

Dossier de projet professionnel

Réalisation de l'application web et mobile

LDVEH



LDVEH
MAKE YOUR STORY

présenté par Clément Machtelinckx

1. Introduction

1.1 Présentation personnelle

Je me nomme Clément Machtelinckx, j'ai trente-trois ans. Mon parcours dans le développement a débuté grâce à l'école *La Plateforme*, où j'ai choisi de me spécialiser dans le développement web. Ce domaine m'a immédiatement séduit par sa capacité à conjuguer rigueur logique, créativité graphique et impact concret sur le quotidien des utilisateurs.

Au fil de ma formation, j'ai pu explorer différentes facettes du développement de l'interface utilisateur aux bases de données et acquérir des compétences transversales en conception, en sécurité, en performance et en déploiement d'applications. C'est dans ce cadre que s'inscrit mon projet professionnel.

1.2 Présentation du projet

Mon projet, intitulé **LDVEH (Livre Dont Vous Êtes le Héros)**, trouve son origine dans une passion d'enfance pour les livres interactifs. L'idée est née d'un constat : ce type d'expérience narrative, pourtant riche et immersive, reste peu représenté dans le paysage numérique actuel. J'ai donc voulu lui redonner vie, en l'adaptant aux supports modernes que sont le mobile et le web.

L'objectif principal de cette application est de rendre l'univers du livre interactif accessible au plus grand nombre. Elle permet aux utilisateurs de lire des histoires, de faire des choix qui influencent le déroulement du récit, et même de créer leurs propres aventures. Dès les premières étapes de conception, j'ai veillé à placer l'expérience utilisateur (UX) au centre du projet, en misant sur une interface claire, intuitive et engageante.

1.3 Project Presentation (in English)

I had the idea to create **LDVEH (Livre Dont Vous Êtes le Héros)** to fill a gap in the digital market regarding interactive storytelling. This type of content, although well-known in printed books from the 80s and 90s, remains underused in modern digital applications.

The aim of the app is to provide everyone with access to a rich narrative experience, in which users can shape the story through their own decisions. They can also create their own adventures and share them with others. From the beginning of the project, I've focused heavily on user experience (UX) and interface design (UI) to ensure the app is intuitive, immersive, and inclusive.

1.4 Compétences couvertes par le projet

Ce projet m'a permis de mobiliser un large spectre de compétences liées au développement logiciel. J'ai travaillé sur la conception d'interface utilisateur, en réalisant des maquettes fonctionnelles et graphiques. J'ai mis en place une base de données relationnelle adaptée à la logique du récit interactif, en structurant les entités telles que les livres, les pages, les choix et les aventuriers.

Le développement côté front-end, que ce soit pour le web ou le mobile, m'a permis de construire des interfaces réactives et adaptées à différents terminaux. Côté back-end, j'ai développé une API REST sécurisée, intégrée à un système d'authentification JWT, et j'ai mis en œuvre une logique métier complète autour de la navigation, du combat, et de la progression dans l'histoire.

J'ai également abordé la mise en place de tests unitaires pour valider la stabilité du code, et conçu un environnement de développement commun à l'aide de Docker. Enfin, j'ai préparé les bases du déploiement afin de rendre le projet exploitable dans un contexte réel.

Ce projet constitue ainsi une synthèse concrète des compétences attendues dans le cadre du titre professionnel visé, tout en étant le reflet d'un engagement personnel fort dans un sujet qui me tient à cœur.

2. Organisation et cahier des charges

2.1 Les utilisateurs du projet

Le projet LDVEH se divise en deux interfaces distinctes, chacune destinée à un public spécifique. L'application mobile est conçue pour les utilisateurs finaux, c'est-à-dire les lecteurs et joueurs. Ces derniers doivent impérativement créer un compte pour accéder à la plateforme. Cette exigence permet de lier chaque progression à un profil unique, garantissant ainsi la sauvegarde de l'état d'avancement et une continuité dans l'expérience utilisateur.

En parallèle, une interface web d'administration est mise à disposition des administrateurs. Ce back-office permet de gérer l'ensemble du contenu de l'application ainsi que les utilisateurs, via un tableau de bord ergonomique et sécurisé. Bien que séparées fonctionnellement, ces deux interfaces interagissent avec la même base de données et la même API, pour assurer la cohérence du système.

2.2 Les fonctionnalités attendues

Application mobile

Page d'accueil :

Lors du lancement de l'application, l'utilisateur accède à une page d'accueil simple et directe. Il peut choisir de se connecter s'il possède déjà un compte, ou bien s'inscrire pour débuter une nouvelle aventure. Cette interface marque le point d'entrée dans l'univers interactif de LDVEH.

Page d'inscription :

Cette page permet à l'utilisateur de créer un compte en renseignant ses informations

personnelles. Une fois le formulaire validé, le compte est enregistré en base de données, et l'utilisateur est automatiquement redirigé vers la page de connexion.

Page de connexion :

Ici, les utilisateurs déjà inscrits peuvent se connecter à leur compte. L'authentification est gérée via un système sécurisé à base de JWT. En cas de succès, l'utilisateur accède immédiatement à l'espace principal de lecture.

Page Livre :

Une fois connecté, l'utilisateur arrive sur une page listant tous les livres disponibles dans la bibliothèque de l'application. Chaque livre est présenté avec son titre, sa couverture et un résumé. En sélectionnant un livre, l'utilisateur lance une nouvelle aventure.

Page Aventure :

Cette page invite l'utilisateur à nommer son aventurier. Ce personnage est alors créé en base et associé au livre sélectionné. Cela permet de suivre précisément la progression du joueur dans cette aventure.

Page "Page" :

Sur cette page, l'utilisateur entre dans la lecture interactive. Il lit un extrait du récit et choisit parmi plusieurs actions qui influencent la suite de l'histoire. La progression est automatiquement sauvegardée à chaque interaction. Cette mécanique repose sur la logique métier du backend pour garantir la cohérence de l'aventure.

Page Profil :

Accessible à tout moment, cette page permet à l'utilisateur de modifier ses informations personnelles (nom, mot de passe, email). Il peut également consulter la liste de ses aventures en cours ou terminées, et reprendre une partie en un clic, grâce à la sauvegarde automatique intégrée.

Application web (interface d'administration)

Page de connexion :

Cette page permet aux administrateurs de se connecter à l'espace d'administration via un formulaire sécurisé. L'accès est limité aux comptes disposant d'un rôle *ROLE_ADMIN*, avec une authentification gérée par Symfony.

Page Dashboard :

Le tableau de bord est le centre de gestion de l'application. Il donne accès à toutes les entités essentielles via une interface claire et structurée, générée grâce au bundle EasyAdmin.

Book :

L'administrateur peut ajouter, modifier ou supprimer les livres disponibles dans l'application. Chaque livre est une entité indépendante, contenant plusieurs pages liées entre elles.

Page :

Chaque page fait partie d'un livre et contient du texte narratif, des choix, et parfois un monstre. L'interface permet de créer de nouvelles ou d'édition celles existantes.

Choice :

Les choix proposés à la fin des pages sont gérés ici. Ils relient les pages entre elles et orientent la navigation de l'utilisateur dans l'histoire.

User :

Permet de consulter les utilisateurs enregistrés dans l'application, de surveiller leur activité ou de résoudre d'éventuels problèmes de compte.

Adventurer :

Cette section affiche tous les personnages créés par les utilisateurs. Chaque aventurier est associé à un livre, une aventure en cours, et une position actuelle dans le récit.

Monster :

Les monstres peuvent être liés à certaines pages et nécessitent un combat avant de pouvoir progresser. Ils sont définis avec un nom, une habileté et une endurance, et sont gérables via cette interface.

Contexte technique :

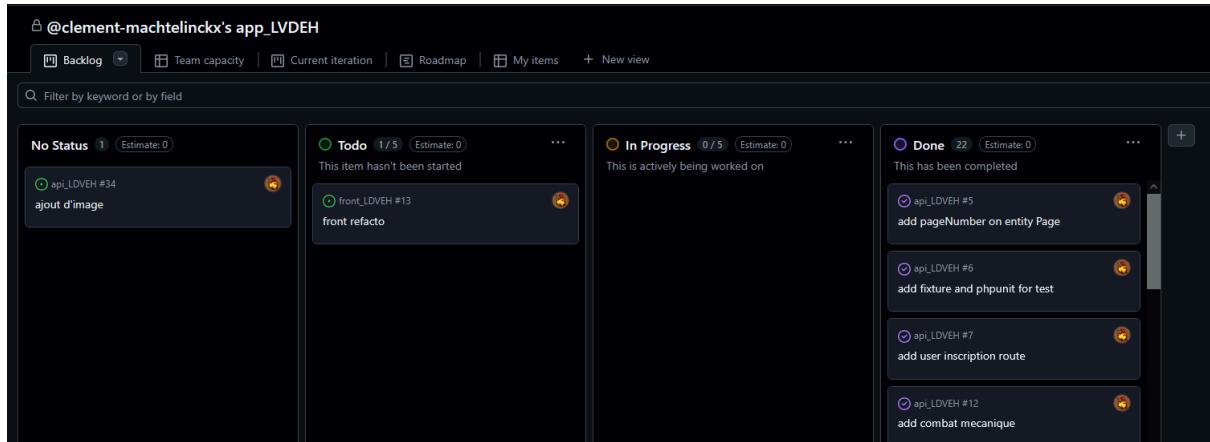
L'application LDVEH a été pensée pour fonctionner sur l'ensemble des systèmes d'exploitation mobiles Android et iOS, en utilisant React Native pour garantir une compatibilité multiplateforme à partir d'un seul code source. Du côté de l'administration, l'interface web a été développée de manière à être pleinement accessible depuis tous les navigateurs modernes (Chrome, Firefox, Edge, Safari), sans restriction liée à l'environnement utilisateur. Cette approche garantit un accès large et une expérience homogène, que ce soit pour les lecteurs comme pour les administrateurs.

2.3 Méthodologie de travail

Bien que j'aie réalisé ce projet en autonomie, j'ai tenu à conserver une organisation inspirée des méthodes de travail en équipe. Mon objectif était de ne pas perdre les bonnes habitudes acquises en formation et en environnement professionnel.

Pour cela, j'ai utilisé **GitHub Projects** afin de suivre l'avancement du projet sous forme de tickets. Chaque tâche était décrite de façon claire, priorisée, et déplacée au fil de sa progression dans des colonnes “À faire”, “En cours”, “Terminée”. Ce suivi visuel m'a permis de garder une vision d'ensemble sur le développement, de limiter les oubliers, et de maintenir une discipline de travail constante.

Cette approche m'a également permis d'anticiper les phases complexes (comme la logique de navigation ou l'intégration des livres), et de documenter mes choix au fur et à mesure.



3. Conception du projet

3.1 choix de développement

choix des langages :

Pour réaliser ce projet, j'ai décidé d'utiliser **PHP** pour le backend et **TypeScript** pour le frontend. J'ai également fait appel à **Python** pour certaines tâches spécifiques lors du développement. Ce choix s'appuie sur les forces de chaque langage et leur complémentarité dans le cadre d'un projet web et mobile. Mon objectif était de m'appuyer sur des technologies maîtrisées, polyvalentes et compatibles avec les outils modernes de développement, tout en gardant une architecture claire et évolutive.



PHP :

PHP est un langage serveur robuste, largement utilisé dans l'industrie pour la création de sites web dynamiques. Sa grande maturité, sa communauté active, ainsi que la richesse de son écosystème en font un choix fiable pour concevoir un backend structuré. De plus, il dispose de frameworks puissants.



TypeScript :

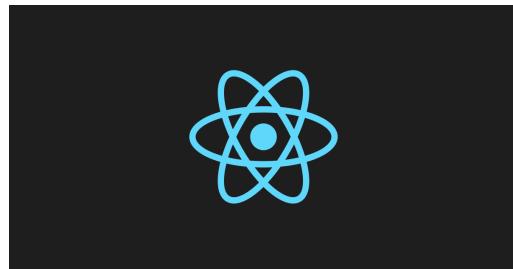
TypeScript est un sur-ensemble typé de JavaScript qui apporte plus de rigueur, de sécurité et de lisibilité au code. Dans ce projet, j'ai utilisé TypeScript avec React Native via Expo pour développer toute la partie mobile. Ce choix m'a permis de bénéficier de l'autocomplétion, de la détection d'erreurs à la compilation et d'une meilleure documentation du code. Grâce à cette base solide, j'ai pu créer une application multiplateforme (iOS/Android) performante, tout en gardant une structure claire, évolutive et adaptée aux bonnes pratiques de développement mobile.

3.2 choix des frameworks :



Symfony :

Pour le développement de mon API, j'ai choisi Symfony, un framework PHP structuré, reconnu pour sa robustesse et sa modularité. D'origine française, il bénéficie d'une documentation riche, d'une large communauté active et d'un écosystème de bundles puissants qui facilitent l'implémentation rapide de fonctionnalités avancées. Il s'est imposé comme un choix naturel pour construire une API REST maintenable, sécurisée et bien organisée.

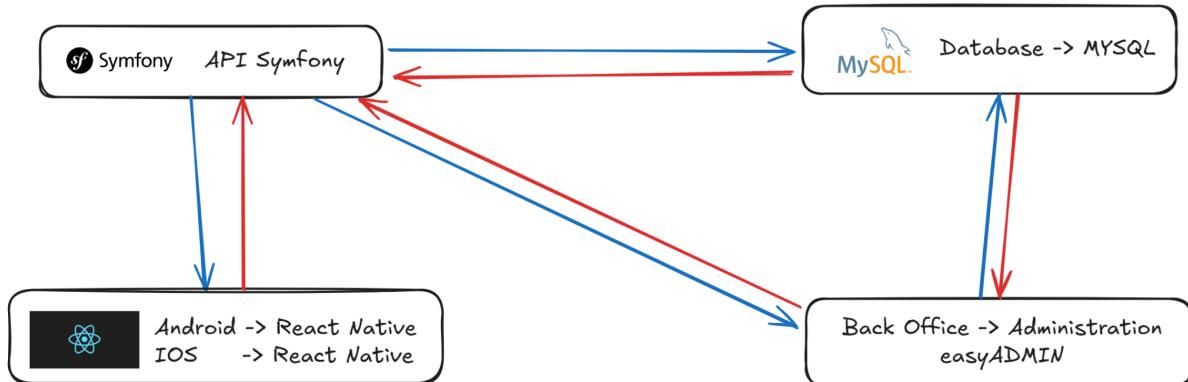


React Native :

Pour la création de mon application mobile, j'ai opté pour React Native, un framework développé par Meta, qui permet de coder une seule fois en TypeScript pour cibler à la fois Android et iOS. Ce choix m'a permis de mutualiser le développement, d'accélérer la mise en œuvre de l'application, tout en profitant de l'écosystème React et de sa logique de composants réutilisables. React Native offre aussi une excellente intégration avec les outils de débogage, le hot reload, et une bonne performance globale sur mobile.



Technologie Stack



logiciel et autre outils :

- **Visual Studio Code** : éditeur de code principal utilisé pour le développement front-end et back-end, apprécié pour ses nombreuses extensions et sa légèreté.
- **Postman** : outil utilisé pour tester les requêtes HTTP de mon API tout au long du développement.
- **GitHub** : plateforme de versionning utilisée pour sauvegarder et suivre l'évolution du projet.
- **GitHub Projects** : outil de gestion de tâches intégré à GitHub pour organiser les étapes de développement.
- **Composer** : gestionnaire de dépendances PHP, utilisé notamment pour installer les packages Symfony.
- **Expo Go** : application mobile permettant d'émuler rapidement l'app React Native sur un téléphone pendant le développement.
- **Canva** : outil de design en ligne utilisé pour créer la charte graphique et le logo du projet.
- **Figma** : outil de design en ligne utilisé pour créer la maquette de l'application.
- **Docker** : technologie de conteneurisation mise en place pour standardiser l'environnement de développement et faciliter le déploiement si besoin.
- **WinSCP** : client SFTP utilisé pour transférer facilement des fichiers entre mon poste local et un serveur distant.
- **PuTTY** : terminal SSH utilisé pour accéder à distance à un serveur Linux et administrer le back-end en ligne de commande.
- **[draw.io](#)** : outil de modélisation visuelle utilisé pour créer des diagrammes techniques comme les modèles MCD et MPD.
- **[dbdiagramme.io](#)** : outil en ligne permettant de générer des schémas de base de données à partir d'un langage de description simplifié, pratique pour la conception du MLD.

