DOSSIER DE PROJET

En vue de l'obtention du titre professionnel

Développeur Web et Web Mobile

Présenté par Xavier BOIRY

Ecole O'clock, promotion Kraken 2020

SOMMAIRE

Compétences du référentiel couvertes par le projet

Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

Maquetter une application :
Les wireframes ont été pensés et réalisés dès le début du processus de création. Premier pas essentiel vers l'application, nous avons tenu à les faire en co-working via un partage d'écran. Nous avons utilisé le site Whimsical.com.

Réaliser une interface utilisateur web statique et adaptable :
La capacité responsive du site a été une préoccupation première dans l'intégration de nos idées. Les questions de positionnement des éléments ont ainsi été pensées dès le départ et nous ont ainsi évité de nombreux pièges. Par ailleurs, l'utilisation de la librairie React rend l'adaptabilité plus aisée.

Réaliser une interface utilisateur web dynamique :
La librairie React a été pensée pour rendre l'interface utilisateur fluide et agréable grâce à sa capacité à ne pas recharger les pages malgré les modifications apportées au DOM de façon dynamique.

Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité

Créer une base de données :
Après la réalisation d'un MCD et d'un dictionnaire de données, nous avons pu créer une base de données et y intégrer les tables correspondantes aux besoins de l'application. Pour ce faire nous avons utilisé Doctrine, l'ORM de Symfony.

Développer les composants d'accès aux données :
Symfony et Doctrine constituent un outil puissant pour interagir avec la base de données. Après la déclaration des entités et des repositories dans Symfony, l'ORM se charge de créer le lien avec la base de données.

Développer la partie back-end d'une application web ou web mobile
L'application représente un gros travail de développement dans sa partie back-end. Il a fallu mettre en place l'API afin de communiquer avec le front-end. Un jeton d'identification a été utilisé pour authentifier l'utilisateur et une partie de l'application n'est visible qu'en fonction de son rôle. Les entités profitent également d'un BREAD complet codé dans les contrôleurs.

Résumé du projet

La formation de développeur web et web mobile dispensée par l'école O'clock se déroule en cinq mois dont le dernier est consacré à la réalisation d'un projet en équipe. C'est le mois d'*apothéose* et le projet présenté ici en est le fruit.

Parmi les projets proposés par mes camarades de promo, Memini a particulièrement attiré mon attention par l'aspect onirique de son objectif. C'est donc tout naturellement que j'ai demandé à y participer. Porté par Michèle, ce projet propose de s'envoyer des messages dans le futur. Ce témoignage à travers le temps m'a immédiatement plu.

Pour mener à bien ce projet, nous avons tenté de définir un MVP (Minimum Valuable Product) ni trop ambitieux ni trop modeste. Afin de respecter le délai d'un mois, nous avons dû faire des choix sur les fonctionnalités réalisables et celles plus hasardeuses que nous avons donc laissées de côté en vue d'une version ultérieure.

Ce projet se structure autour de deux technologies principales que sont React pour le front-end et Symfony pour le back-end. La liaison entre les deux est assurée par la mise en place d'une API REST. Issu de la spécialisation Symfony, je me suis surtout occupé de réaliser l'API, de gérer l'authentification et la sécurité ainsi que d'autres fonctionnalités que je détaillerai dans le présent dossier.

Cahier des charges

Présentation du projet.

Memini est une application web qui permet de s'envoyer des messages dans le futur. Il s'agit donc avant tout de fournir à l'utilisateur une expérience onirique et ludique qui peut s'inscrire dans une volonté d'introspection plus sérieuse. Grâce à cette antagonisme, l'application peut s'adresser à n'importe quel public et recèle un caractère universel.

Si Memini est donc un moyen de se recentrer sur soi-même, l'application permet tout de même la possibilité de partager avec la communauté. En effet, une des grandes forces du projet repose dans la liberté qu'il offre de décider du caractère public ou privé des messages envoyés. La page d'accueil présente à tous (y compris l'utilisateur non connecté) les derniers meminis reçus par les membres du site.

