



Site Internet du Club

Jean-Philippe GURAK

Table des matières

| | |
|---|----|
| A propos de l'auteur | 4 |
| Elan Formation..... | 5 |
| Titre professionnel de niveau III Développeur web, web mobile | 6 |
| Cahier des charges :..... | 7 |
| A. L'association..... | 7 |
| B. Expression des besoins | 7 |
| C. Environnement juridique et contrainte spécifique du projet | 8 |
| 1. Définition du RGPD | 8 |
| 2. Gestion des cookies | 9 |
| 3. Contrainte du projet | 11 |
| Méthodologie de travail | 12 |
| Méthode Agile..... | 12 |
| 1. Méthode MoSCoW | 12 |
| 2. Trello | 13 |
| Outils utilisés :..... | 14 |
| A. Langages informatiques..... | 14 |
| B. Environnement de développement :..... | 15 |
| C. Framework | 17 |
| Développement | 18 |
| A. Modélisation des données | 18 |
| 1. Le niveau conceptuel :..... | 19 |
| 2. Le niveau logique :..... | 19 |
| B. Arborescence du site | 20 |
| C. Maquettage du site | 20 |
| D. Le responsive et les médias queries..... | 21 |
| E. Le référencement..... | 21 |
| Symfony 5.3 comme Framework | 22 |
| A. Le design pattern..... | 22 |
| 1. Définition :..... | 22 |
| 2. Avantages :..... | 22 |
| B. Le MVC de Symfony | 22 |
| 1. Schéma :..... | 23 |
| 2. Utilisation : | 24 |

| | |
|---|----|
| C. Gestion des utilisateurs | 26 |
| 1. Utilisateur | 26 |
| 2. Inscription..... | 26 |
| 3. Authentification..... | 27 |
| 4. Autorisation | 27 |
| Comment Symfony se prémunit des principales failles de sécurité | 29 |
| A. La faille XSS | 29 |
| B. L'attaque CSRF..... | 30 |
| C. L'injection SQL..... | 30 |
| Doctrine ORM de Symfony..... | 32 |
| Système de covoiturage | 34 |
| 1. Fonction pour les covoiturages à venir : | 34 |
| 2. Fonction d'ajout d'un covoiturage | 35 |
| 3. Fonctions d'ajout/suppression de participants : | 36 |
| 4. Fonction d'affichage de ses propres covoiturages :..... | 37 |
| 5. Suppression en cascade des covoiturages :..... | 38 |
| Utilisation de l'API Leaflet | 39 |
| Système de messagerie privée | 42 |
| Système de formulaire de contact | 46 |
| A. Création d'un formulaire parent..... | 46 |
| B. Création d'un formulaire classique | 47 |
| C. Ecouteurs d'évènements | 48 |
| D. Envoi du formulaire | 49 |
| Gestion des données personnelles | 50 |
| Bibliographie | 55 |
| Partie traduction | 56 |
| Remerciements..... | 61 |
| Axes d'amélioration..... | 62 |
| Annexes..... | 63 |
| A. Contact avec la CNIL | 63 |
| B. Trello | 65 |
| C. MCD | 66 |
| D. MLD..... | 67 |
| E. Arborescence du site..... | 68 |

| | |
|---|----|
| F. Maquettage | 69 |
| 1. Desktop accueil visiteur | 69 |
| 2. Desktop prochains match | 69 |
| 3. Desktop footer | 70 |
| 4. Mobile | 70 |
| G. Cartographie..... | 71 |
| 1. Mise en place de la cartographie : | 71 |
| 2. Mise en place des marqueurs et de l'adresse :..... | 72 |
| 3. Création du champ adresse et de la requête URL : | 73 |

A propos de l'auteur

Jean-Philippe Gurak 46ans marié et père de 2 enfants.

Actuellement en réorientation professionnelle après une vingtaine d'années comme militaire en Allemagne, j'ai choisi le développement web comme porte d'entrée aux métiers du numérique. En effet après une remise à niveau chez Elan Formation j'ai continué par cette formation qui me prépare au **Titre Professionnel développeur web et web mobile** et qui était la suite logique pour qui souhaite accéder au monde du web. Je prépare un **titre RNCP de niveau III** correspondant chez Elan Formation Colmar.

Elan Formation

“La formation sur mesure !”

Telle est leur devise, Elan est un centre de formation fort de 25 années d’expérience en intra ou inter dans les domaines de la bureautique, l’informatique, la PAO, le multimédia et techniques de secrétariat.

Il propose avant tout de la formation individualisée et sur mesure. Il dispose de locaux à Strasbourg, Sélestat, Haguenau, Saverne, Colmar, Mulhouse, Metz et Nancy.

Leurs méthodes pédagogiques

- Ecouter et comprendre la demande précise de l’entreprise ou du stagiaire.
- Adapter une formation qui prenne en compte la singularité de l’apprenant.
- Guider le stagiaire en permanence grâce à un formateur et anticiper ses attentes.
- Suivre l’évolution des acquis tout en avançant à son rythme.
- Valoriser la formation et certifier les compétences acquises.
- Valider la pertinence de notre formation.



Titre professionnel de niveau III

Développeur web, web mobile

Les compétences à acquérir afin de prétendre à l'obtention du Titre professionnel de niveau III Développeur Web et Web Mobile présentées ci-dessous sont extraites du REAC (Référentiels Emploi Activités Compétences) Développeur Web et Web Mobile.

Vue synoptique de l'emploi-type

| N° Fiche AT | Activités types | N° Fiche CP | Compétences professionnelles |
|-------------------|---|-------------------|--|
| 1 | Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité | 1 | Maquetter une application |
| | | 2 | Réaliser une interface utilisateur web statique et adaptable |
| | | 3 | Développer une interface utilisateur web dynamique |
| | | 4 | Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce |
| 2 | Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité | 5 | Créer une base de données |
| | | 6 | Développer les composants d'accès aux données |
| | | 7 | Développer la partie back-end d'une application web ou web mobile |
| | | 8 | Elaborer et mettre en oeuvre des composants dans une application de gestion de contenu ou e-commerce |

Cahier des charges :

A. L'association

Le **Handball Club de Horbourg-Wihr** crée en 1975 est une association sportive à caractère familial proche de Colmar. Composé de 11 équipes réparties en 8 catégories, le club compte cette année 129 licenciés avec souvent parents et enfants. Etant membre de ce club depuis 27 ans, d'abord comme joueur puis maintenant comme président, j'ai décidé en concertation avec le bureau directeur du club la création d'un site internet.

B. Expression des besoins

Le site a besoin d'être la vitrine d'information du club. Pour cela j'ai listé les fonctionnalités essentielles et le cahier des charges suivant dans le cadre du **MVP** (minimum viable product).

Le visiteur pourra :

- Voir les dernières photos des événements récents
- Avoir accès aux résultats de toutes les équipes
- Voir les prochains matchs à la une
- Contacter le club
- Voir le post récent du club sur Facebook

Le licencié pourra en plus du visiteur :

- Se connecter/déconnecter
- Avoir accès à son profil, modifier/supprimer son profil, modifier son mot de passe, réinitialiser son mot de passe.
- Avoir accès à la liste des matchs de toutes les équipes ou d'une équipe en particulier.
- Avoir accès à la liste des covoiturages ou d'un covoiturage en particulier
- Créer/supprimer un covoiturage, s'inscrire/se désinscrire d'un un covoiturage, voir la liste de tous ses covoiturages, voir sur une carte l'emplacement du lieu de rendez-vous du covoiturage.
- Accéder aux informations des clubs adverses ainsi que leur localisation sur une carte.
- Accéder à une messagerie privée, écrire et recevoir des messages des autres licenciés.

L'administrateur du site pourra en plus du licencié :

- Créer/modifier un licencié à partir des éléments demandés par la Fédération Française de Handball.
- Ajouter/modifier/supprimer
 - Des matchs
 - Des clubs
 - Des équipes
 - Des weekends

C. Environnement juridique et contrainte spécifique du projet



1. Définition du RGPD

Le règlement général sur la protection des données le **RGPD** s'applique à toute organisation, **publique et privée, qui traite des données personnelles pour son compte ou non, dès lors** :

- qu'elle **est établie sur le territoire de l'Union européenne**,
- ou que son activité cible directement des **résidents européens**.

Par exemple, une société établie en France, qui exporte l'ensemble de ses produits au Maroc pour ses clients moyen-orientaux doit respecter le RGPD.

De même, une société établie en Chine, proposant un site de e-commerce en français livrant des produits en France doit respecter le RGPD.

Le RGPD **concerne aussi les sous-traitants** qui traitent des données personnelles pour le compte d'autres organismes.

Ainsi, si vous traitez ou collectez des données pour le compte d'une autre entité (entreprise, collectivité, association), vous avez des obligations spécifiques pour garantir la protection des données qui vous sont confiées.

Les 8 principes de la protection des données

- **Minimisation des données :**
 - demandé uniquement les informations essentielles au traitement de la licence
- **Temporalité des données :**
 - Le traitement des données n'intervient que le temps ou l'utilisateur de l'application est affilié au club
- **Finalité du traitement :**
 - La finalité est déterminée et explicite dans les conditions générales d'utilisation
- **Licéité du traitement :**
 - Le licencié a consenti de manière libre et éclairé au traitement de ses données personnelles sur le site internet (quand il signe...)
- **Obligation de sûreté :**
 - Les mots de passe en base de données sont hachés. Seul l'administrateur du site à un accès aux données.
 - D'autres mesures de sécurité ont été mise en application sur le site internet
- **Droit des personnes :**
 - Le droit à la portabilité
 - Le droit à l'oubli, un licencié a le droit de supprimer son compte et toutes les données personnelles le concernant seront supprimées
 - Le droit d'accès, l'utilisateur a accès à tout moment à ses informations personnelles et pourra les modifier.
- **Transparence :**
 - Le licencié dispose du rappel de ses droits dans les **CGU** ainsi que la finalité du traitement de ses données personnelles.
- **Protection particulière de certaines données :**
 - Mot de passe haché

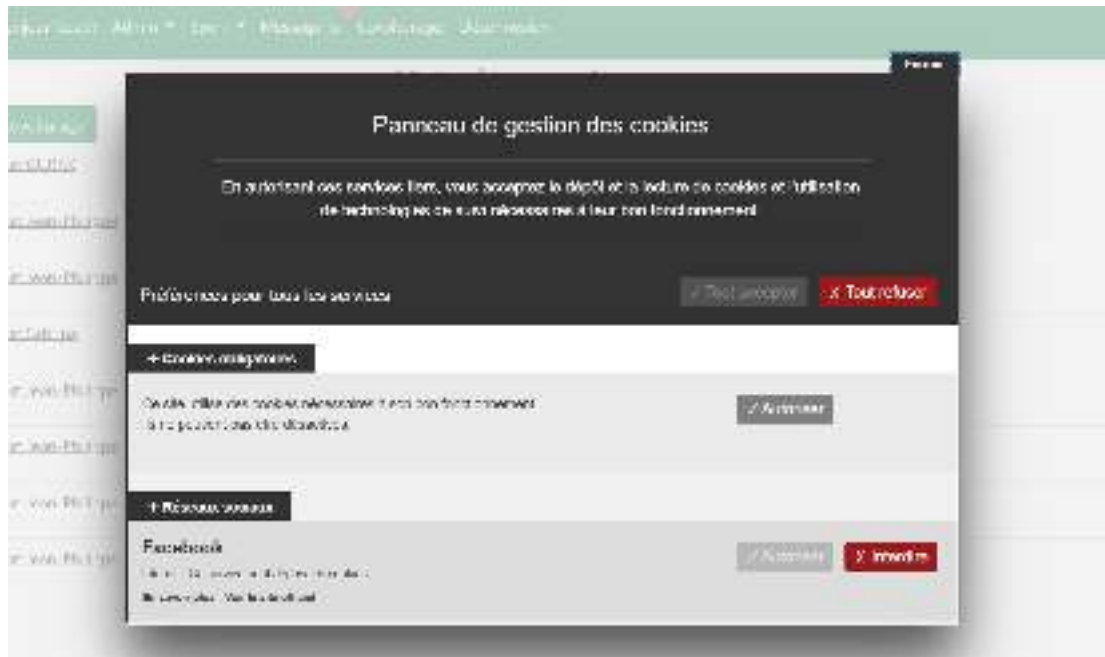
2. Gestion des cookies

Un cookie est un petit fichier stocké par un serveur dans le terminal (ordinateur, téléphone, etc.) d'un utilisateur et associé à un domaine web (c'est à dire dans la majorité des cas à l'ensemble des pages d'un même site web). Ce fichier est automatiquement renvoyé lors de contacts ultérieurs avec le même domaine.

J'ai choisi de m'appuyer sur l'expertise du gestionnaire de cookies TARTE AU CITRON.