3.3 Arborescence du projet :

frontend :

Directory structure:

```
└── clement-machtelinckx-front_Idveh/
    ├── app/
    │   └── page/
    ├── assets/
    │   └── fonts/
    │       └── images/
    ├── components/
    ├── constants/
    ├── hooks/
    ├── screens/
    ├── services/
    │   ├── api.ts
    │   └── auth.ts
    └── store/
```

backend :

Directory structure:

```
└── clement-machtelinckx-api_Idveh/
    ├── bin/
    ├── config/
    ├── migrations/
    ├── public/
    └── src/
        ├── ApiResource/
        ├── Command/
        ├── Controller/
        ├── DataFixtures/
        ├── Entity/
        ├── EventListener/
        ├── Factory/
        ├── Repository/
        ├── Security/
        └── Service/
        ├── templates/
        ├── tests/
        └── translations/
```

4. Conception du front end

Afin de concevoir une interface utilisateur à la fois intuitive, accessible et sécurisée pour mon application web et mobile *LDVEH - Livre Dont Vous Êtes le Héros*, j'ai commencé par poser les bases graphiques et ergonomiques du projet. Cette étape a consisté à élaborer un **moodboard**, une **charte graphique** et une **maquette fonctionnelle**.

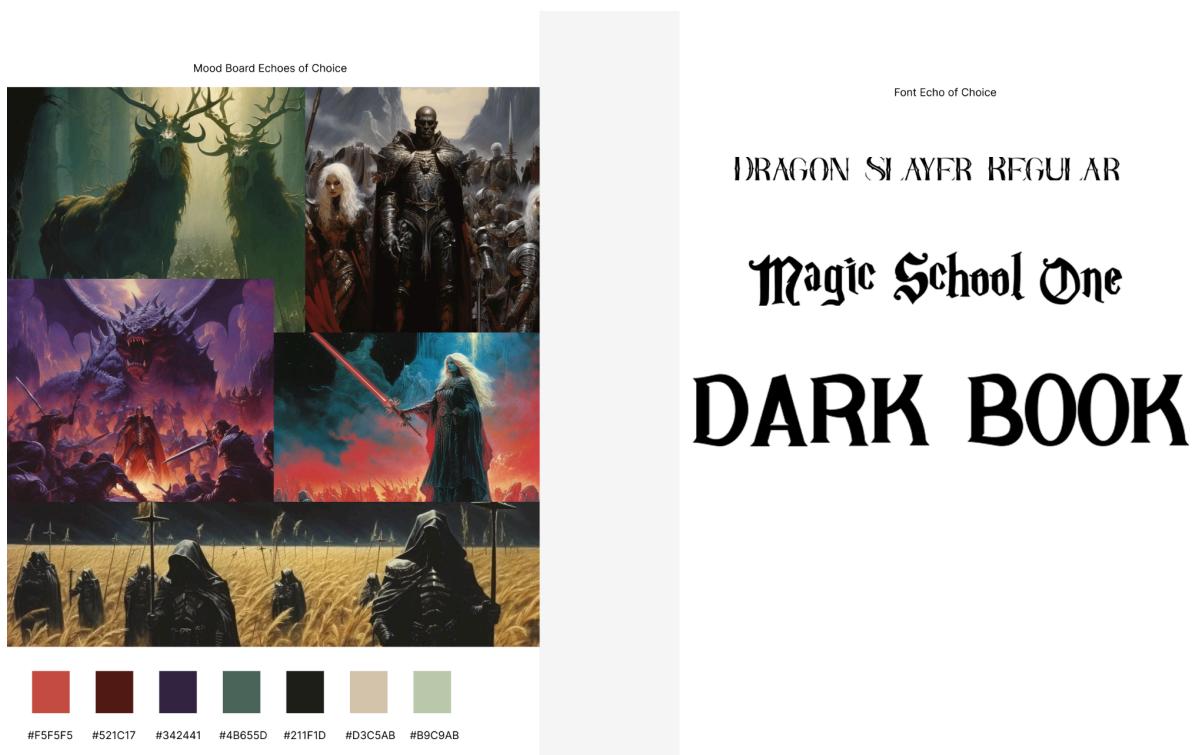
Ces éléments visuels m'ont permis de définir l'univers esthétique de l'application, en cohérence avec son thème dark fantasy, tout en respectant les standards modernes d'ergonomie mobile (approche *mobile-first*) et les recommandations d'accessibilité.

L'objectif était de proposer une expérience utilisateur fluide et immersive, sur tous types d'écrans, tout en facilitant la navigation et l'interaction avec les différents contenus narratifs.

4.1 moodboard

L'application *LDVEH* puise son inspiration dans l'esthétique des années 80, une époque marquée par l'essor du dark fantasy et des livres dont vous êtes le héros. J'ai donc voulu rendre hommage à cet univers visuel en concevant un moodboard qui reflète ces codes : teintes sombres, textures anciennes, typographies gothiques, et références iconographiques aux créatures fantastiques et aux univers médiévaux.

Ce travail visuel a servi de socle à toute la direction artistique du projet, en posant une ambiance cohérente et immersive, pensée pour renforcer l'expérience narrative proposée à l'utilisateur.



4.2 Maquette

J'ai ensuite conçu une maquette complète de l'application à l'aide de l'outil *Figma*, en adoptant une approche *mobile first*. Ce choix répond à une réalité actuelle : la majorité des utilisateurs accèdent désormais aux services web via leur smartphone. Il était donc essentiel pour moi de penser l'ergonomie et le design de mon interface en priorité pour les petits écrans, tout en garantissant une adaptation fluide sur desktop.

La maquette a servi de base concrète à la phase de développement front-end, en guidant la disposition des éléments, la navigation entre les écrans, et le respect de la charte graphique définie dans le moodboard.

[voir annexe page :]

5. Conception du Backend

Après avoir conçu et mis en place l'interface utilisateur, j'ai structuré la partie back-end de mon application autour d'une API REST sécurisée, développée avec Symfony.

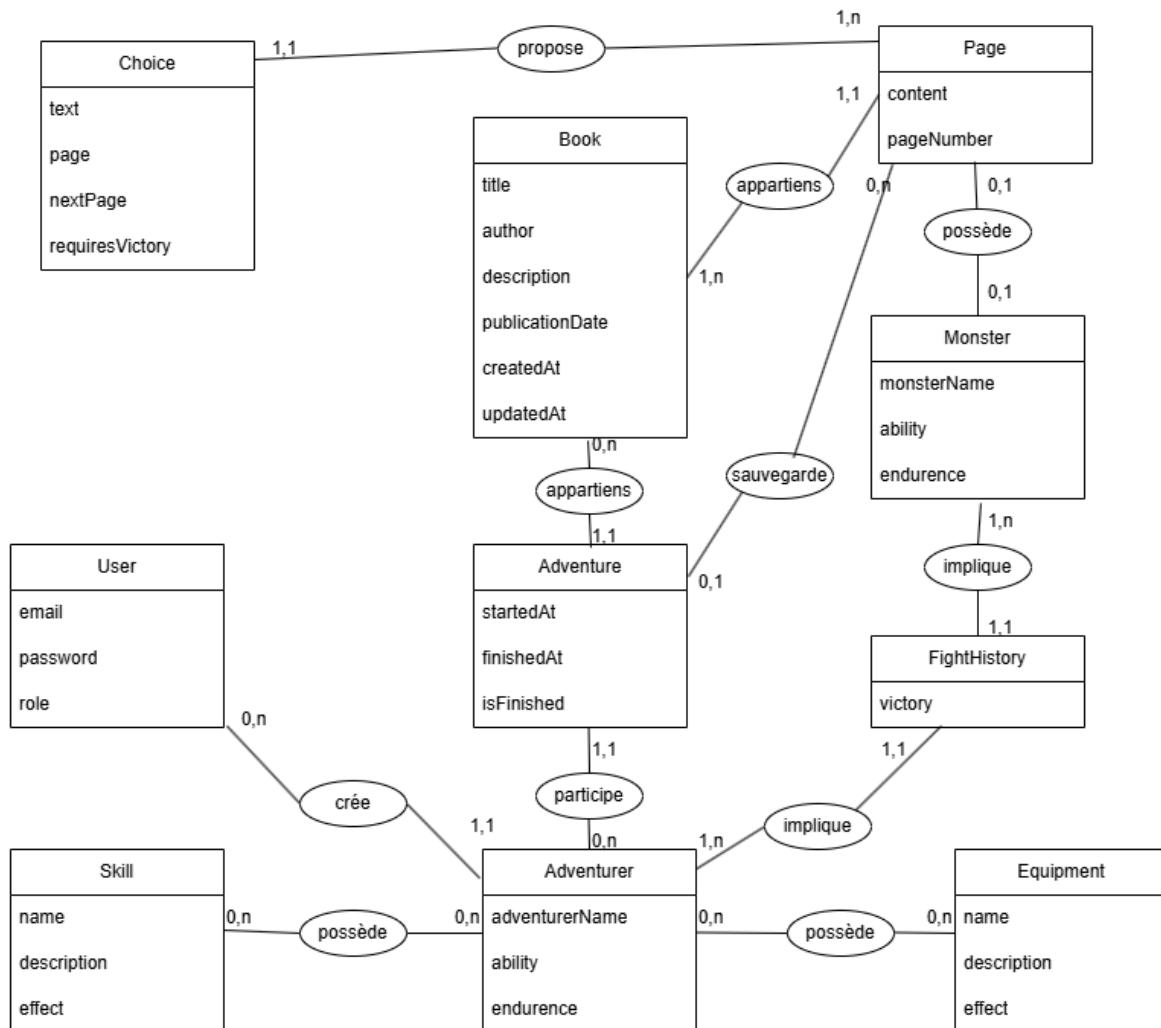
Cette partie gère la logique métier, les accès aux données et les règles de sécurité.

Pour concevoir ma base de données, j'ai suivi la méthode **Merise**.

5.1 Modèle Conceptuel des Données (MCD)

Pour débuter la conception de ma base de données, j'ai réalisé un MCD (Modèle Conceptuel des Données) selon la méthode Merise. Ce schéma m'a permis d'identifier clairement les entités principales de l'application telles que *User*, *Book*, *Page*, *Adventurer* ou encore *Monster*, ainsi que les relations entre elles. Cette étape est cruciale pour garantir une structuration logique et compréhensible du système d'information, indépendamment de toute contrainte technique.

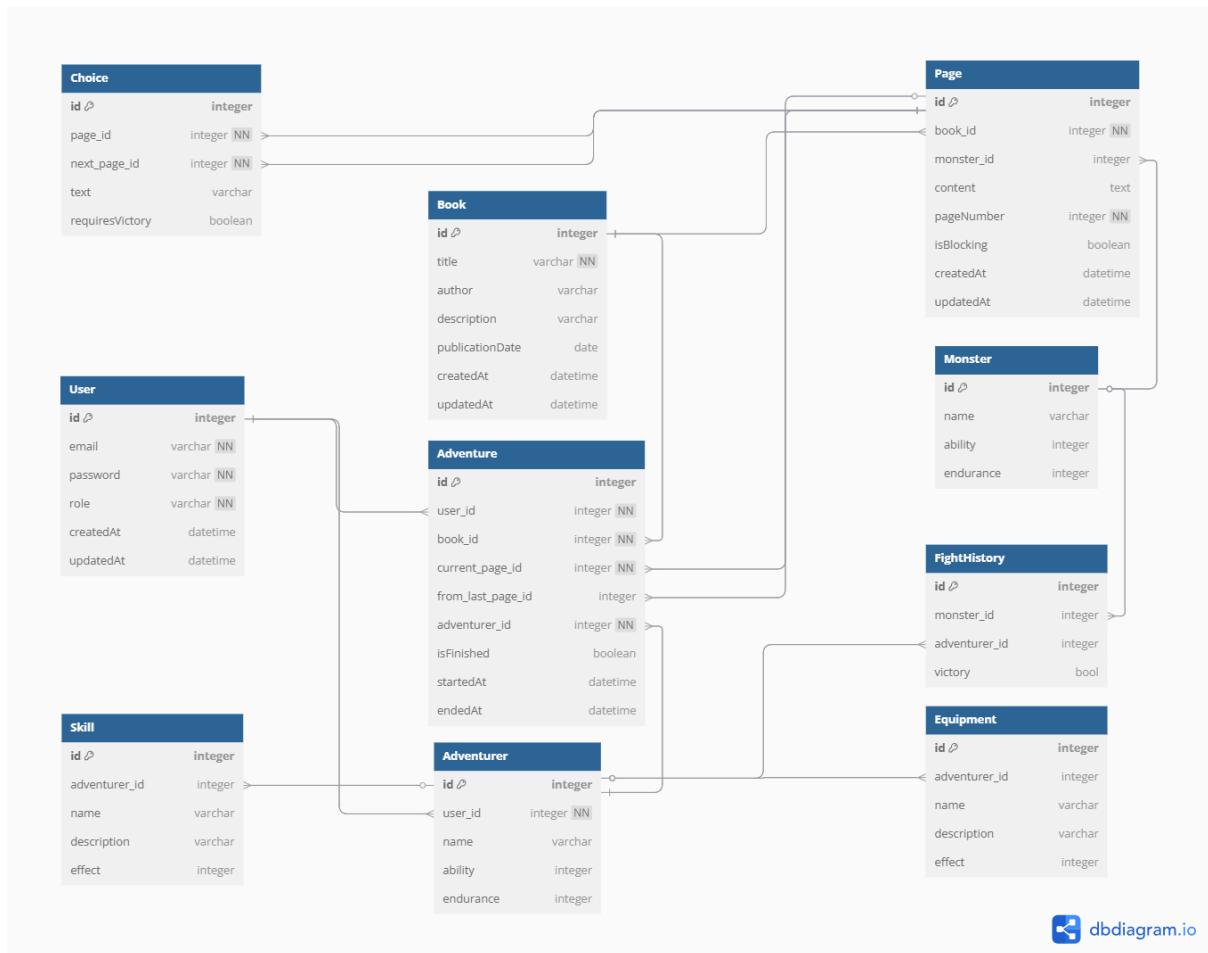
[MCD - draw.io]



5.2 Modèle Logique de Données (MLD)

À partir du MCD, j'ai conçu le MLD (Modèle Logique des Données) en traduisant les relations identifiées en associations techniques, notamment à travers l'ajout de clés étrangères dans les entités concernées. Cette version du modèle reste indépendante du système de gestion de base de données mais reflète déjà la future organisation de mes tables, en tenant compte des cardinalités et des dépendances entre entités.

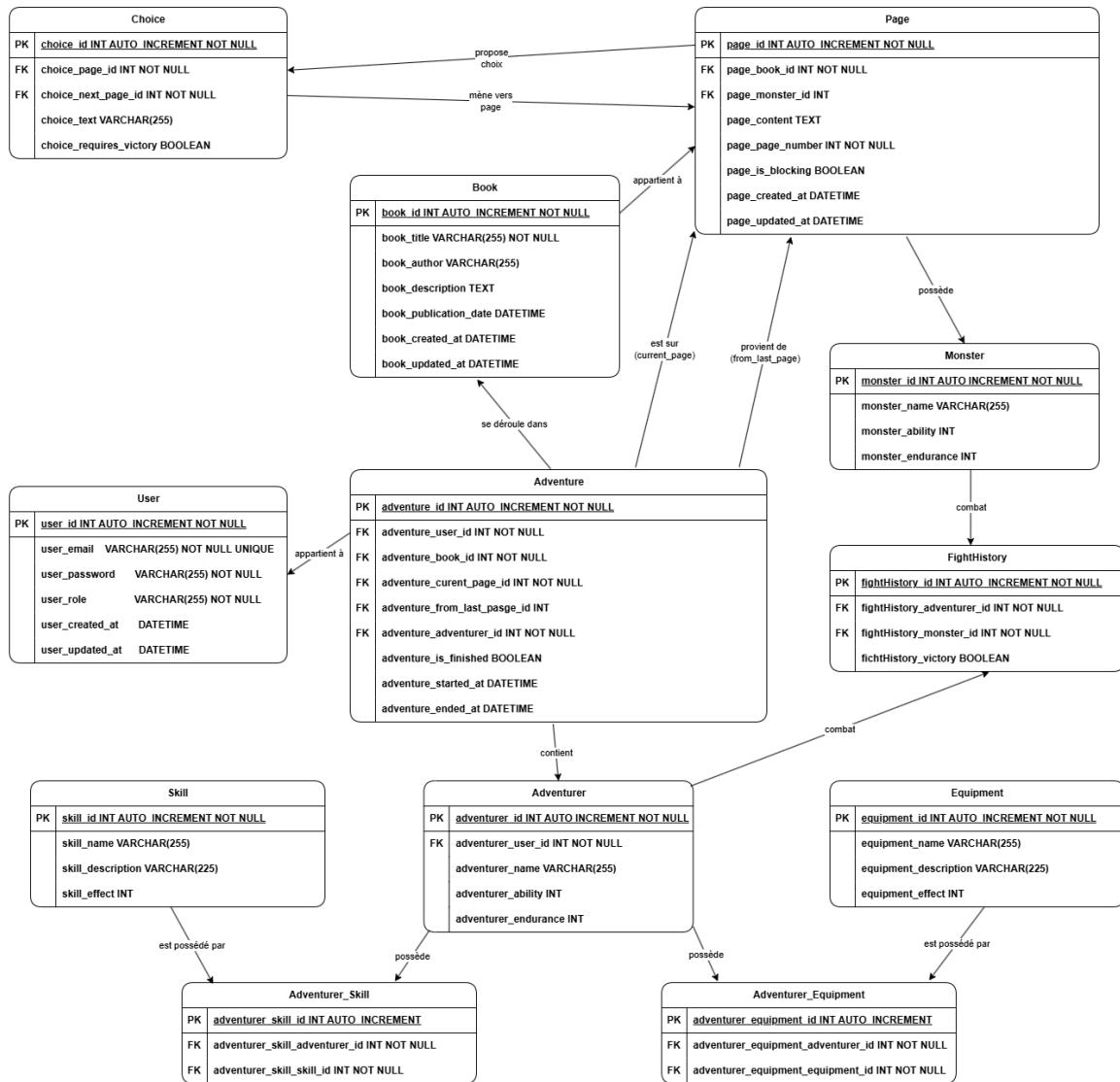
[MLD - dbdiagram.io]



5.3 Modèle Physique de Données (MPD)

Enfin, j'ai finalisé la conception avec le MPD (Modèle Physique des Données) en traduisant le modèle logique en un schéma SQL exploitable. J'ai précisé les types de champs, les contraintes d'intégrité, les index, et les liaisons entre tables. Ce fichier m'a servi de base pour l'implémentation de la base de données réelle dans Symfony via Doctrine, tout en conservant la logique définie en amont.