Pour cette raison, une modération est obligatoire et un back office est prévu afin que les administrateurs du site puissent éventuellement retirer un memini inadéquat.

Objectifs.

Les objectifs et fonctionnalités retenus pour cette première version de l'application sont :

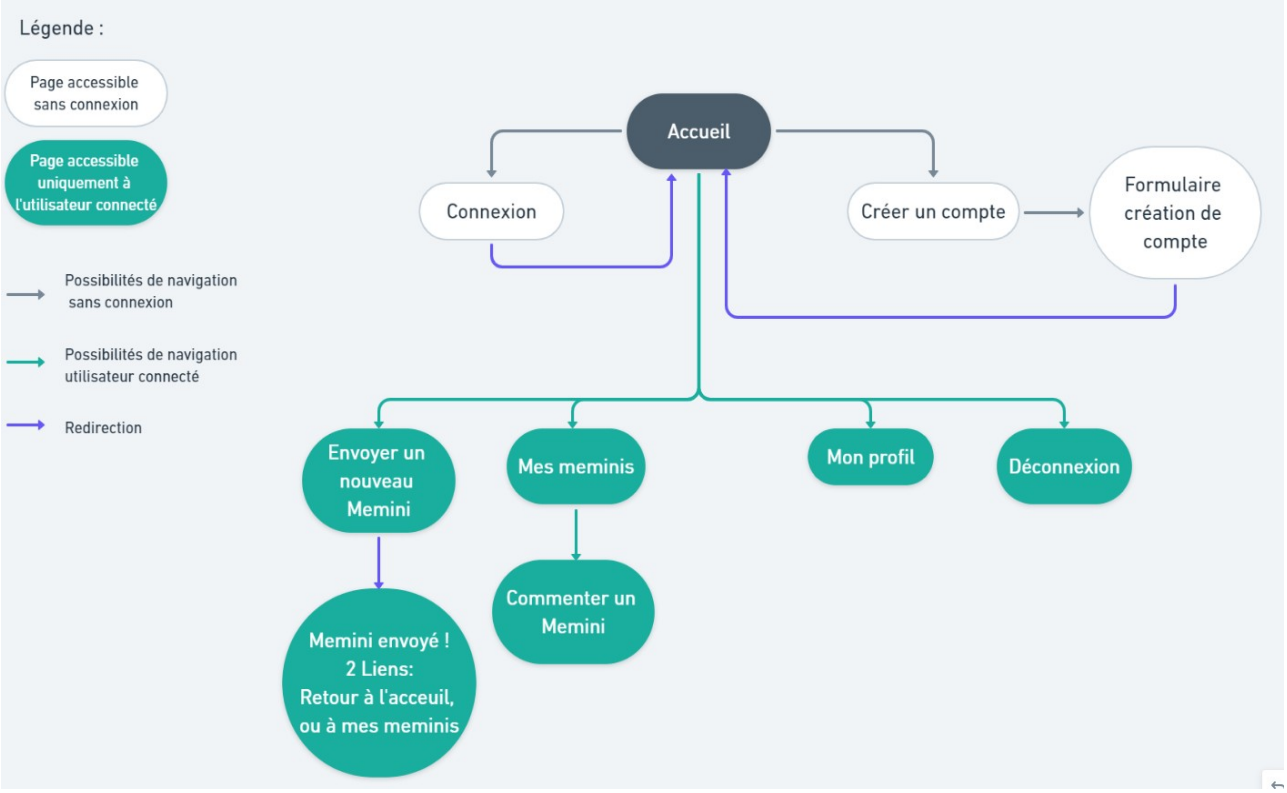
- Pouvoir visualiser les meminis publics sur la page d'accueil.
- Pouvoir s'incrire et se connecter au site.
- Pouvoir écrire un message.
- Pouvoir choisir un tag à attribuer à son message.
- Pouvoir choisir une date de réception du message.
- Pouvoir uploader une image.
- Pouvoir visualiser les messages privés pour un utilisateur connecté.
- Etre notifié par mail de l'arrivée d'un nouveau message.
- Pouvoir réagir à un message en le commentant.
- Pouvoir accéder à mon profil et le modifier.
- Pouvoir modérer les publications via un back office.

