Humanbooster

Stéphane LAY

Formation Developpeur web & web mobile

Période du 28 novembre 2024 au 25 novembre 2025

DOSSIER PROJET

“Dice&Meet” - Application d'événements autour du jeu de société

https://dicemeet.pro

Partenaires institutionnels

UNION EUROPÉENNE

L'EUROPE S'ENGAGE en région Auvergne-Rhône-Alpes avec le FSE

MINISTÈRE DU TRAVAIL, DU PLEIN EMPLOI ET DE L'INSERTION
Liberté Égalité Fraternité

RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

France Travail

REMERCIEMENTS

Je tiens tout d'abord à exprimer mes premiers remerciements envers France Travail qui m'a permis de connaître et de candidater à cette formation et à Human Booster qui m'a donné l'opportunité de suivre cette formation enrichissante en tous points.

Je remercie tout particulièrement :

- Karim Z, pour sa pédagogie exceptionnelle lors de l'apprentissage des bases de données et de MySQL.
- Nicolas M, pour sa gentillesse et sa patience lors de ses cours en JavaScript et de ses comptes rendus sur évaluations ou projets.
- Lucas D, pour son professionnalisme et son implication durant l'apprentissage de Symfony, qui a été crucial dans ma compréhension du back-end.
- Jean D, pour ses nombreux enseignements et conseils en PHP et plus globalement du modèle MVC.
- Cyril C, pour ses cours structurés en web statique qui m'ont permis d'avoir les bonnes bases en HTML et CSS.
- Clement M, pour son approche créative du design et du maquettage.
- David M, pour ses cours qui sont les meilleurs cours d'anglais auxquels je n'ai jamais assisté.
- Nicolas R, pour sa capacité à mettre à l'aise du début à la fin de l'année
- Asna K, pour sa bienveillance et son enthousiasme lors des ateliers TRE.
- L'ensemble des mes camarades, qui ont égayé mon quotidien.

LISTE DES COMPÉTENCES DU RÉFÉRENTIEL

Activités types	**Compétences professionnelles**	**Techniques utilisées**
Développer la partie front-end d'une application web ou web mobile sécurisée	Installer et configurer son environnement de travail en fonction du projet web ou web mobile	Windows, Linux, Visual Studio Code, Cursor
	Maquetter des interfaces utilisateur web ou web mobile	Figma
	Réaliser des interfaces utilisateur statiques web ou web mobile	HTML, CSS, Tailwind
	Développer la partie dynamique des interfaces utilisateur web ou web mobile	Angular, TypeScript, JavaScript
Développer la partie back-end d'une application web ou web mobile sécurisée	Mettre en place une base de données relationnelle	MySQL, PHPMyAdmin, Doctrin ORM, Django REST Framework
	Développer des composants d'accès aux données SQL et NoSQL	MySQL, Symfony,EasyAdmin,Django REST Framework
	Développer des composants métier côté serveur	PHP, Symfony
	Documenter le déploiement d'une application dynamique web ou web mobile	WAMP, VPS OVH , Docker

TABLE DES MATIERES

1. Contexte du projet 6
    1.1 Présentation de l'entreprise et du service 6
    1.2 Cahier des charges / Expression des besoins 7
    1.3 Contraintes, livrables attendus 8
    1.4 Environnement humain et technique 8
    1.5 Objectifs de qualité 8
2. Réalisation du Back-End 9
    2.2 Conception et implémentation de la base de données 10
    2.3 Ajout des fixtures 12
    2.4 Sécurité et authentification 13
    2.5 Présentation de fonctionnalités métier 16
        2.5.1 Mise à jour d'un meetup 19
        2.5.2 Mise à jour du profil utilisateur 19
        2.5.3 Génération de suggestions pour la barre de recherche 22
    2.6 EasyAdmin 23
3. Réalisations Front-End 25
    3.1 Conception des maquettes(cf annexes) 25
    3.2 Initialisation du projet Angular 25
    3.3 Architecture globale de l'application Angular 26
    3.4 Authentification et sécurité 27
    3.5 Présentation de fonctionnalités pertinentes 31
        3.5.1 Page de liste des événements : recherche dynamique, filtres et pagination 31
        3.5.2 Page de profil : affichage et mise à jour 33
        3.5.3 Page d'événement 35
4. Déploiement et optimisation 38
    4.1 Déploiement de l'application 38
    4.2 Optimisations 39
5. Présentation d'un jeu d'essai 41
    5.1 Test manuel 41
    5.2 Test automatisé 43
6. Travail effectué en stage 44
    6.1 Présentation et objectif du stage 44
    6.2 Réalisation de la tâche 45
7. Conclusion 49
ANNEXES 50

1. Contexte du projet

Dans le cadre de la formation, il nous est proposé de réaliser un projet “Business Case”, un projet fictif nous permettant de couvrir toutes les compétences du référentiel au cas où nos autres projets ou expériences ne suffiraient pas à toutes les couvrir.

J’ai réalisé un stage de 2 mois dont je présenterais un extrait de mon travail dans ce dossier. Ce stage, bien que très formateur, ne valide pas l’ensemble des compétences qu’il est nécessaire de présenter dans ce dossier et lors de ma soutenance finale. C’est pourquoi j’ai donc opté pour le projet Business Case afin de pouvoir montrer mes compétences sur chacune des étapes nécessaires au développement d’une application complète en full-stack.

Pour la réalisation du Business Case, l’organisme de formation nous a fourni un sujet de base,la création d’une plateforme de coliving, avec différentes instructions pour nous aider à la création de notre application. J’ai pour ma part préféré un sujet plus personnel et proche de mes centres d'intérêt,bien que très similaire en termes de fonctionnalités attendues, afin d'élaborer cette application avec toute mon implication.

1.1 Présentation de l’entreprise et du service

Une startup, spécialisée dans les outils de mise en relation, souhaiterait mettre en place une application permettant de mettre en relation les joueurs de jeux de société, communauté grandissante ces dernières années. L’objectif serait de permettre aux joueurs de pouvoir faciliter les rencontres physiques autour des jeux de société en fournissant une plateforme qui permet d’organiser des parties, sur la base d’un large répertoire de jeux connus et de différents lieux(notamment les bars à jeux) pouvant accueillir ces événements dans plusieurs villes françaises.

1.2 Cahier des charges / Expression des besoins

L’application doit permettre à des utilisateurs amateurs de jeux de société de se rencontrer physiquement autour de parties organisées dans différentes villes.

Besoins fonctionnels:

- Inscription/Connexion/Profil
    - Création de compte (email + mot de passe)
    - Authentification sécurisée
    - Gestion du profil (infos générales, bio, avatar, traits de caractères)
    - Dashboard recensant les événements et les notifications de l'utilisateur
    - Espace administrateur pour gérer les données
- Catalogue de jeux et de lieux
    - Recherche et filtrage par critères
    - Consultation détail jeu (nom, type, nombre de joueurs, description)
    - Consultation détail lieu (adresse, ville, description)
- Organisation d'événements
    - Création de partie par un utilisateur
    - Choix de jeu, date, lieu, choix des participants
    - Inscription/désinscription à un événement
    - Détail de partie(participants, infos pratiques, chat)
- Messagerie
    - Messagerie privée entre utilisateurs
    - Chat dédié pour chaque événement

Besoins non-fonctionnels:

- Interface intuitive et responsive
- Sécurité des données (mot de passe hashé, protection CSRF)
- Architecture REST pour l'API

1.3 Contraintes, livrables attendus

Ce projet, développé dans un environnement full-stack, s'appuie sur les technologies enseignées durant la formation : Angular pour le front-end, et Symfony pour le back-end. L'ensemble communique avec une base de données MySQL hébergée localement via WAMP, puis par la suite déployée sur un VPS.

Ces choix techniques impliquent le respect de bonnes pratiques, notamment pour la gestion sécurisée des comptes utilisateurs, l'authentification, la structuration de l'API et la cohérence des échanges entre les différentes couches de l'application.

L'application doit être suffisamment stable pour être testée en conditions réelles et permettre une démonstration fluide. Parmi les livrables attendus figurent les maquettes et wireframes des principales pages (liste des événements, messagerie, page d'accueil), le Modèle Physique de Données, la documentation technique associée, ainsi que la présentation finale du projet.

1.4 Environnement humain et technique

Le développement de l'application a été réalisé en autonomie, avec des interventions ponctuelles du formateur, notamment pour la validation du MPD.

D'un point de vue technique, le développement a été effectué sous Windows, principalement dans Visual Studio Code et sa variante Cursor. Le front-end a été développé en Angular 20.2.2 et le back-end en Symfony 7.3.3. L'hébergement local de la base de données MySQL a été assuré par WAMP, puis l'ensemble du projet a été déployé sur VPS. Les tests de l'API ont été menés via Postman, et l'ensemble du projet a été versionné avec Git puis synchronisé sur GitHub.

1.5 Objectifs de qualité

L'objectif principal est de proposer une application agréable à utiliser, avec une interface claire, cohérente et suffisamment fluide pour permettre aux utilisateurs d'organiser facilement des sessions de jeux. L'accent a été mis sur la lisibilité des pages, la simplicité de navigation et la cohérence visuelle de l'ensemble de l'interface.

Du côté du back-end, l'objectif de qualité repose sur la fiabilité et la sécurité du système. Cela inclut la protection des données sensibles, l'utilisation de méthodes d'authentification sécurisées et la mise en place de validations côté serveur afin de garantir l'intégrité des données. L'API a été conçue de manière à rester cohérente et maintenable, ce qui facilitera les évolutions futures.

Enfin, une attention particulière a été portée à la maintenabilité du code et à l'architecture du projet. Le but est de permettre l'ajout de nouvelles fonctionnalités ou l'extension du service sans nécessiter de refonte majeure. La modularité du front-end Angular et l'organisation du code Symfony contribuent à cet objectif. La stabilité, la réactivité et la compatibilité multi-supports (desktop et mobile) font également partie des critères retenus pour assurer une expérience utilisateur satisfaisante.

