

O'Galaxy

Site fictif de réservation de voyages spatiaux

Projet réalisé dans le cadre de la présentation au
Titre Professionnel Développeur Web et Web Mobile

présenté par

Pierre MORIN
O'clock - Promotion Jelly



Introduction	4
Liste des compétences couvertes par le projet	6
1. Activité type 1 : Développer la partie front-end d'une application web / web mobile sécurisée	6
A. Maquetter des interfaces utilisateur web/web mobile (CP2)	6
B. Réaliser des interfaces utilisateur statiques web/web mobile (CP3)	6
C. Développer la partie dynamique des interfaces utilisateur web/web mobile (CP4)	6
2. Activité type 2 : Développer la partie back-end d'une application web / web mobile sécurisée	7
A. Mettre en place une base de données relationnelle (CP5)	7
B. Développer des composants d'accès aux données SQL et NoSQL (CP6)	7
C. Développer des composants métier côté serveur (CP7)	7
Contexte & résumé du projet	8
1. Présentation de l'entreprise et du service	8
2. Cahier des charges	9
A. Objectifs du projet	9
B. Cibles du projet	9
C. Les User Stories	9
D. Routes	10
E. Arborescence du site	10
F. Wireframes	11
G. Charte graphique et logo	12
3. Contraintes du projet et livrables attendus	13
A. Le MVP (Minimum Viable Product)	13
Évolutions potentielles du projet O'Galaxy	14
B. Les délais	14
4. L'environnement humain et technique, objectifs de qualité	15
A. Présentation de l'équipe	15
B. Méthodologie de travail	17
C. Description des Sprints	17
D. Outils utilisés	18
E. Choix des technologies	18
Réalisations personnelles côté front-end	19
1. Présentation de maquettes de l'application, adaptation web et web mobile	19
Wireframes	19
2. Schéma de l'enchaînement des maquettes de l'application	21
3. Captures d'écran d'interfaces utilisateur, adaptation web et web mobile	22
Réalisation de la page d'accueil du site O'Galaxy	22
4. Extraits de code d'interfaces utilisateur statiques	23
5. Extraits de code de la partie dynamique d'interfaces utilisateur	25
Réalisations personnelles côté back-end	28
1. Présentation de la base de données	28
A. Le MCD, Modèle Conceptuel de Données	28
B. Le MLD, Modèle Logique de Données	30

C. Le MPD, Modèle Physique de Données	30
D. Dictionnaire de données	31
E. Script de création de la base de données	32
2. Extraits de code de composants métier	32
A. Installation du security-bundle et du maker-bundle	33
B. Configuration du système de sécurité	33
C. Création de l'entité User	35
D. Mise à jour de la base de données	37
E. Création du formulaire d'authentification	37
F. Affichage du formulaire de connexion sur le site O'Galaxy	39
G. Création du formulaire d'enregistrement	40
3. Extraits de code de composants d'accès aux données	41
A. Création de la route dans le UserController	41
B. Crédit d'un template Twig pour afficher la vue	43
Présentation des éléments de sécurité	44
1. La sécurité, composante essentielle pour tout site web	44
2. Avec Symfony, des outils robustes pour la sécurité	44
A. Authentification et autorisations	44
B. Protection contre les attaques CSRF, les attaques par injection SQL et les attaques XSS	45
C. Hasher les mots de passe	46
D. Validation des données	46
Jeu d'essai	47
Veille sectorielle et veille sécurité	51
1. Veille technologique et sectorielle	51
2. Veille sécurité	51
Conclusion	53
Annexes	55
Annexe n°1 : Tableau des user stories	55
Annexe n°2 : Tableau des routes	58
Annexe n°3 : Dictionnaire de données	60
Annexe n°4 : Script de création de la base de données	63
Annexe n°5 : Templates list.html.twig du dossier templates/back/user	65

Introduction

Au lycée, un très bon ami à moi passait une partie de son temps libre à créer les niveaux de [son propre mod d'Half-Life](#) et il avait créé un site internet assez simple en HTML/CSS pour héberger ses projets, une sorte de portfolio pour montrer ses compétences en level design. Je me rappelle lui avoir posé énormément de questions sur son site et il m'avait conseillé quelques vidéos YouTube et le site Codecademy.com pour débuter en HTML/CSS. J'ai passé quelques heures à regarder les vidéos qu'il m'avait envoyées et à faire les exercices de ce site. Il y avait là quelque chose qui m'intriguait, le côté "logique" du code me parlait beaucoup, mais je n'imaginais pas qu'on puisse en faire son métier.

J'ai donc continué mes études : un Bac Scientifique, une rapide réorientation vers des études d'Histoire après une première année d'étude en IUT Hygiène Sécurité Environnement, une Licence d'Histoire, deux années à tenter de passer les concours d'entrée dans les écoles de journalisme, puis enfin un Master HRII (Histoire, Relations Internationales et Interculturelles). Très vite après la validation de mon mémoire, j'ai cherché un poste dans les métiers de la rédaction web et j'ai été recruté dans une petite agence à Rennes comme rédacteur web / community manager. J'ai pris goût à la gestion des réseaux sociaux et de 2016 à 2023, j'ai travaillé successivement dans trois agences de communication comme Community Manager, puis Social Media Manager et enfin Responsable du Pôle Social Media dans la dernière agence où j'ai travaillé.

