'un élément sur une collection ou un interval, et `v-on`, permettant d'indiquer un événement à écouter et la fonction à exécuter lors du déclenchement de celui-ci.

Un lecteur audio devait être intégré sur le site, pour permettre l'écoute des musiques. Cependant, le lecteur fourni par la balise *HTML* `<audio>` n'étant guère personnalisable, j'ai souhaité développer celui-ci en *javascript*. Cela me permettait d'avoir une maîtrise totale sur le style et la logique du lecteur.

Configuration du composant

Voici les props ajoutées pour la configuration :

- `name` : le préfixe de l'identifiant du lecteur, afin d'effectuer la distinction entre plusieurs instances du même composant.
- `route` : la route permettant de récupérer les musiques
- `volume` : la valeur par défaut du volume du lecteur

```
props:
        {
            name:
            {
                type: String,
                default: '0'
            },
            route:
            {
                type: String,
            },
            volume:
            {
                type: Number,
                default: 1
            }
        },
```

Etats internes du composant

Voici les propriétés internes du composant :

- `musics` : référence chaque piste audio et les informations relatives à chacune d'entre elles.
- `selected` : référence l'élément de `musics[]` en cours de lecture.
- `playing` : indique si le lecteur est en pause ou en cours de lecture.
- `currentTime` : indique en seconde le moment de la musique en cours de lecture.
- `barBoundaries`: indique sous forme d'objet la largeur de la timebar en pixel, ainsi que ses extrémités gauche et droite en pixel dans le repère du document.
- `mouse`: indique l'abscisse en pixel de la souris par rapport au document.

```
data: function()
{
    return {
        // Musics registered
        musics: [],
        // Current track, from musics array
        selected: null,
        // Is playing any track ?
        playing: false,
        // Current track time
        currentTime: 0,
        // Player bar boundaries
        barBoundaries: {left: 0, right: 0, amplitude: 0},

        mouse:
        {
            x: 0
        }
    };
},
```

Récupération et paramétrage des pistes audios

Les chemins vers les ressources audios sur le serveur sont récupérés par un appel *AJAX*, lors de l'exécution du hook `mouted`. La méthode `addMusic` permet ensuite de créer les musiques et de les paramétrer.

```
mounted: function()
{
    this.$http.get(this.route).then(function(response){

        for(var music of response.data)
        {
            this.addMusic(music);
        }

        this.computedSelected = this.musics[0];

    }, function(error){
        console.log("erreur ajax");
    });

    document.addEventListener('dragover', (e) => {
        this.mouse.x = e.clientX;
    }, true);
}
```

`addMusic` procède comme suit :

- Instanciation d'une nouvelle piste audio grâce au constructeur de l'objet `Audio`
- Création d'un objet `music`, constitué de l'Id de la piste en base de données, du titre de la piste, de la piste audio, et de la durée de la piste.
- Ajout de l'objet précédemment créé dans le tableau `musics[]`.
- Ecoute de l'événement `timeUpdate` de la piste audio afin de changer le `currentTime` du composant.
- Ecoute de l'événement `loadedmetadata` afin récupérer le durée de la piste audio et de l'assigner dans l'objet `music`.
- Ecoute de l'événement `ended` afin de changer la piste courante automatiquement lorsque celle en cours se termine.

```
addMusic: function(music)
{
    var id = music.id;
    var title = music.title;
    var audio = new Audio("." + music.path);
    audio.volume = this.volume;
    var duration = 1;

    var music =
    {
        id: id,
        title: title,
        audio: audio,
        duration: duration
    };

    this.musics.push(music);

    var onTimeUpdate = function(e)
    {
        this.currentTime = e.target.currentTime;
    };
    var onLoadedMeta = function(e)
    {
        this.musics.find((music) => { return music.id === id}).duration = e.target.duration;
    };
    var onEnded= function(e)
    {
        var id = this.computedSelected.id;
        var k = this.musics.findIndex((music) => { return music.id === id});

        this.switchOnOff();

        // Ckecking if k is last index
        this.computedSelected = (k >= this.musics.length-1) ? this.musics[0] : this.musics[k+1];

        this.switchOnOff();
    };

    audio.addEventListener('timeupdate', onTimeUpdate.bind(this) );
    audio.addEventListener('loadedmetadata', onLoadedMeta.bind(this) );
    audio.addEventListener('ended', onEnded.bind(this) );
},
```

Affichage du lecteur

Le lecteur audio est composée d'un élément `player-controls` et d'un élément `player-list`. Ce dernier permet simplement d'afficher l'ensemble des musiques disponibles du lecteur. L'élément `player-controls` contient les boutons permettant de lancer et d'interrompre la lecture, ainsi que la timeline. L'élément `player-timeline` est lui-même composé des éléments `player-time`, permettant d'afficher les durées relatives à la piste en cours de lecture, et d'un élément `player-bar`, permettant de symboliser l'avancement de la piste audio sous la forme d'un curseur sur une barre. L'élément `player-bar` sera appelé timebar dans la suite de l'explication.

Voici le template du composant :

```
<template>

    <div v-bind:id="'player-'+name" class="player">

        <div class="player-controls">

            <div class="player-title" >{{computedSelected.title}}</div>

            <div v-if="!playing" v-on:click="switchOnOff" class="player-play player-button ">&#9655</div>
            <div v-else v-on:click="switchOnOff" class="player-pause player-button fas fa-pause"></div>

            <div class="player-timeline">

                <div class="player-time player-current">{{secondsToMinutes(currentTime)}}</div>

                <div class="player-bar" v-on:click="barClick">
                    <div class="player-cursor"
                         draggable="true"
                         v-on:drag="drag"
                         v-on:dragstart="dragstart"
                         v-on:dragend="dragend"
                         v-bind:style="{left: computedLeft}">
                    </div>
                </div>

                <div class="player-time player-total">{{secondsToMinutes(computedSelected.duration)}}</div>

            </div>

        </div>

        <ul class="player-list">
            <li v-for="music in musics" v-on:click="selectMusic" v-bind:data-id="music.id" class="player-item">{{music.title}}</li>
        </ul>
    </div>

</template>
```

Les propriétés internes `selected` et `musics` étant respectivement nulle et vide tant que la requête *AJAX* n'est pas achevée, il a fallu construire une propriété calculée en surcouche de celles-ci, nommée `computedSelected`, afin d'éviter les erreurs. Celle-ci est utilisée pour l'affichage du titre de la piste courante et de la durée totale de cette dernière.

```
computedSelected:
{
    get: function()
    {
        if(this.selected == null)
        {
            if(this.musics.length === 0)
            {
                return {id: -1 , title: 'aucune musique sélectionnée', audio: null, duration: 1};
            }
            else
            {
                return this.musics[0];
            }
        }

        return this.selected;
    },
    set: function(val)
    {
        this.selected = val;
    }
},
```

Pour afficher le moment de la piste en cours de lecture, j'ai utilisé directement la propriété interne `currentTime`. Cependant, `currentTime` et `computedSelected.duration` sont des valeurs numériques en secondes qu'il me fallait formater en chaîne de caractère plus explicite pour l'affichage. D'où la méthode `secondsToMinute` :

```
secondsToMinutes: function(t)
{
    var minutes = "0" + Math.floor(t / 60).toString();
    var seconds = "0" + Math.floor(t % 60).toString();
    var time = minutes.substr(-2) + ":" + seconds.substr(-2);
    return time;
},
```

Le déplacement du curseur sur la timebar est géré grâce à la génération de style inline dans le template. Le propriété calculée `computedLeft` est responsable de la génération dudit style, et exprime le positionnement de la bordure gauche du curseur en tant que pourcentage du temps écoulé par rapport à la durée de la piste.

```
computedLeft: function()
{
    var posX = this.currentTime / this.computedSelected.duration * 100;
    return posX.toString() + '%';
},
```

La liste des musique est affichée dans une liste non-ordonnée, en parcourant tous les éléments de la propriété interne `musics` et en affichant leur titre grâce à la directive `v-for`.

Le bouton play/pause consiste en deux boutons s'affichant alternativement. L'alternance se fait grâce à une condition sur la propriété interne playing et à la directive `v-if`.

Interaction

La méthode `switchOnOff` est déclenchée au clic de l'utilisateur sur les boutons play et pause. Elle permet de changer la propriété interne `playing`, modifiant ainsi l'affichage du bouton. Cette méthode permet également de déclencher ou d'interrompre la lecture en cours.

```
switchOnOff: function()
{
    if(!this.playing)
    {
        this.playing = true;
        this.computedSelected.audio.play();
    }
    else
    {
        this.playing = false;
        this.computedSelected.audio.pause();
    }
},
```

La méthode `selectMusic` est déclenchée au clic de l'utilisateur sur un élément de la liste des musiques. Elle récupère l'Id de la musique correspondante (transmise grâce à un `dataset`), retrouve cette dernière dans le tableau des musiques, et l'assigne à la propriété `selected` afin d'en faire la piste courante du lecteur.

```
selectMusic: function(e)
{
    var id = parseInt(e.target.dataset.id);

    if(id !== this.computedSelected.id)
    {
        this.computedSelected.audio.currentTime = 0;

        if(this.playing)
        {
            this.switchOnOff();
        }

        this.computedSelected = this.musics.find((music)=>{return music.id === id});

        this.switchOnOff();
    }
    else
    {
        if(!this.playing)
        {
            this.switchOnOff();
        }
    }
},
```

Deux types d'interaction permettent de changer le moment de la lecture : un clic sur la timebar ou un drag'n'drop du curseur sur la timebar. Ces deux interactions reposent sur le même principe.

On récupère la position de la souris sur le document et les dimensions de la timebar. La fonction `setCurrentTimeFromMouseX` permet, à partir des données précédentes, de trouver la position de la souris par rapport à la timebar. Cette position, en pourcentage, permet de retrouver le temps voulu par rapport à la durée maximale de la piste en cours.

Une fois le temps courant calculé, il suffit de l'assigner à la propriété `currentTime` de l'objet `Audio`. Cela déclenche l'événement `timechange` et ses écouteurs. et la propriété interne `currentTime` du composant se met à jour. L'affichage suivra naturellement cette mise à jour.

```
setCurrentTimeFromMouseX: function(x)
{
    x = Math.min(x, this.computedBoundaries.right);
    x = Math.max(x, this.computedBoundaries.left);
    x = x - this.computedBoundaries.left;

    var time = this.computedSelected.duration * x / this.computedBoundaries.amplitude;

    this.computedSelected.audio.currentTime = time;
},
```

Le première façon de déclencher ce procédé est un clic de l'utilisateur sur la timebar. L'événement `click` ainsi déclenché va exécuter la fonction `barClick`.

```
barClick: function(e)
{
    this.computedBoundaries = this.$refs.bar.getBoundingClientRect();
    var x = e.clientX;
    this.setCurrentTimeFromMouseX(x);
},
```

La seconde manière utilise les événements drag. Les méthodes `dragStart`, `drag` et `dragEnd` écoutent les événements du même nom sur le curseur.

La méthode `dragstart` permet d'interrompre le lecteur et de récupérer les dimensions de la timebar. Il permet également de définir l'image accrochée au curseur pendant le drag (aucune dans notre cas) et les données relatives au drag. Ces dernières sont en fait les données qui seront transmises si un événement drop est déclenché. Cependant, nous n'utilisons pas l'événement drop pour gérer l'interaction.

La méthode `drag` exécute la méthode `setCurrentTimeFromMouseX`, afin de permettre au curseur de la timebar de suivre le mouvement de la souris.

La méthode `dragend` permet d'exécuter une dernière fois `setCurrentTimeFromMouseX`, puis de relancer la lecture.

```
dragstart: function(e)
{
    if(this.playing)
    {
        this.computedSelected.audio.pause();
    }
    e.dataTransfer.setDragImage(new Image(), 0, 0);
    e.dataTransfer.setData('text/html', 'for mozilla to trigger the event');

    this.computedBoundaries = this.$refs.bar.getBoundingClientRect();
},
drag: function(e)
{
    this.setCurrentTimeFromMouseX(this.mouse.x);
},
dragend: function(e)
{
    this.setCurrentTimeFromMouseX(this.mouse.x);

    if(this.playing)
    {
        this.computedSelected.audio.play();
    }

},
```

Le système de drag fonctionnait parfaitement sur Chrome, mais il a fallu faire des ajustements pour Mozilla. En effet, l'abscisse de la souris était à l'origine récupérée dans l'objet transmis en paramètre des événements drag, via la propriété `clientX`. Cependant, sur Mozilla, ladite propriété était toujours égale à 0, sauf durant l'event `dragstart`. Après plusieurs recherche, cela s'est avéré être un bug inhérent à Mozilla.