2. Réalisation du Back-End

2.1 Initialisation du projet et installation des composants essentiels

Le back-end de l'application a été développé avec Symfony 7.3.3, dans une architecture exclusivement orientée API.
Le projet a été créé via Symfony CLI, sans le flag --webapp, afin de conserver uniquement les composants nécessaires à la construction d'une API avec la commande suivante:

```
>symfony new Dicemeet_API
```

Nous installons ensuite plusieurs composants indispensables :

- Doctrine ORM, qui prendra en charge la correspondance entre les classes PHP et les tables de la base de données MySQL.

```
composer require orm
```

- MakerBundle, facilitant la génération d'entités,de leurs relations entre elles et de contrôleurs.

```
composer require --dev maker
```

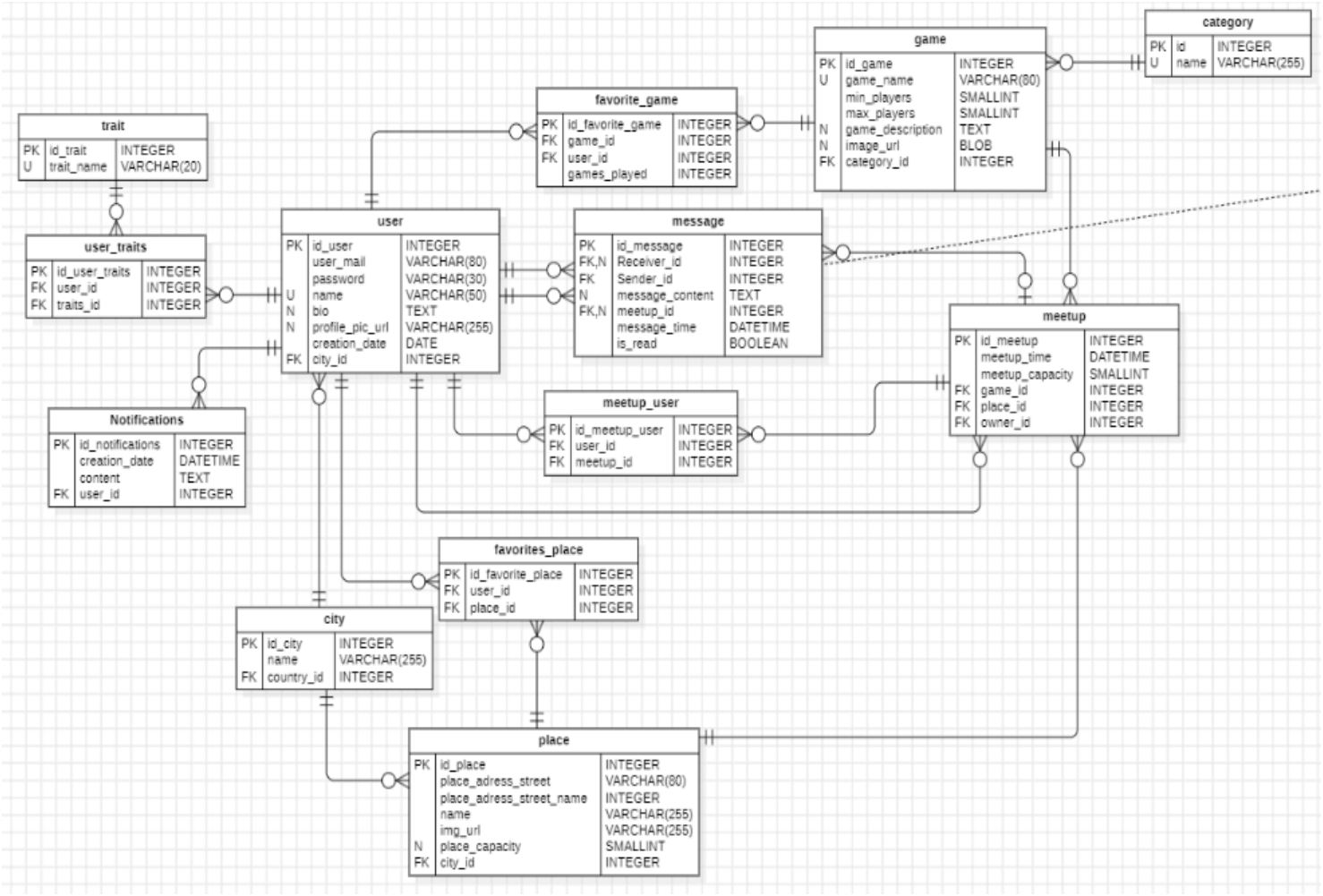
- Security Bundle, essentiel pour mettre en place une authentification sécurisée, la gestion des rôles et le hashage des mots de passe.

```
composer require security
```

J'ai également installé API Platform au début du projet. Cependant, par volonté d'apprentissage et sur les conseils d'un formateur, j'ai décidé de ne pas l'intégrer dans mon application. Cette décision s'est révélée particulièrement formatrice malgré le temps supplémentaire nécessité et m'a permis d'être précis sur les endpoints exposés.

2.2 Conception et implémentation de la base de données

Le modèle de données a évolué progressivement pour s'adapter aux besoins fonctionnels du projet. Voici le Modèle Physique de Données final :



Ce diagramme contient les entités suivantes:

- USER pour les informations de l'utilisateur.
- TRAIT pour la gestion des traits de caractères liés aux utilisateurs (curieux, empathique, rigoureux, discret, spontané, …).
- NOTIFICATIONS pour les notifications concernant un utilisateur.

- PLACE pour les lieux de rencontres disponibles dans l'application.
- CITY pour les villes, liées aux utilisateurs mais aussi aux lieux.
- GAME pour les jeux disponibles dans l'application.
- CATEGORY pour catégoriser les jeux.
- MEETUP pour les événements organisés, liés aux joueurs participants, à l'organisateur, à un lieu et à un jeu.
- MESSAGE qui peuvent concerner soit deux utilisateurs entre eux, soit plusieurs utilisateurs au sein d'un événement pour un chat de groupe.

Une fois le schéma validé, nous pouvons générer les entités correspondantes avec le Maker. Chaque entité est créée avec sa classe dédiée, ses propriétés(simples ou relationnelles), leurs getters/setters et son repository:

```
$ php bin/console make:entity
```

L'environnement est configuré dans un fichier .env.local, où sont définies les variables propres au projet, notamment celle concernant la base de données:

```
DATABASE_URL="mysql://root:@127.0.0.1:3306/dicemeet_API?serverVersion=9.1.0&charset=utf8mb4"
```

Cette variable contient toutes les informations nécessaires à notre ORM pour créer la base de données:

```
$ php bin/console doctrine:database:create
```

Une fois toutes les entités générées, la migration des changements peut être créée avec la commande:

```
php bin/console make:migration
```

Puis l'appliquer à la base de données:

```
$ php bin/console doctrine:migrations:migrate
```

Ce processus a été répété tout au long du projet pour intégrer les ajustements nécessaires à la structure de la base de données.

2.3 Ajout des fixtures

Pour simuler en un environnement réel, nous utilisons les fixtures de Doctrine afin de générer des données de test. Nous allons pour ca avoir besoin d'installer

DoctrineFixturesBundle :

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

Nous allons aussi utiliser la librairie Faker:

```
$ composer require --dev fakerphp/faker
```

Une fois le fichier de fixtures généré, nous préparons les données de test à insérer dans la base. Dans cet extrait, 100 utilisateurs fictifs sont créés en utilisant Faker pour générer leur email, mot de passe sécurisé, nom, date de création, ville, biographie et image de profil (à partir d'une banque d'avatars factices). Des relations sont également ajoutées, comme les traits de personnalité, les jeux favoris ou les lieux favoris.

```php
$users = [];
for ($i = 0; $i < 100; $i++) {
    $user = new User();
    $user->setEmail($faker->email())
        ->setPassword($this->hasher->hashPassword($user, $faker->password()))
        ->setName($faker->name())
        ->setCreationDate(DateTimeImmutable::createFromMutable($faker->dateTimeBetween('-1 years', 'now')))
        ->setCity($cities[array_rand($cities)])
        ->setBio($faker->paragraph())
        ->setImgUrl("https://i.pravatar.cc/150?u=" . $faker->unique()->numberBetween(1, 1000))
        ->setRoles(['ROLE_USER']);

    // Add traits
    $traitCount = rand(0, 4);
    for ($j = 0; $j < $traitCount; $j++) {
        $user->addPersonalityTrait($traits[array_rand($traits)]);
    }
    // Add favorites games
    $favoriteGameCount = rand(0, 3);
    for ($j = 0; $j < $favoriteGameCount; $j++) {
        $favGame = new FavoriteGame();
        $favGame->setUser($user)
            ->setGame($games[array_rand($games)])
            ->setGamesPlayed($faker->numberBetween(1, 200));

        $manager->persist($favGame);
    }


    // Add favorite places
    $favoritePlaceCount = rand(0, 3);
    for ($j = 0; $j < $favoritePlaceCount; $j++) {
        $favPlace = new FavoritePlace();
        $favPlace->setUser($user)
            ->setPlace($places[array_rand($places)]);
        $manager->persist($favPlace);
    }

    $manager->persist($user);
    $users[] = $user;
}
```

2.4 Sécurité et authentification

La sécurité de notre API repose sur deux éléments principaux : l'authentification via JWT et le contrôle d'accès appliqué à l'ensemble des routes. Ces deux éléments garantissent que seules les personnes autorisées peuvent consulter,modifier, ou supprimer des ressources.

Authentification JWT

Pour le JWT, nous avons besoin d'installer LexikJWTAuthentificationBundle:

```
$ composer require lexik/jwt-authentication-bundle
```

Nous pouvons ensuite générer une clé publique et une clé privée:

```
$ php bin/console lexik:jwt:generate-keypair
```

La clé publique permet de chiffrer les données et la clé privée de les déchiffrer. Lorsqu'un utilisateur s'authentifie via la route /api/login_check, Symfony vérifie ses identifiants et génère donc un JWT signé avec la clé privée, qui sera utilisé dans chaque requête vers une route protégée avec le header Authorization: Bearer <token>.

Contrôle d'accès

L'ensemble des règles de sécurité de l'API est défini dans le fichier security.yaml que nous allons détailler.

Le bloc password_hashers utilise la configuration auto, qui délègue à Symfony le choix du meilleur algorithme de chiffrement disponible pour hasher les mots de passe.

```yaml
password_hashers:
    Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
```

Le provider app_user_provider repose sur l'entité USER et identifie chaque utilisateur via son adresse email, ce qui permet la récupération automatique d'un utilisateur à partir du JWT:

```yaml
providers:
    # used to reload user from session & other features (e.g. switch_user)
    app_user_provider:
        entity:
            class: App\Entity\User
            property: email
```

La sécurité de l'API s'appuie sur plusieurs firewalls:

- admin: réservé aux utilisateurs possédant le rôle ROLE_ADMIN.
- login: dédié aux requêtes d'authentification JSON sur /api/login
- api : protège toutes les routes préfixées par /api/private avec JWT
- public : routes accessibles sans authentification

```yaml
firewalls:
    admin:
        pattern: ^/admin
        provider: app_user_provider
        lazy: true
        form_login:
            login_path: app_login
            check_path: app_login
        logout:
            path: app_logout
            target: /login
        #anonymous: true
    login:
        pattern: ^/api/login
        stateless: true
        json_login:
            check_path: /api/login_check
            username_path: email
            password_path: password
            success_handler: lexik_jwt_authentication.handler.authentication_success
            failure_handler: lexik_jwt_authentication.handler.authentication_failure

    api:
        pattern: ^/api/private
        stateless: true
        jwt: ~
    public:
        pattern: ^/api/public
        stateless: true
        security: false
```

Enfin, le bloc access_control définit précisément les droits d'accès selon les préfixes d'URL:

```yaml
access_control:
    - { path: ^/api/login, roles: PUBLIC_ACCESS }
    - { path: ^/api/public,       roles: PUBLIC_ACCESS }
    - { path: ^/api/private,       roles: IS_AUTHENTICATED_FULLY }
    - { path: ^/admin,       roles: ROLE_ADMIN }
```

2.5 Présentation de fonctionnalités métier

2.5.1 Mise à jour d'un meetup

Les fonctionnalités suivantes illustrent les aspects les plus représentatifs du back-end : logique métier, traitement des données, sécurité, événements et sérialisation.

