

DOSSIER PROJET - MEMOIRE

Présenté le 17 Juin 2025 par

M. Cyril CATOEN

à

La Piscine,
208 avenue de la Marne
33700 Mérignac



Pathfinding – Adventures & Safety

Pour obtenir le titre
Développeur Web et Web Mobile (DWWM)

Titre RNCP niveau V

Tuteur pédagogique : M. Mathieu ROZAND

REMERCIEMENTS

Ces cinq mois de formation ont été une véritable source d'enrichissement personnel et professionnel. C'est pourquoi, je tiens à remercier mon tuteur pédagogique Mathieu ROZAND pour son soutien et ses conseils avisés tout au long de la réalisation de mon projet.

Je tiens également à remercier David ROBERT, formateur PHP/Symfony, Romain MAZIERE, formateur SQL pour la qualité des cours prodigues et la pédagogie dont ils ont su faire preuve pour expliciter des notions parfois abstraites ou complexes à appréhender.

Plus généralement je tiens à remercier toutes les personnes qui ont apporté leur contribution à ce projet et qui m'ont épaulé dans sa réalisation ainsi que l'ensemble des collaborateurs de l'organisme de formation La Piscine dont la bonne humeur et la sympathie m'a permis de réaliser mon apprentissage dans les meilleures conditions possibles.

J'aimerais également remercier Araya, Alexandre et Fred pour nos parties régulières de Colonist à midi, moments précieux qui m'ont permis de me détendre et de prendre du recul durant cette période intense.

Enfin, je tiens à faire un aparté pour dire merci à Yohann DUBUIS pour son soutien moral et amical tout au long du projet.

INTRODUCTION – COMPETENCES MISES EN ŒUVRE.....	2
1. PRESENTATION GENERALE DU PROJET	3
1.1. CONTEXTE ET ANALYSE DU BESOIN	3
1.1.1. <i>Résumé du projet</i>	3
1.1.2. <i>Expression fonctionnelle des besoins</i>	4
1.1.3. <i>Cadre de réponse</i>	6
1.1.4. <i>Gestion de projet : Kanban et Gantt</i>	6
1.1.5. <i>Arborescence du site.....</i>	8
1.1.6. <i>Environnement de travail et architecture technique</i>	8
1.2. DESIGN ET EXPERIENCE UTILISATEUR.....	11
1.2.1. <i>Moodboard et création du logo</i>	11
1.2.2. <i>Charte graphique appliquée</i>	12
1.2.3. <i>Zoning, maquettes et responsive design.....</i>	13
1.3. DEFINITION DES CONCEPTS TECHNIQUES	15
1.3.1. <i>Modèle MVC dans une application Web.....</i>	16
1.3.2. <i>Communication client-serveur</i>	17
1.3.3. <i>Présentation des tests unitaires et fonctionnels</i>	18
2. PRESENTATION SPECIFIQUE DU PROJET.....	19
2.1. DEVELOPPEMENT BACK-END	19
2.1.1. <i>Conception et gestion de la base de données relationnelle</i>	19
2.1.2. <i>Modélisation des entités et gestion des contraintes de validation.....</i>	20
2.1.3. <i>Routes et API.....</i>	23
2.1.4. <i>Contrôleurs et composants métiers côté serveur.....</i>	24
2.1.5. <i>Sécurité appliquée : authentification, rôles et autorisations</i>	28
2.2. DEVELOPPEMENT FRONT-END.....	31
2.2.1. <i>Intégration des interfaces utilisateur (HTML/Twig).....</i>	31
2.2.2. <i>Développement de la partie dynamique (Javascript, AJAX).....</i>	33
2.2.3. <i>Gestion des droits et visibilité des données utilisateur</i>	36
2.2.4. <i>CSS/SCSS avancé et Responsive Design.....</i>	37
2.2.5. <i>Emploi de librairies externes</i>	43
2.3. DOCUMENTATION ET DEPLOIEMENT.....	44
2.3.1. <i>Procédure de déploiement (prévisionnelle).....</i>	45
2.3.2. <i>Recherche et exploitation de ressources techniques anglophones</i>	45
2.4. BILAN ET PERSPECTIVES D'EVOLUTION.....	47
2.4.1. <i>Axes d'amélioration</i>	48
2.4.2. <i>Perspectives d'évolution du projet</i>	49
2.4.3. <i>Bilan global</i>	50

INDEX DES FIGURES	52
INDEX DES TABLEAUX	53
RESSOURCES UTILISEES.....	54
ANNEXES	56

INTRODUCTION – COMPETENCES MISES EN ŒUVRE

De nos jours, la pratique des activités extérieures telles que la randonnée, l’itinérance à vélo ou encore l’escalade rencontre un intérêt croissant. Cependant, la sécurité des aventuriers reste une préoccupation majeure, particulièrement dans des zones isolées où la couverture réseau est souvent insuffisante voire inexistante. Afin d’améliorer la sécurité des pratiquants tout en facilitant la communication et la gestion des itinéraires, le projet Pathfinding propose une application web complète, dédiée à la planification, à terme, au suivi GPS, et à la gestion des alertes de sécurité en temps réel.

Dans le cadre de ma formation de Développeur Web et Web Mobile (RNCP niveau 5), le projet Pathfinding a nécessité la mise en œuvre d'un ensemble complet de compétences techniques et méthodologiques. Le développement du projet a débuté **par l'installation et la configuration d'un environnement de travail adapté**, intégrant notamment Symfony pour le développement back-end, Javascript et plusieurs libraires externes pour l’interface utilisateur dynamique, et MySQL pour la gestion de la base de données.

La conception des interfaces utilisateur, réalisée initialement via des maquettes sur Figma, a permis de développer une interface web ergonomique, responsive et intuitive. Cette étape a ensuite été enrichie par le **développement d’interfaces dynamiques**, intégrant notamment des interactions complexes avec l’utilisateur et des éléments temps réel communiquant avec la base de données.

Côté serveur, j’ai conçu et **implémenté une base de données relationnelle**. J’ai également **développé les composants métier et d'accès aux données SQL** nécessaires à l’application, tout en mettant en œuvre plusieurs couches de sécurité pour garantir la fiabilité et la confidentialité des informations traitées.

Enfin, **la documentation du processus de déploiement de l’application a été rédigée** afin d’assurer une transition fluide vers une mise en production opérationnelle future.

Ce mémoire présente dans une première partie le contexte et les enjeux liés au projet Pathfinding, incluant l’expression fonctionnelle des besoins, l’organisation du travail, l’arborescence du site, ainsi que l’environnement technique mis en place. Cette partie décrit également les choix réalisés en matière de design, de maquettes et d’expérience utilisateur, tout en précisant les concepts techniques clés mobilisés (modèle MVC, tests, interactions client-serveur). La seconde partie, quant à elle, détaille spécifiquement les réalisations techniques effectuées durant le projet. Elle aborde d’abord le développement back-end, en traitant notamment de la base de données relationnelle, de la gestion de l’authentification et de la sécurité. Ensuite, elle présente le développement front-end, particulièrement les interfaces dynamiques et adaptatives mises en place. Enfin, cette partie conclut sur la documentation du déploiement, les perspectives d’amélioration futures, ainsi qu’un bilan global du projet.

1. PRESENTATION GENERALE DU PROJET

1.1. Contexte et analyse du besoin

1.1.1. Résumé du projet

Idée initiale : Mon souhait est de créer une plateforme qui permet aux aventuriers de partager leur itinéraire, de rassurer leurs proches et d'améliorer leur sécurité en cas d'imprévu, tout en favorisant les rencontres et l'entraide entre passionnés d'exploration / activités outdoor.

Résumé du projet :

Pathfinding est une plateforme web et mobile dédiée à la **planification, au suivi et au partage d'aventures en pleine nature**, intégrant des outils de **sécurité intelligente** et à terme de **gestion communautaire**.

Conçue pour randonneurs, trekkeurs, campeurs ou voyageurs solitaires, elle permet de créer une aventure, définir un itinéraire (manuel ou GPS/GPX), l'associer à des fichiers (cartes, photos, récits, matériel), le partager avec des proches, et activer une **alerte automatique** en cas d'absence de signal cellulaire.

Au stade **MVP (Minimal Viable Product)**, l'**alerte automatique** permet à l'aventurier de définir ses horaires de départ et retour. S'il ne confirme pas son retour après un délai choisi, un email est envoyé à ses contacts de confiance avec ses infos clés (zone, itinéraire, etc.).

La plateforme participe ainsi à **renforcer la sécurité** des aventuriers isolés.

Pathfinding a pour ambition d'être adopté internationalement comme la plateforme de référence des activités outdoor. Pour cela, plusieurs axes d'évolution sont étudiés :

- **PathTracking** : une application mobile dédiée à la collecte en temps réel de points GPS et à leur transmission à l'aventure en cours sur la plateforme Pathfinding.
- **SafeTracker** : une évolution de l'application mobile sous forme de "boîte noire" indépendante avec une meilleure autonomie, précision et connectivité pour le transfert des coordonnées GPS et d'alertes en temps réel (clic sur un bouton par l'utilisateur, réseau satellite si cellulaire indisponible).
- **Adventures By Pathfinding** : une sélection d'aventures dans des lieux isolés (déserts, montagnes, jungles, toundras...) accompagnés par notre réseau de Guide Pro certifiés.
- **Plus** : marketplace, blog communautaire, système de followers/amis, aventures en groupe, de nombreuses idées à développer au fur et à mesure que la communauté croît !

1.1.2. Expression fonctionnelle des besoins

Afin de répondre aux différents usages et niveaux d'accès attendus par le projet Pathfinding, une gestion des rôles a été mise en place, distinguant les droits selon les profils d'utilisateurs. Ces rôles ont été répartis entre deux grands contextes :

- Le **back-office**, réservé à l'équipe d'administration technique ou fonctionnelle, avec deux niveaux de priviléges : **SuperAdmin** et **Admin**.
- Le **front-office**, dédié à l'expérience utilisateur, avec trois types d'accès : **User (ou Aventurier)**, **Guest (ou Visiteur non connecté)** et **Admin** lorsqu'il interagit comme utilisateur.

Les tableaux ci-dessous synthétisent les fonctionnalités principales accessibles par chaque rôle, en distinguant les actions de consultation, création, modification, suppression et partage de contenus. Cette modélisation permet d'assurer la cohérence des permissions, la sécurité des données et une expérience utilisateur adaptée à chaque profil.

Catégorie	Fonctionnalité	SuperAdmin	Admin
Gestion des utilisateurs	Comptes ADMIN : Créer / Modifier / Suspendre* / Supprimer	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	Comptes USER : Créer / Modifier / Suspendre* / Supprimer	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Gestion des aventures	Voir / Supprimer	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Gestion de la BDD*	DDL : structure de la BDD CREATE, TRUNCATE, ALTER, DROP	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	DCL : gestion des droits d'accès	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	DML : insertion / mise à jour / suppression de données	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	DQL : consultation / requêtes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	TCL : transactions (COMMIT, ROLLBACK)	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Administration du site*	Gestion des messages de contact	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	Gestion du contenu visible sur le site (news, articles, etc.)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Tableau 1 : Fonctionnalités principales Back-office accessible par rôle administratif

*Fonctionnalités futures / Non disponible au stade de la formation

Catégorie	Fonctionnalité	All Admin	User (Aventurier)	Guest (Visiteur)
Accès aux contenus	Voir tout contenu	✓	✗	✗
	Voir ses contenus	✓	✓	✗
	Voir les contenus publics			
	Voir les contenus privés autorisés (liste)	✓	✓	✗
Publications	Voir les aventures privées par lien partagé	✓	✓	✓
	Publier une aventure / récit	✗	✓	✗
	Paramétriser une alerte / timer	✗	✓	✗
Partage	Publier un article* visible par tous	✓	✗	✗
	Partager une aventure	✗	✓	✗
Édition / Suppression	Modifier ses contenus	✓	✓	✗
	Supprimer ses contenus			
	Modérer ou supprimer le contenu d'autres utilisateurs**	✓	✗	✗
Contacts de sécurité	Créer / modifier / supprimer ses contacts de confiance (Safety Contact)	✗	✓	✗
	Gérer ses listes de contacts (Safety List)	✗	✓	✗

Tableau 2 : Fonctionnalités principales Front-office accessibles par rôle

*Fonctionnalités futures / Non disponible au stade de la formation

**Hors contenu lié à la sécurité de l'aventurier : Adventures sous status Ongoing / SafetyContacts / TimerAlert

1.1.3. Cadre de réponse

Pour répondre aux besoins fonctionnels identifiés, l'application Pathfinding s'appuie sur une architecture basée sur le framework Symfony, permettant un découpage clair entre les couches de traitement, de données et de présentation.

Le système de rôles (SuperAdmin, Admin, User, Guest) a été défini de manière à refléter fidèlement les différents niveaux d'accès aux fonctionnalités, en assurant à la fois la sécurité des données et la simplicité d'utilisation.

Les fonctionnalités attendues sont mises en œuvre selon une logique modulaire, avec une gestion fine des permissions appliquée au niveau des contrôleurs et des vues. Symfony offre pour cela des outils robustes tels que les annotations de sécurité, les voters ou encore le système d'authentification par rôles. Cette organisation permet de restreindre l'accès à certaines routes ou actions selon le profil connecté, tout en facilitant les évolutions futures (ajout de nouveaux rôles, permissions spécifiques, etc.).

L'ensemble des droits et périmètres fonctionnels a été synthétisé dans les tableaux ci-dessus, qui font office de référentiel pour le développement de l'interface utilisateur comme pour la sécurisation des traitements côté serveur.

Afin de planifier et structurer la mise en œuvre progressive de ces fonctionnalités, une organisation par tâches a été mise en place à l'aide de l'outil Trello.

1.1.4. Gestion de projet : Kanban et Gantt

Pour organiser et suivre les différentes étapes du développement de Pathfinding, j'ai utilisé **Trello**, un outil de gestion de projet visuel en ligne **fondé sur le principe des tableaux Kanban**.

Le tableau principal est divisé en trois colonnes : À faire, En cours et Terminé, permettant de visualiser l'avancement des tâches de manière claire et fluide. Chaque tâche est représentée sous forme de carte, associée à une étiquette (code, design, admin, base de données, etc.) et peut être déplacée d'une colonne à l'autre selon son statut.

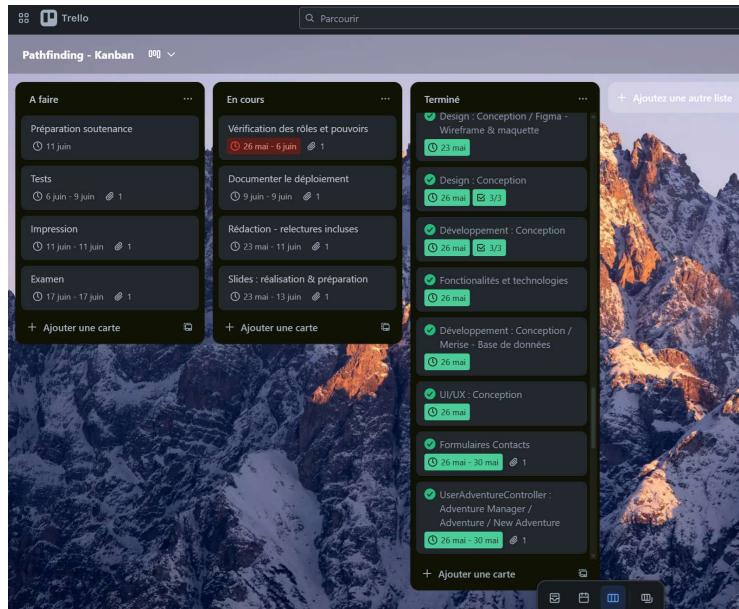


Figure 1 : Tableau Kanban réalisé avec l'outil en ligne Trello

Certaines tâches ont été détaillées à l'intérieur des cartes sous forme de checklists, afin de les rediviser en sous-tâches concrètes, ce qui a facilité leur planification et leur exécution.

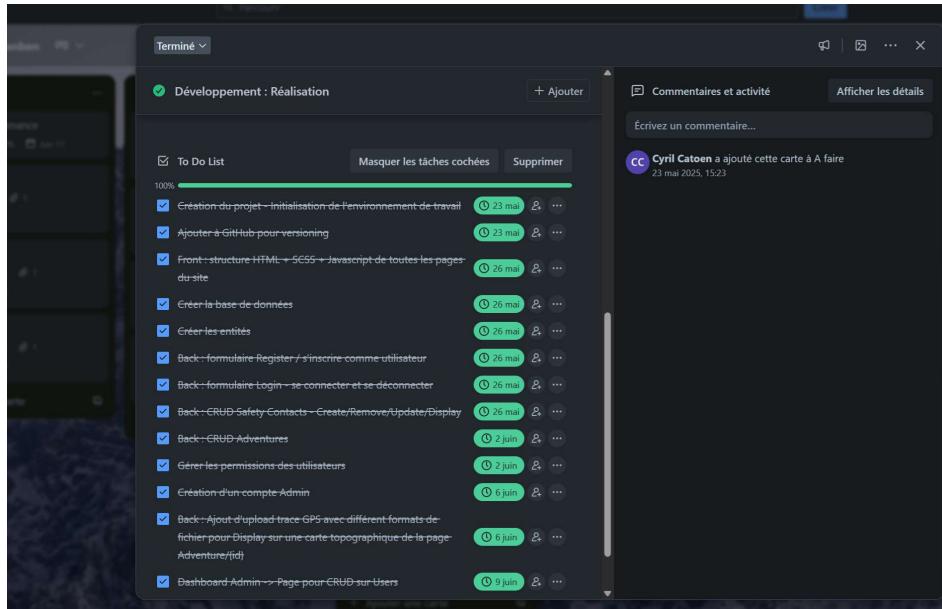


Figure 2 : Exemple de checklist réalisé avec Trello sur une tâche spécifique

En complément, j'ai utilisé le power-up Trello « TeamGantt » pour générer un diagramme de Gantt automatique, basé sur les échéances des cartes, afin d'avoir une vision temporelle de l'ensemble du projet et mieux répartir les charges de travail dans le temps.

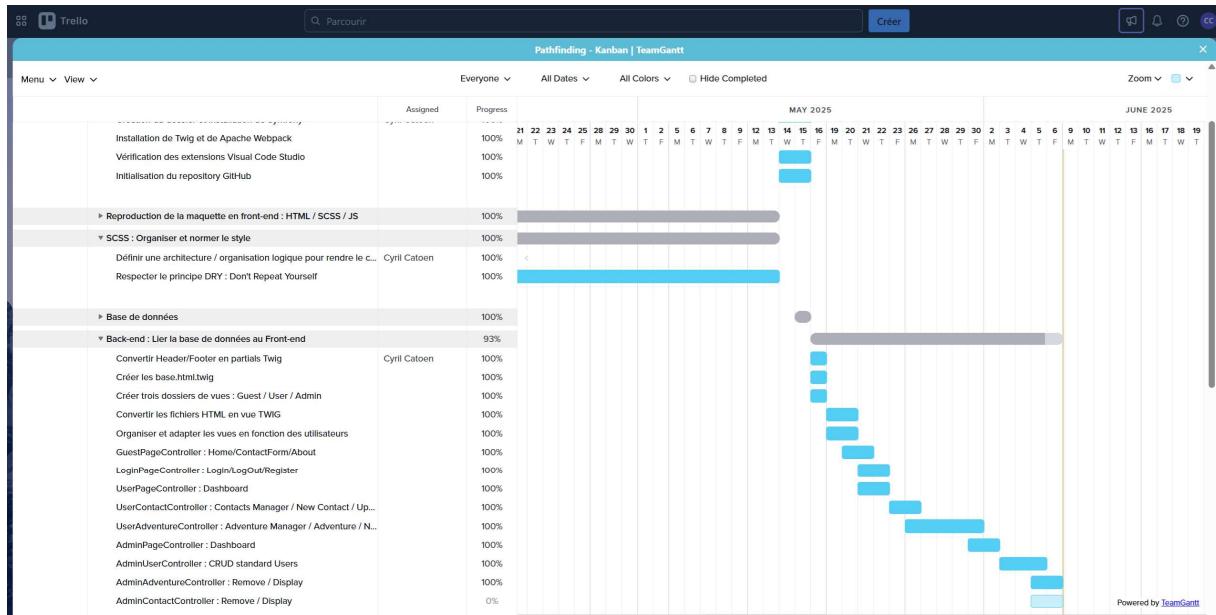


Figure 3 : Diagramme de Gantt réalisé avec le power-up de Trello pour Pathfinding

La planification des tâches ayant été établie, il a ensuite été nécessaire de structurer l'architecture fonctionnelle du site à travers une arborescence claire et cohérente.

1.1.5. Arborescence du site

L'arborescence du site Pathfinding a été conçue pour structurer clairement l'accès aux différentes fonctionnalités selon le type d'utilisateur. Elle distingue les pages accessibles au visiteur (Guest), à l'utilisateur connecté (User/Aventurier) et à l'administrateur (Admin), en s'appuyant sur une hiérarchie fonctionnelle logique et progressive.

L'objectif de cette organisation est de garantir une navigation intuitive tout en respectant les règles de sécurité et d'accessibilité liées aux différents rôles définis précédemment. L'arborescence ci-dessous reflète l'état du projet à la fin de sa phase de prototypage fonctionnel.

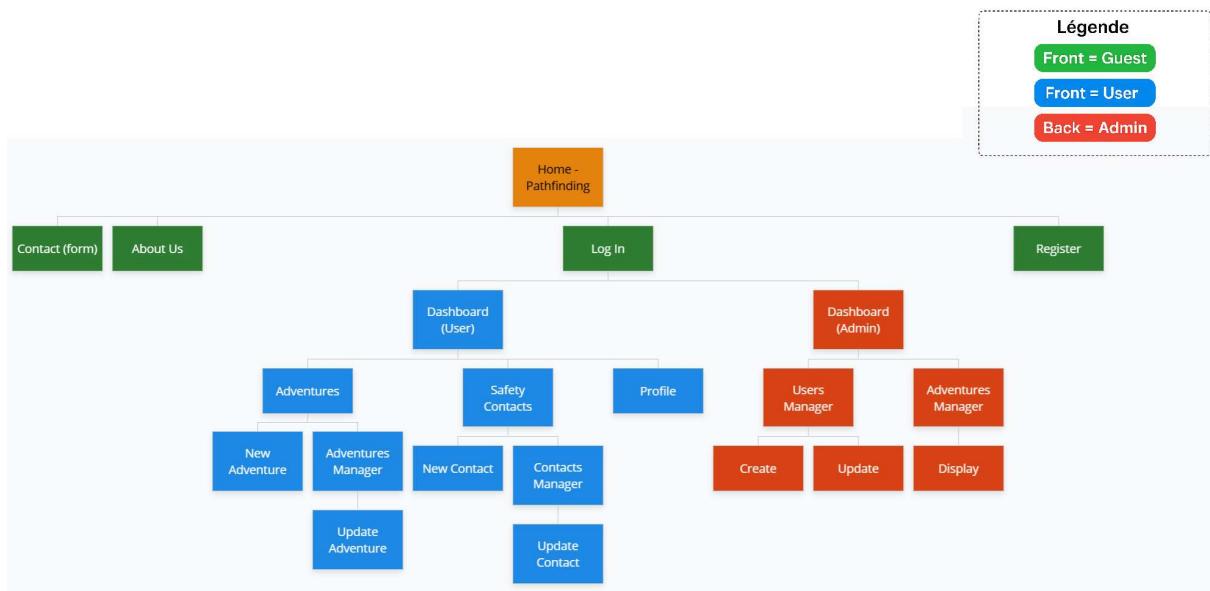


Figure 4 : Arborescence du site web Pathfinding au stade prototype (projet)

Cette structuration du site s'appuie logiquement sur un environnement technique cohérent et une architecture logicielle adaptée aux besoins du projet. C'est ce que détaille la section suivante.