L'outil est compatible avec les principales recommandations de la **CNIL** :

- Recueil des consentements avec désactivation sélectionnées par défaut,
- Bandeau offrant les possibilités de tout accepter, tout refuser, ou de personnaliser,
- Affichage des finalités des cookies, et paramétrage de ces cookies par le visiteur,
- Durée de conservation limitée du consentement à 13 mois,
- Interdiction du dépôt de cookie avant tout consentement (blocage des scripts par défaut)



En l'espèce dans mon projet

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript" src="[[ asset('tarteaucitron/tarteaucitron.js') ]]"></script>
    <script type="text/javascript">
      tarteaucitron.init({
        "privacyUrl": "", /* Privacy policy url */
        "hashtag": "#tarteaucitron", /* Open the panel with this hashtag */
        "cookieName": "tarteaucitron", /* Cookie name */
        "orientation": "middle", /* Banner position (top - bottom) */
        "groupServices": false, /* Group services by category */
        "showAlertBanner": false, /* Show the small banner on bottom right */
        "cookieslist": false, /* Show the cookie list */
      });
    </script>
  </head>
  <body>
    <!-- Content -->
  </body>
</html>
```

On déclare immédiatement après l'ouverture de la balise `<head>` le script propre à tarteaucitron qui fait partie d'un package que l'on aura préalablement enregistré dans le projet. Dans ce package se trouve tous les éléments nécessaires au fonctionnement de tarteaucitron, comme les fichiers JS, CSS, Github entre autres.

Le principe de fonctionnement de TARTE AU CITRON est de remplacer le script propre au site d'origine par celui de tarteaucitron comme ci-dessous :

Script Facebook

```
<script async defer crossorigin="anonymous"
src="https://connect.facebook.net/fr_FR/sdk.js#xfbml=1&version=v12.0"
nonce="4zim1KjS"></script>
```

Script tarteaucitron

```
<script type="text/javascript"> (tarteaucitron.job = tarteaucitron.job ||
[]).push('facebooklikebox'); </script>
```

3. Contrainte du projet

La **Commission Nationale de l'Informatique et des Libertés (CNIL)** a été créée par la loi Informatique et Libertés du 6 janvier 1978.

Elle est chargée de veiller à la **protection des données personnelles** contenues dans les fichiers et traitements informatiques ou papiers, aussi bien publics que privés. Ainsi, elle est chargée de veiller à ce que l'informatique soit au service du citoyen et qu'elle ne porte atteinte ni à l'identité humaine, ni aux droits de l'homme, ni à la vie privée, ni aux libertés individuelles ou publiques.

La CNIL est une **autorité administrative indépendante (AAI)**, c'est-à-dire un organisme public qui agit au nom de l'Etat, sans être placé sous l'autorité du gouvernement ou d'un ministre. Elle est composée de 18 membres élus ou nommés et s'appuie sur des services.

Elle a un rôle d'alerte, de conseil et d'information vers tous les publics mais dispose également d'un pouvoir de contrôle et de sanction.

Dès le départ j'ai contacté la **CNIL** pour demander conseil quant au **RGPD** dans mon projet.

⇒ Voir annexe [Contact avec la CNIL](#)

Méthodologie de travail

Après avoir fait le constat des besoins et des spécificités, j'ai décidé d'utiliser la méthodologie de travail suivante.

Méthode Agile

Les méthodologies de gestion de projet dites méthodes agiles ont toutes un point commun : elles sont directement inspirées du **Manifeste Agile** édité en 2001 par des développeurs de logiciels bien décidés à améliorer leur process et à réduire leur taux d'échec. De là sont donc nées progressivement diverses méthodes unies par une nouvelle manière d'aborder le développement de produit en remplaçant, notamment, le client au cœur de l'action et en prônant l'adaptation des procédés de création au fil de l'évolution du projet. Elles se sont depuis imposées comme des standards sur le terrain du développement d'application.

Une méthodologie **Agile**, quelle qu'elle soit, prévoit le fractionnement des étapes de développement logiciel. Contrairement à la méthode traditionnelle qui prévoit la planification totale du projet avant même son développement, le Manifeste Agile préconise plutôt la fixation d'objectifs à court terme. Le projet est ainsi fragmenté en plusieurs sous-parties que l'équipe qui en a la charge se doit d'atteindre progressivement en réajustant si nécessaire les objectifs pour répondre le plus possible aux attentes du client. Les méthodes agiles mettent un point d'honneur à renforcer les relations entre les membres de l'équipe projet, mais également entre l'équipe et le client. C'est pour cette raison que la flexibilité et la souplesse dans l'organisation sont deux piliers fondamentaux des méthodes agile.

Durant la réalisation de l'application, nous nous réunissions le comité directeur du club et moi-même pour faire un point sur l'évolution du projet.

1. Méthode MoSCoW

La méthode MoSCoW est une technique de priorisation qui permettra de prioriser les besoins et les exigences d'un projet souvent informatique.

Ce mot qui fait référence à la capitale russe est un acronyme des mots suivants :

- **M** pour **must have** ; ce qui doit être fait
- **S** pour **should have this if at all possible** ; ce qui serait bien d'être fait si c'est possible
- **C** pour **could have this if it does not affect anything else** ; ce qu'il faudrait faire s'il n'y a pas d'impact avec d'autres demandes
- **W** pour **won't have this time but would like in the future** ; à faire si nous avons le temps un jour.

Dans le projet, la priorisation a mis en évidence le MVP (minimum viable product) souhaité et souhaitable pour que le site puisse à minima être fonctionnel et utilisable. Il est apparu que la demande initiale devait être complétée par un système de messagerie privée pour pouvoir interagir entre membre en ce qui concerne les covoiturages en premier lieu.

2. Trello

Trello est un outil collaboratif, dont la prise en main se veut immédiate. Le résultat est là : l'application revendique 77 millions d'utilisateurs depuis sa création en 2011. Inspiré par la **méthode Agile Kanban**, Trello s'articule historiquement autour d'un tableau digital de gestion de projet permettant de répartir les tâches, sous forme de cartes, au sein de colonnes (ou listes dans le langage de Trello) se déclinant par exemple en "A faire", "En cours" et "Fait".

Il est possible d'ajouter autant de colonnes à un tableau Trello que d'étapes à un projet. Des tâches sont ensuite assignées aux membres de l'équipe en charge de ce projet. Au fil de leur exécution, il suffit de glisser-déposer les cartes correspondantes d'une colonne à l'autre (voir capture ci-dessous). Des codes couleurs permettent de gérer les priorités. Au jour le jour, voire heure par heure, le tableau fournit ainsi l'état d'avancement des travaux d'un seul coup d'œil. Trello propose aussi des checklists, des dates limites et des notifications pour ne manquer aucune étape.

J'ai appliqué les principes de priorisation en créant des cards avec un code couleur dans mon Trello pour suivre le plan de réalisation de l'application et pouvoir visuellement montrer lors de nos réunions avec le comité directeur du club l'état d'avancement du projet.

⇒ Voir annexe [Trello](#)

Outils utilisés :

Après avoir pris en compte les besoins pour la création du projet j'ai choisi les technologies suivantes :

A. Langages informatiques



HTML 5 :

L'HyperText Markup Language, est le langage de balisage utilisé afin d'intégrer les informations au sein de la page Web.



CSS 3 :

Cascading Style Sheet pour appliquer une mise en forme des informations fournies par l'HTML. Le CCS vient en complément de Bootstrap5 pour un rendu précis sur des éléments particulier.



JavaScript 5 :

langage de programmation de scripts principalement employé dans les pages web interactives. Ici, cela m'a permis d'afficher des boites de confirmation lors des différentes actions des utilisateurs tels que supprimer une information, mais également pour le service de mailer, pour la génération et l'envoi de mail.



PHP 8 :

PHP est un langage informatique de script Open Source, principalement dédié au développement web (d'où son nom : *Hypertext Preprocessor*). Il s'agit d'un des langages les plus utilisés au monde pour créer des sites web dynamiques.

PHP est un langage interprété par le serveur, et non par le navigateur, comme c'est par exemple le cas pour Javascript. Quand un utilisateur accède à une page web au travers de son url, le serveur renvoie un fichier html après avoir exécuté les instructions contenues dans le script.

B. Environnement de développement :



Visual Studio Code :

L'éditeur de code Visual Studio Code m'a permis de développer le projet, pratique, gratuit il répondait parfaitement à mes besoins. Régulièrement mis à jour il intègre également des extensions permettant notamment une relation avec GitHub.



Github et Github Desktop :

Outils collaboratifs pour la gestion de code, utilisé principalement pour la sauvegarde mais également pour le suivi par mon tuteur de stage et mes formateurs.



Drawio :

Pour le maquetage de l'application, simple et gratuit il permet de maquetter rapidement et efficacement. La prise en main est simple et j'ai pu mettre rapidement travailler avec.



Gestionnaire de Base de données HeidiSQL :

HeidiSQL permet de gérer des bases de données Microsoft SQL Server, MySQL/MariaDB et PostgreSQL via une interface claire et complète. Il intègre la possibilité d'éditer les données, de modifier les bases et gérer les utilisateurs, de créer des tables et des vues, ainsi que des déclencheurs (triggers), et des événements planifiés. Enfin il supporte l'export en SQL de la structure et des données. Ayant travaillé avec durant ma formation je me suis appuyé sur ce gestionnaire que je connais.



Trello :

Un outil de gestion de projet, a été utilisé afin d'améliorer l'organisation des tâches.

Looping

Looping :

Application pour la conception du MCD et MLD selon la méthode Merise.

MERISE est une méthode d'analyse et de conception des systèmes d'information basée sur le principe de la séparation des données et des traitements. Elle possède plusieurs modèles qui sont répartis sur 3 niveaux (Le niveau conceptuel, le niveau logique ou organisationnel, le niveau physique).



Laragon :

Est un serveur de développement web pour Windows, qui est portable et qui n'impacte pas l'OS de base. Tout son environnement est isolé !

Nativement Laragon est fourni avec ;

- Apache ou Nginx (au choix ou en même temps mais avec des ports différents.
- Mysql ou MariaDB ou PostgreSQL ou MongoDB (au choix ou en même temps)

L'énorme avantage de Laragon est qu'il est extensible.

On peut si on le souhaite installer les environnements suivants ;

- Node.js / MEAN
- Ruby on Rails
- Python Django / Flask
- Java spring / SpringBoot
- Golang

Ayant utilisé ce serveur durant ma formation et ayant satisfait du travail réalisé j'ai prolongé son utilisation pour mon projet.

C. Framework

Un Framework est un cadre de travail, une infrastructure qui suit un schéma de fonctionnement et qui souvent inclut des bibliothèques. Son utilisation permet de gagner un gain de temps considérable.



Bootstrap 5

Collection d'outils utiles à la création du design de sites et d'applications web qui contient des codes CSS. Facilitant l'utilisation de ce langage. Il permet un code plus court, plus simple et plus efficace que le CSS seul. Il permet de se concentrer sur le code. J'ai choisi ce Framework CSS, car il est d'une part très populaire, de l'autre il met à disposition beaucoup de composants. De plus, Symfony permet d'intégrer des thèmes Bootstrap pour les formulaires.



Symfony 5.3

Symfony : est un ensemble de composant PHP ainsi qu'un framework MVC libre écrit en PHP. Il fournit des fonctionnalités modulable et adaptables qui permettent de faciliter et d'accélérer le développement d'un site web.



Twig

Est un moteur de template souple, rapide et sécurisé pour le langage de programmation PHP, il est utilisé par défaut par le framework Symfony.



Doctrine

Il s'agit d'un Object Relational Mapping. Il est utilisé par défaut par le framework Symfony. Un ORM sert à offrir une couche d'abstraction de connexion à toutes les BD relationnelles (comme PDO) mais aussi des facilités pour réaliser les requêtes courantes sans descendre au niveau des requêtes SQL.



Composer

Est un logiciel de gestionnaire de dépendances libre écrit en PHP. Il permet de gérer les dépendances d'un projet.

Développement

Dans la continuité de l'élaboration de mon projet et après avoir listé les outils pour la réalisation de mon application, il m'est apparu nécessaire de modéliser ma base de données.

A. Modélisation des données

La **modélisation** informatique des données est en réalité un processus de description de la structure, des **associations**, des relations et des impératifs liés à des datas disponibles.

Les modélisateurs utilisent souvent plusieurs modèles pour représenter les mêmes données et s'assurer que la totalité des processus, entités, relations et flux de données a été identifiée.

J'ai utilisé Looping pour la conceptualisation de mes données selon la **méthode Merise**.

MERISE est une méthode française née dans les années 70. Elle fut ensuite mise en avant dans les années 80, à la demande du ministère de l'Industrie qui souhaitait une méthode de conception des SI.

MERISE est donc une méthode d'analyse et de conception des systèmes d'information basée sur le principe de la séparation des données et des traitements. Elle possède un certain nombre de modèles (ou schémas) qui sont répartis sur trois niveaux :

- le niveau conceptuel ;
- le niveau logique ou organisationnel ;
- le niveau physique.

La **modélisation conceptuelle** de données permet d'identifier le niveau de relations le plus élevé entre différentes entités.

La **modélisation logique** des données découle directement du MCD et représentera une couche moins abstraite de notre modélisation.

1. Le niveau conceptuel :

Il s'agit de l'élaboration du **modèle conceptuel des données** (MCD) qui est une représentation graphique et structurée des données. Le **MCD** est basé sur deux notions principales : les entités et les associations.

L'élaboration du MCD passe par les étapes suivantes :

- la recherche des dépendances fonctionnelles entre ces données ;
- l'élaboration du MCD (création des entités puis des associations puis ajout des cardinalités).

⇒ Voir annexe [MCD](#)

On peut voir dans mon MCD la présence des **entités** Users et Covoiturage qui possèdent des propriétés identifiantes ainsi que des propriétés quelconques. Les **cardinalités 1.1** et **0..n** de la relation 'cree' montrent qu'un covoiturage peut être créé par un user unique mais qu'un user peut créer plusieurs covoiturations, dans le même temps la relation 'participe' **0..n** et **0..n** que l'on appelle aussi relation 'ManyToMany' montre qu'un user peut participer à zéro, un ou plusieurs covoiturations et qu'un covoiturage peut avoir zéro, un ou plusieurs users.

2. Le niveau logique :

Dans le **modèle logique de données** (MLD), une entité du MCD devient une table. La propriété identifiante d'une entité dans le MCD devient une **clé primaire** de la table dans le **MLD**. La clé primaire permet d'identifier de façon unique un enregistrement dans la table. Les valeurs de la clé primaire sont **uniques** et obligatoirement non nulles.

Nous pouvons aussi voir l'apparition de tables associatives dans le MLD qui résultent des relations Many To Many du MCD.

⇒ Voir annexe [MLD](#)

Mon MLD qui découle de mon MCD montre à présent que les propriétés identifiantes sont devenues les clés primaires et que la relation 'participe' entre users et covoiturage est devenue une table associative qui a comme clé primaire l'association des clés étrangères qui la compose.

B. Arborescence du site

Après la modélisation, j'ai réfléchi à la manière dont j'allais organiser mon site et mes pages pour que la navigation soit la plus simple et claire possible.

- L'arborescence d'un site web désigne l'organisation du contenu et des pages d'un site internet et les liens entre chaque page. Un site Web est constitué de contenu sur une variété de sujets et présenté sous la forme d'articles ou de pages. L'arborescence est vraiment le squelette ou la structure du site et montre la manière dont son contenu est groupé, lié et présenté au visiteur.
- L'arborescence de votre site web joue un rôle majeur en termes d'utilisabilité et de visibilité, soit en termes d'**UX** (User Experience) et de **SEO** (Search Engine Optimization).

Dans mon projet, j'ai fait attention de pouvoir respecter la règle des 3 clics pour participer à une meilleure **ergonomie** de mon site, j'ai tenu compte de l'expérience utilisateur pour un site soigné et mieux référencé.

⇒ Voir annexe [Arborescence du site](#)

C. Maquettage du site

Après la modélisation puis l'arborescence, je suis passé au maquettage du site.

Une maquette est une représentation graphique d'un site web, présenté sous forme statique à un client, elle permet d'avoir un premier rendu d'un site web en indiquant l'emplacement des blocs afin de concevoir la structure d'un site. Elle permet également d'avoir une première idée de l'interface utilisateur (UI) et de l'expérience utilisateur (UX).

Pour ce projet, j'ai choisi de réaliser mes maquettes avec Draw.io. Celui-ci est une application gratuite en ligne, accessible via son navigateur (protocole https) qui permet de dessiner des diagrammes ou des organigrammes. Cet outil vous propose de concevoir toutes sortes de diagrammes, de dessins vectoriels, de les enregistrer au format XML puis de les exporter.

J'ai essayé de concevoir mes maquettes de la façon la plus ergonomique et la plus intuitive possible pour l'utilisateur. J'ai réalisé les maquettes pour la version desktop puis pour la version mobile.

Lors du développement de mon projet, mes maquettes m'ont grandement facilité la tâche en termes d'intégration web. Le site web final pour la partie accueil est responsive et adaptable. Lors de la mise en place du responsive, j'ai notamment utilisé les **Media Queries**, ce sont des spécifications de CSS3 qui permettent d'attribuer des propriétés CSS en fonction des largeurs d'écran.

⇒ Voir annexe [Arborescence du site](#)

D. Le responsive et les médias queries

Le **Responsive Web Design (RWD)** ajuste automatiquement l'affichage d'une page web à la taille d'écran du terminal utilisé. Cette technique de conception de site web, ou d'interface digitale, répond à un besoin des utilisateurs, toujours plus nombreux à se connecter sur le web depuis un appareil mobile.

Le Responsive Design permet de faciliter la navigation et d'améliorer l'expérience utilisateur lorsqu'il s'agit de consulter le site sur un appareil mobile. Le Responsive Web Design est souvent confondu avec un concept plus large, l'Adaptive Design. **Design responsive ou Design adaptatif**, les deux méthodes de conception visent à améliorer l'ergonomie mobile du site web. C'est un enjeu majeur pour les entreprises, tant en termes de référencement que pour s'adapter aux nouveaux usages.

Les **Media Queries** sont des spécifications de CSS3 qui permettent d'attribuer des propriétés CSS en fonction de conditions particulières (exemple : largeur de l'écran). Ces spécifications sont particulièrement connues pour leurs utilités dans la conception d'un responsive web design.

E. Le référencement

Dans l'univers du **SEO (search engine optimisation)**, le terme **référencement** regroupe les différentes techniques utilisées pour améliorer la position d'un site internet dans les pages de résultats affichées par les moteurs de recherche en réponse aux requêtes des internautes.

Il s'agit de faire en sorte d'apparaître, si possible, sur la première page affichée par **Google** (c'est le moteur de recherche principal qui est visé) lorsqu'un internaute effectue une recherche en lien avec le site web concerné par le **référencement**.

Le référencement fait appel à différentes techniques comme l'optimisation du contenu avec l'utilisation de mots-clés, la mise en place de liens hypertextes pointant vers le site web, etc. Il faut être capable de maîtriser les algorithmes de Google qui cherchent à filtrer les résultats pour ne proposer aux internautes que des réponses de qualité. Un site web qui apparaît en première position dans Google lors d'une recherche a généralement tout compris des enjeux du référencement. A noter qu'il existe également un référencement payant (ou **SEA**) qui s'oppose au référencement naturel (**SEO**) car il faut payer Google pour faire apparaître (via le système AdWords), un site lorsqu'un internaute tape un mot clé.

Dans mon projet j'ai utilisé :

- Un système de balisage HTML cohérent et respectant les conventions de référencement
- Un système de texte alternatif pour les photos -> alt
- Choix des mots clés
- Optimisation du contenu (choix des H1, strong...)
- Balise meta -> title
- Performance de l'application
- Les liens internes et externes

Symfony 5.3 comme Framework

J'ai choisi d'utiliser le framework Symfony pour réaliser mon projet, principalement car nous l'avons utilisé au cours de la formation, ce qui m'a permis de gagner du temps dans le développement de mon projet. Ce framework possède de nombreux avantages notamment la sécurité native ainsi qu'une facilité d'utilisation grâce à sa documentation complète et le soutien de sa communauté. De plus il fournit des fonctionnalités modulables et adaptables qui permettent de faciliter et d'accélérer le développement.

A. Le design pattern

1. Définition :

Les **design patterns** sont des solutions typiques à des problèmes communs en développement logiciel : ils ne sont pas une implémentation concrète d'une solution à un problème, mais plutôt une stratégie à appliquer pour le résoudre de façon élégante et maintenable.

2. Avantages :

L'utilisation des design patterns offre de nombreux avantages. Tout d'abord cela permet de répondre à un problème de conception grâce à une solution éprouvée et validée par des experts. Ainsi on gagne en rapidité et en qualité de conception ce qui diminue également les coûts. De plus, les design patterns sont réutilisables et permettent de mettre en avant les bonnes pratiques de conception. Les design patterns étant largement documentés et connus d'un grand nombre de développeurs ils permettent également de faciliter la communication.

B. Le MVC de Symfony

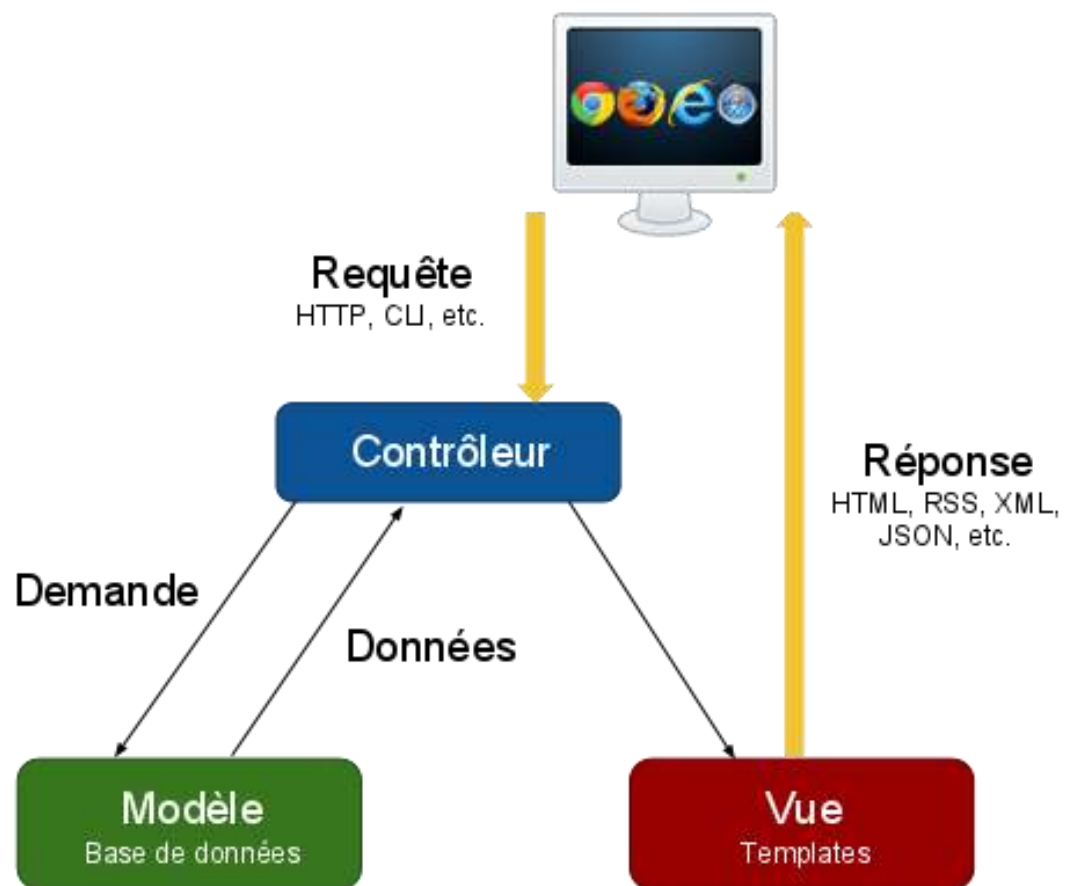
L'architecture logicielle de Symfony correspond au MVC : « modèle-vue-contrôleur ».

Cela nous permet de séparer 3 points essentiels :

- La couche Model, son rôle est de récupérer et gérer les données brutes de la base de données à l'aide de requêtes DQL.
- La couche Controller, qui gère la logique relative aux traitements des demandes, sert également à effectuer des vérifications et des autorisations.
- La couche View, correspond à l'affichage côté client.

1. Schéma :

Voici de manière plus précise et schématique la structure de Symfony. Sur demande de l'utilisateur, le client envoie une requête http au serveur, elle est traitée par le contrôleur frontal de Symfony (index.php) qui fera appel au service de routing et qui vérifiera si la méthode et le contrôleur existe bien. S'ils existent, alors le contrôleur demande les données au modèle qui traduit cette demande par une requête DQL, il récupère les données et les renvoie au contrôleur. Le contrôleur génère la vue, puis il transmet la réponse HTML au client.



2. Utilisation :

Concrètement dans mon projet cela se traduit comme suit :

Le client envoie une requête **HTTP*** qui sera traité par le contrôleur frontal qui grâce au système de routing de Symfony fera appel au bon contrôleur et à la bonne méthode pour traiter la demande de l'utilisateur. Le contrôleur est une classe qui contient des méthodes, chaque méthode correspond à une route particulière et retournera un tableau de données et une vue. Dans cet exemple, l'utilisateur, le user en session souhaite avoir accès à tous les prochains covoiturages. Le Contrôleur fait alors appel au « CovoiturageRepository » (\$repoCovoit) qui correspond au Modèle.

*Requête **HTTP** : une requête http est une demande effectuée par le navigateur WEB au serveur **HTTP** lorsqu'il souhaite télécharger une page web.

