



DOSSIER-PROJET

Nom de naissance ▶ Champromis
Nom d'usage ▶ Champromis
Prénom ▶ Thomas
Adresse ▶ 25 Rue Ballainvilliers, Clermont-Ferrand 63000

Titre professionnel visé

Concepteur Développeur d'Applications - Niveau II

Sommaire

(à adapter selon le projet)

Remerciements	3
Résumé du projet en anglais	4
Liste des compétences du référentiel	5
Cahier des charges ou expressions des besoins de l'application à développer	6
Spécifications techniques	17
Réalisations	18
Conclusions	57

REMERCIEMENTS

Je tiens d'abord à remercier l'établissement Simplon qui m'a permis de me former tout au long de l'année et m'a aidé à trouver une entreprise pour mon alternance.

La formation a été riche et intensive. Nous avons appris à prendre plus de recul sur notre métier et à travailler en groupe.

Je tenais également à remercier nos formateurs David Rigaudie et Yann Bouilhac. Ils ont formé une aide précieuse avec une pédagogie qui me correspondait beaucoup. Ils ont été très disponibles et à l'écoute durant toute la durée de la formation.

Je souhaite évidemment remercier mes camarades. Nous formions un groupe soudé avec des profils très variés. Nous avons beaucoup échangé et j'ai pu me confronter différents points de vue, résoudre des problèmes, être aidé et surtout perfectionner mon travail et ma manière de l'appréhender.

Je voudrais aussi remercier l'entreprise Itarverne qui m'a accueilli durant 1 an et donné l'opportunité de mettre en application mes compétences sur des projets réels, au contact de vrais clients.

Pour finir je remercie toute l'équipe pédagogique de Simplon qui nous a bien encadré malgré la distance. Ils ont aussi été un élément essentiel dans ma montée en compétences.

RÉSUMÉ DU PROJET EN ANGLAIS

The project is fictitious, it consists of creating a football quiz site with open-ended questions where the objective is to list all the expected answers.

This concept is widely used on the "Sporcle" quiz site. The aim is to provide a way to challenge your knowledge of the world of football with tools classified into different categories.

Users who identify themselves on the site have the possibility to like their favorite quizzes, record their scores and create their own quiz.

I first researched existing sites to determine my functionality and design. I had to do a lot of research to enter data from about 20 quizzes that cover all of my categories.

I then used the PHP framework Symfony to develop the project, I also put into practice all the knowledge learned during the last few months.

The result is satisfactory, despite some difficulties, I took great pleasure in combining 2 of my passions: web development and football.

LISTE DES COMPÉTENCES DU RÉFÉRENTIEL

- **Maquetter une application :**
- **Développer des composants d'accès aux données:**
- **Concevoir une base de données:**
- **Mettre en place une base de données:**
- **Concevoir une application:**
- **Développer des composants métier:**
- **Construire une application organisée en couches:**
- **Développer une application mobile:**
- **Préparer et exécuter les plans de tests d'une application:**

CAHIER DES CHARGES OU EXPRESSIONS DES BESOINS DE L'APPLICATION À DÉVELOPPER

Culture Foot

1 . Présentation de l'entreprise :

CultureFoot est une entreprise fictive qui a pour ambition de devenir une référence en terme de quiz sur le monde du football.

2. Naissance du projet :

L'idée est née lors du premier confinement en Mars 2020, mon ami et moi voulions défier nos connaissances sur le football pour enfin savoir lequel de nous deux était le plus cultivé sur l'univers de ce sport.

Nous avons donc découvert Sporcle, un site qui référence des milliers de quiz (tout thème confondu) avec un modèle ludique qui consiste à deviner une liste de réponses possibles à une question donnée.



On voit ci-dessous un quiz qui consiste à deviner les pays gouvernés par chaque chef d'état listés :



On peut en conclure que Sporcle est un site très bien pensé mais avec une quantité de quiz et de thématiques très dense . Le design est également peu adapté à l'univers du jeu en ligne.

L'idée est donc de s'inspirer des forces de Sporcle pour créer un site uniquement basé sur le football, avec une interface utilisateur beaucoup mieux adaptée au thème du jeu et du football.

3. Les objectifs du site :

- Permettre de jouer à des quiz sur le football divisés en 4 catégories
- Proposer une expérience utilisateur agréable et adaptée à la thématique du Quiz
- Permettre à un utilisateur de créer un compte pour liker ses quiz préférés mais aussi de créer ses propres quiz et d'accéder à ses statistiques de jeu.

4. La cible de l'application :

Le site cible les passionnés de football souhaitant défier leurs connaissances sur l'histoire et l'actualité de ce sport.

On peut imaginer un public assez jeune (entre 15 et 35 ans) et sensible aux exploits de leurs joueurs préférés.

5. Arborescence de l'application :

- Une page d'accueil
- Une page qui recense tous les quiz
- Une page qui recense les quiz de chaque catégories
- Une page de création de quiz (Intitulé, chrono, catégorie, réponses attendues, indices)
- Une page qui recense les quiz de la communauté
- Une page 'Mon compte' qui recense les infos du profil (Nom, Prénom, pourcentages de réussite, quiz créés, nombre de quiz parfaits, quiz likés)
- Une page de connexion
- Une page d'inscription
- Une page qui recense les quiz les plus joués depuis 1 semaine
- Une page qui recense les quiz les plus difficiles

6. Contenu des quiz :

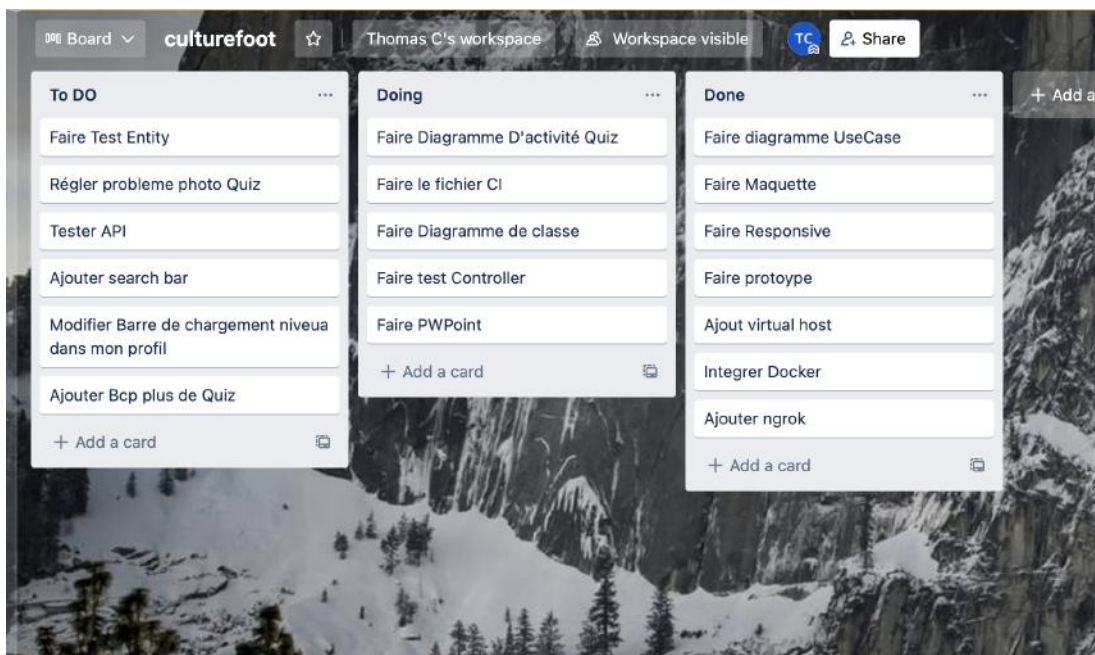
- J'ai trouvé les informations sur mes quiz en faisant des recherches sur des sites spécialisés sur le football : L'Équipe, TransferMarkt,...
- J'ai également rempli quelques quiz à l'aide de mes connaissances •

7 Gestion De Projet

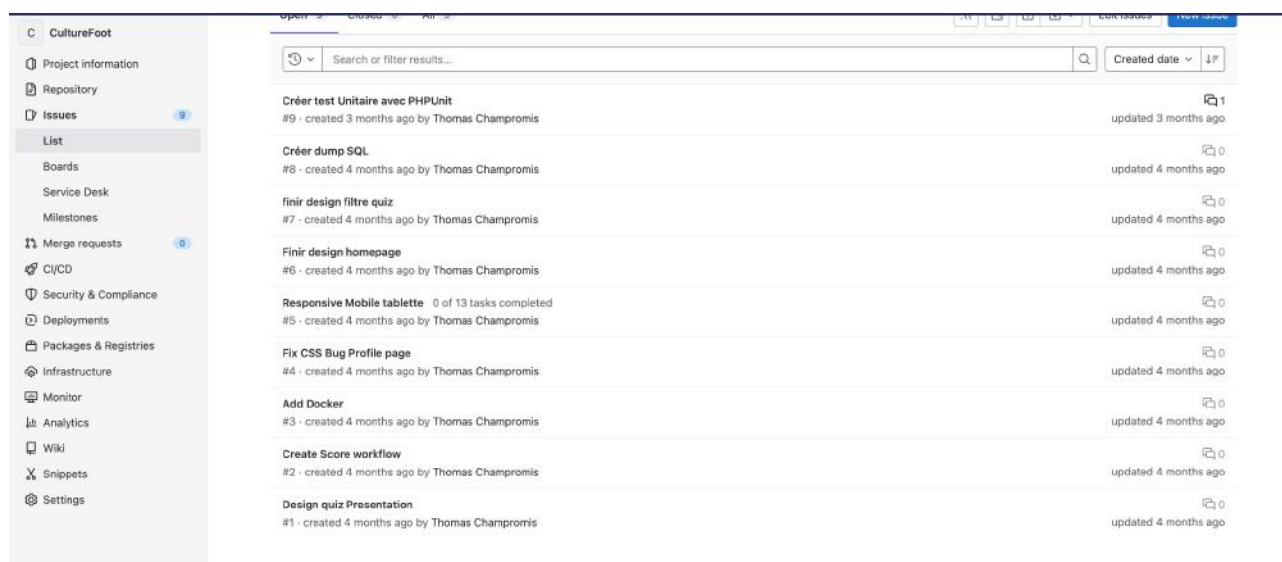
J'ai utilisé l'outil Trello pour organiser mon projet, J'ai découpé mon travail en différentes tâches que j'ai placé dans 3 colonnes en fonction de la progression de mon travail :

- "A faire"
- "En cours"
- "Terminé"

J'ai mis à jour cet outil régulièrement pour avoir un suivi en temps réel du projet . Cela permet également de prendre de la hauteur et surtout de gagner en efficacité .



J'ai également utilisé le système d'"Issue" présent sur mon répo Gitlab pour associer des problématiques à des commits afin de retrouver plus facilement les solutions techniques apportées à chaque problématique



RÉALISATIONS

LA CONCEPTION :

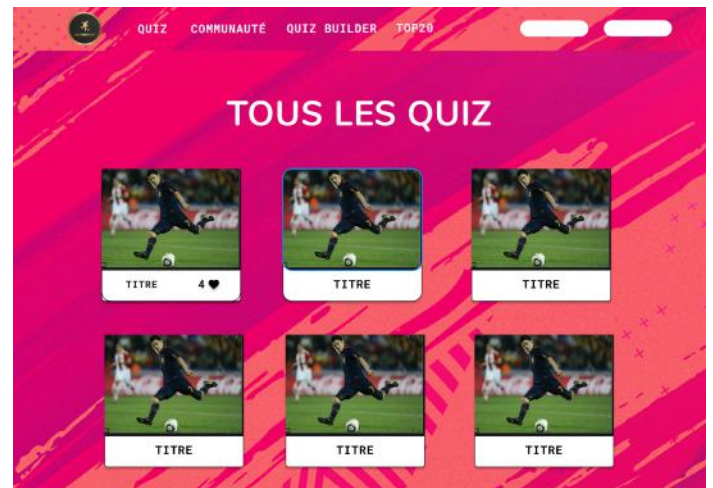
Le Maquettage :

J'ai décidé d'opter pour un site coloré qui colle bien avec la cible du projet. Le style est épuré (material design) pour afficher une UI/UX intuitive. Pour le maquettage, j'ai utilisé l'outil Figma ,un outil d'édition graphique, qui permet notamment de réaliser des prototypes.