1.1.6. Environnement de travail et architecture technique

Le développement de l'application Pathfinding repose sur une architecture web classique dite client-serveur, dans laquelle le front-end (client) dialogue avec le back-end (serveur) à travers le réseau.

Côté client / Front-end, l'interface utilisateur repose sur trois langages fondamentaux du web : **HTML5** pour la structure, **CSS3** (et son préprocesseur **SASS**) pour le style et la mise en page responsive, ainsi que **JavaScript** pour les interactions dynamiques (carrousels, modales, mise à jour asynchrone des contenus via `fetch()` ou `AJAX`).



L'utilisation du préprocesseur Dart Sass (v1.88.0 / dart2js 3.7.3) a permis de structurer proprement les feuilles de style tout en tirant parti des fonctionnalités avancées comme l'imbrication, les variables, mixins ou conditions.



Côté serveur, la logique métier est assurée par PHP 8.3.1, couplé au framework Symfony 7.2.6. Symfony a permis d'organiser le projet selon le modèle MVC (Modèle – Vue – Contrôleur), en facilitant la gestion des routes, des droits d'accès et des entités métiers.



Le moteur de **template Twig (v3.21)** a été utilisé pour le rendu des vues côté front, tandis que la couche de persistance repose sur **Doctrine ORM (v3.3.3)** pour gérer les interactions entre objets PHP et **base de données relationnelle MySQL (v5.7.24)**.



Enfin, le **serveur local** a été configuré à l'aide d'**Apache 2.4.33 via MAMP**, afin d'assurer un environnement de développement stable et reproductible.

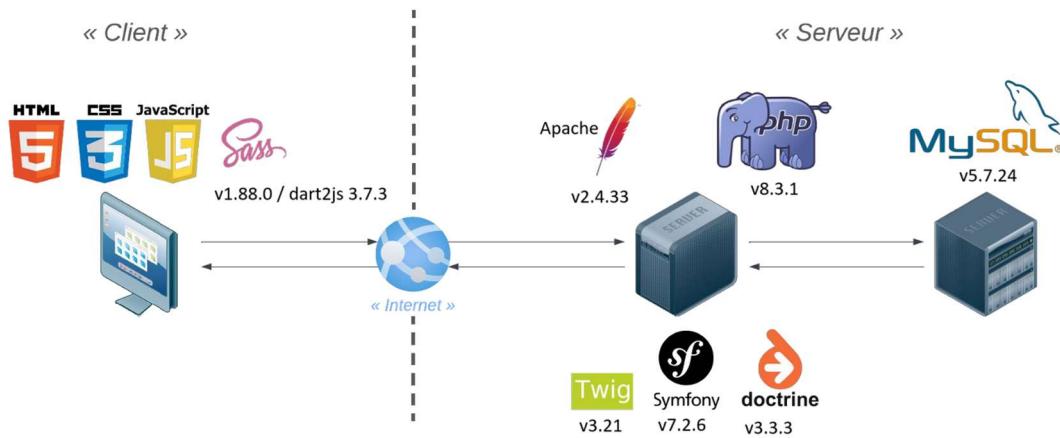
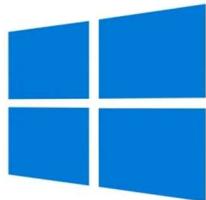
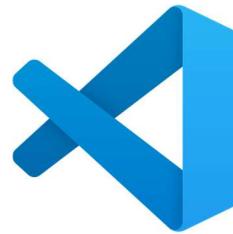


Figure 5 : Synthèse de l'architecture technique du projet Pathfinding

En complément de cette architecture technique, plusieurs outils ont été mobilisés pour assurer une productivité optimale tout au long du projet. Le développement s'est fait sous **Windows 11**, en local via **MAMP** pour l'émulation du **serveur Apache** et la gestion de la base de données MySQL.



L'éditeur de code principal utilisé est **Visual Studio Code**, enrichi de plusieurs extensions utiles telles que *Prettier*, *Emmet*, *Live Server*, ou encore *Live Sass Compiler* pour améliorer le confort de développement front-end.



Pour la conception graphique des interfaces, la réalisation des maquettes et la structuration des composants visuels, j'ai utilisé **Figma**. L'édition d'éléments visuels ponctuels (icônes, images, retouches) a été faite avec **Gimp**.



Enfin, la gestion de version du code source s'est appuyée sur **Git**, avec un hébergement des dépôts sur **GitHub**, permettant un suivi rigoureux des évolutions du projet. La rédaction du dossier projet, la documentation utilisateur et la présentation orale ont été réalisées avec la **suite Microsoft Office (Word et PowerPoint)**.



Une fois l'environnement technique en place, l'étape suivante a consisté à concevoir une interface utilisateur claire, accessible et responsive. La section suivante présente donc les choix réalisés en matière de design graphique, d'ergonomie et d'expérience utilisateur.

1.2. Design et expérience utilisateur

La conception de l'interface utilisateur de Pathfinding ne s'est pas limitée à une simple recherche d'esthétique : elle devait à la fois **inspirer la confiance (sentiment de sécurité)**, **inviter à l'exploration**, et **guider l'utilisateur dans son parcours** sans jamais le perdre. L'expérience utilisateur a donc été pensée dès les premières étapes du projet à travers un travail de **direction artistique cohérente**, alliant **identité visuelle, hiérarchie d'information, et lisibilité**.

Cette section revient sur les différentes étapes de cette démarche : de la création du **moodboard** jusqu'à la **maquette finale**, en passant par la définition d'une **charte graphique fonctionnelle**, le **zoning**, et l'adaptation **responsive** de l'interface.

Les éléments graphiques présentés ici sont issus de mes travaux personnels mais également inspirés de projets professionnels existants (comme AllTrails ou Lynkx), pour nourrir une interface claire, intuitive, et pleinement centrée sur les usages du terrain.

1.2.1. Moodboard et création du logo

Afin de donner une direction graphique claire au projet Pathfinding, j'ai commencé par concevoir un **moodboard visuel** réunissant des éléments inspirants liés à l'univers du voyage, de la nature et de la randonnée.

Ce moodboard (cf. *Figure 6*) m'a permis de définir les **valeurs esthétiques et émotionnelles** du projet : **exploration, minimalisme, lisibilité** et **contraste naturel**, à travers des visuels de sentiers, de cartes, de boussoles et d'UI outdoor modernes.

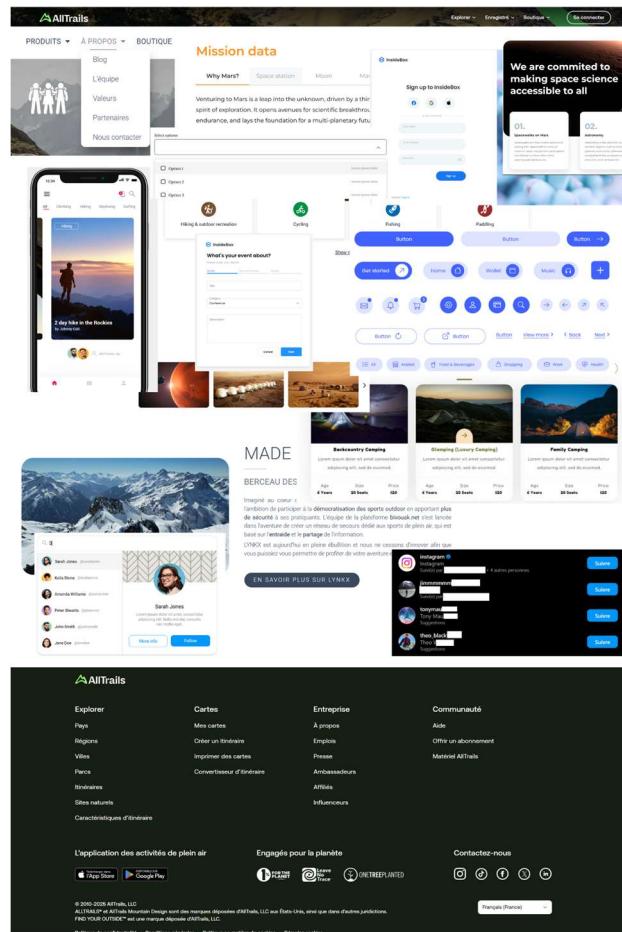


Figure 6 : Moodboard de Pathfinding (version A4 en annexe)

Le **logo de Pathfinding** a été décliné dans la continuité de cette ambiance. Il représente un **chemin stylisé formant une montagne**, à la fois symbole d'itinéraire, d'élévation et de cap. Le tracé épuré permet une **lecture immédiate** et une **utilisation souple** sur fond clair ou foncé, en version icône ou titre. Il a été conçu sous Figma, en gardant une logique responsive.



Pathfinding - Logo

Figure 7 : Logo de Pathfinding – version verte

1.2.2. Charte graphique appliquée

La **charte graphique** de Pathfinding repose sur une palette contrastée, à la fois naturelle et structurante :

- Vert forêt #2E7D32 : **ancrage dans la nature**
- Bleu France #1E88E5 : **repère visuel et appel à l'action**
- Beige lin #F6EDE5 et blanc pur #FFFFFF : **fond neutre et lisible**
- Orange terre brûlée #D84315 : **signal fort (alerte, statut)**
- Gris anthracite #263238 et noir #000000 : **typographie et structure**



Figure 8 : Palette de couleurs du projet réalisé avec Coolors

L'**outil en ligne Coolors** a été utilisé pour vérifier le contraste (accessibilité) entre le texte et le fond, ainsi on utilise des polices d'écritures blanches sur les fonds bleus, rouge alerte, gris anthracite et noir, et des polices d'écritures gris anthracite/noir sur les autres.

Pour la **typographie**, j'ai opté pour **Times New Roman**, une police intemporelle, robuste et disponible sur tous les navigateurs. Elle évoque **la solidité, la lisibilité** et reste cohérente avec le ton professionnel du projet. Les titres sont hiérarchisés de H1 à H4, avec un corps de texte à 16px (1rem), selon une grille fluide.

H1 Lorem Ipsum 40px

H2 Lorem Ipsum

H3 Lorem Ipsum

H4 Lorem Ipsum

p, span, label, input, a lorem ipsum 16px - 1rem

Font : Times New Roman

Figure 9 : Typographie simple et lisible

Cette charte assure une **identité forte**, tout en respectant les critères d'accessibilité et de cohérence responsive.

1.2.3. Zoning, maquettes et responsive design

Le processus de maquettage s'est déroulé en **trois étapes** :

a) Zoning

Première étape structurante, le zoning m'a permis de **visualiser l'organisation fonctionnelle** des différentes pages du site sans entrer dans le détail graphique. Chaque zone est pensée pour guider l'utilisateur : zones de recherche, blocs d'informations, navigation principale, etc.

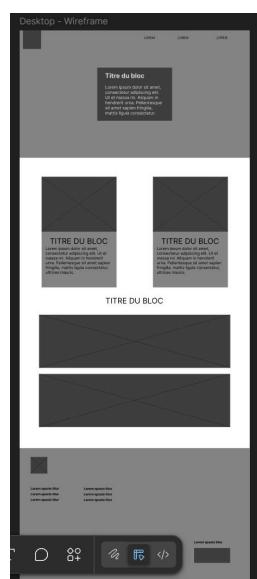


Figure 10 : Zoning de la page d'accueil version ordinateur/bureau

b) Wireframe

À partir du zoning, j'ai réalisé des **wireframes** plus précis, distinguant les composants de l'interface. Ils intègrent les logiques d'affichage desktop et mobile, en tenant compte des bonnes pratiques UX/UI (navigation fixe, blocs responsives, affichage conditionnel des boutons...).



Figure 11 : Zoning de la page d'accueil version ordinateur/bureau

c) Maquette finale

La dernière phase a consisté à créer la **maquette graphique complète**, en intégrant la charte (typographies, couleurs, logo). Les écrans réalisés concernent notamment :

- La page d'accueil visiteur
- La page d'accueil aventurier
- La page d'une aventure
- La page de contact de confiance

Chaque page a été déclinée en **version desktop et mobile**, pour garantir un affichage adaptatif fluide, optimisé par l'usage de media queries en SASS/SCSS.

L'exemple de la maquette de la page Aventure est présentée à la page suivante.

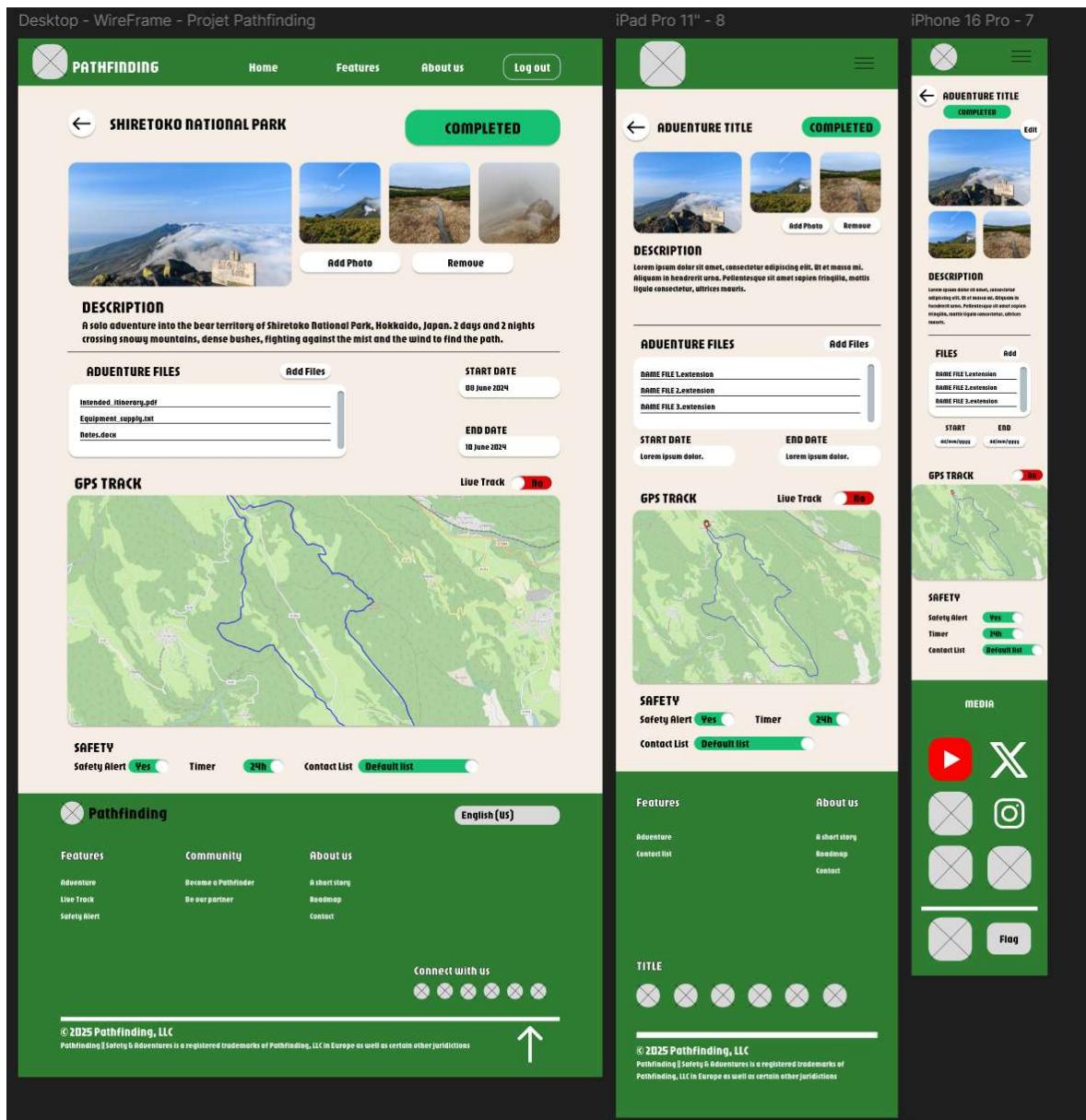


Figure 12 : Maquette de la page Aventure en version Web et Web Mobile (Responsive)

L'identité visuelle de Pathfinding posée et l'interface utilisateur progressivement structurée, le développement du projet a nécessité la mobilisation de plusieurs concepts fondamentaux du développement web, à la fois côté client et côté serveur. Afin de mieux comprendre les choix techniques effectués et les mécanismes de fonctionnement du projet, il est essentiel de revenir sur certains principes clés.

1.3. Définition des concepts techniques

Avant d'aborder les étapes concrètes du développement de Pathfinding, il est utile de clarifier les principaux concepts architecturaux et techniques qui ont servi de fondation au projet. Le premier d'entre eux est le modèle MVC, largement utilisé dans les frameworks modernes tels que Symfony.

1.3.1. Modèle MVC dans une application Web

Le modèle **MVC** (Modèle-Vue-Contrôleur) constitue l'architecture logicielle centrale adoptée pour le développement de Pathfinding, en particulier via le framework Symfony. Il repose sur une **séparation logique des responsabilités**, facilitant la maintenance, l'évolutivité et la lisibilité du code.

Cette organisation a permis de structurer efficacement l'application autour de trois couches distinctes mais interconnectées :

- **Le Modèle** regroupe les **entités Doctrine** et la **logique métier**, telles que la création d'une aventure, la gestion des statuts ou le déclenchement d'alertes de sécurité. C'est là que sont définies les règles métiers, les validations et les relations entre les objets manipulés.
- **La Vue**, construite principalement en **Twig** et **stylisée en SCSS**, permet d'afficher dynamiquement les informations à l'utilisateur. Elle rend visibles les aventures, les photos, ou encore les comptes à rebours d'alerte, avec un affichage adaptatif pensé pour les écrans mobiles comme pour le bureau.
- **Le Contrôleur**, enfin, fait le lien entre les requêtes entrantes et les traitements à exécuter. Il capte les actions de l'utilisateur (comme "ajouter un contact", "modifier une aventure" ou "changer la visibilité") et appelle les services ou entités concernées avant de retourner une réponse adaptée.

Cette structure a constitué une **colonne vertébrale fiable** tout au long du développement, notamment pour **isoler les traitements sensibles côté serveur** et **garantir la cohérence des échanges entre l'interface et les données**.

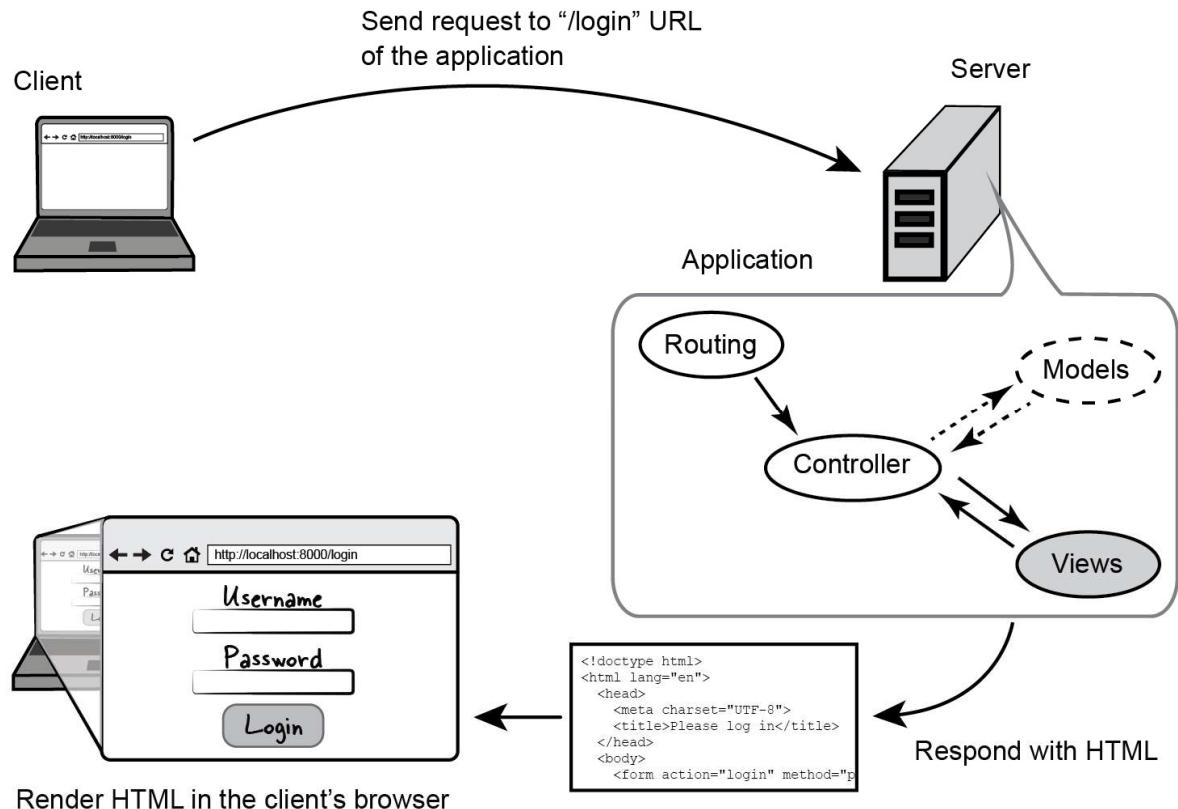


Figure 13 : Schéma illustrant le fonctionnement du modèle MVC pour afficher la page login

1.3.2. Communication client-serveur

Le **projet Pathfinding** repose sur une architecture **client-serveur** classique, dans laquelle l'interface utilisateur (front-end) communique avec le serveur (back-end) pour obtenir, enregistrer ou modifier des données. Chaque action entreprise par l'utilisateur — comme remplir un formulaire, visualiser une aventure ou uploader une photo — déclenche **une requête adressée au serveur suivant le protocole HTTP** (HyperText Transfer Protocol). Ce dernier définit comment les messages sont structurés, envoyés, et reçus.

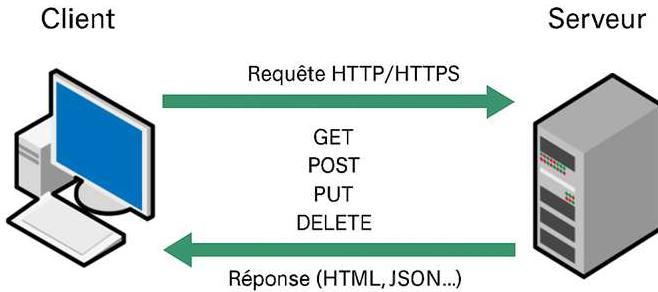


Figure 14 : Cycle requête-réponse entre client et serveur

Pour sécuriser ces échanges, notamment les données personnelles (identifiants, alertes, aventures privées, coordonnées GPS), on utilise HTTPS, version chiffrée de HTTP. Ce protocole repose sur SSL, remplacé aujourd'hui par TLS (Transport Layer Security), garantissant la confidentialité et l'intégrité des informations échangées.