```
// Je déclare ma route pour afficher la liste des covoiturages à venir
#[Route('/nextCovoituragesList', name: 'nextCovoituragesList')]
// Je crée une fonction nextCovoiturage dans laquelle j'aurais besoin
// du Repository de l'entité Covoiturage, de la classe Request de la Symfony\Component\HttpFoundation\Request
// et pour finir la classe Response me permettant d'utiliser une interface orientée objet
// pour construire la réponse qui doit être renvoyée au client :
public function nextCovoiturage(CovoiturageRepository $repoCovoit, Request $request): Response
{
    // J'instancie ma variable et lui dit de chercher dans le repository la fonction findNextCovoiturages
    $covoiturages = $repoCovoit->findNextCovoiturages();

    // Je retourne la vue dans laquelle je passe le tableau dans lequel

    // J'instancie ma variable et lui dit de créer un formulaire en allant chercher le formulaire ResearchType
    $form = $this->createForm(ResearchType::class);

    // J'instancie mon form et lui dit de passer en requête la requête.
    $research = $form->handleRequest($request);

    // Si mon formulaire est soumis et est valide
    if ($form->isSubmitted() && $form->isValid()) {

        // J'instancie ma variable et lui dit de passer la fonction findResearch de mon repo
        $covoiturages = $repoCovoit->findResearch(

            // A mon formulaire je vais aller chercher dans les datas equipe et weekend
            $research->get('equipe')->getData(),
            $research->get('weekend')->getData(),
            // $research->get('conducteur')->getData()
        );
    }

    return $this->render('covoiturage/nextCovoituragesList.html.twig', [

        // seront passés les variables $covoiturages et $form
        'covoiturages' => $covoiturages,
        'form' => $form->createView()
    ]);
}
```

La couche modèle, appelée aussi couche « métier », correspond à la partie de l'architecture ayant trait aux données : les entités, les repositories. Les modèles gèrent l'accès aux données, le plus souvent dans une base de données.

Le modèle traite la demande du contrôleur en récupérant les données, puis les retourne au contrôleur.

```

50     public function findNextCovoiturages()
51     {
52         $currentdate = (new \DateTime('now'))->format('Y-m-d'); //Date du jour
53
54         return $this->createQueryBuilder('c')
55             ->innerJoin('c.weekend', 'w')
56             ->where('w.dateMatch >= :dateMatch')
57             ->setParameter('dateMatch', $currentdate)
58             ->orderBy('w.dateMatch', 'ASC')
59             ->getQuery()
60             ->getResult();
61     }

```

Le Contrôleur retourne alors une vue avec les données.

```

48     <table class="table table-hover">
49         <thead>
50             <tr>
51                 <th scope="col">Nom du covoiturage</th>
52                 <th scope="col">Conducteur</th>
53                 <th scope="col">Place libres</th>
54                 <th scope="col">Date</th>
55                 <th scope="col">Equipes</th>
56                 <th scope="col">S'inscrire</th>
57             </tr>
58         </thead>
59         <tbody>
60             <% for covoiturage in covoiturages %>
61                 <tr>
62                     <td>
63                         <a class="text-body" href="{{ path('covoitageshow', {'id': covoiturage.id }) }}">{{ covoiturage.nom }}</a>
64                     </td>
65                     <td>
66                         {{ covoiturage.createur }}
67                     </td>
68                     <td>
69                         <% if covoiturage.noPlace == covoiturage.users | length %>
70                             <span>complet</span>
71                         <% else %>
72                             {{ covoiturage.noPlace - covoiturage.users | length }}
73                         <% endif %>
74                     </td>
75                     <td>
76                         {{ covoiturage.weekend }}
77                     </td>
78                     <td>
79                         {{ covoiturage.equipe }}
80                     </td>
81                     <td>
82                         <% if covoiturage.users is empty %>
83                             <a href="#">S'inscrire</a>
84                         <% else %>
85                             <% if ann.user in covoiturage.users %>

```

C. Gestion des utilisateurs

Symfony nous permet de créer rapidement et simplement un système de gestion des utilisateurs.

1. Utilisateur

Grâce au MakerBundle, nous pouvons utiliser la commande « php/bin console make :user » qui nous permet de créer une entité User, elle possédera les propriétés suivantes par défaut un identifiant unique tel que le numLicence, un mot de passe ainsi qu'un rôle. Lorsqu'une entité est créée avec la console, le repository de l'entité est également créé (il correspond au Modèle de l'architecture MVC).

Par défaut, cette commande créera une classe « User Provider », qui permettra notamment de recharger l'utilisateur à partir de la session.

La méthode de hachage des mots de passe des utilisateurs est définie dans le fichier security.yaml (*ci-dessous*), dans notre cas l'algorithme de hachage est laissé par défaut en « auto », car il permet de sélectionner le meilleur algorithme possible selon Symfony.