Contrôleur

Le contrôleur expose une route PATCH sécurisée permettant la mise à jour partielle d'un meetup. Il gère:

- La vérification de l'existence du meetup
- La vérification des droits via un Voter
- La validation des données reçues
- Le déclenchement d'événements en cas de modifications(lieu,date,etc)
- La persistance des modifications

```php
#[Route('/api/private/events/{id}', name: 'app_meetup_update', methods: ['PATCH'])]
public function updateEvent(MeetupRepository $meetupRepository,
int $id,
Request $request,
EntityManagerInterface $em,
PlaceRepository $placeRepository,
GameRepository $gameRepository,
EventDispatcherInterface $eventDispatcher): Response
{
    $event = $meetupRepository->find($id);

    if (!$event) {
        return $this->json(['error' => 'Event not found'], 404);
    }

    $this->denyAccessUnlessGranted('MEETUP_UPDATE', $event);

    $data = json_decode($request->getContent(), true);

    if (!$data) {
        return $this->json(['error' => 'Invalid JSON'], 400);
    }

    if (isset($data['date'])) {
        $date = \DateTimeImmutable::createFromFormat('Y-m-d\TH:i', $data['date']);
        if (!$date) {
            return $this->json(['error' => 'Invalid date format. Expected Y-m-d H:i:s'], 400);
        }
        $eventDispatcher->dispatch(new DateUpdateMeetupEvent($event),DateUpdateMeetupEvent::NAME);
        $event->setTime($date);
    }

    if (isset($data['place'])) {
        $place = $placeRepository->findOneBy(['name' => $data['place']]);
        if (!$place) {
            return $this->json(['error' => 'Place not found'], 404);
        }
        $eventDispatcher->dispatch(new PlaceUpdateMeetupEvent($event),PlaceUpdateMeetupEvent::NAME);
        $event->setPlace($place);
    }

    $em->persist($event);
    $em->flush();

    return $this->json($event, 200);
}
```

Voter

L’accès à la ressource est contrôlé par:

```php
$this->denyAccessUnlessGranted('MEETUP_UPDATE', $event);
```

Puis la logique de décision est déléguée au MeetupVoter, qui autorise l’action uniquement si l’utilisateur est administrateur ou s’il est créateur de l'événement.

```php
final class MeetupVoter extends Voter
{
    public function __construct(private Security $security){}

    public const UPDATE = 'MEETUP_UPDATE';
    public const DELETE = 'MEETUP_DELETE';

    protected function supports(string $attribute, mixed $subject): bool
    {

        return in_array($attribute, [self::UPDATE, self::DELETE])
            && $subject instanceof Meetup;
    }

    protected function voteOnAttribute(string $attribute, mixed $subject, TokenInterface $token): bool
    {
        $user = $token->getUser();

        if (!$user instanceof UserInterface || $subject->getOwner() !== $user) {
            return false;
        }

        if($this->security->isGranted('ROLE_ADMIN')) {
            return true;
        }
        if($subject->getOwner() === $user) {
            return true;
        }
        return false;
    }
}
```

Système d'événement

Lorsqu'une modification est effectuée, un événement spécifique est déclenchée manuellement via l’EventDispatcher, par exemple dans le cas de la modification de la date:

```php
$eventDispatcher->dispatch(new DateUpdateMeetupEvent($event),DateUpdateMeetupEvent::NAME);
```

Ainsi l'événement suivant est émis:

```php
class DateUpdateMeetupEvent
{
    public const NAME = 'meetup.date.updated';
    public function __construct(public Meetup $meetup)
    {
    }
    public function getMeetup(): Meetup
    {
        return $this->meetup;
    }
}
```

Cet événement est écouté par un subscriber qui sera chargé de notifier les utilisateurs concernés via la création d'une Notification:

```php
class MeetupUpdateSubscriber implements EventSubscriberInterface
{
    public function __construct(
        private EntityManagerInterface $em
    ) {
    }
    public function onMeetupDateUpdated($event): void
    {
        $notifiedUsers = $event->getMeetup()->getUsers();
        foreach ($notifiedUsers as $user) {
            if ($user !== $event->getMeetup()->getOwner()) {
                $notification = new Notification();
                $notification->setUser($user);
                $notification->setContent(
                    "La date de l'evenement n°" . $event->getMeetup()->getId() . " de'"
                $notification->setCreatedAt(new \DateTimeImmutable());
                $user->addNotification($notification);
                $this->em->persist($notification);
                $this->em->persist($user);
            }
        }
        $this->em->flush();
    }
```

Normalizer

La réponse retournée par return $this->json($event) est traitée par un normalizer personnalisé qui override le normalizer de base, ce qui permet de formater l'objet retourné au front-end sans exposer l'intégralité de l'entité, tout en contrôlant l'ordre,le nom et le contenu des propriétés retournées :

```php
class MeetupNormalizer implements NormalizerInterface
{
    public function __construct(
        #[Autowire(service: 'serializer.normalizer.object')]
        private NormalizerInterface $normalizer
    ) {
    }

    public function normalize($object, ?string $format = null, array $context = []): array
    {
        // $data = $this->normalizer->normalize($object, $format, $context);

        $data["id"] = $object->getId();
        $data['game'] = $object->getGame()->getName();
        $data['place'] = $object->getPlace()->getName();
        $data['city'] = $object->getPlace()->getCity()->getName();
        $data['date'] = $object->getTime()->format('Y-m-d H:i:s');
        $data['participants'] = count($object->getUsers());
        $data['maxParticipants'] = $object->getCapacity();
        $data['ownerId'] = $object->getOwner()->getId();


        return $data;
    }

    public function supportsNormalization($data, ?string $format = null, array $context = []): bool
    {
        return $data instanceof Meetup;
    }

    public function getSupportedTypes(?string $format): array
    {
        return [Meetup::class => true];
    }
}
```

2.5.2 Mise à jour du profil utilisateur

La mise à jour du profil utilisateur constitue une fonctionnalité centrale car elle combine plusieurs aspects techniques :

- La gestion de formats de requête différents (JSON et FormData)
- La mise à jour des informations textuelles et relationnelles
- L'upload sécurisé d'un fichier (image de profil)

Cette fonctionnalité est exposée via une route sécurisée /api/private/users/me accessible uniquement à l'utilisateur authentifié.

Selon le type de requête reçue, l'API extrait les données soit depuis $request soit depuis le body JSON:

```php
#[Route('/api/private/users/me', name: 'user_update', methods: ['PATCH', 'POST'])]
public function updateUser(
    Request $request,
    CityRepository $cityRepository,
    PersonalityTraitRepository $personalityTraitRepository,
    EntityManagerInterface $em
): Response {
    /** @var EntityUser $user */
    $user = $this->getUser();

    if (!$user) {
        return $this->json(['error' => 'Utilisateur non connecté'], 401);
    }

    $contentType = $request->headers->get('Content-Type') ?? '';

    if (strpos($contentType, 'multipart/form-data') !== false) {
        $data = $request->request->all();
        $files = $request->files;
    } else {
        $data = json_decode($request->getContent(), true) ?? [];
        $files = $request->files;
    }
```

Une fois les données de la requête extraites, le contrôleur met à jour les différentes propriétés de l'utilisateur. Dans le cas des traits par exemple, les données préexistantes sont vidées puis reconstruites:

```php
if (isset($data['traits'])) {
    $traitsArray = is_string($data['traits']) ? json_decode($data['traits'], true) : $data['traits'];
    if (is_array($traitsArray)) {
        foreach ($user->getPersonalityTraits() as $t) {
            $user->removePersonalityTrait($t);
        }
        foreach ($traitsArray as $inputTrait) {
            $id = $inputTrait['id'] ?? $inputTrait;
            $trait = $personalityTraitRepository->find($id);
            if ($trait) {
                $user->addPersonalityTrait($trait);
            }
        }
    }
}
```

Ensuite, lorsque le champ img est présent dans la requête multipart, le back-end vérifie la validité du fichier, génère un nom unique puis déplace le fichier dans le répertoire dédié aux uploads:

```php
if ($files->has('img')) {
    $file = $files->get('img');
    if ($file && $file->isValid()) {
        $extension = $file->guessExtension() ?: $file->getClientOriginalExtension() ?: 'bin';
        $fileName = uniqid() . '.' . $extension;

        $file->move(
            $this->getParameter('uploads_directory'),
            $fileName
        );

        $user->setImgUrl($fileName);
    }
}
```

Enfin, l'utilisateur mis à jour est persisté puis renvoyé, en passant par un normalizer dédié à l'entité User, dont voici un extrait:

```php
public function normalize($object, ?string $format = null, array $context = []): array
{
    $baseUrl = 'http://localhost:8000'; // A CHANGER SI DEPLOIEMENT SERVEUR
    $maxfavorites = 3; // Nombre maximum de jeux et bars favoris à afficher

    $data["id"] = $object->getId();
    $data["name"] = $object->getName();
    $data["email"] = $object->getEmail();
    $data["createdAt"] = $object->getCreationDate()->format('Y-m-d ');
    $data['city'] = $object->getCity() ? $object->getCity()->getName() : '';

    // On differencie les users fixtures avec les vrais users
    if ($data['id'] > 100) {
        $data['imgUrl'] = $baseUrl . '/uploads/' . $object->getImgUrl();
    }else {
        $data['imgUrl'] = $object->getImgUrl();
    }
```

2.5.3 Génération de suggestions pour la barre de recherche

Cette fonctionnalité illustre l'utilisation de query parameters et des requêtes personnalisées dans les repositories. Le contrôleur analyse les paramètres fournis (type,query) afin de sélectionner le type de recherche à effectuer, permettant de suggérer des résultats à l'utilisateur lorsqu'il effectue une recherche coté front-end:

```php
#[Route('/api/public/search', name: 'app_search', methods: ['GET','OPTIONS'])]
public function search(Request $request,
CityRepository $cityRepository,
GameRepository $gameRepository,
PlaceRepository $placeRepository): Response
{
    $query = $request->query->get('search','');
    $type = $request->query->get('type','');
    $results = [];


    $cityResults = $cityRepository->searchByName($query);
    foreach ( $cityResults as $city) {
        if ($type == 'city' || $type == '') {
            $results[] = [
            'id' => $city->getId(),
            'name' => $city->getName(),
            'type' => 'ville'
        ];
        }
    }
```

Pour optimiser la recherche, une méthode dédiée a été ajouté aux repositories grace a QueryBuilder, permettant de filtrer les résultats :

```php
public function searchByName(string $query): array
{
    return $this->createQueryBuilder('c')
        ->where('c.name LIKE :query')
        ->setParameter('query', '%' . $query . '%')
        ->setMaxResults(10)
        ->getQuery()
        ->getResult();
}
```

2.6 EasyAdmin

Afin d'apporter une interface de gestion complète des données à nos utilisateurs ayant le ROLE_ADMIN, nous allons installer le bundle EasyAdmin qui permet d'obtenir un panneau d'administration:

```
$ composer require easycorp/easyadmin-bundle
```

Nous pouvons ensuite créer un contrôleur pour notre dashboard administrateur:

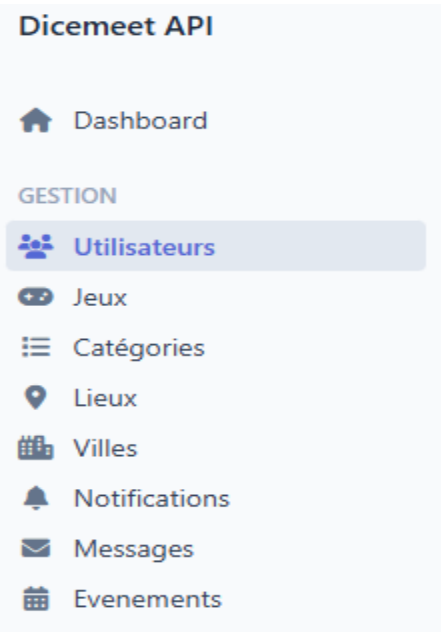
```
$ php bin/console make:admin:dashboard
```