Ces échanges sont orchestrés via le système de **routes** de Symfony : une URL (par exemple /user/adventure/5) est associée, par routage, à un **contrôleur**, qui traite la demande, interroge la base de données, puis retourne une **réponse**. Selon les cas, cette réponse peut être un **code HTML/Twig** (vue complète), ou un fragment de données au format **JSON** — dans ce dernier cas, on parle souvent d'**API REST** ou d'**API RESTful**.

Une **API RESTful** désigne un ensemble de routes organisées autour de ressources (ex : utilisateur, aventure, photo), accessibles selon des **méthodes HTTP standardisées** :

- **GET** pour lire,
- **POST** pour créer,
- **PUT** ou **PATCH** pour modifier,
- **DELETE** pour supprimer.

Pathfinding utilise ce modèle pour plusieurs fonctionnalités, comme l'édition d'une aventure ou la suppression d'une photo, grâce à des appels AJAX (Asynchronous JavaScript and XML), qui permettent d'interagir avec le serveur sans recharger la page. Cela rend l'interface plus fluide et réactive — un aspect clé dans la gestion des photos, des compteurs dynamiques ou des contacts de confiance.

Ce mécanisme client-serveur, structuré autour de **protocoles (HTTP/HTTPS)**, **routes**, **contrôleurs**, **formats de réponse** et **appels AJAX**, a été essentiel pour proposer une expérience utilisateur à la fois sécurisée, dynamique et moderne.

Après avoir assuré une communication fiable et sécurisée entre le front-end et le back-end, il est essentiel de vérifier que chaque composant fonctionne comme attendu, indépendamment ou en interaction avec d'autres. C'est là qu'interviennent les tests unitaires et fonctionnels, garants de la stabilité et de la qualité du code.

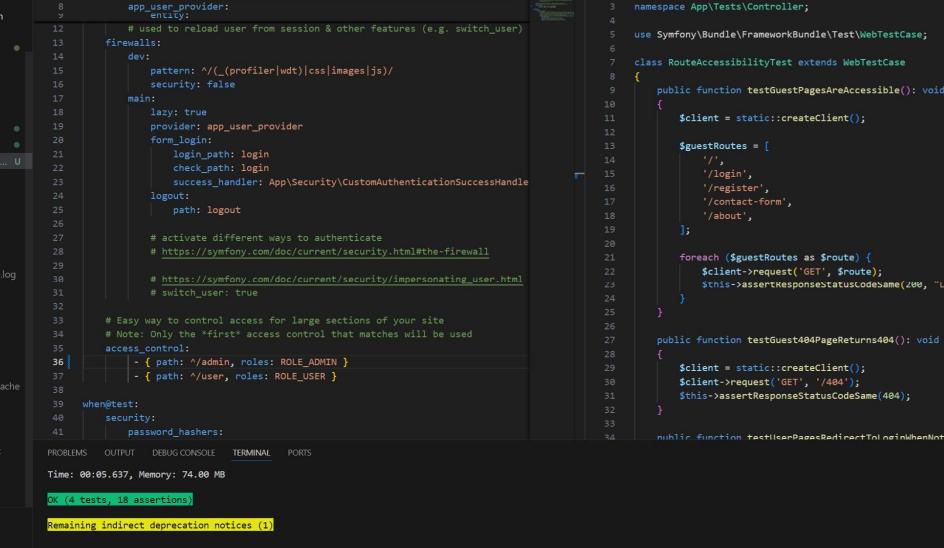
1.3.3. Présentation des tests unitaires et fonctionnels

Un **test unitaire** consiste à vérifier de façon automatisée le bon comportement d'une **unité isolée du code**, généralement une fonction ou une méthode. L'objectif est de s'assurer que cette portion de code produit le résultat attendu pour différentes entrées, indépendamment du reste de l'application. Ce type de test est particulièrement utile pour identifier rapidement les bugs à la racine, éviter les régressions et renforcer la robustesse du code. Par exemple, un test unitaire pourrait vérifier que la méthode de calcul de la date d'expiration d'une alerte renvoie bien la bonne valeur en fonction de la durée renseignée.

Un **test fonctionnel**, quant à lui, évalue le fonctionnement global d'une fonctionnalité complète, en simulant un scénario réel d'utilisation : saisie de formulaire, validation, affichage, interaction avec l'interface ou l'API, etc. Il vérifie que toutes les briques techniques de l'application (contrôleurs, vues, modèles) fonctionnent **ensemble** pour délivrer le résultat attendu. Par exemple, un test fonctionnel pourrait tester tout le processus de connexion : affichage de la page login, saisie des identifiants, validation et redirection vers le tableau de bord.

Dans Pathfinding, j'ai mis en place un test fonctionnel simple: le contrôle de l'accessibilité des routes à un utilisateur non connecté (Visiteur, cf. **Figure 15**). Le **contrôle de qualité s'est donc majoritairement fait manuellement**, en testant chaque page et chaque fonctionnalité à la main, via l'inspecteur de navigateur (vérification des requêtes HTTP, du rendu visuel, des erreurs console, etc.).

Une évolution à court terme du projet consisterait à mettre en place un **test automatique**, capable de parcourir toutes les routes du site en tant qu'utilisateur connecté puis admin, et de vérifier qu'elles retournent bien un **code HTTP 200** (page accessible) avec les bonnes restrictions d'accès, sans erreur critique. Cela permettrait de valider en un clic la stabilité minimale de l'interface avant tout nouveau déploiement.



The screenshot shows a dual-monitor setup with two instances of the Visual Studio Code (VS Code) code editor. Both monitors are displaying Symfony projects.

Left Monitor (Project: PATHFINDING-MVP):

- File Explorer:** Shows files like bootstrap.php, controllers.json, bin, config, migrations, public, src, templates, tests, and a Controller folder containing RouteAccessibilityTest.php.
- Code Editor:** Displays the `security.yaml` file. The configuration includes providers (app_user_provider), firewalls (dev, main), and access control rules for paths like `/admin` and `/user`.
- Terminal:** Shows the output of a test run: "Time: 00:05.637, Memory: 74.00 MB" and "Ok (4 tests, 18 assertions)".
- Bottom Status Bar:** Shows "Remaining indirect deprecation notices (1)".

Right Monitor (Project: RouteAccessibilityTest):

- File Explorer:** Shows files like RouteAccessibilityTest.php.
- Code Editor:** Displays the `RouteAccessibilityTest.php` file, which contains unit tests for the `RouteAccessibilityTest` class.
- Terminal:** Shows the output of a test run: "Time: 00:05.637, Memory: 74.00 MB" and "Ok (4 tests, 18 assertions)".
- Bottom Status Bar:** Shows "Remaining indirect deprecation notices (1)".

Figure 15 : Test fonctionnel pour vérifier l'accessibilité des routes à un Visiteur

Après avoir posé le cadre du projet Pathfinding dans cette première partie, la seconde, débutant page suivante, aborde son développement, en commençant par le back-end, socle logique de l'application.

2. PRESENTATION SPECIFIQUE DU PROJET

2.1. Développement Back-end

Avant toute logique métier ou affichage côté client, le projet repose sur une base de données relationnelle solide, pensée dès le départ pour refléter la complexité des interactions entre utilisateurs, aventures et mécanismes de sécurité. C'est donc naturellement par la conception de cette base et sa gestion via Doctrine ORM que débute le développement back-end.

2.1.1. Conception et gestion de la base de données relationnelle

Le projet **Pathfinding** repose sur une base de données relationnelle construite pour répondre à des besoins spécifiques : structurer les données utilisateurs, gérer des aventures, des alertes, des contacts de confiance, des fichiers (GPX, photos, documents) et sécuriser l'accès à ces données sensibles. Le choix initial s'est porté sur **MySQL**, accessible localement via **phpMyAdmin**, mais la structure est pensée pour pouvoir évoluer vers **PostgreSQL/PostGIS**, afin d'optimiser la gestion des coordonnées GPS à moyen terme.

Comme tout **SGBD** (Système de Gestion de Base de Données), MySQL permet d'effectuer les opérations **CRUD** (Create, Read, Update, Delete), essentielles au bon fonctionnement d'une application dynamique.

Chaque **table** représente une entité logique : User, Adventure, SafetyContact, SafeAlert, TimerAlert, etc. Chaque ligne correspond à un enregistrement, chaque colonne à une propriété (ex : name, email, status). Les relations sont organisées via des **clés primaires** (PK) et **clés étrangères** (FK), permettant de relier les données : un utilisateur peut avoir plusieurs aventures, chaque aventure peut avoir plusieurs fichiers, mais appartient à un seul utilisateur. Les cardinalités utilisées suivent les standards 1–1, 1–N et N–N, selon les cas (ex : relation N–N entre Contact et ContactList).

Enfin, l'intégrité des données est assurée par l'usage des contraintes suivantes :

- NOT NULL : aucune donnée vide pour les champs critiques (email, titre...),
- UNIQUE : unicité garantie pour les adresses email,
- PRIMARY KEY : identifie chaque entrée de manière fiable,
- FOREIGN KEY : assure la cohérence entre tables liées.

Les **types de données** utilisés dans la base Pathfinding ont été choisis en fonction des besoins métiers et des performances attendues. Parmi les principaux types utilisés :

- INT pour les identifiants et statuts numériques,
- VARCHAR (x) pour les champs texte courts (nom, email, etc.),
- TEXT pour les descriptions plus longues (ex : récit d'aventure),
- ENUM ou VARCHAR contrôlé pour les valeurs limitées (ex : visibilité ou type d'aventure),
- BOOLEAN pour les valeurs binaires (public/privé, activé/désactivé),
- DATETIME_IMMUTABLE pour assurer l'intégrité des dates de création, d'alerte, etc.

Cette base solide a permis de générer des entités Symfony cohérentes, enrichies ensuite de contraintes de validation côté code, que nous allons explorer dans la section suivante.

2.1.2. Modélisation des entités et gestion des contraintes de validation

Le **modèle relationnel** a été conçu à partir de la méthode MERISE dans le logiciel **Looping**, avec une première phase en **MCD** (Modèle Conceptuel de Données), puis traduite en **MLD** (Modèle Logique de Données), ce qui a permis de réfléchir en amont aux entités métiers du projet et à leurs relations.

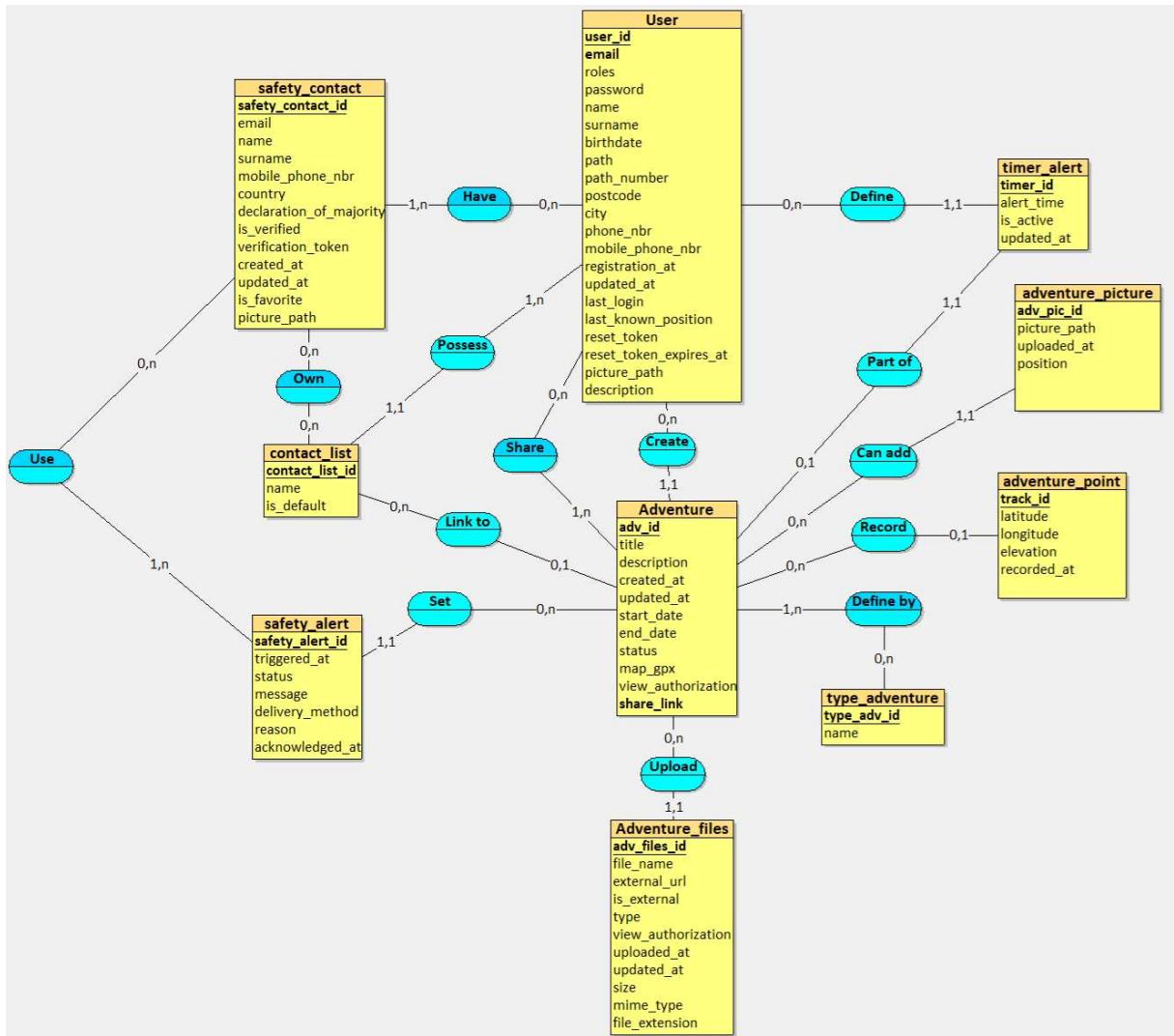


Figure 16 : Modèle Conceptuel de Données de Pathfinding

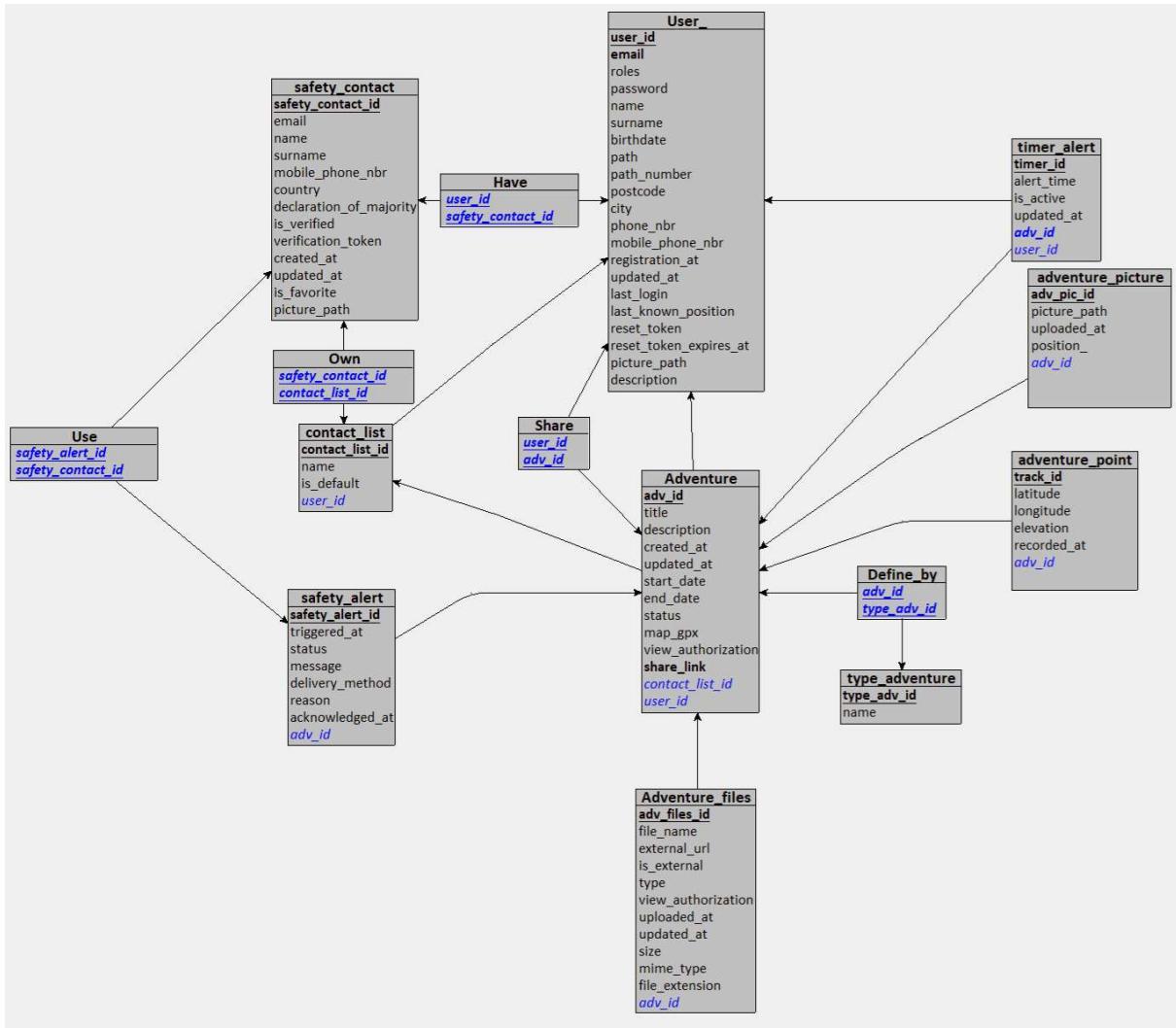


Figure 17 : Modèle Logique de Données de Pathfinding

Une fois la structure validée, elle a été transposée sous forme d'**entités Symfony** à l'aide de **Doctrine ORM**. Chaque entité PHP (par exemple Adventure.php, User.php, Contact.php) reflète une table SQL, avec ses attributs, types et relations.

L'implémentation de **contraintes de validation** Symfony (@Assert) dans les entités a permis d'intégrer la logique métier au plus proche des données. Cela garantit :

- une meilleure **cohérence** à l'enregistrement,
- une **sécurité** renforcée côté back-end,
- et une **UX** plus fluide via des retours d'erreur précis dans les formulaires.

Exemples :

- `@Assert\Length(min=3)` pour les noms,
- `@Assert\Regex("/^a-zA-Z\-[a-zA-Z\]+$/u")` pour valider les prénoms,
- `@Assert\Email` pour les adresses mail,
- `@Assert\Count(min=2, max=5)` pour limiter les contacts dans une liste.

```

src > Entity > User.php
14  #[ORM\Entity(repositoryClass: UserRepository::class)]
15  #[ORM\UniqueConstraint(name: 'UNIQ_IDENTIFIER_EMAIL', fields: ['email'])]
16  class User implements UserInterface, PasswordAuthenticatedUserInterface
17  {
18      #[ORM\Id]
19      #[ORM\GeneratedValue]
20      #[ORM\Column]
21      private ?int $id = null;
22
23      #[ORM\Column(length: 180)]
24      #[Assert\NotBlank]
25      #[Assert\Email]
26      #[Assert\Length(min: 5)]
27      private ?string $email = null;
28
29      /**
30      * @var list<string> The user roles
31      */
32      #[ORM\Column]
33      private array $roles = [];
34
35      /**
36      * @var string The hashed password
37      */
38      #[ORM\Column]
39      #[Assert\NotBlank]
40      #[Assert\Length(min: 8)]
41      #[Assert\Regex(
42          pattern: '/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[\W_]).+$/',
43          message: 'Le mot de passe doit contenir au moins une majuscule, une minuscule, un chiffre et un caractère spécial.')
44      ]
45      private ?string $password = null;
46
47      #[ORM\Column(length: 100, nullable: true)]
48      private ?string $resetToken = null;
49
50      #[ORM\Column(type: 'datetime_immutable', nullable: true)]
51      private ?\DateTimeImmutable $resetTokenExpiresAt = null;
52
53      #[ORM\Column(length: 255)]
54      #[Assert\NotBlank]
55      #[Assert\Length(min: 2)]
56      #[Assert\Regex(
57          pattern: '/^[\p{L} -]+$/u',
58          message: "Le nom ne peut contenir que des lettres, espaces ou tirets.")]
59      private ?string $name = null;
60
61      #[ORM\Column(length: 255)]
62      #[Assert\NotBlank]
63      #[Assert\Length(min: 2)]
64      #[Assert\Regex(
65          pattern: '/^[\p{L} -]+$/u',
66          message: "Le nom ne peut contenir que des lettres, espaces ou tirets.")]
67      private ?string $surname = null;
68
69      #[ORM\Column(type: Types::DATE_MUTABLE)]
70      #[Assert\NotBlank]
71      #[Assert\LessThanOrEqual(
72          value: 'today -18 years',
73          message: 'Vous devez avoir au moins 18 ans pour vous inscrire.')]
74      private ?\DateTime $birthdate = null;

```

Figure 18 : Exemples d'Asserts sur l'Entity User de Pathfinding

Enfin, une fois les entités créées et validées, la base a été générée automatiquement via les **migrations Doctrine** (php bin/console doctrine:migrations:migrate), avec quelques ajustements manuels réalisés dans les fichiers des entités tel que Adventure.php (type ENUM, relation entre entités, ...).