Evolutions potentielles.

Les objectifs précédemment cités représente le MVP que l'on s'est imposé afin de respecter le délai imparti. Cependant, le potentiel de l'application appelle d'autres fonctionnalités à développer ultérieurement :

- Donner plus de contenu à son profil utilisateur avec la possibilité d'y ajouter un texte descriptif par exemple.
- Ajouter la possibilité de choisir entre un thème sombre et un thème clair.
- Partager ses messages avec d'autres utilisateurs sans pour autant les rendre visibles de tous.
- Pouvoir consulter le profil d'un autre utilisateur.
- Pouvoir personnaliser le tag d'un message.
- Pouvoir choisir une date précise pour la réception du message.
- Pouvoir joindre une vidéo au message.
- Pouvoir commenter le message d'un autre utilisateur.
- Fonctionnalité mot de passe oublié.

Arborescence de l'application.



User stories.

L'application prévoit trois utilisateurs :

- L'utilisateur non connecté.
- L'utilisateur connecté.
- Le modérateur / administrateur.

Les user stories sont les suivantes :

En tant que...	Je veux...	Afin de...
Visiteur non connecté	Avoir accès à la page d'accueil	Voir les messages publics
Visiteur non connecté	Avoir accès à la page d'inscription	Pouvoir m'inscrire
Visiteur non connecté	Avoir accès à la page de connexion	Pouvoir me connecter
Visiteur connecté	Avoir accès à la page d'accueil	Voir les messages publics
Visiteur connecté	Avoir accès à la page de mes messages	Lire mes messages
Visiteur connecté	Avoir accès à la page de mes messages	Écrire un message
Visiteur connecté	Avoir accès à la page de mon profil	De pouvoir l'éditer
Visiteur connecté	Avoir accès à la page de déconnexion	Me déconnecter
Administrateur	Avoir accès à la page d'accueil	Voir les messages publics
Administrateur	Avoir accès au back office	Modérer les messages publics
Administrateur	Avoir accès à la page de déconnexion	Me déconnecter

Le versionning.

Pour des raisons d'efficacité, nous avons opté pour la méthode *git flow* concernant l'organisation de notre progression durant la réalisation du projet. Cette méthode décrit une façon propre et claire d'avancer à plusieurs sur un même projet en utilisant le logiciel git.

Pour appréhender ce modèle il faut se représenter cinq branches existantes en parallèle :

- main
- hotfix
- release
- develop
- feature

L'organisation opérée autour de ces cinq branche est relativement simple. La branche de travail accueillant le code en cours de développement est la branche *develop*. Chaque nouvelle fonctionnalité doit être développée dans une branche *feature* spécifique. Une fois la fonctionnalité finalisée, on peut la merger sur *develop*. Arrivé au stade où toutes les fonctionnalités souhaitées sont réalisées et rassemblées sur *develop* on peut merger sur *release* pour ensuite seulement merger sur *main.*La branche *hotfix* quant à elle n'est utilisée que lorsqu'un bug est identifié dans la version de production et qu'il est urgent de le régler. On ne fait alors qu'un aller retour entre cette branche et la branche *main.*

Les technologies.

Les deux technologies principales utilisées pour la réalisation de ce projet sont respectivement React pour le front-end et Symfony pour le back-end. Plus précisément nous avons :
Pour le front-end :

- React-Redux
- React-Router
- Axios
- Intégration custom css

Pour le back-end :

- Identification par JWT
- MariaDB
- ORM Doctrine
- Back-office EasyAdmin
- CRON
- Mailer

Réalisation du projet

Présentation de l’équipe.