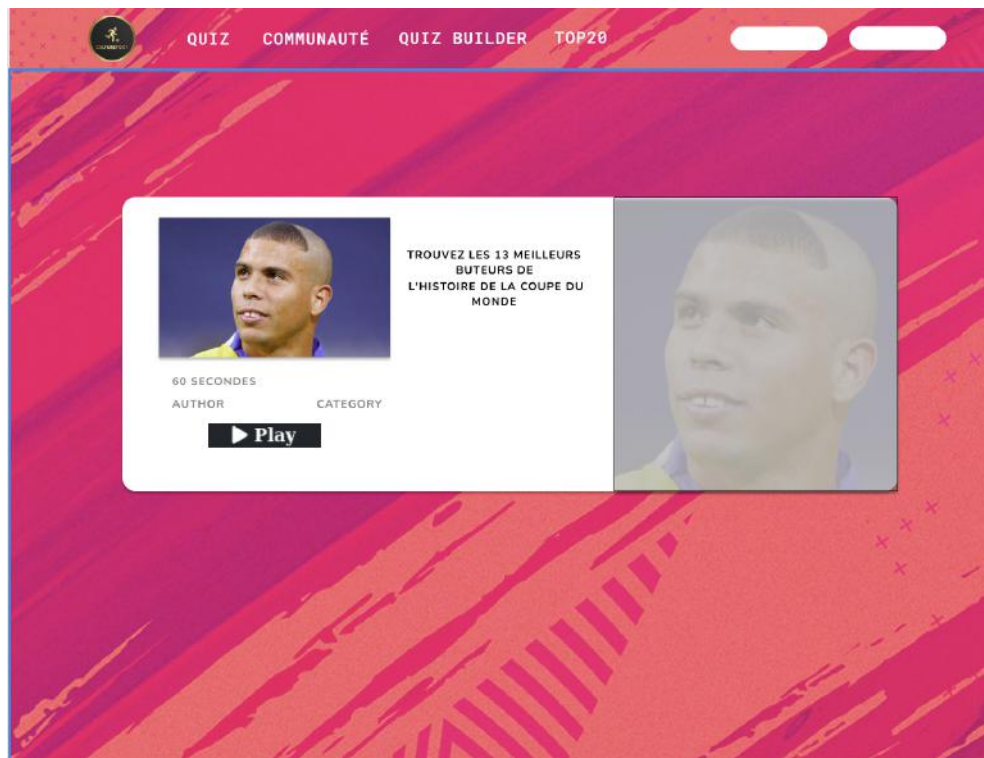
1 / Homepage



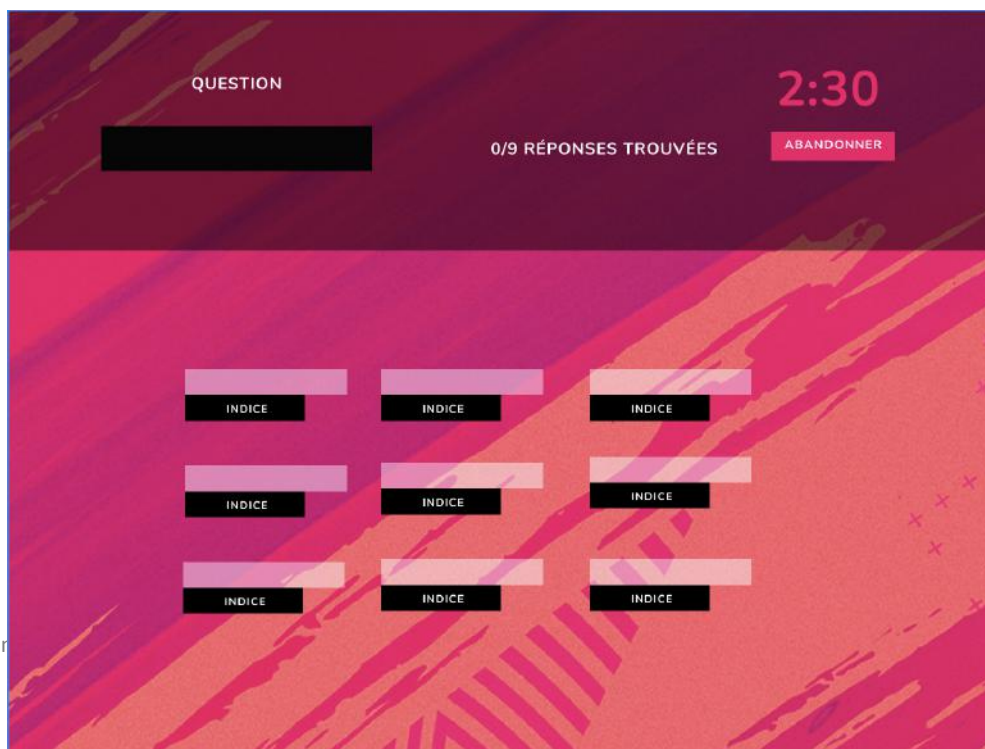
2 / Page des Quiz



3 / Détail Quiz



4 / Affichage du Jeu



5 / Affichage du score

VOTRE SCORE X SUR X (XX %)

★★★★★

JEUNE ESPOIR, ENTRAÎNES TOI ENCORE
+ 450 XP

Numero	Réponse	Résultat	% des autres joueurs
1	Ronaldo	XP	0%
2	Ronaldo	XP	12%
3	Ronaldo	XP	12%
4	Ronaldo	XP	12%
5	Ronaldo	XP	12%
6	Ronaldo	XP	12%
7	Ronaldo	XP	12%
8	Ronaldo	XP	12%

Numero	Réponse	Résultat	% des autres joueurs
1	Ronaldo	XP	0%
2	Ronaldo	XP	12%
3	Ronaldo	XP	12%
4	Ronaldo	XP	12%
5	Ronaldo	XP	12%
6	Ronaldo	XP	12%
7	Ronaldo	XP	12%
8	Ronaldo	XP	12%

6 / Page Profil

DAVID ROBIN

300 / 2000 XP
1700 XP RESTANT

NIVEAU

62 62 62 62

LES QUIZ QUE J'AI LIKÉ :

7 / Formulaire de Login

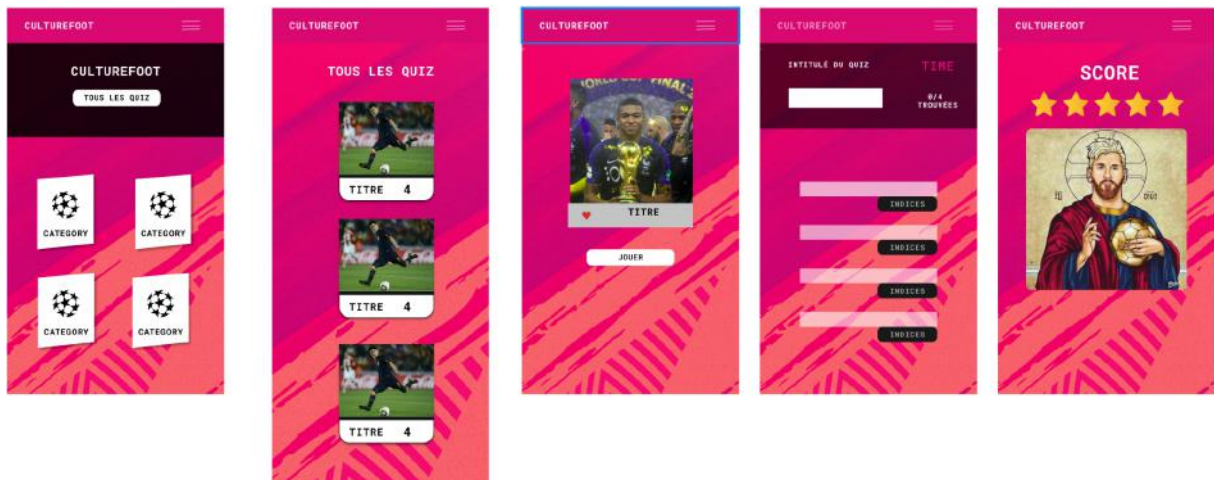
SE CONNECTER

NAME

PASSWORD

SE CONNECTER

8 / Responsive



→ Diagramme UML :

Ils vont me permettre de modéliser la structure et/ou le comportement des données. En effet il s'agit d'un langage de modélisation unifié permettant à "n'importe qui" de comprendre le modèle plus facilement qu'en lisant des paragraphes d'explication.

1) Diagramme de cas d'utilisation (use case en Anglais)

Je réalise le diagramme de cas d'utilisation de l'application.

Ce diagramme permet de voir l'interaction d'un joueur anonyme, connecté ou administrateur avec l'application via différents cas d'utilisations (représenté par des cercles avec une action).

Les acteurs et les cas d'utilisation sont reliés entre eux par différents types de relation.

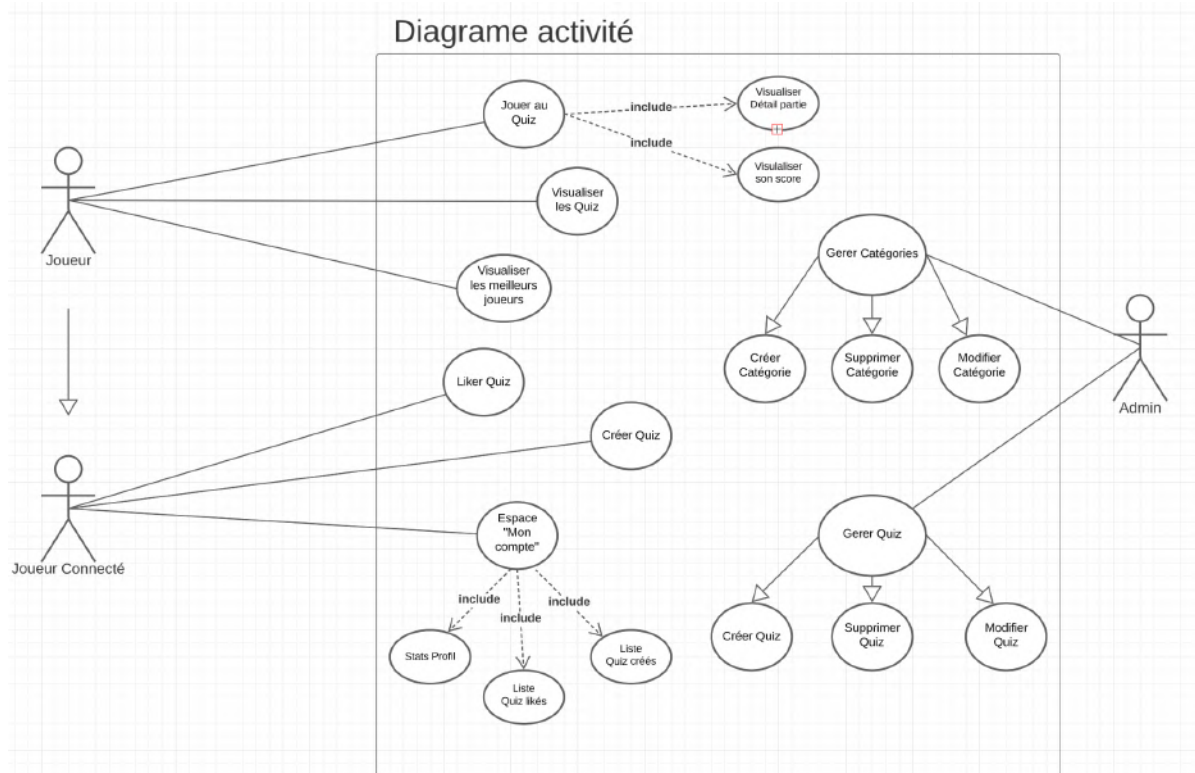
- Relation d'association : représentée par un trait simple. Le chemin de communication

entre un acteur et un cas d'utilisation.

- L'inclusion : représentée par une flèche en pointillé avec le terme « include ». Cela implique qu'avant de réaliser la première action il faut que l'action incluse soit réalisée.