Puis enfin créer un contrôleur CRUD pour chaque entité que nous voulons faire apparaître dans notre interface d'administration:

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

Nous pouvons ensuite personnaliser l'organisation du tableau de bord afin d'afficher uniquement les entités pertinentes pour l'administration. Pour chacune, EasyAdmin génère automatiquement un ensemble d'opérations CRUD :

```php
public function configureMenuItems(): iterable
{
    yield MenuItem::linkToDashboard('Dashboard', 'fa fa-home');
    yield MenuItem::section('Gestion');
    yield MenuItem::linkToCrud('Utilisateurs', 'fas fa-users', User::class);
    yield MenuItem::linkToCrud('Jeux', 'fas fa-gamepad', Game::class);
    yield MenuItem::linkToCrud('Catégories', 'fas fa-list', Category::class);
    yield MenuItem::linkToCrud('Lieux', 'fas fa-map-marker-alt', Place::class);
    yield MenuItem::linkToCrud('Villes', 'fas fa-city', City::class);
    yield MenuItem::linkToCrud('Notifications', 'fas fa-bell', Notification::class);
    yield MenuItem::linkToCrud('Messages', 'fas fa-envelope', Message::class);
    yield MenuItem::linkToCrud('Evenements', 'fas fa-calendar-alt', Meetup::class);
}
```



EasyAdmin nous permet désormais de gérer facilement les données de l'application sans passer par l'API et d'offrir une interface interne facile à prendre en main pour les utilisateurs.

3. Réalisations Front-End

3.1 Conception des maquettes(cf annexes)

La conception du front-end a commencé par la réalisation de trois écrans clés :

- Landing page : elle présente plusieurs appels à l'action encourageant l'inscription, met en avant quelques événements populaires et propose une section de confiance ainsi qu'un aperçu des principales catégories de jeux de société.

- Liste des événements : cette page intègre une barre de recherche permettant de filtrer les meetups et une liste paginée d'événements, présentée sous forme de cartes pour faciliter la navigation.

- Messagerie : elle affiche une barre latérale regroupant l'ensemble des conversations privées de l'utilisateur, classées chronologiquement, permettant de basculer rapidement d'une discussion à l'autre.

L'objectif était d'obtenir une interface sobre, lisible et cohérente avec l'univers des jeux de société, sans complexité visuelle inutile.
Les maquettes et wireframes ont été réalisés sous Figma et ont servi de base pour structurer la navigation, l'organisation des blocs et les composants principaux développés ensuite dans Angular. Le design a été modifié par la suite pour s'adapter aux besoins du projet.

3.2 Initialisation du projet Angular

Le front-end a été initialisé à l'aide d'Angular CLI, qui génère automatiquement la structure de base de l'application à l'aide de la commande:

```
$ ng new DiceMeet
```

Plusieurs fichiers essentiels sont créés dès l'initialisation, notamment :

- index.html :point d'entrée de l'application contenant la balise <app-root>, qui sert d'ancrage à l'interface Angular dans le cadre du fonctionnement en Single

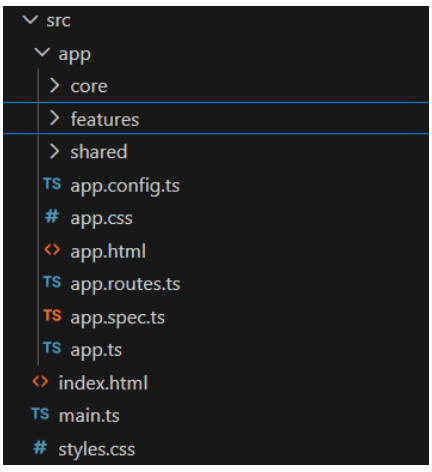
Page Application.

- styles.css : fichier de styles global regroupant les règles communes à toutes les pages (variables CSS, tailles de police, couleurs, etc.).
- angular.json : fichier central de configuration d'Angular CLI, regroupant les paramètres de build, les chemins et les options utilisées par le framework.
- package.json : liste des dépendances et métadonnées du projet, gérées via npm.
- src/app/app.routes.ts : fichier définissant les routes principales de l'application.
- src/app/app.config.ts : fichier de configuration global d'Angular, chargé au démarrage de l'application.

Cette structure initiale a ensuite été organisée et enrichie afin de poser les bases du développement des différentes pages, composants et services du projet.

3.3 Architecture globale de l'application Angular

La structure du projet Angular repose principalement sur l'organisation du dossier src. La capture ci-dessous présente sa structure initiale et les dossiers qui y ont été ajoutés au fur et à mesure du projet.



L'architecture s'articule autour de trois dossiers principaux :

- core : contient l'ensemble des services de l'application, notamment le service d'authentification et ceux permettant d'interagir avec les entités de l'API Symfony (utilisateurs, événements, messages, etc.).

- features : regroupe les composants correspondant aux pages majeures de l'application. Chaque page comporte ses propres fichiers : template HTML, logique TypeScript, styles et fichier de tests.

- shared : rassemble les composants réutilisables (header, footer, searchbar, chatbox…), ainsi que le dossier models, qui contient les interfaces TypeScript construites à partir des réponses normalisées de l'API Symfony.

De nouveaux composants et services ont été générés tout au long du développement via Angular CLI afin d'enrichir progressivement la structure du projet:

```
$ ng generate component features/home-component
```

```
$ ng generate service core/services/game-service
```

Mais avant d'implémenter les différentes pages, la première étape essentielle a été de mettre en place le système d'authentification, indispensable pour sécuriser la majorité des routes de l'application.

3.4 Authentification et sécurité

L'authentification repose entièrement sur les tokens JWT générés par le back-end Symfony. Le front-end doit donc pouvoir gérer son obtention, son stockage et son ajout automatique aux requêtes ainsi que la protection des routes nécessitant une authentification.

AuthService

L'AuthService centralise toute la logique liée à l'authentification. C'est lui qui communique avec l'API Symfony, gère le stockage du token et informe le reste de l'application de l'état de connexion de l'utilisateur.

Les principales méthodes qu'il possède sont les suivantes :

- login : Envoie une requête POST à /api/login_check avec dans le corps de la requête un objet {email , motdepasse}. Si l'authentification réussit, l'API renverra un token JWT.

- register : Permet de créer un nouveau compte via la route /api/public/register et le back se chargera de la validation et de la création du compte.

- saveToken : Stocke le JWT dans localStorage et met à jour un BehaviorSubject permettant aux composants de réagir instantanément à l'état de l'authentification.

- getToken : Récupère le token stocké localement pour l'utiliser dans les requêtes protégées.

- logout : Supprime le token du stockage local et notifie l'application que l'utilisateur n'est plus authentifié via le BehaviorSubject.

```
login(credentials: { email: string; password: string }) {
  return this.http.post<{ token: string }>(`${this.baseUrl}/login_check`, credentials);
}

register(data: { email: string; name: string; password: string }) {
  return this.http.post<{ success?: boolean; error?: string }>(
    `${this.baseUrl}/public/register`,
    data
  );
}

saveToken(token: string) {
  localStorage.setItem('jwt_token', token);
  this.loggedIn$.next(true);
}

getToken(): string | null {
  return localStorage.getItem('jwt_token');
}

logout() {
  localStorage.removeItem('jwt_token');
  this.loggedIn$.next(false);
}
```

Grâce à ces méthodes, toute la logique d'authentification reste centralisée dans un unique service. Les composants (comme Login ou Header) n'ont ainsi qu'à appeler des méthodes simples, sans gérer eux-mêmes la persistance ou le traitement du JWT.

Interceptor

Le rôle de l'interceptor est d'intercepter toute requête sortante de notre front pour lui ajouter, si on en dispose, du token JWT dans le header :

```
const token = authService.getToken();
if (token) {
  req = req.clone({
    setHeaders: { Authorization: `Bearer ${token}` }
  });
}
```

AuthGuard

L'authguard nous permet d'empêcher l'utilisateur d'accéder aux pages gardées si le token d'authentification n'est pas présent ou valide, et de le rediriger via une méthode canActivate :

```
canActivate(): boolean {
  if (this.auth.isAuthenticated()) {
    return true;
  }
  this.router.navigate(['/login']);
  return false;
}
```

Ainsi ce guard peut être appliqué lors de la configuration d'une route dans le app.routes.ts :

```
{ path: 'dashboard', component: DashboardComponent, canActivate: [AuthGuard] },
```

Login et subscribe components

La page de login fait partie des rares pages accessibles sans authentification.

Son composant s'appuie sur un formulaire réactif composé d'un champ email et d'un champ mot de passe, chacun associé à des validateurs Angular:

```
<main>
    <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
        <label for="email">Adresse mail</label>
        <input type="email" id="email" formControlName="email">

        <label for="password">Mot de passe</label>
        <input type="password" id="password" formControlName="password">

        @if (loginForm.errors?.['invalidLogin']) {
        <div>
            <small>Email ou mot de passe incorrect.</small>
        </div>
        }

        <button type="submit" [disabled]="loginForm.invalid">Connexion</button>
    </form>

    <p class="subscribeCTA">Vous n'avez pas encore de compte ? <a routerLink="/subscribe">Créez en un</a></p>
</main>
```

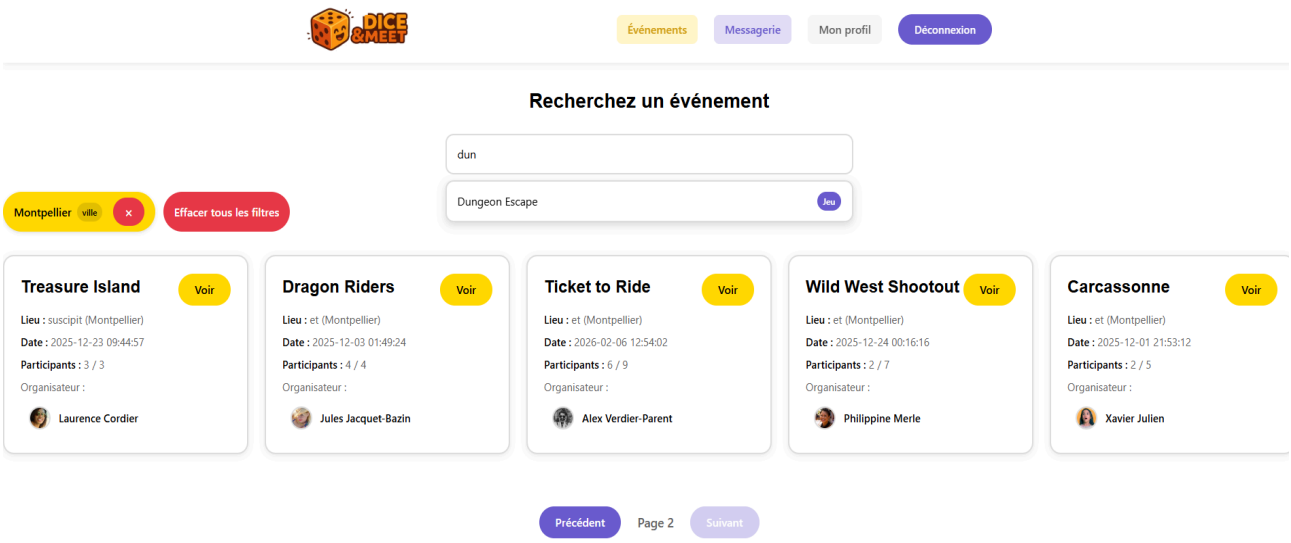
Lors de la soumission du formulaire, on vérifie la validité des champs, on enregistre le token en cas de succès et on redirige vers la page d'accueil. On affichera une erreur si l'API renvoie un statut 401:

```
export class LoginComponent {
  loginForm = new FormGroup({
    email: new FormControl('', [Validators.required, Validators.email]),
    password: new FormControl('', Validators.required)
  })

  authService = inject(AuthService);
  router = inject(Router);

  onSubmit() {
    if (this.loginForm.valid) {
      this.authService.login({
        email: this.loginForm.value.email!,
        password: this.loginForm.value.password!
      }).subscribe({
        next: (response) => {
          this.authService.saveToken(response.token);
          this.router.navigate(['/']);
        },
        error: (err) => {
          if (err.status === 401) {
            this.loginForm.setErrors({ invalidLogin: true });
          }
        }
      });
    }
  }
```

3.5 Présentation de fonctionnalités pertinentes

3.5.1 Page de liste des événements : recherche dynamique, filtres et pagination

La page de liste des meetups permet à l'utilisateur de rechercher un événement, d'ajouter des filtres (jeu, lieu, ville…) et de parcourir les résultats à l'aide d'une pagination simple et lisible.