L’équipe du projet Memini se compose de quatre développeurs juniors :

- Michèle. Elle est derrière l’idée du projet et pour cette raison occupe la place de product owner. Spécialisée dans le front avec la librairie React, elle s’est notamment occupée de programmer l’interface graphique.
- Cyril. L’autre spécialiste de React asssure également le rôle de scrum master afin d’apporter au quotidien la direction à tenir. Il a naturellement codé l’interface aux côtés de Michèle.
- Patrick. Ayant suivi la spécialisation Symfony, il assure le rôle de lead dev back au sein de l’équipe.
- Xavier, moi-même. Egalement issu de la spécialisation Symfony, je me suis essentiellement occupé de la partie back-end. Je tenais également le rôle de git master, m’assurant que le git flow était bien compris et respecté.

Gestion de projet.

Pour mener à bien ce projet et nous organiser au quotidien durant ce mois de développement, nous avons adopté la méthode Scrum. Ainsi commencions-nous chaque matin par un petit debriefing de la veille et un briefing du travail à effectuer dans la journée.

A l’échelle du projet entier, nous avons fait en sorte qu’un sprint dure une semaine. Ainsi le mois pouvait être divisé en quatre sprints dont les objectifs furent les suivants :

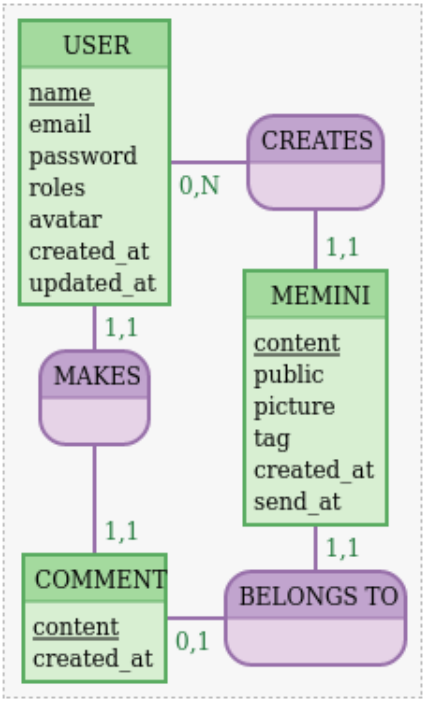
- Sprint 0. Ce fut la période de conception. Aucun code ne fut produit durant cette semaine mais toute la partie conceptuelle fut mise au point. Dans un esprit de groupe, nous avons réfléchi ensemble et réalisé conjointement toute la partie théorique. Nous avons ainsi rédigé le cahier des charges, réalisé les wireframes, le MCD, le dictionnaire de données et également commencé à rédigé notre journal de bord. A l’issu de ce sprint nous étions prêts à nous lancer dans la programmation.
- Sprint 1. Nous avons pu mettre en place les premières fonctionnalités et élaboré la première ébauche de l’API. Nous avons commencé ici à travailler de façon plus isolée ou alors en binôme. Notre travail s’est concentré sur les tâches de base d’après lesquelles nous pourrions élaborer toute la suite du projet.
- Sprint 2. Ce sprint était crucial puisqu’il fallait absolument que l’on ait un minimum viable à son issu. En dépit des problèmes rencontrés, nous avons finalement atteint nos objectifs et le MVP pensé durant le sprint 0 fut atteint.
- Sprint 3. Cette dernière semaine était dédiée aux tests et au debuggage.