[MPD - draw.io]



6. Développement du Backend

Une fois la conception de ma base de données finalisée à travers les différents modèles (MCD, MLD, MPD), j'ai entamé le développement de la partie serveur de mon application. Pour cela, j'ai choisi le framework *Symfony*, reconnu pour sa robustesse, sa modularité, et son respect des bonnes pratiques d'architecture logicielle. Grâce à sa structure claire en bundles et à son intégration avec *Doctrine ORM*, Symfony m'a permis de bâtir une API REST structurée, sécurisée et maintenable, parfaitement adaptée aux besoins de mon application interactive. J'ai également pu séparer les différentes responsabilités entre contrôleurs, services métier, gestion des entités et logique de sécurité, en respectant une architecture en couches propre au modèle MVC.

6.1 Développement de la partie administration

J'ai commencé le développement de la partie backend par la création des entités principales nécessaires au bon fonctionnement de l'application : *User*, *Book*, *Page*, *Adventurer*, entre autres. Chacune de ces entités a été définie et reliée avec Doctrine ORM, afin d'assurer une structure de données cohérente et facilement exploitable par l'API.

Dans un second temps, j'ai mis en place un système d'authentification spécifique pour les administrateurs via le composant de sécurité natif de Symfony. Cela a été géré à l'aide d'un contrôleur dédié, *SecurityController*, permettant de gérer la connexion au back-office via une session utilisateur classique.

[SecurityController]

```
12 references | 0 implementations
class SecurityController extends AbstractController
{
    #[Route(path: '/login', name: 'app_login')]
    15 references | 0 overrides
    public function login(AuthenticationUtils $authenticationUtils, Security $security, JwtTokenManagerInterface $jwtManager, SessionInterface $session): Response
    {
        // Vérifie si l'utilisateur est déjà connecté en tant qu'admin
        if ($security->isGranted('ROLE_ADMIN')) {
            $user = $security->getUser();
            $token = $jwtManager->create(user: $user);

            // Stocker le token dans une session
            $session->set(name: 'jwt_token', value: $token);
        }

        return $this->redirectToRoute(route: 'api_doc');
    }

    $error = $authenticationUtils->getLastAuthenticationError();
    $lastUsername = $authenticationUtils->getLastUsername();

    return $this->render(view: 'security/login.html.twig', parameters: [
        'last_username' => $lastUsername,
        'error' => $error,
    ]);
}

#[Route(path: '/api/login', name: 'api_login', methods: ['POST'])]
15 references | 0 overrides
public function loginCheck(
    Request $request,
    JwtTokenManagerInterface $jwtManager
): JsonResponse {
    // Vérifier si l'utilisateur a bien soumis des identifiants
    $user = $this->getUser();
    if (!$user instanceof UserInterface) {
        return new JsonResponse(data: ['error' => 'Invalid login credentials'], status: JsonResponse::HTTP_UNAUTHORIZED);
    }

    // Générer le token JWT
    $token = $jwtManager->create(user: $user);

    return new JsonResponse(data: [
        'token' => $token,
    ]);
}
```

Une des complexités techniques rencontrées à ce stade était la gestion de deux logiques d'authentification distinctes. D'un côté, l'interface d'administration utilise un système de session et de cookies adapté à un usage navigateur. De l'autre, l'application mobile interagit avec le backend via des requêtes API sécurisées, en mode *stateless*, avec des **tokens JWT**.

Pour répondre à cette double exigence, j'ai dû configurer précisément le fichier *security.yaml*. Cette configuration comprend la déclaration de plusieurs **firewalls**, la définition des providers, l'attribution des rôles (comme *ROLE_ADMIN* ou *ROLE_USER*), et l'activation de la sécurité **stateless** pour les routes API.

[security.yaml]

```
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false

    api:
        pattern: ^/(api|fight|page)
        stateless: true
        provider: app_user_provider
        json_login:
            check_path: /api/login
            success_handler: lexik_jwt_authentication.handler.authentication_success
            failure_handler: lexik_jwt_authentication.handler.authentication_failure
            username_path: email
            password_path: password
        jwt: ~

    main:
        lazy: true
        provider: app_user_provider
        stateless: false
        form_login:
            login_path: app_login
            check_path: app_login
            success_handler: lexik_jwt_authentication.handler.authentication_success
            failure_handler: lexik_jwt_authentication.handler.authentication_failure
            default_target_path: /admin
        logout:
            path: app_logout
            target: /login
        security: true

access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/api/login, roles: PUBLIC_ACCESS }
    - { path: ^/api/register, roles: PUBLIC_ACCESS }
    - { path: ^/api, roles: IS_AUTHENTICATED_FULLY }
    - { path: ^/api/my-adventurers, roles: IS_AUTHENTICATED_FULLY }
    - { path: ^/fight, roles: IS_AUTHENTICATED_FULLY }
    - { path: ^/page, roles: IS_AUTHENTICATED_FULLY }
    - { path: ^/login, roles: PUBLIC_ACCESS }
    - { path: ^/, roles: ROLE_USER }
```

Cette architecture permet à la fois de sécuriser efficacement l'accès à l'administration, et de garantir une communication API fiable pour les clients mobiles.

L'un de mes objectifs principaux pour la partie administration était de rendre l'interface totalement autonome. L'idée était que même une personne sans connaissances techniques puisse gérer l'application au quotidien.

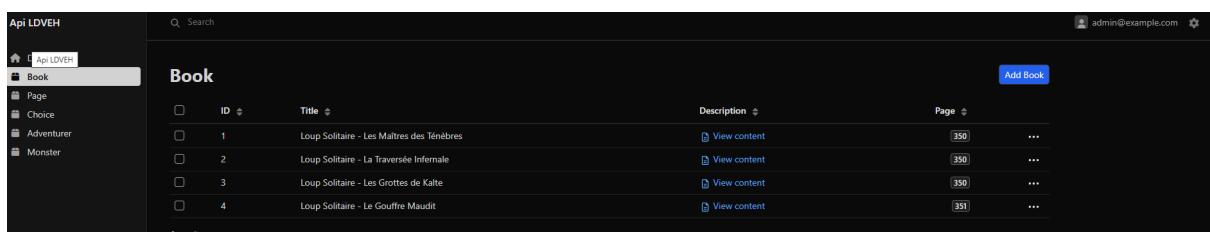
Cela inclut des actions comme la création de nouveaux livres, l'ajout de pages ou de choix narratifs, le suivi des utilisateurs et de leurs aventures en cours, ou encore la correction de données en cas d'erreur. Ce besoin d'accessibilité m'a naturellement conduit vers

EasyAdmin, un bundle Symfony reconnu pour sa simplicité de mise en œuvre et sa puissance.

EasyAdmin permet de générer une interface de gestion *CRUD* moderne, sécurisée et totalement personnalisable. Il s'intègre directement avec les entités Symfony, ce qui m'a permis de créer rapidement un tableau de bord d'administration complet.

Grâce à cette interface, l'administrateur peut visualiser, filtrer, modifier ou supprimer des éléments via une interface graphique claire. Les formulaires sont générés automatiquement, tout en permettant l'ajout de règles de validation ou d'actions personnalisées si nécessaire.

[EasyAdmin dashboard]



The screenshot shows the EasyAdmin Book dashboard. On the left, there's a sidebar with a tree view of entities: Page, Choice, Adventurer, and Monster. The 'Book' entity is selected. The main area has a header 'Book' with a search bar and an 'Add Book' button. Below is a table with four rows of book data:

ID	Title	Description	Page
1	Loup Solitaire - Les Maîtres des Ténèbres	View content	350
2	Loup Solitaire - La Traversée Infernale	View content	350
3	Loup Solitaire - Les Grottes de Kalte	View content	350
4	Loup Solitaire - Le Gouffre Maudit	View content	351

Cette solution m'a permis de gagner un temps précieux sur le développement d'outils d'administration tout en garantissant une interface intuitive pour une utilisation réelle en production.

6.2 Développement de l'API publique

Une fois l'administration fonctionnelle, j'ai concentré mes efforts sur la construction de l'API REST, point névralgique entre le backend Symfony et l'application mobile développée en React Native. Cette API a été pensée pour offrir aux utilisateurs une expérience fluide, interactive et sécurisée, en permettant des opérations clés telles que l'inscription, la connexion, le démarrage d'une aventure, la navigation dans les choix narratifs, la sauvegarde de la progression et la gestion de leur profil.

Afin d'assurer la sécurité des échanges entre le client mobile et le serveur, j'ai intégré le bundle *LexikJWTAuthenticationBundle*, qui repose sur le mécanisme des JSON Web Tokens (JWT). Lorsqu'un utilisateur s'authentifie avec succès, un token chiffré lui est délivré. Ce token est ensuite inclus dans chaque appel HTTP, via l'en-tête **Authorization: Bearer**, ce qui permet de valider l'identité et les droits de l'utilisateur à chaque requête.

La configuration de ce système repose sur plusieurs fichiers essentiels du projet. Le fichier *lexik_jwt_authentication.yaml* définit notamment l'emplacement des clés de chiffrement et la passphrase. Le fichier *security.yaml*, quant à lui, précise les règles d'accès aux différentes

routes, distingue les firewalls pour l'admin et l'API, et gère les providers d'utilisateurs. Du côté du frontend, le fichier `auth.ts` se charge de stocker et d'injecter automatiquement le token dans tous les appels API, assurant une communication transparente et sécurisée.

Ce système stateless s'avère particulièrement robuste, notamment pour les applications mobiles où la gestion de session classique (par cookies) n'est pas adaptée. Il permet une gestion précise des droits d'accès tout en assurant un bon niveau de performance.

[screen lexik_jwt_authentication.yaml / auth.ts ligne 7] (a modif)

```
import AsyncStorage from '@react-native-async-storage/async-storage';

const TOKEN_KEY = 'access_token';

export async function saveToken(token: string) {
|   await AsyncStorage.setItem(TOKEN_KEY, token);
}

export async function getToken(): Promise<string | null> {
|   return await AsyncStorage.getItem(TOKEN_KEY);
}

export async function clearToken() {
|   await AsyncStorage.removeItem(TOKEN_KEY);
}
```

Pour faciliter et accélérer la mise en place de mon API, j'ai opté pour l'utilisation d'**API Platform**, un bundle puissant et largement adopté dans l'écosystème Symfony. Grâce à lui, la majorité des endpoints REST nécessaires à l'application sont générés automatiquement à partir des entités Doctrine, simplement en les annotant correctement.

Concrètement, chaque entité de mon domaine métier, comme `Book`, `Page` ou `Adventurer`, a été enrichie avec l'annotation `@ApiResource`. Cela permet à API Platform de créer, sans effort supplémentaire, des routes RESTful telles que l'accès à la liste des livres, la création d'une aventure, la mise à jour d'un personnage ou encore la récupération d'une page d'histoire. Cette automatisation m'a permis de concentrer mes efforts sur la logique métier, sans avoir à gérer manuellement chaque route ou contrôleur pour les opérations classiques (CRUD).

Afin de garantir une bonne séparation entre les données exposées en lecture et en écriture, j'ai configuré des *groupes de sérialisation* (`read`, `write`) sur les propriétés des entités. Cela me permet de maîtriser précisément quelles données sont visibles par l'utilisateur et lesquelles sont modifiables. Dans certains cas, j'ai également activé des filtres, comme la recherche par titre, pour améliorer l'expérience utilisateur côté mobile.

Ce système s'avère non seulement efficace mais aussi robuste, car il respecte nativement les standards REST, tout en offrant une intégration parfaite avec d'autres composants de Symfony comme le système de sécurité ou la validation des entités.

[annotation ApiResource sur Book.php]

```
use ApiPlatform\Metadata\ApiResource;
use App\Repository\BookRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\DBAL\Types\Types;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Serializer\Annotation\Groups;
const string BookRe
#[ORM\Entity(repositoryClass: BookRepository::class)]
#[ApiResource(
    normalizationContext: ['groups' => ['book:read']],
    denormalizationContext: ['groups' => ['book:write']]
)]
#[ORM\HasLifecycleCallbacks]
52 references | 1 implementation
class Book
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    #[Groups(groups: ['book:read', 'adventurer:read'])]
    2 references
    private ?int $id = null;
```

Toutes les opérations de mon application ne peuvent pas être couvertes par les fonctionnalités génériques offertes par **API Platform**. En particulier, certaines actions critiques comme la gestion des **combats**, la **progression d'un joueur** ou le **traitement des choix narratifs** nécessitent une logique métier spécifique, qui doit être traitée avec rigueur et précision.

Pour répondre à ces besoins, j'ai développé des **services Symfony personnalisés**, entièrement découpés, afin de centraliser la logique métier la plus complexe. Le premier, *AdventureService.php*, est responsable de la gestion des aventures : création, suivi de la progression, passage de page en page, et détection des fins de parcours. Il assure la cohérence entre l'état du joueur et l'avancement de l'histoire, en mettant à jour les entités *Adventure* et *Page*.

Le second service, *CombatService.php*, gère les mécaniques de combat. Lorsqu'une page comporte un monstre, ce service est chargé de vérifier si le joueur est en mesure de poursuivre, ou s'il doit d'abord remporter un affrontement. Il intègre également les règles de jeu (calculs de dégâts, conditions de victoire, endurance, etc.), garantissant une progression fidèle aux contraintes du scénario.

[screen CombatService.php – méthode fight()]

```

16 references | 0 implementations
└ class CombatService
{
    6 references | 0 overrides
    public function __construct(
        private FightHistoryRepository $fightHistoryRepo,
        private EntityManagerInterface $em
    ) {}

    1 reference | 0 overrides
    public function fight(Adventurer $adventurer, Monster $monster): array
    {

        $adventurerScore = $adventurer->getAbility() + random_int(min: 1, max: 10);
        $monsterScore = $monster->getAbility() + random_int(min: 1, max: 10);

        $winner = $adventurerScore >= $monsterScore ? 'adventurer' : 'monster';

        if ($winner === 'adventurer') {
            $this->recordFight(adventurer: $adventurer, monster: $monster, victory: true);
        }

        return [
            'adventurer' => [
                'adventurerName' => $adventurer->getAdventurerName(),
                'base' => $adventurer->getAbility(),
                'roll' => $adventurerScore - $adventurer->getAbility(),
                'total' => $adventurerScore,
            ],
            'monster' => [
                'monsterName' => $monster->getMonsterName(),
                'base' => $monster->getAbility(),
                'roll' => $monsterScore - $monster->getAbility(),
                'total' => $monsterScore,
            ],
            'winner' => $winner,
            'log' => sprintf(
                format: 'Adventurer: %d + %d = %d | Monster: %d + %d = %d → %s wins',
                values: $adventurer->getAbility(),
                $adventurerScore - $adventurer->getAbility(),
                $adventurerScore,
                $monster->getAbility(),
                $monsterScore - $monster->getAbility(),
                $monsterScore,
                ucfirst(string: $winner)
            )
        ]
    }
}

```

Ces services ne sont jamais appelés directement par le frontend. Ils sont **injectés automatiquement dans les contrôleurs** via le mécanisme d'autowiring de Symfony. Ainsi, chaque contrôleur concerné *AdventureController.php*, *PageController.php*, ou encore *FightController.php* peut déléguer les traitements à ces services métier, assurant une meilleure séparation des responsabilités et une architecture en couches respectant les bonnes pratiques de développement.