Ainsi, j'ai tenté d'écouter l'événement `mouseOver` sur le document pour traquer la position de la souris. Cependant, `mouseOver` n'était pas déclenché durant les événements drag. Finalement, l'événement `dragOver` du document m'a permis de récupérer l'abscisse de la souris. J'ai configuré le l'écouteur du `dragOver` dans le hook `mounted`.

Egalement, sur Mozilla, les événements drag s'interrompaient inopinément si l'on n'indiquait pas explicitement le type MIME et le contenu des données faisant l'objet du drag. Même si ces données sont à destination de l'event drop, dont je n'ai pas l'usage, il a fallu néanmoins les préciser dans la méthode `dragStart`.

Afin de tester le bon fonctionnement du système de déplacement du curseur de la timebar, j'ai installé Vue Devtools, extension du navigateur permettant de visualiser l'état interne des composants.

Test et jeu d'essai :

Le test a consisté en un clic maintenu sur le curseur du lecteur, puis au déplacement de celui-ci. L'idée était de constater le changement de l'état interne du lecteur lors du déplacement et du relâchement du curseur.

Données en entrée :

Variation de la position du curseur entre le début et la fin de la timebar.

Données attendues :

Tel qu'expliqué précedemment, la propriété currentTime du composant devrait être mise à jour en fonction de la position du curseur, variant entre 0 et le nombre de secondes de la durée de la piste en cours de lecture.

Données en sortie :

Conforme aux données attendues.

Slider photo

Afin de permettre au visiteur de voir les photos de l'association, j'ai développé un composant dédié à l'affichage de celles-ci. Etant donné qu'il n'existe pas de limite au nombre de photos pouvant être postées par l'administrateur, le composant devait pouvoir charger les photos en différé, et seulement en cas de besoin. De plus, des contraintes liées à la mise en page du site imposait d'avoir du contenu débordé, accessible seulement par interaction de l'utilisateur. Le composant en question répond à ces deux problématiques et prend la forme d'un slider. Celui-ci est composé de deux slides, se relayant tour à tour pour afficher du contenu à l'utilisateur.

Configuration du slider

Voici les props chargées de transmettre la configuration au composant :

- `name` : cette chaîne de caractère indique le préfixe de l'identifiant du slider, afin d'effectuer la distinction entre plusieurs instances du même composant.
- `scrollOpt` : cet objet permet de préciser la durée de la transition entre chaque slide, ainsi que l'algorithme utilisé pour l'interpolation des positions.
- `nb` : cet entier précise le nombre de photos affichées sur chaque slide.
- `route` : cette chaîne de caractère permet d'indiquer la route à utiliser pour récupérer les photos.
- `addStyle` : cette fonction permet d'ajouter un style inline sur chaque photo.

```
props:
{
    name:
    {
        type: String,
        default: "main",
    },
    scrollOpt:
    {
        type: Object,
        default: () =>
        {
            return {
                duration: 1000,
                algo: "EIO_cos"
            }
        }
    },
    nb:
    {
        type: Function,
        default: () => {return 1}
    },
    route:
    {
        type: String,
        default: null
    },
    addStyle:
    {
        type: Function,
        default: () => {return ""}
    }
}
```

Propriétés internes

Voici les propriétés internes du composant, autour desquelles s'articule l'essentiel de sa logique :

- `isMounted`: ce booléen permet de savoir si le composant a été compilé et le template inséré dans DOM.
- `slideProcess` : cet objet fournit les données de l'animation de transition entre les slides, et est formaté selon les besoins de l'Interpolator.
- `datas` : ce tableau liste les identifiants et le chemin des photographies sur le serveur.
- `datas1` et `datas2` : ces deux tableaux indiquent les éléments de datas qui seront effectivement affichés par chacune des deux slide.
- `index` : cet entier précise à partir de quel index du tableau `datas` doit s'effectuer la sélection des éléments à afficher.
- `nextSlide` : cette propriété garde une référence de la slide qui sera affichée lors de la prochaine interaction de l'utilisateur.

```
data: function()
{
    return {
        isMounted: false,
        slideProcess:
        {
            id: null,
            duration: this.scrollOpt.duration,
            val: 0,
            endVal: 100,
            algo: this.scrollOpt.algo
        },
        datas: [{id: 0, path: ""}],
        datas1: [],
        datas2: [],
        index: 0,
        nextSlide: null,
    }
}
```

Affichage du slider

Le slider est composé d'une `slider-box`, constituant la fenêtre d'affichage du contenu, et des boutons `slider-left` et `slider-right`, permettant de faire défiler le contenu.

La `slider-box` elle-même est composée de deux `slider-slide`, se relayant pour afficher le contenu. Lorsque que l'une des deux slides apparaît dans la fenêtre de visibilité, l'autre disparaît. Les slides sont positionnées de manière absolue par rapport au bloc parent, la `slider-box`, et l'apparition et la disparition s'effectuent en faisant varier le positionnement du bord gauche des slides.

Chaque slide est composée d'un nombre précis de `slide-part`. Ce nombre est défini lors de la configuration du composant et chaque `slide-part` a pour rôle d'afficher une photographie. Également, le nombre de `slide-part` affichées par slide est défini grâce à une fonction retournant un entier variant selon la taille de l'écran. Ainsi, sur les petits écrans orientés au format portrait, seul une photo sera affichée par slide, tandis que dans les autres cas, quatres photos seront affichées. Les `slide-part` sont générées en itérant sur les propriétés internes `datas1` et `datas2`, grâce à la directive `v-for`.

```
nb:function()
{
  if(window.matchMedia("(max-aspect-ratio: 3612/2672), (max-device-width : 769px)").matches)
  {
      if(window.matchMedia("(orientation: portrait)").matches)
      {
          return 1;
      }
      else
      {
          return 4;
      }
  }
  return 4;
},
```

```
<template>
    <div v-bind:id="'slider-'+name" class="slider">
        <div class="slider-box">

            <div class="slider-slide"
                ref="slider-slider-1"
                :style="transform1">

                <div class="slider-slide-part" v-for="(data, index) in datas1">
                    <img v-display:main
                        :key="index"
                        :src="data.path"
                        :style="data.style"/>
                </div>

            </div>

            <div class="slider-slide"
                ref="slider-slider-2"
                :style="transform2">

                <div class="slider-slide-part" v-for="(data, index) in datas2">
                    <img v-display:main
                        :key="index"
                        :src="data.path"
                        :style="data.style"/>
                </div>

            </div>

        </div>

        <div id="slider-left" class="slider-btn fas fa-chevron-left"
            v-on:click="slide('left')">
        </div>

        <div id="slider-right" class="slider-btn fas fa-chevron-right"
            v-on:click="slide('right')">
        </div>
    </div>
</template>
```

Interaction

Durant le hook `mounted` du composant, on effectue un appel *AJAX* à l'aide de *Vue-resources*, afin de récupérer le chemin et l'identifiant des photos sur le serveur. On exécute également la méthode `initialize`, afin d'initialiser les données de la première slide visible au chargement de la page. Le contenu de la deuxième slide, celle actuellement cachée de l'utilisateur, sera mise à jour au défilement du slider.

```
mounted: function()
{
    this.$http.get(this.route).then(function(response){
        this.datas = [];
        for(var image of response.data)
        {
            image.style = this.addStyle();
            this.datas.push(image);
        }
        this.initialize();
    }, function(error){
        console.log("erreur ajax - slider");
    });

    this.initialize();
    this.isMounted = true;
}
```

```
initialize: function()
{
    this.slideProcess.val = 0;
    this.nextSlide = this.$refs['slider-slide-1'];
    this.updateDatas();
    this.nextSlide = this.$refs['slider-slide-2'];
}
```

Au clic de l'utilisateur sur l'un des boutons de défilement, la méthode `slide` est exécutée. Celle-ci permet de lancer l'animation du positionnement par la gauche des deux slide, d'incrémenter ou décrémenter la propriété `index` et de mettre à jour les données des slide en fonction de celle-ci.

```
slide: function(type)
{
    if(this.slideProcess.id !== null)
    {
        return;
    }

    if(type === "right")
    {
        // We find the closest hundred before the current value.
        this.slideProcess.endVal = Math.ceil((this.slideProcess.val - 100) / 100) * 100;

        this.index -= this.nb();

    }
    else
    {
        // We find the closest hundred after the current value.
        this.slideProcess.endVal = Math.floor((this.slideProcess.val + 100) / 100) * 100;

        this.index += this.nb()
    }

    this.updateDatas();

    this.interpolator.start(this.slideProcess, 10);
},
```

L'animation s'effectue en changeant la valeur finale `endVal` du positionnement dans la propriété interne `slideProcess`, puis en lançant les interpolations successives grâce à l'*Interpolator*. La valeur finale d'interpolation est toujours un multiple de 100, et on incrémente ou décrémente de 100 selon que l'on fasse défiler le contenu vers la droite ou la gauche. A partir de là, la valeur courante de l'animation change au cours des interpolations successives jusqu'à atteindre la valeur finale. Cette valeur courante est réactive.

Deux propriétés calculées sont construites en surcouche de la valeur courante, `transform1` et `transform2`, dont le rôle est de générer le positionnement par la gauche de chacune des deux slide, et de le rendre effectif sous la forme d'un style inline.

Ces propriétés calculées fonctionnent de la sorte: elles récupèrent la valeur courante d'interpolation et la transposent dans l'interval [-100, 99]. Lorsqu'une slide atteint la position -100, alors celle-ci est entièrement cachée de l'utilisateur, et elle doit être considérée comme la prochaine slide à mettre à jour, ce qui se traduit pas son assignation à la propriété `nextSlide`. Enfin, les propriétés calculées retourne le style inline sous forme de chaîne de caractère. Il est à noter que l'on ajoute un décalage de 100 au calcul du positionnement pour générer un écart entre les deux slide.

```
transform1: function()
{
    // getting a the remainer of the division.
    var remainer = ((100 + this.slideProcess.val) % 200);
    // getting the modulus (positive remainer).
    var modulus = (remainer + 200)%200;
    // translating the value between -100 and 99
    var pos =  modulus -  100;

    if(pos === -100 && this.isMounted)
    {
        this.nextSlide = this.$refs["slider-slide-1"];
    }

    return {
        transform: "translate(" + pos + "%, 0%)",
    }
},
```

```
transform2: function()
{
    var remainer = ((this.slideProcess.val) % 200);
    var modulus = (remainer + 200)%200;
    var pos = modulus -  100;

    if(pos === -100 && this.isMounted)
    {
        this.nextSlide = this.$refs["slider-slide-2"];
    }

    return {
        transform: "translate(" + pos + "%, 0%)",
    }
},
```

La mise à jour du contenu de chaque slide s'effectue grâce à la méthode `updateDatas`. Celle-ci récupère un certain nombre d'éléments du tableau `datas`, et ce à partir de l'index en cours défini par la propriété interne `index`. Ensuite, elle affecte ces mêmes éléments à la propriété `datas1` ou `datas2`, selon la prochaine slide référencée par `nextSlide`. Le contenu des slide se met automatiquement à jour à l'actualisation de `datas1` et `datas2`, grâce au système de réactivité de *Vue*.

```
updateDatas: function()
{
    var datas = [];

    for(var i= this.index; i < this.index + this.nb(); i++)
    {
        var l = this.datas.length;
        var j = ((i % l) + l) % l;
        datas.push(this.datas[j]);
    }

    if(this.nextSlide === this.$refs['slider-slide-1'])
    {
        this.datas1 = datas;
    }
    else if(this.nextSlide === this.$refs['slider-slide-2'])
    {

        this.datas2 = datas;
    }

},
```

Cette manière d'afficher du contenu permet de ne requêter que les photos que l'utilisateur souhaite visionner, ce qui s'avère avantageux pour les forfaits mobiles et les connexions limitées.

Visionneuse photo et directive display

J'ai mis en place un système permettant de visualiser les image du site en grand format. Ce système est composé de trois sous-parties fonctionnant conjointement :

- le composant *PhotoDisplayer* : il contient le template *HTML* et la logique de la visionneuse responsable de l'affichage des images.
- le store *displayers* : il permet d'enregistrer tous les composants *PhotoDisplayer* instanciés dans l'application afin d'en déclencher l'affichage.
- la directive *display* : cette directive peut être placée sur n'importe quelle balise image. Elle permet de qualifier les images pouvant être affichées par la visionneuse, et transmet les données de la balise aux visionneuses enregistrées dans le store.