```
1 security:
2     # encoders:
3     password_hashers:
4         App\Entity\User:
5             algorithm: auto
```

Symfony utilise le meilleur algorithme au moment de son utilisation pour hacher les mots de passe. Il s'agit d'une méthode de hachage* récente et sécurisée. Lorsque l'on récupère le mot de passe en clair de l'utilisateur, on le hache avant de le stocker en base de données. Lorsque l'utilisateur souhaite se connecter, on récupère son mot de passe en clair on le hache puis on compare les deux mots de passe hachés (celui en base et celui venant d'être utilisé) pour voir s'ils correspondent. Le hachage est définitif. Un salt aléatoire est généré par la fonction **password_hash()** pour chaque mot de passe haché. Depuis PHP8 un salt fixe est ignoré.

*Hachage : Opération qui consiste à appliquer une fonction mathématique à un groupe de données de taille variable afin de générer un code unique de taille fixe, que l'on utilisera pour l'authentification et le stockage d'information.

2. Inscription

Pour créer un formulaire d'inscription Symfony fournit également, grâce au MakerBundle, la commande « php bin/console make :registration-form » permettant de générer un contrôleur et un formulaire d'inscription.

3. Authentification

Symfony nous permet de générer un formulaire d'authentification facilement avec la commande « `php bin/console make:auth` » dans la console. Cette commande génère un « `SecurityController` » qui contient les méthodes pour se connecter et se déconnecter, ainsi qu'une vue qui inclut un formulaire HTML basique.

La section « Firewall » du fichier `security.yaml` (ci-dessous), permet de définir comment les utilisateurs s'authentifieront.

« `logout` » permet de choisir quel chemin déclenchera la déconnexion et vers quel chemin l'utilisateur sera rediriger.

Dans mon projet, nous voyons que la route de déconnexion est le « `app_logout` » et qu'après la déconnexion nous serons redirigés vers la page home

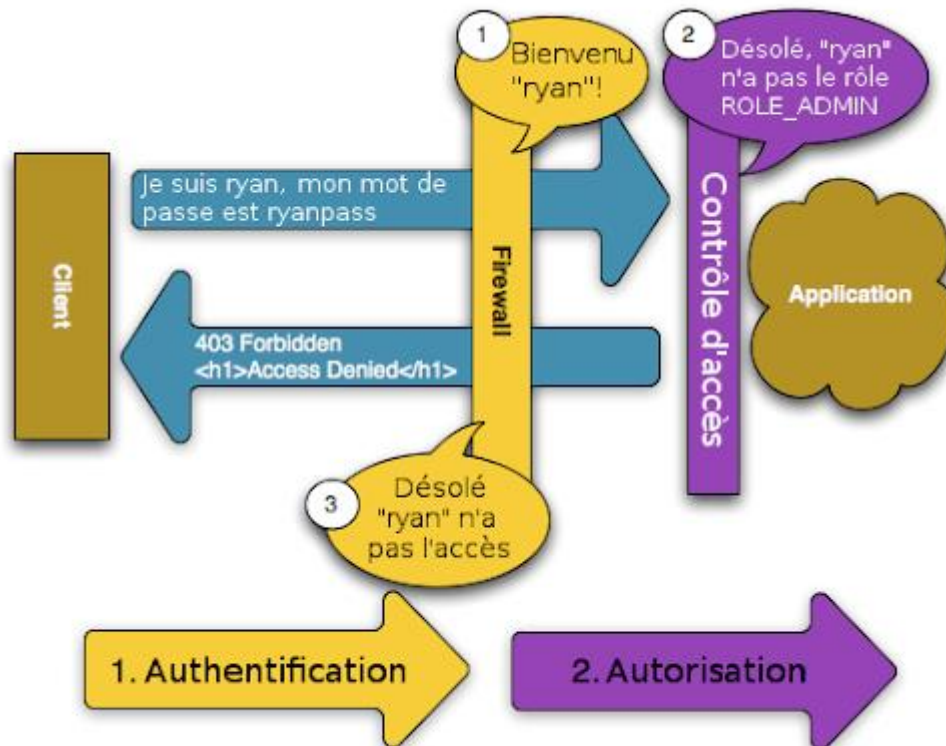
```
16 firewall:
17     csrf:
18         # La partie ci-dessous ne permet d'avoir le debug toolbar et pouvoir accéder aux fichiers statiques dans les assets
19         pattern: ^/( (profile|admin)|css|images|js)/
20         security: false
21     main:
22         lazy: true
23         provider: app_user_provider
24         custom_authenticator: App\Security\Authenticator
25         logout:
26             path: app_logout
27             # where to redirect after logout
28             target: home
29
```

4. Autorisation

La dernière partie du fichier `security.yaml` (ci-dessous) concerne les autorisations. Elle définit une protection de sécurité des URL de mon application. Autrement dit pour une route qui commence par « `/admin` » uniquement les utilisateurs authentifiés et possédant le rôle `admin` pourront y avoir accès, les autres utilisateurs auront l'accès refusé. Il en va de même pour les routes qui commencent par « `/profil` », seuls les utilisateurs ayant le rôle `User` pourront y accéder, l'accès sera refusé aux autres utilisateurs anonymes.

```
38 access_control:
39     - { path: ^/admin, roles: ROLE_ADMIN }
40     - { path: ^/profile, roles: ROLE_USER }
41
```

Ce schéma résume très bien le fonctionnement de l'authentification et de l'autorisation. Dans un premier temps, on cherche à savoir qui (authentification) puis on cherche à savoir quel rôle il possède (autorisation).



Dans Symfony, il faut distinguer l'authentification de l'autorisation. En effet la deuxième n'intervient qu'après la première, lorsqu'un visiteur est sur l'application il peut naviguer sans être authentifié et pourra accéder à certaines parties du site (ou pas d'ailleurs) suivant le schéma de sécurité mis en place. Lorsqu'un visiteur est identifié (par un nom d'utilisateur et un mot de passe ou un cookie par exemple) il se verra affecter un 'rôle'(basique par défaut) et pourra avoir accès à des parties du site spécifique qui lui seront réservées.

Dans mon projet après s'être identifié le visiteur suivant son 'rôle' pourra avoir des possibilités d'interagir sur le site. Par exemple un visiteur aillant le 'rôle user' pourra consulter la liste des matchs à venir pour son «équipe» ainsi que les informations s'y rapportant, un visiteur aillant le 'rôle admin' pourra lui interagir sur la mise en ligne d'informations ou la suppression de celles-ci.

Dans mon projet nous pouvons voir un exemple pour la liste des matchs

```
// Je déclare ma route pour afficher la liste des matchs
// Seul quelqu'un ayant un rôle pourra s'y rendre
#[Route('profile/joue', name: 'joue')]

// Je crée une fonction index dans laquelle j'aurai besoin
// du Repository de l'entité Joue, de la classe Request de la Symfony\Component\HttpFoundation\Request
// et pour finir la classe Response me permettant d'utiliser une interface orientée objet
// pour construire la réponse qui doit être renvoyée au client :
public function index(JoueRepository $repo, Request $request): Response
```


Comment Symfony se prémunit des principales failles de sécurité

J'ai choisi Symfony car nativement il gère les 3 principales failles de sécurité.



A. La faille XSS

La faille XSS ou Cross Site Scripting consiste à injecter du code malveillant à travers des formulaires GET ou POST. Dans Symfony, le gestionnaire de template Twig se prémunit des attaques XSS au niveau des inputs des formulaires en « échappant » les caractères: ils sont transformés en entités html, non-interprétables en tant que code.

Sans framework :

```
<?php
$new = htmlspecialchars("<a href='test'>Test</a>", ENT_QUOTES);
echo $new; // <a href='test'>Test</a>
?>
```

Dans mon projet :

```
{ { form_start(contactForm) }}
<form>
  <div class="row">
    <div class="col">
      { { form_row(contactForm.nom | escape) }}
    </div>
    <div class="col">
      { { form_row(contactForm.email | escape) }}
    </div>
  </div>
  <div class="row">
    { { form_row(contactForm.message | escape) }}
  </div>
  <div class="row" onclick="rgpd()">
    { { form_row(contactForm.envoyer) }}
  </div>
  { { form_end(contactForm) }}
</form>
```

B. L'attaque CSRF

Les attaques CSRF consistent à tromper l'utilisateur de façon à le faire valider une action malveillante grâce à son authentification. Pour s'en prémunir, il faut créer un jeton d'authentification lié à la session, et l'on vérifie pour chaque action que la requête dispose de ce jeton. Ainsi il est clair que l'action vient bien de l'application. Dans Symfony, les formulaires créés avec Symfony Form component et les form builders incluent des jetons d'authentification par défaut.

Sans framework on utiliserait `hash_equals` pour comparer le token que l'on aurait dans un champ « hidden » de notre formulaire avec celui créé avec la session que l'on aurait mis en place.

dans mon projet :

Dans mon formulaire de contact créé avec le form builder, Symfony a implémenté un champ de type hidden contenant un token ici avec l'id='contact_token' et ayant une valeur hachée. Il sera comparé au moment du submit. Si les tokens correspondent alors le message sera envoyé sinon le formulaire ne sera pas envoyé.

```
</div>
<input type="hidden" id="contact_token" name="contact[ token]"
value="17ed7421d81.q1wVuIhn4Y3D/Xn-zZvZghbmZ-
2QF7kh2m64zi3u6AF.405Futk7iPSAdjiz16646M&kC7nYJMWYriD1_nSnuk7tXk3A_iWS0p54MA">
```

C. L'injection SQL

L'Injection SQL est une méthode permettant d'injecter du code SQL dans un formulaire par exemple. Entre autres, elle peut permettre à un utilisateur malveillant de récupérer des données ou d'exécuter du code SQL afin de supprimer toute la base de données. L'échappement des données avec Twig et les contraintes de validation de Symfony permettent de filtrer les champs des formulaires. Elles sont à mettre en place pour chaque champ de formulaire. Grâce à Doctrine, lors de requêtes personnalisées, il faut utiliser « `setParameter` », elles deviennent alors des requêtes paramétrées et évitent les injections SQL.

Les requêtes préparées correspondent à une façon de créer et d'exécuter nos requêtes selon trois étapes : une étape de préparation, une étape de compilation et finalement une dernière étape d'exécution.

Préparer ses requêtes comporte des avantages notables notamment dans le cas où l'on doit exécuter un grand nombre de fois une même requête ou si l'on doit insérer des données envoyées par les utilisateurs.

Il y a 3 temps pour la requête **la préparation, la compilation et l'exécution**

Tout d'abord, une première phase de préparation dans laquelle nous allons créer un template ou schéma de requête, en ne précisant pas les valeurs réelles dans notre requête mais en utilisant plutôt des marqueurs nommés (sous la forme :nom) ou des marqueurs interrogatifs (sous la forme ?).

Ces marqueurs nommés ou interrogatifs (qu'on peut plus globalement nommer marqueurs de paramètres) vont ensuite être remplacés par les vraies valeurs lors de l'exécution de la requête. Notez que vous ne pouvez pas utiliser les marqueurs nommés et les marqueurs interrogatifs dans une même requête SQL, il faudra choisir l'un ou l'autre.

Une fois le template créé, la base de données va analyser, compiler, faire des optimisations sur notre template de requête SQL et va stocker le résultat sans l'exécuter.

Finalement, nous allons lier des valeurs à nos marqueurs et la base de données va exécuter la requête. Nous allons pouvoir réutiliser notre template autant de fois que l'on souhaite en liant de nouvelles valeurs à chaque fois.

Le risque d'injection SQL est minimisé puisque notre requête est pré-formatée et nous n'avons donc pas besoin de protéger nos paramètres ou valeurs manuellement.

```
public function findHomeMatch()
{
    $currentdate = (new \DateTime('now'))->format('Y-m-d'); //Date du jour

    $qb = $this->createQueryBuilder('j')
        ->leftJoin('j.weekend', 'w')
        ->where('w.dateMatch >= :date')
        ->setParameter('date', $currentdate)
        ->orderBy('w.dateMatch', 'ASC')
        ->setMaxResults(3);
    $query = $qb->getQuery();
    return $query->execute();
}
```


Doctrine ORM de Symfony

ORM signifie Object-Relationnal Mapper. Un ORM sert à offrir une couche d'abstraction de connexion à toutes les BD relationnelles (comme PDO) mais aussi des facilités pour réaliser les requêtes courantes sans descendre au niveau des requêtes SQL et pour générer automatiquement des entités dans le langage utilisé avec les *getters* et *setters* correspondants.

Le **Data Mapper** est une couche qui synchronise la donnée stockée en base avec les objets PHP. En d'autres termes :

- il peut insérer, mettre à jour des entrées en base de données à partir de données contenues dans les propriétés d'un objet ;
- il peut supprimer des entrées en base de données si les "entités" liées sont identifiées pour être supprimées ;
- il "hydrate" des objets en mémoire à partir d'informations contenues en base.

L'implémentation dans le projet Doctrine de ce Data Mapper s'appelle l'Entity Manager, les entités ne sont que de simples objets PHP mappés.

C'est l'Entity Manager qui est responsable de la gestion de nos entités. C'est pourquoi nous avons besoin de l'injecter dans nos contrôleurs pour "persister" et "flusher" nos entités.

Dans mon projet je fais appel à Doctrine lorsque j'interroge ou modifie ma base de données.

Ci-dessous je crée une nouvelle catégorie si elle n'existe pas, je fais appel au formulaire et je récupère les data puis je prépare ma base de données (persist) et je mets à jour ma base de données (flush).

```

// Les routes suivantes si c'est un ajout ou une modification
#[Route('/categorie/new', name:'categorie_add')]
#[Route('/categorie/{id}/edit', name:'categorie_edit')]

// Je crée une fonction nouveau dans laquelle je passe catégorie et je lui dit que pour le moment elle est à null
// je passe également la request et la reponse
public function new(Categorie $categorie = null, Request $request): Response
{
    // S'il n'y a pas cette catégorie alors c'est une nouvelle
    if (!$categorie) {
        $categorie = new Categorie();
    }

    // J'instancie ma variable form et je lui dit que créer un formulaire sur la base de categorie
    $form = $this->createForm(CategorieType::class, $categorie);

    // Je crée la request et je lui passe la requete
    $form->handleRequest($request);

    // Si mon formulaire est soumis et est valide
    if ($form->isSubmitted() && $form->isValid()) {

        //Pour ma variable categorie je vais chercher les datas dans mon form
        $categorie = $form->getData();

        // Je fais appel à entity manager et je persist et flush pour ma Add
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->persist($categorie);
        $entityManager->flush();

        //Je retourne à la liste des catégories
        return $this->redirectToRoute('categorieList');
    }
}

```

Système de covoiturage

Ma fonctionnalité représentative dans ce projet est le système de covoiturage

1. Fonction pour les covoiturages à venir :

```
// Je déclare ma route pour afficher la liste des covoiturages à venir
Route('/nextCovoituragesList', name: 'nextCovoituragesList')
// Je crée une fonction nextCovoiturage dans laquelle j'aurais besoin
// du Repository de l'entité Covoiturage, de la classe Request de la Symfony\Component\HttpFoundation\Request
// et pour finir la classe Response me permettant d'utiliser une interface orientée objet
// pour construire la réponse qui doit être renvoyée au client :
public function nextCovoiturage(CovoiturageRepository $repoCovoit, Request $request): Response
{
    // J'instancie ma variable et lui dit de chercher dans le repository la fonction findNextCovoiturages
    $covoiturages = $repoCovoit->findNextCovoiturages();

    // Je retourne la vue dans laquelle je passe le tableau dans lequel

    // J'instancie ma variable et lui dit de créer un formulaire en allant chercher le formulaire ResearchType
    $form = $this->createForm(ResearchType::class);

    // J'instancie mon form et lui dit de passer en requête la requête.
    $research = $form->handleRequest($request);

    // Si mon formulaire est soumis et est valide
    if ($form->isSubmitted() && $form->isValid()) {
        // J'instancie ma variable et lui dit de passer la fonction findResearch de mon repo
        $covoiturages = $repoCovoit->findResearch(
            // A mon formulaire je vais aller chercher dans les datas equipe et weekend
            $research->get('equipe')->getData(),
            $research->get('weekend')->getData(),
            // $research->get('conducteur')->getData()
        );
    }

    return $this->render('covoiturage/nextCovoituragesList.html.twig', [
        // seront passés les variables $covoiturages et $form
        'covoiturages' => $covoiturages,
        'form' => $form->createView()
    ]);
}
```

La fonction de mon repository à laquelle je fais appel plus haut :

```
public function findNextCovoiturages()
{
    $currentdate = (new \DateTime('now'))->format('Y-m-d'); //Date du jour

    return $this->createQueryBuilder('c')
        ->innerJoin('c.weekend', 'w')
        ->where('w.dateMatch >= :dateMatch')
        ->setParameter('dateMatch', $currentdate)
        ->orderBy('w.dateMatch', 'ASC')
        ->getQuery()
        ->getResult();
}
```

2. Fonction d'ajout d'un covoiturage.

S'il n'y a pas de covoiturage à ce nom alors je crée un nouveau covoiturage

Je vais chercher le user en session et je l'instancie à \$user

Je crée un formulaire et je traite la requête

J'instancie ma variable avec les datas de mon formulaire et le setCreateur avec en paramètre le \$user.

Je fais appel à Doctrine pour mettre à jour ma base de données.