Cette structuration garantit également une meilleure testabilité du code, une maintenance facilitée, et une extensibilité naturelle si de nouvelles règles de gameplay venaient à être introduites.

6.3 Développement des composants métier

La logique métier principale de l'application repose sur la progression du joueur à travers l'aventure, un processus que j'ai choisi de centraliser dans un point unique : le contrôleur *PageController*. Ce choix d'architecture permet de conserver un haut niveau de cohérence dans le déroulé narratif, tout en limitant la duplication de logique dans plusieurs services ou fichiers.

[**PageController - extrait `viewPage()`**]

```
if ($fromPageId) {
    if (!$fromPage) {
        return $this->json(['error' => 'Page précédente introuvable'], 404);
    }

    $isAccessible = false;
    foreach ($fromPage->getChoices() as $choice) {
        if ($choice->getNextPage()->getId() === $pageId) {
            $isAccessible = true;
            break;
        }
    }

    if (!$isAccessible) {
        return $this->json([
            'error' => 'Cette page n\'est pas accessible depuis la page précédente.',
            'fromPageId' => $fromPageId,
            'requestedPageId' => $pageId,
        ], 403);
    }
}

if (!$combatService->canAccessPage($fromPage, $adventurer)) {
    return $this->json([
        'error' => 'Vous devez vaincre le monstre pour continuer.',
        'monsterId' => $fromPage->getMonster()->getId(),
        'monsterName' => $fromPage->getMonster()->getMonsterName(),
    ], 403);
}

// ✅ Mise à jour de la progression
$adventureService->updatePage($adventure, $targetPage, $fromPage);
}
```

C'est ce contrôleur qui gère toutes les vérifications lorsqu'un utilisateur tente de consulter une nouvelle page. Il commence par s'assurer que cette page est bien accessible à partir de la page précédente, en analysant la liste des *Choice* disponibles. Cette vérification garantit

que le joueur suit bien le chemin prévu par le scénario, sans forcer l'accès à une page via manipulation d'URL.

En parallèle, si la page contient un monstre, une règle métier supplémentaire s'applique : l'utilisateur doit d'abord remporter un combat avant de pouvoir accéder au contenu. Le service `CombatService` est alors invoqué pour s'assurer que les conditions du combat ont été respectées et que le joueur est en droit de continuer son aventure.

Une fois toutes les vérifications terminées, le service `AdventureService` est appelé pour enregistrer la progression. Il met à jour l'état de l'aventure en enregistrant la page actuelle, la précédente, et en conservant un historique structuré des décisions du joueur. Cela

[AdventureService - updatePage()]

```
public function updatePage(Adventure $adventure, Page $newPage, Page $fromPage): void
{
    $adventure->setCurrentPage($newPage);
    $adventure->setFromLastPage($fromPage);
    $this->em->flush();
}
```

permet, entre autres, une reprise fluide des parties en cours et une validation métier solide côté serveur.

Ce mécanisme complet empêche les tentatives de triche (comme accéder à une page arbitraire en modifiant l'URL) et renforce la sécurité ainsi que l'intégrité des données de jeu. Enfin, la réponse API renvoyée au frontend est propre, lisible, et formatée en JSON pour une intégration directe côté React Native.

Gestion métier des pages par numéro

Pour assurer une cohérence entre la logique de l'interface utilisateur et la structure de la base de données, j'ai choisi de gérer les pages non pas par leur identifiant technique (ID auto-incrémenté), mais par un identifiant métier : le numéro de page (`pageNumber`) combiné à l'ouvrage auquel la page appartient (`book`).

Cela m'a permis de rapprocher la structure du code de la logique narrative propre aux livres dont vous êtes le héros, où les décisions sont orientées par des sauts de page numériques. Pour éviter toute ambiguïté, j'ai mis en place une **contrainte d'unicité** sur le couple (`book_id, page_number`) directement dans l'entité `Page` via une annotation Doctrine :

[entity Page]

```

#[ORM\Entity(repositoryClass: PageRepository::class)]
#[ORM\Table(name: "page", uniqueConstraints: [
    new ORM\UniqueConstraint(name: "unique_page_number_per_book", columns: ["book_id", "page_number"])
])]


```

Cette contrainte empêche qu'un même numéro soit utilisé deux fois dans le même livre tout en autorisant des doublons entre livres différents.

Dans l'interface d'administration (EasyAdmin), cette logique est également respectée : j'ai configuré les **relations Choice -> Page** pour utiliser le champ *pageNumber* comme étiquette, via un **choice_label**, rendant ainsi la gestion des sauts de page beaucoup plus lisible pour l'administrateur.

[PageCrudController]

```

    public function configureFields(string $pageName): iterable
    {
        return [
            IdField::new('id')->hideOnForm(), // Cacher l'ID lors de l'ajout
            TextField::new('content', 'Contenu de la page'),
            IntegerField::new('pageNumber', 'Numéro de page'),
            AssociationField::new('book', 'Livre associé'),
        ];
    }
}


```

Enfin, lors de l'import automatique des livres via JSON, la commande *importBookCommand* vérifie si une page pour un numéro donné existe déjà. Si ce n'est pas le cas, elle est créée dynamiquement, garantissant ainsi l'intégrité du livre tout en permettant une création semi-automatisée depuis un simple fichier texte.

Ce choix de conception rend l'application plus robuste, plus intuitive et mieux alignée avec la structure logique attendue par les utilisateurs.

6.4 Intégration des livres et pages à partir de fichiers PDF

L'intégration des contenus narratifs dans mon application repose sur une démarche semi-automatisée, conçue pour transformer des *livres dont vous êtes le héros* (LDVEH) au format PDF en données exploitables par mon backend. Ces ouvrages, déjà rédigés par des auteurs, suivent une structure classique : une succession de paragraphes numérotés, chacun se concluant par des choix menant à d'autres pages.

Pour éviter une saisie manuelle fastidieuse, j'ai développé un script *Python* sur mesure. Ce script scanne chaque page du PDF, détecte les numéros de paragraphes, extrait leur contenu textuel, et isole les phrases de transition contenant les choix. Ces données sont ensuite converties en un fichier JSON structuré.

Chaque objet JSON généré représente une *page* du livre et contient plusieurs champs :

- le numéro de page d'origine,
- le contenu narratif principal,
- la liste des *choix* associés (chacun pointant vers un autre numéro de page),
- et, le cas échéant, les informations sur un monstre à combattre.

[screen script Python]

```
import pdfplumber
import re
import json

pdf_path = "Loup Solitaire 04 - Le Gouffre Maudit.pdf"

paragraphs = []

with pdfplumber.open(pdf_path) as pdf:
    full_text = "\n".join(page.extract_text() for page in pdf.pages if page.extract_text())

# Regex pour détecter les paragraphes : numéro + texte
raw_paragraphs = re.findall(r"\n?(\\d+)\\n(.*)\\n\\d+\\n|\\z", full_text, flags=re.S)

# Nouvelle regex pour matcher tous types de choix
choice_pattern = re.compile(
    r"(.?)(?:\\.|\\n)?\\s*(?:rendez[- ]vous au|alle(?:z|r) au)\\s+(\\d+)",
    flags=re.IGNORECASE
)

for number, content in raw_paragraphs:
    page_number = int(number)
    content = content.strip()

    choices = []
    for match in choice_pattern.finditer(content):
        choice_text = match.group(1).strip()
        next_page = int(match.group(2))

        # fallback si pas de texte avant
        if not choice_text:
            choice_text = f"Aller au {next_page}"

        choices.append({
            "text": choice_text,
            "nextPage": next_page
        })

    paragraphs.append({
        "pageNumber": page_number,
        "content": content,
        "choices": choices
    })

with open("parsed_book.json", "w", encoding="utf-8") as f:
    json.dump(paragraphs, f, indent=2, ensure_ascii=False)

print(f"{len(paragraphs)} paragraphes exportés avec succès ✅")
```

Une fois ce fichier JSON prêt, je l'intègre dans la base de données à l'aide d'une *commande Symfony personnalisée*, **importBookCommand**. Cette commande lit chaque entrée du

JSON, crée dynamiquement les entités correspondantes (*Book*, *Page*, *Choice*, *Monster*) et les insère en base via Doctrine ORM.

Ce mécanisme me permet de gérer de manière fiable et rapide l'importation de nouveaux livres. Il garantit aussi que l'histoire peut être relue, corrigée et validée en amont avant son insertion définitive. Une fois importé, un livre devient non modifiable par l'utilisateur final. Seul un administrateur peut y accéder via l'interface back-office.

Ce système d'import constitue donc une brique essentielle de mon projet, assurant un haut niveau de qualité de données tout en optimisant le temps de développement et de mise en ligne des contenus.

En résumé, cette chaîne automatisée d'intégration me permet de gérer efficacement le contenu narratif de l'application, tout en conservant une maîtrise totale sur sa qualité, sa cohérence et sa sécurité.

[extrait JSON Livre]

```
        ],
    },
    {
        "pageNumber": 17,
        "content": "Vous levez votre arme pour frapper la créature, dont la gueule\nhérissée de crocs tranchants.",
        "choices": [
            {
                "text": "le 0",
                "nextPage": 53
            },
            {
                "text": ". Si vous tirez 1 ou 2",
                "nextPage": 274
            },
            {
                "text": "Si vous tirez entre 3 et 9",
                "nextPage": 331
            }
        ],
        "isBlocking": true,
        "monster": {
            "monsterName": "Kraan",
            "ability": 16,
            "endurance": 24
        }
    },
    ...
]
```

Détail de la structure d'une page dans le JSON

Une fois le script d'extraction Python exécuté, chaque paragraphe narratif issu du PDF est transformé en un objet JSON structuré. Ce format de données me permet de stocker et manipuler chaque page du livre comme une entité indépendante, tout en respectant la logique du récit interactif.

Chaque page du fichier JSON contient plusieurs champs essentiels. Le premier, **pageNumber**, correspond au numéro du paragraphe dans le livre d'origine. Il est fondamental pour assurer la continuité de l'histoire et permettre une navigation fiable entre les pages.

Le champ **content** contient le texte intégral du paragraphe, tel qu'il apparaît dans l'application. Il peut s'agir d'une simple narration, d'un choix stratégique, ou encore de l'annonce d'un combat. Ce contenu est restitué directement à l'utilisateur dans l'interface mobile.

Le champ **choices** est un tableau. Chaque élément du tableau représente un choix proposé à l'utilisateur. Il contient une propriété **text**, qui correspond à la formulation partielle de l'action ou de la décision (exemple : "Si vous tirez 0 à 4...") et une propriété **nextPage**, qui indique le numéro de la page vers laquelle ce choix mène.

Certaines pages incluent également des affrontements. Dans ce cas, un champ **monster** est ajouté, contenant un objet avec plusieurs propriétés :

- **monsterName** : le nom du monstre rencontré,
- **ability** : son score d'habileté,
- **endurance** : son total de points de vie.

Ces informations sont utilisées dynamiquement par le backend pour lancer un combat, gérer les règles du jeu, et déterminer l'issue de l'affrontement.

Enfin, une page peut contenir un champ **isBlocking**, défini à **true** si le joueur ne peut pas progresser sans remplir une condition précise (vaincre un monstre, réussir un test, etc.). Ce champ permet à l'API de bloquer temporairement l'accès aux pages suivantes tant que l'obstacle n'est pas levé.

Grâce à cette structure, chaque page du livre est autonome, connectée aux autres via des liens logiques, et enrichie de mécaniques de gameplay. Ce système rend l'import robuste, automatisable et parfaitement adapté à l'univers des *livres dont vous êtes le héros*.

6.5 Gestion des erreurs, cohérence métier et sécurisation des accès API

Dans le développement d'une API REST, la gestion des erreurs ne se limite pas au simple retour d'un code HTTP. Elle doit garantir l'intégrité du système, la clarté des réponses pour le frontend, et surtout la cohérence métier. J'ai structuré ma logique en ce sens, en plaçant ces trois aspects au cœur de chaque contrôleur de mon backend Symfony.

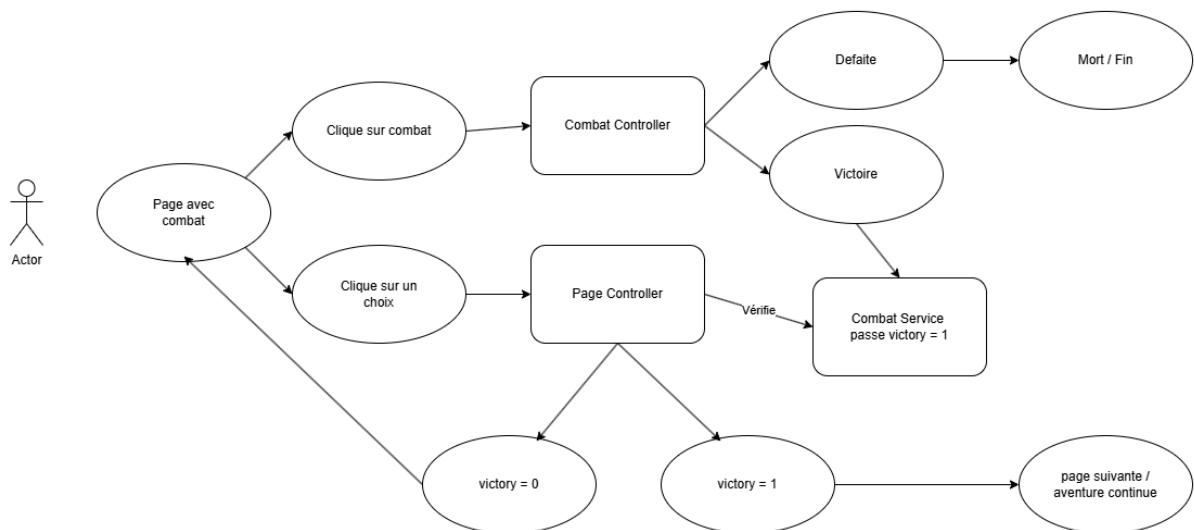
La gestion des erreurs est notamment très explicite dans *PageController.php* et *AdventureController.php*. Par exemple, si un joueur tente d'accéder à une page qui ne correspond à aucun des choix proposés depuis sa position actuelle, une réponse **403 Forbidden** est renvoyée, avec un message clair expliquant la nature du blocage.

[screen code erreur PageController]

```
if (!$isAccessible) {
    return $this->json([
        'error' => 'Cette page n\'est pas accessible depuis la page précédente.',
        'fromPageId' => $fromPageId,
        'requestedPageId' => $pageId,
    ], 403);
}
```

Un autre cas typique est la présence d'un monstre sur une page. Si le joueur n'a pas encore remporté le combat, l'accès à la page suivante est temporairement bloqué. Cette vérification est assurée par le *CombatService*, qui détermine si les conditions du scénario sont remplies.

[diagramme - Combat]



[screen erreur CombatService]

```
public function canAccessPage(Page $page, Adventurer $adventurer): bool
{
    $monster = $page->getMonster();

    if (!$monster || !$page->isCombatIsBlocking()) {
        return true; // aucun monstre ou le combat est non-bloquant
    }

    return $this->hasDefeated($adventurer, $monster);
}

public function hasDefeated(Adventurer $adventurer, Monster $monster): bool
{
    return $this->fightHistoryRepo->findOneBy([
        'adventurer' => $adventurer,
        'monster' => $monster,
        'victory' => true,
    ]) !== null;
}
```

Pour chaque situation, l'API renvoie un objet JSON structuré, contenant non seulement le code HTTP (400, 403, 404...), mais également un message lisible par l'utilisateur et, lorsque nécessaire, des données complémentaires (ID manquant, blocage narratif, combat en attente...).

Sur le plan de la sécurité, j'ai mis en place un double système de contrôle via deux firewalls distincts définis dans *security.yaml*. Le premier protège l'interface d'administration via une authentification classique (form_login). Le second sécurise l'API publique, en mode stateless, à l'aide de tokens JWT gérés par le bundle *LexikJWTAuthenticationBundle*.

[Screen sécurité - security.yaml ou controller]

Certaines routes critiques sont protégées directement dans la configuration, mais d'autres incluent des vérifications métier personnalisées dans leurs contrôleurs. C'est le cas notamment dans *AdventureController.php* pour démarrer une aventure, *AdventurerController.php* pour consulter un personnage, ou encore *PageController.php*, où chaque changement de page est validé pour éviter toute triche via l'URL.