La SearchBar écoute la saisie utilisateur, applique un debounce pour limiter les requêtes, puis interroge le back:

```
<div class="search-container">
  <input
    type="text"
    [(ngModel)]="searchTerm"
    (input)="onSearchChange()"
    [placeholder]="getPlaceholder()"
    class="search-input"
  />

  @if (suggestions$ | async ; as suggestions) {
    @if (suggestions.length > 0) {
      <ul class="suggestions">
        @for (filter of suggestions; track $index) {
          <li (click)="selectItem(filter)">
            {{ filter.name }}
            <span class="label">{{ filter.type }}</span>
          </li>
        }
      </ul>
    }
  }
</div>
```

```
suggestions$ = this.searchInput$.pipe(
  debounceTime(300),
  switchMap(query => {
    if (!query) return of([]);

    if (this.searchType) {
      return this.searchService.search(query, this.searchType);
    }

    return this.searchService.search(query);
  })
);

onSearchChange() {
  this.searchInput$.next(this.searchTerm);
}

selectItem(filter: Filter) {
  this.filtersChanged.emit(filter);
  this.searchTerm = '';
  this.searchInput$.next('');
}
```

Ainsi, lors d'une recherche, le front appelle une route dédiée à la recherche via le SearchService.Cet endpoint renvoie des objets Filter { id, name, type }, facilement exploitables par le composant parent. Le composant parent peut manipuler les suggestions sous forme de filtres et déclencher la mise à jour des meetups affichés:

```
public search(query: string, searchType?: 'games' | 'places' | 'city'): Observable<Filter[]> {
  if (searchType) {
    return this.http.get<Filter[]>(`${this.apiUrl}?search=${query}&type=${searchType}`);
  }
  return this.http.get<Filter[]>(`${this.apiUrl}?search=${query}`);
}
```

Puis le composant parent peut gérer les filtres sélectionnés et mettre à jour la liste des meetups:

```
addfilter(filter: Filter) {
  if (!this.filters.includes(filter)) {
    this.filters.push(filter);
    this.currentPage = 1;
    this.UpdateMeetups();
  }
}

deleteFilter(filter: Filter) {
  this.filters = this.filters.filter(f => f.name !== filter.name || f.type !== filter.type);
  this.currentPage = 1;
  this.UpdateMeetups();
}

UpdateMeetups() {
  this.meetups$ = this.filters.length > 0 ? this.eventService.getFilteredEvents(this.filters) : this.eventService.getAll();
  this.meetups$ = this.meetups$.pipe(
    map(meetups => {
      const start = (this.currentPage - 1) * this.pageSize;
      const end = start + this.pageSize;
      this.meetupNumber = meetups.length;
      return meetups.slice(start, end);
    })
  );
}
```

Enfin, les meetups seront affichés via le composant <app-meetup-card> et un message apparaîtra si aucun résultat ne correspond aux filtres. La pagination est elle aussi gérée:

```
@if (meetupNumber > pageSize ) {
<div class="pagination">
  <button (click)="changePage(currentPage - 1)" [disabled]="currentPage === 1">Précédent</button>
  <span>Page {{ currentPage }}</span>
  <button (click)="changePage(currentPage + 1)" [disabled]="currentPage * pageSize >= meetupNumber">Suivant</button>
</div>
}
```

```
changePage(page: number) {
  this.currentPage = page;
  this.UpdateMeetups();
}
```

3.5.2 Page de profil : affichage et mise à jour

La page de profil permet à l'utilisateur de consulter et modifier ses informations personnelles : image, nom, biographie, ville, traits de personnalité ainsi que ses jeux et lieux favoris.

Au chargement de la page, un FormGroup est initialisé avec les informations du profil récupérées via le UserService; Cela permet d'afficher immédiatement le profil et préparer les champs à une modification:

```
async ngOnInit() {
  this.user = await firstValueFrom(this.userService.getCurrentUser());

  this.profileForm.patchValue({
    img: this.user.imgUrl,
    name: this.user.name,
    bio: this.user.bio,
    city: this.user.city,
    traits: this.user.traits ?? []
  });
}
```

L'utilisateur peut sélectionner une nouvelle image, qui est immédiatement affichée grâce à un FileReader:

```
onFileSelected(event: Event): void {
  const input = event.target as HTMLInputElement;
  if (input.files && input.files.length > 0) {
    const file = input.files[0];
    this.profileForm.patchValue({ img: file });
    this.profileForm.get('img')?.markAsDirty();

    const reader = new FileReader();
    reader.onload = () => {
      this.imgPreview = reader.result as string;
    };
    reader.readAsDataURL(file);
  }
}
```

L’utilisateur peut gérer jusqu'à trois traits parmi ceux disponibles. Lorsqu’il modifie un slot ou en ajoute un, la valeur du champ de formulaire traits est marquée comme modifiée:

```
addTrait(trait: Trait) {
  const currentTraits: Trait[] = this.profileForm.get('traits')?.value || [];
  if (!currentTraits.includes(trait)) {
    if (this.currentTraitSlot !== null) {
      currentTraits[this.currentTraitSlot] = trait;
    } else {
      currentTraits.push(trait);
    }
    this.profileForm.get('traits')?.setValue(currentTraits);
    this.profileForm.get('traits')?.markAsDirty();
    this.selectTraits = false;
  }
}
```

Lors de l’enregistrement, seules les champs modifiés sont envoyés à l’API, ce qui permet de limiter les requêtes inutiles et de ne modifier que le nécessaire côté back-end:

```
  const changes: Partial<User> = {};
  Object.keys(this.profileForm.controls).forEach(key => {
    const control = this.profileForm.get(key);
    if (control?.dirty) {
      changes[key as keyof User] = control.value;
    }
  });
```

Une fois les modifications effectuées, si une nouvelle image est présente, on construit un FormData, sinon on envoie un JSON simple:

```
updateUser(data: Partial<User> | FormData) {
  if (data instanceof FormData) {
    data.append('_method', 'PATCH');
    return this.http.post<User>(`${this.apiUrl}/users/me`, data);
  } else {
    return this.http.patch<User>(
      `${this.apiUrl}/users/me`,
      data,
      { headers: { 'Content-Type': 'application/json' } }
    );
  }
}
```

3.5.3 Page de gestion d'événement

Cette page regroupe toutes les fonctionnalités avancées liées à un événement : affichage des informations, gestion des participants, modification de l'événement par l'organisateur et discussion via la chat.

On récupère tout d'abord les données liées à l'événement sous forme d'observables:

```
ngOnInit() {
  this.loadData();
}
loadData() {
  this.currentEvent$ = this.eventService.getEventById(this.eventId).pipe(
    catchError(err => {
      if (err.status === 404) {
        this.router.navigate(['/not-found']);
      }
      return of(undefined);
    })
  );
  this.currentUser$ = this.userService.getCurrentUser();
  this.eventUsers$ = this.eventService.getEventUsers(this.eventId);
  this.eventMessages$ = this.eventService.getEventMessages(this.eventId).pipe(
    catchError(() => of([]))
  );
}
```

La page adapte automatiquement l'affichage selon si l'utilisateur est organisateur ou participant et aussi s'il peut rejoindre l'événement (place disponible):

```
isOwner(event: Meetup | undefined, user: User | undefined): boolean {
  if (!event || !user) return false;
  return event.ownerId === user.id;
}
isParticipant(users: User[] | undefined, user: User | undefined): boolean {
  if (!users || !user) return false;
  return users.some(u => u.id === user.id);
}
isEventFull(event: Meetup | undefined): boolean {
  if (!event) return false;
  return event.participants >= event.maxParticipants;
}
getUserAction(event: Meetup | undefined, users: User[] | undefined, user: User | undefined):
 'loading' | 'canJoin' | 'isParticipant' | 'isFull' {
  if (!event || !user || !users) return 'loading';
  if (this.isParticipant(users, user)) return 'isParticipant';
  if (this.isEventFull(event)) return 'isFull';
  return 'canJoin';
}
```

L’organisateur peut aussi supprimer ou modifier l'événement. S'il souhaite modifier le lieu, le composant SearchBar sera à nouveau utilisé ici afin de suggérer des lieux en fonction de l’input utilisateur et de remplir automatiquement la ville grâce à PlaceService:

```
<app-search-bar [searchType]="'places'" (filtersChanged)="onPlaceFilterSelected($event)"></app-search-bar>
```

```
onPlaceFilterSelected(filter: Filter) {
  if (filter.type === 'lieu') {
    this.hasFormChanges = true;
    this.eventForm.get('place')?.setValue(filter.name);
    this.placeService.getPlaceById(filter.id).subscribe({
      next: (place) => {
        this.eventForm.get('city')?.setValue(place.cityName);
      },
      error: () => {
        alert('Erreur lors de la récupération du lieu');
      }
    });
  }
}
```

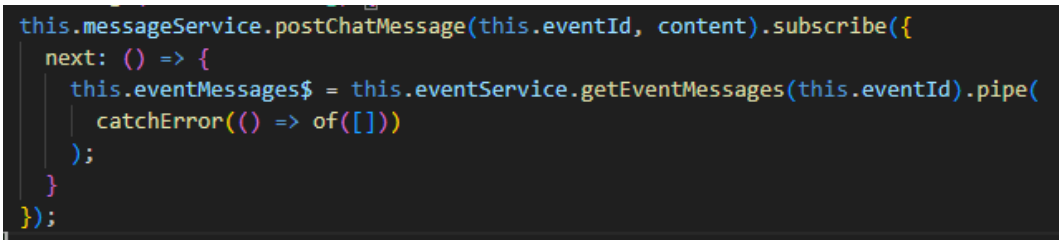
Une fois les modifications effectuées, les champs du FormGroup qui ont été édités seront envoyés à l’API:

```
this.eventService.updateEvent(this.eventId, updateData).subscribe({
  next: () => {
    alert('Événement modifié avec succès');
    this.cancelEdit();
    this.loadData();
  },
```

Les participants à l'événement sont affichés avec une distinction visuelle pour l’organisateur. Si l’utilisateur est organisateur, il peut exclure les utilisateurs:

```
@for (participant of users; track participant.id) {
  <div class="participant-item">
    @if (participant.id === event.ownerId) {
      <p><strong>{{ participant.name }} (Organisateur)</strong></p>
    } @else {
      <p>{{ participant.name }}</p>
      @if (isOwner(event, user)) {
        <button (click)="kickUser(participant.id)">Exclure</button>
      }
    }
```

La page inclut aussi le composant <app-chat-box> utilisé aussi dans la messagerie privée. Après chaque message envoyé, la liste est rechargée automatiquement:



```
this.messageService.postChatMessage(this.eventId, content).subscribe({
  next: () => {
    this.eventMessages$ = this.eventService.getEventMessages(this.eventId).pipe(
      catchError(() => of([]))
    );
  }
});
```

4. Déploiement et optimisation

Site consultable : https://dicemeet.pro/

4.1 Déploiement de l'application

Préparation du serveur

Le projet a été déployé sur un VPS OVH sous Ubuntu 25.04, configuré pour héberger à la fois l'API Symfony et l'application Angular compilée.