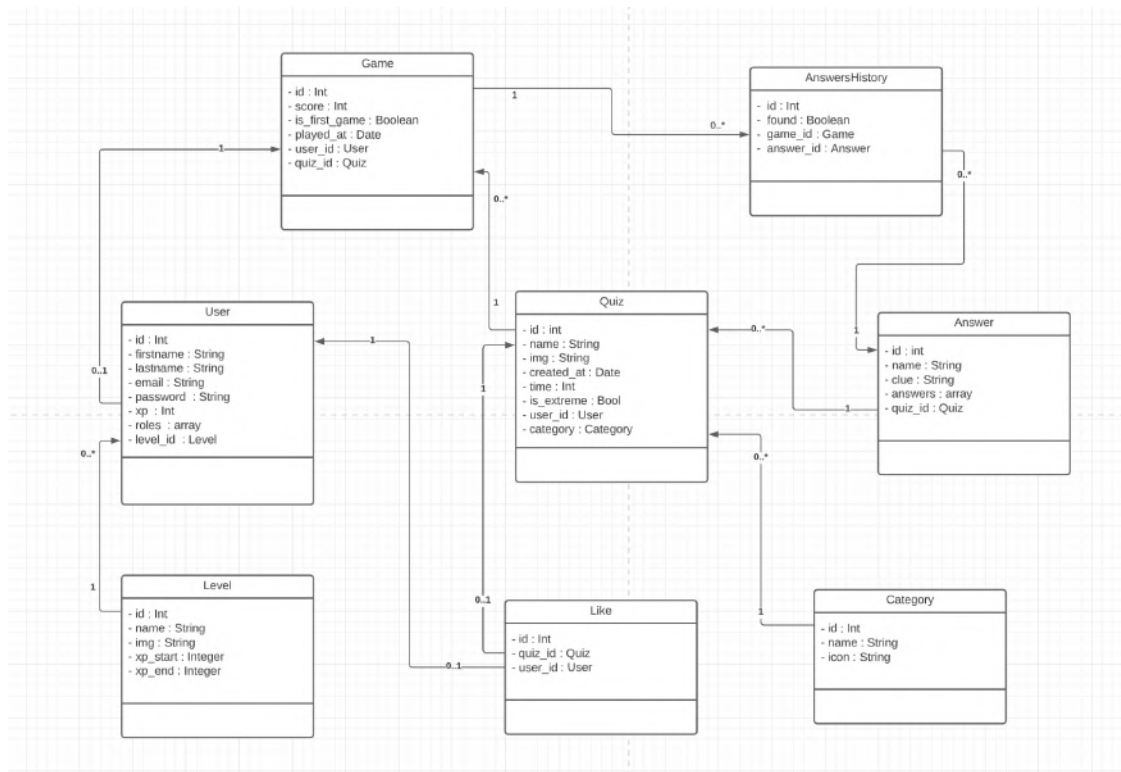
- La généralisation : Représentée par une flèche en forme de triangle et un trait plein. Ici

l'action A pointée est une généralisation des actions B, une sorte d'héritage.



2) Diagramme de classe

→ Modéliser toutes les entités de l'application et leurs différentes relations



SPÉCIFICATIONS TECHNIQUES

Besoin	Technologie choisie	Justification
<ul style="list-style-type: none"> - Organisation du travail - Versioning 	Git Gitlab	<ul style="list-style-type: none"> - Permet de créer plusieurs versions de son projet - Permet de partager son projet
<ul style="list-style-type: none"> -Conteneuriser l'application 	Docker	<ul style="list-style-type: none"> - Permet de créer plusieurs containers - Permet d'avoir un projet indépendant de l'os
<ul style="list-style-type: none"> - Front-Office 	CSS 3	<ul style="list-style-type: none"> - Langage dédiée au front-office
	Bootstrap 5	<ul style="list-style-type: none"> - Framework CSS très utilisé - Grosse communauté - Très pratique pour le mobile-first - Gain de temps
	Ngrok	<ul style="list-style-type: none"> - Outil pour visualiser mon projet sur mobile notamment
	Javascript	<ul style="list-style-type: none"> - Langage pour animer et dynamiser des pages web
	Axios	<ul style="list-style-type: none"> - Librairie Javascript - Permet de faciliter les requêtes Ajax notamment
<ul style="list-style-type: none"> - Back-Office 	Symfony 5 :	<ul style="list-style-type: none"> - Framework PHP complet basée sur la POO et le modèle MVC - Très bonne documentation - Très grosse communauté - Gain de temps - Sécurité
	PHPMyAdmin + MYSQL :	<ul style="list-style-type: none"> - Grosse communauté - Gratuit - Facile d'utilisation

RÉALISATIONS

1. Installation du projet avec Docker :

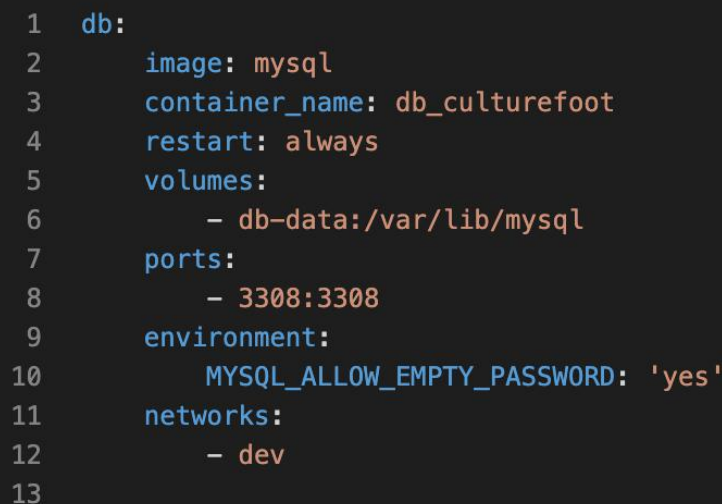
Pour commencer ,j'ai utilisé Docker pour rendre mon projet indépendant de l'OS et de la configuration de chaque ordinateur que j'utilise.

Cela me permet de ne pas tout réinstaller à chaque fois que je change de PC.

C'est un outil très puissant qui m'a permis de créer 3 conteneurs différents pour développer mon application:

- Un conteneur MYSQL

Celui ci va permettre de contenir la base de données du projet en récupérant au préalable une image mysql.

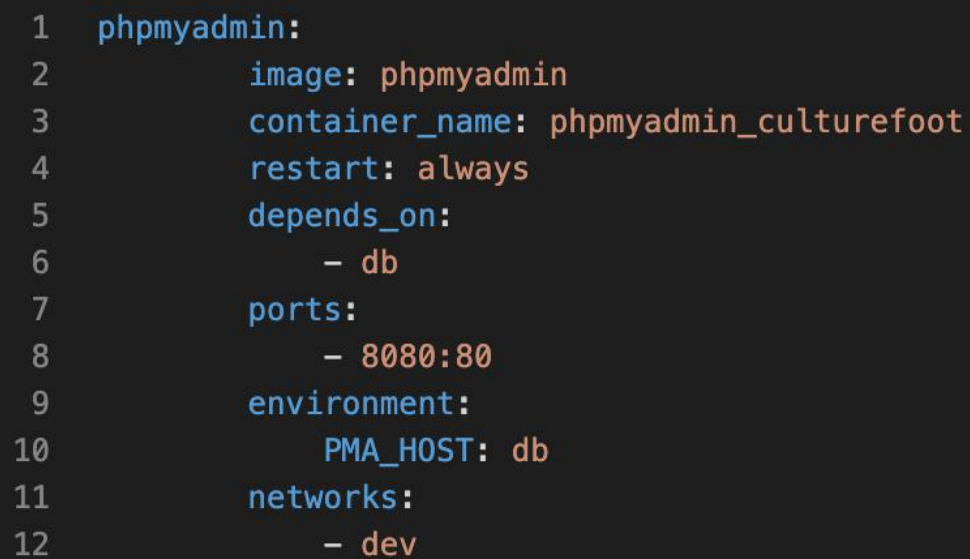


```
1 db:
2   image: mysql
3   container_name: db_culturefoot
4   restart: always
5   volumes:
6     - db-data:/var/lib/mysql
7   ports:
8     - 3308:3308
9   environment:
10     MYSQL_ALLOW_EMPTY_PASSWORD: 'yes'
11   networks:
12     - dev
13
```

- Un conteneur PHPMYADMIN :

Il va me permettre d'interagir plus facilement avec ma base de données .

On remarque notamment l'utilisation de "depends on" qui permet au conteneur phpMyAdmi d'attendre le conteneur MySQL avant de démarrer.



```
1  phpmyadmin:
2      image: phpmyadmin
3      container_name: phpmyadmin_culturefoot
4      restart: always
5      depends_on:
6          - db
7      ports:
8          - 8080:80
9      environment:
10         PMA_HOST: db
11      networks:
12         - dev
```

- Un conteneur pour mon application Symfony

```
1  www:
2      build: php
3      container_name: website_culturefoot
4      ports:
5      - "8741:80"
6
7      volumes:
8      - ./php/vhosts:/etc/apache2/sites-enabled
9      - ./:/var/www/
10     restart: always
11     networks:
12     - dev
```

Ici on utilise "build" et non "image" car nous voulons utiliser l'image personnalisée que nous avons créée dans notre DockerFile. Il contiendra la bonne version de php et de symfony, ainsi que des librairies essentielles pour notre projet tel que Composer.

→ Dockerfile :

```
1  FROM php:7.4-apache
2
3  RUN echo "ServerName localhost" >> /etc/apache2/apache2.conf
4
5  RUN apt-get update \
6      && apt-get install -y --no-install-recommends locales apt-utils git libicu-dev g++ libpng-dev libxml2-dev libzip-dev libonig-dev libxslt-dev;
7
8  RUN echo "en_US.UTF-8 UTF-8" > /etc/locale.gen && \
9      echo "fr_FR.UTF-8 UTF-8" >> /etc/locale.gen && \
10     locale-gen
11
12 RUN curl -sSk https://getcomposer.org/installer | php -- --disable-tls && \
13     mv composer.phar /usr/local/bin/composer
14
15 RUN apt-get install wget
16 # Symfony CLI
17 RUN wget https://get.symfony.com/cli/installer -O - | bash && mv /root/.symfony/bin/symfony /usr/local/bin/symfony
18
19 RUN docker-php-ext-configure intl
20 RUN docker-php-ext-install pdo pdo_mysql gd opcache intl zip calendar dom mbstring zip gd xsl mysqli
21 RUN pecl install apcu && docker-php-ext-enable apcu
22
23 WORKDIR /var/www/
```

On configure ensuite notre virtual host en créant un fichier vhosts avec les informations requises dans la documentation de Symfony :

```
1 <VirtualHost *:80>
2     ServerName culturefoot
3     DocumentRoot /var/www/public
4     DirectoryIndex /index.php
5
6     <Directory /var/www/public>
7         AllowOverride None
8         Order Allow,Deny
9         Allow from All
10
11     FallbackResource /index.php
12 </Directory>
13
14 # uncomment the following lines if you install assets as symlinks
15 # or run into problems when compiling LESS/Sass/CoffeeScript assets
16 # <Directory /var/www/project>
17 #     Options FollowSymlinks
18 # </Directory>
19
20 # optionally disable the fallback resource for the asset directories
21 # which will allow Apache to return a 404 error when files are
22 # not found instead of passing the request to Symfony
23 <Directory /var/www/project/public/bundles>
24     FallbackResource disabled
25 </Directory>
26 ErrorLog /var/log/apache2/project_error.log
27 CustomLog /var/log/apache2/project_access.log combined
28
29 # optionally set the value of the environment variables used in the application
30 #SetEnv APP_ENV prod
31 #SetEnv APP_SECRET <app-secret-id>
32 # SetEnv DATABASE_URL "mysql://db_user:db_pass@host:3306/db_name"
33 </VirtualHost>
34
```

Une fois nos conteneurs créés et configurés convenablement, il suffit de lancer les commandes suivantes :

- Pour créer nos conteneurs : “docker-compose build”
- Pour lancer nos conteneurs : “docker-compose up -d”
- Pour exécuter des commandes à l’intérieur de notre conteneur Symfony : “docker exec

-it <nom du conteneur> bash"

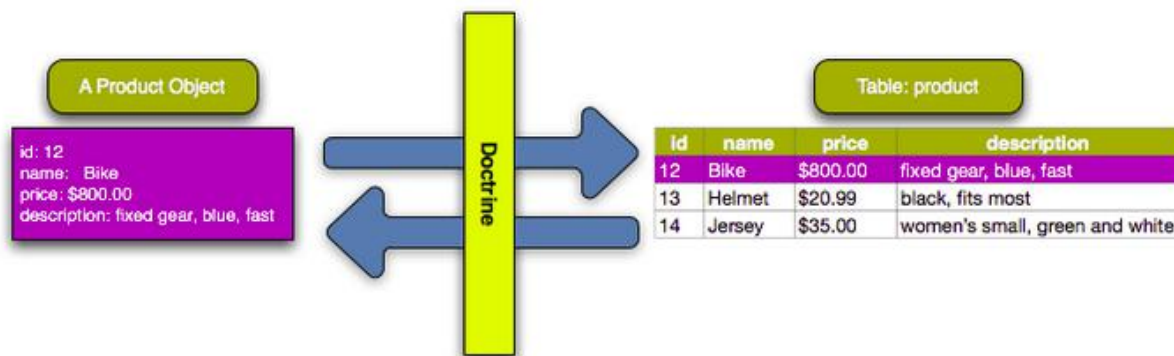
2. Créer La Base De Données

J'ai ajouté les informations nécessaires pour créer une base de données dans le dossier ".env"

```
DATABASE_URL="mysql://root:@db_culturefoot/culturefoot"
```

J'utilise ensuite la commande : `php bin/console doctrine:database:create` pour que la base de données se crée.