```

#[IsGranted('ROLE_USER', message: 'Vous n\'avez pas accès à cette route.')]
#[Route('/covoiturage/new', name: 'covoiturage_add')]
#[Route('/covoiturage/{id}/edit', name: 'covoiturage_edit')]

public function new(Covoiturage $covoiturages = null, Request $request): Response
{
    if (!$covoiturages) {
        $covoiturages = new Covoiturage();
    }

    $user = $this->getUser(); // Je recupere le user en session
    $form = $this->createForm(CovoiturageType::class, $covoiturages);

    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {

        $covoiturages = $form->getData();
        $covoiturages->setCreateur($user);

        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->persist($covoiturages);
        $entityManager->flush();

        return $this->redirectToRoute('nextCovoituragesList');
    }

    return $this->render('covoiturage/add_edit.html.twig', [
        'formAddCovoiturage' => $form->createView(),
        'editMode' => $covoiturages->getId() !== null
    ]);
}

```

3. Fonctions d'ajout/suppression de participants :

Je récupère le user en session et je l'instancie à \$user ensuite je addUser en lui passant le \$user en paramètre.

```

#[IsGranted('ROLE_USER')]
#[Route('/covoiturage/{id}/join', name: 'covoiturage_join')]

public function join(Covoiturage $covoiturations = null, Request $request): Response
{
    $user = $this->getUser(); // je recupere le user en session

    $covoiturations->addUser($user);

    $entityManager = $this->getDoctrine()->getManager();
    $entityManager->persist($covoiturations);
    $entityManager->flush();

    return $this->redirectToRoute('nextCovoiturationsList');
}

#[IsGranted('ROLE_USER')]
#[Route('/covoiturage/{id}/deleteParticipant', name: 'covoiturage_deleteParticipant')]

public function deleteParticipant(Covoiturage $covoiturage, EntityManagerInterface $manager)
{
    $user = $this->getUser(); // je recupere le user en session

    $covoiturage->removeUser($user);
    $manager->persist($covoiturage);
    $manager->flush();

    return $this->redirectToRoute('nextCovoiturationsList');
}

```

4. Fonction d'affichage de ses propres covoiturations :

```

#[Route('/covoiturageListByCreateur', name: 'covoiturageListByCreateur')]
public function creator(CovoiturageRepository $repoCovoit): Response
{
    $id = $this->getUser()->getId();
    $covoiturations = $repoCovoit->searchByCreateur($id);

    return $this->render('covoiturage/covoiturageListByCreateur.html.twig', [
        'covoiturations' => $covoiturations,
    ]);
}

```


Je vais chercher dans mon repository la fonction que j'ai créée pour afficher les covoiturages par créateur.

```
public function searchByCreateur($id){  
    $query = $this->createQueryBuilder('c');  
    $query->leftJoin('c.createur', 'cr');  
    $query->andWhere('cr.id = :id_user');  
    $query->setParameter('id_user', $id);  
    return $query->getQuery()->getResult();  
}
```

5. Suppression en cascade des covoiturages :

Quand je supprime un user, je dois m'assurer que les covoiturages auxquels il participe mais également qu'il a créé soient supprimés également.

Dans l'Entité Covoiturage \$createur est lié à l'Entité User par le \$conducteur et je demande à l'ORM Doctrine de supprimer en cascade tout ce qui est en relation avec le User par le createur_id lorsqu'un User est supprimé.

```
/**  
 * @ORM\ManyToOne(targetEntity=User::class, inversedBy="conducteur")  
 * @JoinColumn(name="createur_id", referencedColumnName="id", onDelete="CASCADE")  
 */  
private $createur;
```

Lorsqu'un User est supprimé il doit aussi être supprimé de tous les covoiturages auxquels il participe :

```
/**  
 * @ORM\OneToMany(targetEntity=Covoiturage::class, mappedBy="createur", orphanRemoval=true)  
 */  
private $conducteur;  
  
/**  
 * @ORM\ManyToMany(targetEntity=Covoiturage::class, inversedBy="users")  
 */  
private $participant;
```

Utilisation de l'API Leaflet

Pour donner de la cohérence à mon site, j'ai décidé de m'appuyer sur l'API de géolocalisation Leaflet pour mettre en place un système de cartographie pour des besoins spécifique dans mon projet tel que localiser une salle sur une carte ou un emplacement de covoiturage.

Une **API** (c'est-à-dire une interface de programmation d'applications) est une interface standardisée et sécurisée qui permet aux applications de communiquer et de fonctionner entre elles. Ce type d'interface est spécialement conçu pour la récupération et la mise à jour d'informations sans intervention manuelle de l'utilisateur.

Les API publiques (ou ouvertes) sont des interfaces tierces disponibles pour une consommation externe. Un fournisseur de données ou une organisation peut choisir de rendre tout ou partie de ses données disponibles via des API pour un usage public.

L'API elle, peut utiliser n'importe quel moyen de communication pour initier l'interaction entre les applications. Les données échangées peuvent être dans n'importe quel format mais **souvent ce sont les formats XML ou JSON qui sont utilisés**. On parle par exemple d'APIs de type **JSON/Rest**.

Le **JSON** est un format qui stocke des informations structurées et est principalement utilisé pour transférer des données entre un serveur et un client.

Le fichier est principalement une alternative plus simple et plus légère au **XML** (Extensive Markup Language) qui a des fonctions similaires.

⇒ Voir annexe [1. Mise en place de la cartographie :](#)

Je déclare 2 variables et leurs donne comme valeur une latitude et une longitude qui seront celles utilisés comme base de travail.

```
var startlat = 48.085461884393816;  
var startlon = 7.3907848117717885;
```

Je déclare une variable dans laquelle de fait passer plus arguments

```
var options = {  
  center: [startlat, startlon],  
  zoom: 12  
}
```

Le centrage de la carte par rapport aux 2 variables et le zoom à 12

Je suis dans un fichier javascript donc je vais aller chercher un élément par son id dans le document

```
document.getElementById('lat').value = startlat;  
document.getElementById('lon').value = startlon;
```


Je déclare deux autres variables pour terminer la mise en place de la cartographie

```
var map = L.map('map', options);  
var nzoom = 12;
```

Je déclare ma carte et lui injecte les parametres dont je vais avoir besoin pour la mise en service.

Leaflet utilise un token d'activation que l'on doit utiliser en même temps que la carte ce qui le rend unique au projet.

Ensuite on besoin de savoir quel type de cartographie, ici dans mon projet je m'appuie sur openStreetMap.

Lorsque tout est initié j'ajoute à ma carte tous les éléments via le addTo

```
.addTo(map);
```

⇒ Voir annexe [2. Mise en place des marqueurs et de l'adresse :](#)

Je déclare ma variable et lui passe comme arguments la latitude et la longitude

Je l'ajoute à ma carte grace au addTo

```
.addTo(map);
```

Je vais chercher dans le document l'élément par son id

```
document.getElementById('lat').value = lat;  
document.getElementById('lon').value = lon;
```

Je créé un popup et lui injecte la latitude et la longitude

```
myMarker.bindPopup("Lat " + lat + "<br />Lon " + lon).openPopup();
```

Pour la fonction chooseadress dans laquelle je passe lat1 et lng1 comme arguments, me servira à aller récupérer les coordonnées d'une adresse pour pouvoir ensuite les injecter dans mes marqueurs

Je ferme le popup myMarker, ensuite je setview la map avec lat1 et lng1.

A myMarker je lui setLatLng puis j'instancie

```
lat = lat1.toFixed(8);  
lon = lng1.toFixed(8);
```

Je vais chercher dans le document l'élément par son id

```
document.getElementById('lat').value = lat;  
document.getElementById('lon').value = lon;
```

⇒ Voir annexe 3. Création du champs adresse :

Je vais créer ma fonction qui me permettra de renseigner une adresse pour pouvoir faire une requête et la passer dans l'Url

Je crée une fonction qui me permettra d'aller chercher dans le document l'élément par son id

```
var inp = document.getElementById("addr");
```

Je vais créer une nouvelle requête http via le XMLHttpRequest

```
var xmlhttp = new XMLHttpRequest();
```

Je vais instancier \$url = et je vais aller chercher l'adresse qui me servira à récupérer des coordonnées latitude et longitude pour les injecter dans mon marqueur.

```
var url = "https://nominatim.openstreetmap.org/search?format=json&limit=3&q=" + inp.value;
```

Je fais appel à une fonction de callback si la demande a eu une réponse et que c'est OK alors ==4 et le statut ==200

```
xmlhttp.onreadystatechange = function() {  
  if (this.readyState == 4 && this.status == 200)
```

J'ouvrirais une URL avec

```
xmlhttp.open("GET", url, true);
```

J'enverrai la requête

```
xmlhttp.send();
```

Systeme de messagerie privée

Le principe de la messagerie privée est de créer des messages en base de données qui seront écrit par des licenciés pour des licenciés et donc de les relier entre eux via la table User. Ensuite on affichera sous différentes vues des formulaires soit d'écriture soit de lecture. Ci-dessous le processus employé.

Je crée une Entité Message que je lie à celle de User via le sender et le recipient en ManyToOne, en effet un User peut écrire et recevoir plusieurs messages sur le principe du sender->envoi et le recipient-> reçoit :

```
/**
 * @ORM\ManyToOne(targetEntity=User::class, inversedBy="sent")
 * @ORM\JoinColumn(nullable=false)
 */
private $sender;

/**
 * @ORM\ManyToOne(targetEntity=User::class, inversedBy="received")
 * @ORM\JoinColumn(nullable=false)
 */
private $recipient;
```

Coté Entité User on peut s'apercevoir que orphanRemoval=true permettra la suppression des messages envoyés et reçus :

```
/**
 * @ORM\OneToMany(targetEntity=Message::class, mappedBy="sender", orphanRemoval=true)
 */
private $sent;

/**
 * @ORM\OneToMany(targetEntity=Message::class, mappedBy="recipient", orphanRemoval=true)
 */
private $received;
```

Je fais appel au formulaire qui hérite de mon formulaire anti-bot :
Ce formulaire permettra aux licenciés (User) en messagerie privée.

```
class MessageType extends HoneyPotType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        parent::buildForm($builder, $options);
        $builder
            ->add('title', TextType::class, [
                'attr' => [
                    'class' => "form-control",
                ],
                'label' => 'Titre',
            ])
            ->add('message', TextareaType::class, [
                'attr' => [
                    'class' => "form-control",
                ],
            ])
            ->add('recipient', EntityType::class, [
                'class' => User::class,
                'choice_label' => function ($allChoices, $currentChoiceKey) {
                    return $allChoices->getPrenom() . " " . $allChoices->getNom();
                    // return 'form.choice.' . $currentChoiceKey;
                },
                'attr' => [
                    'class' => "form-control"
                ],
                'label' => 'Destinataire',
                'attr' => ['autocomplete' => 'new-password'],
            ])
            ->add('envoyer', SubmitType::class, [
                'attr' => [
                    'class' => "btn btn-primary mt-3"
                ],
            ]);
    }
}
```

Dans mon contrôleur je crée une fonction pour gérer les datas de mon formulaire d'envoi :

```
#[Route('/send', name: 'send')]

public function send(Request $request): Response
{
    $message = new Message;
    $form = $this->createForm(MessageType::class, $message);

    // Permet de récupérer le formulaire dans la requête et de le traiter
    $form->handleRequest($request);

    // On vérifie que le formulaire est soumis et est valide
    // dans cet ordre car on ne peut pas vérifier que le formulaire est valide s'il n'est pas soumis
    if ($form->isSubmitted() && $form->isValid()) {

        // J'instancie ma variable avec pour l'expéditeur le user connecté
        $message->setSender($this->getUser());

        // Je fais appel à l'entity manager
        $em = $this->getDoctrine()->getManager();
        $em->persist($message);
        $em->flush();

        // J'informe le user que le message été envoyé
        $this->addFlash("message", "Message envoyé avec succès.");
        return $this->redirectToRoute("messages");
    }

    return $this->render("messages/send.html.twig", [
        "form" => $form->createView()
    ]);
}
```

Dans le contrôleur je crée une fonction me permettant de gérer la lecture des messages en base, le `isRead` est un booléen que l'on a instancié à `false` par défaut. Quand le message que l'on récupère par son `id` est lu on le passe à `true`

```
$message->setIsRead=true
```

```
#[Route('/read/{id}', name: 'read')]

public function read(Message $message): Response
{
    $message->setIsRead(true);
    $em = $this->getDoctrine()->getManager();
    $em->persist($message);
    $em->flush();

    return $this->render('messages/read.html.twig', compact("message"));
}

#[Route('/delete/{id}', name: 'delete')]

public function delete(Message $message): Response
{
    $em = $this->getDoctrine()->getManager();
    $em->remove($message);
    $em->flush();

    return $this->redirectToRoute("received");
}
```

Ensuite les vues Twig simuleront comme chez un fournisseur de mail classique un système de boîte de réception et d'envoi. J'en profite pour utiliser les nouvelles fonctionnalités offertes par Bootstrap 5, telle que la classe `offcanvas` qui fonctionne sous l'action de l'utilisateur pour afficher des fenêtres et donner du dynamisme au site entre autres chose.

Basé sur les éléments fondamentaux des modals et les partageant, le nouveau composant `offcanvas` est doté d'une toile de fond, d'un défilement du corps et d'un placement configurable.

Système de formulaire de contact

Avant de créer mon formulaire de contact, j'ai décidé de créer un système anti-bots dit du « pot de miel ». Ce pot de miel nous permettra de nous prémunir des attaques par force brut et sera aussi un anti-spam. Pour cela j'ai procédé en 3 étapes :

A. Création d'un formulaire parent

Je crée un formulaire qui me servira de parent pour tous les formulaires à venir :