Le *PhotoDisplayer* est un simple bloc de contenu encapsulant une image. Il est configuré pour couvrir toute la surface de l'écran, de sorte que l'utilisateur ne puisse plus interagir avec le reste du site tant que la visionneuse n'est pas fermée. Un style inline régit son apparition et sa disparition, grâce à la propriété *CSS* `display` que l'on alterne entre `none` et `block`. Également, au clic de l'utilisateur, on déclenche la méthode `close`, qui change le style `display` à `none`.

Afin de pouvoir faire le lien entre les directives et la visionneuse, il fallait que le store *displayers* ait connaissance de cette dernière. J'ai effectué l'inscription de la visionneuse dans le store lors de l'exécution du hook `mounted`.

```
<template>
    <div class="photo-displayer" :id="'photo-displayer-'+name" :style="style" v-on:click="close">

        <img :src="src"/>

    </div>
</template>

<script>
import store from './s-displayers.js';

export default
{
    props:
    {
        name:
        {
            type: String,
            default: "main"
        }
    },
    data: function()
    {
        return {
            style:
            {
                display: "none",
            },
            src: ""
        }
    },
    methods:
    {
        open: function()
        {
            this.style.display = "";
        },
        close: function()
        {
            this.style.display = "none";
            this.src = "";
        }
    },
    mounted: function()
    {
        store.commit('register', this);
    }
}
</script>
```

Le comportement du store est relativement simple. Il possède un état interne `displayers` permettant de garder en mémoire les différentes visionneuses présentes sur le site, et les enregistre sous une clef correspondant à leur nom. Cet enregistrement se fait grâce à la mutation `register`, exécutée à partir du composant. Le store comporte également une action `display`, permettant de récupérer une de ses visionneuses, de lui transmettre la source d'une image et de déclencher sa méthode `open` pour l'afficher à l'utilisateur.

```
import Vue from 'Vue';
import VueX from 'vuex';
Vue.use(VueX);

export default new VueX.Store({

    state:
    {
        displayers:
        {

        }
    },
    mutations:
    {
        register: function(state, displayer)
        {
            state.displayers[displayer.name] = displayer;
        }
    },
    actions:
    {
        display: function(context, display)
        {
            var displayer = context.state.displayers[display.displayerName];

            displayer.src = display.src;
            displayer.open();
        }
    }

});
```

La directive *display* se met dans les balises images dont on souhaite permettre l'affichage par la visionneuse. Il faut également indiquer en argument de la directive le nom de la visionneuse qui se chargera de l'affichage.

Exemple : `<img v-display:mainDisplayer src= “some/src.png” />`

La directive permet d'écouter l'événement `click` sur l'élément qu'elle qualifie. Lorsque cet événement se déclenche, elle exécute l'action `display` du store précédemment défini, en lui passant en argument le nom de la visionneuse pour l'affichage et la source de l'image à afficher. Cette action déclenchera alors l'ouverture de la visionneuse.

```
import Vue from 'Vue';
import store from './s-displayers.js';

export default (function()
{
    Vue.directive('display',
    {
        bind:function(el, binding, vnode)
        {
            var display = {displayerName: binding.arg, src: el.src};

            el.addEventListener('click', ()=>{
            store.dispatch('display', display);
            })
        },

        update:function(el, binding, vnode)
        {
            var display = {displayerName: binding.arg, src: el.src};

            el.addEventListener('click', ()=>{
            store.dispatch('display', display);
            })
        },

    });
})();
```

Interpellations et directives hail

Le système d'interpellation permet d'exécuter à distance les méthodes d'un composant (composant interpellé), en écoutant les événements d'un élément de la page (élément interpellateur). Cela permet, combiné au store *Vue-x*, de faciliter l'interaction entre les composants et les divers éléments de la page, sans devoir parcourir l'entièreté de l'arborescence de *Vue*, ni devoir référencer globalement les composants et les éléments concernés.

La directive *hailer* définit un élément que nous appellerons l'interpellateur. Elle se voit préciser en argument une chaîne de caractère qui sera le nom de l'interpellateur. Lorsque cette directive est liée à un élément du *DOM*, elle écoute par défaut l'événement `click` sur celui-ci, ou tout autre événement qui lui serait précisé en valeur de la directive.

Exemple: `<button v-hailer:boutonNav="mousedown">`.

Lorsque l'événement en question est déclenché, elle exécute l'action `hail` du store.

```
import Vue from 'Vue';
import store from './s-hailed.js';

export default (function()
{
    Vue.directive('hailer',
    {
        bind: function(el, binding)
        {
            var event = (binding.value === undefined) ? 'click' : binding.value;
            var hailer = binding.arg;
            el.addEventListener(event, function(){
                store.dispatch('hail', hailer);
            });
        }
    });
})();
```

La directive *hailed* permet de définir les éléments interpellés. Il doit lui être précisé en argument le nom de l'interpellateur auquel elle réagit, et en valeur le nom de la méthode qu'elle doit déclencher ainsi que les paramètres qui doivent être passés à celle-ci.

Exemple: `<div v-hailed:boutonNav="['show','true']">`.

A la liaison avec un composant, cette directive récupère la méthode précisée dans les valeurs, appelée `reaction`, et ses paramètres sous forme de tableau. Afin de pouvoir préparer la `reaction` à l'exécution, il nous faut utiliser la méthode `bind` pour lui passer ses arguments.

Malheureusement, nos arguments sont uniquement disponibles sous forme de tableau, alors que bind n'admet que des argument listés de manière traditionnelle. Pour résoudre ce problème, il nous faut appliquer la méthode `apply` à la fonction `bind` afin de passer un tableau d'argument à cette-dernière. Une fois la réaction correctement configurée, il nous ne reste plus qu'à l'enregistrer dans le store avec le nom de l'interpellateur comme clef d'enregistrement.

```
import Vue from 'Vue';
import store from './s-hailed.js';

export default (function()
{
    Vue.directive('hailed',
    {
        bind: function(el, binding, vnode)
        {
            var method = binding.value[0];
            binding.value[0] = vnode.componentInstance;
            var args = binding.value;

            var reaction = vnode.componentInstance[method];
            reaction = reaction.bind.apply(reaction, args);

            var hailed =
            {
                by: binding.arg,
                reaction: reaction
            }
            store.commit('register', hailed);

        }
    });
})();
```

Le store est l'élément faisant la liaison entre l'interpellateur et l'interpellé. Il possède une propriété interne `hailedBy`, dont chaque entrée correspond à un interpellateur. A chaque entrée sont inscrites les réaction attendues des interpellés. Une mutation `register` permet justement d'effectuer cet enregistrement, et est exécutée depuis la directive `hailed`. Une action `hail` permet de déclencher toutes les réactions à un interpellateur spécifique. Cette méthode est exécutée depuis la directive `hailer`.

```
import Vue from 'Vue';
import VueX from 'vuex';
Vue.use(VueX);

export default new VueX.Store({

    state:
    {
        hailedBy:
        {

        }
    },
    mutations:
    {
        register: function(state, hailed)
        {
            if(!state.hailedBy.hasOwnProperty(hailed.by))
            {
                state.hailedBy[hailed.by] = [];
            }

            state.hailedBy[hailed.by].push(hailed.reaction);
        }
    },
    actions:
    {
        hail: function(context, hailer)
        {
            var hailedBy = context.state.hailedBy;

            if(!hailedBy.hasOwnProperty(hailer))
            {
                return;
            }

            for(var reaction of hailedBy[hailer])
            {
                reaction();
            }
        }
    }

});
```

Dialogue et directive dial

Le système de dialogue permet l'interaction entre plusieurs composants *Vue*. Il permet, comme précédemment avec l'interpellation, d'éviter les références globales et le parcours fastidieux de l'arborescence des composants. Le principe consiste en l'inscription d'un composant sur un canal, grâce à la directive *dial*, ce qui lui permettra de dialoguer avec d'autres composants. A l'inscription sur le canal du store *Vue-x*, elle devra préciser deux événements : un événement `talk`, et un événement `answer`. Lorsque, sur un canal, l'événement `talk` d'un élément est déclenché, alors les événements `answer` de tous les autres éléments seront déclenchés en retour.

La directive *dial* doit être liée à un composant. Le nom des événements doivent être précisés en valeur de la directive, et le nom du canal de destination doit être indiqué en argument de celle-ci.

Exemple : `<div v-dial:someChannel="['onPop','onVanish']">`.

Les deux événements doivent être définis dans les propriétés internes du composant qualifié par la directive.

A la liaison avec un élément du *DOM*, la directive récupère : les événements sur l'instance du composant, l'élément en question et le canal de destination pour procéder à l'enregistrement dans le store.

```
import Vue from 'Vue';
import store from './s-chans.js';

export default (function()
{
    Vue.directive('dial',
    {
        bind: function(el, binding, vnode)
        {
            var dialer = {};
            dialer.el = el;
            dialer.talk = vnode.componentInstance._data[binding.value[0]];
            dialer.answer = vnode.componentInstance._data[binding.value[1]];
            dialer.chan = binding.arg;

            store.dispatch('register', dialer);
        }
    });
})();
```

L'enregistrement dans le store s'effectue en premier lieu grâce à l'action `register`. Celle-ci déclenchera tout d'abord la mutation `register`, permettant l'ajout d'un nouvel élément dans un canal donné. L'action `register` exécute ensuite l'action `transmit`, permettant d'écouter l'événement `talk` du nouvel inscrit, lui demandant de déclencher les événements `answer` des autres participants.

```
import Vue from 'Vue';
import VueX from 'vuex';
Vue.use(VueX);

export default new VueX.Store({

    state:
    {
        chans:
        {

        }
    },
    mutations:
    {
        register: function(state, dialer)
        {
            if(!state.chans.hasOwnProperty(dialer.chan))
            {
                state.chans[dialer.chan] = [];
            }

            var datas = [dialer.el, dialer.answer];
            state.chans[dialer.chan].push(datas);
        }
    },
    actions:
    {
        register: function(context, dialer)
        {
            context.commit('register', dialer);
            context.dispatch('transmit', dialer);
        },
        transmit: function(context, dialer)
        {
            dialer.el.addEventListener(dialer.talk.type,() => {
                for(var otherDialer of context.state.chans[dialer.chan])
                {
                    if(otherDialer[0] !== dialer.el)
                    {
                        otherDialer[0].dispatchEvent(otherDialer[1]);
                    }
                }
            });
        }
    }

});
```

Volets déroulants

Le contenu du site s'affiche aux utilisateurs par le biais de volets déroulants. L'apparition et la disparition des volets se fait grâce à l'*Interpolator*, dans le même principe que ce qui a été vu avec le *Slider*. Je ne détaillerais donc pas leur logique interne car elle est très similaire à ce qui a déjà été expliqué auparavant. Cependant, les interactions des volets avec les autres éléments de la page sont intéressantes en ce qu'elles synthétisent et illustrent bien les systèmes d'interpellation et de dialogue.

La navigation sur le site se fait grâce à la barre de navigation, composée de boutons de navigation. Ces derniers sont qualifiés par la directive *hailer* et deviennent de ce fait des interpellateurs.

```
<nav>
    <app-nav-button v-hailer:navbtn v-hailer:presentation name="presentation" label="Qui sommes-nous ?"></app-nav-button>

    <app-nav-button v-hailer:navbtn v-hailer:music name="musicAndPhotos" label="Musiques et photos"></app-nav-button>

    <app-nav-button v-hailer:navbtn v-hailer:shows name="shows" label="&#0201vénements"></app-nav-button>

    <app-nav-button v-hailer:navbtn v-hailer:contact  name="contact" label="Nous contacter"></app-nav-button>

    <app-nav-button v-hailer:navbtn v-hailer:friends name="friends" label="Liens amis"></app-nav-button>
</nav>
```

Chaque volet déroulant est qualifié par la directive *hailed*, et peut, de ce fait, être interpellé. Ici les volets déroulants déclenchent la méthode `transit` avec `up` passé en paramètres, lorsqu'ils sont interpellés par un bouton de navigation. Cette méthode permet de faire apparaître le volet avec un effet de translation de bas en haut. De plus, elle permet également de déclencher l'événement `UpE`.