Désormais , l'ORM Doctrine va se charger d'interagir entre mes objets PHP et ma base de données:



Pour cela , je vais créer des migrations qui vont générer un langage SQL afin de synchroniser mon projet symfony et ma base de données.

L'opération s'effectue en deux temps:

- Créer la migrations avec la commande : `php bin/console make:migration`
- Synchroniser la base de données : `php bin/console doctrine:migration:migrate`

3. Authentification Et Sécurité

Titre Professionnel Développeur.se logiciel

J'ai utilisé la commande "`php bin/console make:user`" afin de générer une entité pour le utilisateurs avec des propriétés basiques mais essentielles (e-mail, pseudo, mot de passe ...).

Pour l'authentification j'ai utilisé la commande "`php bin/console make:auth`" La page contient un formulaire d'inscription avec vérifications de sécurité. Le mot de passe de l'utilisateur est crypté par une méthode de hachage (decrypt) sécurisée avant d'être stocké en base de données.

Exemple d'un mot de passe en base de données :

`$2y$13$Vng/FsHFjDCXqO8E56Gden4Vxlo39uVyyKkUuXnXoX...`

Nous pouvons aussi configurer la gestion des rôles dans le fichier `security.yaml`*: Avec Symfony on peut hiérarchiser les utilisateurs en leur attribuant des "rôles" (Admin, Modérateur, utilisateur, etc ..)

On pourra donc autoriser des routes uniquement aux utilisateurs possédant un rôle précis.

```
access_control:
  - { path: ^/admin, roles: ROLE_ADMIN }
  - { path: ^/user, roles: ROLE_USER }
```

4. Création du Controller:

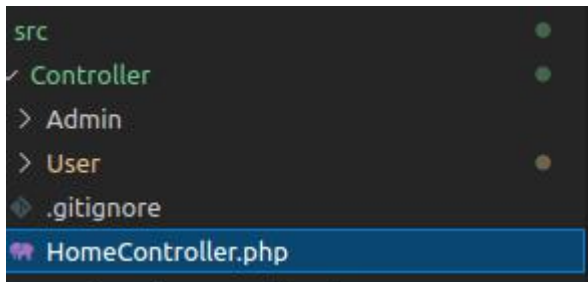
J'ai utilisé la commande `php bin/console` qui propose une multitude de fonctionnalités. Dans ce cas présent, j'ai besoin de créer un controller qui me permettra d'organiser les rendus de mes routes.

Je tape donc la commande suivante :

Titre Professionnel Développeur.se logiciel

```
thomas@thomas-ThinkPad-T440:/opt/lampp/htdocs/projets/TestQuiz$ php bin/console make:controller  
Choose a name for your controller class (e.g. GentlePopsicleController):  
> 
```

Je nommerai le controller 'HomeController' puisqu'il est mon controller principale.



Une fois la commande effectuée, un fichier est généré automatiquement avec une gestion de route par défaut.

5. Création des entités :

J'ai utilisé la commande `php bin/console` pour créer les "entités", des classes qui représentent les tables de ma base de données sous forme d'objet. J'ai donc utilisé `make:entity` pour créer mon entité Quiz.

```
thomas@thomas-ThinkPad-T440:/opt/lampp/htdocs/projets/TestQuiz$ php bin/console make:entity  
Class name of the entity to create or update (e.g. GentleGnome):  
> Quiz
```

J'ai ensuite précisé les propriétés de l'entité en détaillant leur type (string, number, boolean, relation, etc ...).

Suite à cette action, Symfony a généré 2 fichiers :

- Quiz.php : la page qui crée la classe 'Quiz' avec ses propriétés
- QuizRepository.php : la page qui permet d'encapsuler les requêtes lancées à la base de données.

6. Afficher La Navbar :

J'ai créé une barre de navigation pour accéder aux principales fonctionnalités du site .

1- Personnaliser la navbar en fonction du profil connecté



J'ai

utilisé une condition avec twig pour détecter si mon utilisateur est connecté afin de modifier l'affichage de ma navbar, dans le cas où l'utilisateur est connecté, on affiche donc une navbar personnalisée :



2-

Pouvoir filtrer les quiz par catégorie dans la navbar

La problématique était d'afficher toutes les catégories enregistrées en base de données dans un menu déroulant sur toutes les pages, étant donné que la barre de navigation est présente sur toutes les pages du site.

J'ai donc créé une variable globale qui sera envoyée à toutes mes vues. J'ai configuré ceci dans le fichier twig.yaml :

```
globals:
  NavCategory: '@App\Repository\CategoryRepository'
```

J'utilise ensuite cette variable 'NavCategory' pour récupérer toutes les noms de toutes mes catégories dans mon fichier twig :

```
1  {% Boucle pour afficher les catégories grâce a la variable globale 'NavCategory' %}
2      {% for category in NavCategory.findBy([],[],4) %}
3          <li>
4              <a class="dropdown-item dropdown-custom" href="{{ path('quiz_by', {name: category.name}) }}">
5                  
6                  {{ category.name }}</a>
7          </li>
8      {% endfor %}
```

7. Afficher les Quiz :

Dans mon contrôleur, j'ai créé une route sur la page qui affiche tous les quiz proposés par CultureFoot (en excluant les quiz créés par les utilisateurs) .

J'ai ensuite stocké la valeur de tous les quiz dans une variable que je renvoie à ma vue:

```

1  /**
2   * @Route("/quiz", name="quiz_all")
3   */
4  public function allQuiz(QuizRepository $quizRepository, UserRepository $userRepository): Response
5  {
6      $admin_user = $userRepository->findOneBy(['email' => "thomas.champromis@gmail.com"]);
7
8      // On récupère tous les quiz créés par l'administrateur et on les stock dans une variable pour les envoyer dans notre fichier Twig
9      $quiz = $quizRepository->findBy(['user' => $admin_user->getId()]);
10
11      return $this->render('home/quiz_all.html.twig', [
12          'quizz' => $quiz,
13      ]);
14  }
15
16

```

Dans mon fichier twig j'ai utilisé Bootstrap afin de faciliter la partie Front-Office.

J'utilise la variable que j'ai envoyée à ma vue pour créer une boucle "for" qui va afficher une carte pour chaque quiz.

J'afficherai dans un premier temps le nom du quiz et son image.

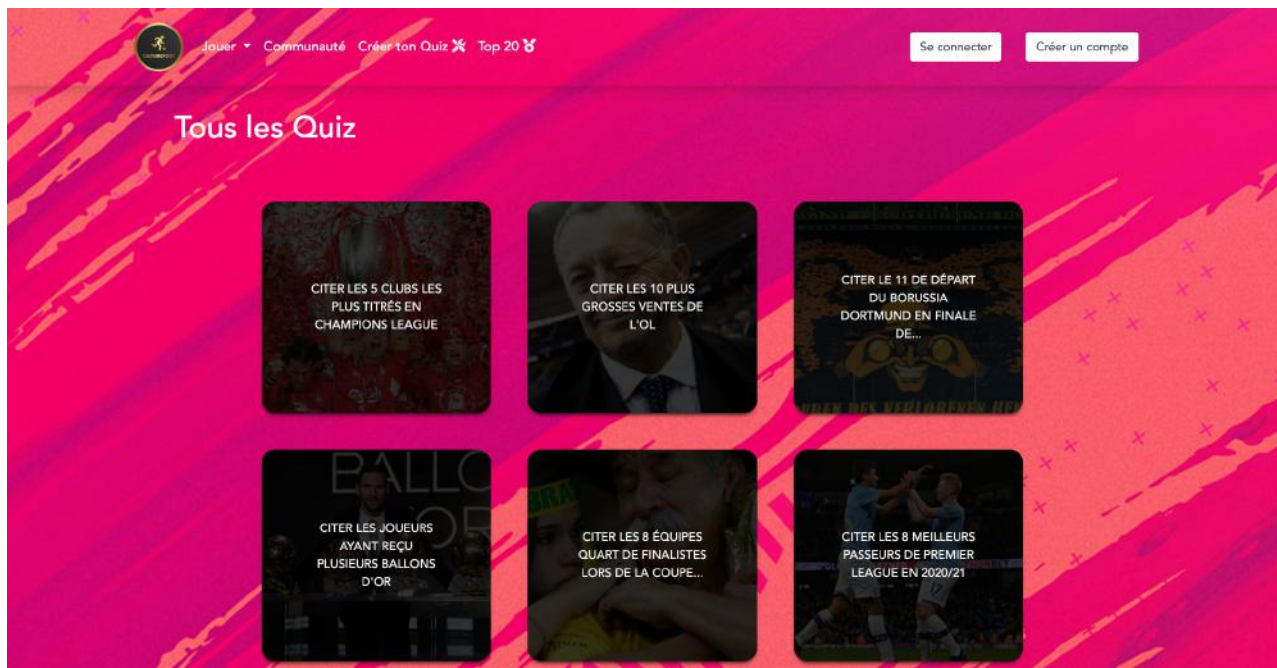
```

1      {% for quiz in quizz %}
2
3
4      <div class="thumbnail apparition m-4">
5      <span class="title hidden">{{ quiz.name|u.truncate(59,'...') }}</span>
6      <div class="thumbnail_container">
7      <div class="thumbnail_img" style="background-image: url(img/quiz/{{ quiz.img }});" data-parent="{{ quiz.name|u.truncate(59,'...') }}"></div>

```

J'ai utilisé des fonctions de twig comme 'truncate' ici . Cela me permet de limiter le nombre de caractères que je souhaite afficher dans le nom du Quiz.

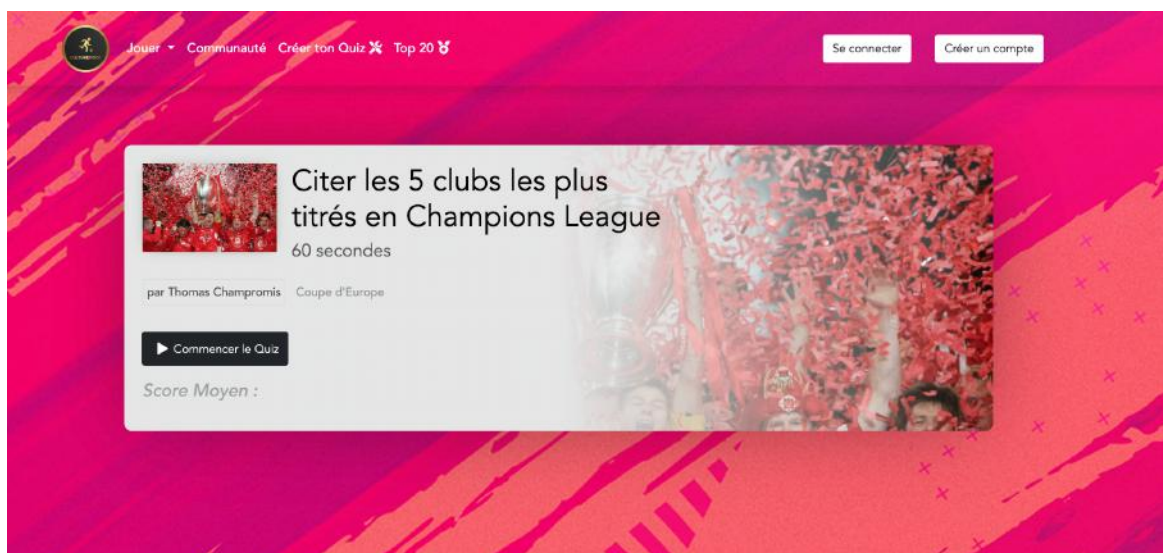
J'ai également créé un lien pour chaque carte qui renvoie vers le quiz concerné (je passe l'id du quiz en paramètre de la fonction 'path') .