Afin de travailler en groupe sans pour autant être en présentiel, nous avons eu recours à plusieurs outils que nous avions pu éprouver tout au long de notre formation :

- Discord. Ce fut le moyen que l'on a le plus utilisé afin de communiquer entre nous grâce à sa possibilité de passer en vocal. Outre cette fonctionnalité essentielle de l'application, un partage d'écran est également possible et nous a été très utile à de multiples reprises. Ce fut aussi un moyen pour nos référents pédagogiques de passer s'assurer que tout se passait bien.
- Trello. Site web indispensable à la méthode Scrum, nous l'avons consulté et mis à jour chaque matin afin d'orienter notre travail.
- Slack. Plateforme d'échange pratique et gratuite, elle nous a suivi tout au long de la formation et nous l'avons utilisé afin d'échanger plus facilement entre nous et accéder à l'entraide que pouvait nous fournir nos collègues.
- Github. Le site complétant le logiciel git nous a été utile afin de gérer notre repository plus efficacement. Mais c'est surtout la section des issues qui nous a été le plus utile pour bénéficier de l'aide nécessaire en cas de blocage.
- Google Drive. C'est grâce à ce service que nous avons notamment pu travailler conjointement durant la phase de conception du projet. En plus de l'espace de stockage, nous avons pu profiter de l'application Google Docs afin de rédiger ensemble les différents documents.

Mes réalisations.

Durant le sprint 0, la tendance était de travailler ensemble et beaucoup de choses ont donc été faites en groupe. Cependant, j'ai tout de même réalisé seul le MCD ainsi qu'une bonne partie des routes de l'API.



URL	NOM	MÉTHODE
/home	home	GET
/api/login_check	connexion	GET
/api/logout	déconnexion	GET
/api/v1/user/browse	user_browse	GET
/api/v1/user/read/{id}	user_read	GET
/user/add	user_add	POST
/api/v1/memini/browse	memini_browse	GET
/api/v1/memini/read/{id}	memini_read	GET
/api/v1/memini/add	memini_add	POST
/api/v1/memini/delete/{id}	memini_delete	DELETE
/api/v1/user/read	user_read	GET
/api/v1/user/edit	user_edit	PATCH
/api/v1/comment/add	comment_add	POST

Une des premières fonctionnalités que nous avons voulu mettre en place au début du sprint 1 a été l'authentification. Pour ce faire, nous avons opté pour le JWT (Json Web Token). Cette technologie fonctionne suivant un principe d'échange entre le front-end et le back-end afin d'authentifier les deux parties l'une à l'autre. Le token est généré côté back à l'aide d'une clé privée. Il est composé de trois parties : un entête, un payload et une signature, ces trois parties étant encodés en base64 et concaténées par des points. Ce système permet une sécurité relative entre les deux parties de l'application et permet à l'utilisateur de s'identifier facilement.

Je me suis chargé de la partie back-end en installant le bundle JWT de Lexik pour Symfony (LexikJWTAuthenticationBundle) disponible sur github. J'ai ensuite tout simplement suivi la documentation fournie. Il m'a fallu générer les clés à l'aide du terminal puis configurer les fichiers security.yaml et routes.yaml.

Après avoir codé la route de connexion à l'API dans Symfony, j'ai pu tester ce bundle JWT. Pour ce faire, j'ai utilisé Insomnia, un logiciel de client API équivalent à Postman. J'ai effectué ce test en entrant des identifiants de connexion valide et en constatant qu'un token JWT m'était bien renvoyé.

Par la suite j'ai eu l'occasion de me familiariser encore un peu plus avec cette technologie puisque j'ai constaté un bug qui fut résolu par la manipulation du token. En effet, j'ai découvert qu'un utilisateur connecté avait accès à tous les meminis, c'est-à-dire aux siens mais également à ceux de tous les autres utilisateurs. Il m'a donc fallu trouver un moyen de retrouver le numéro d'id de l'utilisateur en base de données pour faire correspondre ses informations personnelles et non celles des autres. Plutôt que de faire transiter cette information en clair entre le front et le back, je me suis servi du JWT pour retrouver une information déjà présente et plus sécurisée.