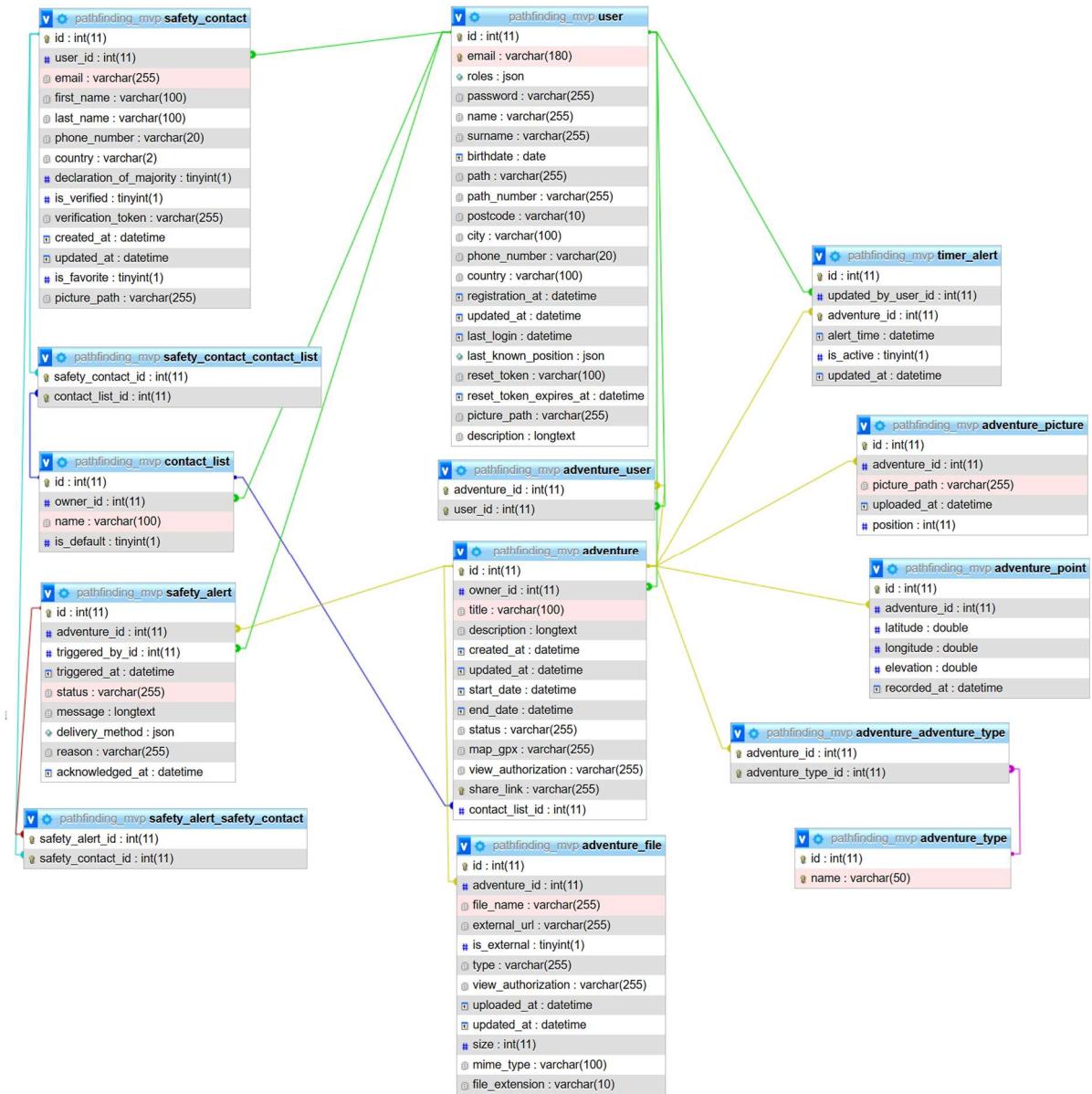


Figure 19 : Entités créées pour le projet Pathfinding – 08/06/2025

Une fois les entités définies et validées, il a fallu penser aux interactions concrètes avec ces données. C'est là qu'entrent en jeu les routes et les API, qui orchestrent la communication entre le front-end, l'utilisateur, et la logique métier du back-end.

2.1.3. Routes et API

Dans Symfony, une **route** définit un point d'entrée vers une fonctionnalité du back-end. Chaque route est associée à une URL, une méthode HTTP (GET, POST, etc.) et à un **contrôleur** chargé de traiter la requête. Symfony utilise l'annotation `#[Route(...)]` ou la configuration YAML/PHP pour relier l'URL à la méthode à exécuter.

Les routes de Pathfinding sont principalement regroupées par domaine fonctionnel. Par exemple, les routes liées aux contacts de sécurité (SafetyContact) permettent :

- d'afficher et gérer tous ses contacts (GET /user/contacts-manager),
- de créer un contact (GET|POST /user/create-contact),
- de modifier un contact (GET|POST /user/update-contact/{id}),
- de supprimer un contact (GET|POST /user/delete-contact/{id}),
- ou de modifier dynamiquement l'affichage d'une liste (GET /user/contacts-ajax, POST /user/update-contact-list).

Certaines routes (comme contacts-ajax) retournent un **fragment HTML rendu côté serveur** mais injecté dynamiquement via **AJAX**, afin d'offrir une meilleure réactivité sans recharger la page.

```

34     #[Route('/user/contacts-manager', name: 'contacts-manager', methods: ['GET', 'POST'])]
35     public function displayListContacts(
36         Request $request,
37         SafetyContactRepository $safetyContactRepository,
38         ContactListRepository $contactListRepository,
39         EntityManagerInterface $entityManager,
40         ParameterBagInterface $parameterBag
41     ): Response {

```

Figure 20 : Exemple de route pour afficher la liste des contacts de confiance

Symfony ne gère pas uniquement les routes synchrones : il peut aussi servir de **base API REST**, notamment via des réponses JSON. Dans le projet Pathfinding, cette approche est utilisée uniquement en interne (AJAX) et non exposée publiquement sous forme d'API documentée (type OpenAPI).

Les routes constituent seulement la porte d'entrée : c'est dans les **contrôleurs** que se trouvent les véritables traitements métiers. Ceux-ci s'appuient sur des entités Doctrine, des services, et une logique serveur spécifique à Pathfinding.

2.1.4. Contrôleurs et composants métiers côté serveur

Chaque route de Pathfinding appelle un **contrôleur** Symfony, qui agit comme un chef d'orchestre : il récupère les données, applique la logique métier, et renvoie une réponse adaptée (HTML ou JSON).

a) Architecture MVC (Modèle-Vue-Contrôleur)

Dans Symfony et Pathfinding :

- le **Contrôleur** reçoit la requête utilisateur,
- il interagit avec les **modèles (entités)** via Doctrine ORM,
- et rend une **vue Twig** ou un **fragment de réponse JSON/HTML**.

b) Interaction avec les entités et la base de données

Les contrôleurs utilisent des **repositories Doctrine**, générés automatiquement à partir des entités (ex : SafetyContactRepository), pour interroger la base de données avec des requêtes personnalisées.

```

286     #[Route('/share/{shareLink}', name: 'adventure_share')]
287     public function share(string $shareLink, AdventureRepository $repo): Response {
288         $adventure = $repo->findOneBy(['shareLink' => $shareLink]);
289
290         if (!$adventure || !$adventure->isVisibleTo(null)) {
291             throw $this->createNotFoundException();
292         }
293
294         return $this->render('adventure/public_view.html.twig', [
295             'adventure' => $adventure
296         ]);
297     }
298 }
```

Figure 21 : Exemple de route pour partager une aventure à un visiteur

Dans l'**exemple de route ci-dessus**, Symfony définit une route /share/{shareLink}, qui accepte un paramètre dynamique shareLink directement dans l'URL. Cette route est nommée adventure_share, ce qui permet de la référencer facilement dans le code (génération de liens, redirections...).

Lorsqu'un utilisateur visite cette URL, Symfony déclenche la méthode share() du contrôleur, en lui transmettant la valeur du shareLink. Grâce à l'injection de dépendances, le contrôleur reçoit également automatiquement une instance du repository AdventureRepository, qui est responsable des interactions avec l'entité Adventure.

Le contrôleur utilise ensuite ce repository pour rechercher une aventure en base de données dont le champ shareLink correspond à la valeur extraite de l'URL. Si aucune aventure n'est trouvée, ou si celle-ci est en visibilité restreinte (c'est-à-dire privée), le contrôleur génère une erreur 404 via createNotFoundException().

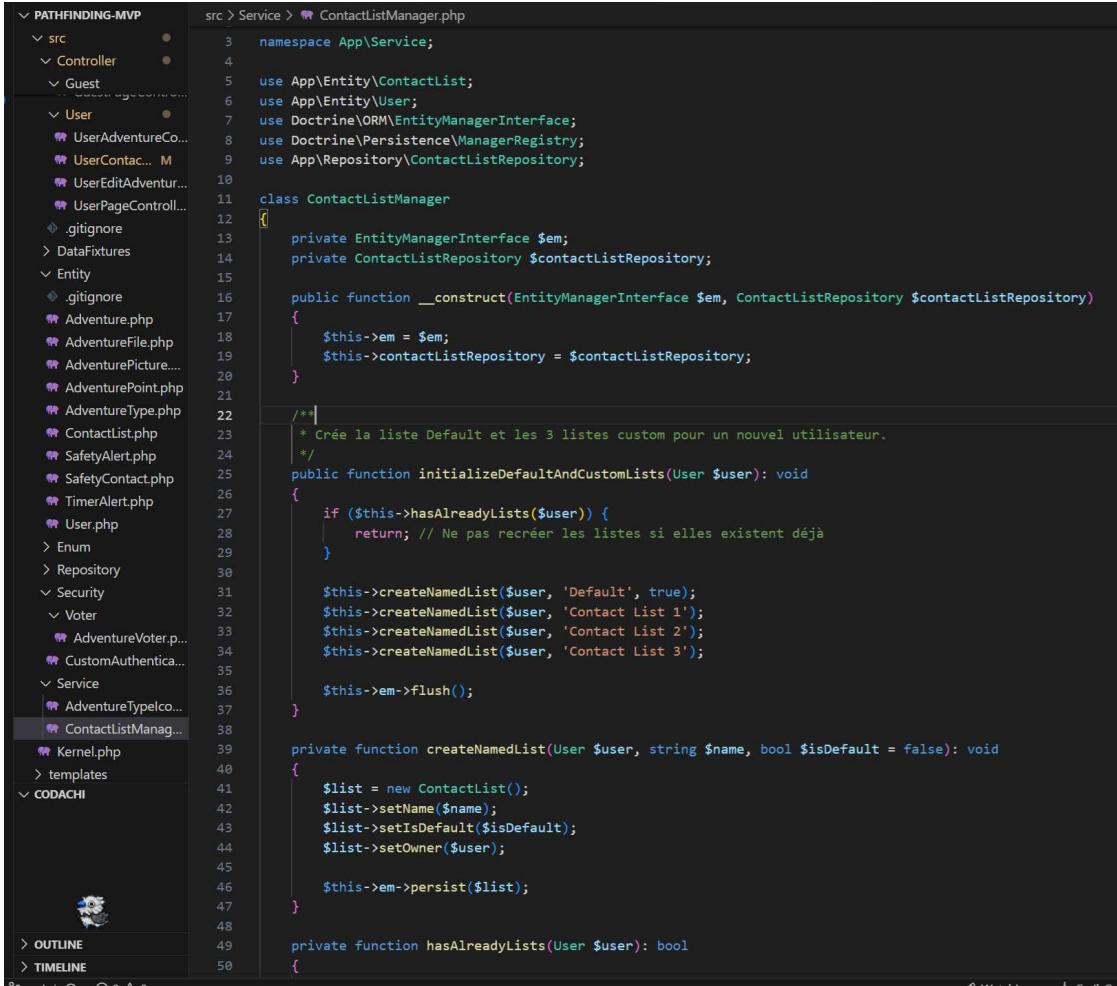
Enfin, si l'aventure existe et peut être consultée, elle est transmise à la vue Twig public_view.html.twig, qui se chargera de l'affichage final. Ce processus illustre la chaîne complète : **route → contrôleur → requête en base → logique métier → rendu de la vue**.

c) Logique métier embarquée

Certaines règles spécifiques à Pathfinding sont appliquées côté serveur, par exemple :

- Une **liste de contacts** ne peut contenir **plus de 5 membres**, et la liste par défaut ajoute automatiquement les 2 premiers contacts de confiance, elle ne peut pas être vide sauf si l'utilisateur n'a aucun contact de confiance.
- Chaque utilisateur dispose d'une **liste Default** et jusqu'à **3 listes personnalisées**.
- La modification d'un contact déclenche des mises à jour conditionnelles (ex. : favoris, listes liées).
- La **validation** des données repose sur le composant Symfony ValidatorInterface, couplé aux **contraintes Assert définies dans l'entité**.

Le contrôleur délègue certaines tâches complexes à des **services métier personnalisés**, comme **ContactListManager**, chargé d'initialiser les listes ou de valider les contraintes liées aux favoris. Cela garantit une meilleure **répartition des responsabilités** et une **réutilisabilité** du code.



```

src > Service > ContactListManager.php
  3  namespace App\Service;
  4
  5  use App\Entity\ContactList;
  6  use App\Entity\User;
  7  use Doctrine\ORM\EntityManagerInterface;
  8  use Doctrine\Persistence\ManagerRegistry;
  9  use App\Repository\ContactListRepository;
10
11 class ContactListManager
12 {
13     private EntityManagerInterface $em;
14     private ContactListRepository $contactListRepository;
15
16     public function __construct(EntityManagerInterface $em, ContactListRepository $contactListRepository)
17     {
18         $this->em = $em;
19         $this->contactListRepository = $contactListRepository;
20     }
21
22     /**
23      * Crée la liste Default et les 3 listes custom pour un nouvel utilisateur.
24      */
25     public function initializeDefaultAndCustomLists(User $user): void
26     {
27         if ($this->hasAlreadyLists($user)) {
28             return; // Ne pas recréer les listes si elles existent déjà
29         }
30
31         $this->createNamedList($user, 'Default', true);
32         $this->createNamedList($user, 'Contact List 1');
33         $this->createNamedList($user, 'Contact List 2');
34         $this->createNamedList($user, 'Contact List 3');
35
36         $this->em->flush();
37     }
38
39     private function createNamedList(User $user, string $name, bool $isDefault = false): void
40     {
41         $list = new ContactList();
42         $list->setName($name);
43         $list->setIsDefault($isDefault);
44         $list->setOwner($user);
45
46         $this->em->persist($list);
47     }
48
49     private function hasAlreadyLists(User $user): bool
50     {
51
52     }

```

Figure 22 : Extrait du composant métier ContactListManager

Le service ContactListManager est un composant métier centralisant la logique de gestion des listes de contacts. Son constructeur reçoit deux dépendances injectées automatiquement par Symfony :

- **EntityManagerInterface** pour interagir avec la base de données (persistance, enregistrement),
- **ContactListRepository** pour effectuer des requêtes sur les listes déjà existantes.

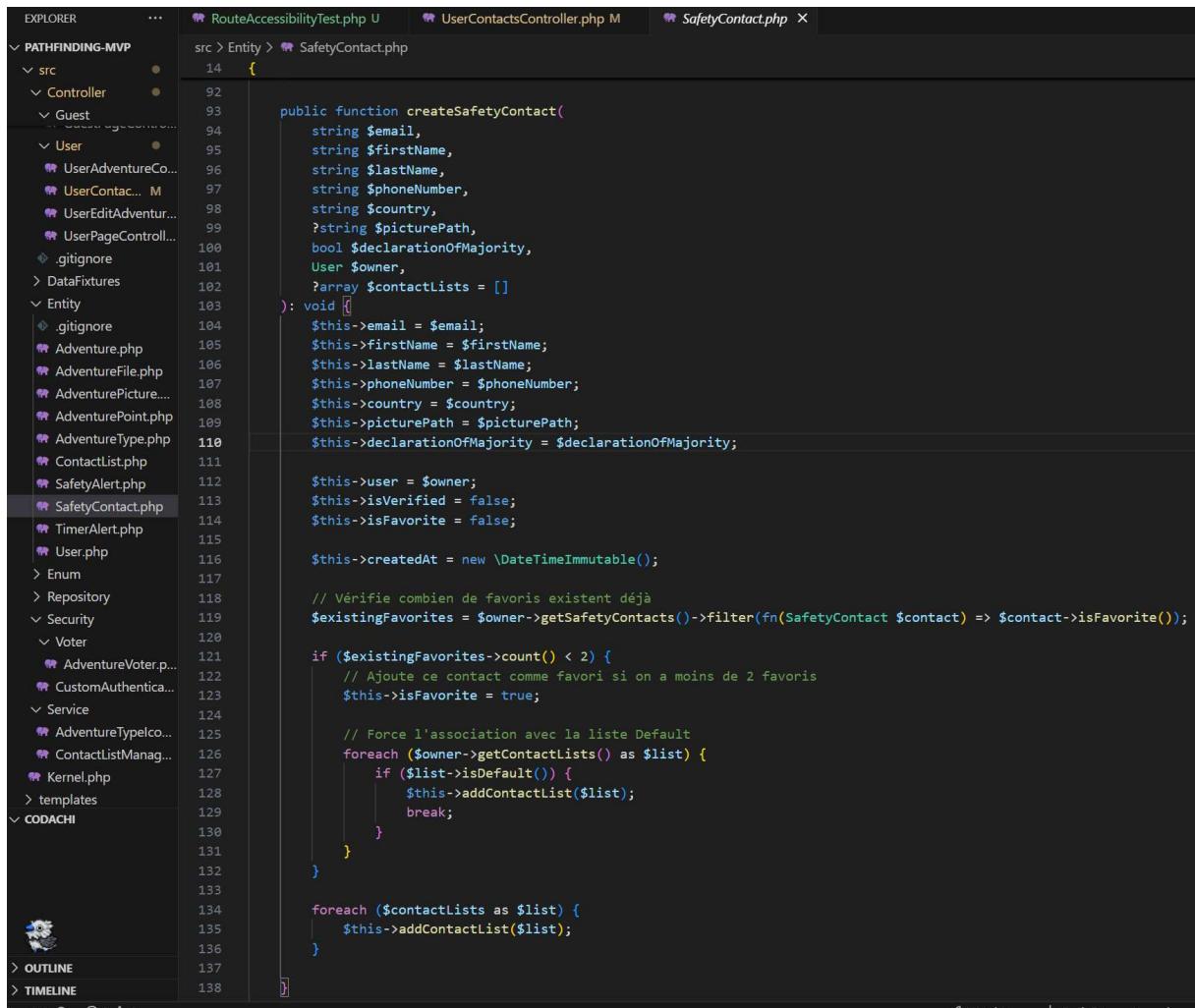
La méthode publique **initializeDefaultAndCustomLists()** est appelée lors de l'inscription ou du premier accès d'un utilisateur. Elle vérifie d'abord, via la méthode privée **hasAlreadyLists()**, si l'utilisateur possède déjà des listes. Si ce n'est pas le cas, elle crée automatiquement quatre listes :

- Une liste “Default”, marquée par le booléen `isDefault = true`,
- Trois listes personnalisées nommées génériquement “Contact List 1”, “Contact List 2” et “Contact List 3”.

Chaque liste est instanciée via la méthode `createNamedList()`, où elle est associée à l'utilisateur, puis persistée. Un seul appel à `flush()` est réalisé à la fin pour optimiser la transaction.

Ce service illustre l'intérêt de **déléguer les traitements métiers hors des contrôleurs** : la création automatisée et cohérente des listes de contacts est encapsulée dans une classe dédiée, ce qui renforce la **réutilisabilité**, la **testabilité** et la **lisibilité** du code.

Le projet Pathfinding adopte une approche complémentaire consistant à encapsuler certaines opérations directement dans les entités. Par exemple :



```

EXPLORER      ...
PATHFINDING-MVP
src > Entity > SafetyContact.php M
14  {
15
16  public function createSafetyContact(
17      string $email,
18      string $firstName,
19      string $lastName,
20      string $phoneNumber,
21      string $country,
22      ?string $picturePath,
23      bool $declarationOfMajority,
24      User $owner,
25      ?array $contactLists = []
26  ): void {
27      $this->email = $email;
28      $this->firstName = $firstName;
29      $this->lastName = $lastName;
30      $this->phoneNumber = $phoneNumber;
31      $this->country = $country;
32      $this->picturePath = $picturePath;
33      $this->declarationOfMajority = $declarationOfMajority;
34
35      $this->user = $owner;
36      $this->isVerified = false;
37      $this->isFavorite = false;
38
39      $this->createdAt = new \DateTimeImmutable();
40
41      // Vérifie combien de favoris existent déjà
42      $existingFavorites = $owner->getSafetyContacts()->filter(fn(SafetyContact $contact) => $contact->isFavorite());
43
44      if ($existingFavorites->count() < 2) {
45          // Ajoute ce contact comme favori si on a moins de 2 favoris
46          $this->isFavorite = true;
47
48          // Force l'association avec la liste Default
49          foreach ($owner->getContactLists() as $list) {
50              if ($list->isDefault()) {
51                  $this->addContactList($list);
52                  break;
53              }
54          }
55
56          foreach ($contactLists as $list) {
57              $this->addContactList($list);
58          }
59      }
60  }
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138

```

Figure 23 : Extrait de logique métier déportée dans l'entité SafetyContact

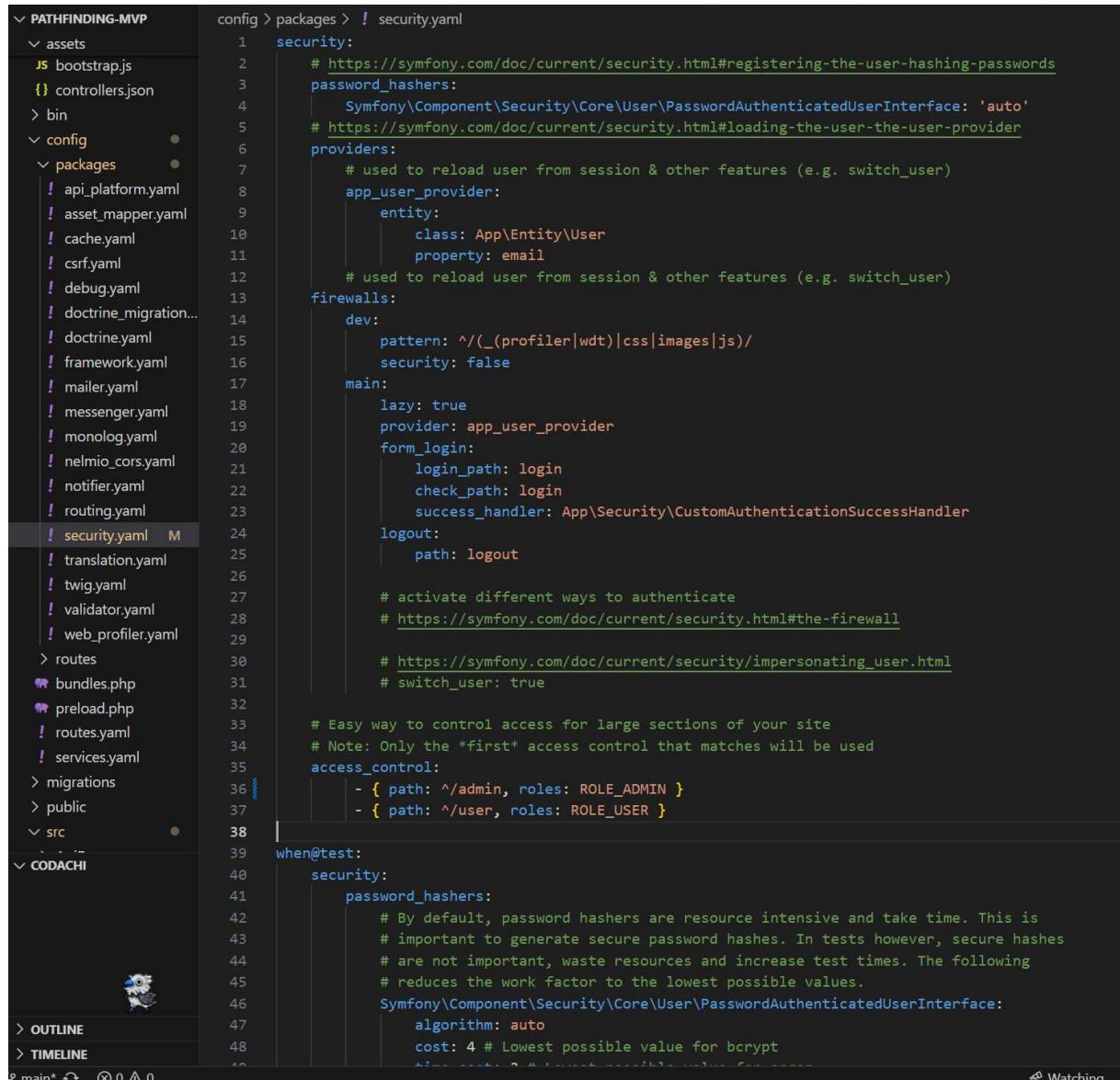
Ici, la méthode `createSafetyContact()` est définie dans l'entité elle-même et regroupe l'ensemble des opérations nécessaires à l'initialisation complète d'un contact. Cette stratégie permet aux entités de porter leur propre logique métier, tout en allégeant considérablement les contrôleurs. Ces derniers se concentrent ainsi sur la gestion des flux HTTP, laissant les entités et les services gérer les règles métiers. Cela contribue à un code plus cohérent, maintenable, et aligné avec les principes de conception orientée objet.