8. Déroulé D'un Quiz:

Étape 1 : Présentation du quiz

Quand on clique sur un quiz, on accède à une page qui résume les caractéristiques principale de chaque quiz (intitulé, image, temps de jeu):

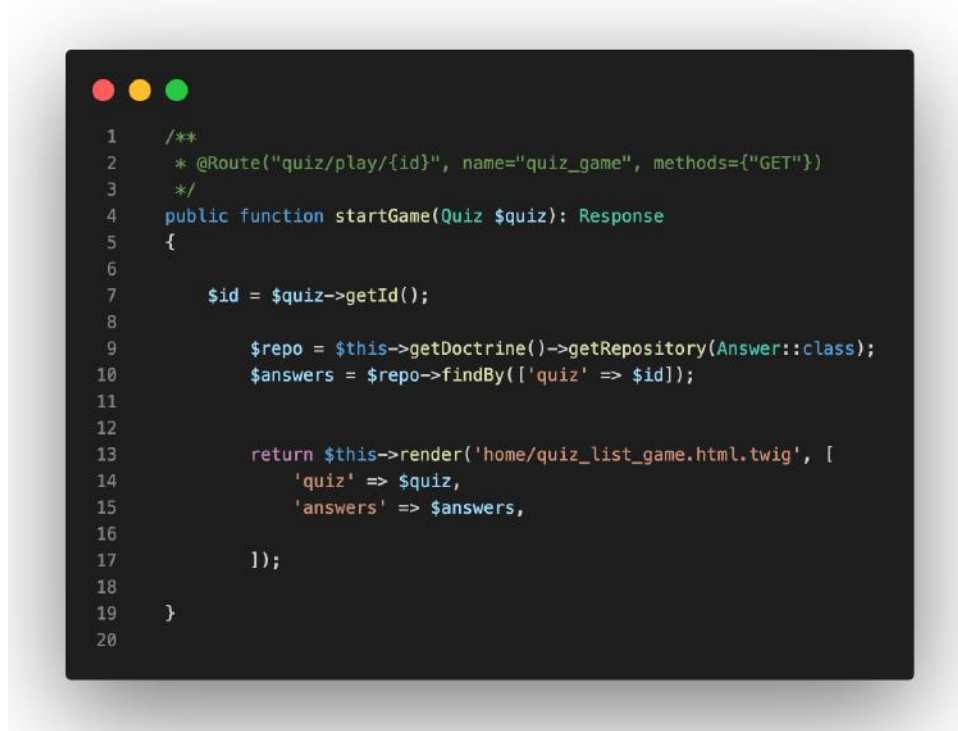


Ensuite

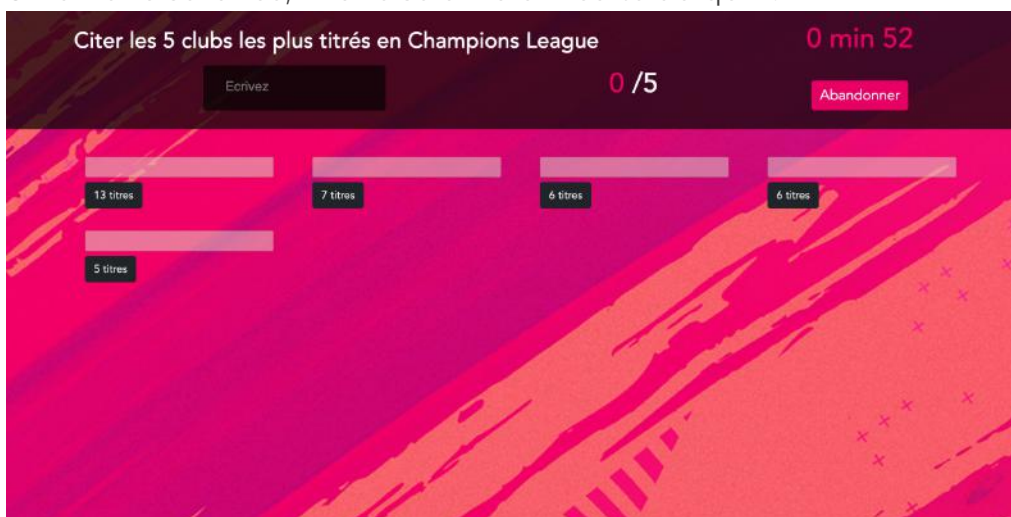
on clique sur play pour lancer le quiz et le jeu démarre !

Étape 2 : Début du jeu

Mon controller récupère toutes les réponses du quiz concerné et les envoie à ma vue :



Un chrono se lance, il varie selon la difficulté du quiz .



Le but étant de trouver un maximum de réponse à la problématique posée en écrivant nos réponses dans la barre de recherche.

Les réponses attendues sont masquées mais sont accompagnées d'un ou plusieurs

indices .

Le timer défile avec une fonction Javascript :

```

1 // Défilement du chrono
2 function timer() {
3   var sec = document.getElementById("timerDisplay").innerText;
4
5   // Variables pour convertir les secondes stockées en base de données
6   let totalSeconds = sec;
7   let minutes = Math.floor(totalSeconds / 60);
8   let seconds = totalSeconds % 60;
9
10  var timer = setInterval(function () {
11    document.getElementById("timerDisplay").innerHTML =
12      minutes + " min " + seconds;
13    seconds--;
14
15    if (seconds < 0 && minutes > 0) {
16      minutes = minutes - 1;
17      seconds = 59;
18    } else {
19      if (seconds < 0 && minutes == 0) {
20        endgame();
21      }
22    }
23  }, 1000);
24 }
25

```

Étape 3 : Fonctionnement du quiz

A chaque fois qu'une lettre est saisie, une requête ajax est envoyée pour vérifier

Citer les 5 clubs les plus titrés en Champions League

0 min 50

2 / 5

Abandonner

Ecrivez

Real Madrid		Liverpool	
13 titres	7 titres	6 titres	6 titres
5 titres			

si la réponse correspond à celle attendue.

Si la réponse est valide, on renvoie les réponses en format Json à notre vue

Route API qui reçoit la requête Ajax :

```

2 //==
3 * @Route("/api/json-answers", name="json_answers", methods={"GET"})
4 */
5 public function searchJson(Request $request, QuizRepository $quizRepository, AnswerRepository $answerRepository)
6 {
7     // On récupère le quiz concerné dans la requête
8     $quiz = new Quiz();
9     $quizId = $request->get('quiz');
10    $quiz = $quizRepository->findOneBy(['id' => $quizId]);
11
12    $goodAnswers = [];
13
14    // convertir string en minuscule sans accents
15    function stripAccents($str)
16    {
17        return strir(utf8_decode($str), utf8_decode("àáâãäåæçèéêëëîíïðñòóôõöøýþÿÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖØÝÞŒƆHJLMPQSTUVWXYZ"));
18    }
19
20    // On récupère toutes les réponses du quiz concerné
21    $answers = $answerRepository->findBy(['quiz' => $quiz]);
22    // On compare la réponse tapée par l'utilisateur avec toutes les réponses attendues
23    foreach ($answers as $answer) {
24        // si elle est exacte, on renvoie la réponse correspondante pour l'afficher dans le navigateur
25        if (stripAccents($answer->getName()) == stripAccents($request->get('answer'))) {
26            array_push($goodAnswers, $answer);
27        }
28    }
29
30    $response = $this->json($goodAnswers, 200, [], ['groups' => 'answers:play']);
31
32    return $response;
33 }
34

```

Étape 4 : Fin du jeu

Le jeu se stoppe lorsque l'utilisateur clique sur le bouton "Abandonner" ou quand le time arrive à son terme.

Les réponses manquantes sont alors envoyées et indiquées en noir et rouge.



Le score est stocké à la fin de la partie et envoyé vers la page des scores.

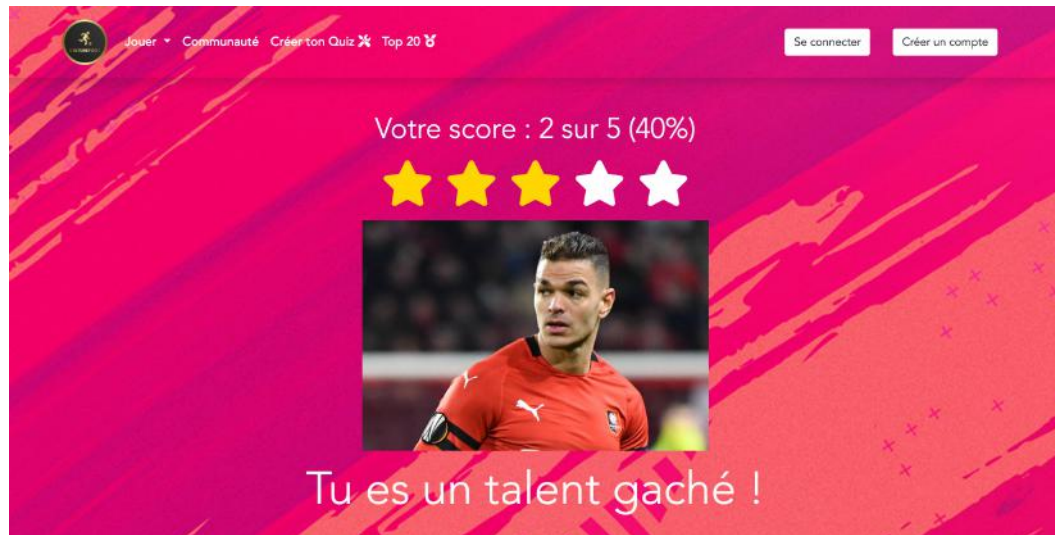
Étape 5 : Affichage du score

```

1  /**
2   * @Route("quiz/{id}/ton-score/", name="quiz_user_result", methods={"GET", "POST"})
3   */
4   public function userResult(Quiz $quiz, ?UserInterface $user, Request $request, AnswerRepository $answerRepository): Response
5   {
6       // On crée un score de partie
7       $game = new Game();
8       $repo = $this->getDoctrine()->getRepository(User::class);
9       // On récupère le joueur
10
11
12       $game->setQuiz($quiz)
13       ->setScore($request->request->get('percent'))
14       ->setPlayedAt(new \DateTime());
15
16       $player = $user ? $repo->findOneBy(['email' => $user->getUsername()]) : false;
17       $repo_game = $this->getDoctrine()->getRepository(Game::class);
18       if ($player){
19           $firstgame = $this->QuizNeverPlayed($repo_game, $player->getId(), $quiz->getId());
20           $game->setUser($player)
21           ->setIsFirstGame($firstgame);
22       }
23
24       // On spécifie le player et le quiz concerné pour créer l'historique de la partie
25       $entityManager = $this->getDoctrine()->getManager();
26       $entityManager->persist($game);
27       $entityManager->flush();
28

```

Le score s'affiche ensuite sous forme de pourcentage , avec des étoiles , une image et un commentaire correspondants à la performance du joueur :





Le code dans le fichier Twig correspondant à un résultat de 0% :