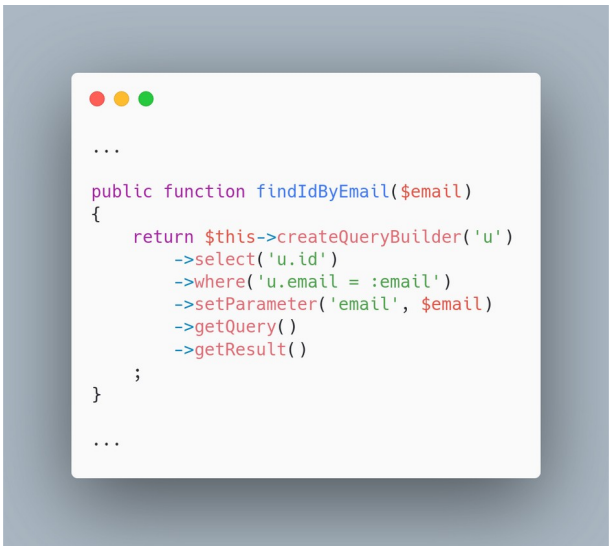
Il m'a donc suffit d'extraire l'information présente dans le payload du JWT afin de l'exploiter. En effet le payload contenait, sinon l'id de l'utilisateur, son email que j'allais alors pouvoir récupérer pour en déduire le numéro d'id par la suite.

```
...

/**
* @Route("browse", name="browse", methods={"GET"})
*/
public function browse(Request $request, MeminiRepository $meminiRepository, UserRepository
$userRepository): Response
{
  $jwt = $request->headers->get('authorization');
  $jwtArray = explode('.', $jwt);
  $tokenPayload = base64_decode($jwtArray[1]);
  $jwtPayload = json_decode($tokenPayload);
  $username = $jwtPayload->username;
  $userId = $userRepository->findIdByEmail($username);
  $meminis = $meminiRepository->findAllPersonalMeminis($userId);
  return $this->json($meminis);
}

...
```

Fig. Ici un extrait du controller dans lequel je manipule le JWT.



```
...

public function findIdByEmail($email)
{
    return $this->createQueryBuilder('u')
        ->select('u.id')
        ->where('u.email = :email')
        ->setParameter('email', $email)
        ->getQuery()
        ->getResult()
    ;
}

...
```

Fig. Ici un extrait du repository dans lequel je retrouve l'id d'après l'email.

Ce sprint 1 fut surtout marqué par la programmation de l'API. Afin de suivre le rythme tenu par le front-end et ne pas pénaliser leurs efforts, l'obligation d'obtenir une API fonctionnelle rapidement se faisait sentir aussi ai-je redoublé d'efforts en ce sens. J'ai adapté mes routes à la nomenclature du BREAD et entamé le codage. Mais j'ai rapidement rencontré une erreur qui m'a fait perdre un peu de temps.

En effet, la route Browse sur l'entité memini provoquait une référence circulaire. Et pour cause : Symfony récupérant toutes les informations sur l'entité suivait naturellement la relation que l'entité possédait avec celle des commentaires, laquelle possédait également une référence à l'entité memini. Afin de résoudre ce bug, j'ai utilisé l'annotation *@Ignore* sur le getter correspondant que j'ai alors remplacé par une méthode custom :

```
...

public function getCommentsList(): ?array
{
    if ($this->comments) {
        $commentsJson = [];
        foreach ($this->comments as $comment) {
            $commentsJson[] = [
                'id' => $comment->getId(),
                'name' => $comment->getName(),
            ];
        }
        return $commentsJson;
    } else {
        return null;
    }
}

/**
 * @Ignore()
 * @return Collection|Comment[]
 */
public function getComments(): Collection
{
    return $this->comments;
}

...
```

Le sprint 2 fut surtout l'occasion de terminer l'API en complétant le BREAD des différentes entités et nous ne rentrerons pas plus en détail ici. Je préfère illustrer ce sprint par une fonctionnalité qui nous tenait à coeur et que j'ai mis en place côté back, à savoir l'upload d'une image pour accompagner l'envoi d'un nouveau message.