```
// Notre formulaire sera de la classe ci-dessous et héritera de l'abstractType
class HoneyPotType extends AbstractType

    private LoggerInterface $logger;

    private RequestStack $requestStack;

    public function __construct(LoggerInterface $logger, RequestStack $requestStack)
    {
        $this->$logger = $logger;
        $this->$requestStack = $requestStack;
    }

    // Je déclare mes constantes (convention de nomage en rapport avec ce pourquoi elles sont déclarées)
    // ainsi que convention d'écriture MAG pour les constantes)
    protected const NAME_FORM_ID = "honeypot";

    protected const NAME_FORM_ID = "lastname";

    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        // Je crée mon formulaire avec 2 champs qui serviront à l'envoi des bots
        $builder->add($builder->getForm(), TextType::class, $this->buildFormFieldConfiguration());
        $builder->add($builder->getForm(), TextType::class, $this->buildFormFieldConfiguration());

        // Je ajoute un événement d'écoute sur mon formulaire
        $builder->addEventListener($builder->getForm(), $this->logger, $this->requestStack);
    }

    // Ici ma fonction détaille la construction des champs
    protected function buildFormFieldConfiguration(): array
    {
        return [
            'name' => [
                'autocomplete' => 'off', // pas de remplissage auto
                'label' => ' ', // j'ajoute la formulation de l'utilisateur
                'class' => 'form-control' // je donne une classe à mon input pour pouvoir le restreindre pour l'utilisateur
            ],
            'label' => [
                'class' => 'form-control' // je donne une classe à mon label pour pouvoir le restreindre
            ],
            'data' => ' ', // pas de data
            'required' => false, // pas de relation
            'required' => false // champs non requis
        ];
    }
}
```


B. Création d'un formulaire classique

Je crée un formulaire classique qui hérite de mon extension de formulaire :

```
// Mon formulaire héritera de celui que j'ai créé pour lutter contre les spams
class ContactType extends HoneyPotType

    public function buildForm(FormBuilderInterface $builder, array $options):void
    {
        parent::buildForm($builder, $options);
        $builder
            ->add('nom', TextType::class, [
                "attr" => [
                    "class" => "form-control"
                ]
            ])
            ->add('email', EmailType::class, [
                "attr" => [
                    "class" => "form-control"
                ]
            ])
            ->add('message', TextareaType::class, [
                "attr" => [
                    "class" => "form-control"
                ]
            ])
            ->add('envoyer', SubmitType::class, [
                "attr" => [
                    "class" => "btn btn-primary mt-3"
                ]
            ])
    }
}
```


C. Ecouteurs d'évènements

Je crée un écouteur d'évènements pour pouvoir appeler ma fonction et piéger d'éventuels Bots :

```
// lorsque l'utilisateur click sur envoyer le pre-submit intervient
public static function getSubscribedEvents()
{
    // On passe la fonction ci-dessous qui prend un formEvent et nous permet de récupérer les datas qui ont été saisies
    return [
        FormEvents::PRE_SUBMIT -> 'checkHoneyJar'
    ];
}

// la fonction qui a été appelée
public function checkHoneyJar(FormEvent $event): void
{
    // On récupère la requête courante pour pouvoir avoir l'adresse IP et l'utiliser plus bas
    $request = $this->requestStack->getCurrentRequest();

    if (!$request) {
        return;
    }

    // si la requête courante ne retourne rien alors on récupère les datas via l'event
    $data = $event->getData();
}
```

D. Envoi du formulaire

Le formulaire est envoyé classiquement comme n'importe quel autre :

```
#[Route('/contact', name: 'contact')]
public function index(Request $request, MailerInterface $mailer): Response
{
    $form = $this->createForm(ContactType::class);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $contact = $form->getData();

        $message = (new Email())
            ->From($contact['email'])
            ->To('hibclwTest@test.fr')
            ->html(
                $this->renderView(
                    'email/contact.html.twig',
                    [
                        'contact' => $contact,
                    ]
                )
            );
        $mailer->send($message);

        $this->addFlash('message', 'Le message a bien été envoyé');
        return $this->redirectToRoute('home');
    }

    return $this->render('contact/index.html.twig', [
        'contactForm' => $form->createView()
    ]);
}
```

L'avantage de ce procédé est qu'il est portable et utilisable simplement et rapidement. Les constantes sont facilement identifiables et leurs fonctions sont claires.

Gestion des données personnelles

Dans le cadre de la gestion des données personnelles, les personnes concernées par ces mêmes données personnelles ont un droit de regard, de rectification, de suppression de leurs données et également le droit à l'oubli.

Vous êtes sur votre profil:

Identité: Camille Gu

Email: camille.gu@test.fr

Né(e) le: 07-12-2005





Sexe: F

Catégorie: u18

Equipe: EQ2u18F

Adresse: 52 impasse des mimosas
58490 Ottmarsheim

Téléphone: 907080504



Au survol du profil des toggle explicatifs guide l'utilisateur.

Un système de changement de mot de passe a été mis en place.

```
#[Route('/security/changePassword', name: 'changePassword')]
//Je crée une fonction pour changer de mot de passe,
// pour cela je passe les paramètres dont j'ai besoin entre parenthèse
public function changePassword(User $user = null, EntityManagerInterface $manager,
Request $request, UserPasswordEncoderInterface $passwordEncoder)
{
    //$this->getUser() récupère user en session !
    $user = $this->getUser();

    //Je dis que ma variable ira chercher le formulaire du nom ci-dessous
    $form = $this->createForm(ChangePasswordType::class);

    // Permet de récupérer le formulaire dans la requête et de le traiter
    $form->handleRequest($request);

    // On vérifie que le formulaire est soumis et est valide
    // dans cet ordre car on ne peut pas vérifier que le formulaire
    // est valide s'il n'est pas soumis
    if ($form->isSubmitted() && $form->isValid()) {

        // l'instancie mes variables avec les datas ancien et nouveau
        // mot de passe que je vais chercher dans le form
        $oldPassword = $form->get('oldPassword')->getData();
        $newPassword = $form->get('newPassword')->getData();

        // on compare le mot de passe rentré avec le mot de passe du user en session
        if (password_verify($oldPassword, $user->getPassword())) {

            // Je set le password au User
            $user->setPassword(

                // Je passe mes 2 éléments nécessaire à hashPassword
                $passwordEncoder->hashPassword($user, $newPassword)
            );

            // Je fais appel à l'EntityManagerInterface puis je prépare et flush en base de données
            $manager = $this->getDoctrine()->getManager();
            $manager->persist($user);
            $manager->flush();

            // l'informe le user de la réussite et je le redirige à l'accueil
            $this->addFlash('message', "votre mot de passe a bien été changé !");
            return $this->redirectToRoute('home');

            // Sinon je l'informe du problème et je le laisse sur le formulaire
        } else {
            $this->addFlash('danger', 'votre ancien mot de passe n'est pas bon;');
            return $this->redirectToRoute('changePassword');
        }
    }

    // j'ai besoin de retourner la vue ci dessous ainsi que du formulaire
    return $this->render('security/changePassword.html.twig', [
        'form' => $form->createView(),
        'title' => "Changement du mot de passe"
    ]);
}
```

Un système de réinitialisation de mot de passe a été aussi intégré, cela se passe en 2 temps.

1^{er} temps, je crée une fonction qui dans sa première partie effectuera un travail classique de création et de validation :

```
#[Route('/security/forgottenPassword', name: 'forgottenPassword')]

public function forgottenPassword(EntityManagerInterface $manager, Request $request,
MailerInterface $mailer, TokenGeneratorInterface $tokenGenerator, UserRepository $userRepository): Response
{
    // On instancie la variable $form et on lui dit de récupérer le formulaire ci-dessous
    $form = $this->createForm(ForgottenPasswordType::class);

    // Permet de récupérer le formulaire dans la requête et de le traiter
    $form->handleRequest($request);

    // On vérifie que le formulaire est soumis et est valide
    // dans cet ordre car on ne peut pas vérifier que le formulaire est valide s'il n'est pas soumis
    if ($form->isSubmitted() && $form->isValid()) {

        if ($request->isMethod('POST')) {
            $email = $form->get('emailResetPassword')->getData();

            // On cherche le user par son email
            $user = $userRepository->findOneByEmail($email);

            // si le user par son mail n'existe pas
            if (!$user) {

                // l'informe l'utilisateur
                $this->addFlash('danger', 'Cette adresse mail n\'existe pas');

                // on redirige vers le formulaire de connexion
                return $this->redirectToRoute('app_login');
            }

            // Je génère un Token unique
            $token = $tokenGenerator->generateToken();
        }
    }
}
```


Dans la deuxième partie, la fonction génère un jeton qui sera associé au User, ainsi qu'un email demandant au User de valider la réinitialisation de son mot de passe.

```
try {
    // Je set le token au user
    $user->setResetToken($token);

    // Je fais appel à mon manager pour mettre à jour dans ma Bdd
    $manager = $this->getEntityManager()->getManager();
    $manager->persist($user);
    $manager->flush();

    // Je crée une exception et j'informe l'utilisateur en cas de problème
} catch (\Exception $e) {
    $this->addFlash('warning', 'une erreur est survenue : ' . $e->getMessage());
    return $this->redirectToRoute('app_login');
}

// Je génère une URL qui va comporter la route permettant de changer le mot de passe
$url = $this->generateUrl('resetpassword', ['token' => $token], UrlGeneratorInterface::ABSOLUTE_URL);

// Je crée l'email qui comportera l'URL générée au dessus dans un lien cliquable directement
$message = (new Mail())
    ->from('Hack4Test@test.fr')
    ->to($user->getEmail())
    ->subject('Récupération de mot de passe test')
    ->html(
        "<p>vous recevoir ce mail car vous avez demandé la réinitialisation de votre mot de passe.</p>
        sur le site du handball club de l'Anjou-mhr.</p>
        <p>Si vous n'êtes pas à l'origine de cette action veuillez ne pas tenir compte de ce mail.</p>
        vous pouvez également nous le signaler.</p>
        <p>Pour réinitialiser votre mot de passe : " . "<a href='" . $url . "'>Cliquez ici</a>".
        " . "</p>";
    );

// J'utilise le mailer de symfony pour l'envoi de mon email
$mailer->send($message);

// J'informe le user de la réussite et je le redirige
$this->addFlash('info', 'le mail de récupération de mot passe a bien été envoyé vous pouvez aller le consulter.');
```

```
return $this->redirectToRoute('app_logout');
```

```
return $this->render('security/forgottenPassword.html.twig', [
    'form' => $form->createView(),
    'title' => 'Réinitialisation du mot de passe'
]);
```

Dans un 2^{ème} temps, je crée une fonction qui sera appelée via une route qui inclura le jeton précédemment créé. On vérifie aussi que le jeton de la route correspond à celui dans la BdD du User.

```

a[route('/resetPassword/{token}', name: 'resetPassword')];

public function resetPassword(
    EntityManagerInterface $manager,
    Request $request,
    string $token,
    UserPasswordEncoderInterface $passwordEncoder,
    UserRepositoryInterface $userRepository
) {
    // Je demande à mon manager d'aller chercher dans le repository de User
    $user = $manager->getRepository(User::class)

        // De chercher dans ma BD le resetToken et de l'instancier à ma variable
        ->findOneBy(['resetToken' => $token]);

    //S'il ne correspond pas au User
    if (!$user) {

        // J'informe que le token est inconnu et je retourne sur le formulaire de login
        $this->addFlash('danger', 'token inconnu');
        return $this->redirectToRoute('app_login');
    }

    // Sinon j'instancie ma variable et je lui dis de créer un formulaire sur la base ci dessous
    $form = $this->createForm(NewPasswordType::class);

    // Permet de récupérer le formulaire dans la requête et de le traiter
    $form->handleRequest($request);

    // On vérifie que le formulaire est soumis et est valide
    // dans cet ordre car on ne peut pas vérifier que le formulaire est valide s'il n'est pas soumis
    if ($form->isSubmitted() && $form->isValid()) {

        // Je set le resetToken à null pour mon User
        $user->setResetToken(NULL);

        // J'instancie ma variable avec les datas de mon formulaire
        $password = $form->get('password')->getData();

        // Je set le password à mon User
        $user->setPassword(

            // Je passe mes 2 éléments nécessaire à hashPassword
            $passwordEncoder->hashPassword($user, $password)
        );

        // Je fais appel à mon EntityManagerInterface et je prépare et flush ma base de données
        $manager = $this->getDoctrine()->getManager();
        $manager->persist($user);
        $manager->flush();

        // J'informe le User de la réussite et je le redirige pour se logger
        $this->addFlash('info', 'Votre mot de passe a bien été réinitialisé');
        return $this->redirectToRoute('app_login');
    }

    // J'aurais besoin de retourner la vue ci-dessous
    return $this->render('security/newPassword.html.twig', [
        'form' => $form->createView(),
        'token' => $token
    ]);
}

```

Bibliographie

Documentation Symfony

Documentation Twig

Stackoverflow

OWASP

W3 School

La communauté Symfony

La communauté PHP

Pierre Giraud

Partie traduction

J'ai dû me documenter pour résoudre un problème lors de mes envois d'email et je me suis reposé sur la documentation de Symfony, dont je vous joins la traduction.

Creating & Sending Messages ¶

To send an email, get a `Symfony\Component\Mailer\Mailer` instance by type-hinting `Symfony\Component\Mailer\MailerInterface` and create an `Symfony\Component\Mime\Email` object:

Création et envoi de messages

Pour envoyer un email, je vais aller chercher le mailer de `Symfony/Component`

En instanciant le mailer interface et en créant un nouvel objet email.

```
// src/Controller/MailerController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Mailer\MailerInterface;
use Symfony\Component\Mime\Email;

class MailerController extends AbstractController
{
    /**
     * @Route("/email/")
     */
    public function sendEmail(MailerInterface $mailer): Response
    {
        $email = (new Email())
            ->from('hello@example.com')
            ->to('you@example.com')
            //->cc('cc@example.com')
            //->bcc('bcc@example.com')
            //->replyTo('fabien@example.com')
            //->priority(Email::PRIORITY_HIGH)
            ->subject('Time for Symfony Mailer!')
            ->text('Sending emails is fun again!')
            ->html('<p>See Twig integration for better HTML integration!</p>');

        $mailer->send($email);

        // ...
    }
}
```

That's it! The message will be sent via the transport you configured.

C'est ça ! Le message sera envoyé suivant votre configuration.

Email Addresses ¶

All the methods that require email addresses (`from()`, `to()`, etc.) accept both strings or address objects:

Adresses email

Toutes les méthodes qui nécessite une adresse email (`de()`, `à()`, etc.) accepte les deux, soit une chaîne de caractère soit un objet.