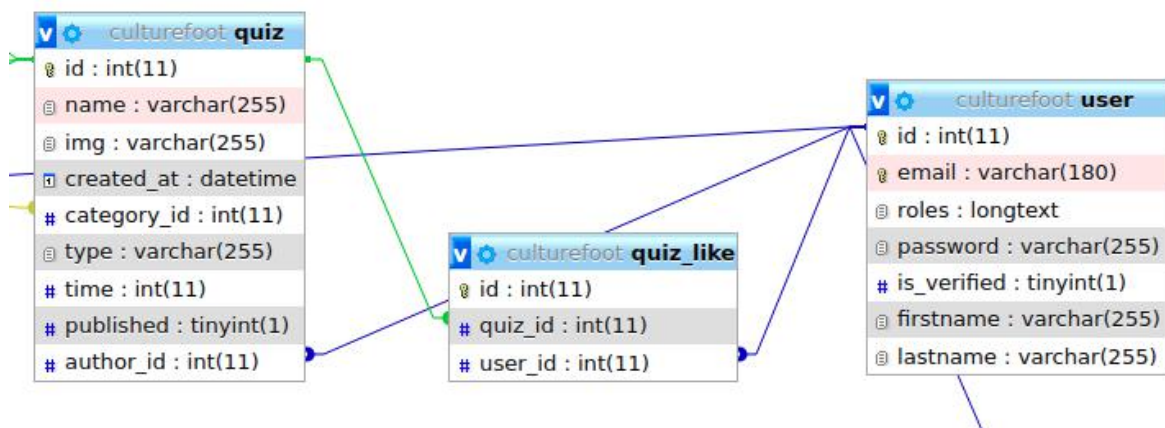
```
<h2 class='display-6 text-center'>Votre score :
  {{ founds }}
  sur
  {{ answers }}
  ({{ percent|number_format(0, '.', ',') }}%)
</h2>

{% if (percent == 0) %}
  <div class="result-container">
    <div class="star">
      <i class="fa fa-star"></i>
      <i class="fa fa-star"></i>
      <i class="fa fa-star"></i>
      <i class="fa fa-star"></i>
      <i class="fa fa-star"></i>
    </div>

    <img class='result-img' src='https://medias.lequipe.fr/img-photo-jpg/
    16-05-2018-groupama-stadium-lyon-fra-uefa-el-olympique-marseille-vs-atletico-madrid-finale-im/1500000001354375/
    214:50%2C2000:1239-1000-666-70/0a1c4.jpg' alt="" srcset="">
    <br/>
    <br/>
    <h2>
      Tu es CATASTROPHIQUE
    </h2>
  </div>
</div>
```

9. Système De Likes

Pour mon système de Like, j'ai créé une table de jointure nommée 'likes' en relation Many to Many entre ma table 'user' et ma table 'quiz' :



```

1  {# Si l'utilisateur à déjà liker ce Quiz , on affiche un logo coeur plein #}
2      {% if app.user and quiz.isLiked(app.user) %}
3          <a href="{{ path('quiz_like' , {id : quiz.id}) }}" class=' js-like mx-3 like'>
4              <span class="js-likes">
5                  {{ quiz.likes | length }}
6              </span>
7              <i class="fas fa-heart">
8                  <span class="js-label"></span>
9              </i>
10             </a>
11         {# Sinon on affiche un logo vide #}
12         {% else %}
13             <a href="{{ path('quiz_like' , {id : quiz.id}) }}" class=' js-like mx-3 like'>
14                 <span class="js-likes">
15                     {{ quiz.likes | length }}
16                 </span>
17                 <i class="far fa-heart">
18                     <span class="js-label"></span>
19                 </i>
20             </a>
21         </span>
22     {% endif %}
  
```

Dans mon fichier twig, je créer un bouton like qui active la fonction ci-dessous via une requête Ajax en javascript :

- Fonction dans mon controller

```

1  /**
2   * @Route("quiz/{id}/like/", name="quiz_like", methods={"GET", "POST"})
3   */
4   public function like(Quiz $quiz, QuizLikeRepository $quizLikeRepository): Response
5   {
6       $user = $this->getUser();
7       // Si aucun user n'est connecté , une page d'erreur est renvoyée
8       if (!$user) return $this->json([
9           'code' => 403,
10          'message' => "Unuserized"
11      ], 403);
12
13      // Si le quiz est déjà liké par l'utilisateur, alors cette action supprimera le like
14      if ($quiz->isLiked($user)) {
15          $like = $quizLikeRepository->findOneBy([
16              'quiz' => $quiz,
17              'user' => $user,
18          ]);
19          $this->getDoctrine()->getManager()->remove($like);
20          $this->getDoctrine()->getManager()->flush();
21
22          return $this->json([
23              'code' => '200',
24              'message' => 'ce like a été supprimé',
25              'likes' => $quizLikeRepository->count(['quiz' => $quiz])
26          ], 200);
27      }
28
29      // Sinon , le quiz est liké par l'utilisateur
30      $like = new QuizLike();
31      $like->setQuiz($quiz)
32          ->setUser($user);
33
34      $this->getDoctrine()->getManager()->persist($like);
35      $this->getDoctrine()->getManager()->flush();
36
37      // on retourne en Json le nombre de fois que le quiz a été liké
38      return $this->json(['code' => 200, 'message' => 'Like ajouté', 'likes' => $quizLikeRepository->count(['quiz' => $quiz]), 200);
39  }
40

```

3

- Requête Ajax en Javascript grâce à la librairie “axios” :

```

function OneClickBtnLike(event){
  console.log(this);
  event.preventDefault();

  const url = this.href;
  const countLike = this.querySelectorAll('.js-likes');
  var like;
  let icone = this.querySelectorAll('i');

  axios.get(url).then(function(response){

    like = response.data.likes;
    countLike[0].innerHTML = like;

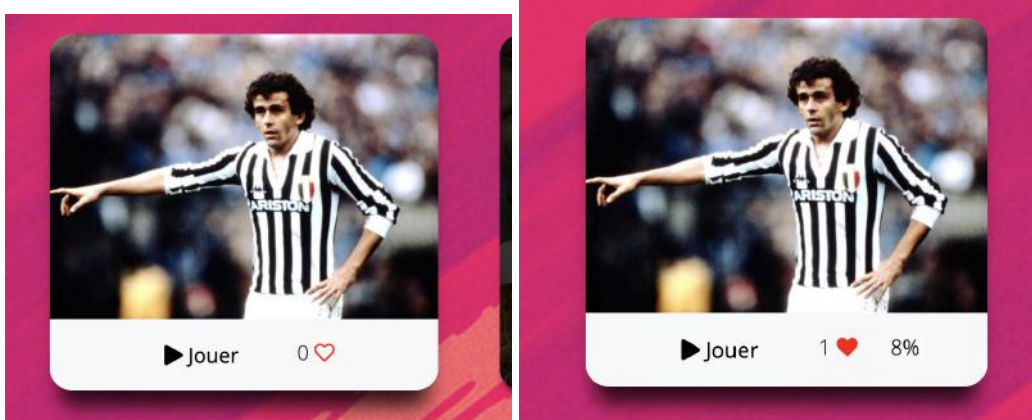
    if (icone[0].classList.contains('fas')){
      icone[0].classList.replace('fas', 'far')
    }
    else
    {icone[0].classList.replace('far', 'fas')}

  });

  // const spanCount = this.querySelectorAll('span.js-likes')
}

var likeBtn = document.querySelectorAll('a.js-like').forEach(function(link){
  link.addEventListener("click", OneClickBtnLike)
});

```



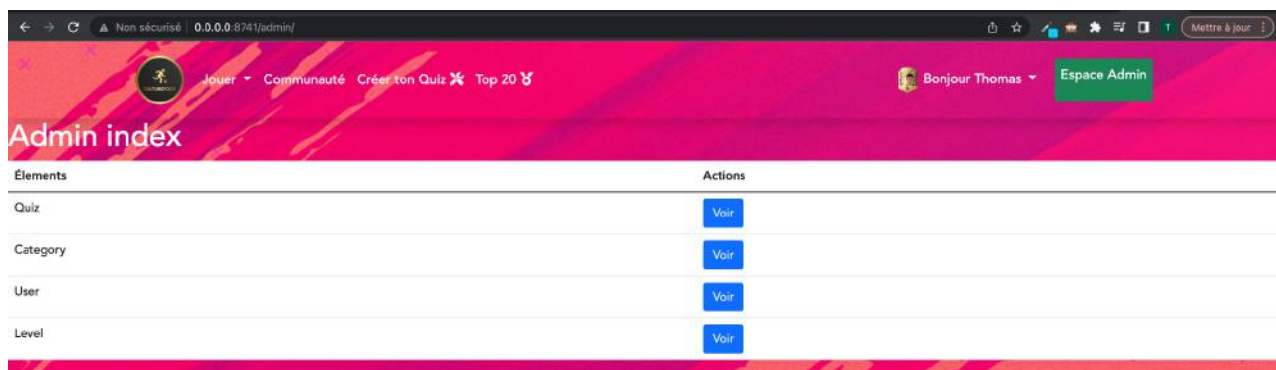
Quand l'utilisateur like un quiz, le visuel se met à jour dynamiquement

10. La partie “administrateur”

J'ai créé une page dédiée à l'administrateur du site afin qu'il puisse voir, ajouter, modifier et supprimer des éléments du site.

Il peut ainsi manipuler les quiz(et leurs réponses), les catégories, les niveaux et les utilisateurs du site .

J'ai créé un contrôleur simple nommé 'AdminController' qui permet d'afficher une page afin de choisir le type d'éléments que l'administrateur veut modifier. Ce dernier sera placé dans un dossier 'Admin' afin d'autoriser son accès à l'administrateur uniquement.



J'ai ensuite généré des 'crud' avec la commande suivante fournie par Symfony :

```
php bin/console make:crud
```

Ces derniers vont me permettre de voir, créer, modifier et supprimer très facilement les éléments de mon choix.

L'exemple pour l'entité Quiz

Dans le cas où je veux appliquer cette méthode au quiz de mon site, Cela va générer :

- un Contrôleur nommé 'QuizController' qui permettra d'effectuer toutes les actions souhaitées sur nos Quiz
- Des fichiers html.twig pour chaque action
- un formulaire adapté à l'entité dans le dossier 'Form' nommé 'QuizType':


```

namespace App\Form;

use App\Entity\Answer;
use App\Entity\Quiz;
use App\Entity\Category;
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
use Symfony\Component\Form\Extension\Core\Type\CollectionType;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Vich\UploaderBundle\Form\Type\VichImageType;

class QuizAdmin extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('name')
            ->add('imgFile', VichImageType::class, [])
            ->add('time')
            ->add('categoryId', EntityType::class, [
                'class' => Category::class,
                'choice_label' => 'name',
            ])
            ->add('answers', CollectionType::class, [
                'entry_type' => AnswerType::class,
                'entry_options' => ['label' => false],
                'allow_add' => true,
                'by_reference' => false,
                'allow_delete' => true,
            ]);
    }
}

```

Le fichier 'QuizType' nous permet de créer un formulaire adapté aux propriétés de l'entité. Dans ce cas, il nous servira pour créer et modifier des quiz. Une fois notre crud créé, la gestion des quiz est opérationnelle :

Quiz index						
Id	Name	Img	CreatedAt	Type	Time	actions
3	Citer les 5 clubs les plus titrés en Champions League	liverpool-2005-621b94a4126f3270946502.jpg	2022-02-27 16:11:32	liste	60	Voir Modifier
4	Citer les 10 plus grosses ventes de l'OL	jlm-auias-621b967f8e8b6153781164.jpg	2022-02-27 16:19:27	liste	180	Voir Modifier
5	Citer le 11 de départ du Borussia Dortmund en finale de LDC 2012/2013	x1080-622efb4416ab6850886355.jpg	2022-03-14 09:22:28	liste	240	Voir Modifier
6	Citer les joueurs ayant reçu plusieurs Ballons d'Or	mini-r5i4403-622f03d4a1e6c614914922.jpg	2022-03-14 09:59:00	liste	180	Voir Modifier
7	Citer les 8 équipes Quart de finalistes lors de la Coupe du Monde 2014	rtr3xpea-1-622f0585d8673516316827.jpg	2022-03-14 10:06:13	liste	180	Voir Modifier
8	Citer les 8 meilleurs passeurs de Premier League en 2020/21	images0-persgroep-net-622f08dc4bcc2771557606.jpg	2022-03-14 10:20:28	liste	180	Voir Modifier
9	Citer les clubs anglais vainqueurs de la Ligue des Champions	rosbif-au-four-622f0ab9d69a1334677930.jpeg	2022-03-14 10:28:25	liste	180	Voir Modifier

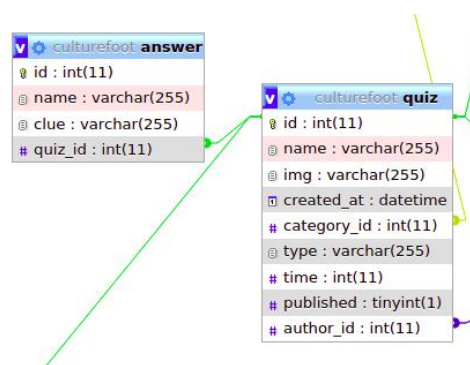
On utilise ce procédé pour la gestion des catégories et des utilisateurs afin d'avoir toutes les fonctionnalités souhaitées.