Ainsi, la structuration du code autour des routes, contrôleurs, services et entités permet une organisation claire et évolutive des fonctionnalités serveur. Pour garantir un accès sécurisé à ces ressources et protéger les données sensibles des utilisateurs, le projet Pathfinding repose également sur une gestion rigoureuse des mécanismes d'authentification, de rôles et d'autorisations, que nous détaillons dans la partie suivante.

2.1.5. Sécurité appliquée : authentification, rôles et autorisations

La sécurité constitue un pilier fondamental du projet Pathfinding, qui manipule des données sensibles telles que les informations personnelles des utilisateurs, les coordonnées GPS ou encore les listes de contacts d'urgence.

Le système repose sur le composant Security de Symfony, configuré avec une authentification personnalisée par formulaire et un système de rôles hiérarchisés (ROLE_USER, ROLE_ADMIN, ROLE_SUPERADMIN (futur)), permettant de gérer précisément les autorisations d'accès aux différentes routes et fonctionnalités. Le contrôle d'accès est défini à la fois dans le fichier `security.yaml` via des `access_control` (ex : `/user` uniquement accessible aux utilisateurs connectés), et dans les contrôleurs eux-mêmes via des vérifications explicites.



```

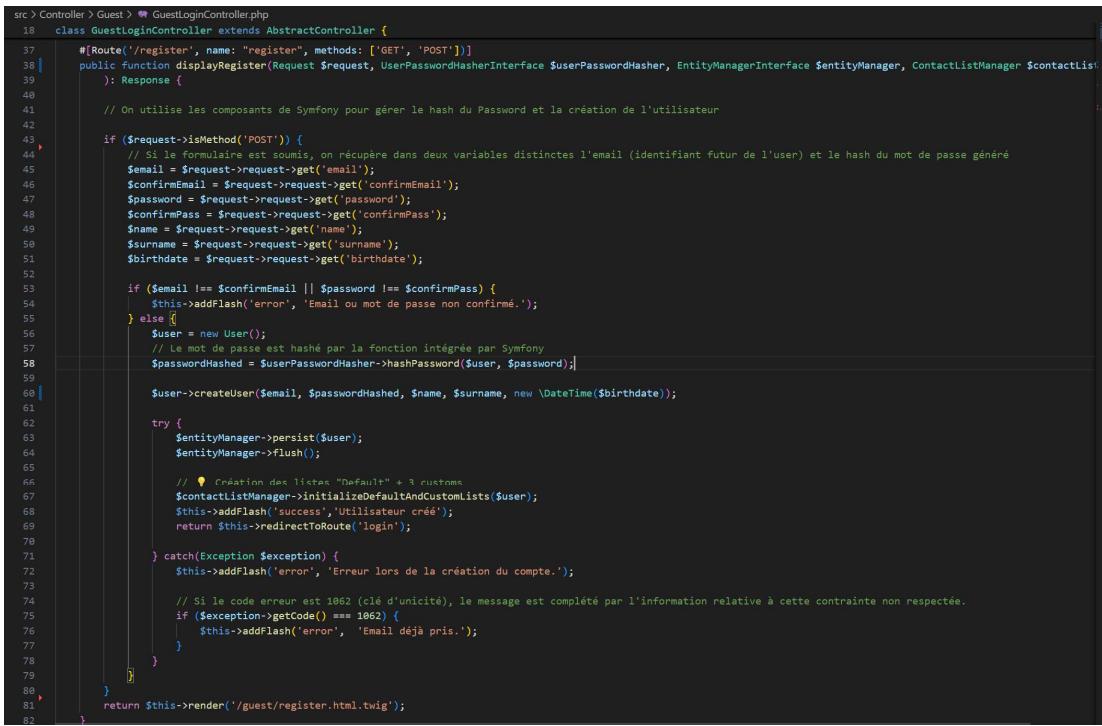
PATHFINDING-MVP
  assets
    JS bootstrap.js
    {} controllers.json
  > bin
  > config
    packages
      api_platform.yaml
      asset_mapper.yaml
      cache.yaml
      csrf.yaml
      debug.yaml
      doctrine_migration...
      doctrine.yaml
      framework.yaml
      mailer.yaml
      messenger.yaml
      monolog.yaml
      nelmio_cors.yaml
      notifier.yaml
      routing.yaml
      security.yaml M
      translation.yaml
      twig.yaml
      validator.yaml
      web_profiler.yaml
  > routes
    bundles.php
    preload.php
    routes.yaml
    services.yaml
  > migrations
  > public
  > src
    CODACHI
      < icon>
  > OUTLINE
  > TIMELINE
  main < icon> 0.0 ▲ 0
  When: test
  security:
    password_hashers:
      # By default, password hashers are resource intensive and take time. This is
      # important to generate secure password hashes. In tests however, secure hashes
      # are not important, waste resources and increase test times. The following
      # reduces the work factor to the lowest possible values.
      Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:
        algorithm: auto
        cost: 4 # Lowest possible value for bcrypt

```

Figure 24 : Configuration `security.yaml` et redirection personnalisée après connexion

- **Authentification et chiffrement des mots de passe**

Lors de l'inscription, les mots de passe sont hachés avec l'algorithme Bcrypt, fourni par Symfony via l'interface `UserPasswordHasherInterface`. Il s'agit d'un hachage unidirectionnel, ce qui signifie qu'il est mathématiquement impossible de retrouver le mot de passe initial à partir du mot de passe hashé stocké en base. Cela garantit une haute protection en cas de fuite de données.



```

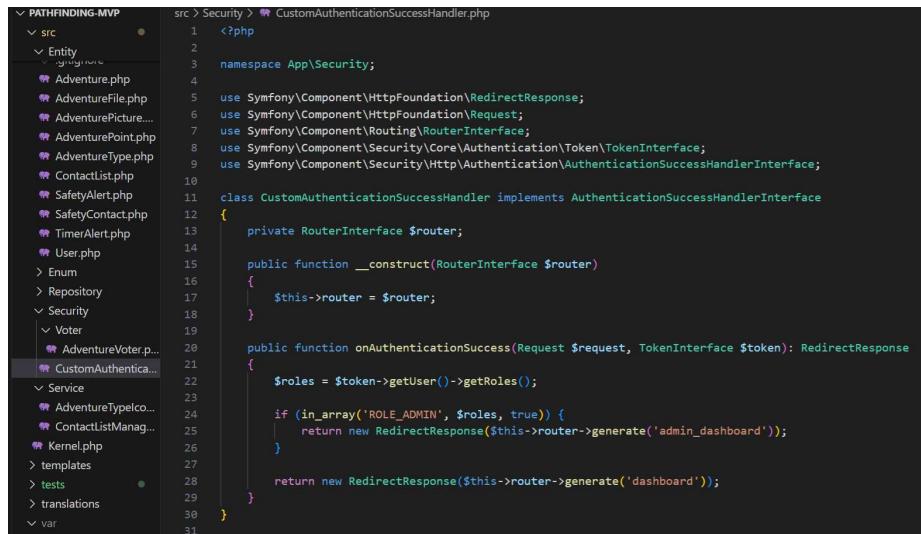
src > Controller > Guest > GuestLoginController.php
18  class GuestLoginController extends AbstractController {
19
20      #[Route('/register', name: "register", methods: ['GET', 'POST'])]
21      public function displayRegister(Request $request, UserPasswordHasherInterface $userPasswordHasher, EntityManagerInterface $entityManager, ContactListManager $contactListManager)
22      {
23          // On utilise les composants de Symfony pour gérer le hash du Password et la création de l'utilisateur
24
25          if ($request->isMethod('POST')) {
26              // Si le formulaire est soumis, on récupère dans deux variables distinctes l'email (identifiant futur de l'user) et le hash du mot de passe générés
27              $email = $request->get('email');
28              $confirmEmail = $request->get('confirmEmail');
29              $password = $request->get('password');
30              $confirmPass = $request->get('confirmPass');
31              $name = $request->get('name');
32              $surname = $request->get('surname');
33              $birthdate = $request->get('birthdate');
34
35              if ($email == $confirmEmail || $password != $confirmPass) {
36                  $this->addFlash('error', 'Email ou mot de passe non confirmé.');
37              } else {
38                  $user = new User();
39                  // Le mot de passe est hashé par la fonction intégrée par Symfony
40                  $passwordHashed = $userPasswordHasher->hashPassword($user, $password);
41
42                  $user->createUser($email, $passwordHashed, $name, $surname, new \DateTime($birthdate));
43
44                  try {
45                      $entityManager->persist($user);
46                      $entityManager->flush();
47
48                      // 🌟 Création des listes "Default" + 3 customs
49                      $contactListManager->initializeDefaultAndCustomLists($user);
50                      $this->addFlash('success', 'Utilisateur créé');
51                      return $this->redirectToRoute('login');
52                  } catch (Exception $exception) {
53                      $this->addFlash('error', 'Erreur lors de la création du compte.');
54
55                      // Si le code erreur est 1062 (clé d'unicité), le message est complété par l'information relative à cette contrainte non respectée.
56                      if ($exception->getCode() === 1062) {
57                          $this->addFlash('error', 'Email déjà pris.');
58                      }
59                  }
60              }
61          }
62
63          return $this->render('/guest/register.html.twig');
64      }
65  }

```

Figure 25 : Extrait d'un contrôleur d'inscription avec hachage du mot de passe

- **Authentification personnalisée**

Une classe `CustomAuthenticationSuccessHandler` permet de rediriger dynamiquement l'utilisateur après connexion selon ses rôles. Par exemple, un `ROLE_ADMIN` est envoyé vers le tableau de bord d'administration, tandis qu'un `ROLE_USER` classique est redirigé vers son espace personnel.



```

src > Security > CustomAuthenticationSuccessHandler.php
1  <?php
2
3  namespace App\Security;
4
5  use Symfony\Component\HttpFoundation\RedirectResponse;
6  use Symfony\Component\HttpFoundation\Request;
7  use Symfony\Component\Routing\RouterInterface;
8  use Symfony\Component\Security\Core\Authentication\Token\TokenInterface;
9  use Symfony\Component\Security\Http\Authentication\AuthenticationSuccessHandlerInterface;
10
11  class CustomAuthenticationSuccessHandler implements AuthenticationSuccessHandlerInterface
12  {
13      private RouterInterface $router;
14
15      public function __construct(RouterInterface $router)
16      {
17          $this->router = $router;
18      }
19
20      public function onAuthenticationSuccess(Request $request, TokenInterface $token): RedirectResponse
21      {
22          $roles = $token->getUser()->getRoles();
23
24          if (in_array('ROLE_ADMIN', $roles, true)) {
25              return new RedirectResponse($this->router->generate('admin_dashboard'));
26          }
27
28          return new RedirectResponse($this->router->generate('dashboard'));
29      }
30  }
31

```

Figure 26 : Redirection personnalisée après authentification selon le rôle de l'utilisateur

- **CSRF token et sécurisation des formulaires**

Tous les formulaires sensibles utilisent un CSRF token (Cross-Site Request Forgery), généré automatiquement par Symfony et vérifié à la soumission. Cela empêche toute tentative d'exploitation par un site tiers malveillant.

- **Doctrine, protection contre l'injection SQL et validation**

Le projet utilise l'ORM Doctrine, qui protège contre les injections SQL en effectuant une sanitisation automatique des paramètres dans les requêtes (ex. : `findOneBy(['email' => $input])`).

En complément, des contraintes de validation sont appliquées à chaque entité (via les annotations `@Assert`) et du côté client via JavaScript, garantissant l'intégrité des données dès leur saisie. Ces contraintes concernent la structure des emails, la complexité des mots de passe, la confirmation des champs, les dates valides, etc.

- **Protection des fichiers et gestion des uploads**

Les fichiers sensibles tels que les photos, documents GPX ou avatars ne sont pas stockés en base, mais dans des répertoires sécurisés du projet (définis dans services.yaml). Le fichier .gitignore empêche leur versionnage accidentel. Chaque upload est soumis à un contrôle strict de type MIME, taille maximale, et validé côté front et back pour éviter tout dépassement ou attaque par surcharge.

Figure 27 : Configuration de services.yaml

Ces mesures de sécurité garantissent un accès contrôlé, fiable et structuré à l'ensemble des fonctionnalités de Pathfinding. Elles s'intègrent pleinement dans la logique du code serveur, que ce soit au niveau des contrôleurs, des entités ou des services, en lien avec les composants métiers du projet.

La partie suivante se concentre donc sur l'aspect visuel et interactif de l'interface utilisateur, en détaillant les éléments clés du front-end et leur articulation avec le back-end.

2.2. Développement Front-end

Après avoir posé les fondations back-end de Pathfinding, cette seconde partie du développement se concentre sur la couche **front-end**, qui façonne l'expérience utilisateur au quotidien. Elle englobe l'intégration des interfaces via Twig, le comportement dynamique à l'aide de JavaScript, la gestion des droits d'affichage côté interface, ainsi qu'un design réactif et moderne.

2.2.1. Intégration des interfaces utilisateur (HTML/Twig)

L'architecture front-end de Pathfinding repose sur le moteur de templates **Twig**, intégré nativement à Symfony. Elle suit une approche modulaire et hiérarchique :

- Le fichier **base.html.twig** constitue le squelette principal : il inclut les balises `<head>`, les blocs CSS, JavaScript et la structure HTML de base. Tous les autres templates en héritent via `{% extends 'base.html.twig' %}`. Il y a un fichier base par rôle (admin, user, guest).
- Le contenu spécifique à chaque page est injecté dans des blocs personnalisés (`{% block main %}`, `{% block title %}`, etc.). Le bloc main reçoit une **classe unique** liée à la page courante (par exemple `main-dashboard`, `main-profile`), afin de permettre une **ciblage précis en SCSS** et d'assurer la séparation des styles.

```
templates > user > / base.html.twig > ↵ html > ↵ body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <!-- Permet de configurer le site en responsive -->
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7
8      <!-- Import du fichier style.css -->
9      <link rel="stylesheet" href="{{ asset('css/main.css') ~ '?v=' ~ app.request.server.get('REQUEST_TIME') }}>
10
11     <!-- Import de la bibliothèque externe Font Awesome pour les icons des social media -->
12     <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.0/css/all.min.css" />
13
14     <!-- Défini l'icône à gauche du titre de l'onglet du navigateur - de base une planète + message d'erreur dans l'inspecteur si rien de défini -->
15     <link rel="icon" href="{{ asset('assets/img/favicon.ico') }}" type="image/x-icon">
16
17     {% block link %}
18     {% endblock link %}
19
20     <!-- Nomme l'onglet du navigateur -->
21     {% block metatitle %}
22     {% endblock metatitle %}
23
24 </head>
25 <body>
26
27     {% include 'user/partials/_header.html.twig' %}
28
29     {% block main %}
30     {% endblock main %}
31
32     {% include 'user/partials/_footer.html.twig' %}
33
34     <!-- ! Import de la bibliothèque externe JS - SweetAlert2 pour les messages pop-ups -->
35     <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
36     <!-- ! Empêche d'afficher les données utilisateurs après déconnexion en cas d'utilisation de la flèche retour en arrière du navigateur
37     | en redirigeant vers la page login -->
38     <script src="{{ asset('js/antibackForward.js') }}" defer></script>
39     <script src="{{ asset('js/main.js') }}" defer></script>
40     <script src="{{ asset('js/menu.js') }}" defer></script>
41     <script src="{{ asset('js/components/confirmDeleteButton.js') }}" defer></script>
42
43     {% block script %}
44     {% endblock script %}
45
46 </body>
47 </html>
```

Figure 28 : Fichier `base.html.twig` de User – projet Pathfinding - 09/06/2025

- L'affichage est organisé autour de **partials réutilisables** :
 - `/_header.html.twig`, `/_footer.html.twig` pour les éléments persistants,
 - `/_contact-table.html.twig`, `/_photo-display-owner.html.twig`, `/_photo-display-viewer.html.twig`, etc. pour des composants spécifiques.
- Les vues sont **classées par répertoire** selon les profils utilisateurs :
 - Guest/ pour les visiteurs (login, register, etc.)
 - User/ pour les utilisateurs connectés,
 - Admin/ pour les interfaces administrateur.

Cette structure permet une **clarté de l'arborescence** et facilite les évolutions futures.

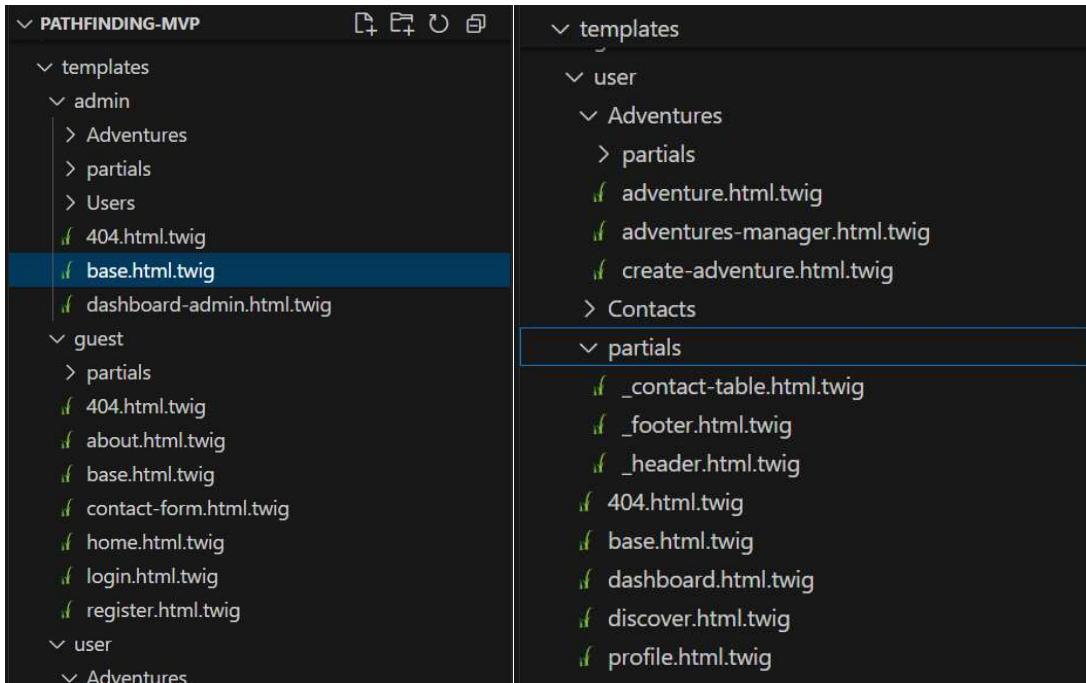


Figure 29 : Arborescence des templates Twig

De plus, certaines vues adaptent leur rendu en fonction des rôles ou droits d'accès via des conditions Twig (`{% if app.user %}` ou contrôles d'attributs comme `if isOwner`, `currentRoute == 'home' ? 'header-index' : 'header-login'`, `{% if adventure.status.value != 'achieved' %}`).

```
<div class="w100 flex-between mt-8">
  <div id="photo-data"
    data-adventure-id="{{ adventure.id }}"
    data-uploaded-count="{{ pictures|length }}"
    data-pictures2="{{ pictures|map(p => { id: p.id, path: asset(p.picturePath) | json_encode|e("html_attr") })}"
    data-pictures="{{ pictures|map(p => asset(p.picturePath))|json_encode(constant("JSON_UNESCAPED_SLASHES"))|e("html_attr") }}"
    data-delete-url-template="{{ path('adventure_photo_delete', { adventureId: '_AID_', photoId: '_PID_' }) }}"
  </div>
  {% set isAchieved = adventure.status.value == 'achieved' %}
  {% set hasPictures = pictures is not empty %}

  {% if isAchieved %}
    <div class="photo-section w100">
      {% if isOwner %}
        {% include 'user/adventures/partials/_photo-display-owner.html.twig' %}
      {% else %}
        {% include 'user/adventures/partials/_photo-display-viewer.html.twig' %}
      {% endif %}
    </div>
  {% else %}
    <div class="adventure-card-empty">
      <i class="fas fa-hiking"></i>
    </div>
  {% endif %}
</div>
```

Figure 30 : Extrait du template adventure.html.twig de User – Contrôles d'attributs

Pour anticiper une évolution vers Webpack encore, Vite ou un autre gestionnaire d'assets, **chaque script dispose de son propre fichier JavaScript**, et chaque page intègre dynamiquement dans un bloc `{% block javascripts %}` les scripts dont elle a besoin (hors menu du header ou autres scripts généralistes / comportements répétés dans tous le site). Ce choix favorise la **modularité** et limite les conflits de scripts.

Les données dynamiques sont injectées dans les vues Twig à partir des objets PHP ou des tableaux transmis par les contrôleurs. L'accès se fait simplement via la syntaxe `{} variable.attribut {}` pour les objets (ex : `{} user.email {}`) ou `{} tableau['clé'] {}` pour les structures de type tableau. Twig permet aussi des conditions d'affichage ou des boucles pour générer du contenu en fonction des données (ex : `{% for contact in contactList %}`). Cette approche rend l'interface réactive au contexte utilisateur (propriétaire, invité, rôle admin, etc.) tout en restant lisible et sécurisée.

```

99  {% for adventure in otherAdventures %}
100   <tr class="adventure-row" data-title="{{ adventure.title|lower }}" data-start="{{ adventure.startDate.getTimestamp }}" data-status="{{ adventure.status.value }}"
101     <td>{{ adventure.title }}</td>
102     <td>{{ adventure.status.value }}</td>
103     <td>{{ adventure.startDate|date('Y-m-d') }}</td>
104     <td>{{ adventure.endDate|date('Y-m-d') }}</td>
105     <td>
106       {% if adventure.timerAlert %}
107         <span>
108           | {{ adventure.timerAlert.alertTime|date('d/m/Y H:i') }}</span>
109       {% else %}
110         <span> Off</span>
111       {% endif %}
112     </td>
113   <td>
114     <a href="{{ path('adventure', {'id': adventure.id}) }}"
115       <i class="fas fa-pen"></i>
116     </a>
117   </td>
118   <td>
119     <a href="{{ path('delete-adventure', {'id': adventure.id}) }}" class="delete-adventure-btn">
120       <i class="fas fa-times"></i>
121     </a>
122   </td>
123 </tr>
124 {% endfor %}
125

```

Figure 31 : Exemple d'itération sur une liste d'objets dans la vue Adventure-manager

L'interactivité repose ensuite sur les scripts JavaScript, abordés dans la prochaine section.