Nous avons aussi acheté un nom de domaine puis avons configuré les entrées DNS A pointant vers l'IP du VPS:

☐	dicemeet.pro.	0	A	51.254.207.219
☐	www.dicemeet.pro.	0	A	51.254.207.219

Nous pouvons désormais accéder à notre serveur en connexion ssh (ssh user@ip_serveur). Nous commençons par configurer Apache et les fichiers .conf contenant les VirtualHost afin de faire pointer les domaines vers les fichiers souhaités,par exemple pour le front-end:

```
<VirtualHost *:80>
    ServerName dicemeet.pro
    ServerAlias www.dicemeet.pro
    DocumentRoot /var/www/dicemeet/frontend

    <Directory /var/www/dicemeet/frontend>
```

```
<VirtualHost *:443>
    ServerName dicemeet.pro
    ServerAlias www.dicemeet.pro
    DocumentRoot /var/www/dicemeet/frontend

    <Directory /var/www/dicemeet/frontend>
```

Une fois le certificat HTTPS installé via Certbot, Apache bascule automatiquement en HTTPS grâce au second VirtualHost. Nous installons ensuite toutes les dépendances nécessaires au fonctionnement du front-end et de l'API (snap,PHP,Composer,MySQL,etc).

Back-end

Le back-end à été installé en clonant le repo git sur le serveur, puis en utilisant composer install. Nous n'avons plus qu'à configurer les variables d'environnement sur .env.local, utiliser doctrine pour reconstruire la base de données et générer les fixtures comme décrit plus tôt.

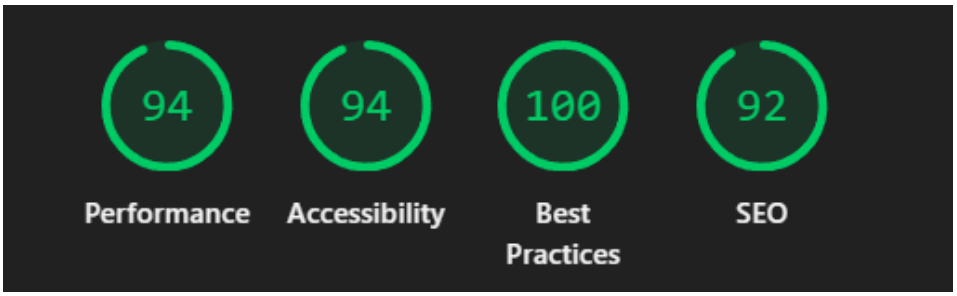
Nous avons aussi généré la paire de clés avec Lexik Bundle afin de pouvoir signer des tokens JWT. Enfin, nous avons dû configurer un fichier .htaccess dans le dossier public du back-end pour rediriger les requêtes vers index.php et transmettre correctement le header Authorization.

Front-end

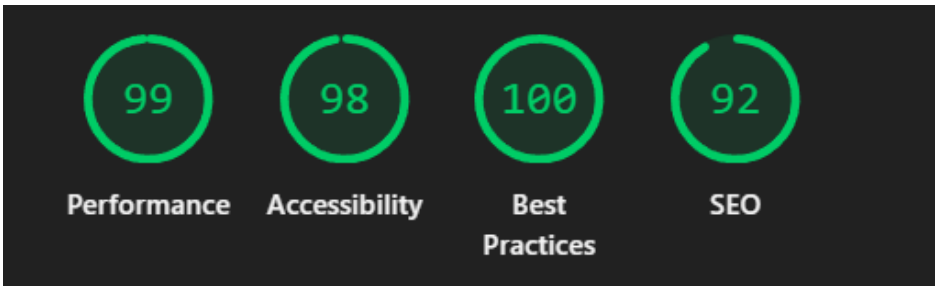
Le front-end a été directement compilé sur ma machine avec ng build. Les fichiers nécessaires contenus dans ce build ont été transférés sur mon serveur avec scp. Un fichier .htaccess a aussi été configuré pour bien gérer les rafraîchissements de page et la transmission du token JWT.

4.2 Optimisations

Nous avons ensuite optimisé l'application sur les différents paramètres mesurés avec LightHouse. Voici les résultats obtenus pour la page de Dashboard en desktop:



Et en mobile:



Accessibilité

Pour l'accessibilité, nous avons ajoutés des labels aria sur les éléments interactifs, par exemple sur mon header:

```
<nav [class.mobile-open]="mobileMenuOpen" role="navigation" aria-label="Menu principal">
    <a routerLink="/login" (click)="closeMobileMenu()">Connexion</a>
    <a routerLink="/subscribe" (click)="closeMobileMenu()">Inscription</a>
    <a routerLink="/events" (click)="closeMobileMenu()">Événements</a>
</nav>
```

Nous avons aussi systématiquement ajouté des attributs alt descriptifs:

```
<img src="assets/herosectionimg.webp" alt="Logo representant un dé de jeu">
```

Performance

Pour la performance, nous avons pris soin de convertir toutes les images en format webp, format adaptable et avec une compression optimale. Nous avons aussi mis en place HTTP/2 qui est indispensable pour de meilleurs résultats.

SEO

Pour le SEO, nous avons pris soin d'utiliser des éléments html sémantiques, de bien titrer nos pages et d'ajouter des balises meta description dans le head:

```
<meta property="og:title" content="DiceMeet - Trouvez des joueurs !">
  <meta name="description" content="Trouvez des joueurs pour vos jeux de société préférés avec DiceMeet. Créez des

<meta property="og:description" content="Trouvez des joueurs pour vos jeux de société préférés avec DiceMeet.">
<meta property="og:type" content="website">
<meta property="og:url" content="https://dicemeet.pro">
<meta property="og:image" content="assets/Logo.webp">
```

5. Présentation d’un jeu d'essai

Dans cette partie, nous allons tester de manière manuelle et automatisée la fonctionnalité d’inscription d’un utilisateur et détailler les résultats obtenus.

5.1 Test manuel

Etape 1: Accès à la page d’inscription

Action : Clic sur le bouton “Inscription” dans la barre de navigation.

Résultat attendu : Redirection vers la page d’inscription. L’url correspond à https://dicemeet.pro/subscribe et le formulaire d’inscription s’affiche.

Etape 2: Validation des champs du formulaire

Le formulaire comporte un champ email, un champ nom d’utilisateur et des champs mot de passe et confirmation de mot de passe.

Email

Action : Saisie de mail valide

Résultat attendu : Champ validé sans erreur

Action : Champ vide ou email non valide

Résultat attendu : Erreur affichée “Email requis” ou “Format invalide”

Nom utilisateur

Action : Saisie de nom valide

Résultat attendu : Champ validé sans erreur

Action : Saisie de nom de moins de 4 caractères

Résultat attendu : Erreur affichée “Minimum 4 caractères”

Mot de passe

Action : Saisie de mots de passe correspondants et valides

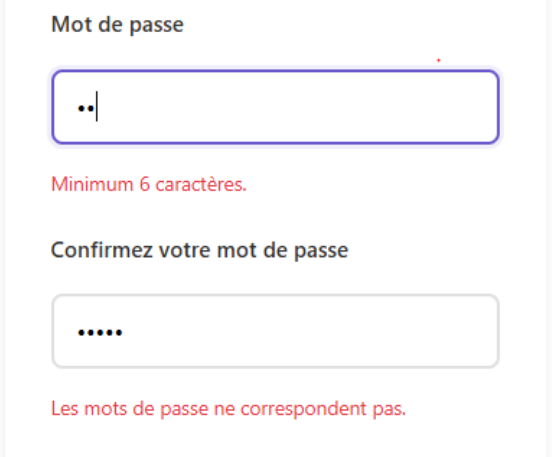
Résultat attendu : Champs validés sans erreur

Action : Saisie de mot de passe vide ou de moins de 6 caractères

Résultat attendu : Erreur affichée “Mot de passe requis” ou “Minimum 6 caractères”

Action : Saisie de mots de passe non correspondants

Résultat attendu : Erreur affichée “Les mots de passe ne correspondent pas.”



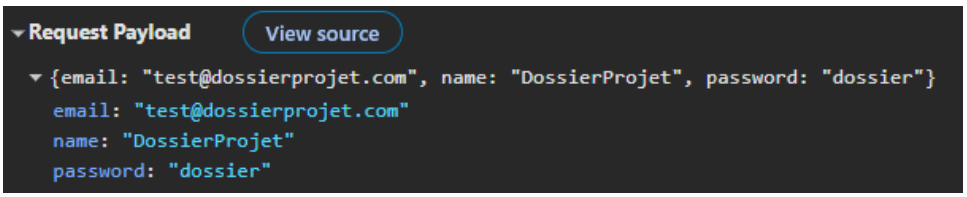
Étape 3: Soumission du formulaire

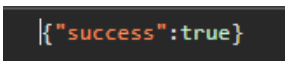
Action : Clic sur le bouton “Créer un compte” avec des champs non remplis ou invalides

Résultat attendu : Bouton désactivé et le formulaire ne pourra pas être soumis et transféré au back-end

Action : Clic sur le bouton “Créer un compte” avec des champs valides

Résultat attendu : Redirection vers page de login et requête envoyée, avec réponse back-end





Étape 4: Persistance en back-end

Action : Vérifier que l'utilisateur à été créé avec les bonnes informations en base de données avec un mot de passe hashé

Résultat attendu : Tous les champs sont enregistrés correctement et le mot de passe n'est pas en clair

```
mysql> SELECT * FROM user WHERE name="DossierProjet";
+-----+---------+----------------------+-------------+--------------------------------------------------------------+-----+---------+---------------------+--------------+
| id  | city_id | email                | roles       | password                                                     | bio | img_url | creation_date       | name         |
+-----+---------+----------------------+-------------+--------------------------------------------------------------+-----+---------+---------------------+--------------+
| 103 |    NULL | test@dossierprojet.com | ["ROLE_USER"] | $2y$13$BVj5A7bUPc8UIx2OQe/xKexVQZXT6gv.FyYHrApFy3trDgW9uOp1G | NULL | NULL    | 2025-11-22 12:19:13 | DossierProjet |
+-----+---------+----------------------+-------------+--------------------------------------------------------------+-----+---------+---------------------+--------------+
```

5.2 Test automatisé

Nous avons aussi réalisé un test sur ce composant dans Angular . Ce test est réalisé avec Jasmine et exécuté via Karma. Ce test permet de s'assurer que les champs sont vides lors de l'initialisation:

```
it('should have form controls initialized as empty', () => {
  const form = component.subscribeform;
  expect(form.get('email')?.value).toBe('');
  expect(form.get('name')?.value).toBe('');
  expect(form.get('password')?.value).toBe('');
  expect(form.get('confirmPassword')?.value).toBe('');
});
```

Ce test est présent dans le fichier spec.ts et peut être exécuté avec tous les autres tests avec la commande ng test. Karma ouvre ensuite une page de navigateur pour nous donner les résultats:

```
SubscribeComponent
  • should have form controls initialized as empty
```

D'autres tests pourraient être mis en place afin de garantir la stabilité de notre application en vérifiant le bon fonctionnement de toutes nos fonctionnalités lors d'ajout ou de modifications futures.