Afin de faire transiter l'image du formulaire mis en place en front jusqu'au back, nous avons opté pour l'encodage et le décodage en base64. Grâce à des classes natives de PHP, il m'a alors été facile d'effectuer le bon traîtement et de pouvoir manipuler l'information.

Etant donné que notre application fait appel deux fois à l'upload d'images (celle d'un memini et celle d'un profil), j'ai jugé bon de créer un nouveau service pour traiter cette problématique. Cela évite effectivement d'avoir du code qui se répète d'une classe à l'autre. Toutefois, afin de différencier ces deux types d'image dans notre arborescence, j'ai codé de façon à enregistrer les fichiers dans deux répertoires différents. De plus, une fois les données récupérées, je renomme le fichier selon un identifiant qui lui sera unique et en fonction de son utilité (photo d'avatar ou image de memini). Voici ci-dessous le code complet de ce service :

```php
<?php

namespace App\Service;

use Symfony\Component\Filesystem\Filesystem;

class ImageUploader
{

    public function upload($image, $type)
    {
        $fp = fopen("/var/www/html/projet-memini/app-symfony/public/uploads/" . $type . "s/tmp.png",
"w+");
        $data = explode(',', $image);
        fwrite($fp, base64_decode($data[1]));
        fclose($fp);

        $filesystem = new Filesystem();
        $uniqueID = uniqid();
        $filesystem->rename("/var/www/html/projet-memini/app-symfony/public/uploads/" . $type .
"s/tmp.png", "/var/www/html/projet-memini/app-symfony/public/uploads/" . $type . "s/" . $type . "-" .
$uniqueID . ".png");
        $fileName = "uploads/" . $type . "s/" . $type . "-" . $uniqueID . ".png";
        return $fileName;
    }
}
```

Fig. Le paramètre type *est soit* avatar *soit* picture *afin de nommer et de classer l'upload correctement.*

Veille technologique sur la sécurité.

Pour ce projet nous avons utilisé le Json Web Token comme moyen d'authentification. La solution est séduisante, elle semble légère, souple et sécurisée. Pourtant en creusant un peu le sujet on s'aperçoit qu'un débat accompagne le JWT où qu'on en parle sur internet.

Parmi les critiques qu'on peut trouver, la première concerne le stockage en front dans le localstorage. Si cette option prévient les attaques de type CSRF elle en attire en revanche d'autres puisque le jeton est nécessairement disponible à travers javascript.

L'autre problème concerne l'encodage puisque la plupart du temps un JWT n'est pas chiffré. On peut certes contourner ce problème grâce au protocole HTTPS mais la question de la sécurité se pose quand-même.

Enfin en matière de sécurité, notons qu'il existe toujours un risque qu'une personne malveillante récupère la clé privée d'un serveur si celui-ci est mal configuré ou si la clé est stockée dans un endroit inaproprié.

Parmi les autres inconvénients du JWT, notons les problèmes liés à l'expiration du jeton. Car non seulement il reste valide même après la déconnexion de l'utilisateur mais en plus il n'y a pas de moyen simple pour éviter son expiration non souhaitée, forçant alors l'utilisateur à se reconnecter.

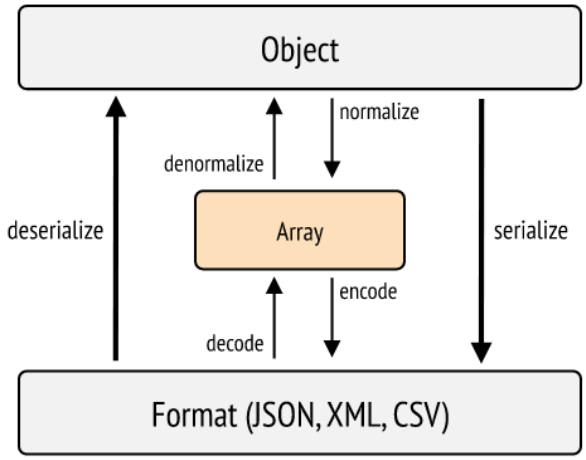
Pour toutes ces raisons j'estime qu'il y a de véritables raisons de se poser la question de l'utilisation d'un JWT. Dans un premier temps dans le cadre de notre projet, ce fut un outil parfait. Mais sur le long terme si l'aventure continue peut-être faudra-t-il remettre tout ceci en question.