11. Création De Quiz :

La création de quiz est importante car on la retrouve dans les fonctionnalités disponibles pour l'utilisateur et pour l'administrateur.

La problématique consiste à créer un quiz et ses réponses simultanément alors qu'il s'agit de 2 entités différentes qui ne sont donc pas présentes sur la même table.

Les 2 tables sont reliées par une relation One(Quiz) To Many(Answer) :



Dans le formType de l'entité 'Quiz', j'ajoute un champ pour ajouter autant de réponses

souhaitées. J'utilise le `CollectionType` fournie par Symfony.

```
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder
        ->add('name')
        ->add('imgFile', VichImageType::class, [])
        ->add('time')
        ->add('categoryId', EntityType::class, [
            'class' => Category::class,
            'choice_label' => 'name',
        ])

        ->add('answers', CollectionType::class, [
            'entry_type' => AnswerType::class,
            'entry_options' => ['label' => false],
            'allow_add' => true,
            'by_reference' => false,
            'allow_delete' => true,
        ]);
}
```

Je peux ainsi insérer un formulaire de l'entité `Answer` dans mon formulaire de quiz.

Pour créer autant de réponses que je souhaite dans ma vue, j'utilise une technique expliquée dans la documentation de Symfony, à savoir le 'prototype' :

```
{% form_start(form) %}
{% form_row(form.name, {'label': 'Titre'}) %}
{% form_row(form.imgFile, {'label': 'Image'}) %}
{% form_row(form.time, {'label': 'Chrono (en secondes)'}) %}

<h3>Réponses</h3>
{# Prototype qui va générer du html dynamiquement #}
<ul class="reponses" data-prototype="{% form_widget(form.answers.vars.prototype)|e('html_attr') %}">
    {% for answer in form.answers %}
        <li>{% form_row(answer.name) %}
            {% form_row(answer.clue) %}</li>
    {% endfor %}
</ul>
<button class="btn btn-primary" type="submit">Créer le Quiz</button>

{% form_end(form) %}
```

J'ai ensuite implémenté et adapté un code en jQuery fourni par la documentation de Symfony pour créer des champs de formulaire dynamiquement.


```

// setup an "add a tag" link
var $addTagLink = $('<a href="#" class="add_tag_link btn btn-dark">Ajouter une réponse</a>');
var $newLinkLi = $('<li></li>').append($addTagLink);

jQuery(document).ready(function() {
  // Get the ul that holds the collection of tags
  var $collectionHolder = $('ul.reponses');

  // add the "add a tag" anchor and li to the tags ul
  $collectionHolder.append($newLinkLi);

  // count the current form inputs we have (e.g. 2), use that as the new
  // index when inserting a new item (e.g. 2)
  $collectionHolder.data('index', $collectionHolder.find('input').length);

  $addTagLink.on('click', function(e) {
    // prevent the link from creating a "#" on the URL
    e.preventDefault();

    // add a new tag form (see code block below)
    addTagForm($collectionHolder, $newLinkLi);
  });
});

function addTagForm($collectionHolder, $newLinkLi) {
  // Get the data-prototype explained earlier
  var prototype = $collectionHolder.data('prototype');

  // get the new index
  var index = $collectionHolder.data('index');

  // Replace '$$name$$' in the prototype's HTML to
  // instead be a number based on how many items we have
  var newForm = prototype.replace(/__name__/g, index);

  // increase the index with one for the next item
  $collectionHolder.data('index', index + 1);

  // Display the form in the page in an li, before the "Add a tag" link li
  var $newFormLi = $('<li></li>').append(newForm);

  // also add a remove button, just for this example
  $newFormLi.append('<a href="#" class="remove-tag">Supprimer</a>');

  $newLinkLi.before($newFormLi);

  // handle the removal, just for this example
  $('<.remove-tag>').click(function(e) {
    e.preventDefault();

    $(this).parent().remove();

    return false;
  });
}

```

Page pour créer un quiz :

Une fois le formulaire envoyé, je n'ai plus qu'à compléter manuellement les propriétés non précisées dans le formulaire et envoyer le nouveau quiz dans la base de données.

```

1  $quiz = new Quiz();
2  $form = $this->createForm(QuizUser::class, $quiz);
3  $form->handleRequest($request);
4
5  if ($form->isSubmitted() && $form->isValid()) {
6
7
8
9      if (sizeof($quiz->getAnswers()) == 0 || $quiz->getTime() < 10){
10         $this->addFlash(
11             'success',
12             'Ajouter des réponses et vérifiez que le timeur est supérieur a 10 secondes'
13         );
14         return $this->render('home/quiz-builder.html.twig', [
15             'quiz' => $quiz,
16             'form' => $form->createView(),
17         ]);
18     }
19
20     $category = $categoryRepository->findoneBy(['name' => 'community']);
21     $entityManager = $this->getDoctrine()->getManager();
22     $quiz->setCreatedAt(new \DateTime())
23         ->setType('liste')
24         ->setCategory($category)
25         ->setPublished(true)
26         ->setUser($user)
27         ->setIsExtreme(false);
28
29     $entityManager->persist($quiz);
30     $entityManager->flush();
31
32     return $this->redirectToRoute('home');
33 }

```

12. Afficher les statistiques de l'utilisateur :

L'utilisateur a la possibilité d'aller consulter les statistiques de son profil.



Il pourra consulter son score moyen et le nombre de quiz qu'il a créé

1.Score Moyen

J'ai donc créé une fonction pour calculer le score moyen de l'utilisateur :

```
public function getAverageRating(): ?int
{
    $scores = [];
    // Pour chaque partie faite par l'utilisateur
    foreach ($this->game as $game) {
        // j'envoie le score dans un tableau
        array_push($scores, $game->getScore());
    }
    // Si le tableau n'est pas vide, je fais la moyenne des scores présent a l'interieur
    if (sizeof($scores) > 0) {
        $average = array_sum($scores) / count($scores);
        return ($average);
    }
    return 0;
}
```

2.Nombre de quiz créés

Pour afficher le nombre de quiz créés par un utilisateur, j'utilise simplement la propriété de l'entité User intitulée '\$myquiz' qui contient les quiz créés par l'utilisateur.

J'ai donc juste a appeler la taille de cette propriété pour avoir la donnée que je souhaite :

```
<div class="container d-flex flex-row m-5">
  <div class="col-12 col-md-6 ">
    <H4>Nombres de quiz créés : <br>
    <span class='display-1'>{{ user.myquiz|length }}</span>
  </H4>
</div>
```

3. Nombres de Quiz Parfaits

L'utilisateur a la possibilité de visualiser le nombre de quiz parfait qu'il a effectué : c'est à dire le nombre de parties avec un score de 100% des réponses trouvées.

J'ai codé une fonction dans mon entité User qui permet cela . J'utilise une boucle qui va stocker dans un tableau toutes les parties avec un score de 100% , ensuite je compte le nombre d'éléments présents dans le tableau pour retourner la valeur demandée.

```
1 public function getPerfectGame(): ?int
2 {
3     $perfectgame = [];
4     // Pour chaque partie faite par l'utilisateur
5     foreach ($this->game as $game) {
6         $score = $game->getScore();
7         // Si le score est égale a 100% on le stock dans notre tableau
8         if ($score === 100) {
9             array_push($perfectgame, $game);
10        }
11    }
12    // ON retourne la taille du tableau
13    return sizeof($perfectgame);
14 }
```

13. Convertir le score du quiz en EXP :

Chaque Joueur connecté verra ses scores enregistrés dans l'application , il aura également un système d'XP pour lui permettre d'avoir une progression dans son experience de jeu. J'ai utilisé un service nommé "XpCalculatorService" pour convertir le score d'une partie en XP. J'ai défini un barème pour récompenser les quiz complètement validés

```
1  class XpCalculatorService
2  {
3
4      private $game;
5
6      public function __construct(Game $game)
7      {
8          $this->game = $game;
9      }
10
11     public function ScoreToXP(): int
12     {
13         $score = $this->game->getScore();
14         switch ($score) {
15             case ($score <50):
16                 $xp = $score;
17                 break;
18
19             case ($score <75):
20                 $xp = $score * 1.5 ;
21                 break;
22
23             case ($score <90):
24                 $xp = $score * 2;
25                 break;
26
27             case ($score <100):
28                 $xp = $score * 2.5;
29                 break;
30
31             case ($score === 100):
32                 $xp = $score * 4;
33                 break;
34         }
35         return $xp ;
36     }
37 }
38
```

14. Gérer le système de niveau :

Chaque Joueur aura la possibilité d'acquérir de l'exp pour ensuite changer de niveau et avoir une "carte" représentant ce dernier dans la page "Mon compte"



Pour cela j'ai créé un service qui va associer le bon niveau de l'utilisateur.


```
1  class LevelCalculatorService
2  {
3
4      private $user;
5      private $level;
6
7      public function __construct(User $user, Level $level)
8      {
9          $this->user = $user;
10         $this->level = $level;
11     }
12
13     public function UpdateLevel(LevelRepository $levelRepository)
14     {
15         $xp = $this->user->getXp();
16         $all_levels = $levelRepository->findAll();
17         foreach ($all_levels as $level) {
18             if (($level->getXpStart() <= $xp) && ($xp <= $level->getXpEnd())){
19                 $this->level = $level;
20                 break;
21             }
22         }
23         return $this->level;
24     }
25
26
27 }
```

15. Afficher les quiz populaires :

En page d'accueil , nous avons la possibilité de visualiser les quiz du site les plus joués depuis 1 semaine par les joueurs.