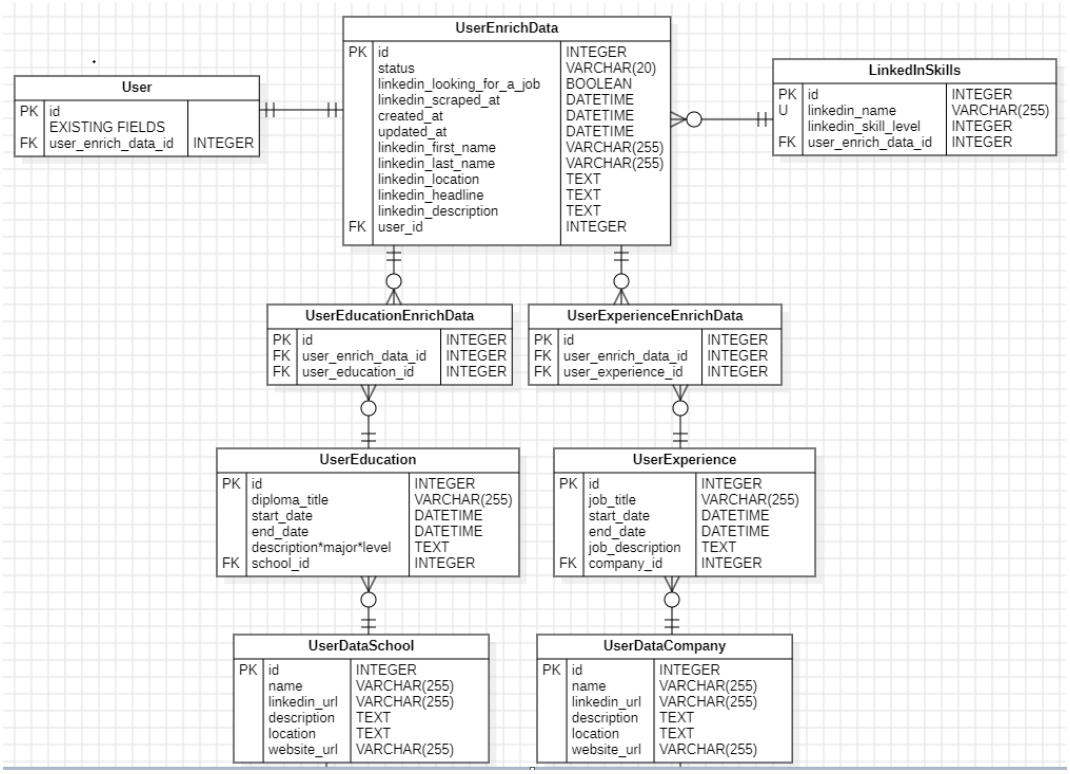
6. Travail effectué en stage

6.1 Présentation et objectif du stage

Axi Technologies est une startup spécialisée en intelligence artificielle, composée d'un service de consulting premium et d'une plateforme de formation sur laquelle j'ai travaillé : https://datascientist.fr/. Cette plateforme propose plusieurs formations autour de l'IA, de la data science et du cloud engineering.

Au sein de l'entreprise on m'a attribué plusieurs tâches mais j'aimerais présenter ici la tâche principale qui m'a été confiée : le data enrichment. Durant mon stage, j'ai travaillé exclusivement sur Linux, sur un environnement Python et DRF entièrement dockerisé, en tant que développeur back-end, en étroite collaboration avec Antonin, designer, Dung, développeur front-end, Haitem, commercial, et encadré de près par mon responsable Romain et le CEO de l'entreprise Hatim.

L'objectif de ce travail est de pouvoir enrichir le profil d'un utilisateur à partir de son nom complet, en créant un endpoint qui pourrait permettre de remplir le MPD suivant:



Il a d’abord fallu déterminer comment récupérer ces données, autrement dit quels services utiliser. Plusieures options existent:

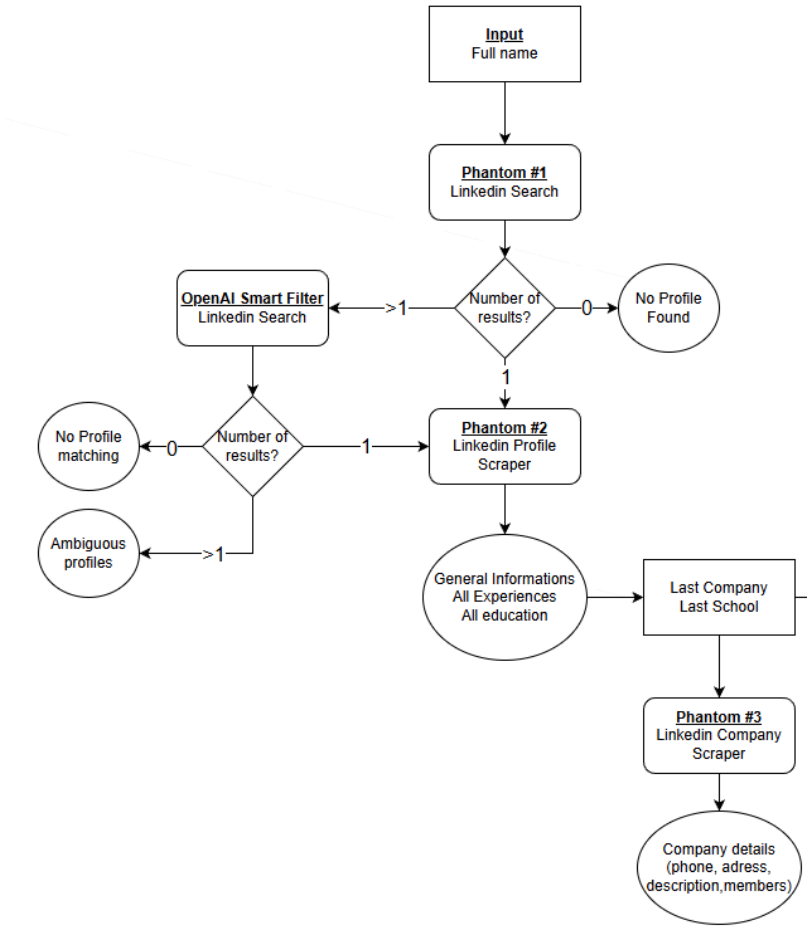
- Les SERP(Search Engine Results Page), qui permettent de recueillir beaucoup de résultats mais ne sont pas assez précis et ne peuvent pas recueillir assez d’informations.

- Les bases de données B2B/B2C, qui sont des bases de données très fournies mais qui ne suffisent pas à enrichir totalement le modèle.

- Le scraping, en l'occurrence le scraping LinkedIn, par le biais de l’api du service PhantomBuster. C’est cette solution que nous avons retenue et que je me suis chargé d'implémenter, tout d’abord sur un projet POC puis que j’ai finalement intégré en endpoint sur la plateforme.

6.2 Réalisation de la tâche

Après de nombreux tests, j’ai finalement retenu le schéma de fonctionnement suivant:



Tout d'abord, nous effectuons une recherche linkedin sur le nom souhaité. En fonction du résultat, plusieurs cas possibles:

- Aucun profil, nous retournons "No profile found".
- Un seul profil, nous pouvons enrichir ce profil.
- Plusieurs profils : dans ce cas, nous utilisons OpenAI avec l'outil Langchain afin d'essayer de filtrer les profils récupérés. Grâce à Langsmith, outil permettant de visualiser efficacement les résultats de nos prompts, j'ai pu grandement améliorer la qualité du filtre en passant des instructions précises (nom complet exact, profil ayant un rapport avec data ou IA). Nous récupérons ensuite son output et si un seul profil est retenu, nous pouvons enrichir le profil. Dans les autres cas nous retournons une erreur.

Une fois le bon profil récupéré, nous scrappons le profil linkedin de l'utilisateur, ainsi que la page entreprise de sa dernière expérience professionnelle.

Enfin, j'ai réalisé les deux endpoints: l'endpoint POST qui permet de lancer la pipeline d'enrichissement et l'endpoint GET, qui permet de récupérer le statut du processus ou les données enrichies si le processus est terminé. Voici la logique de l'endpoint POST:

```python
class UserEnrichViewSet(APIView):
    permission_classes = [IsAuthenticated, IsAdminUser, IsProfessor]

    def post(self, request, user_id):
        user = get_object_or_404(User.all_objects, id=user_id)

        lock_id = f"enrich_user_{user_id}"
        with advisory_lock(lock_id):
            enrich_data, _ = UserEnrichData.objects.get_or_create(user=user)

            if enrich_data.status == "error":
                return Response(
                    {
                        "message": "Error during enrichment",
                        "error_message": enrich_data.error_message,
                    },
                    status=status.HTTP_400_BAD_REQUEST,
                )

            if enrich_data.status == "done":
                return Response(
                    {"message": "User already enriched"},
                    status=status.HTTP_200_OK,
                )

            if enrich_data.status == "pending":
                return Response(
                    {
                        "message": "Enrichment already in progress",
                    },
                    status=status.HTTP_200_OK,
                )

            enrich_data.status = "pending"
            enrich_data.save()
            enrich_user.delay(user.id)

            return Response(
                {
                    "message": "Enrichment pipeline started",
                },
                status=status.HTTP_200_OK,
            )
```

Afin d'éviter plusieurs requêtes simultanées, on utilise pglock, qui permet de ne laisser passer que la première requête effectuée et de ne pas en autoriser de doublons tant que la première n'est pas terminée. De plus, pour que cette tâche soit exécutée de manière asynchrone, nous utilisons le service Celery.