Grâce au système de dialogue entre composants et à la directive *dial*, on a inscrit chaque volet déroulant sur le canal nommé `section`, tel que suit : si un volet déclenche l'événement `UpE`, alors les autres volets du canal `section` déclencheront l'événement `downE`. Or, ces mêmes volets écoutent leur événement interne `downE`, afin de déclencher la méthode `transit(down)` qui permettra à l'élément de transiter vers le bas et de disparaître. Ainsi, lorsqu'un volet apparaît, on s'assure que les autres volets disparaissent. Voici les sections instanciées dans la page home :

```
<app-section
    v-hailed:presentation="['transit', 'up']"
    v-dial:section="['upE','downE']"
    name="presentation"
    header="Qui sommes-nous ?" >
    <app-presentation
        v-bind:route-pres="props.presentation.routePres"
        v-bind:route-members="props.presentation.routeMembers">
    </app-presentation>
</app-section>

<app-section
    v-hailed:contact="['transit', 'up']"
    v-dial:section="['upE','downE']"
    name="contact"
    header="Nous contacter">
    <app-contact>

    </app-contact>
</app-section>

<app-section
    v-hailed:music="['transit', 'up']"
    v-dial:section="['upE','downE']"
    name="musicAndPhotos"
    header="Musiques et photos">

    <app-audio :route="props.audio.routeAudio"></app-audio>

    <app-slider v-bind="props.sliderPhoto"></app-slider>
</app-section>

<app-section
    v-hailed:shows="['transit', 'up']"
    v-dial:section="['upE','downE']"
    name="shows"
    header="&#0201vénements">
    <app-shows
        v-bind:route-shows="props.shows.routeShows"
    >
    </app-shows>

    <app-slider v-bind="props.sliderPoster"></app-slider>
</app-section>

<app-section
    v-hailed:friends="['transit', 'up']"
    v-dial:section="['upE','downE']"
    name="friends"
    header="Liens amis">
    <app-friends v-bind:route-friends="props.friends.routeFriends"></app-friends>
</app-section>
```

Pour résumer, un bouton de navigation interpelle un volet déroulant pour déclencher son apparition (système d'interpellation). En réponse à cette apparition, les autres volets déroulants disparaissent (système de dialogue).

Authentification

Afin de permettre à l'administrateur du site de se connecter au back office, j'ai mis à disposition le formulaire d'authentification préconisé par *Symfony*. Celui-ci est relativement simple à quelques détails près :

- un champ caché permet de transmettre un jeton *CSRF* lors de la soumission du formulaire.
- un second champ caché permet d'indiquer la route vers laquelle rediriger l'utilisateur si l'authentification est validée.
- le formulaire permet l'affichage de messages flash lors d'une erreur d'authentification. Les données des messages flash sont récupérées dans la route permettant de vérifier la tentative de connexion.

```
<h1>Administration</h1>

    <form method="post">

        {% if error %}
        <div class="flash-msg">
            {{ error.messageKey|trans(error.messageData, 'security') }}
        </div>
        {% endif %}

        <label for="inputLogin">Identifiant</label>
        <input type="text" value="{{ last_username }}" name="_username" id="inputLogin"  placeholder="Votre identifiant" required
autofocus>

        <label for="inputPassword">Mot de passe</label>
        <input type="password" name="_password" id="inputPassword" placeholder="Votre mot de passe" required>

        <input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}">

        <input type="hidden" name="_target_path" value="/atable"/>

        <button class="login-btn" type="submit">
            Se connecter
        </button>

    </form>
```

Le fichier de configuration `security.yaml` me permet de définir les modalités d'authentification.

```
security:

    encoders:
        App\Entity\Admin:
            algorithm: bcrypt
            cost: 12

    providers:
        adminProv:
            entity:
                class: App\Entity\Admin
                property: login

    firewalls:

        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js|musics|fonts|webfonts)/
            security: false

        main:
            anonymous: true
            provider: adminProv

            form_login:
                login_path: login
                check_path: login

            logout:
                path:   /logout
                target: home

    access_control:
        - { path: ^/send, roles: ROLE_ADMIN }
        - { path: ^/atable-admin, roles: ROLE_ADMIN }
```

L'`encoder` précise sur quelle entité récupérer le mot de passe, ainsi que l'algorithme utilisé pour encrypter et comparer les mots de passe.

Le provider `adminProv` permet d'indiquer quelle entité *Doctrine* représente l'utilisateur, ainsi que la propriété qui sera utilisée pour l'authentification.

Le pare-feu `dev` permet de désactiver la sécurité sur les ressources publiques du site. Le pare-feu `main` permet de définir la sécurité générale du site. Celui-ci autorise les visiteurs anonymes, afin de laisser le visiteur accèder à la page principale, et définit le provider `adminProv` comme moyen d'identifier les utilisateurs. L'entrée `form_login` définit la route servant le formulaire de connexion et la route testant la connexion. L'entrée `logout` indique le point d'arrêt à utiliser pour la déconnexion, ainsi que la route vers laquelle rediriger l'utilisateur déconnecté.

L'`access control` nous permet de restreindre les autorisations des pare-feux à certains utilisateurs en particulier, en fonction de leur rôle. Nous protégeons donc les routes du back office en restreignant leur accès aux utilisateurs ayant le rôle `ROLE_ADMIN`.

L'écouteur d'événements `Encrypt.php` me permet de crypter le mot de passe lors de l'enregistrement ou de la mise à jour d'un nouvel administrateur en base de données, en écoutant les événements du cycle de vie des entités *Doctrine*.

```php
class Encrypt
{
    private $encoder;

    public function __construct(UserPasswordEncoderInterface $encoder)
    {
        $this->encoder = $encoder;
    }

    public function prePersist(LifecycleEventArgs $args)
    {
        $entity = $args->getObject();

        if($entity instanceof Admin)
        {
            // If new admin
            if(is_null($entity->getId()))
            {
                $this->encrypt($entity);
                return;
            }
        }
    }

    public function preUpdate(LifecycleEventArgs $args)
    {
        $entity = $args->getObject();

        if($entity instanceof Admin)
        {
            if($args->getNewValue('password') !== '""')
            {
                $this->encrypt($entity);
            }
            else
            {
                $entity->setPassword($args->getOldValue('password'));
            }

            if($args->getNewValue('mail_password') === '""')
            {
                $entity->setMailPassword($args->getOldValue('mail_password'));
            }
        }
    }

    private function encrypt($entity)
    {
        $plainPassword = $entity->getPassword();
        $entity->setPassword($this->encoder->encodePassword($entity, $plainPassword));
    }
}
```

Nous récupérons par injection de dépendance une instance `UserPasswordEncoderInterface` afin d'encoder les mots de passe en accord avec la configuration de la sécurité.

La méthode `encrypt` permet de crypter le mot de passe, grâce à l'`encoder`.

La méthode `prePersist` est exécutée avant l'insertion d'un nouvel utilisateur en base de données. Nous vérifions d'abord que l'on manipule une entité de type `Admin`, puis que cet `Admin` soit bien nouveau en vérifiant que l'identifiant soit null. Nous procédons ensuite à l'encryptage.

La méthode `preUpdate` effectue les mêmes traitements, à l'exception près que si l'utilisateur est mis à jour avec un mot de passe vide, alors on garde son ancien mot de passe, sinon on encrypte le nouveau. Avec le mot de passe de l'adresse mail on procède de même, sauf que celui-ci n'est jamais encrypté et est enregistré en dur dans la base - cela est dû aux contraintes de connexion du serveur SMTP.

Back office

Le back office du site a été généré grâce au bundle *EasyAdmin*. Celui-ci permet de créer tous les composants nécessaires au CRUD sur les entités *Doctrine* : les vues, les routes et les formulaires avec contraintes de validation.

La configuration des différents volets du back office se fait dans le fichier `config/packages/easy_admin.yaml`
.
L'entrée `site_name` permet de modifier l'entête de l'interface du back office. Celle-ci permettant d'injecter du *HTML*, je m'en suis servi pour générer une en-tête composée de deux liens. Le premier permet de revenir au site afin de visualiser les modifications effectuées par l'administrateur, le second permet de se déconnecter de l'interface d'administration.

```
site_name: '<a href="%env(DOMAIN)%" style="text-decoration: underline">Retour au site</a> - <a href="%env(DOMAIN)%/logout" style="text-decoration: underline; color: red;">Se déconnecter</a>'
```

L'entrée `design` permet de configurer quelques options esthétiques pour l'interface. J'ai changé la couleur principale du back office, et ai modifié le favicon de l'onglet.

```
design:
        brand_color: '#CE9443'
        assets:
            favicon: './images/favicon.png'
```

L'entrée `entities` permet d'indiquer quelles sont les entités *Doctrine* prises en compte pour la génération des volets de l'interface d'administration. Pour chaque entité l'on peut configurer : le contrôleur gérant la génération du volet d'administration associé, l'intitulé du volet, et une configuration pour chaque action du CRUD.

Nous avons également la possibilité d'interdire certaines actions du CRUD, et de rajouter des liens vers des actions personnalisées, développées dans des routes à part entière. Par exemple, le volet lié à l'entité `Newsletter` propose d'envoyer la newsletter, tandis que le volet de l'entité `Admin` se voit privé des actions `search`, `new` et `delete`.

```
Newsletter:
            class: App\Entity\Newsletter
            label: 'Newsletter'
            list:
                title: 'Newsletter'
                actions:
                    - {label: 'Envoyer', name: 'sendNewsletter', type: 'route'}
                fields:
                    - { property: 'title', label: 'Titre' }
                    - { property: 'content', label: 'Texte' }
                    - { property: 'sent', label: 'Envoyé', type: 'boolean' }
            form:
                fields:
                    - { property: 'title', label: 'Titre' }
                    - { property: 'content', label: 'Texte'}
```

```yaml
Admin:
    controller: App\Controller\Admin\AdminController
    class: App\Entity\Admin
    disabled_actions: ['search', 'new', 'delete']
    label: 'Compte administrateur'
    list:
        title: 'Informations du compte'
        fields:
            - { property: 'login', label: 'Identifiant' }
            - { property: 'mail', label: 'Mail de contact' }
        dql_filter: "entity.login != 'root'"
    form:
        fields:
            - { property: 'login', label: 'Identifiant' }
            - { property: 'mail', label: 'Mail de contact' }
            - { property: 'password', label: 'Mot de passe'}
```

Nous pouvons aussi filtrer préalablement les résultats de recherche grâce à un filtre *DQL*. Dans l'exemple ci-dessus, la liste des administrateurs affichera tous les administrateurs sauf celui dont l'identifiant est `root`.

Serialisation et traduction des dates

Afin d'afficher les dates des événements du groupe, il me fallut auparavant les traduire en français. Pour ce faire, j'ai développé le `DateTranslater`, utilisé en tant que service *Symfony*. L'idée derrière ce service était de conserver les possibilités de formatage offertes par l'objet `DateTime` natif à *Php*, tout en incluant mon propre formatage sur le nom de mois et des jours.

Le `DateTranslater` dispose de deux tableaux associatifs `$days` et `$monthes`, listant les correspondances entre les chaînes de caractères issues du formatage par un `DateTime` et leur traduction en français. La méthode `format` permet, à partir d'une instance de `DateTime` et d'un format, de formater la date tout en traduisant les jours et les mois en français.

```php
class DateTranslater
{
    private $days = [
        "Mon" => "lundi",
        "Tue" => "mardi",
        "Wed" => "mercredi",
        "Thu" => "jeudi",
        "Fri" => "vendredi",
        "Sat" => "samedi",
        "Sun" => "dimanche"
    ];

    private $monthes = [
        "Jan" => "janvier",
        "Feb" => "février",
        "Mar" => "mars",
        "Apr" => "avril",
        "May" => "mai",
        "Jun" => "juin",
        "Jul" => "juillet",
        "Aug" => "août",
        "Sep" => "septembre",
        "Oct" => "octobre",
        "Nov" => "septembre",
        "Dec" => "décembre"
    ];
```

```
    public function format(\DateTimeInterface $date, $format)
    {
        $chars = str_split($format);
        $dateStr = "";
        foreach($chars as $char)
        {
            if($char === "M")
            {
                $dateStr .= $this->monthes[$date->format($char)];
            }
            elseif($char === "D")
            {
                $dateStr .= $this->days[$date->format($char)];
            }
            else
            {
                $dateStr .= $date->format($char);
            }
        }

        return $dateStr;
    }
}
```

Mon interface d'administration affichant par défaut les dates en anglais, cela pouvait s'avérer gênant pour les administrateurs. J'ai donc développé des méthodes supplémentaires dans mes entités, dont le rôle est de retourner les dates en français. Les méthodes `getFrBenginAt` et `getFrEndAt` traduisent respectivement la sortie des getters `getBeginAt` et `getEndAt`.

```
public function getFrBeginAt()
{
    $beginAt = $this->getBeginAt();
    $translater = new DateTranslater();

    $when = $translater->format($beginAt, 'D d M Y');
    $when .= ' à ';
    $when .= $translater->format($beginAt, 'H').'h'.$translater->format($beginAt, 'i');
    return $when;
}
```

Afin de récupérer les dates des événements dans mes composants *Vue* via un appel *AJAX*, il me fallait tout d'abord les formater en *JSON*. La méthode native de *php* `json_encode` permet justement de procéder à ce formatage, dans la mesure où l'algorithme de formatage est précisé via l'interface `JsonSerializable`. J'ai donc implémenté cette interface dans mon entité et ai défini la méthode `jsonSerialize` associée à celle-ci. Cette méthode doit retourner sous forme de tableau associatif les propriétés de l'entité, formatées de la manière souhaitée.