J'ai aussi pris soin de centraliser les règles métiers pour éviter leur duplication. Chaque action critique comme commencer une aventure, progresser dans le récit ou livrer un

combat est validée par des conditions strictes : l'utilisateur doit avoir un aventurier actif, l'aventure ne doit pas être terminée, et la page cible doit être atteignable selon le scénario en cours. Ces logiques sont encapsulées dans des services tels que *AdventureService.php* et *CombatService.php*, ce qui garantit une lisibilité et une maintenabilité optimales du code.

Validation métier et réponses cohérentes

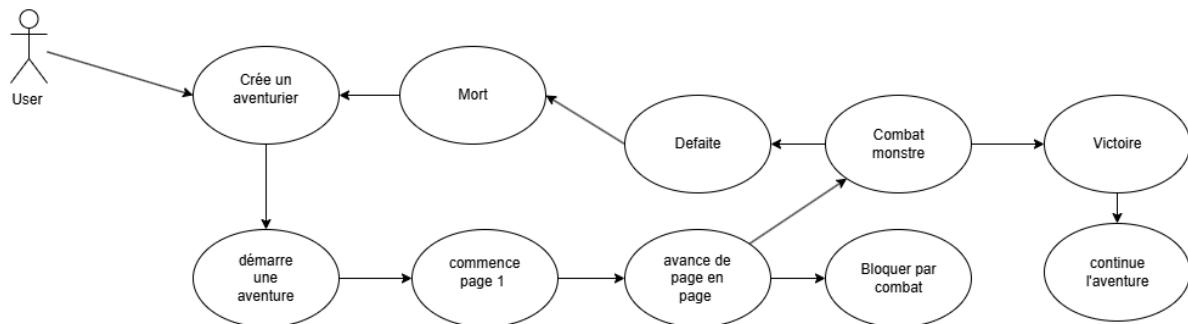
Chaque action métier commence une aventure, faire un choix, combattre un monstre suit une séquence validée par des règles métiers strictes.

Par exemple, **pour effectuer un choix ou progresser dans une histoire**, les conditions suivantes doivent être respectées :

- L'utilisateur doit avoir **déjà créé un aventurier**.
- L'aventure doit être **en cours et non terminée**.
- La **page suivante** doit être **accessible depuis la précédente**.
- En cas de page bloquante, un **combat éventuel doit être résolu avec succès** avant de poursuivre.

Lors du démarrage d'une aventure en revanche, c'est justement l'étape de création de l'aventurier et de l'instance d'aventure qui a lieu. Cette action constitue l'initiation du parcours narratif, et **ne requiert donc pas de prérequis** comme une aventure existante.

[diagramme - aventure]



Ces vérifications ne sont pas laissées au hasard. Elles sont regroupées dans les services (*AdventureService.php*, *CombatService.php*) pour éviter la duplication de logique et centraliser la maintenance.

En résumé, chaque réponse API est construite pour être à la fois fiable, cohérente avec l'état de l'aventure, et sécurisée. Les erreurs sont traitées de manière propre et explicite, sans confusion côté frontend. Cette organisation permet de garantir une expérience fluide pour l'utilisateur, tout en empêchant les appels malveillants ou la triche par manipulation des paramètres.

La cohérence métier est donc assurée à tous les niveaux : dans le code, dans la configuration, et dans les règles de progression du jeu.

7. Développement du front mobile

Le développement de l'interface mobile de l'application **LDVEH** a été réalisé à l'aide de **React Native**, en s'appuyant sur le framework **Expo**. Ce choix technologique m'a permis de produire une application réellement multiplateforme, fonctionnant aussi bien sur Android que sur iOS, à partir d'un seul et même code source.

J'ai opté pour React Native car il repose sur **TypeScript**, un langage que j'utilisais déjà pour le développement web, ce qui m'a permis de rester cohérent dans la stack technique du projet. En complément, **Expo** m'a facilité les phases de test et d'émulation sur mobile, notamment grâce à son intégration rapide avec des appareils physiques via l'application *Expo Go*.

L'approche modulaire de React Native m'a permis de structurer l'interface de manière claire et évolutive, avec des composants réutilisables, des écrans bien séparés, et une architecture facilitant les appels API et la navigation. L'expérience utilisateur a été pensée dès le début pour mobile, avec un design sobre et inspiré de l'univers *dark fantasy* du livre d'origine, tout en garantissant lisibilité, confort de lecture et intuitivité.

Le front mobile joue un rôle central dans l'expérience LDVEH : c'est à travers lui que les utilisateurs lisent, interagissent avec l'histoire, font des choix, et vivent leur aventure. Chaque interaction, chaque décision est connectée en temps réel à l'API backend, ce qui m'a poussé à accorder une attention particulière à la fluidité des transitions, à la gestion des états et à la robustesse des communications réseau.

7.1 Conception de l'architecture mobile

Pour structurer mon application mobile LDVEH, j'ai adopté une architecture modulaire basée sur la séparation claire des responsabilités. Chaque partie de l'application est organisée dans un dossier dédié, ce qui facilite la lisibilité du code et sa maintenance à long terme.

Les **écrans principaux** de navigation sont regroupés dans le dossier *screens*, tandis que les **composants réutilisables**, tels que les boutons, les entêtes ou les cartes de livre, sont stockés dans le dossier *components*. J'ai également créé un dossier *services* pour centraliser tous les appels API et la logique liée aux requêtes HTTP. À l'intérieur, le fichier *api.ts* prend en charge la communication générale avec le backend, tandis que le fichier *auth.ts* gère spécifiquement l'authentification, y compris la gestion des tokens JWT, leur stockage local, et leur injection dans les en-têtes des requêtes sécurisées.

J'ai aussi utilisé des *hooks personnalisés* pour extraire certaines logiques comme la récupération des données ou la navigation conditionnelle, ce qui m'a permis de garder des

écrans plus propres et centrés sur l'affichage. Cette structure, inspirée des bonnes pratiques de React Native, rend le code plus intuitif, et m'a permis de monter rapidement un prototype tout en gardant une base évolutive pour les fonctionnalités futures.

7.2 Navigation entre les écrans

La navigation au sein de l'application mobile LDVEH repose sur **expo-router**, une solution qui s'appuie sur la structure de routage dynamique inspirée de Next.js, mais adaptée à l'environnement React Native. Ce choix m'a permis de tirer parti d'un système de navigation clair et intuitif, dans lequel chaque écran correspond directement à un fichier ou à un dossier situé dans le répertoire **app/**.

Cette organisation rend la logique de navigation extrêmement lisible. Par exemple, le fichier **app/books/index.tsx** représente l'écran qui liste tous les livres disponibles. Lorsqu'un utilisateur s'authentifie via **LoginScreen.tsx**, il est automatiquement redirigé vers cette page. En sélectionnant un ouvrage, il est ensuite guidé vers **app/adventure/start.tsx**, où il peut créer un personnage en saisissant un nom d'aventurier. Une fois cette étape franchie, il entre dans l'expérience interactive via **app/adventure/page/[id].tsx**, qui gère dynamiquement le rendu de chaque paragraphe de l'histoire.

Cette architecture simple et robuste rend la navigation fluide et le code facilement maintenable. Elle facilite également l'ajout de nouveaux écrans ou de transitions sans nécessiter une reconfiguration complexe des routes.

7.3 Appels API et gestion de session

L'ensemble des communications entre l'application mobile et le backend Symfony repose sur une **API REST sécurisée**. Tous les appels sont centralisés dans le fichier **api.ts**, qui encapsule les requêtes **fetch** avec les en-têtes appropriés, les méthodes HTTP, et la gestion des erreurs. Cela permet de simplifier l'écriture des appels réseau dans les composants de l'application et de garantir une logique cohérente à travers toute l'interface.

L'authentification est gérée via **JWT** (JSON Web Token). À la connexion, un token est reçu depuis l'API Symfony. Ce token est ensuite stocké de manière persistante et sécurisée à l'aide de **AsyncStorage**, une solution de stockage local adaptée à React Native. Le fichier **auth.ts** est responsable de la récupération de ce token et de son intégration automatique dans les en-têtes **Authorization (Bearer token)** pour chaque requête.

```

import AsyncStorage from '@react-native-async-storage/async-storage';

const TOKEN_KEY = 'access_token';

export async function saveToken(token: string) {
  await AsyncStorage.setItem(TOKEN_KEY, token);
}

export async function getToken(): Promise<string | null> {
  return await AsyncStorage.getItem(TOKEN_KEY);
}

export async function clearToken() {
  await AsyncStorage.removeItem(TOKEN_KEY);
}

```

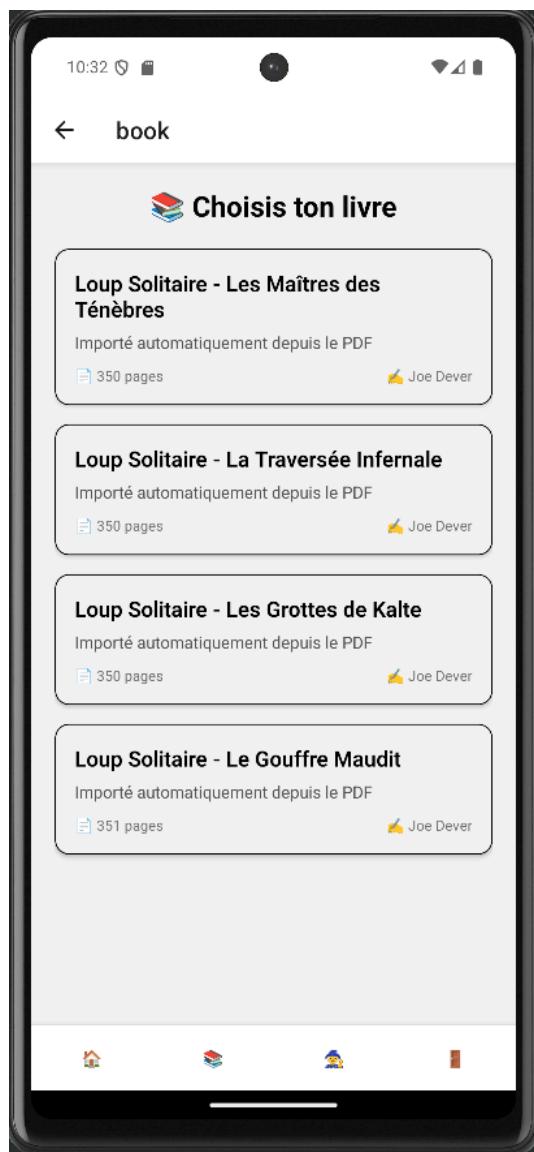
Pour compléter cette architecture, j'utilise également **Zustand**, une librairie légère de gestion d'état, afin de stocker les données de session de l'utilisateur (comme son profil, son aventurier actif, ou son token). Cela permet un accès global et rapide à ces informations depuis n'importe quel écran, sans avoir à passer systématiquement par des props ou des contextes complexes. **Zustand** facilite aussi la mise à jour de l'interface en fonction des changements d'état (connexion, déconnexion, chargement, etc.).

Cette combinaison d'outils assure une **communication fluide, sécurisée et maintenable** entre le frontend mobile et le backend, tout en garantissant une **expérience utilisateur stable et réactive**.

7.4 Accès aux livres et création d'une aventure

Dès que l'utilisateur se connecte à l'application, il est automatiquement redirigé vers l'écran **BooksScreen.tsx**. Cet écran joue un rôle central, car il présente l'ensemble des livres interactifs disponibles. Pour ce faire, une requête **GET /api/books** est effectuée vers l'API Symfony. Les livres sont ensuite affichés dynamiquement sous forme de **cartes interactives** contenant leur titre, une courte description, et un bouton permettant de commencer une aventure.

[screen des page front end bookCard/ bookList]



Lorsqu'un utilisateur sélectionne un livre, il est redirigé vers l'écran **StartAdventureScreen.tsx**, où débute réellement son expérience narrative. Sur cette page, il est invité à **saisir un nom d'aventurier**, qui correspondra à son personnage tout au long de l'histoire. Une fois le nom renseigné, l'application envoie une requête **POST /api/adventurers** pour créer l'entité correspondante dans la base de données. Ce nouvel aventurier est alors automatiquement associé à une **nouvelle aventure** active, gérée côté serveur via le **service AdventureService.php**.

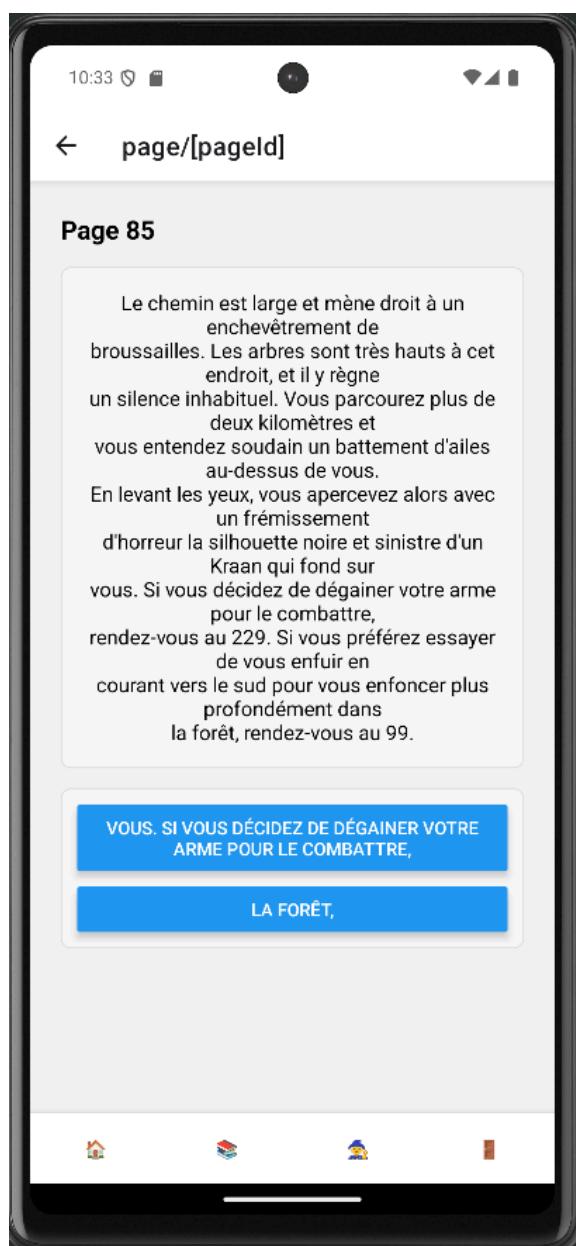
Cette étape constitue le **point de départ d'une session de jeu complète**. Dès que le personnage est créé, l'utilisateur est redirigé vers la première page de l'histoire, où débute la lecture interactive.

7.5 Affichage d'une page

Une fois l'aventure commencée, l'utilisateur accède à l'écran **PageScreen.tsx**, qui constitue le cœur de l'expérience interactive. À chaque chargement de page, une requête **GET /api/page/{id}** est envoyée à l'API Symfony. Cette requête renvoie toutes les données nécessaires à l'affichage dynamique : **le numéro de page, le contenu narratif, les choix disponibles**, et s'il y a lieu, **les informations du monstre à combattre**.

Le contenu est ensuite affiché dans un composant React structuré, permettant à l'utilisateur de lire le texte de la page dans un format clair et immersif. **Les choix proposés** sont rendus sous forme de **boutons cliquables**, chacun correspondant à une action possible ou une direction à suivre dans le récit.

[sceen du front page]



Lorsqu'un utilisateur clique sur l'un de ces choix, une **nouvelle requête GET** est envoyée au backend, incluant cette fois l'**ID de la page précédente**. Cette information est cruciale, car

elle permet au serveur de **vérifier que la transition est valide**. Si l'utilisateur tente de forcer une navigation non autorisée ou incohérente (par exemple, en modifiant l'URL manuellement), une réponse d'erreur est renvoyée, avec un **message clair expliquant la nature du blocage**.

Ce système garantit une navigation **100 % contrôlée par le backend**, préservant à la fois la logique narrative du livre et l'intégrité de la progression de l'aventure.

7.6 Gestion des états et expérience utilisateur

L'un des enjeux principaux dans ce type d'application narrative est de garantir une **expérience de jeu fluide et persistante**. À chaque interaction avec une page, les données de progression comme la position actuelle dans le livre, les choix effectués ou encore l'état du combat sont **immédiatement synchronisées avec le backend**. Cette synchronisation permet à l'utilisateur de **quitter l'application à tout moment et de reprendre sa partie exactement là où il l'avait laissée**, sans perte d'information.