```
// ...
use Symfony\Component\Mime\Address;

$email = (new Email())
    // email address as a simple string
    ->from('fabien@example.com')

    // email address as an object
    ->from(new Address('fabien@example.com'))

    // defining the email address and name as an object
    // (email clients will display the name)
    ->from(new Address('fabien@example.com', 'Fabien'))

    // defining the email address and name as a string
    // (the format must match: 'Name <email@example.com>')
    ->from(Address::create('Fabien Potencier <fabien@example.com>'))

// ...
}
```

Use `addTo()`, `addCc()`, or `addBcc()` methods to add more addresses:

Utiliser `addTo()` `addCc()`, or `addBcc()` pour ajouter plus d'adresses :

```
$email = (new Email())
    ->to('foo@example.com')
    ->addTo('bar@example.com')
    ->cc('cc@example.com')
    ->addCc('cc2@example.com')

// ...
}
```

Alternatively, you can pass multiple addresses to each method.

Une autre possibilité est de passer plusieurs adresses dans chaque méthode :

```
$toAddresses = ['foo@example.com', new Address('bar@example.com')];

$email = (new Email())
    ->to(...$toAddresses)
    ->cc('cc1@example.com', 'cc2@example.com')

    // ...
}
```

Message Headers ¶

Messages include a number of header fields to describe their contents. Symfony sets all the required headers automatically, but you can set your own headers too. There are different types of headers (Id header, Mailbox header, Date header, etc.) but most of the times you'll set text headers:

En-têtes de messages

Les messages incluent un certain nombre de champs d'en-têtes pour décrire leur contenu. Symfony définit les champs requis automatiquement, mais vous pouvez créer vos propres en-têtes aussi. Il y a différents types d'en-têtes (en-tête d'ID, en-tête de boîte aux lettres, en-tête de date, etc.), mais le plus souvent vous définirez des en-têtes de texte :

Message Contents ¶

The text and HTML contents of the email messages can be strings (usually the result of rendering some template) or PHP resources:

Contenu de message

Le texte et le contenu HTML des messages peut être une chaîne de caractère (couramment le résultat d'un rendu d'un modèle) ou d'une ressource PHP :

```
$email = (new Email())
    // ...
    // simple contents defined as a string
    ->text('Lorem ipsum...')
    ->html('<p>Lorem ipsum...</p>')

    // attach a file stream
    ->text(fopen('/path/to/emails/user_signup.txt', 'r'))
    ->html(fopen('/path/to/emails/user_signup.html', 'r'))
}
```

File Attachments ¶

Use the `attachFromPath()` method to attach files that exist on your file system:

Fichiers joints

Utilisez la méthode `attachFromPath()` pour joindre des fichiers qui existent dans les fichiers du système.

```
$email = (new Email())
// ...
->attachFromPath('/path/to/documents/terms of use.pdf')
// optionally you can tell email clients to display a custom name for the file
->attachFromPath('/path/to/documents/privacy.pdf', 'Privacy Policy')
// optionally you can provide an explicit MIME type (otherwise it's guessed)
->attachFromPath('/path/to/documents/contract.doc', 'Contract', 'application/msword')
;
```

Alternativement vous pouvez utiliser la méthode `attach()` pour joindre des contenus à partir d'un flux :

Sinon on peut utiliser la méthode `attach()` qui consiste à joindre un flux :

```
$email = (new Email())
// ...
->attach(fopen('/path/to/documents/contract.doc', 'r'))
;
```

Embedding Images ¶

If you want to display images inside your email, you must embed them instead of adding them as attachments. When using Twig to render the email contents, as explained [later in this article](#), the images are embedded automatically. Otherwise, you need to embed them manually.

First, use the `embed()` or `embedFromPath()` method to add an image from a file or stream:

Intégration d'images

Si vous voulez disposer des images dans votre email, vous devez les intégrer au lieu de les ajouter en pièces-jointes. Quand on utilise Twig pour le rendu du contenu des emails, comme expliqué plus loin, les images sont intégrées automatiquement. Sinon, vous devez les intégrer manuellement.

Premièrement, utiliser la méthode `embed()` ou `embedFromPath` pour ajouter une image depuis un fichier ou un flux :

```
$email = (new Email())
// ...
// get the image contents from a PHP resource
->embed(fopen('/path/to/images/logo.png', 'r'), 'logo')
// get the image contents from an existing file
->embedFromPath('/path/to/images/signature.gif', 'footer-signature')
;
```

The second optional argument of both methods is the image name ("Content-ID" in the MIME standard). Its value is an arbitrary string used later to reference the images inside the HTML contents:

Le deuxième argument optionnel des 2 méthodes est le nom de l'image (« Content-ID » dans la norme MIME). Sa valeur est une chaîne de caractères arbitraire qui sera utilisée plus tard comme référence d'images à l'intérieur du contenu HTML :