Je n'ai effectué aucun formatage sur la description ou la localisation des événements. Cependant, le site étant destiné à une population francophone, il m'a fallu traduire les dates en français grâce au service `DateTranslater`. De plus, afin de fournir un énoncé cohérent quelque soit la situation, il m'a fallu modifier le formatage des dates selon plusieurs critères :

- l'événement est-il déjà passé ?
- la date de fin de l'événement est-elle précisée ?
- le début et la fin de l'événement se déroulent-ils le même jour ?

```php
public function jsonSerialize()
{
    $translater = new DateTranslater();
    $beginAt = $this->getBeginAt();
    $endAt = $this->getEndAt();
    $now = new \DateTime();
    $when = "";
    $coming;

    if((!is_null($endAt) && $endAt<$now) || (is_null($endAt) && $beginAt<$now))
    {
        $coming = false;

        $when = $translater->format($beginAt, 'M Y');
    }
    else
    {
        $coming= true;
        if(is_null($endAt))
        {
            $when = $translater->format($beginAt, 'D d M Y');
            $when .= ' / ';
            $when .= $translater->format($beginAt, 'H').'h'.$translater->format($beginAt, 'i');
        }
        elseif($endAt->format('D M Y') === $beginAt->format('D M Y'))
        {
            $when = $translater->format($beginAt, 'D d M Y');
            $when .= ' / ';
            $when .= $translater->format($beginAt, 'H').'h'.$translater->format($beginAt, 'i');
            $when .= ' - ';
            $when .= $translater->format($endAt, 'H').'h'.$translater->format($endAt, 'i');
        }
        else
        {
            $when = $translater->format($beginAt, 'D d M Y');
            $when .= ' / ';
            $when .= $translater->format($beginAt, 'H').'h'.$translater->format($beginAt, 'i');
            $when .= ' - ';
            $when .= $translater->format($endAt, 'D d M Y');
            $when .= ' / ';
            $when .= $translater->format($endAt, 'H').'h'.$translater->format($endAt, 'i');
        }
    }


    return [
        "coming" => $coming,
        "when" => $when,
        "where" => $this->getLocation(),
        "description" => $this->getDescription()
    ];
}
```

Newsletter

L'abonnement à la newsletter s'effectue à l'aide d'un formulaire. Lorsque que le bouton 'S'inscrire' est cliqué par l'utilisateur, une requête *AJAX* est envoyée au serveur avec le mail du visiteur désirant s'inscrire. Le format du mail est testé, grâce à une expression régulière, puis on teste la présence d'un mail similaire dans la base de données. Enfin, le nouvel abonné est enregistré dans la base de données si le format convient et qu'il n'était pas déjà inscrit. Un message flash indiquant les erreurs survenues ou confirmant le bon traitement de sa demande est retourné à l'utilisateur.

```php
public function subscribe(Request $req, EntityManagerInterface $em, NewsletterSubscriberRepository $repo)
{
    if($req->isXmlHttpRequest())
    {
        if($req->getMethod() === "POST")
        {
            $datas = json_decode($req->getContent(), true);
            $valid = preg_match('/^.+@.+\..+$/', $datas["mailAdress"]);

            if($valid === 1)
            {
                $sub = $repo->findOneByMail($datas["mailAdress"]);
                if(!is_null($sub))
                {
                    return new JsonResponse(["flash" => [
                        "type" => "error",
                        "msg" => "Vous êtes déjà inscrit."
                    ]]);
                }

                $sub = new NewsletterSubscriber();
                $sub->setMail($datas["mailAdress"]);
                $em->persist($sub);
                $em->flush();

                return new JsonResponse(["flash" => [
                    "type" => "success",
                    "msg" => "Vous êtes désormais inscrit à la newsletter !"
                ]]);
            }
            else
            {
                return new JsonResponse(["flash" => [
                    "type" => "error",
                    "msg" => "Le format de l'adresse mail est invalide."
                ]]);
            }

        }
        else
        {
            return new JsonResponse(["flash" => [
                "type" => "error",
                "msg" => "You can't use this route with "+$req->getMethod()+ " method."
            ]]);
        }
    }
    else
    {
        return redirectToRoute('home');
    }
}
```

Un lien présent dans la newsletter, spécifique à chaque abonné, permet d'accéder à une route procédant à la désinscription. Cette route étant accessible sans authentification, il était nécessaire de recourir à autre chose que les identifiants en base de données pour garantir l'unicité des liens. C'est la raison pour laquelle j'ai généré un code unique et aléatoire dans le constructeur de l'entité, grâce à la fonction `uniqId` native de *php*. Ainsi, les utilisateurs malintentionnés ne peuvent pas désinscrire les abonnés en saisissant des identifiants aléatoires.

```php
public function unsubscribe($code, NewsletterSubscriberRepository $subRepo, EntityManagerInterface $em, PresentationRepository
$presRepo)
{
    $msg="";
    $sub = $subRepo->findOneByCode($code);
    if(!is_null($sub))
    {
        $em->remove($sub);
        $em->flush();
        $msg = "Vous êtes bien désinscrit de notre newsletter !";
    }
    else
    {
        $msg = "La tentative de désinscription a échoué. N'hésitez pas à nous contacter si besoin.";
    }

    $entities = $presRepo->findAll();
    $description = $entities[0]->getDescription();

    return $this->render('newsletter/unsubscribe.html.twig', ["msg" => $msg, 'description' => $description]);
}
```

Une action personnalisée 'envoyer' a été ajoutée au volet d'administration des newsletters. Cette action est paramétrée dans le fichier de configuration d'*EasyAdmin*, et permet d'accéder à la route envoyant la newsletter, lui transmettant l'identifiant de cette dernière.

Le bundle *SwiftMailer* a été utilisé pour gérer les connexions au serveur SMTP et la transmission des données du mail. Celui-ci fournit trois classes permettant de gérer l'envoi de mail.

La classe `Swift_SmtpTransport` permet de créer et configurer la connexion au serveur SMTP responsable de l'envoi du mail. En paramètre de son constructeur sont passés l'hôte et le port. Ces deux données ont été configurées comme variables d'environnement locales de l'application puis injectées dans le contrôleur. Il faut ensuite lui indiquer le mail de l'utilisateur, et le mot de passe correspondant. Ces deux informations sont présentes en base de données et modifiables dans l'interface d'administration.

La classe `Swift_Mailer` permet de créer l'objet responsable de l'envoi des données au serveur SMTP.

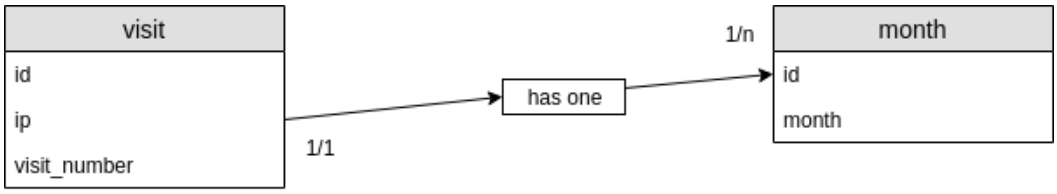
La classe `Swift_Message` permet d'instancier un mail sous forme d'objet. Ces différentes méthodes permettent de préciser l'envoyeur, le sujet du mail, le contenu du mail et les destinataires. Dans notre contexte, nous configurons d'abord l'adresse de l'envoyeur et le sujet du mail, car ceux-ci sont invariables quelque soit le destinataire. Ensuite, pour chaque abonné autorisé à recevoir la newsletter, nous générons la vue à partir d'un template *HTML* qui constitue le corps du mail. Nous envoyons enfin le mail, grâce à l'instance de `Swift_Mailer`. La propriété `sent` de la newsletter est assignée à `true`, afin que l'administrateur sache que sa newsletter a déjà été envoyée.

```php
public function send(Request $req, NewsletterRepository $newsRepo, NewsletterSubscriberRepository $subsRepo, AdminRepository $adminRepo,
EntityManagerInterface $em)
{
    $news = $newsRepo->findOneById($req->query->get('id'));
    $subs = $subsRepo->findAll();
    $admin = $adminRepo->findEffectiveAdmin();

    $transport = new \Swift_SmtpTransport($this->params->get('smtpHost'), (int) $this->params->get('smtpPort'));
    $transport->setUsername($admin->getMail());
    $transport->setPassword($admin->getMailPassword());

    $mailer = new \Swift_Mailer($transport);

    $message = new \Swift_Message();
    $message->setFrom($admin->getMail())
            ->setSubject('Newsletter : '.$news->getTitle());

    foreach($subs as $sub)
    {
        if(!$sub->getAuthorized())
        {
            continue;
        }
        $message->setTo($sub->getMail())
                ->setBody($this->renderView('newsletter/mail.html.twig', ['news' => $news, 'sub' => $sub]), 'text/html');
        $result = $mailer->send($message);

        if($result === 0)
        {
            $this->addFlash('error', "Erreur : la newsletter n'a pas été envoyée à ".$sub->getMail().".");
        }
    }

    $news->setSent(true);
    $em->flush();
    $this->addFlash('success', "La newsletter a bien été envoyée !");


    return $this->redirectToRoute('easyadmin', ["entity" => "Newsletter"]);
}
```

Compteur de visiteurs

L'association m'a demandé de développer une fonctionnalité permettant de visualiser le nombre de visiteurs ayant consultés le site. Pour cela j'ai créé deux entités, `Month` et `Visit`, afin de représenter le mois en cours et les visites.

`Visit` représente l'association entre une adresse Ip, un nombre d'occurence et un mois. Ainsi, il est possible de compter le nombre de visiteurs et de visites par mois, grâce à la relation *oneToMany* qu'entretient la table `visit` avec la table `month`.



Le décompte des visites s'effectue sur la route `home`, c'est-à-dire la route servant la page principale du site.

```php
public function home(VisitManager $vm, Request $request, PresentationRepository $repo)
{
    if(is_null($this->getUser()))
    {
        $ip = $request->getClientIp();
        $vm->registerVisit($ip);
    }

    $entities = $repo->findAll();
    $description = $entities[0]->getDescription();

    return $this->render('home/home.html.twig', ["description" => $description]);
}
```

On vérifie dans un premier temps qu'aucun administrateur n'est connecté, car l'on ne souhaite pas comptabiliser les visites de l'administrateur. On récupère ensuite l'ip du visiteur, puis on procède à la logique du décompte des visites grâce au service `VisitManager` créé dans cette optique.

Le `VisitManager` a besoin des `Repository` des entités `Month` et `Visit` pour récupérer les données dans la base, ainsi que de l'`EntityManager` afin d'enregistrer des données dans la base. On récupère ceux-ci grâce à l'injection de dépendance dans le constructeur.