Pour assurer cette fluidité, j'ai intégré **des retours visuels lors des chargements** (indicateurs de chargement, transitions douces) afin de toujours informer l'utilisateur de l'état du système. En cas d'erreur réseau ou d'appel mal formé, des **messages explicites** sont affichés pour éviter toute confusion.

L'interface elle-même a été pensée pour s'aligner avec l'ambiance **dark fantasy** du projet. Elle reste volontairement sobre, avec une typographie lisible, des contrastes marqués et une navigation intuitive centrée sur la narration.

7.7 Reprise d'une aventure en cours

L'application permet à chaque utilisateur de **reprendre une partie interrompue à tout moment**, sans perdre sa progression. Cette fonctionnalité repose sur un écran dédié, accessible via *AdventurersScreen.tsx*, qui affiche **l'ensemble des personnages créés** par l'utilisateur ainsi que **le titre du livre associé à chaque aventure**.

Chaque ligne de cette liste résume l'état de l'aventure, qu'elle soit en cours ou terminée, et indique **la dernière page atteinte**. Un bouton d'action permet de **reprendre directement la lecture**, en redirigeant automatiquement l'utilisateur vers l'écran *PageScreen.tsx*, avec toutes les données nécessaires (aventurier, position, état de l'histoire) déjà chargées.

Cette mécanique est rendue possible grâce à **l'enregistrement régulier de la progression côté backend**, orchestré par *AdventureService.php* via la méthode *updatePage()*. À chaque

interaction significative (changement de page, choix effectué, combat résolu), les informations sont mises à jour en base de manière fiable.

Grâce à ce système, l'utilisateur conserve une continuité de lecture même après fermeture de l'application, garantissant une expérience immersive, fluide et sécurisée.

8. Préparation et exécution des tests

8.1 Tests côté backend

Pour valider la stabilité du cœur métier de l'application, j'ai mis en place des **tests unitaires ciblés** à l'aide de **PHPUnit**, le framework de test standard de l'écosystème Symfony. Ces tests visent principalement à sécuriser les parties critiques du projet, notamment celles liées à la logique métier de progression et de combat.

Les tests sont stockés dans le dossier `tests/`, organisé de manière claire par domaine fonctionnel. Par exemple :

tests/Service/AdventureServiceTest.php : vérifie le bon enregistrement de la progression dans une aventure, la validation des pages atteintes, ainsi que la gestion des fins de scénario.

tests/Service/CombatServiceTest.php : s'assure que les règles du système de combat sont correctement appliquées : gestion de l'attaque, calcul des points d'endurance, détection de victoire ou de défaite.

[screen php bin/console phpunit command]

```
C:\project_CDA\api_LDVEH>php vendor\phpunit\phpunit\phpunit
PHPUnit 10.5.45 by Sebastian Bergmann and contributors.

Runtime:      PHP 8.3.6
Configuration: C:\project_CDA\api_LDVEH\phpunit.xml.dist

..... 65 / 67 ( 97%)
.. 67 / 67 (100%)

Time: 00:01.133, Memory: 10.00 MB

OK, but there were issues!
Tests: 67, Assertions: 98, PHPUnit Deprecations: 1.
```

Chaque test utilise des données simulées en mémoire via Doctrine, sans altérer la base réelle. Cela permet d'effectuer des vérifications sur les cas limites (ex. : tentative de progression vers une page interdite, combat perdu, etc.) tout en garantissant une exécution rapide et fiable.

Ces tests ont été essentiels pour détecter plusieurs bugs logiques en cours de développement, notamment sur les enchaînements de choix ou la validation des combats bloquants.

8.1.1 Matrice de tests fonctionnels et de sécurité

En complément des tests unitaires réalisés côté backend, j'ai également conçu une **matrice de tests manuels** couvrant l'ensemble des fonctionnalités principales de l'application. Cette matrice vise à valider le bon comportement de l'API, l'interface d'administration et les parcours utilisateur, tout en prenant en compte la **sécurité**, la **navigation**, les **règles métiers** et les **cas limites**.

La matrice est organisée en plusieurs sections thématiques :

- **Administration** (interface EasyAdmin, création d'entités, accès selon rôle)
- **Navigation dans les livres** (choix, pages bloquantes, combats, redirections)
- **Utilisateurs & Aventuriers** (authentification, création, permissions)
- **Combats** (déclenchement, règles de victoire/défaite, sauvegarde des tours)
- **Sécurité** (authentification JWT, droits d'accès, restrictions)
- **Choix & Pages** (redirections, conditions, objets nécessaires)

Chaque test est identifié par un **ID unique**, décrit le scénario, les données d'entrée, le résultat attendu, le type de test, et son statut de validation.

Cette matrice m'a permis de garantir que **toutes les fonctionnalités critiques étaient bien testées**, que les règles métiers codées dans le backend étaient respectées, et que les utilisateurs ne pouvaient pas contourner les contraintes prévues. Elle joue également un rôle de documentation technique utile pour tout futur développeur ou testeur.

La matrice complète des tests fonctionnels et de sécurité est disponible en annexe, pages 56 à 59

8.1.2 Gestion des références circulaires lors de la sérialisation

Pendant le développement de mon API, j'ai rencontré un blocage important lié à la structure de mes entités. Les entités **Page** et **Choice** étaient liées entre elles dans les deux sens : une page contenait plusieurs choix, et chaque choix possédait un lien vers une autre page. Cette logique, nécessaire pour représenter un récit interactif, posait un problème lors de la sérialisation des données envoyées au frontend.

Lorsque j'appelais une page via l'API, Symfony tentait de sérialiser non seulement la page, mais aussi ses choix, puis les pages liées à ces choix, puis les choix de ces pages, et ainsi de suite... Le processus entrait alors dans une boucle infinie, rendant la réponse inutilisable, voire provoquant des erreurs fatales ou des temps de réponse bloqués.

Pour corriger ce comportement, j'ai restreint la profondeur de sérialisation des relations entre les entités à l'aide de l'annotation **@MaxDepth(1)** sur les relations concernées dans mes entités Doctrine. J'ai également activé explicitement cette limitation dans la configuration du serializer Symfony, ce qui permet de couper proprement la chaîne de sérialisation dès qu'un certain niveau de relation est atteint.

Cette correction a été essentielle pour stabiliser les appels API et rendre les données lisibles et performantes côté client. Ce type de problématique, fréquent dès qu'on travaille avec des modèles fortement relationnels, m'a permis de mieux comprendre les subtilités de la sérialisation en environnement Symfony.

j'ai trouvé réponse a mes question sur stack Overflow au lien suivant

<https://stackoverflow.com/questions/59268438/a-circular-reference-has-been-detected-when-serializing-the-object-of-class-app>

8.2 Tests côté frontend

Bien que l'architecture React Native utilisée pour le frontend soit compatible avec des frameworks de test comme **Jest** ou **React Native Testing Library**, je n'ai pas mis en place de tests unitaires automatisés faute de temps.

En revanche, chaque fonctionnalité majeure a fait l'objet de **tests manuels rigoureux**, effectués sur plusieurs plateformes :

- **Smartphone Android physique via Expo Go**
- **Navigateurs (Chrome, Firefox) en mode web via Expo Web**

Les flux critiques testés comprennent : l'authentification, la sélection de livres, la création d'aventure, la navigation entre pages, le système de choix et la reprise d'une aventure en cours.

8.3 Justificatif complémentaire

En parallèle de ce projet, j'ai travaillé dans un environnement professionnel où j'ai contribué à la **maintenance de plus de 500 tests automatisés** sur une plateforme e-commerce. Cette expérience m'a permis d'acquérir des **compétences avancées en stratégie de test**, en analyse de logs, et en outils de diagnostic comme Postman ou PHPUnit.

```
cmachtelinckx@LP210011 ~/dev/amg-api (335-ajout-de-groupe-pour-comptabilite) $ dcu
[+] Running 5/5
✓ Container amg-api-memcached-1 Started
✓ Container amg-api-db-1 Started
✓ Container amg-api-app-1 Started
✓ Container amg-api-ssl-1 Started
✓ Container amg-api-front-1 Started
cmachtelinckx@LP210011 ~/dev/amg-api (335-ajout-de-groupe-pour-comptabilite) $ dce bin/phpunit
PHPUnit 9.5.28 by Sebastian Bergmann and contributors.

Testing ..... 63 / 785 (  8%)
..... 126 / 785 ( 16%)
..... 189 / 785 ( 24%)
..... 252 / 785 ( 32%)
..... 315 / 785 ( 40%)
..... 378 / 785 ( 48%)
..... 441 / 785 ( 56%)
..... 504 / 785 ( 64%)
..... 567 / 785 ( 72%)
..... 630 / 785 ( 80%)
..... 693 / 785 ( 88%)
..... 756 / 785 ( 96%)
..... 785 / 785 (100%)

Time: 00:53.360, Memory: 203.00 MB

OK (785 tests, 2571 assertions)
cmachtelinckx@LP210011 ~/dev/amg-api (335-ajout-de-groupe-pour-comptabilite) $
```

9. Déploiement de l'application

9.1 Mise en place de l'environnement Docker pour le développement

Dans une optique de professionnalisation et de collaboration, j'ai mis en place un environnement de développement conteneurisé à l'aide de Docker. Cela m'a permis de

garantir que, si d'autres développeurs devaient rejoindre le projet, ils pourraient rapidement disposer d'une configuration identique à la mienne, quel que soit leur système d'exploitation (Windows, Mac, Linux).

J'ai utilisé un fichier **docker-compose.yml** pour lancer les services nécessaires au bon fonctionnement du backend Symfony : un serveur PHP, une base de données MySQL, et phpMyAdmin pour l'administration. Ce fichier est situé à la racine du projet backend, prêt à être exécuté via la commande **docker-compose up**.

Cependant, pour des raisons de performances (mon poste de développement étant peu puissant), j'ai principalement développé en local sans Docker. Cela m'a permis de conserver un temps de réponse rapide durant le développement et les tests. Néanmoins, le fichier **docker-compose.yml** est opérationnel et testé, ce qui garantit une portabilité maximale du projet.

Ce choix d'anticiper une possible collaboration m'a permis d'adopter des pratiques professionnelles et de structurer mon projet de façon à ce qu'il soit facilement maintenable et partageable.

9.2 Déploiement du backend en production (Scalingo)

Une fois l'API fonctionnelle et testée en local, j'ai procédé à son déploiement en ligne via la plateforme **Scalingo**, une PaaS française spécialisée dans l'hébergement d'applications web. Ce choix m'a permis de disposer d'un environnement de production stable, sécurisé, et accessible publiquement pour les démonstrations.

Le déploiement a été effectué à l'aide d'un **dépôt Git distant** pointant vers Scalingo, avec les étapes suivantes :

```
scalingo create ldveh1
```

```
scalingo git:remote -a ldveh1
```

```
git push scalingo main
```

Voir annexe : [Capture du terminal lors du déploiement Scalingo] – [image cd du deployment]

Le backend est désormais accessible à l'URL suivante :

<https://ldveh1.osc-fr1.scalingo.io>

9.3 Configuration de l'environnement côté frontend

Pour que le frontend React Native communique correctement avec l'API, j'ai mis en place un système simple de **switch d'environnement** via une variable **BASE_URL** dans le fichier **api.ts**. Cela me permet de changer rapidement l'environnement cible (local ou production) sans modifier le reste du code.

```
const URL = 'https://ldveh1.osc-fr1.scalingo.io';

// const BASE_URL = 'https://localhost:8000'; // For local development,
// const BASE_URL = 'http://localhost:8000'; // For local development with browser
// const BASE_URL = 'https://ldveh1.osc-fr1.scalingo.io'; // For production
export const BASE_URL = URL;
export const API_URL = `${URL}/api`;
```

Cette approche garantit une transition fluide entre les environnements de développement, de test et de production.

10. Jeu d'essai – Combat bloquant la progression

Pour tester le bon fonctionnement des règles métiers liées à l'accès conditionnel aux pages, j'ai réalisé un scénario centré sur une mécanique de combat. Certaines pages de l'application ne sont accessibles que si le joueur remporte un affrontement contre un monstre. Cette logique est encadrée par le service **CombatService** et exécutée via le contrôleur **PageController**.

L'objectif de ce test était de m'assurer que la progression dans une aventure respecte bien les contraintes prévues par le scénario, notamment lorsqu'une page est marquée comme bloquante (**isBlocking = true**) et qu'elle contient un monstre.

À l'aide de **Postman**, j'ai simulé un utilisateur qui débute son aventure sur une première page

[image début d'aventure]

POST <https://localhost:8000/api/adventure/start>

Params Authorization (1) Headers (10) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary

```
1 {  
2   "bookId": 1,  
3   "adventurerName": "aventurierExemple"  
4 }
```

Body Cookies Headers (8) Test Results | ⌂

{ } JSON ▾ ▶ Preview ⚡ Visualize ▾

```
1 {  
2   "message": "Aventure démarrée",  
3   "adventureId": 8,  
4   "adventurerId": 12,  
5   "pageId": 1,  
6   "fromPageId": null  
7 }
```

[user commence à la première page du livre]

The screenshot shows a Postman request for `https://127.0.0.1:8000/page/1/adventurer/12/`. The 'Authorization' tab is selected, showing a Bearer Token. The response status is `200 OK` with a response time of 432 ms and a size of 2.2 KB. The response body is a JSON object:

```
1 {  
2     "pageId": 1,  
3     "pageNumber": 1,  
4     "content": "Il faut vous hâter, car quelque chose vous dit qu'il serait\\n imprudent de vous attarder parmi les ruines fumantes du\\n monastère détruit. Les monstres volants peuvent, en effet,  
\\nreparaître à tout moment. Il n'y a d'ailleurs pas de temps à\\n perdre : vous devez au plus vite prendre la route de Holmgard, la\\ncapitale du Sommeland, pour aller annoncer au Roi  
l'\\n terrible nouvelle : les Guerriers Kai, l'\\n élite du pays, ont tous été\\n massacrés, à l'\\n exception de vous-m\\nême. Or sans l'autorité et le\\n savoir des Seigneurs Kai pour commander  
l'\\n armée, le royaume\\n du Sommeland se trouve à la merci de ses plus anciens\\nennemis: les Maîtres des Ténèbres. En retenant vos larmes à\\n grand-peine, vous dites adieu à vos  
compagnons morts et vous\\nfaites le serment de les venger. Vous tournez alors le dos aux\\nruines et vous descendez avec précaution le sentier escarpé qu'ils\\n'ouvre devant vous. Au pied  
de la colline, le chemin aboutit à\\n une bifurcation. Là, deux autres sentiers mènent l'un et l'autre à\\n une grande forêt en empruntant deux directions différentes. Si\\nvous souhaitez  
prendre le sentier de droite, rendez-vous au 86.\\n Si vous préférez suivre celui de gauche, rendez-vous au 275.\\n Enfin, si vous maîtrisez la Discipline Kai du Sixième Sens,  
\\n rendez-vous au 141.",  
5     "monsterId": null,  
6     "monster": null,  
7     "canAccess": true,  
8     "isBlocking": false,  
9     "choices": [  
10         {  
11             "text": "vous souhaitez prendre le sentier de droite,",  
12             "nextPage": 86  
13         },  
14         {  
15             "text": "vous préférez suivre celui de gauche,",  
16             "nextPage": 275  
17         }  
].
```

Depuis une page source (page A), un choix mène vers une autre page (page B), qui contient un monstre. Sans victoire préalable, l'accès à cette page doit être refusé. Lors de l'appel à l'API **GET /api/page/{id}/from{id}** avec les bons paramètres (aventurier et page précédente), le backend déclenche le **CombatService**, qui vérifie si le combat a déjà été gagné. Si ce n'est pas le cas, l'API renvoie un code 403 Forbidden, avec un message clair précisant que le combat est requis.

[Page bloquante avec combat]

The screenshot shows a REST client interface with the following details:

- URL:** https://127.0.0.1:8000/page/229/adventurer/12/from/85
- Method:** GET
- Authorization:** Bearer Token (with token value eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9)
- Response Status:** 200 OK
- Response Headers:** 765 ms, 1.57 KB
- Response Body (JSON):**

```

1  {
2    "pageId": 229,
3    "pageNumber": 229,
4    "content": "Le Kraan vole au-dessus de votre tête en soulevant des nuages de poussière par le seul battement de ses ailes immenses. Bientôt, vous avez le nez et les yeux pleins de poussière et vous vous mettez à tousser et à cligner les paupières. Puis, soudain, le monstre vous attaque. Vous allez devoir le combattre jusqu'à la mort de l'un de vous deux, mais en raison de la poussière qui vous désavantage, il vous faut réduire de 1 point votre total d'HABILETÉ pendant toute la durée de l'affrontement.\nKRAAN HABILETÉ: 16 ENDURANCE: 26\nSi vous êtes vainqueur, vous avez le choix entre : vérifier si la créature ne transportait pas quelque objet qui pourrait mériter notre intérêt. Rendez-vous alors au 267 ; ou poursuivre votre chemin le long du sentier orienté à l'est. Rendez-vous pour cela au 125.",
5    "monsterId": 18,
6    "monster": "Kraan",
7    "canAccess": false,
8    "isBlocking": true,
9    "choices": [
10      {
11        "text": "vérifier si la créature ne transportait pas quelque objet qui pourrait mériter notre intérêt.",
12        "nextPage": 267
13      },
14      {
15        "text": "poursuivre votre chemin le long du sentier orienté à l'est.",
16        "nextPage": 125
17      }
18    ],
19    "endingType": null
20  }

```

Pour empêcher les aventuriers de fuir le combat, une erreur 403 Forbidden est renvoyée s'ils essaient de quitter la page sans avoir vaincu le monstre.