Pour cela j'ai utilisé une méthode dans le Repository de mon entité Game, pour renvoyer l'id des 6 quiz les plus joués depuis 1 semaine.

```
1
2  public function findHotQuiz()
3  {
4      $conn = $this->getEntityManager()->getConnection();
5      $dateLastWeek = date("d-m-Y", strtotime("-1 week"));
6      $sql = '
7          SELECT quiz_id
8          FROM game
9          WHERE played_at >(STR_TO_DATE(:date , "%d-%m-%Y"))
10         GROUP BY quiz_id
11         ORDER BY COUNT(id) DESC;
12         LIMIT 6;
13     ';
14     $stmt = $conn->prepare($sql);
15     $resultSet = $stmt->executeQuery(['date' => $dateLastWeek]);
16
17     // returns an array of arrays (i.e. a raw data set)
18     return $resultSet->fetchAllAssociative();
19 }
20 }
21
```

J'utilise ensuite cette fonction dans mon controller pour récupérer les objets Quiz correspondants et les envoyer à ma vue.

```

1  /**
2   * @Route("/hot-quiz", name="hot_quiz")
3   */
4  public function hotQuiz(GameRepository $gameRepo, QuizRepository $quizRepo): Response
5  {
6      $quiz = [];
7      $results = $gameRepo->findHotQuiz();
8      foreach ($results as $q) {
9          array_push($quiz, $quizRepo->findOneBy(['id' => $q]));
10     }
11
12     return $this->render('home/hot_quiz.html.twig', [
13         'quizz' => $quiz,
14     ]);
15 }

```

16. Afficher l'historique des réponses dans le score :

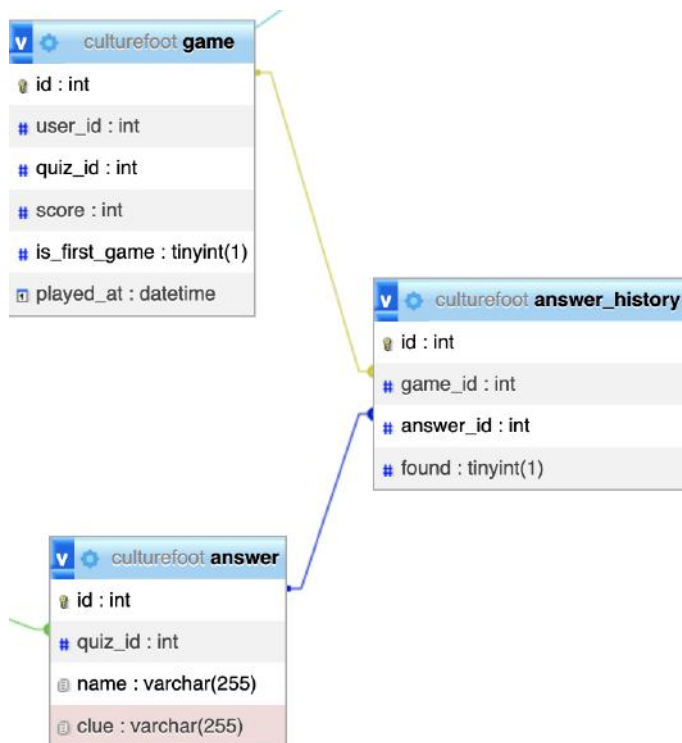
Lorsque l'on termine un quiz, nous avons un tableau qui affiche l'historique de chaque réponse c'est à dire le pourcentage des joueurs ayant répondu juste pour chaque réponse attendue dans le quiz .

Réponse	Résultat	% de joueurs ayant trouvé cette réponse
Liverpool	Trouvée	83.3 %
Real Madrid	Trouvée	33.3 %

Réponse	Résultat	% de joueurs ayant trouvé cette réponse
Milan AC	Raté	0 %
Bayern Munich	Raté	0 %
Barcelone	Raté	0 %

[Retour a l'accueil](#)
[Recommencer ce Quiz](#)

J'arrive à récupérer ces infos en utilisant la table "AnswerHistory" qui enregistre le résultat de chaque réponse pour chaque partie terminée .



```

1  if(($request->request->get('found')) != ""){
2      $array_answers_found = explode(",", $request->request->get('found'));
3      foreach ($array_answers_found as $el) {;
4          $answerHistory = new AnswerHistory();
5          $answer = $answerRepository->findOneBy(['id' => $el]);
6          array_push($answers_correct, $answer);
7          $answerHistory->setGame($game)
8              ->setAnswer($answer)
9              ->setFound(true);
10         $entityManager = $this->getDoctrine()->getManager();
11         $entityManager->persist($answerHistory);
12         $entityManager->flush();
13     }
  
```

On remarque dans le screen précédent que je crée une instance de la classe AnswerHistory

pour chaque réponse trouvée par l'utilisateur (je récupère les réponses dans la requête) . J'ai ensuite créé une fonction dans mon entité Answer pour récupérer le pourcentage moyen de validation de chaque réponse pour ensuite l'afficher dans ma vue.

```
1 public function getAverageScore()
2 {
3     $histories = $this->getAnswerHistories();
4     $number_answers_found = 0;
5     foreach ($histories as $history) {
6         if ($history->IsFound() == true) {
7             $number_answers_found++;
8         };
9     }
10    $saverage_number = $number_answers_found > 0 ? $number_answers_found / sizeof($histories) * 100 : 0;
11    return $saverage_number;
12 }
13 }
```

17. Tests avec PHPUnit :

J'ai intégré des tests unitaires dans mon application, le principe des tests est simple :

- Exécuter du code provenant de l'application
- Vérifier que tout s'est bien déroulé comme prévu.

J'ai utilisé PHPUnit. C'est la référence en matière de tests avec PHP .Cet outil fournit tout une suite de classes et méthodes utiles pour faciliter l'écriture de tests.

J'installe cette dépendance grâce à Composer en utilisant la commande:

```
composer require --dev phpunit/phpunit
```

Mes fichiers de tests sont créés dans le dossier "Tests" à la racine du projet. On ajoute le mot "test" à la fin de chaque nom de fichier à tester. Cela va permettre à phpunit d'exécuter le contenu de ces fichiers lorsqu'on lance la commande de test :

```
"php bin/phpunit "
```

→ Tests du Controller

J'ai créé un fichier intitulé HomeControllerTest.php :

```
1
2 class HomeControllerTest extends WebTestCase
3 {
4
5
6     public function testHomepage(): void
7     {
8         $client = static::createClient();
9         $crawler = $client->request('GET', '/');
10
11         $this->assertResponseStatusCodeSame(200);
12         $this->assertCount(1, $crawler->filter('h1'));
13     }
14
15     public function testAllQuizPage(): void
16     {
17
18         $client = static::createClient();
19         $crawler = $client->request('GET', '/quiz');
20         $this->assertResponseStatusCodeSame(200);
21         $this->assertSelectorTextContains('h1', 'Tous les Quiz');
22     }
23
24 }
```


Premièrement, je teste le statut de la réponse http de la page d'accueil du site et nombre de balise H1 dans le body HTML de cette dernière. On peut voir que les lignes 11 et 12 utilisent des fonctions qui se chargent de vérifier si la valeur retournée est égal à la valeur attendue.

Deuxièmement, j'ai également testé la page qui recense tous les quizz de l'application, en vérifiant le contenu du titre de la page.

→ Test du Repository (en utilisant DataFixture)

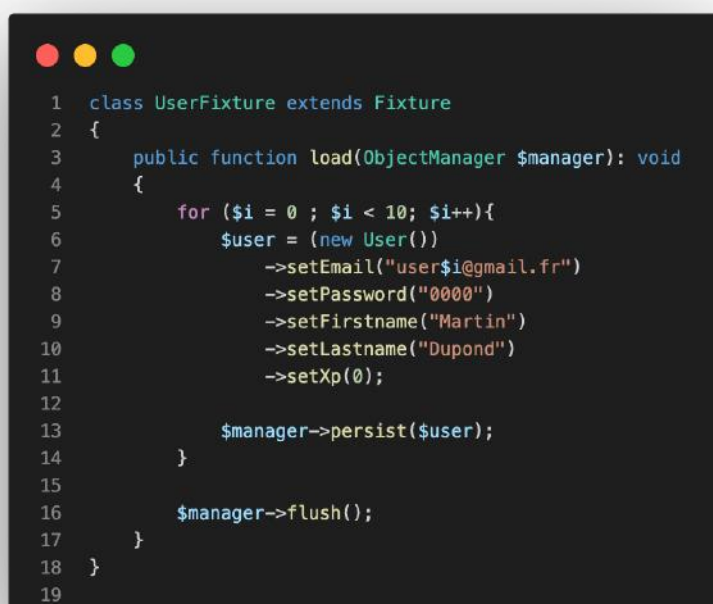
J'utilise la librairie DoctrineFixtureBundle pour insérer un système de fixtures. Ce système va permettre de déclarer un ensemble de données à rentrer dans la base avant d'exécuter chaque test.

Pour cela j'utilise la commande :

```
composer require --dev orm-fixtures
```

Je crée un fichier UserFixture.php dans le dossier "DataFixtures" qui s'est automatiquement créé dans le dossier Src de mon application.

Dans ce fichier je crée une fonction qui me permet de créer 10 utilisateurs dans ma base de données de test.



```
1 class UserFixture extends Fixture
2 {
3     public function load(ObjectManager $manager): void
4     {
5         for ($i = 0 ; $i < 10; $i++){
6             $user = (new User())
7                 ->setEmail("user$i@gmail.fr")
8                 ->setPassword("0000")
9                 ->setFirstname("Martin")
10                ->setLastname("Dupond")
11                ->setXp(0);
12
13            $manager->persist($user);
14        }
15
16        $manager->flush();
17    }
18 }
19
```

Je crée ensuite mon fichier pour tester mon Repository.

J'inclus une fonction qui permet de compter le nombre d'utilisateurs en s'assurant qu'il est bien égal au nombre que l'on a défini dans notre fixture, ici 10.

```
1 public function testCountUsers(): void
2 {
3     $userfixtures = $this->databaseTool->loadFixtures([
4         UserFixture::class,
5     ]);
6     $users = self::$container->get(UserRepository::class)->count([]);
7     $this->assertEquals(10, $users);
8
9 }
```

→ Test de l'Entité

Je crée un fichier pour tester les getters et les setters de mon entité QUIZ.

```
1 class QuizEntityTest extends KernelTestCase
2 {
3     public function testQuizEntity()
4     {
5         $quiz = (new Quiz())
6             ->setName('TestQuiz1')
7             ->setTime(90);
8
9         $this->assertEquals('TestQuiz1', $quiz->getName());
10        $this->assertEquals(90, $quiz->getTime());
11    }
12 }
13
14 }
15
```

18. Intégration continue avec Gitlab-CI :

L'intégration continue est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.

Pour la mettre en place j'ai utilisé gitlab-ci qui permet d'automatiser les builds, les tests, les livraisons et les déploiements de l'application.

J'ai donc créé un fichier gitlab-ci.yml à la racine du projet :

```
1  image: jakzal/phpqa:php7.4
2
3
4  cache:
5    paths:
6      - vendor/
7
8  stages:
9    - PhpVersionCheck
10   - UnitTests
11
12
13  hello :
14    stage: PhpVersionCheck
15    script:
16      - php -v
17
18  phpunit:
19    stage: UnitTests
20    services:
21      - name: mysql:5.7
22        alias: mysql
23
24    only:
25      - main
26
27  variables:
28    MYSQL_ROOT_PASSWORD: myapptest
29    MYSQL_DATABASE: myapptest
30    MYSQL_USER: myapptest
31    MYSQL_PASSWORD: myapptest
32    DATABASE_URL: 'mysql://myapptest:myapptest@mysql:3306/myapptest'
33
34  before_script:
35    - apt-get update && apt-get install -y git libzip-dev
36    - curl -sSk https://getcomposer.org/installer | php -- --disable-tls && mv composer.phar /usr/local/bin/composer
37    - docker-php-ext-install mysqli pdo pdo_mysql zip
38    - composer install
39    - php bin/console doctrine:database:create --env=test
40    - php bin/console doctrine:schema:update --force --env=test
41
42  script:
43    - php bin/phpunit
44  allow_failure: false
```

Je récupère une image de php7.4 qui correspond à la bonne version de PHP utilisée pour l'application.

Je met en cache les fichiers du dossiers vendor pour plus de performances.

J'ai créé 2 stages :

- 1 qui vérifie tout simplement la version de php utilisée
- 1 qui lance tous les tests de mon application

Pour ce dernier, j'utilise un service qui me permet d'utiliser un environnement mysql afin de générer ma base de données de test . Je définie les variables nécessaires pour la créer avec la commande `"php bin/console doctrine:database:create --env=test"` .

Une fois l'environnement de test configuré , je lance la commande 'php bin/phpunit' pour lancer les tests de mon application.

CONCLUSIONS

J'ai beaucoup aimé développer ce projet personnel , il m'a permis de mettre en place un concept de site imaginé avant mon entrée en formation.
Ce fut encore plus stimulant de lier ma passion du football avec le développement d'une application comme celle

J'ai pu comprendre l'organisation et l'utilisation d'un framework tel que Symfony mais aussi de me familiariser avec la conception complète d'une application .

Je suis globalement satisfait de mon travail malgré les difficultés rencontrées et les plusieurs axes d'amélioration envisagés.
Je vais également me servir de cette expérience pour mieux optimiser l'organisation de mon travail à l'avenir .

Cela représente une base solide sur laquelle s'appuyer pour mes projets futurs
Ce projet m'a aussi permis de bien me rendre compte que depuis le début de la formation , j'ai pu acquérir des connaissances sur de multiples technologies différentes. Il a surtout renforcé ma faculté à apprendre, à savoir m'organiser et à rechercher les bonnes informations en cas de difficulté technique.