La méthode `registerVisit` permet d'enregistrer les visites sur le site. On vérifie dans un premier temps que la consultation de la page ne s'effectue pas en local. Ensuite on crée une objet `DateTime` correspondant au premier jour du mois en cours. Si aucun `month` en base de données ne correspond à cette date, alors on crée un nouveau mois ainsi qu'une nouvelle visite qui lui est associée. Si le mois en cours est bien présent en base de donnée, alors on vérifie si il est lié à une visite indiquant la même adresse Ip que la visite ayant déclenchée le processus. Si tel est le cas, on incrémente le nombre d'occurrence des visites de 1, sinon on crée une nouvelle visite.

```php
class VisitManager {

    private $monthRepo;
    private $visitRepo;
    private $em;

    public function __construct(MonthRepository $monthRepo, VisitRepository $visitRepo, EntityManagerInterface $em)
    {
        $this->monthRepo = $monthRepo;
        $this->visitRepo = $visitRepo;
        $this->em = $em;
    }

    public function registerVisit($ip)
    {
        if($ip !== "::1")
        {

            $now = new \DateTime();
            $currentMonth = $now->format('Y').'-'.$now->format('m').'-'.'01';
            $currentMonth = new \DateTime($currentMonth);

            $month = $this->monthRepo->findOneByMonth($currentMonth);

            if(is_null($month))
            {
                $month = new Month();
                $visit = new Visit();
                $visit->setIp($ip);
                $visit->setVisitNumber(1);
                $month->setMonth($currentMonth);
                $month->addVisit($visit);
                $this->em->persist($month);
                $this->em->persist($visit);
                $this->em->flush();
            }
            else
            {
                $visit = $this->visitRepo->findOneByMonthAndIp($month, $ip);

                if(is_null($visit))
                {
                    $visit = new Visit();
                    $visit->setIp($ip);
                    $visit->setVisitNumber(1);
                    $visit->setMonth($month);
                    $this->em->persist($visit);
                    $this->em->flush();
                }
                else
                {
                    $nb = $visit->getVisitNumber();
                    $visit->setVisitNumber($nb + 1);
                    $this->em->flush();
                }
            }
        }
    }
}
```

La recherche de visites pour le mois en cours et pour l'ip en cours d'analyse est effectuée par une requête personnalisée, configurée via le querie builder dans le `Repository` de l'entité `Visit`.

```php
public function findOneByMonthAndIp($month, $ip)
{
    return $this->createQueryBuilder('v')
    ->where('v.ip = :ip')
    ->setParameter('ip', $ip)
    ->andWhere('v.month = :month')
    ->setParameter('month', $month)
    ->getQuery()
    ->getOneOrNullResult();
}
```

La méthode `where` permet de récupérer les éléments de la table selon une condition, basée ici sur le champ `ip`. La méthode `andWhere` permet d'ajouter une condition supplémentaire. Dans ce cas précis, cette seconde condition est faite sur une instance de l'entité `Month`, ce qui résultera en un inner join avec la table `month` dans la requête *SQL* finale. La méthode `setParameter` permet d'insérer une valeur dans la requête à partir d'une variable. Cela est très utile lorsque l'on souhaite utiliser des valeurs saisies par un utilisateur et éviter les injections *SQL*; ce qui n'est néanmoins pas le cas ici. Enfin, la méthode `getQuery` envoie la requête et récupère les résultats de la base de données, et la méthode `getOneOrNullResult` formate le résultat pour le rendre exploitable.

Veille, sécurité et hack de l'API

En fin de projet, je me suis demandé si les routes dédiées aux appels *AJAX* étaient accessibles en inter-domaines. Bien que ces routes ne servent qu'à récupérer des données publiques et à effectuer des actions non sensibles, j'ai pensé qu'il serait préférable que celles-ci ne soient accessibles qu'à partir du domaine courant.

J'ai tenté de faire des appels à l'API depuis un autre domaine, afin de connnaître l'étendu des protections mises en place. Mes tentatives de hack des routes se sont soldées par des échecs grâce à deux éléments de sécurité : la clause de même origine (same-origin policy) de la réponse du serveur, et les requêtes préventives (preflight request) du navigateur.

J'ai entrepris de me renseigner plus avant sur ces deux mesures de sécurité, afin de savoir si elles étaient suffisantes pour interdire les accès inter-domaines. Lors de mes recherches, je me suis servi de la documentation MDN, de StackOverflow ainsi que divers blogs de développeur. Je me suis servi des mots clefs *"same-origin policy", "CORS", preflight request", "protect API from cross-domain", "Does same-origin policy prevent from cross-domain route access".*

La clause de même-origine trouve son origine dans l'en-tête `Access-Control-Allow-Origin` de la réponse du serveur. Celle-ci permet d'indiquer au navigateur les domaines qui peuvent interagir avec lui.

L'utilisation de requêtes préventives permet justement d'empêcher un tel accès. Celles-ci sont des requêtes préconfigurées, et envoyées au serveur avant la requête AJAX initiale. Lorsque la requête préventive est interprétée par le serveur, et si le serveur retourne réponse dotée d'une en-tête `Access-Control-Allow-Origin` adéquat, alors la requête *AJAX* initiale sera bien envoyée.

Pour résumer, la première mesure de sécurité intervenant lors des requêtes *AJAX* inter-domaines est l'envoi d'une requête préventive par le navigateur. La seconde mesure est l'en-tête Access-Control-Allow-Origin retournée par le serveur, et son contenu précise si l'envoi de la requête *AJAX* originale est autorisée. Cette en-tête détermine également si le serveur se comporte selon la clause de même-origine, ou autorise la partage de ressources inter-origine.

Voici un extrait de la documentation MDN explicitant le fonctionnement des requêtes préventives :

https://developer.mozilla.org/en-US/docs/Glossary/Preflight_request

«A CORS preflight request is a CORS request that checks to see if the CORS protocol is understood.

It is an OPTIONS request, using three HTTP request headers: Access-Control-Request-Method, Access-Control-Request-Headers, and the Origin header.

A preflight request is automatically issued by a browser, when needed. In normal cases, front-end developers don't need to craft such requests themselves.

For example, a client might be asking a server if it would allow a DELETE request, before sending a DELETE request, by using a preflight request.

If the server allows it, then it will respond to the preflight request with an Access-Control-Allow-Methods response header, which lists DELETE.»

Traduction :

«Une requête préventive CORS est une requête CORS vérifiant si le protocol CORS est configuré.

C'est une requête utilisant la méthode OPTIONS, utilisant trois en-têtes de requête HTTP : Access-Control-Request-Method, Access-Control-Request-Headers, et l'en-tête Origin.

Une requête préventive est automatiquement générée par le navigateur, si nécessaire. En tant normal, les développeurs front-end n'ont pas besoin de créer de telles requêtes par eux-même.

Par exemple, un client peut demander à un serveur s'il autorise les requêtes avec DELETE avant d'envoyer celles-ci, en utilisant une requête préventive.

Si le serveur l'autorise, alors il répondra à la requête préventive avec l'en-tête de réponse Access-Control-Allow-Methods, qui listera la méthode DELETE.»

Constatant que les sécurités implémentées sur le navigateur étaient suffisantes pour protéger mes routes des appels *AJAX* inter-domaine, j'ai souhaité savoir ce qu'il en était des requêtes ne provenant pas d'un navigateur. En effet, celles-ci ne disposent pas, a priori, des requêtes préventives comme mesure de sécurité, et je souhaitais savoir dans quelle mesure la clause de même-origine pouvait protéger l'accès à mes routes. Voici le test que j'ai mis en place :

Jeu d'essai :

J'ai effectué les tests sur la route gérant l'envoi du mail de contact. Les requêtes sont générées depuis le serveur grâce à *cURL*, un module *php* permettant d'envoyer des requêtes *HTTP*. J'ai fourni à mon script *php* les données nécessaires à la génération d'un mail de contact. J'ai également régler les en-têtes de la requête de manière à passer outre les vérifications sur la nature `XmlHttpRequest` de celle-ci et l'utilisation de la méthode `POST`.

```php
$datas =
[
    'contactMail' => 'abc@def.fr',
    'contactSubject' => 'securityTest',
    'contactContent' => 'lorem ipsum sit dolor amet',
];


$curl = curl_init();
curl_setopt($curl, CURLOPT_URL, 'http://lafringaledetrad.myservertest.fr/contact');
curl_setopt($curl, CURLOPT_POSTFIELDS, json_encode($datas) );
curl_setopt($curl, CURLOPT_ENCODING, 'UTF-8');
curl_setopt($curl, CURLOPT_HTTPHEADER, array('X-Requested-With: XMLHttpRequest'));
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

$output = curl_exec($curl);
curl_close($curl);
```

Données en entrée :

Contenu du mail de contact:

- mail : `abc@def.fr`
- sujet: `'securityTest'`
- content: `'lorem ipsum sit dolor amet'`

Header:

- `'X-Requested-With: XMLHttpRequest'`

Données attendues :

Un mail devrait être reçu sur la boite mail de l'association, affichant les données passées lors de la configuration de la requête.

Données en sortie :

Conforme aux données attendues.

Interprétation des résultats :

Étant donné que la clause de même origine intervient au moment de la génération de la réponse du serveur, et non durant l'exécution du processus de la route, le mail est malgré tout envoyé. Ainsi la clause de même-origine ne protège pas l'accès à une route, elle empêche seulement le client d'accéder au contenu de la réponse du serveur.

Les tests ont révélé que la clause de même-origine était insuffisante, et des ouvertures se sont présentées à moi pour améliorer la sécurité. Voici les mesures envisagées :

- Utilisation d'en-têtes personnalisées : une telle méthode aurait été facile à détourner en étudiant le code source du site sur le navigateur, car j'aurais alors dû paramétrer toutes mes requêtes *AJAX* avec cette en-tête personnalisée.

- Utilisation d'une clé générée à chaque rendu de la page: pour les mêmes raisons que ci-dessus, cette solution n'était pas optimale.

Alors que je recherchais les moyens optimaux de protéger l'accès à mes routes en cross-domain, je suis tombé sur un post du site *StackOverflow*. Le sujet traitait de la sécurité du web dans sa globalité, sans s'attarder sur une technologie en particulier. Les propos d'un utilisateur ont en parti changer ma vision des choses quant à la sécurité du site et à la nécessité de l'améliorer. Les voici tels que je les ai interprétés :

Les mesures de sécurité mise en place sur le web ont pour vocation de dissuader les attaques et de les rendre plus difficiles. Cependant, il est vain de chercher à mettre en place une sécurité absolue, car celle-ci n'existe pas. Si une personne souhaite rassembler suffisamment d'effort et de patience à hacker un site, elle peut réussir à force de tests. Les couches dissuasives de sécurité mise en place sur un site doivent être proportionnelles à la sensibilité des données que fournit celui-ci et à sa notoriété.

Par conséquent, j'ai interrompu là mes recherches en matière de sécurité, estimant que le site que je développais n'avait aucun contenu sensible et qu'il serait trop peu fréquenté pour nécessiter plus de sécurité que n'en prodigue par défaut le navigateur en matière de cross-domain.

Conclusion

Ce projet a rempli une grande partie des attentes que je nourrissais.

Concernant les échanges avec le client, j'ai appris qu'il ne faut pas confondre ses demandes et son besoin. En effet, il arrive parfois que les demandes ne correspondent pas au besoin, et puissent même le desservir.

J'ai pu appréhender cette problématique et tenter d'y répondre lors du maquettage du design de l'application, sans grand succès toutefois. Néanmoins, avec le recul, je pense qu'une attitude plus empreinte de 'confiance professionnelle' et plus de pédagogie m'auraient permis de passer outre les demandes du client pour mieux répondre à son besoin.

J'ai pu m'intéresser aux problématiques liées au développement d'une librairie.

D'une part, il est difficile de concevoir des modules suffisamment génériques pour être intégrés dans une librairie, surtout lorsque le contexte de développement est un projet spécifique.

D'autre part, j'ai dû faire l'impasse sur l'utilisation de dépendances externes pour ma librairie, celles-ci pouvant amener une trop grande complexité lors de l'initialisation d'un nouveau projet. Cependant, ignorer des outils tiers extrêmement performants sur lesquels construire mes modules pourrait s'avérer problématique au long terme - je pense notamment à des librairies telles qu'*animate.js* qui m'auraient éviter de concevoir l'*Interpolator*, en plus de m'apporter des fonctionnalités supplémentaires.

Je sais maintenant qu'il est possible de gérer ses librairies personnelles via un gestionnaire de dépendance tel que *NPM*, ce qui pourra probablement répondre à cette problématique.

J'ai découvert la richesse et la complexité des outils de développement front, et le paramétrage d'une pipeline d'assets, acquérant par là même une certaine maîtrise du framework *Vue*, ainsi qu'une connaissance des bonnes pratiques qui lui sont associées.

Pendant la rédaction de ce dossier, alors que je passais en revu le travail effectué, j'ai constaté que chacun des mes composants devraient être retravaillés en accord avec ces bonnes pratiques. Cela est quelque peu perturbant au premier abord, mais il est également très satisfaisant, en tant que développeur junior, de savoir repérer les maladresses et les mauvaises pratiques dans ses propres réalisations.

J'ai appris à mes dépens qu'une mise en page devait être conceptualisée en profondeur avant le processus de développement, tant pour l'adaptabilité de celle-ci que pour la maintenabilité des feuilles de style. Je suis néanmoins déçu de ne pas avoir réussi à produire une interface attrayante et ergonomique, que j'aurais pu exhiber fièrement au sein de mes productions.

J'ai constaté après la mise en ligne du site et la suppression du fichier robot.txt -qui me permettait d'éviter le référencement en phase de tests - que le site était relativement mal référencé. J'ai appris que le référencement dépendait en grande partie de la taille du contenu du site. Il conviendra donc pour les sites vitrines d'avoir suffisemment de contenu pour assurer un bon référencement.