[quitter une page de combat sans avoir vaincu le monstre]

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** https://127.0.0.1:8000/page/125/adventurer/12/from/229
- Authorization:** Bearer Token (selected)
- Headers (8):** Not explicitly listed, but a warning message indicates sensitive data is present.
- Token:** eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ...
- Body:** JSON response containing an error message.
- Error Message (JSON):**

```
1: {  
2:   "error": "Vous devez vaincre le monstre pour continuer.",  
3:   "monsterId": 18,  
4:   "monsterName": "Kraan"  
5: }
```
- Status:** 403 Forbidden (highlighted with a red box).

Ensuite, j'ai simulé un combat réussi à l'aide d'un appel à l'endpoint de combat (**POST /api/fight**), ce qui met à jour l'état de l'aventurier dans la base de données.

[image combat]

The screenshot shows a POST request to `https://localhost:8000/fight`. The request body is a JSON object with two fields: `adventurerId` (value 12) and `monsterId` (value 18). The response body is a JSON object containing details of the combat, including adventurer and monster stats and a log message indicating the adventurer won.

```
1 {  
2   "adventurerId": 12,  
3   "monsterId": 18  
4 }
```

```
1 {  
2   "adventurer": {  
3     "adventurerName": "aventurierExemple",  
4     "base": 19,  
5     "roll": 6,  
6     "total": 25  
7   },  
8   "monster": {  
9     "monsterName": "Kraan",  
10    "base": 16,  
11    "roll": 6,  
12    "total": 22  
13  },  
14  "winner": "adventurer",  
15  "log": "Adventurer: 19 + 6 = 25 | Monster: 16 + 6 = 22 => Adventurer wins"  
16 }
```

Chaque combat est enregistré dans la table **fightHistory**, avec les IDs de l'aventurier et du monstre, ainsi qu'un champ **victory** indiquant le vainqueur : 1 pour l'aventurier, 0 pour le monstre. Seuls les aventuriers victorieux peuvent continuer leur progression.

[image base de donné table FlightHistory]

			id	adventurer_id	monster_id	victory	
← T →		▼					
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	1	18	1
<input type="checkbox"/>	 Edit	 Copy	 Delete	2	7	18	1
<input type="checkbox"/>	 Edit	 Copy	 Delete	3	1	18	1
<input type="checkbox"/>	 Edit	 Copy	 Delete	5	10	18	1
<input type="checkbox"/>	 Edit	 Copy	 Delete	6	11	18	1
<input type="checkbox"/>	 Edit	 Copy	 Delete	7	12	18	1

Une fois le combat remporté, j'ai effectué un nouvel appel vers la page B. L'accès est alors autorisé, la page se charge correctement, et la progression de l'aventurier est mise à jour via le **AdventureService**.

Après avoir vaincu le monstre, l'aventurier est libre de continuer son aventure.

[image page suivante après combat]

The screenshot shows a POSTMAN interface with the following details:

- Method:** GET
- URL:** <https://127.0.0.1:8000/page/125/adventurer/12/from/229>
- Authorization:** Bearer Token
- Headers:** (8)
- Body:** (JSON) - The response body is a JSON object with the following content:

```
1 {  
2     "pageId": 125,  
3     "pageNumber": 125,  
4     "content": "Le chemin mène à une grande clairière. Vous remarquez aussitôt sur le sol d'étranges empreintes de pattes griffues. De toute évidence, en juger par le nombre d'empreintes et la surface qu'elles couvrent, ce sont au moins cinq de ces répugnantes créatures qui se sont rassemblées l'autre côté de la clairière. Deux chemins s'enfoncent dans la forêt. L'un est orienté à l'ouest, l'autre au sud. Si vous souhaitez emprunter rendez-vous au 27. Si vous préférez prendre celui qui va vers l'ouest, rendez-vous au 214. Enfin, si vous maîtrisez la Discipline Kai de l'Ori  
5     "monsterId": null,  
6     "monster": null,  
7     "canAccess": true,  
8     "isBlocking": false,  
9     "choices": [  
10         {  
11             "text": "si vous maîtrisez la Discipline Kai de l'Orientation, rendez-vous au 301.",  
12             "nextPage": 301  
13         },  
14         {  
15             "text": "emprunter le sentier orienté au sud.",  
16             "nextPage": 27  
17         },  
18         {  
19             "text": "prendre celui qui va vers l'ouest.",  
20             "nextPage": 214  
21         }  
22     ]  
23 }
```

Ce test m'a permis de valider la cohérence de toute la chaîne de décision côté backend, du déclenchement de la règle métier jusqu'à la réponse API. Il illustre aussi la capacité de l'application à gérer des situations dynamiques et à garantir une progression narrative contrôlée.

11. Conclusion

Ce projet m'a permis de consolider l'ensemble des compétences que j'ai acquises durant ma formation, tout en découvrant de nouveaux outils concrets. J'ai notamment appris à utiliser React Native et Zustand, que je n'avais jamais utilisés auparavant. Ces technologies m'ont permis de créer une interface mobile performante et structurée, en m'adaptant aux exigences du développement multiplateforme.

Au-delà des aspects techniques, ce projet m'a beaucoup apporté en termes de rigueur, de structuration et de capacité à concevoir une application complète, de l'idée initiale jusqu'aux tests. Il m'a aussi poussé à affiner ma réflexion sur l'expérience utilisateur et sur les choix techniques à long terme.

J'ai particulièrement pris plaisir à développer LDVEH, un projet qui me tient à cœur et qui m'a poussé à aller plus loin dans la logique métier et dans l'expérience utilisateur. Travailler sur une application liée à un univers que j'aime m'a donné une motivation constante, et m'a permis de maintenir une exigence élevée tout au long du développement.

Sur le plan humain, cette expérience m'a permis de mieux comprendre l'importance de l'environnement de travail. Travailler seul sur un projet qui me passionne m'a apporté beaucoup de liberté, mais aussi montré les limites de l'isolement, notamment face à certaines tâches longues ou répétitives.

Avec le recul, même si des divergences de rythme ou d'implication peuvent exister dans un groupe, j'aurais sans doute eu intérêt à conserver une dynamique collective. Ne serait-ce que pour pouvoir déléguer certaines parties du travail et bénéficier de regards croisés sur les choix techniques.

Pour l'avenir, j'aimerais pousser LDVEH encore plus loin. Mon objectif serait d'en faire un véritable SaaS, accessible à un large public. Pour cela, je prévois de me renseigner sur les droits d'auteur liés aux livres utilisés, d'intégrer des fonctionnalités de monétisation (système de jetons pour lancer une partie ou achat direct de livres), ainsi que de mettre en place un système publicitaire intégré. Le potentiel est là, et je compte bien continuer à faire évoluer ce projet.

12. Remerciements

Je tiens à remercier toute l'équipe pédagogique de **La Plateforme**, et particulièrement **Aïcha** et **Thierry**, pour leur accompagnement, leur disponibilité et leurs conseils tout au long de ma formation.

Un grand merci également à mes camarades de classe. Grâce à eux, j'ai passé trois années riches en apprentissage, en entraide, et en défis techniques relevés ensemble. Le partage de nos connaissances et la bienveillance du groupe m'ont énormément apporté, autant sur le plan professionnel que personnel.

Merci à tous ceux qui ont rendu cette aventure possible.

13 . Annexes

diagramme de séquence UML : PageController

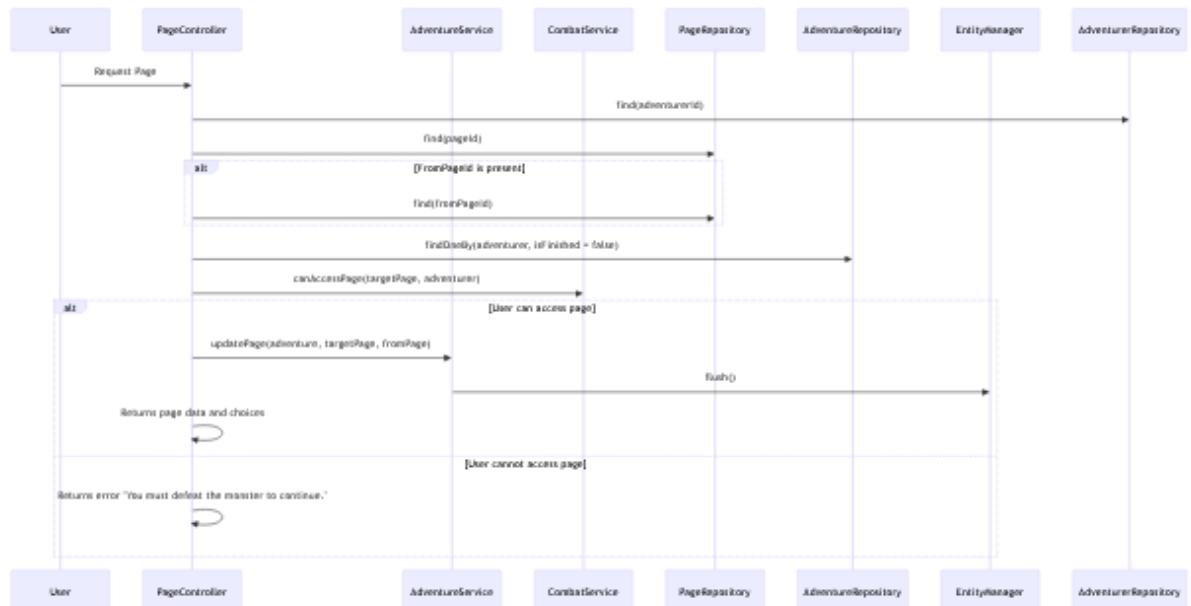
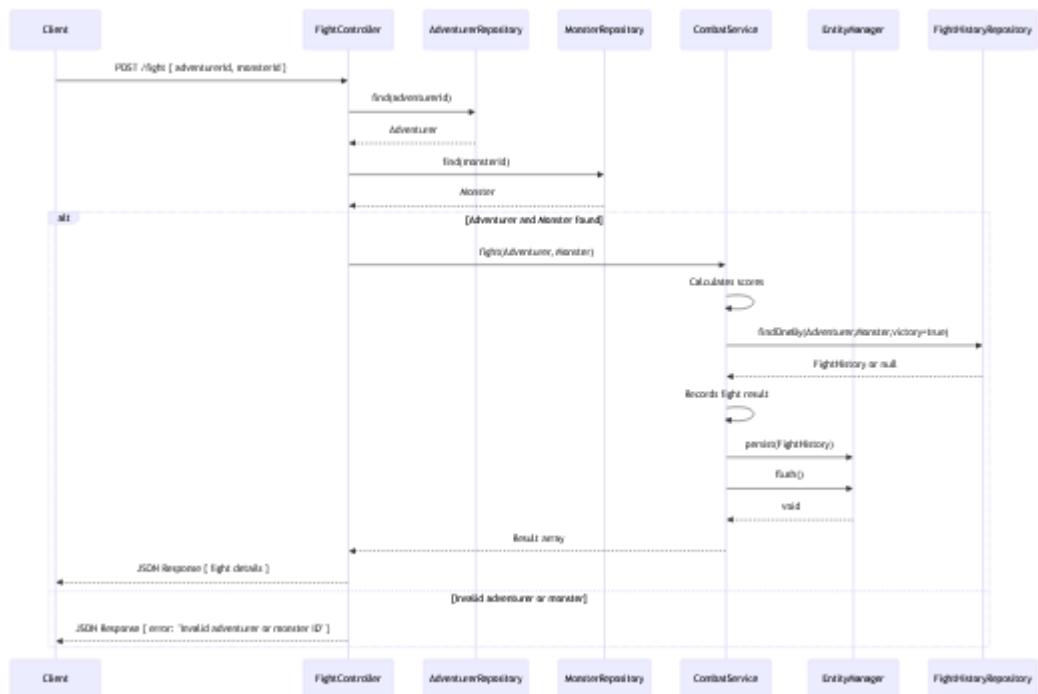
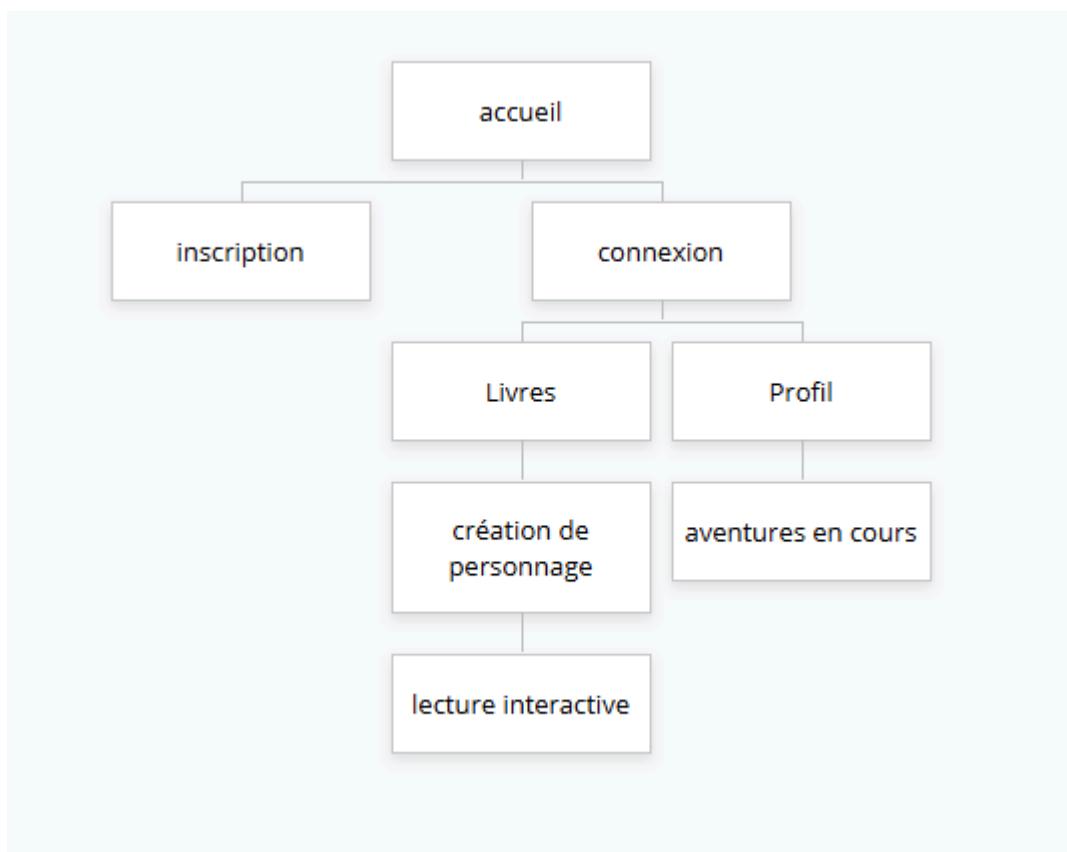