2.2.2. Développement de la partie dynamique (Javascript, AJAX)

L'interactivité du front-end repose sur des scripts JavaScript dédiés, organisés par page ou par composant fonctionnel. Plutôt que de centraliser l'ensemble du JS dans un seul fichier, Pathfinding adopte une approche modulaire : chaque vue ou élément dynamique dispose de son propre fichier, ce qui améliore la lisibilité, la maintenabilité et permet d'anticiper une migration future vers un système de bundling plus évolué (Webpack, Vite...).

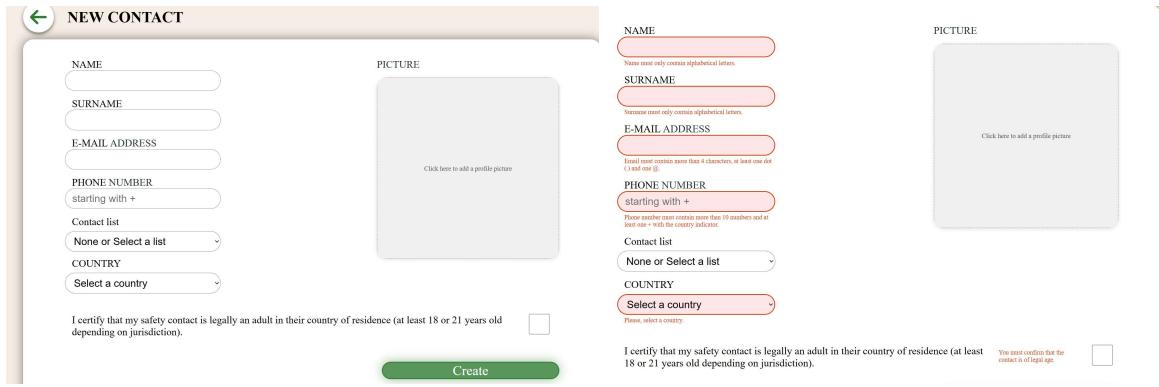
Le JavaScript est principalement utilisé pour :

- manipuler le DOM selon le rôle de l'utilisateur (afficher/masquer des éléments),
- valider des champs en direct (avant envoi du formulaire),
- améliorer l'UX (comptage de photos, countdown, animation légère),
- gérer les événements utilisateurs (clics, saisie, etc.).

Exemple 1 – Validation dynamique d'un formulaire

Un exemple courant est le formulaire de création de contact. Avant l'envoi, plusieurs vérifications sont effectuées côté client : nom et prénom non vides (et composés uniquement de lettres), téléphone au format international (+XXX), email valide, pays correctement sélectionné, et case “majeur” cochée. Ces vérifications évitent à l'utilisateur des erreurs frustrantes une fois le formulaire soumis.

Cette logique côté client doit impérativement rester **cohérente avec les contraintes de validation côté Back-end** (Assert Symfony) et le **format attendu en base de données**. Cela garantit non seulement l'intégrité des données, mais aussi une expérience utilisateur fluide et sans surprise.



```

public > <js> > <validationForms> ...
1  // Vérifie si tous les champs sont remplis
2  const testFormBeforeSubmit = document.getElementById('submitBtn')
3  const form = document.querySelector('form')
4  testFormBeforeSubmit.addEventListener("click", function(event) {
5      event.preventDefault(); // Empêche l'envoi du formulaire si la validation échoue
6      if (ValidateForm()) {
7          form.submit() // Envoie le formulaire après validation réussie
8      }
9  })
10, | 
11, | //check que tous les champs sont remplis et true/
12, | function ValidateForm() {
13, |     const nameField = document.getElementById('name');
14, |     const surnameField = document.getElementById('surname');
15, |     const emailField = document.getElementById('email');
16, |     const phoneField = document.getElementById('phone');
17, |     const countryField = document.getElementById('country');
18, |     const majorityCheckbox = document.getElementById('declarationOfMajority'); // Peut être null
19, | 
20, |     // Nettoyer les anciennes erreurs
21, |     [nameField, surnameField, emailField, phoneField, countryField].forEach(field => {
22, |         field.classList.remove('field-error');
23, |         const existingError = field.parentElement.querySelector('.error-message');
24, |         if (existingError) existingError.remove();
25, |     });
26, | 
27, |     let isValid = ValidateInputs(nameField, surnameField, emailField, phoneField, countryField);
28, | 
29, |     // Vérification spécifique pour la création uniquement
30, |     if (majorityCheckbox && !majorityCheckbox.checked) {
31, |         const label = document.querySelector('label[for="declarationOfMajority"]');
32, |         if (label && label.nextElementSibling.classList.contains('error-message')) {
33, |             const error = document.createElement('div');
34, |             error.textContent = "You must confirm that the contact is of legal age.";
35, |             error.style.color = "#00431B";
36, |             label.insertAdjacentElement('afterend', error);
37, |             isValid = false;
38, |         }
39, |     }
40, | 
41, |     return isValid;
42, | }
43, | 
44, | return isValid;
45, | }

```

```

47  /* Check all the inputs */
48  function ValidateInputs(surnameField, nameField, emailField, phoneField, countryField) {
49      let isValid = true;
50      ...
51      const nameRegex = /^[a-zA-Z]+$/;
52      const surnameRegex = /^[a-zA-Z]+$/;
53      const phoneRegex = /^[0-9]{10,15}+$/;
54      ...
55      if (!nameRegex.test(surnameField.value)) {
56          if ("Surname must only contain alphabetical letters.", surnameField);
57          isValid = false;
58      }
59      ...
60      if (!nameRegex.test(nameField.value)) {
61          if ("Name must only contain alphabetical letters.", nameField);
62          isValid = false;
63      }
64      ...
65      if (!emailRegex.test(emailField.value) || emailField.value.length <= 4) {
66          if ("Email must contain more than 4 characters, at least one dot (.) and one @.", emailField);
67          isValid = false;
68      }
69      ...
70      if (!phoneRegex.test(phoneField.value) || phoneField.value.length <= 10) {
71          if ("Phone number must contain more than 10 numbers and at least one + with the country indicator.", phoneField);
72          isValid = false;
73      }
74      ...
75      if (countryField.value === "" || countryField.value === "default") {
76          if ("Please, select a country.", countryField);
77          isValid = false;
78      }
79      ...
80      return isValid;
81  }
82  ...
83  /* Centralise errors */
84  function fail(message, field) {
85      console.log("Validation failed: " + message);
86      field.classList.add('field-error');
87      // Supprimer un ancien message s'il existe
88      const existingError = field.parentElement.querySelector('.error-message');
89      if (existingError) {
90          existingError.remove();
91      }
92  }

```

Figure 32 : Exemple de retour d'erreurs via Javascript sur la création de contact

Exemple 2 – Rechargement AJAX des contacts

L'autre pilier de l'interactivité repose sur des appels **AJAX** pour recharger dynamiquement des portions de page sans rechargement complet. C'est le cas pour la liste des contacts de sécurité : selon la liste sélectionnée ou le tri choisi, un appel est envoyé au back-end (via une route dédiée), qui retourne un fragment HTML mis à jour.



```

147 const ajaxUrl = "{{ path('contacts-ajax') }}";
148 const listSelector = document.getElementById('list-selector');
149 const sortAzButton = document.querySelector('.sort-az');
150 const sortRecentButton = document.querySelector('.sort-recent');
151 let currentSort = 'az';
152
153 function loadSafetyList(listId, sort = 'az') {
154     fetch(` ${ajaxUrl}?list=${listId}&sort=${sort}`)
155         .then(res => res.text())
156         .then(html => {
157             document.getElementById('safety-contact-table').outerHTML = html;
158             // Réappliquer le filtre après recharge
159             setupSearchBar('search-safety-contacts', 'clear-search-safety', '#safety-contact-table tr.non-member');
160         });
161 }

```

Figure 33 : Changement dynamique de l'affiche des membres d'une liste en AJAX

L'appel `fetch()` contacte une route Symfony (`contacts-ajax`) qui retourne un fragment Twig (`_contact-table.html.twig`) mis à jour selon les paramètres passés (`list, sort`). Après chargement, le DOM est remplacé, et les filtres (recherche) sont réappliqués.

Filtrage dynamique côté front

Une barre de recherche permet également de filtrer les contacts en temps réel, selon le prénom ou le nom. Les contacts de la liste sélectionnée s'affichent en permanence quelque soit le résultat de la recherche pour aider l'utilisateur à sélectionner facilement les membres de sa liste.

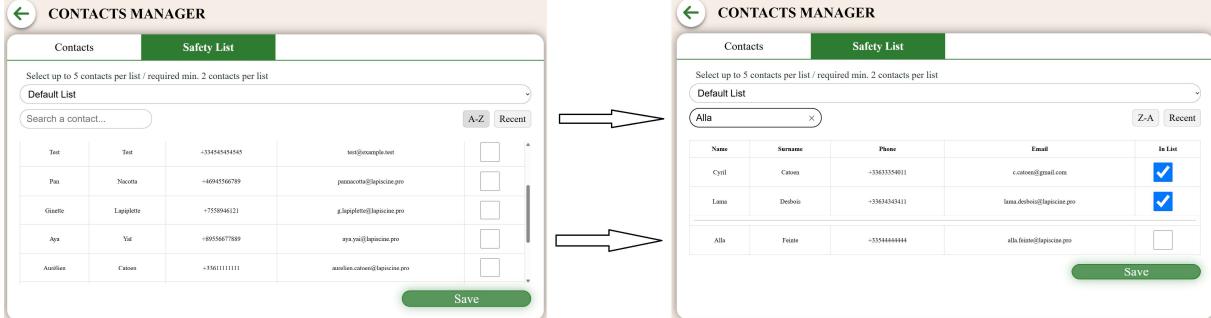


Figure 34 : Recherche dynamique par nom dans une liste de contacts

Ce script correspondant à cet affichage (cf. figure 34) est présenté page suivante (cf. figure 35). Il est générique pour pouvoir être réutilisé sur plusieurs pages en modifiant les dernières lignes relatives à l'événement et au DOM. Il montre comment la logique côté client peut renforcer les performances perçues de l'application.

```

function setupearchBar(inputId, clearId, rowSelector) {
    const input = document.getElementById(inputId);
    const clear = document.getElementById(clearId);

    if (!input || !clear) return;

    function filterRows() {
        const val = input.value.toLowerCase();
        clear.style.display = val ? 'inline' : 'none';

        document.querySelectorAll(rowSelector).forEach(row => {
            const name = row.cells[0] ? row.cells[0].textContent.toLowerCase() || '';
            const surname = row.cells[1] ? row.cells[1].textContent.toLowerCase() || '';
            row.style.display = (name.includes(val) || surname.includes(val)) ? '' : 'none';
        });
    }

    input.addEventListener('input', filterRows);

    clear.addEventListener('click', () => {
        input.value = '';
        clear.style.display = 'none';
        document.querySelectorAll(rowSelector).forEach(row => {
            row.style.display = '';
        });
    });
}

filterRows(); // Appel initial
}

window.setupearchBar = setupearchBar;

document.addEventListener('DOMContentLoaded', () => {
    setupearchBar('search-all-contacts', 'clear-search-all', '#tab-contacts tbody tr');
    setupearchBar('search-safety-contacts', 'clear-search-safety', '#safety-contact-table tr.non-member');
})

```

Figure 35 : Script de la barre de recherche de SafetyList dans la vue Contact-Manager

Cette approche combine les avantages de Twig (affichage serveur rapide et structuré) avec la souplesse du JavaScript pour des interactions fluides et modernes.

Au-delà de l'interactivité, une application telle que Pathfinding doit veiller à **protéger les données selon le rôle de l'utilisateur et ses droits d'accès**. Certaines pages, certains composants ou éléments d'interface ne doivent être visibles que dans des contextes bien précis. Cette gestion granulaire est abordée dans la section suivante.

2.2.3. Gestion des droits et visibilité des données utilisateur

Les règles d'accès définies côté serveur (voir section 2.1.5) sont relayées en front pour garantir une interface cohérente, sécurisée et lisible, adaptée au profil connecté (visiteur, utilisateur « propriétaire » ou non, administrateur).

En Twig, l'affichage des blocs est conditionné par des instructions logiques telles que :

`{% if isOwner %}`

Contenu visible uniquement par le propriétaire de la page/vue affichée

`{% endif %}`

Un exemple concret a été présenté dans la figure 30 (section 2.2.1), illustrant l'usage combiné de tests logiques et d'attributs pour ajuster dynamiquement le rendu.

Cette approche permet de masquer totalement les éléments non autorisés (invisibles dans le DOM, même via l'inspecteur). Ce filtrage visuel repose également sur les attributs des entités (statut d'une aventure, niveau de visibilité, etc.) pour affiner l'accès à chaque composant.

Les vues sont sensibles à l'état des contenus. Par exemple :

- Une aventure marquée **achieved** déclenche l'affichage d'un carrousel photo,
- Une aventure **ongoing** propose un formulaire d'édition.

Ce comportement repose sur :

- des tests conditionnels Twig (if adventure.status.value == 'achieved'),
- des classes CSS injectées dynamiquement (en fonction des rôles, statuts ou événements JS),
- des composants partiels différenciés (ex. : _photo-display-owner.html.twig vs _photo-display-viewer.html.twig).

Côté JavaScript, certaines interactions (ex. suppression de photo, modification de données) ne sont accessibles que si l'utilisateur est le propriétaire de l'objet concerné.

Cette gestion homogène des droits côté client contribue à une meilleure ergonomie, tout en limitant les comportements non prévus – même sans recharge de page.

La section suivante approfondit l'aspect visuel et adaptatif des interfaces utilisateurs.

2.2.4. CSS/SCSS avancé et Responsive Design

Le style visuel de Pathfinding repose sur une architecture SCSS organisée selon le modèle **6:1**, inspiré du standard 7:1. Elle sera amenée à évoluer en **7:1** avec l'ajout d'un dossier theme/ dédié à la gestion du mode clair/sombre. Aujourd'hui, l'arborescence comprend :

- abstracts/ : **variables**, **mixins**, **fonctions** réutilisables
- base/ : règles de **reset**, **typographie**, et style de base HTML
- components/ : composants visuels réutilisables (boutons, cartes, formulaires...)
- layout/ : structure globale des vues (header, footer, containers...)
- pages/ : style spécifique à chaque page
- vendors/ : librairies externes

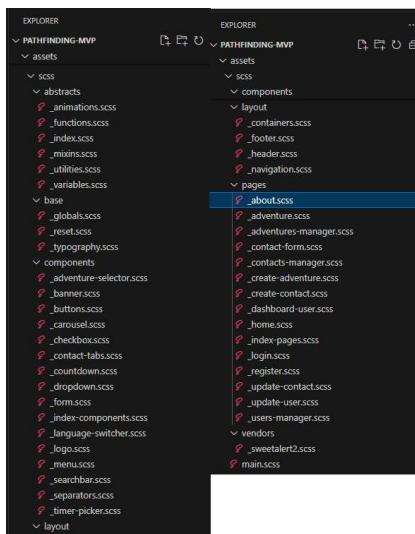


Figure 36 : Arborescence et architecture 6 : 1 du SCSS dans Pathfinding

Un fichier `_index-xxx.scss` dans chaque dossier, de plus de 4 styles `_xxx.scss`, **forward** ses sous-fichiers, et tous les index sont importés dans `assets/scss/main.scss`, **seul fichier SCSS compilé** vers `public/css/main.css`.

```

assets > scss > main.scss
1 // Import bases
2 @use 'abstracts/index' as *;
3 // @use 'abstracts/variables' as var;
4 // @use 'abstracts/mixins' as mix;
5 @use 'base/reset';
6 @use 'base/globals';
7 @use 'base/typography';
8
9 // Layout global
10 @use 'layout/header';
11 @use 'layout/footer';
12 @use 'layout/navigation';
13 @use 'layout/containers';
14
15 // Composants
16 @use 'components/index-components';
17
18 // Pages
19 @use 'pages/index-pages'; // Page d'accueil
20
21 // Vendors
22 @use 'vendors/sweetalert2'; // Page d'accueil
23

```



```

assets > scss > components > _index-components.scss
1 @forward 'adventure-selector';
2 @forward 'banner';
3 @forward 'buttons';
4 @forward 'carousel';
5 @forward 'checkbox';
6 @forward 'contact-tabs';
7 @forward 'countdown';
8 @forward 'dropdown';
9 @forward 'form';
10 @forward 'language-switcher';
11 @forward 'logo';
12 @forward 'menu';
13 @forward 'searchbar';
14 @forward 'separators';
15 @forward 'timer-picker';

```

Figure 37 : Emploi de `@use` et `@forward` pour importer les styles dans `main.scss`

Cette structure, simple et scalable, facilite la maintenance en séparant clairement les responsabilités.

Le code utilise plusieurs fonctionnalités puissantes de SCSS :

- **Imbrication et spécificité des sélecteurs** : balises, classes, ID
- **Variables \$** : palette de couleurs, tailles typographiques, espacements standardisés
- **Mixins** : blocs de style paramétrables utilisés via `@include`
- **%classe et @extend** : héritage de styles partagés
- **Boucles (@for)** : génération automatique des classes `.w100`, `.w95`, `.w90`, etc. par pas de 5 %
- **Animations CSS (@keyframes), hover, transform/transition** : effet visuel se déclenchant automatiquement suivant un évènement donné ou de manière cyclique (exemple « pulse »)
- **Unités et calculs dynamiques** : rem, calc(), clamp(), vw, vh, % — pour adapter les éléments aux tailles d'écran

Les paragraphes ci-après apportent des précisions complémentaires sur l'emploi de certaines fonctionnalités SCSS.

Spécificité des sélecteurs : classes, ID, balises et bonnes pratiques

En CSS/SCSS, la spécificité détermine quelle règle prévaut quand plusieurs ciblent le même élément. Le projet Pathfinding suit des conventions strictes pour assurer un style cohérent et maintenable :

- Classes (`.btn`, `.main-dashboard`) : largement utilisées pour styliser les composants. Elles sont réutilisables et modulaires.
- IDs (`#sliderBall`, `#navID`) : réservés aux éléments uniques et aux interactions JavaScript ciblées. Évités dans les styles SCSS sauf exception. Ils peuvent permettre parfois un ciblage plus aisés d'un élément HTML à styliser.

- Balises HTML (h1, p, a) : stylisées globalement dans les fichiers base/typography.scss, avec des tailles de police responsives via clamp().

Des sélecteurs imbriqués sont utilisés avec prudence pour garder une bonne lisibilité :

```
3 main.contact-form {  
4  
5     section{  
6         min-height: 100vh;  
7  
8         .button-type-4{  
9             width: 40%;  
10            @include button-color-shadow($button-border-green, $button-green, 0 0 24px, $box-shadow-green);  
11            transition: scale 0.1s ease-out;  
12  
13            &:hover{  
14                @include button-color-shadow($button-border-blue, $button-blue, 0 0 24px, $box-shadow-blue);  
15                scale: 1.1;  
16                transition: scale 0.2s ease-in;  
17            }  
18        }  
19    }  
20}
```

Figure 38 : Exemple d'imbrication avec balise, classe, mixin, variables et & :hover

Pour éviter les conflits ou une surspécification, le projet évite le !important, respecte une hiérarchie claire dans les feuilles de styles et limite l'imbrication à 4 niveaux maximums.

Positionnement et héritage de style (@extend)

Dans certains composants dynamiques de Pathfinding (menus déroulants, sliders, éléments masqués/repositionnés), le positionnement relative et absolute est crucial. Il permet d'ancrer un élément par rapport à son conteneur ou à la fenêtre, pour gérer son apparition/disparition fluide.

Par exemple :

- Les dropdowns du header admin sont en absolute, ancrés à un parent relative.

Figure 39 : Exemple d'utilisation de positionnement, variables et imbrication

- Le slider d'animation sur certains boutons utilise left: -100% ou transform combiné à position.

```

9  /* Styles pour le bouton */
0  .toggle-btn {
1    display: inline-block;
2    position: relative;
3    cursor: pointer;
4  }
5
6  .toggle-btn input[type="checkbox"] {
7    display: none;
8  }
9
0  .slider {
1    position: relative;
2    background-color: #red;
3    border-radius: 2.5vh;
4    width: 9vh;
5    height: 2.5vh;
6    transition: background-color 0.3s ease;
7    overflow: hidden;
8  }
9
0  .ball {
1    position: absolute;
2    top: 0;
3    width: 2.5vh;
4    height: 2.5vh;
5    background-color: #white;
6    border-radius: 50%;
7    transition: transform 0.3s ease;
8  }
9
0

```

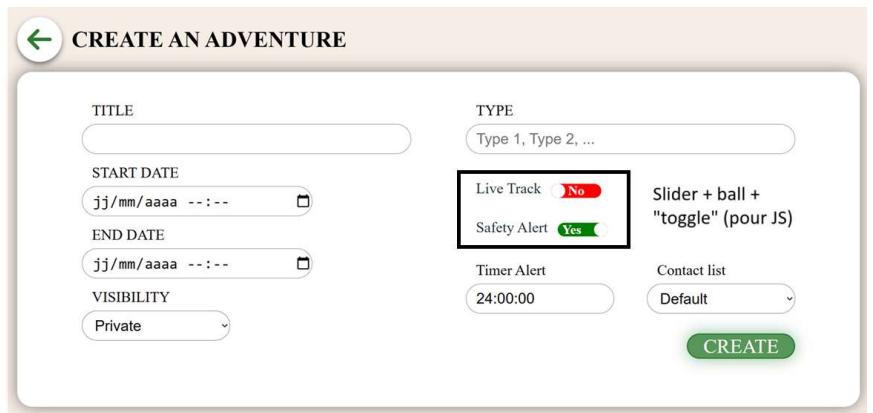


Figure 40 : Exemple d'animation employant Javascript et SCSS

D'autre part, SCSS permet d'hériter de règles CSS communes via @extend. Ainsi, une classe de type :

```

8  %container-commun-shadow{
9    border-radius: 25px;
0    background-color: $white-color;
1    box-shadow: 0 0 24px $box-shadow-black;
2

```

Figure 41 : Exemple de classe type

... est utilisée dans :

```

2  .container-type-1 {
3    @include container-shape($size: 20vw, $margin: 0, $shape: rectangle-vertic);
4    @include flex($direction: column);
5    @extend %container-commun-shadow;
6

```

Figure 42 : Exemple d'emploi de @extend dans une classe

Cela garantit un design cohérent tout en évitant les répétitions de code.

Ce fonctionnement s'inscrit dans la **logique DRY** (Don't Repeat Yourself), également appliquée à l'architecture SCSS du projet.

rem : unité responsive

Contrairement au px fixe, **1rem dépend de la taille de police définie à la racine (html)**, souvent 16px par défaut. En ajustant dynamiquement cette taille (via media queries ou accessibilité), tout le design s'adapte proportionnellement : c'est une base de **design scalable**.

calc() et clamp()

- calc()** permet de combiner dynamiquement des unités : width: calc(100% - 2rem); ajuste la largeur en fonction de l'espace dispo.
- clamp()** fixe une valeur **entre un minimum, une valeur idéale, et un maximum** : font-size: clamp(1rem, 2vw, 2.5rem); s'adapte automatiquement à la largeur de l'écran, **sans media query**.

```

7  a, span, p {
8    font-size: clamp(1rem, 2.5vw, 1.5rem); // Progression entre 1rem (16px) et 1.5rem (24px)
9  }

```

Figure 43 : Exemple d'emploi de clamp(), rem et vw dans _typography.scss

vw, vh et %

Les unités vw (viewport width), vh (viewport height) et le % (pourcentage) jouent également un rôle clé dans l'adaptabilité de l'interface. Contrairement aux unités fixes comme px, elles permettent à chaque élément de s'ajuster dynamiquement à la taille de l'écran.