Voici un extrait de la réponse du endpoint GET:

Response body


```
{
  "first_name": "Romain",
  "last_name": "[redacted]",
  "location": "Lyon, Auvergne-Rhône-Alpes, France",
  "headline": "Lead Développeur chez AXI Technologies",
  "description": "",
  "linkedin_img_url": "https://media.licdn.com/dms/image/v2/C4D03AQG9uPJF0KqEaw/profile-displayphoto-shrink_800_800/profile-displayphoto-shrink_800_800/0/1599498930636?e=1763596
800&v=beta&t=hFQSpYp-1FznL16zMP0pehfDyGexvjlsKE9_UYUt_Vs",
  "looking_for_a_job": false,
  "status": "done",
  "error_message": null,
  "linkedin_scraped_at": "2025-11-03T09:11:20.354505Z",
  "experiences": [
    {
      "job_title": "Lead Développeur",
      "date_range": "nov. 2022 - aujourd'hui",
      "job_description": "",
      "company": {
        "name": "AXI Technologies",
        "linkedin_url": "https://www.linkedin.com/company/67711472",
        "image_url": "https://media.licdn.com/dms/image/v2/C4D0BAQF4UrEPEVXIZw/company-logo_400_400/company-logo_400_400/0/1630519221649/axi_technologies_fr_logo?e=1763596800&v=
beta&t=xWre3Jl24-W8__A6D_YtgyamkALgl1CoRy-eLtl3mrg",
        "description": "AXI Technologies is an AI startup building SaaS solutions and products for next generation AI. \n\nAXI Technologies est une data player spécialisée en In
```

7. Conclusion

En conclusion, la réalisation de ce projet a représenté un travail particulièrement formateur et structurant, tant sur le plan technique que méthodologique. J'ai pu mener à bien l'ensemble des étapes nécessaires à la conception d'une application web complète : une API Symfony sécurisée, un front-end Angular cohérent et maintenable, ainsi qu'un déploiement sur VPS pleinement fonctionnel. Le fait d'avoir pu mettre en ligne une version stable, performante et fidèle aux exigences initiales constitue une réelle satisfaction.

Ce projet a été développé en parallèle de mon stage, ce qui a représenté une contrainte supplémentaire en termes de temps et d'organisation. Malgré cela, j'ai pu produire un projet abouti, en respectant les bonnes pratiques de sécurité, d'architecture et d'optimisation. Cette expérience m'a clairement permis de valider ma capacité à mener un projet complet de bout en bout.

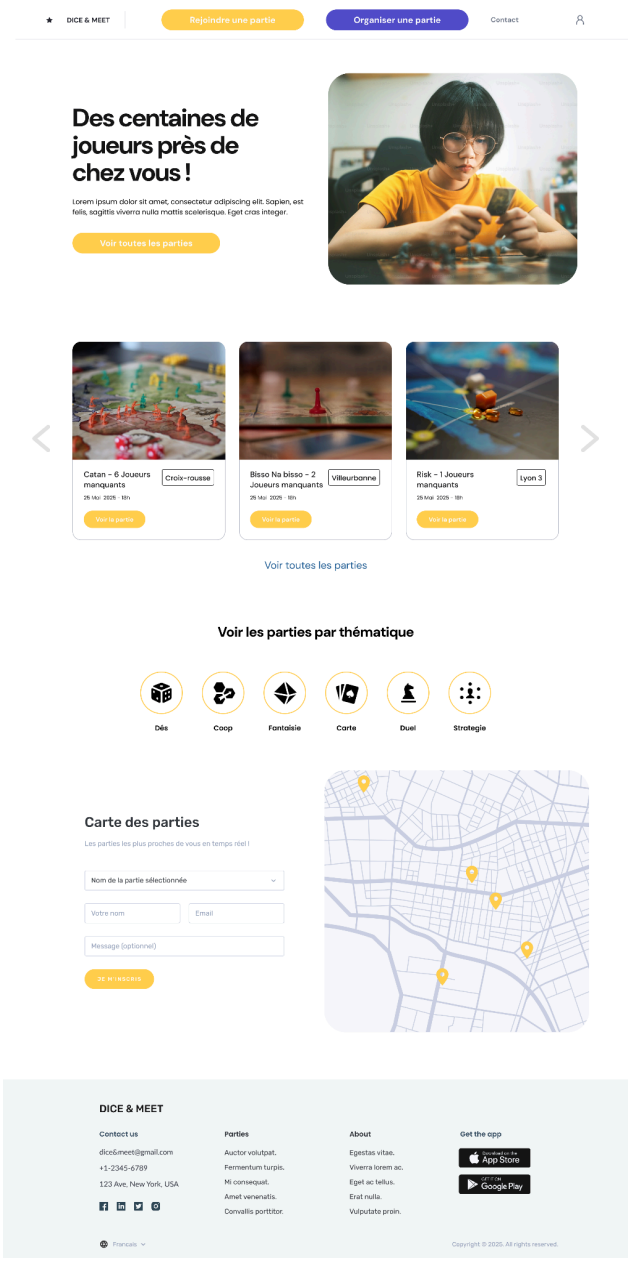
Certaines limites subsistent néanmoins. J'aurais souhaité préparer un jeu de maquettes plus complet et plus proche de l'interface finale ; au fil du développement, plusieurs choix graphiques ont été faits directement dans Angular. De même, le projet aurait bénéficié de davantage de tests automatisés, côté front comme côté back, ainsi que d'une dockerisation complète de l'ensemble, ce qui aurait facilité le déploiement et la maintenance.

Enfin, cette année marque pour moi une véritable reconversion professionnelle réussie. Je me suis pleinement épanoui dans le développement web et plus largement dans l'informatique. J'ai la chance de pouvoir poursuivre en alternance au sein de l'entreprise qui m'a accueilli en stage, et j'envisage désormais de poursuivre vers un master en alternance dans le domaine de la data. Ce projet, par sa difficulté et son envergure, a confirmé ma motivation et ma volonté de continuer à évoluer dans ce domaine qui me passionne.

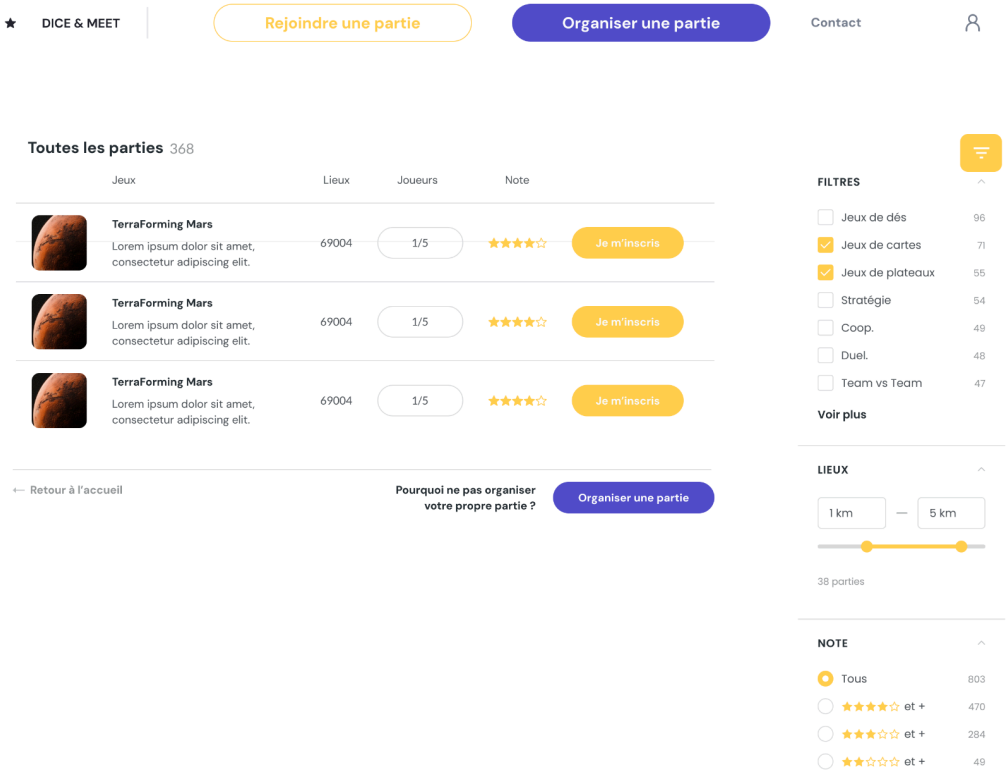
ANNEXES

I - Maquettes et wireframes

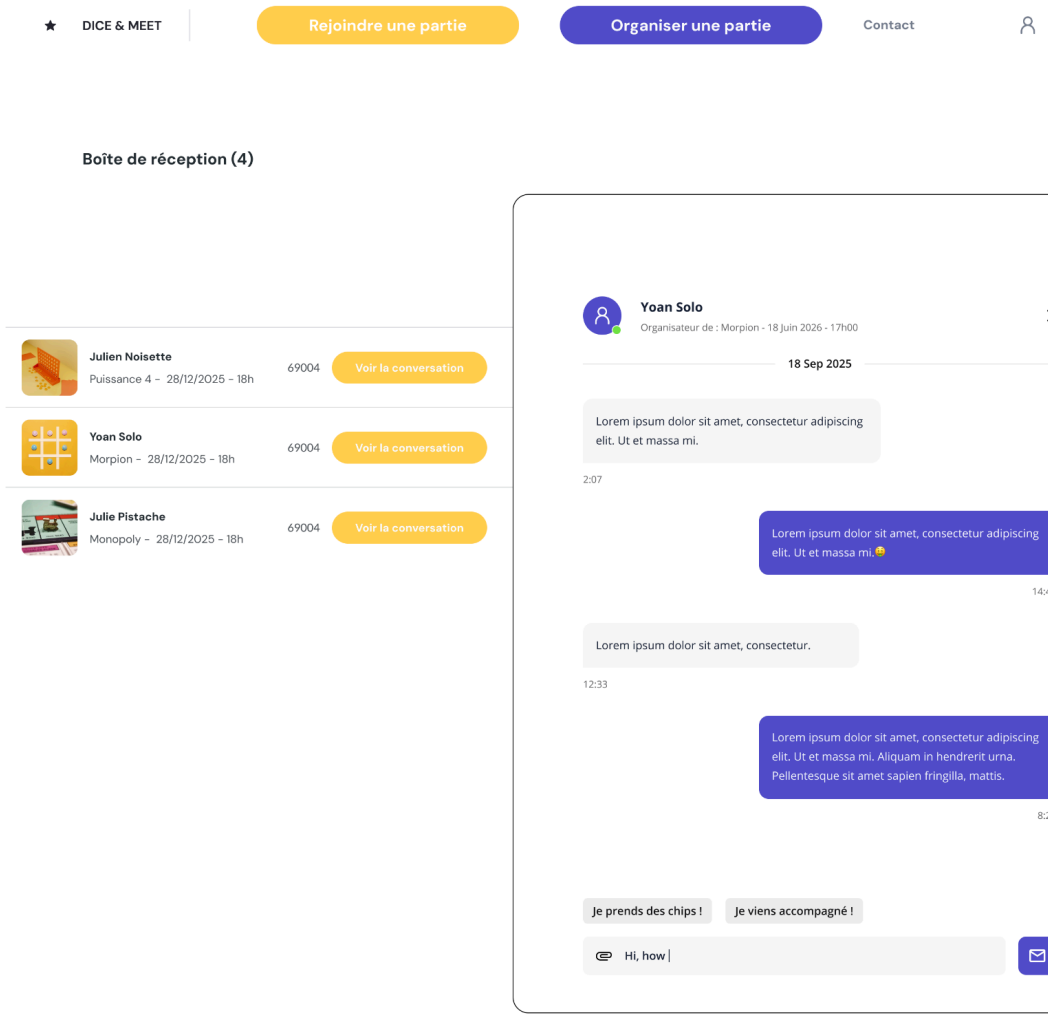
Landing page



Liste événements



Messagerie



II-Interface EasyAdmin