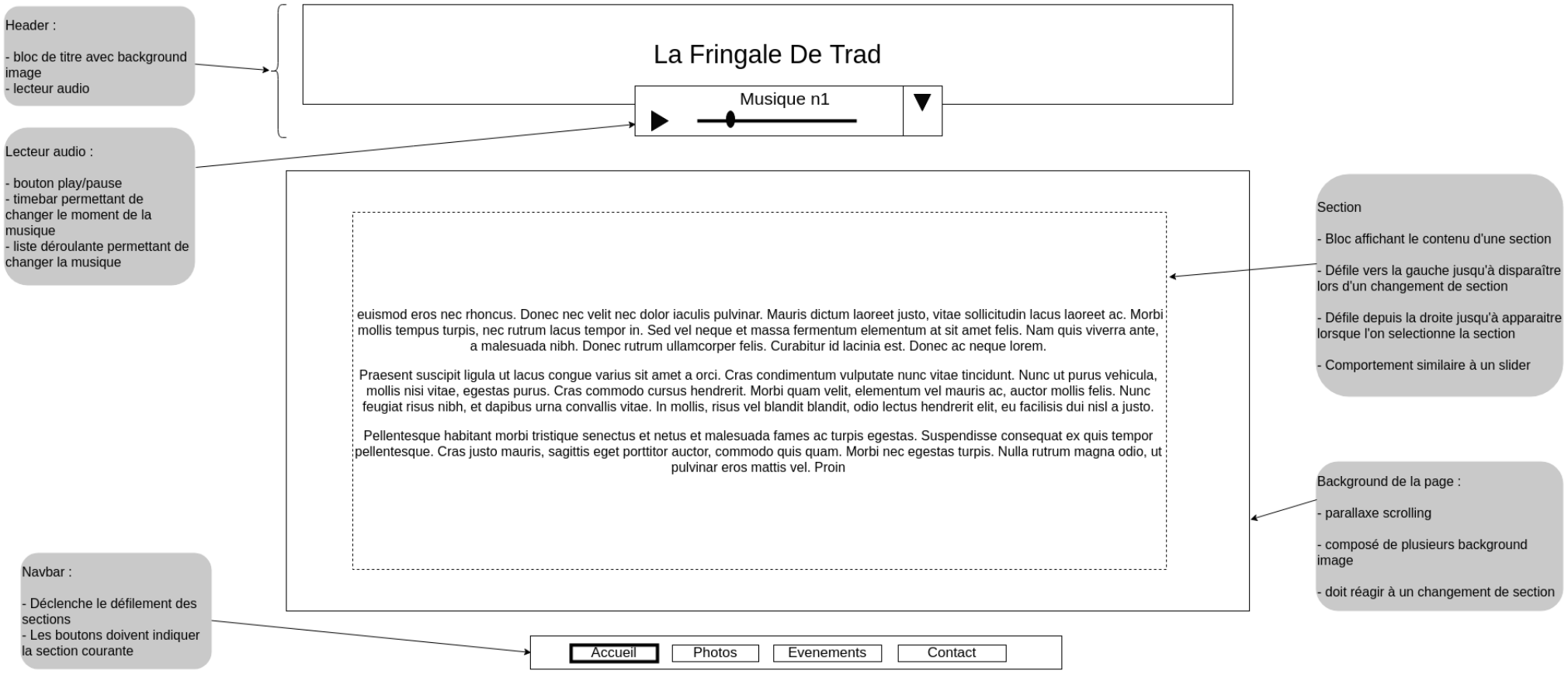
Cependant, j'ai également appris que le contenu récupéré en AJAX n'intervenait pas du tout dans le contenu pris en compte lors du référencement. Afin de surmonté cette problématique deux ouvertures s'offrent à moi :

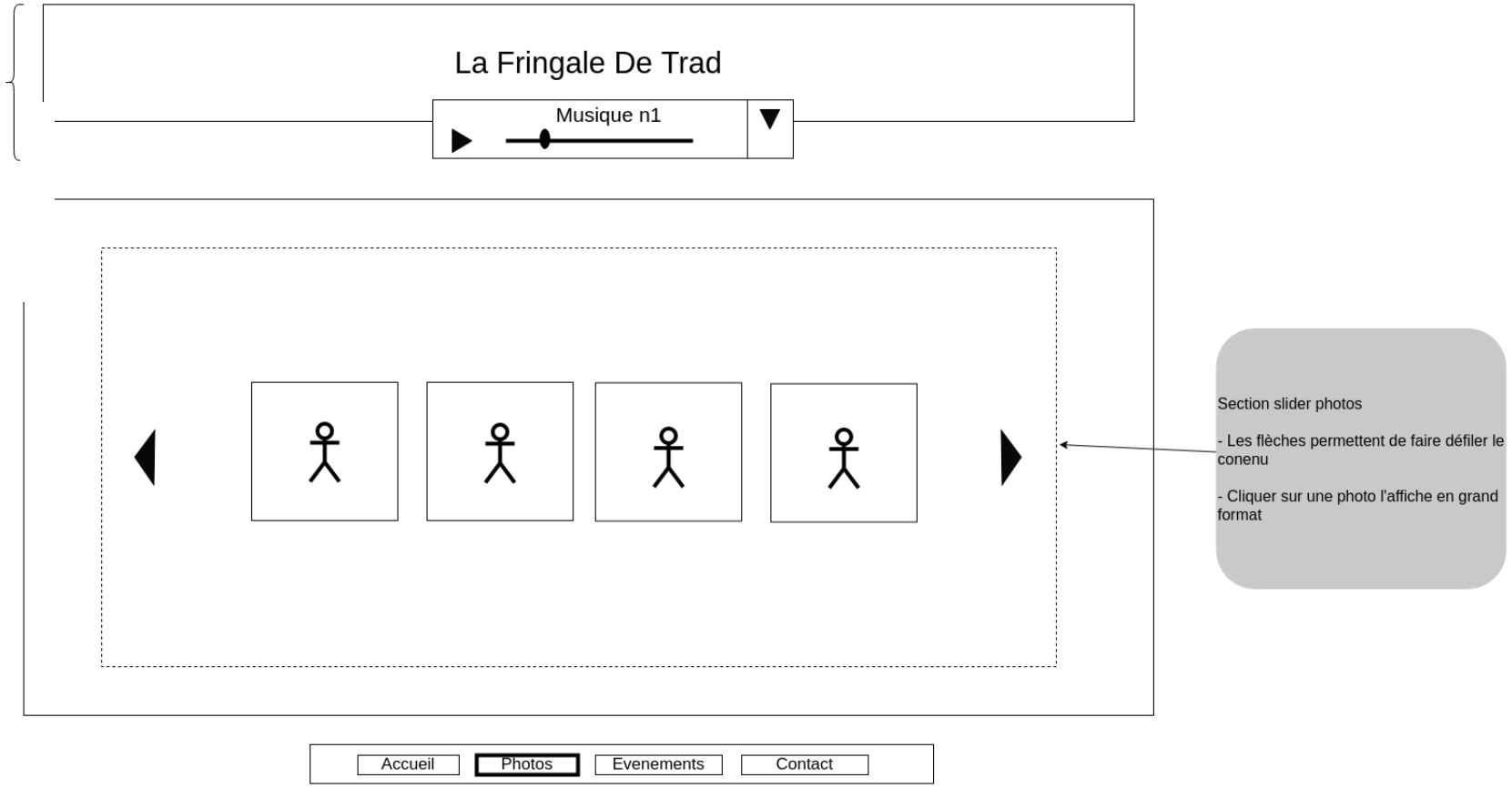
- Utiliser le Server-Side Rendering afin de générer le contenu directement au sein de mes composants.
- Exploiter au mieux le système de `slot` de Vue, afin de passer le contenu généré par Symfony dans le template de mes composants.

A l'issue de ce projet, je dispose de plus d'ouvertures que je ne l'aurais espéré pour continuer d'accroître mes compétences.

Annexes

Annexes - Wireframes




La Fringale De Trad
Musique n1
Section slider photos
- Les flèches permettent de faire défiler le conenu
- Cliquer sur une photo l'affiche en grand format
Accueil
Photos
Evenements
Contact

Administrateurs (`admin`)

Cette table contient les informations liées à l'administration du site.

champ	type	spécificités	commentaires
id	TINYINT	Primary, Not null, Auto-increment, Unsigned	-
login	VARCHAR(64)	Not null	identifiant de connexion
password	VARCHAR(255)	Not null	mot de passe de connexion
mail	VARCHAR(255)	Unique, Not null	adresse mail pour la gestion de la newsletter et des mails de contact
mail_password	VARCHAR(255)	Not null	mot de passe de l'adresse mail

Inscrits à la newsletter (`newsletter_subscriber`)

Cette table contient les informations des personnes inscrites à la newsletter.

champ	type	spécificités	commentaires
id	INT	Primary, Not null, Auto-increment, Unsigned	-
mail	VARCHAR(64)	Unique, Not null	adresse mail de l'inscrit
authorized	BOOLEAN	Not null, default(true)	indique si l'inscrit est autorisé à recevoir la newsletter
code	VARCHAR(255)	Not null	code unique et aléatoire nécessaire à la génération des liens de désincription

Newsletters (`newsletter`)

Cette table contient les informations nécessaires à la génération des nawsletters.

champ	type	spécificités	commentaires
id	INT	Primary, Not null, Auto-increment, Unsigned	-
title	VARCHAR(255)	Nullable	sujet de la newsletter
content	Longtext	Not null	contenu de la newsletter
sent	boolean	Not null, default(false)	indique si la newsletter a été envoyée

Présentation (`presentation`)

Cette table contient les informations nécessaires au rendu de la section `Qui sommes-nous ?` du site.

champ	type	spécificités	commentaires
id	TINYINT	Primary, Not null, Auto-increment, Unsigned	-
text	VARCHAR	Nullable	texte de la présentation
description	VARCHAR	Nullable	texte de la présentation pour les moteurs de recherche
photo1_path	VARCHAR	Nullable	chemin vers la première photo de présentation
photo1_path	VARCHAR	Nullable	chemin vers la seconde photo de présentation
photo1_subtitle	VARCHAR	Nullable	sous-titre de la première photo
photo2_subtitle	VARCHAR	Nullable	sous-titre de la seconde photo

Membres (`member`)

Cette table décrit la composition du groupe de musique.

champ	type	spécificités	commentaires
id	TINYINT	Primary, Not null, Auto-increment, Unsigned	-
name	VARCHAR(64)	Not null	nom du membre du groupe
instruments	VARCHAR(128)	Nullable	instruments du membre du groupe

Photos (`photo`)

Cette table contient le chemin vers les photographies du groupe.

champ	type	spécificités	commentaires
id	INT	Primary, Not null, Auto-increment, Unsigned	-
path	VARCHAR	Not null, Unique	chemin vers la photo à partir du dossier `public`

Posters (`poster`)

Cette table contient le chemin vers les affiches de concert du groupe.

champ	type	spécificités	commentaires
id	INT	Primary, Not null, Auto-increment, Unsigned	-
path	VARCHAR	Not null, Unique	chemin vers l'affiche à partir du dossier `public`

Musiques (`music`)

Cette table contient les informations liées aux resources audios.

champ	type	spécificités	commentaires
id	INT	Primary, Not null, Auto-increment, Unsigned	-
title	VARCHAR(64)	Not null	titre de la piste audio
path	VARCHAR	Not null, Unique	chemin vers la ressource audio à partir du dossier `public`

Evenements (`dance`)

Cette table décrit les événements de l'association.

champ	type	spécificités	commentaires
id	INT	Primary, Not null, Auto-increment, Unsigned	-
begin_at	DATETIME	Not null	moment du début de l'événement
end_at	DATETIME	Nullable	moment de la fin de l'événement
description	VARCHAR(255)	Not null	description de l'événement
location	VARCHAR(128)	Not null	localisation de l'événement

Liens amis (`friend_link`)

Cette table contient les informations nécessaires au rendu de la section `Liens amis` du site.

champ	type	spécificités	commentaires
id	INT	Primary, Not null, Auto-increment, Unsigned	-
name	VARCHAR(64)	Not null	nom de l'ami
content	VARCHAR(255)	Nullable	description des activités de l'ami
link	VARCHAR	Nullable	lien vers le site de l'ami

Visites (`visit`)

Cette table enregistre le nombre de visites effectuées sur le site par une adresse ip pour un mois donné.

champ	type	spécificités	commentaires
id	INT	Primary, Not null, Auto-increment, Unsigned	-
month	ENTITY	Not null	identifiant de l'entité `month`
ip	VARCHAR(45)	Not null	adresse ip du visiteur
visit_number	INT	Not null, default (0)	nombre d'occurence des visites