```
$email = (new Email())
// ...
->embed(fopen('/path/to/images/logo.png', 'r'), 'logo')
->embedFromPath('/path/to/images/signature.gif', 'footer-signature')
// reference images using the syntax 'cid:' + "image embed name"
->html(' ...  ...')
```

Remerciements

Je remercie mes formateurs de chez Elan Formation pour leurs disponibilités, leurs enthousiasmes quand ils transmettent leurs connaissances.

Gilles Muess et son côté tout est possible sans difficultés.

Stéphane Smail et sa bienveillance en toutes circonstances

Mes camarades de formation, Béatrice, Victor, Melvin, Alexis, Thomas, Killian, Florian, Martin et Terence.

Enfin ma famille, Nathan et Camille mes enfants qui m'ont supportés presque sans rien dire et mon épouse Sabrina sans qui rien ne serait possible dans l'équipe que nous formons.

Axes d'amélioration

Crypter les messages en BdD

Ajouter Full Calendar

Créer une newsletter

Empêcher l'inscription d'un participant à plusieurs covoitages en même temps

Annexes

A. Contact avec la CNIL



Bonjour Monsieur Jean-Philippe Gurak,

Nous vous remercions de nous avoir contactés.

Vous souhaitez réaliser un site internet pour votre club de sport dans lequel seraient publiés des documents sur lesquels figurent les données personnelles des licenciés. Vous vous interrogez sur les mesures à mettre en oeuvre afin d'être conforme au Règlement général sur la protection des données.

Je vous informe que les règles de la protection des données s'articulent autour de 5 grands principes

- **Le principe de finalité** : le responsable d'un fichier ne peut enregistrer et utiliser des informations sur des personnes physiques que dans un but bien précis, légal et légitime ;
- **Le principe de proportionnalité et de pertinence** : les informations enregistrées doivent être pertinentes et strictement nécessaires au regard de la finalité du fichier ;
- **Le principe d'une durée de conservation limitée** : il n'est pas possible de conserver des informations sur des personnes physiques dans un fichier pour une durée indéfinie. Une durée de conservation précise doit être fixée, en fonction du type d'information enregistrée et de la finalité du fichier ;
- **Le principe de sécurité et de confidentialité** : le responsable du fichier doit garantir la sécurité et la confidentialité des informations qu'il détient. Il doit en particulier veiller à ce que seules les personnes autorisées aient accès à ces informations ;

- [Les droits des personnes](#)

Afin de vous aider à comprendre les obligations en matière de protection des données personnelles, avec le Règlement européen sur la protection des données (RGPD), la CNIL, en partenariat avec la Banque Publique d'Investissement BPIFrance, met à votre disposition un guide pratique de sensibilisation.

Ce guide vous rappelle les grands principes du RGPD, détaille un plan d'action en 4 étapes et vous fournit des fiches pratiques couvrant les principaux fichiers mis en oeuvre dans la plupart des TPE-PME.

Vous trouverez ce guide en version numérique ou .pdf sur notre site internet à l'adresse <https://www.cnil.fr/fr/la-cnil-et-bpifrance-sassocient-pour-accompagner-les-tpe-et-pme-dans-leur-appropriation-du-reglement>.

Vous trouverez également de nombreuses publications et documentations sur notre site, je vous rappelle également que nos téléconseillers répondent gratuitement à vos questions les lundis, mardis, jeudis et vendredis de 10h à 12h en appelant au 01 53 73 22 22 (prix d'un appel local).

Cordialement,
Catherine Planel

Les informations que vous nous communiquez font l'objet d'un enregistrement informatique destiné à faciliter nos échanges. Elles sont destinées uniquement aux services en charge de répondre à votre demande. Vous pouvez en obtenir une copie ou la rectification par simple réponse à ce courriel. Consultez notre page sur [vos droits](#) pour plus d'informations.

Retrouvez toute l'actualité de la CNIL sur [cnil.fr](https://www.cnil.fr)

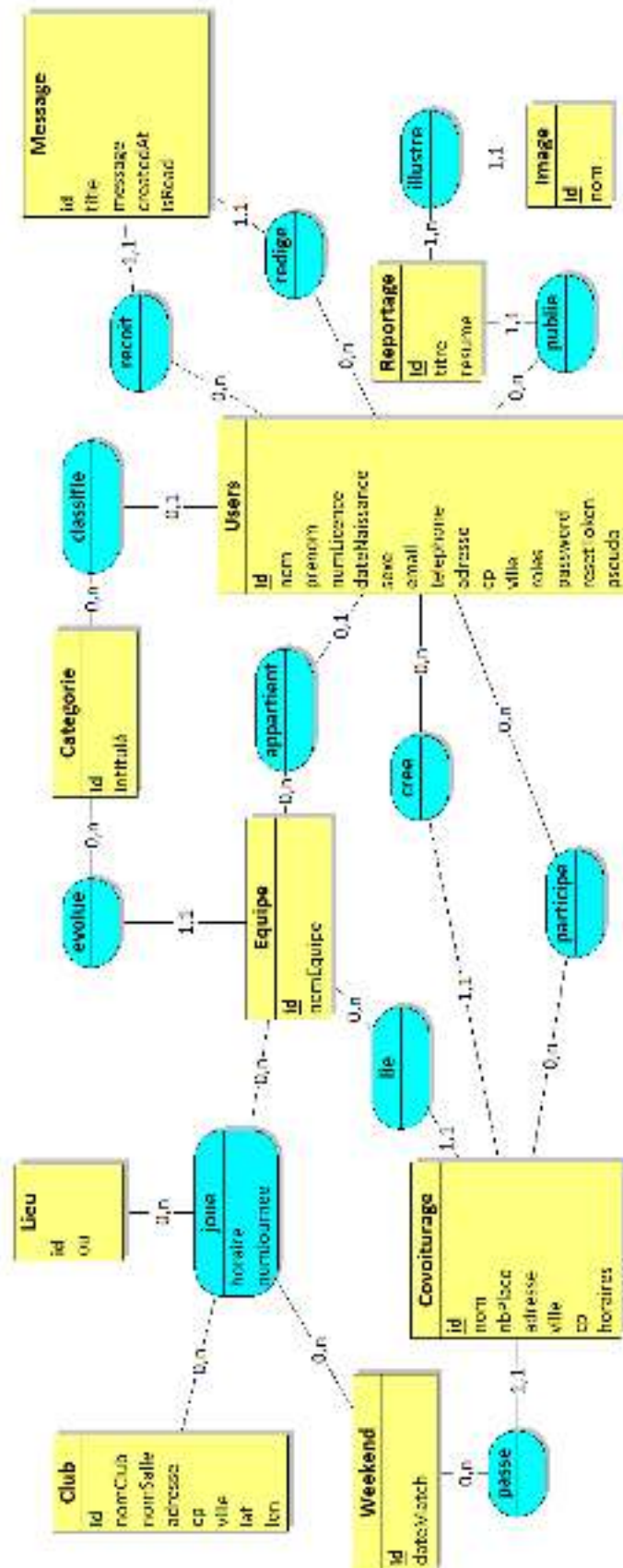
Tél : 01 53 73 22 22

Commission Nationale de l'Informatique et des Libertés
3 Place de Fontenoy
TSA 80715
75334 PARIS CEDEX 07

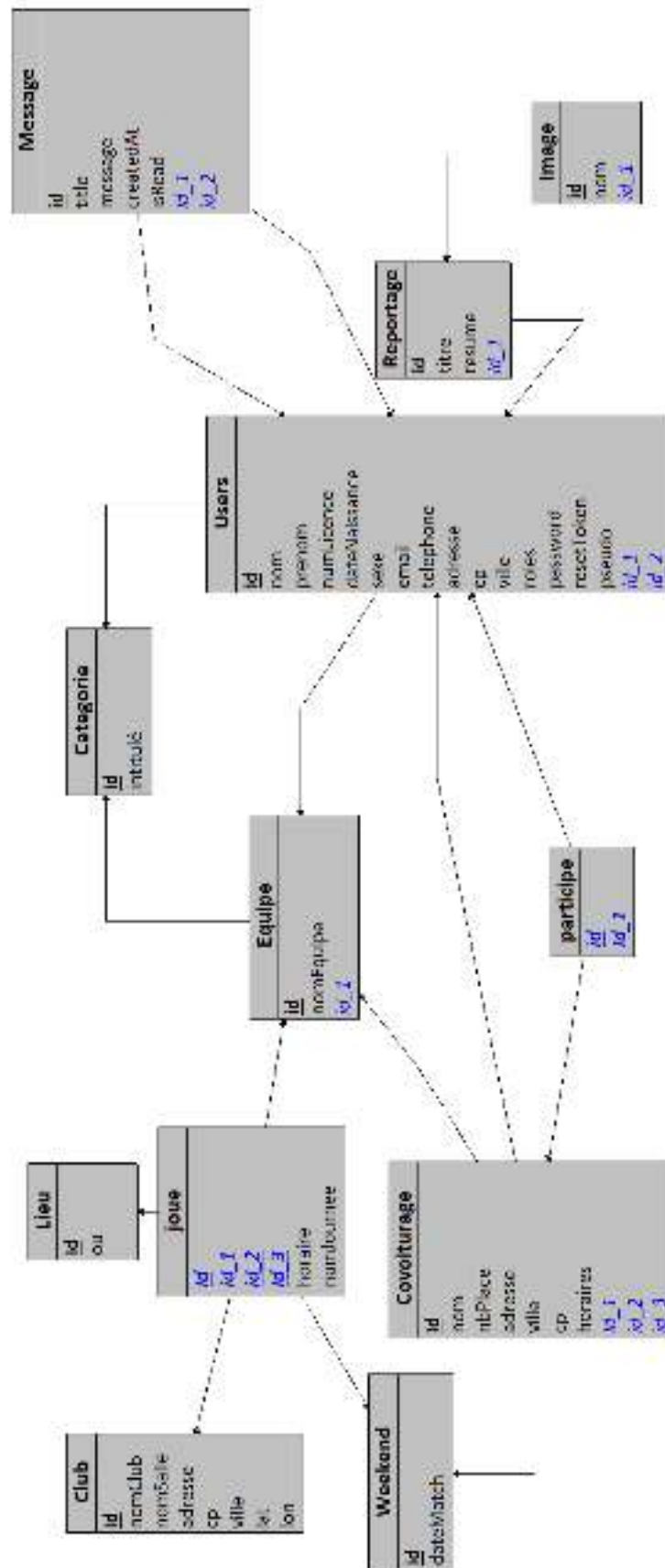
B. Trello



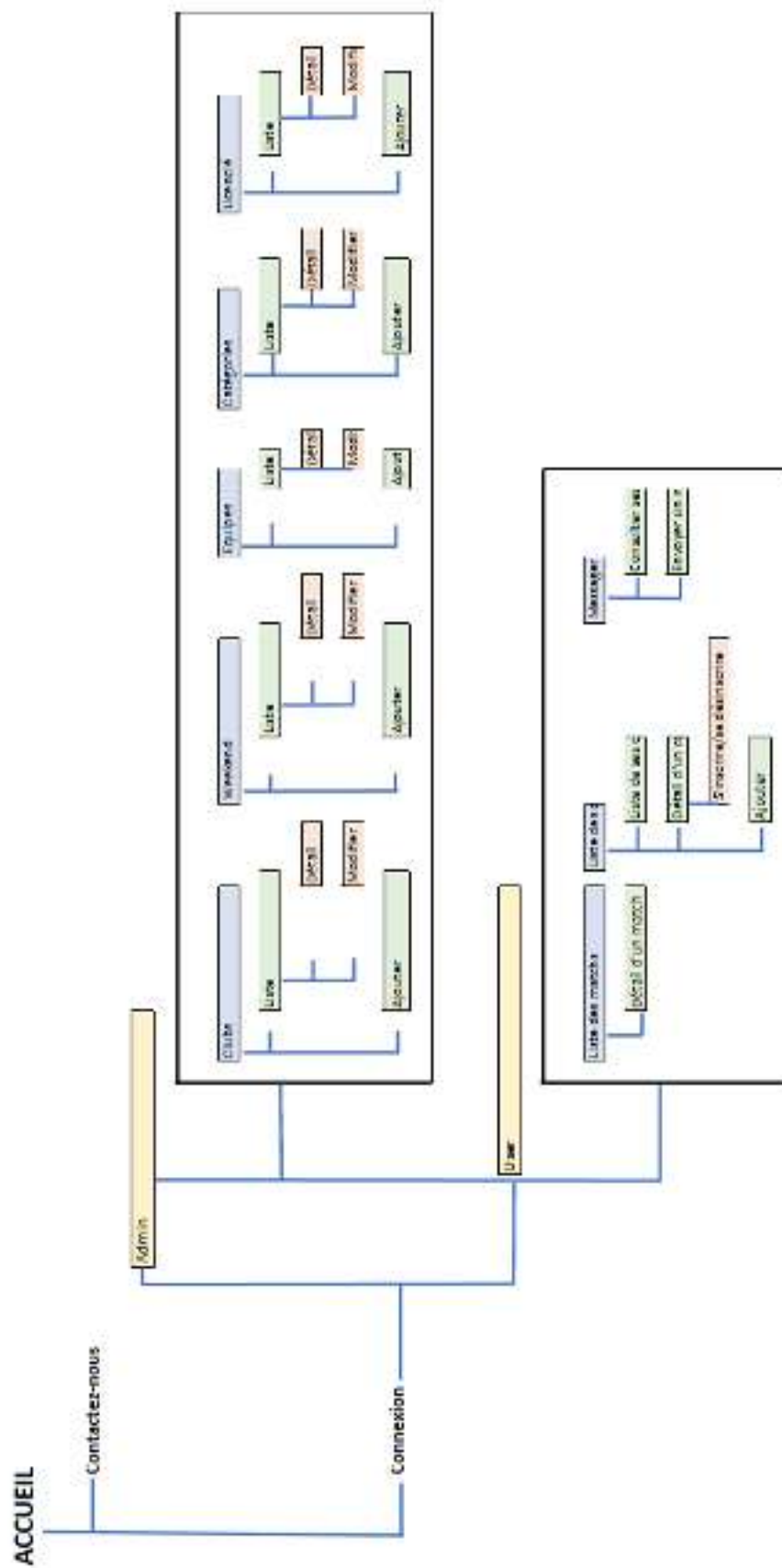
C. MCD



D. MLD

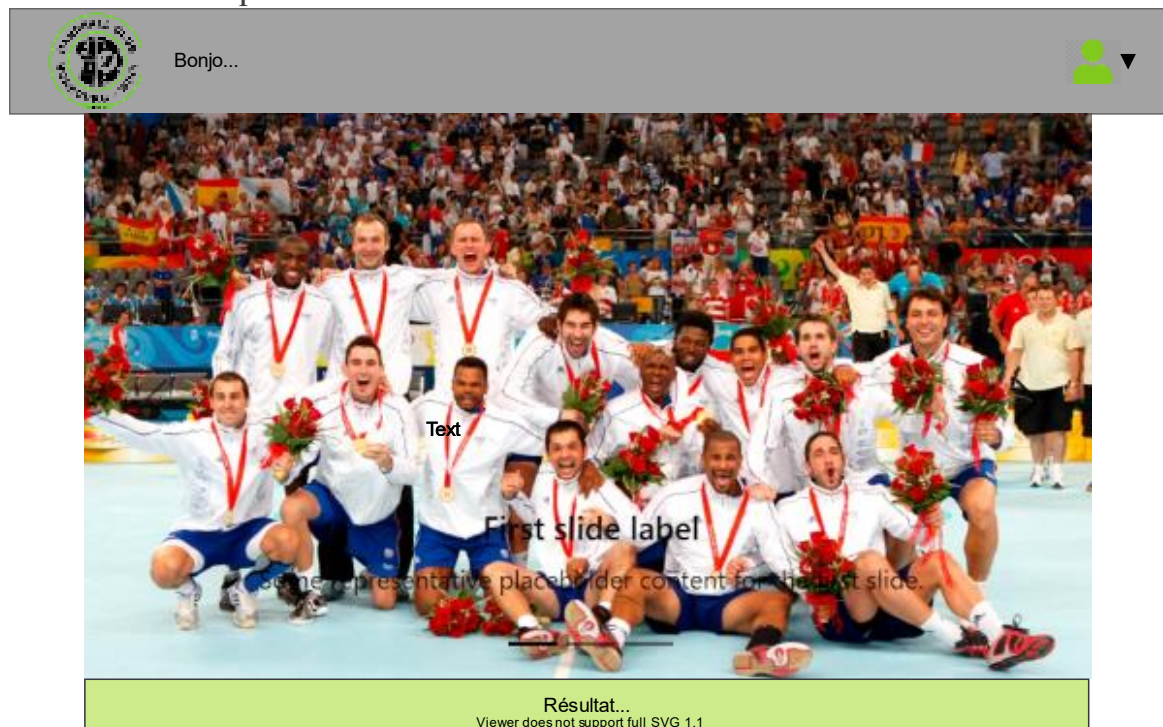


E. Arborescence du site

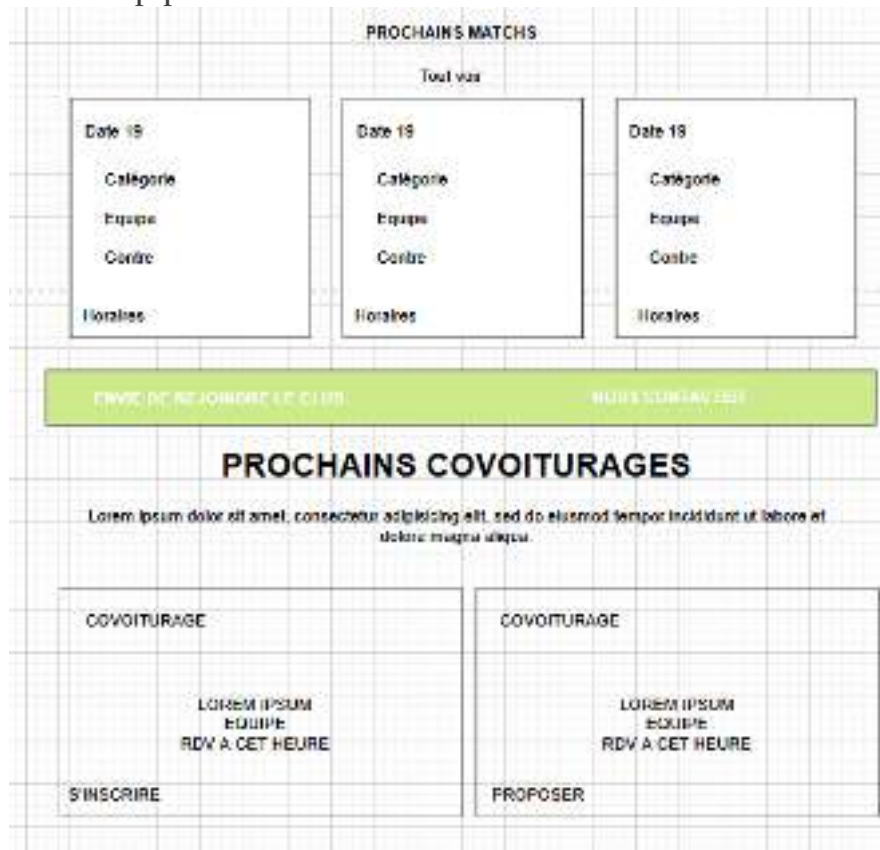


F. Maquettage

1. Desktop accueil visiteur



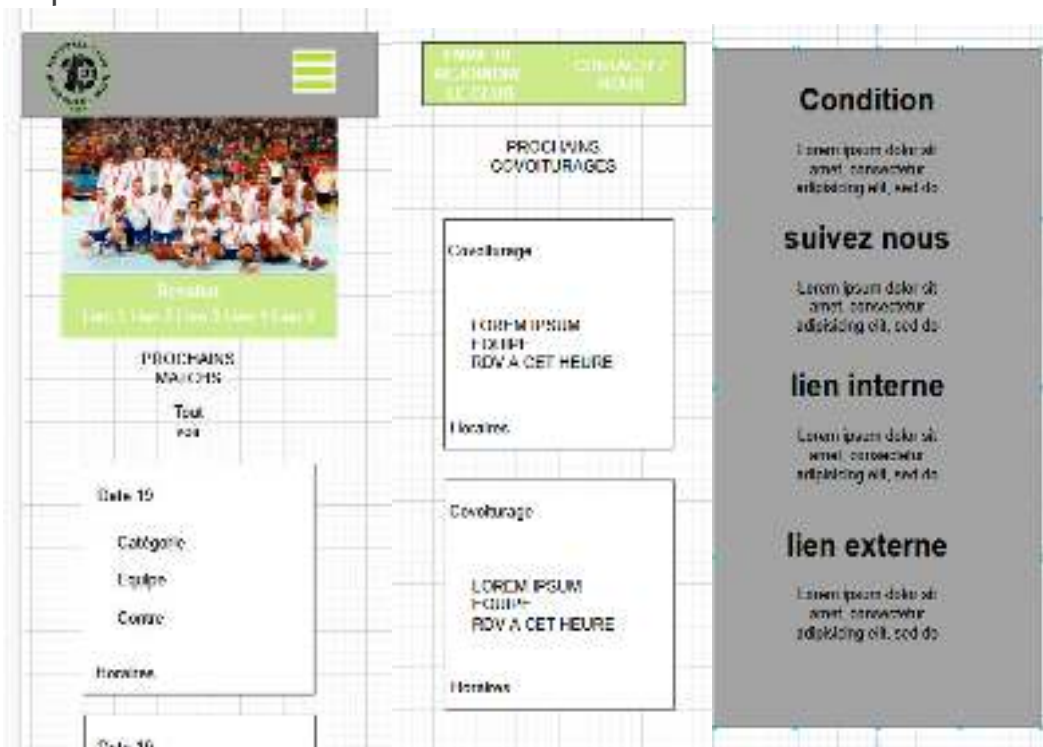
2. Desktop prochains match



3. Desktop footer



4. Mobile



G. Cartographie

1. Mise en place de la cartographie :

```
// 9,rue de Lorraine 68180 Horbourg-Wihr
var startlat = 48.085461884393816;
var startlon = 7.3907848117717885;

var options = {
  center: [startlat, startlon],
  zoom: 12
};

document.getElementById('lat').value = startlat;
document.getElementById('lon').value = startlon;

var map = L.map('map', options);
var nzoom = 12;

L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token={accessToken}', {
  attribution: 'Map data &copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> <a href="https://www.mapbox.com/">Mapbox</a>',
  maxZoom: 24,
  id: 'mapbox/satellite-streets-v11', // les détails de la carte comme le type d'affichage peut être changé via les param sur le site mapbox(https://docs.mapbox.com/api
  tileSize: 512,
  zoomOffset: -1,
  accessToken: 'pk.eyJ1IjoianRmdSIsImEiOiIjZjZlYjZjZDljcwaxLHNdsc6c3kpyaXRhXzI0Lz02ODQ0bDh9LCKvO2hA' // Le token est récupéré sur le site mapbox après inscription
}).addTo(map);
```


2. Mise en place des marqueurs et de l'adresse :

```
var myMarker = L.marker([startlat, startlon], { title: "Coordinates", alt: "Coordinates", draggable: true }).addTo(map).on('dragend', function() {  
    var lat = myMarker.getLatLng().lat.toFixed(8);  
    var lon = myMarker.getLatLng().lng.toFixed(8);  
    var czoom = map.getZoom();  
    if (czoom < 18) { nzoom = czoom + 2; }  
    if (nzoom > 18) { nzoom = 18; }  
    if (czoom != 18) { map.setView([lat, lon], nzoom); } else { map.setView([lat, lon]); }  
    document.getElementById('lat').value = lat;  
    document.getElementById('lon').value = lon;  
    myMarker.bindPopup("Lat " + lat + "<br />Lon " + lon).openPopup();  
});  
  
function chooseAddr(lat1, lng1) {  
    myMarker.closePopup();  
    map.setView([lat1, lng1], 18);  
    myMarker.setLatLng([lat1, lng1]);  
    lat = lat1.toFixed(8);  
    lon = lng1.toFixed(8);  
    document.getElementById('lat').value = lat;  
    document.getElementById('lon').value = lon;  
    myMarker.bindPopup("Lat " + lat + "<br />Lon " + lon).openPopup();  
    // marker.bindPopup("<b>landball club</b><br>9,rue de Lorraine</br>").openPopup();  
}
```

3. Création du champ adresse et de la requête URL :

```
function myfunction(arr) {  
    var out = "<br />";  
    var i;  
  
    if (arr.length > 0) {  
        for (i = 0; i < arr.length; i++) {  
            out += "<div class='address' title='Cliquez pour voir sur la carte' onclick='chooseAddr(" + arr[i].lat + ", " + arr[i].lon + "'>";  
        }  
        document.getElementById('results').innerHTML = out;  
    } else {  
        document.getElementById('results').innerHTML = "Désolé, aucun résultat...";  
    }  
}  
  
function addr_search() {  
    var inp = document.getElementById("addr");  
    var xmlhttp = new XMLHttpRequest();  
    var url = "https://nominatim.openstreetmap.org/search?format=json&limit=3&q=" + inp.value;  
    xmlhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200) {  
            var myArr = JSON.parse(this.responseText);  
            myfunction(myArr);  
        }  
    };  
    xmlhttp.open("GET", url, true);  
    xmlhttp.send();  
}
```