- 1vw représente 1 % de la largeur de la fenêtre (viewport),
- 1vh représente 1 % de la hauteur visible de cette même fenêtre,
- % s'applique par rapport au parent direct de l'élément.

Dans Pathfinding, ces unités sont utilisées pour :

- Dimensionner des blocs responsives sans media queries, comme les cartes ou les conteneurs d'aperçu (width: 90vw, height: 80vh, etc.),
- Créer des formes proportionnelles avec le mixin container-shape() basé sur des vh,
- Adapter les espacements ou zones de scroll sur des sections verticales (% ou vh sur height),
- Faire apparaître/disparaître dynamiquement des composants (ex : dropdowns ou sliders qui sortent littéralement du viewport avec top: 100vh ou left: -150vw).

Combinées à clamp() avec des valeurs exprimées min/max exprimés en px ou rem ou à des calc() personnalisés, ces unités permettent une gestion fluide, cohérente et efficace du responsive sans dépendre uniquement de points de rupture.

Responsive Design

Le responsive repose à la fois sur l'emploi d'unités naturellement responsive (vh, vw, %), de calculs dynamiques avec les fonctions calc() et clamp(), de conditions dans Twig et de **media queries** standardisées avec l'utilisation de variable \$breakpoint. Exemple dans le footer :

```

assets > scss > layout > _footer.scss > ...
108 .social-links {
126   h3:nth-of-type(2){
127     font-size: clamp(2rem, 7.5vw, 3rem);
128     text-align: center;
129     @media (min-width: calc($breakpoint-sm + 1px)) {
130       display: none;
131     }
132   }

```

Figure 44 : Exemple d'emploi de @media, calc() et \$breakpoint-sm dans le footer

Le titre deuxième titre h3 dans le parent comportant la classe .social-links disparaît à partir d'une taille d'écran minimal calculé de \$breakpoint-sm + 1px, soit $578 + 1 = 579$ px. C'est-à-dire que cette information sera visible sur tous les mobiles (578px correspond aux plus grands smartphones) et disparaîtra sur tablettes et ordinateurs.

La figure 45 page suivante illustre le résultat visuel de cette manipulation avec @media sur h3 :nth-of-type(2).

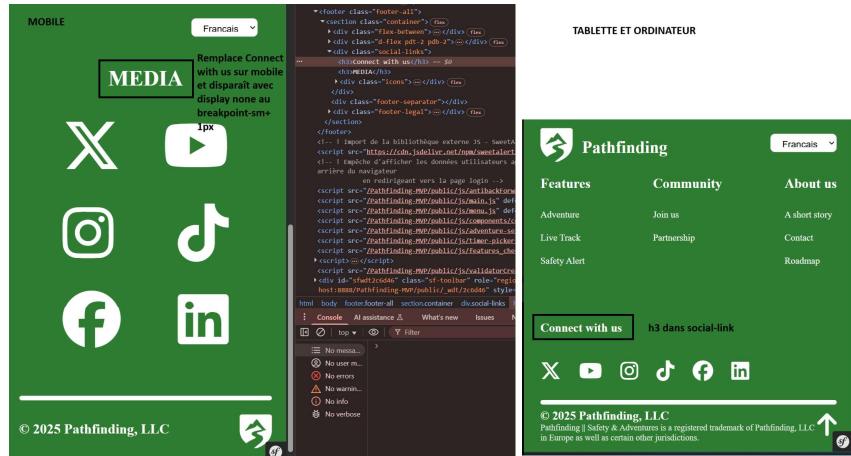


Figure 45 : Exemple de design responsive mobile/tablette/ordinateur avec le footer

Chaque main possède une **classe unique** (ex : main-dashboard, main-profile) permettant de cibler précisément les styles d'une page, tout en conservant une **base CSS commune minimale**.

Classes utilitaires - Intérêts

Une attention particulière est portée à l'usage des **classes utilitaires** : il s'agit de classes à comportement **unique, simple et explicite** (par exemple .d-flex, .gap-4, .w100). Leur but est de **faciliter l'ajout ponctuel de règles CSS courantes** directement dans les balises HTML, **sans devoir créer une classe dédiée** lorsqu'il n'y aurait qu'un seul cas d'utilisation.

```
assets > scss > abstracts > _utilities.scss ...
1 // Margin + Padding + Gap helpers : m = margin ; pd = padding ; gap = gap
2 // t = top ; r = right ; b = bottom ; l = left
3 @for $i from 1 through 8 [
4   .mt-#{$i} { margin-top: $i * 0.5rem; }
5   .mb-#{$i} { margin-bottom: $i * 0.5rem; }
6   .ml-#{$i} { margin-left: $i * 0.5rem; }
7   .mr-#{$i} { margin-right: $i * 0.5rem; }
8   .pdt-#{$i} { padding-top: $i * 0.5rem; }
9   .pdb-#{$i} { padding-bottom: $i * 0.5rem; }
10  .pdl-#{$i} { padding-left: $i * 0.5rem; }
11  .pdr-#{$i} { padding-right: $i * 0.5rem; }
12  .gap-#{$i} { gap: $i * 0.5rem; }
13 ]
14
15 // Text align
16 @each $align in center, right, left, justify { // @each permet d'éviter de répéter sur plusieurs lignes .text-nomDirection
17   .text-#{$align} { text-align: $align; }

```

Figure 46 : Création de classes utilitaires (1 seul comportement) par boucles

Cela présente plusieurs avantages :

- Évite l'imbrication excessive dans les fichiers SCSS (qui pourrait générer des conflits de spécificité ou des effets de bord),
- Réduit la prolifération de classes uniques et donc la dette technique,
- Permet une lecture rapide du style appliqué à un composant, directement dans le template.

Par exemple, au lieu de créer une classe photo-row-spacing uniquement utilisée sur un bloc d'aperçu d'images, on applique simplement :

```
<div class="d-flex gap-4">
```

Cela exprime immédiatement le **comportement visuel attendu** (un conteneur en display : flex avec un gap de 2rem, ici gap-4, itération de 0.5rem) sans surcharge inutile.

```
templates > user > profile.html.twig > ...
7  {% block main %}
8  <main class="profile">
9    <section class="container">
10      <form method="post" action="{{ path('user_profile') }}" class="mt-8 mb-8" enctype="multipart/form-data">
11        <input type="hidden" name="_csrf_token" value="{{ csrf_token('edit-profile') }}>
12
13        <div class="flex-1 l-gap-3">
14          <div class="w100 flex-center gap-1">
15            <div class="w45">
16              {% set pictureUrl = asset('uploads/user-picture/' ~ app.user.picturePath)|e('css') %}
17              <label for="picture-profile" id="uploadLabel" class="flex-center container-circle-1">
18                {% if app.user.picturePath %}
19                  style="background-image: url('{{ pictureUrl }}');"
20                {% endif %}
21                {% if not app.user.picturePath %}
22                  Click here to add a profile picture
23                {% endif %}
24              </label>
25              <input type="file" id="picture-profile" name="picture-profile" class="upload-input">
26            </div>
27            <div class="d-flex gap-2 w45">
28              <div class="w45">
```

Figure 47 : Exemple d'application de classes utilitaires dans profile.html.twig

Ce principe rejoint une logique utilitaire, modulaire et pragmatique, parfaitement adaptée à un projet comme Pathfinding qui cherche à rester lisible, réactif et maintenable — aussi bien dans son style que dans ses interactions.

2.2.5. Emploi de librairies externes

Plusieurs bibliothèques front-end ont été intégrées pour améliorer l'ergonomie, l'interactivité et la richesse visuelle de l'interface :

-  **Font Awesome** : utilisée pour insérer des icônes (ex : boutons, liens, navigation mobile).

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.0/css/all.min.css" />
```



Figure 48 : Utilisation d'icônes font-awesome pour sélectionner le type d'aventure

-  **Leaflet.js** : pour afficher des **cartes topographiques interactives**, notamment les traces GPX des aventures. Compatible avec des fonds OpenStreetMap.

```
L.map('map').setView([lat, lon], zoom);
```

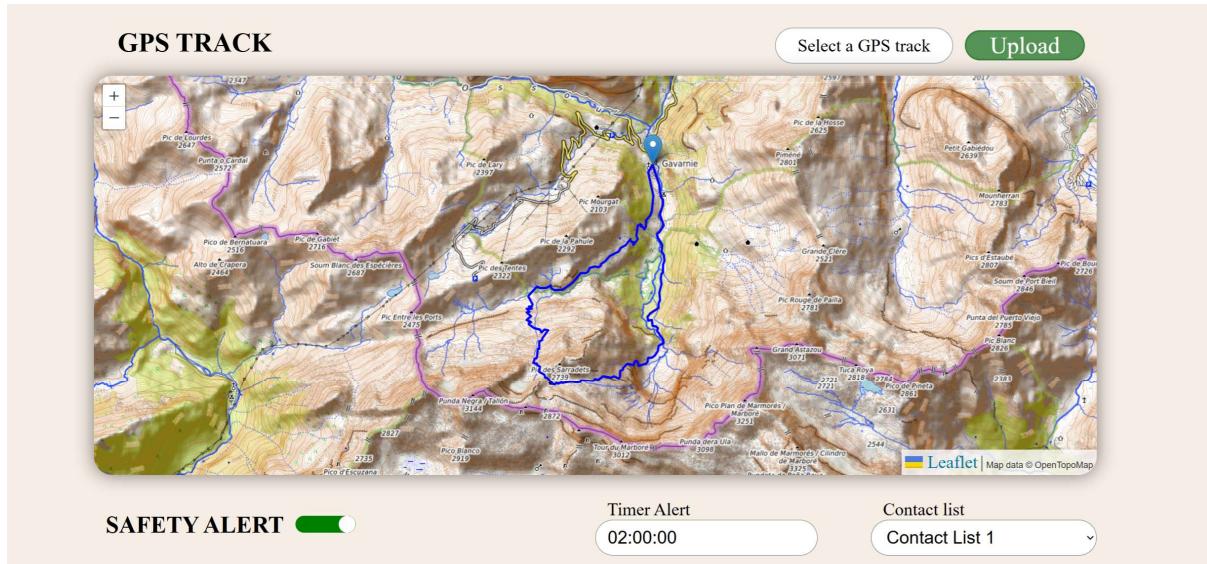


Figure 49 : Utilisation de Leaflet pour afficher l'itinéraire suivi lors d'une aventure

- **SweetAlert2** : permet d'afficher des **popups personnalisées** (confirmation de suppression, erreur de validation, messages de succès) avec un rendu UX moderne.

```
Swal.fire('Suppression confirmée !', 'Le contact a été supprimé.', 'success');
```

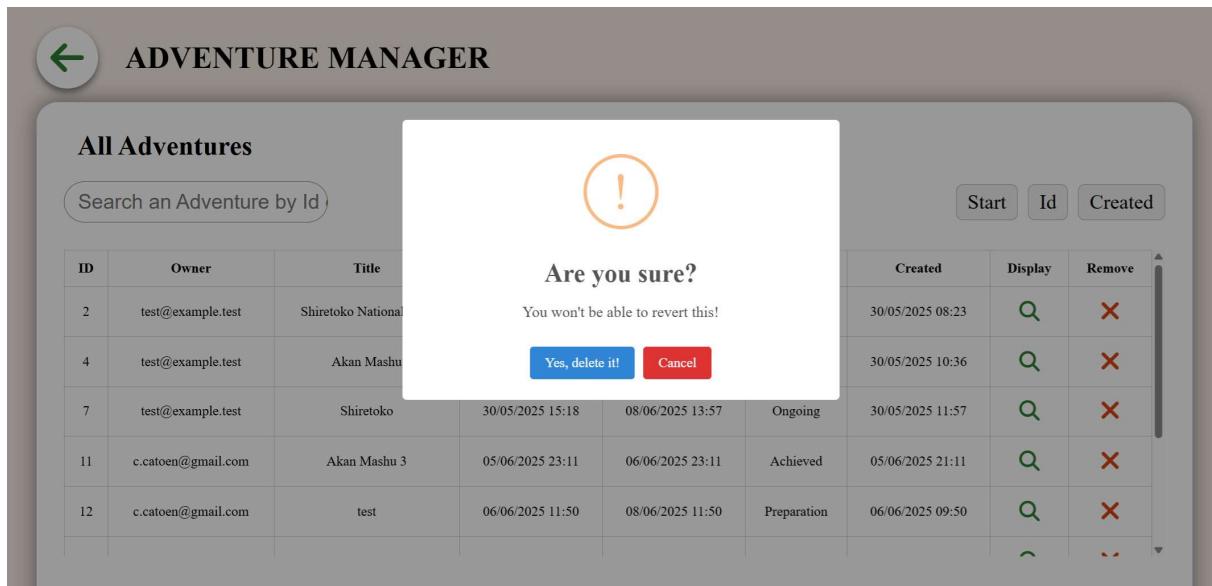


Figure 50 : Utilisation de SweetAlert2 pour les alertes et pop-up – exemple du remove

Ces librairies sont appelées **via CDN** dans le layout ou intégrées à des fichiers JS spécifiques, selon leur usage.

2.3. Documentation et déploiement

Au-delà du développement fonctionnel et visuel, le projet Pathfinding nécessite la mise en place d'une documentation claire ainsi qu'une stratégie prévisionnelle de déploiement adaptée. Cette section décrit les outils utilisés pour documenter le code, faciliter la maintenance, et assurer une mise en ligne fluide, testée et évolutive.

2.3.1. Procédure de déploiement (prévisionnelle)

Bien que le déploiement réel ne soit pas requis dans le cadre de la formation, j'ai anticipé cette étape en définissant une **procédure de mise en production** de l'application Symfony, basée sur les bonnes pratiques actuelles.

Le déploiement de Pathfinding suivrait les grandes étapes suivantes :

- **Choix d'un hébergement** : serveur mutualisé, VPS ou dédié selon les ressources nécessaires et les contraintes de sécurité. À terme, un **VPS** ou un **hébergement type OVH Pro** serait préféré pour plus de flexibilité.
- **Nom de domaine personnalisé** (ex : pathfinding.com) relié au serveur via les enregistrements DNS.
- **Connexion SSH** au serveur distant pour exécuter les commandes de déploiement.
- **Clonage du projet** depuis le dépôt GitHub :
>> git clone https://github.com/mon-utilisateur/pathfinding.git
- **Configuration de l'environnement de production** via un fichier .env.local contenant les identifiants de la base de données, le mailer, etc.
- **Installation des dépendances Symfony** :
>> composer install
- **Migration de la base de données** :
>> php bin/console doctrine:migrations:migrate
- **Nettoyage du cache Symfony** et installation des assets :
>> php bin/console cache:clear
>> php bin/console assets:install public

Ces étapes garantissent que l'application est fonctionnelle et prête à être servie aux utilisateurs en environnement de production.

Note : la connexion SSH (Secure Shell) est indispensable pour sécuriser les échanges et manipuler le serveur distant à distance. Ce protocole est couramment utilisé pour les déploiements Symfony.

2.3.2. Recherche et exploitation de ressources techniques anglophones

Le développement du projet Pathfinding s'est appuyé à plusieurs reprises sur des ressources techniques anglophones, issues de documentations officielles ou de plateformes de référence. Ces recherches ont permis de résoudre des problématiques spécifiques ou d'implémenter des solutions plus propres et maintenables. Trois exemples illustrent cette démarche.

a) Utilisation des data-attributes pour la mise à jour dynamique par AJAX

La documentation officielle de Symfony recommande l'usage de data-attributes pour faire circuler des données entre le front-end et le back-end de manière fluide, notamment via JavaScript. Le site symfony.com/doc/current/frontend/server-data.html a été une ressource déterminante.

Dans Pathfinding, ces attributs sont utilisés pour enrichir les blocs HTML avec des métadonnées invisibles, réutilisées côté JS dans les appels `fetch()` ou les mises à jour DOM. Cela concerne :

- Le nombre de photos (`data-uploaded-count`),
- Le stockage en front-end de toutes les métadonnées nécessaires sur les fichiers uploadés d'une aventure, au format JSON (`data-uploaded-files`),
- Le chemin d'accès des photos (`data-pictures`),
- Les URL pour les routes à appeler avec `fetch` (`data-update-url`, `data-timer-url`, ...)
- Les coordonnées GPS ou les fichiers partagés (`data-points`, `data-track`),
- Les identifiants id (`data-adventure-id`, `data`),
- Etc.

```
<div class="uploaded-files" id="uploaded-files-container" data-uploaded-files='{{ adventure.adventureFiles|map(f => {
    id: f.id,
    name: f.fileName,
    type: f.type.value,
    typeLabel: f.type.label(),
    typeIcon: f.type.icon(),
    size: f.size,
    url: path("download_adventure_file", { id: f.id })
})|json_encode|e("html_attr") }}'>
```

Figure 51 : Utilisation de `data-attribute` pour les données des documents d'aventure

Chaque champ modifié côté utilisateur déclenche un appel AJAX vers une route Symfony, mettant à jour les données en base sans rechargement de page.

b) Usage du tag `set` dans Twig pour optimiser le rendu conditionnel

La documentation Twig (twig.symfony.com/doc/3.x/tags/set.html) a été utilisée pour factoriser certaines vues complexes. Par exemple, dans Pathfinding, le tag `set` est utilisé pour **stocker temporairement une valeur calculée**, afin d'éviter de répéter une logique conditionnelle dans le template, tout en améliorant la lisibilité.

Un exemple typique est celui du traitement de l'image de profil utilisateur :

```
{% set pictureUrl = user.picturePath
? asset('uploads/user-picture/' ~ user.picturePath)|e('css') : null %}
```

L'objectif est de déterminer dynamiquement si une photo de profil a été uploadée, et dans ce cas, construire l'URL publique de cette image à partir de son nom de fichier.

Si `user.picturePath` existe, on utilise `asset()` pour construire l'URL complète (ex : `/uploads/user-picture/avatar123.jpg`). Sinon, on assigne `null` pour afficher un message alternatif

Le résultat est ensuite **réutilisé dans la balise `<label>`** pour appliquer un fond conditionnel (cf. figure 52, page suivante).