Recherche à partir d’un site anglophone, extrait et traduction.

Afin d’illustrer ce chapitre je vous propose un extrait de la documentation de Symfony sur le serializer. Ce fut une ressource très utile pour ma culture métier.

> *The Serializer component is meant to be used to turn objects into a specific format (XML, JSON, YAML, ...) and the other way around.*

In order to do so, the Serializer component follows the following schema.



As you can see in the picture above, an array is used as an intermediary between objects and serialized contents. This way, encoders will only deal with turning specific **formats** into **arrays** and vice versa. The same way, Normalizers will deal with turning specific **objects** into **arrays** and vice versa.

Serialization is a complex topic. This component may not cover all your use cases out of the box, but it can be useful for developing tools to serialize and deserialize your objects.

Installation ¶

```
$ composer require symfony/serializer
```

If you install this component outside of a Symfony application, you must require the `vendor/autoload.php` file in your code to enable the class autoloading mechanism provided by Composer. Read this article for more details.

To use the `ObjectNormalizer`, the PropertyAccess component must also be installed.

Usage ¶

This article explains the philosophy of the Serializer and gets you familiar with the concepts of normalizers and encoders. The code examples assume that you use the Serializer as an independent component. If you are using the Serializer in a Symfony application, read How to Use the Serializer after you finish this article.

Traduction par mes soins :

Le composant Serializer est pensé pour être utilisé pour changer les objets dans un format spécifique (XML, JSON, YAML, ...) et sa réciproque.

Pour ce faire, le composant Serializer suit le schema suivant.

(...)

Comme vous pouvez le voir sur l'image ci-dessus, un tableau est utilisé comme un intermédiaire entre les objets le contenu serialisé. De cette manière, les encoders n'auront qu'à s'occuper de changer des formats spécifiques en tableau et vice versa. De la même façon, les normaliseurs s'occuperont de changer les objets spécifiques en tableaux et vice versa.

La sérialisation est un sujet complexe. Ce composant ne couvrira pas forcément tous vos cas d'usage sans souci, mais il peut être utile pour développer des outils afin de sérialiser et désérialiser vos objets.

Installation.

(...)

Si vous installez le composant en dehors de l'application Symfony, vous devez inclure le fichier vendor/autoload.php dans votre code pour activer le mécanisme d'auto chargement de classe fourni par Composer. Lire cet article pour plus de détails.

Pour utiliser l'ObjectNormalizer, le composant PropertyAccess doit également être installé.

Usage.

Cet article explique la philosophie du Serializer et vous rendra familier des concepts des normalisers et des encodeurs. Les exemples de code suppose que vous utilisez le Serializer comme un composant indépendant. Si vous utilisez le Serializer dans une application Symfony, lisez Comment utiliser le Serializer une fois cet article fini.

Conclusion.

Ce mois d’apothéose fut intense à bien des égards. Nous avons travaillé sans relâche afin de livrer le meilleur de nous-mêmes et de produire une application web la plus riche possible dans le délai imparti. J’estime personnellement que le jeu en valait largement la chandelle au vu du résultat.

Ce mois d’apothéose fut également une source d’apprentissage inattendue. Nous avons pu apprécier à quel point c’est en développant que l’on devient développeur et j’ai le sentiment d’avoir réellement fini par comprendre tout ce que j’ai appris durant les quatre mois de formation précédant cette aventure.

Enfin ce mois d’apothéose fut une belle aventure humaine où l’on apprend à travailler et à avancer en équipe autour d’un projet que l’on porte au jour le jour. Je ressens une énorme gratitude envers mes collègues d’un mois, tous mes collègues de formation et bien sûr O’clock.