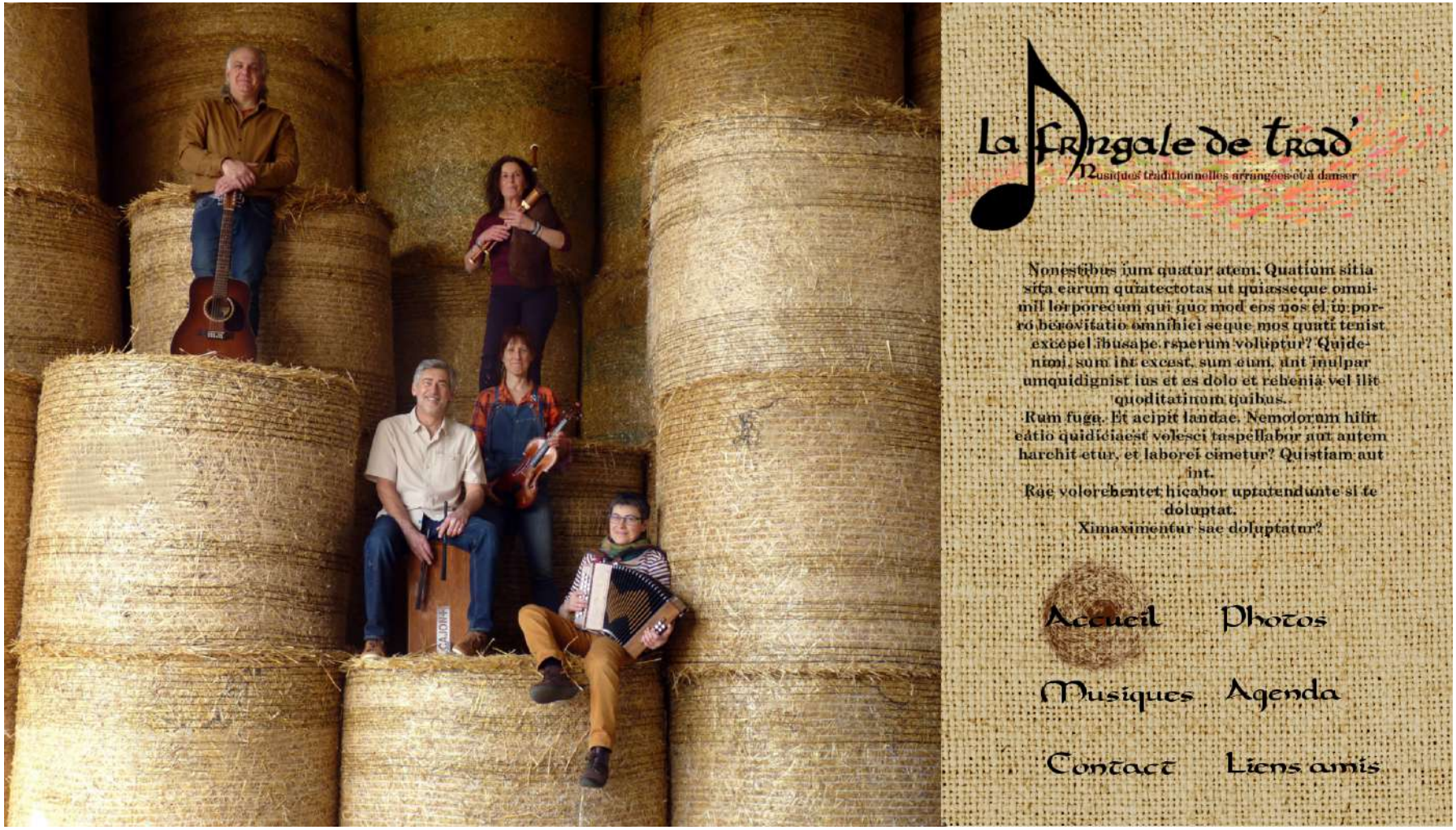
Mois (`month`)

Cette table ccontient différents mois, liés à des visites sur le site.

champ	type	spécificités	commentaires
id	INT	Primary, Not null, Auto-increment, Unsigned	-
month	DATETIME	Not null	date au premier jour du mois
visits	ENTITIES	Not null	référence vers les entités `visit` du mois

Annexes - Maquette (première mouture)

Annexes - Maquette (recommandation au client)



```
const path = require('path');
const webpack = require('webpack');
const MiniCssExtractPlugin = require('mini-css-extract-plugin');
const FixStyleOnlyEntriesPlugin = require("webpack-fix-style-only-entries");
const VueLoaderPlugin = require('vue-loader/lib/plugin');
const OptimizeCssAssetsPlugin = require('optimize-css-assets-webpack-plugin');


var configDefault =
{
    mode: 'development',
    entry:
    {
        "css/style": './resources/style.scss',
        "js/app": './resources/app.js'
    },
    output:
    {
        path: path.resolve('./public/'),
        filename: '[name].js'
    },
    watch: true,
    module:
    {
        rules: [
        {
            test: /\.scss$/,
            use: [
            {
                loader: MiniCssExtractPlugin.loader,
            },
            {
                loader: 'css-loader',
                options: {
                    url: false,
                }
            },
            {
                loader: 'sass-loader',
            }
            ]
        },
        {
            test: /\.vue$/,
            loader: 'vue-loader'
        },
        {
            test: /\.js$/,
            exclude: /(node_modules|bower_components)/,
            use: {
              loader: 'babel-loader',
              options: {
                presets: ['@babel/preset-env']
              }
            }
          }
        ]
    },
    plugins: [
        new FixStyleOnlyEntriesPlugin(),
        new MiniCssExtractPlugin({
            filename: "css/style.css",
          }),
        new VueLoaderPlugin()
    ],
    resolve:
    {
        alias:
        {
            Vue: path.resolve('./node_modules/vue/dist/vue.js'),
            Config: path.resolve('./resources/config.js'),

            TenbuoModules: path.resolve('./resources/tenbuoLib/modules'),
            TenbuoUtils: path.resolve('./resources/tenbuoLib/utils'),
            TenbuoStyle: path.resolve('./resources/tenbuoLib/style'),
            Modules: path.resolve('./resources/modules')
        }
    }
};

var configModifiers =
{
    "prod": function()
    {
        configDefault.mode = "production";
        configDefault.resolve.alias.Vue = path.resolve('./node_modules/vue/dist/vue.min.js');
        configDefault.plugins.push(new OptimizeCssAssetsPlugin({assetNameRegExp: /\.css$/g }));
    }
};

module.exports = function(e)
{
    if(e !== null && e !== undefined)
    {
        configModifiers[e]();
    }
    return configDefault;
}
```

```yaml
easy_admin:

    site_name: '<a href="%env(DOMAIN)%" style="text-decoration: underline">Retour au site</a> - <a href="%env(DOMAIN)%/logout"
style="text-decoration: underline; color: red;">Se déconnecter</a> '

    user:
        display_name: false
        display_avatar: false

    design:
        brand_color: '#CE9443'
        assets:
            favicon: './images/favicon.png'

    formats:
        datetime: 'D d M Y à H:i:s'


    entities:
        Admin:
            controller: App\Controller\Admin\AdminController
            class: App\Entity\Admin
            disabled_actions: ['search', 'new', 'delete']
            label: 'Compte administrateur'
            list:
                title: 'Informations du compte'
                fields:
                    - { property: 'login', label: 'Identifiant' }
                    - { property: 'mail', label: 'Mail de contact' }
                dql_filter: "entity.login != 'root'"
            form:
                fields:
                    - { property: 'login', label: 'Identifiant' }
                    - { property: 'mail', label: 'Mail de contact' }
                    - { property: 'password', label: 'Mot de passe'}

        Membre:
            class: App\Entity\Member
            disabled_actions: ['search']
            label: 'Membres du groupe'
            list:
                title: 'Membres du groupe'
                fields:
                    - { property: 'name', label: 'Nom' }
                    - { property: 'instruments', label: 'Instruments' }
            form:
                fields:
                    - { property: 'name', label: 'Nom' }
                    - { property: 'instruments', label: 'Instruments' }

        Abonne:
            class: App\Entity\NewsletterSubscriber
            label: 'Abonnés à la newsletter'
            list:
                title: 'Abonnées à la newsletter'
                fields:
                    - { property: 'mail', label: "Mail de l'abonné" }
                    - { property: 'authorized', label: 'Autorisé' }
            form:
                fields:
                    - { property: 'mail', type: 'email', label: "Mail de l'abonné" }

        Newsletter:
            class: App\Entity\Newsletter
            label: 'Newsletter'
            list:
                title: 'Newsletter'
                actions:
                    - {label: 'Envoyer', name: 'sendNewsletter', type: 'route'}
                fields:
                    - { property: 'title', label: 'Titre' }
                    - { property: 'content', label: 'Texte' }
                    - { property: 'sent', label: 'Envoyé', type: 'boolean' }
            form:
                fields:
                    - { property: 'title', label: 'Titre' }
                    - { property: 'content', label: 'Texte'}
```

```yaml
Presentation:
    controller: App\Controller\Admin\PresentationController
    class: App\Entity\Presentation
    disabled_actions: ['search', 'new', 'delete']
    label: 'Section présentation'
    list:
        title: 'Informations de la section présentation'
        fields:
            - { property: 'text', label: 'Texte de la présentation',sortable: false }
            - { property: 'photo1_path', label: 'Photo n°1', type: 'image', sortable: false, base_path: './' }
            - { property: 'photo2_path', label: 'Photo n°2', type: 'image', sortable: false, base_path: './' }
            - { property: 'photo1_subtitle', label: 'Photo n°1 - sous-titre', sortable: false }
            - { property: 'photo2_subtitle', label: 'Photo n°2 - sous-titre', sortable: false }
    form:
        fields:
            - { property: 'text', label: 'Texte de la présentation' }
            - { property: 'description', label: 'Description pour les moteurs de recherche' }
            - { property: 'photo1_path', label: 'Photo n°1'}
            - { property: 'photo2_path', label: 'Photo n°2'}
            - { property: 'photo1_subtitle', label: 'Photo n°1 - sous-titre' }
            - { property: 'photo2_subtitle', label: 'Photo n°2 - sous-titre' }

Ami:
    class: App\Entity\FriendLink
    label: 'Liens amis'
    list:
        title: 'Liens amis'
        fields:
            - { property: 'name', label: 'Nom' }
            - { property: 'content', label: 'Description' }
            - { property: 'link', label: 'Lien', type: 'url' }
    form:
        fields:
            - { property: 'name', label: 'Nom' }
            - { property: 'content', label: 'Description' }
            - { property: 'link', label: 'Lien'}

Evenement:
    class: App\Entity\Dance
    label: 'Evénements'
    list:
        title: 'Evénements'
        sort: ['begin_at', 'DESC']
        fields:
            - { property: 'fr_begin_at', label: 'Commence le'}
            - { property: 'fr_end_at', label: 'Se termine le' }
            - { property: 'location', label: 'Se déroule à' }
            - { property: 'description', label: 'Description' }
    form:
        fields:
            - { property: 'begin_at', label: 'Commence le'}
            - { property: 'end_at', label: 'Se termine le' }
            - { property: 'location', label: 'Se déroule à' }
            - { property: 'description', label: 'Description', type_options: {required: false} }

Photo:
    class: App\Entity\Photo
    label: 'Photos'
    disabled_actions: ['edit', 'search']
    list:
        title: 'Photos'
        fields:
            - { property: 'path', label: 'Photo', type: 'image', base_path: './' }
    form:
        fields:
            - { property: 'path', label: 'Photo', type: 'file' }

Affiche:
    class: App\Entity\Poster
    label: 'Affiches'
    disabled_actions: ['edit', 'search']
    list:
        title: 'Affiches'
        fields:
            - { property: 'path', label: 'Affiche', type: 'image', base_path: './' }
    form:
        fields:
            - { property: 'path', label: 'Affiche', type: 'file' }
```

```yaml
Musique:
    controller: App\Controller\Admin\MusicController
    class: App\Entity\Music
    label: 'Musiques'
    list:
        title: 'Musiques'
        fields:
            - { property: 'title', label: 'Titre' }
    form:
        fields:
            - { property: 'title', label: 'Titre' }
            - { property: 'path', label: 'Musique' }
    edit:
        fields:
            - { property: 'title', label: 'Titre' }
            - { property: 'path', label: 'Musique - Laissez vide si inchangé' }

Visites:
    class: App\Entity\Month
    label: 'Visites'
    disabled_actions: ['edit', 'new', 'delete']
    list:
        title: 'Visites'
        fields:
            - { property: 'get_fr_month', label: 'Mois'}
            - { property: 'get_visitor_nb', label: 'Nombre de visiteurs'}
            - { property: 'get_visits_nb', label: 'Nombre de visites'}
```

Remerciements

Je souhaite remercier les membres de l'association La Fringale de Trad' pour leur bienveillance à l'égart de mon travail. Je remercie tout particulièrement Martine Dubien pour sa tutelle et son implication.