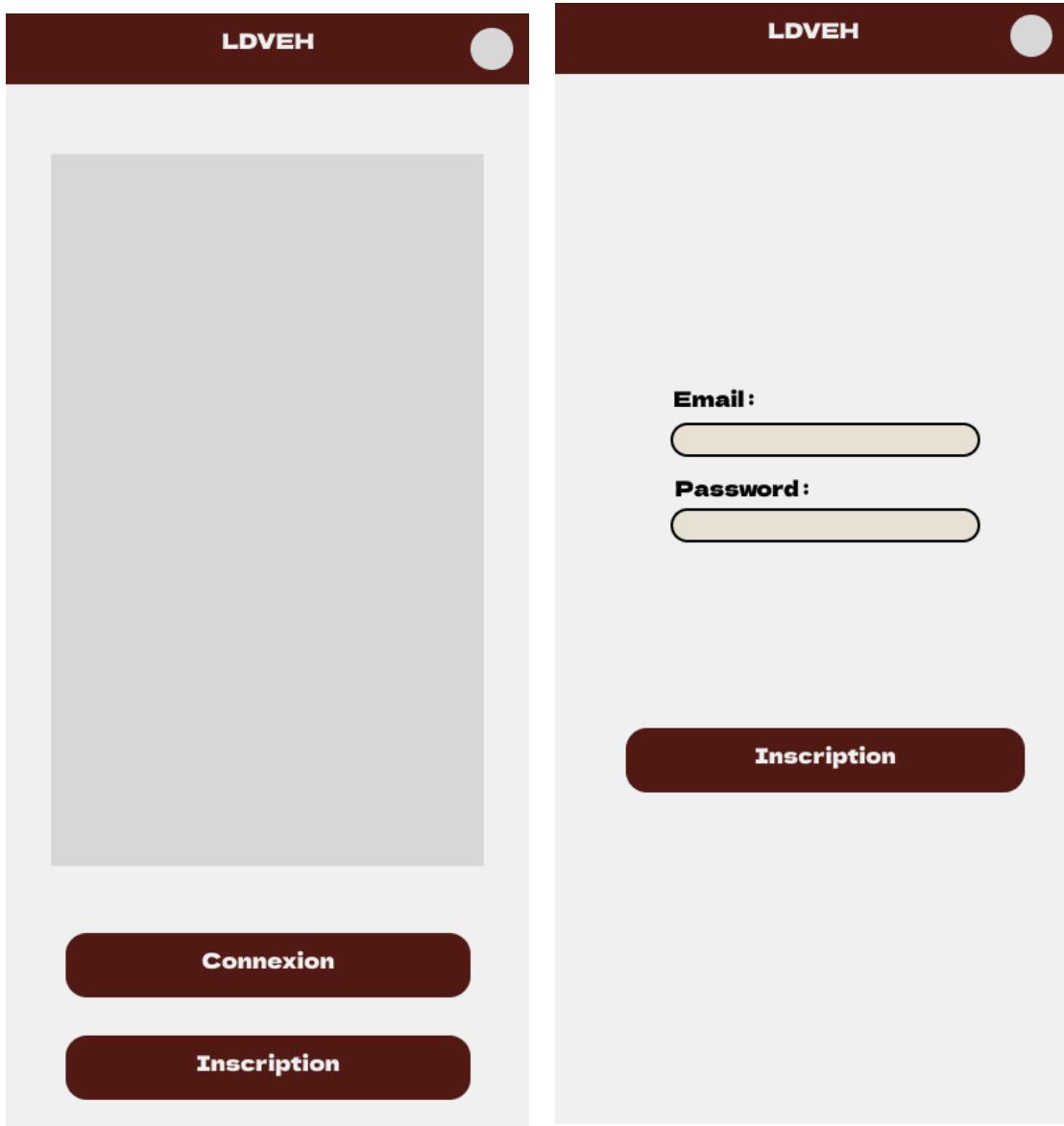
diagramme de séquence UML : FightController



site map :



Maquette :



Livres :

Loup Solitaire - Les Maîtres des Ténèbres

350 pages

auteur

Loup Solitaire - La Traversée Infernale

350 pages

auteur

Loup Solitaire - Les Grottes de Kalte

350 pages

auteur

Loup Solitaire - Gouffre Maudit

350 pages

auteur

Tes Aventuriers :

nom aventurier

Habilité : 18 | Endurance : 17

Loup solitaire - Les Maîtres des Ténèbres - Page 180

nom aventurier

Habilité : 20 | Endurance : 15

Loup solitaire - La Traversée Infernale - Page 132

nom aventurier

Habilité : 14 | Endurance : 30

Loup solitaire - Les Grottes de Kalte - Page 265

Page n° 100

Vous suivez cinq autres flèches qui vous mènent au tronc d'un vieil arbre mort, encore enraciné dans le sol. Vous remarquez alors que le tronc est creux, mais que ce creux se prolonge sous terre à une profondeur que vous ne pouvez évaluer en raison de l'obscurité. Possédez-vous un Anneau de Lumière?
Si oui, rendez-vous au 322.
Sinon, rendez-vous au 120.

Aller au 332**Aller au 120**

Home
Profile
Livres
Aventuriers

autres flèches qui tronc d'un vieil arbre mort, encore enraciné dans le sol. Vous remarquez alors que le tronc est creux, mais que ce creux se prolonge sous terre à une profondeur que vous ne pouvez évaluer en raison de l'obscurité. Possédez-vous un Anneau de Lumière?
Si oui, rendez-vous au 322.
Sinon, rendez-vous au 120.

Aller au 332**Aller au 120**

Matrice de tests - Administration (Back-office)

Administration (Back-office)						
ID Test	Entité	Scénario	Entrée	Résultat attendu	Type de test	OK ?
T701	Accès Admin	Admin connecté accède au BO	ROLE_ADMIN	Dashboard EasyAdmin affiché	UI/API	
T702	Accès Admin	Utilisateur simple tente accès	ROLE_USER	Erreur 403	Sécurité	
T703	Book	Création d'un livre	Titre, pages	Livre créé avec succès	Formulaire	
T704	Book	Données manquantes	Titre vide	Erreur validation formulaire	UI/Form	
T705	Page	Ajout d'une page à un livre	Numéro, contenu	Page ajoutée avec relation OK	Fonctionnel	
T706	Choice	Ajout d'un choix avec redirection	Texte, nextPage	Choix valide, visible côté API	Fonctionnel	
T707	Monster	Modification des stats	HAB, END modifiés	Mise à jour effective	UI/API	
T708	Equipment	Ajout d'un équipement	Nom, bonus, type	Objet ajouté à l'inventaire base	UI/API	
T709	Skill	Suppression d'une compétence	Skill ID	Compétence supprimée sans conflit	Fonctionnel	
T710	User	Visualisation d'un utilisateur	ROLE_ADMIN	Données accessibles mais non modifiables si restrictions	Sécurité/Form	
T711	Validation	Formulaire avec erreur métier	Choix sans redirection	Message d'erreur clair	Formulaire/UI	
T712	Sécurité entité	CRUD restreint à admin	POST/PUT/DELETE	Accès refusé si pas admin	API/UI	

Matrice de tests - Navigation / Aventure

Aventure / Livres-jeux						
ID Test	Fonction	Scénario	Entrée	Résultat attendu	Type de test	OK ?
T201	Création Aventure	Utilisateur connecté, données valides	Book ID, user token	Aventure créée, page départ définie	Fonctionnel/API	
T202	Création Aventure	Utilisateur non authentifié	Aucune auth	Erreur 401	Sécurité/API	
T203	Création Aventure	Book ID invalide	ID non existant	Erreur 404 ou 400	API	
T204	Navigation	Aller à une page valide	Page ID suivant depuis choix	Page atteinte, contenu + choix	Fonctionnel/API	
T205	Navigation	Aller à une page interdite	Page non liée au choix actuel	Erreur 403 ou 400	Métiers/API	
T206	Navigation	Page avec condition non remplie	Choix conditionnel (ex: compétence requise)	Erreur 403 ou pas de choix affiché	Fonctionnel	
T207	Navigation	Choix avec condition remplie	Compétence requise OK	Page suivante atteinte	Fonctionnel/API	
T208	Navigation	Page avec combat	Page contient un combat	Combat déclenché ou simulé	Fonctionnel	
T209	Sauvegarde	En cours d'aventure	Aller à une page => état sauvegardé	Page et stats enregistrés	Intégration/API	
T210	Sauvegarde	Multi-sessions	Reprendre une aventure déjà entamée	Reprise sur bonne page	Fonctionnel/API	
T211	Fin d'aventure	Arriver à une page finale	Page sans suite	Statut aventure : terminé	Fonctionnel/API	
T212	Fin d'aventure	Page de mort / échec	EndingType = death	Aventure marquée comme échouée	Fonctionnel	
T213	Sécurité	Accès aventure d'un autre user	GET aventure ID d'autrui	Erreur 403 ou 404	Sécurité	

Matrice de tests - Aventurier & Joueur

Aventurier / Joueur						
ID Test	Fonction	Scénario	Entrée	Résultat attendu	Type de test	OK ?
T101	Création Aventurier	Données valides	Nom aventurier, user connecté	Aventurier créé, 201, lié à user	Fonctionnel/API	
T102	Création Aventurier	Sans être connecté	Aucun token JWT	Erreur 401	Sécurité/API	
T103	Création Aventurier	Nom manquant	"", token valide	Erreur 400, validation	Unitaire/API	
T104	Association User	Lier user -> aventurier	POST avec token user	L'ID du user est bien enregistré	Intégration	
T105	Association User	Un user crée plusieurs aventuriers	Créations multiples	Comportement conforme à règles (1 seul ? multiple ?)	Fonctionnel/API	
T106	Création Aventurier	Mauvais format des données	Champ inattendu ou mal typé	Erreur 400	API	
T107	Création Aventurier	Récupération après création	GET aventurier du user	Données cohérentes	Fonctionnel/API	
T108	Sécurité	User A tente d'accéder à aventurier de User B	GET aventurier avec autre ID	Erreur 403 ou 404	Sécurité/API	

Matrice de tests - Combats

Combat						
ID Test	Fonction	Scénario	Entrée	Résultat attendu	Type de test	OK ?
T401	Déclenchement combat	Page avec monstre	Page ID avec combat	Combat lancé automatiquement	Fonctionnel/API	OK
T402	Déclenchement combat	Page sans combat	Page ID sans monstre	Aucun combat lancé	Fonctionnel	KO
T403	Calcul dommages	Attaque sans équipement	Stats de base	Dégâts calculés avec HAB	Unitaire/Métier	KO
T404	Calcul dommages	Attaque avec équipement	Arme équipée	Bonus appliqué aux dégâts	Unitaire/Métier	KO
T405	Calcul dommages	Défense avec compétence	Bouclier ou compétence défensive	Réduction des dégâts	Unitaire/Métier	KO
T406	Combat complet	Du début à la fin	Monstre, utilisateur	Combat se termine (victoire ou mort)	Intégration	OK
T407	Résolution victoire	Joueur gagne	Combat gagné	Passage à la page suivante	Fonctionnel/API	KO
T408	Résolution défaite	Joueur meurt	Combat perdu	Aventure terminée, statut "échec"	Fonctionnel/API	KO
T409	Historique combat	Sauvegarde à chaque tour	Tour par tour	Historique complet enregistré	Intégration	KO
T410	Historique visibilité	Récupération historique	GET historique	Liste des actions, dégâts, tours	Fonctionnel/API	KO
T411	Sécurité historique	Accès autre utilisateur	GET d'un autre combat	Erreur 403 ou 404	Sécurité/API	OK
T412	Combat avec conditions	Combat dépendant de stats	HP faible ou malus actif	Combat plus difficile ou désavantage	Métier	KO

Matrice de tests - Monstres

Monstres						
ID Test	Fonction	Scénario	Entrée	Résultat attendu	Type de test	OK ?
T501	Création monstre	Admin, données valides	Nom, HAB, END	Monstre créé, réponse 201	Fonctionnel/API	OK
T502	Création monstre	Utilisateur non-admin	POST monstre	Erreur 403	Sécurité/API	OK
T503	Création monstre	Données invalides	Stats manquantes ou négatives	Erreur 400	Validation/API	OK
T504	Lecture monstre	GET ID existant	ID d'un monstre	Détails du monstre renvoyés	API	OK
T505	Lecture monstre	GET ID inexistant	ID faux	Erreur 404	API	OK
T506	Liste des monstres	GET collection	Aucun filtre	Liste des monstres disponibles	API	OK
T507	Monstre en combat	Affectation à une page	Page avec monstre ID	Monstre bien utilisé dans combat	Intégration	OK
T508	Combat contre monstre	HAB/END correctes	Données en combat	Stats cohérentes, logique OK	Fonctionnel/API	OK
T509	Monstre inconnu en combat	ID monstre supprimé	Page avec mauvais ID	Erreur 500 ou 404, gestion correcte	Robustesse	OK
T510	Modification monstre	Admin modifie stats	PUT/PATCH avec nouvelles valeurs	Changement pris en compte	Fonctionnel/API	OK
T511	Suppression monstre	Admin supprime un monstre utilisé	DELETE monstre lié	Erreur ou cascade générée	Intégrité	OK

Matrice de tests - Pages & Choix

Pages & Choix						
ID Test	Fonction	Scénario	Entrée	Résultat attendu	Type de test	OK ?
T301	Lecture page	Page accessible sans condition	Page ID standard	Contenu + choix affichés	Fonctionnel/API	Vert
T302	Lecture page	Page non accessible (hors aventure)	Page ID invalide	Erreur 403 ou 404	Sécurité/API	Vert
T303	Lecture page	Page de fin (pas de choix)	Page avec ending	Contenu affiché, aucun choix	Fonctionnel	Vert
T304	Lecture page	Page avec choix conditionnel	Page avec choix lié à compétence	Seul le choix valide est affiché	Métier/API	Vert
T305	Choix sans compétence	Pas la compétence requise	Choix caché ou bloqué	Erreur ou invisibilité	Fonctionnel/API	Orange
T306	Choix avec compétence	Possède la compétence	Choix accessible, redirection	Fonctionnel/API	Vert	Orange
T307	Choix multiple	Page avec plusieurs options	Sélectionner un choix	Redirection vers bonne page	Fonctionnel	Vert
T308	Mauvais choix	Page A → Choix vers page C mais on force vers page D	Erreur 403 ou 400	Sécurité/métier	Vert	Orange
T309	Redirection page	Page A → Choix → Page B	Click sur choix	Page B atteinte, état mis à jour	Fonctionnel/API	Vert
T310	Choix et inventaire	Choix nécessitant un objet	Possède ou non l'objet	Comportement adapté (visible ou non)	Fonctionnel	Orange
T311	Suivi aventure	Navigation via choix	Historique enregistré (si implémenté)	Suivi des pages visitées	Intégration	Vert

Matrice de tests - Sécurité

Sécurité						
ID Test	Fonction	Scénario	Entrée	Résultat attendu	Type de test	OK ?
T601	Auth JWT	Appel route protégée sans token	Aucun token	Erreur 401	Sécurité/API	Vert
T602	Auth JWT	Appel route avec token invalide	Token bidon	Erreur 401	Sécurité/API	Vert
T603	Auth JWT	Appel route avec token valide	Token user	Accès autorisé 200	Sécurité/API	Vert
T604	Rôles	Accès admin avec user standard	GET /admin	Erreur 403	Sécurité/API	Vert
T605	Rôles	Accès admin avec user ROLE_ADMIN	GET /admin	Accès OK, affichage dashboard	Sécurité/API	Vert
T606	EasyAdmin CRUD	Création via interface admin	ROLE_ADMIN connecté	Entité créée avec succès	Fonctionnel	Vert
T607	EasyAdmin CRUD	Accès à CRUD avec rôle USER	ROLE_USER connecté	Erreur 403	Sécurité/UI	Vert
T608	API & Rôles	Suppression entité restreinte	DELETE par non-admin	Erreur 403	API	Vert
T609	Token expiré	JWT expiré	Appel API	Erreur 401	Sécurité/API	Vert
T610	Token renouvelé	Nouveau token après login	Auth flow complet	Nouveau token OK	Authentification	Vert

Matrice de tests - Authentification & Utilisateur

Utilisateur / Authentification						
ID Test	Fonction	Scénario	Entrée	Résultat attendu	Type de test	OK ?
T001	Inscription	Données valides	Email, password	Compte créé, réponse 201	Fonctionnel/API	
T002	Inscription	Email déjà utilisé	Email existant, password	Erreur 409 ou 400	Fonctionnel/API	
T003	Inscription	Email invalide	"invalid mail", password	Erreur 400, validation	Unitaire/API	
T004	Inscription	Mot de passe trop court	Email, "123"	Erreur 400, validation	Unitaire/API	
T005	Inscription	Champs manquants	Aucun ou partiel	Erreur 400	API	
T006	Connexion	Email & mot de passe valides	Email correct, password correct	Token JWT + 200	Fonctionnel/API	
T007	Connexion	Mauvais mot de passe	Email correct, password faux	Erreur 401	Fonctionnel/API	
T008	Connexion	Email inexistant	Email inconnu, password	Erreur 401	Fonctionnel/API	
T009	Connexion	Email vide	"", password	Erreur 400	API	
T010	Connexion	Mot de passe vide	Email, ""	Erreur 400	API	
T011	Sécurité	Accès à une route protégée sans token	Aucun token	Erreur 401	Sécurité/API	
T012	Sécurité	Accès à une route protégée avec token invalide	Token bidon	Erreur 401	Sécurité/API	
T013	Sécurité	Accès à une route protégée avec token valide	Token JWT valide	Accès autorisé 200	Sécurité/API	

screen App mobile