```

51
52
53
54
55
56
57
58
59
60
61
    <label for="picture-user">PICTURE</label>
    <div class="adventure-card">
        [% set pictureUrl = user.picturePath ? asset('uploads/user-picture/' ~ user.picturePath)|e('css') : null %]
        <label for="picture-user" id="uploadLabel" class="upload-label">
            [% if pictureUrl %]
                style="background-image: url('{{ pictureUrl }}');"
            [% endif %]
            [% if not pictureUrl %]
                Click here to add a profile picture
            [% endif %]
        </label>
    </div>

```

Figure 52 : Utilisation de set pour construire l'url de la photo de profile de l'utilisateur

Sans set, il aurait fallu **répéter** plusieurs fois la même logique asset(...) dans les blocs if, ce qui alourdit le code et rend l'ensemble moins maintenable.

c) Application de la propriété CSS perspective pour le carrousel photo

Pour améliorer l'expérience visuelle sur la galerie d'images d'une aventure, la propriété CSS perspective (documentée ici : [w3schools.com/CSSref/css3_pr_perspective.php](https://www.w3schools.com/CSSref/css3_pr_perspective.php)) a été exploitée pour créer un **effet 3D fluide et immersif**.

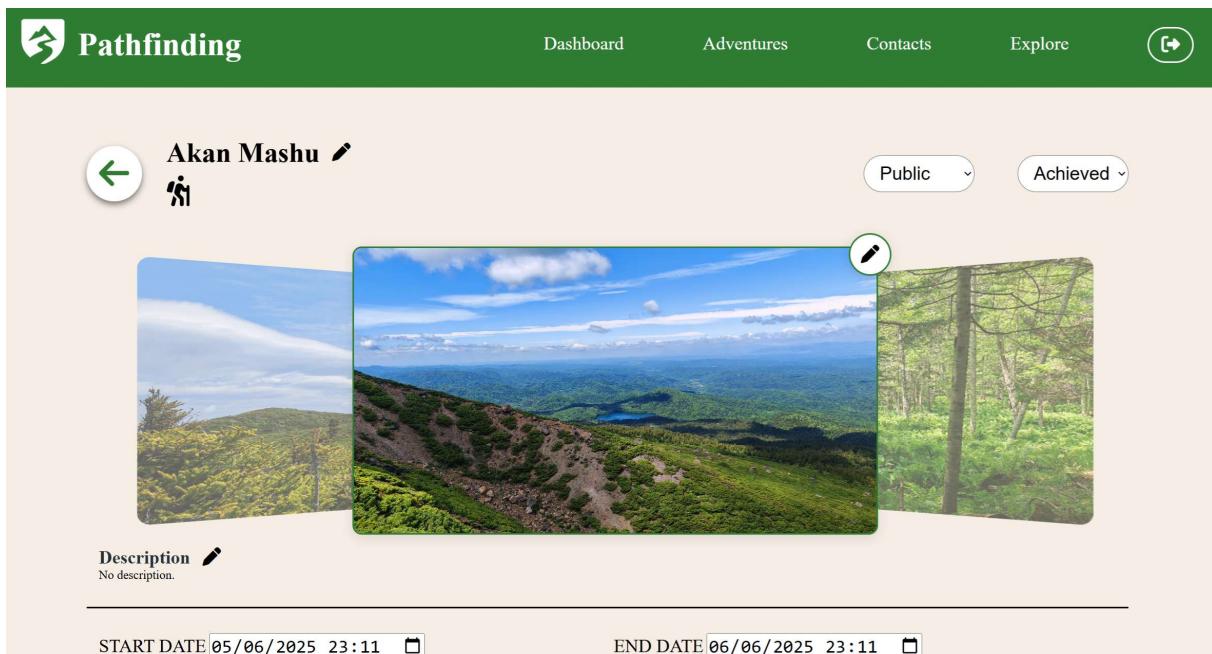


Figure 53 : Résultat visuel obtenu avec l'utilisation de perspective sur le carrousel

L'image centrale est placée au premier plan, tandis que les images précédente et suivante sont légèrement pivotées avec un transform: rotateY() et un translateZ() inverse. La propriété perspective: 1000px appliquée sur le conteneur donne cette illusion de profondeur et renforce l'attractivité du carrousel, tout en restant 100 % CSS (aucun plugin externe nécessaire).

2.4. Bilan et perspectives d'évolution

Le développement de Pathfinding sur ces deux derniers mois a permis de valider l'ensemble des fondations techniques du projet, tant sur le plan back-end que front-end. Ce socle robuste s'appuie sur des technologies modernes, une architecture modulaire, et une gestion fine de l'expérience utilisateur.

Au-delà du périmètre initial du MVP, plusieurs pistes d'évolution ont été identifiées pour enrichir l'application, renforcer sa scalabilité et améliorer ses fonctionnalités.

2.4.1. Axes d'amélioration

Même si Pathfinding est aujourd'hui pleinement fonctionnel dans sa version MVP, de nombreuses pistes d'amélioration ont été identifiées afin d'enrichir l'expérience utilisateur, de renforcer sa fiabilité et de le rendre plus attrayant pour un public plus large.

Optimisation de l'expérience utilisateur

Travailler avec un designer professionnel pour améliorer le rendu visuel général du site représente une évolution clé à court terme. L'objectif est de proposer une interface à la fois moderne, immersive et en accord avec l'univers outdoor du projet. En parallèle, une bêta fermée pourrait être ouverte pour recueillir des retours d'utilisateurs réels. Ces critiques constructives permettront d'identifier les points de friction, d'ajuster certains parcours utilisateurs (UX) et de prioriser les évolutions sur des bases concrètes.

Optimisation des performances

Certaines pages riches comme l'affichage d'une aventure complète (photos, documents, carte GPX, alertes) subissent parfois des ralentissements visibles, notamment lors de la manipulation de plusieurs fichiers ou de gros médias. Des messages de performance relevés dans la console navigateur signalent la nécessité :

- d'optimiser la gestion JS/CSS des éléments dynamiques (ex. modales, suppression en AJAX),
- de limiter la taille des images uploadées ou leur nombre par défaut affiché,
- d'implémenter du lazy-loading sur les composants visuels lourds.

Robustesse, sécurité et fiabilité

Le projet doit devenir un outil **complémentaire de sécurité** pour les aventuriers, et à ce titre, ne **peut se permettre ni panne ni faille critique**. Il est donc nécessaire :

- d'apprendre et d'implémenter les pratiques recommandées en audits de sécurité (OWASP, monitoring, alertes),
- d'anticiper les cas de défaillance serveur (timeout, accès concurrent, saturation),
- d'intégrer des tests unitaires et fonctionnels automatisés sur les flux critiques : création d'alerte, envoi d'email, partage sécurisé...

Dashboard & back-office évolué

Le module d'administration sera enrichi avec un ensemble d'exploitation de données permettant de :

- surveiller les **statistiques d'activité** : nombre d'utilisateurs actifs sur 24h, 7j, 30j ;
- consulter le **poids total des fichiers hébergés** (documents, photos) ;
- identifier les utilisateurs inactifs ou suspects ;
- purger ou archiver certaines données si besoin.

2.4.2. Perspectives d'évolution du projet

Pathfinding a été conçu pour **évoluer en profondeur** selon un schéma progressif et ambitieux. Plusieurs axes se dessinent déjà pour les mois à venir :

LiveTrack mobile (v1)

Développement d'une **application mobile légère (Flutter ou React Native)** permettant de partager sa position GPS automatiquement pendant une aventure en cours, à intervalles réguliers (par exemple toutes les 10–15 minutes). Basée sur le réseau GSM (2G suffisant), cette solution permettra une **sécurité améliorée sans matériel externe**, tout en économisant la batterie du téléphone.

Boîtier GPS autonome (v2) – prototype

Une version matérielle complémentaire est prévue : un **tracker GPS autonome** communiquant via GSM, LoRa, satellite, DVA selon la couverture. Objectifs :

- transmission de la trace même en zone blanche,
- autonomie jusqu'à **3 semaines**,
- **étanchéité IP67+**,
- **alerte manuelle ou détection intelligente d'alerte via capteurs (ex : accéléromètre)**,
- intégration directe avec la plateforme Pathfinding (alerte, visualisation). Ce boîtier pourra être développé en interne ou en partenariat avec un acteur hardware existant, et distribué via la plateforme.

Aventures collaboratives & Guide professionnel

Les aventures deviendront prochainement **collaboratives**, avec la possibilité d'inviter d'autres membres à une aventure existante, de créer des rôles (organisateur, participant, suiveur) et de partager des ressources, photos ou checkpoints.

Parallèlement, une **fonctionnalité “Guide professionnel”** permettra à certains profils certifiés de proposer des aventures **payantes** ou **semi-autonomes**, encadrées, dans des zones exigeantes (haute montagne, désert, jungle...).

Marketplace + vente d'équipements

Une **place de marché interne** viendra à terme proposer :

- du contenu premium (guides, itinéraires commentés),
- une **section e-commerce ciblée** sur les besoins réels : sécurité, autonomie, signalement, orientation avec un éventuel marché seconde main entre aventuriers.

Intégration avec autres systèmes

A moyen terme, Pathfinding aidera l'aventurier à organiser ses aventures avec l'exploitation de ressources externes, notamment : blog aventure, services météo, plugins externes.

Système influenceurs & découverte

Une section **Discover** mettra en avant les aventures de membres influents (créateurs outdoor, photographes, guides), pour inspirer les utilisateurs. Un espace leur sera dédié pour construire leur communauté via la plateforme. Attirer les influenceurs sur la plateforme participera à son adoption de masse (marketing naturel, bouches à oreilles).

Gamification légère

À partir de 3 000 utilisateurs actifs, un système de **badges**, **challenges saisonniers**, ou **points d'exploration** pourrait être intégré. L'objectif est de motiver la pratique régulière sans dénaturer l'aspect authentique et sécurisé de l'expérience.

Internationalisation

Le site sera traduit en **anglais**, **allemand**, **espagnol**, **chinois**, **russe**, **japonais**, **hindi**, avec une priorisation sur les **régions pratiquant fortement les activités outdoor** (Allemagne, Canada, Suisse, USA, Japon...). L'adaptation culturelle des textes et interfaces sera soignée.

Développement de partenariats

Certaines de ces fonctionnalités pourront être développées **en collaboration** avec des acteurs tiers pour accélérer l'évolution de la plateforme tel que la conception et vente du boîtier autonome / tracker.

Monétisation

La croissance de la communauté impliquera des infrastructures plus coûteuses, rendant nécessaire la monétisation de certaines fonctionnalités : limitation du nombre de photos par aventure, accès avancé à l'application mobile via abonnement ou achat unique, commissions sur la marketplace, publicités discrètes, etc. L'objectif n'est pas de générer des millions, mais d'assurer un modèle économique viable qui soutienne une croissance saine, garantissant ainsi l'avenir de la plateforme et ses ambitions : sauver des vies.

2.4.3. Bilan global

Ce projet a été bien plus qu'un simple exercice de fin de formation : il a été le prolongement naturel des compétences acquises ces derniers mois. Grâce à la formation, j'ai pu acquérir des bases solides en **HTML**, **SCSS**, **JavaScript**, **PHP**, **Symfony**, **Twig** et **MySQL**, autant d'outils fondamentaux qui m'ont permis de me lancer sereinement.

Pathfinding est né d'un brainstorming en début de formation, d'une expérience personnelle marquante vécue seul dans les montagnes d'Hokkaido, au Japon, et d'une envie profonde d'entreprendre. En me fixant mes propres objectifs, j'ai été amené à dépasser le cadre du programme. J'ai appris à **intégrer de l'AJAX**, à configurer mon environnement de travail (**services.yaml**, **security.yaml**, **fichiers de routing personnalisés**), à structurer proprement mon code selon une architecture **SCSS 6:1** qui deviendra 7:1, et à exploiter des fonctionnalités avancées comme les **mixins paramétrables**, les **boucles SCSS**, ou encore les **animations CSS**. Le projet m'a poussé à apprendre par nécessité, à expérimenter, à chercher des solutions concrètes à des problèmes réels.

J'ai aussi compris une chose essentielle : **on ne code pas pour coder**, on développe pour répondre à un besoin. Ce besoin peut être personnel, collectif, ou commercial, mais dans tous les cas, il doit être clair, légitime et éprouvé. Si l'objectif est de créer un produit viable, il faut adopter une logique de **pragmatisme** : architecture propre, modularité, documentation, évolutivité, en vue d'une **collaboration future en équipe**.

Un enseignement en particulier m'a marqué : celui de **David ROBERT**, qui a insisté sur l'importance de **ne pas chercher la perfection dès le départ**, mais de **prototyper, tester progressivement, soumettre à un public**, ne serait-ce que pour **vérifier que le besoin existe réellement**. Cette vision m'a permis de sortir d'une logique figée pour avancer vers un développement itératif, vivant.

Pathfinding n'est pas un projet figé. C'est une base solide sur laquelle je vais continuer à bâtir. **Et si la formation m'a permis de monter dans le train avec des bagages bien remplis, ce projet en est aujourd'hui la locomotive.**

L'aventure ne fait que commencer !

INDEX DES FIGURES

Figure 1 : Tableau Kanban réalisé avec l'outil en ligne Trello	6
Figure 2 : Exemple de checklist réalisé avec Trello sur une tâche spécifique.....	7
Figure 3 : Diagramme de Gantt réalisé avec le power-up de Trello pour Pathfinding	7
Figure 4 : Arborescence du site web Pathfinding au stade prototype (projet)	8
Figure 5 : Synthèse de l'architecture technique du projet Pathfinding.....	9
Figure 6 : Moodboard de Pathfinding (version A4 en annexe)	11
Figure 7 : Logo de Pathfinding – version verte	12
Figure 8 : Palette de couleurs du projet réalisé avec Coolors	12
Figure 9 : Typographie simple et lisible.....	13
Figure 10 : Zoning de la page d'accueil version ordinateur/bureau	13
Figure 11 : Zoning de la page d'accueil version ordinateur/bureau	14
Figure 12 : Maquette de la page Aventure en version Web et Web Mobile (Responsive)	15
Figure 13 : Schéma illustrant le fonctionnement du modèle MVC pour afficher la page login	16
Figure 14 : Cycle requête-réponse entre client et serveur	17
Figure 15 : Test fonctionnel pour vérifier l'accessibilité des routes à un Visiteur	18
Figure 16 : Modèle Conceptuel de Données de Pathfinding	20
Figure 17 : Modèle Logique de Données de Pathfinding.....	21
Figure 18 : Exemples d'Asserts sur l'Entity User de Pathfinding	22
Figure 19 : Entités créées pour le projet Pathfinding – 08/06/2025	23
Figure 20 : Exemple de route pour afficher la liste des contacts de confiance.....	24
Figure 21 : Exemple de route pour partager une aventure à un visiteur.....	25
Figure 22 : Extrait du composant métier ContactListManager	26
Figure 23 : Extrait de logique métier déportée dans l'entité SafetyContact.....	27
Figure 24 : Configuration security.yaml et redirection personnalisée après connexion.....	28
Figure 25 : Extrait d'un contrôleur d'inscription avec hachage du mot de passe.....	29
Figure 26 : Redirection personnalisée après authentification selon le rôle de l'user.....	29
Figure 27 : Configuration de services.yaml.....	30
Figure 28 : Fichier base.html.twig de User – projet Pathfinding - 09/06/2025	31
Figure 29 : Arborescence des templates Twig	32
Figure 30 : Extrait du template adventure.html.twig de User – Contrôles d'attributs	32
Figure 31 : Exemple d'itération sur une liste d'objets dans la vue Adventure-manager	33
Figure 32 : Exemple de retour d'erreurs via Javascript sur la création de contact	34
Figure 33 : Changement dynamique de l'affiche des membres d'une liste en AJAX	35
Figure 34 : Recherche dynamique par nom dans une liste de contacts	35
Figure 35 : Script de la barre de recherche de SafetyList dans la vue Contact-Manager	36
Figure 36 : Arborescence et architecture 6 : 1 du SCSS dans Pathfinding	37
Figure 37 : Emploi de @use et @forward pour importer les styles dans main.scss	38

Figure 38 : Exemple d'imbrication avec balise, classe, mixin, variables et & :hover	39
Figure 39 : Exemple d'utilisation de positionnement, variables et imbrication.....	39
Figure 40 : Exemple d'animation employant Javascript et SCSS.....	40
Figure 41 : Exemple de classe type	40
Figure 42 : Exemple d'emploi de @extend dans une classe	40
Figure 42 : Exemple d'emploi de clamp(), rem et vw dans _typography.scss.....	41
Figure 44 : Exemple d'emploi de @media, calc() et \$breakpoint-sm dans le footer	41
Figure 45 : Exemple de design responsive mobile/tablette/ordinateur avec le footer	42
Figure 45 : Création de classes utilitaires (1 seul comportement) par boucles	42
Figure 45 : Exemple d'application de classes utilitaires dans profile.html.twig.....	43
Figure 48 : Utilisation d'icon font-awesome pour sélectionner le type d'aventure	43
Figure 49 : Utilisation de Leaflet pour afficher l'itinéraire suivi lors d'une aventure	44
Figure 50 : Utilisation de SweetAlert2 pour les alertes et pop-up – exemple du remove	44
Figure 51 : Utilisation de data-attribute pour les données des documents d'aventure	46
Figure 52 : Utilisation de set pour construire l'url de la photo de profile de l'user	47
Figure 53 : Résultat visuel obtenu avec l'utilisation de perspective sur le carrousel	47

INDEX DES TABLEAUX

Tableau 1 : Fonctionnalités principales Back-office accessible par rôle administratif	4
Tableau 2 : Fonctionnalités principales Front-office accessibles par rôle	5

RESSOURCES UTILISEES

Ce projet Pathfinding a été réalisé en s'appuyant sur de nombreuses ressources, documentations officielles, tutoriels, articles et librairies. Voici les principales sources qui ont guidé le développement, l'architecture, la configuration, ainsi que la compréhension des concepts techniques.

1. Supports de formation

- **LaPiscine - Formation Développement Web et Web Mobile**
Supports officiels, exercices, cours vidéos et ressources pédagogiques.

2. Documentation des technologies et langages

- **Symfony** : <https://symfony.com/doc/current/index.html>
- **Twig** (moteur de templates Symfony) : <https://twig.symfony.com/doc/3.x/>
- **CSS** : <https://www.w3schools.com/CSS/> et <https://developer.mozilla.org/fr/docs/Web/CSS>
- **SCSS / Sass** : <https://sass-lang.com/documentation>
- **HTML5** : <https://developer.mozilla.org/fr/docs/Web/HTML>
- **JavaScript (AJAX & JSON)** :
<https://developer.mozilla.org/fr/docs/Web/Guide/AJAX>
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/JSON
- **MySQL - types de données et DateTimeImmutable en PHP** :
<https://dev.mysql.com/doc/refman/8.0/en/data-types.html>
<https://www.php.net/manual/fr/class.datetimeimmutable.php>

3. Architecture et bonnes pratiques

- **Architecture SCSS 7:1** : <https://sass-guidelin.es/#the-7-1-pattern>
- **Utilisation de % / @extend en SCSS** :
<https://sass-lang.com/documentation/style-rules/extend>
- **LoggerInterface et gestion des logs dans Symfony** :
<https://symfony.com/doc/current/logging.html>
- **Tests unitaires et fonctionnels – définition et bonnes pratiques** :
<https://symfony.com/doc/current/testing.html>
<https://www.guru99.com/unit-testing-guide.html>
- **Guide pour écrire des prompts efficaces pour ChatGPT** :
<https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-openai-api>
- **Définition AJAX et JSON** :
<https://developer.mozilla.org/fr/docs/Web/Guide/AJAX>
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/JSON

4. Front-end, UI/UX et librairies

- **SweetAlert2 (fenêtres modales et alertes personnalisées)** : <https://sweetalert2.github.io/>
- **FontAwesome (icônes vectorielles)** : <https://fontawesome.com/>
- **Leaflet.js (cartographie interactive)** : <https://leafletjs.com/>
- **Carrousels JavaScript (tutoriels)** :
https://www.w3schools.com/howto/howto_js_slideshow.asp

5. Concepts et fonctionnalités Symfony avancées

- **Voter pour la gestion des droits** : <https://symfony.com/doc/current/security/voters.html>
- **Custom Authentication** :
https://symfony.com/doc/current/security/custom_authenticator.html
- **Enum en PHP et Symfony** : <https://www.php.net/manual/fr/language.enumerations.php>
- **DataFixtures** : <https://symfony.com/bundles/DoctrineFixturesBundle/current/index.html>
- **Services Symfony** : https://symfony.com/doc/current/service_container.html

6. Sécurité web

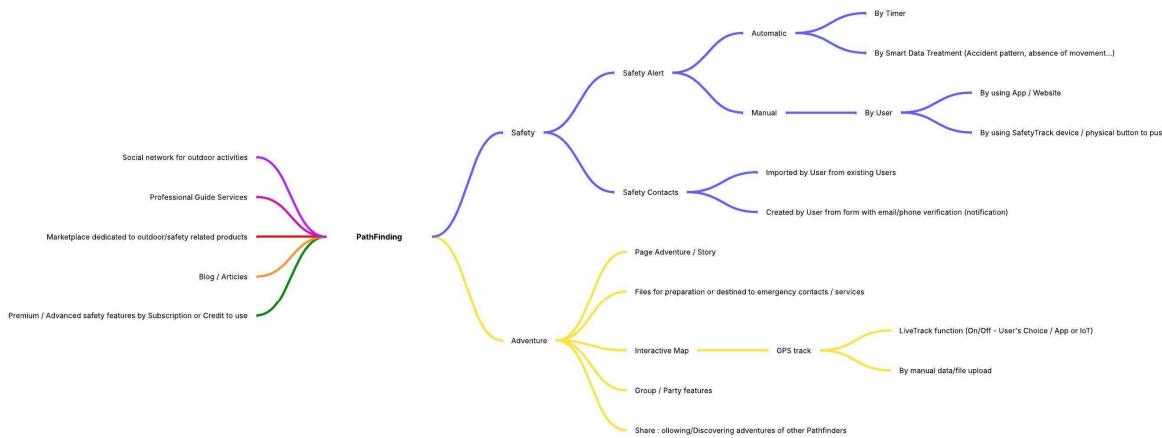
- **CSRF (Cross-Site Request Forgery)** : <https://symfony.com/doc/current/security/csrf.html>

7. Configuration et outils de développement

- **Configuration VSCode (settings.json) pour liveSassCompile & Twig** :
<https://marketplace.visualstudio.com/items?itemName=ritwickdey.live-sass>

ANNEXES

Annexe 1 : Mind-mapping



Cette mind map, réalisée en avril 2025, synthétise la structure fonctionnelle principale du projet Pathfinding :

- **Safety (Sécurité)** : Comprend la gestion des alertes de sécurité (*Safety Alert*) avec deux modes : automatique (via minuterie ou analyse intelligente de données) et manuelle (par l'utilisateur via l'application ou un dispositif physique). Elle intègre également la gestion des contacts de sécurité, pouvant être importés depuis d'autres utilisateurs ou créés via un formulaire avec vérification.
- **Adventure (Aventure)** : Regroupe la gestion des pages d'aventure/histoire, les fichiers liés à la préparation ou destinés aux contacts/services d'urgence, ainsi que les cartes interactives affichant les traces GPS, soit en LiveTrack, soit via un upload manuel. Ce module inclut aussi des fonctions sociales, comme la gestion de groupes ou la découverte/partage des aventures d'autres utilisateurs.
- **Autres composantes**
 - Réseau social dédié aux activités outdoor
 - Services de guides professionnels
 - Marketplace spécialisée dans les produits liés à la sécurité et à l'aventure
 - Blog et articles d'information
 - Fonctionnalités avancées premium accessibles par abonnement ou crédits

Annexe 2 : Analyse concurrentielle

Concurrent n°1 : LINKX

<https://www.lynkx.eu/>

Forces	Faiblesses
<ul style="list-style-type: none"> • Balise de communication GPS • Carte interactive • Intégration à la communauté Bivouak.net • App BIVOUAK (préparation aventure) • Abordable, pas d'abonnement, paiement à l'usage • Détection d'accident • Partenariat locaux et fonds pour investir/développer (FrenchTech, capital social > 165 000 euros, + de 5 salariés sur le développement) • Existe depuis 2021 (4 ans d'ancienneté) 	<ul style="list-style-type: none"> • Pas de communication satellite : GSM / LoRa / IoT - dépendant de la présence d'autres balises en secteur isolé • Peu connue du grand public malgré articles dans les journaux • App téléchargé par environ 1000 utilisateurs, nombre encore faible pour un système d'entraide international • Communauté locale : Alpes Française • Pas de portée internationale pour le moment

Concurrent n°2 : AllTrails

<https://www.alltrails.com/>

Forces	Faiblesses
<ul style="list-style-type: none"> • Grande base de données mondiale d'itinéraires • UX soignée • App traduire en de nombreuses langues • Fort SEO / référencement • Adoption massive et internationale • Abordable (de gratuit à 24 euros / an) 	<ul style="list-style-type: none"> • Pas de système de sécurité (alerte, contacts, GPS) • Faible dimension "suivi d'aventure" / communauté • Nombreux concurrents locaux gratuits en matière de proposition d'itinéraires • Pas de prévention

Concurrent n°3 : Garmin

<https://www.garmin.com/fr-FR/c/outdoor-recreation/satellite-communicators/>

Forces	Faiblesses
<ul style="list-style-type: none"> • Leader technologique en hardware • Technologie satellite fiable • Couverture mondiale (réseau Iridium) • Fonction SOS active 24h/24, 7j/7, très bonne autonomie, intégration avec les produits Garmin (montres, cartes, ...) 	<ul style="list-style-type: none"> • UX technique et inadapté au grand public • Coûteux (achat + abonnement) • Pas de plateforme communautaire liée au produit • Interface orientée pro/expédition

Annexe 3 : Analyse SWOT

L'analyse SWOT ci-après met en lumière les forces, faiblesses, opportunités et menaces de Pathfinding et a permis d'orienter sa stratégie de développement.

Forces	Faiblesses
<ul style="list-style-type: none"> Concept hybride unique : plateforme sociale, préparation d'aventure et sécurité intégrée Fonction Safety Alert intelligente, gratuite en version simple, suivi GPS manuel ou automatique via appli Interface moderne pensée pour le grand public comme pour les professionnels Création d'aventures collaboratives avec gestion des rôles et des partages Écosystème extensible : blog, boutique, gamification, guides 	<ul style="list-style-type: none"> MVP non public (Minimal Viable Product) Pas encore de balise / boîtier propre pour la communication d'urgence Application mobile à développer pour le LiveTrack (iOS / Android) Nécessite un nombre d'utilisateurs critique pour déclencher la dynamique communautaire Pas de marketing structuré à ce stade Déploiement, sécurité, RGPD, coûts serveurs à anticiper Ressources financières et humaines limitées face aux leaders du marché
Opportunités	Risques
<ul style="list-style-type: none"> Combler un vide de marché entre les apps sociales et les outils de sécurité professionnels Partenariats avec des fabricants (LoRa, GSM, satellite) pour intégrer des balises tierces ou développer. Audience large : randonneurs, campeurs, motards, grimpeurs, chasseurs, alpinistes, spéléologues, ... toutes activités outdoor en autonomie. Système freemium, possibilité à des proches de vous offrir le service premium (parents, grand-parents, amis, ... qui s'inquiètent pour vous) Participer à la création d'un écosystème LoRa communautaire → plus il y a de balises, plus les chances d'être retrouvé sont grandes Capter des influenceurs aventure/outdoor : récit d'aventure en homepage, partenariat média, ambassadeurs, etc. Lancer un podcast / chaîne YouTube pour parler sécurité, aventure et faire rayonner le projet 	<ul style="list-style-type: none"> Arrivée d'un acteur majeur avec les mêmes idées et des moyens considérables Dépendance temporaire au réseau GSM, app ou balises tiers en attendant le produit propre à Pathfinding Réglementations variables (RGPD, partage de position, radiofréquences...) selon les pays Les coûts d'infrastructure croîtront avec la communauté (serveurs, maintenance, SLA), à anticiper. Les utilisateurs avancés privilégient les solutions comme Garmin InReach pour la sécurité ou leurs réseaux sociaux (Instagram, YouTube...) pour le partage.