Pendant toutes ces années, le développement web me faisait de l'œil et il m'arrivait régulièrement de refaire un petit tour sur mon compte Codecademy.com. Mais difficile de se lancer dans l'inconnu quand on a un métier stable et un enfant à charge. L'année dernière, ma compagne a eu une opportunité professionnelle à Bourges et comme mon entreprise ne souhaitait pas que je poursuive avec eux en télétravail, j'ai sauté sur l'occasion pour faire de cette idée qui trottait dans ma tête une réalité. Après une démission pour suivre de conjoint et une réunion avec ma conseillère France Travail où j'ai pu expliquer mon projet de façon claire et précise, j'ai obtenu un financement de France Travail pour effectuer une formation de six mois chez O'clock afin d'apprendre le métier de développeur web et web mobile PHP en vue de passer le Titre Professionnel DWWM.

Après avoir suivi les six mois de formation et la spécialisation sur le framework Symfony, l'école nous propose d'effectuer un projet de groupe en condition réelle : l'Apothéose. L'Apothéose, c'est une période d'un mois durant laquelle nous devons réaliser un projet de A à Z avec d'autres apprenants de notre promotion. C'est donc dans ce cadre que j'ai réalisé le projet O'Galaxy avec Guillaume Ribas, Jessica Troilo, Agnès Julien et Elodie Gauthier.

Ce projet nous a notamment permis d'acquérir les bonnes pratiques de travaux en groupe et d'approfondir nos connaissances du framework Symfony. Et plus personnellement, ce projet de groupe a confirmé plusieurs choses : j'aime travailler en équipe et je ne regrette pas d'avoir franchi le pas vers le monde du développement web !

Liste des compétences couvertes par le projet

1. Activité type 1 : Développer la partie front-end d'une application web / web mobile sécurisée

A. Maquetter des interfaces utilisateur web / web mobile (CP2)

Quand on se lance dans un projet de création de site web / web mobile, la première étape ne consiste pas à ouvrir son éditeur de code mais à travailler sur les documents de conception du projet.

Dans le cadre de ce travail préliminaire, nous avons tout d'abord défini les **user stories** de notre projet, c'est-à-dire définir clairement par écrit les actions que pourront faire les utilisateurs et administrateurs de notre site.

Les **user stories** prennent souvent la forme suivante : “**En tant qu'**[insérer rôle], **je peux** [insérer action], **afin de** [insérer finalité de l'action].”

Nous avons ensuite procédé à la création des **wireframes** de notre projet pour les formats mobiles et desktop. Les **wireframes** sont des schémas simplifiés qui servent à planifier la structure et la disposition des éléments d'une interface utilisateur d'un site web ou d'une application. Ces schémas se concentrent uniquement sur le contenu, les fonctionnalités et l'agencement et ne se préoccupent pas de l'aspect esthétique et visuel.

Enfin, nous avons mis en place une charte graphique comprenant une palette de couleur, un logo et une police pour assurer une cohérence visuelle à notre projet.

B. Réaliser des interfaces utilisateur statiques web / web mobile (CP3)

Pour notre projet, nous avons imaginé des interfaces utilisateur simples et intuitives. De plus, à une époque où la navigation sur smartphone est majoritaire, nous avons conçu un site intuitif et fonctionnel que ce soit sur smartphone ou sur desktop.

C. Développer la partie dynamique des interfaces utilisateur web / web mobile (CP4)

Après avoir défini nos **user stories** lors de la phase de conception du projet, nous nous sommes vite rendus compte que nous aurions besoin de dynamiser certains

éléments de notre front-end, notamment pour l'affichage de certaines fenêtres modales ou pour dynamiser la FAQ de notre site. Pour cela, nous avons utilisé du JavaScript Vanilla mais également le framework front-end Bootstrap qui comprend une collection de composants CSS & JavaScript.

2. Activité type 2 : Développer la partie back-end d'une application web / web mobile sécurisée

A. Mettre en place une base de données relationnelle (CP5)

Quand on réfléchit à la conception d'un site web ou d'une application, nous avons vu qu'il est important d'établir des documents de conception pour avoir une idée précise de ce qu'il va falloir ensuite coder. Pour les données, c'est sensiblement la même chose, il est très important d'établir en amont la façon dont nous allons les structurer.

Nous avons donc commencé par dresser la liste des données que nous voulions intégrer à notre site web O'Galaxy. Une fois la liste établie, nous avons produit un **Modèle Conceptuel de Données** (MCD), un **Modèle Logique de Données** (MLD), un **Modèle Physique de Données** (MPD) et un **dictionnaire de données**. Pour gérer la base de données de notre site web, nous avons choisi d'utiliser le système de gestion de bases de données relationnelles open source **MariaDB**.

B. Développer des composants d'accès aux données SQL et NoSQL (CP6)

Comme nous utilisons le framework Symfony pour notre projet O'Galaxy, nous avons utilisé l'**ORM (Object Relational Mapping) Doctrine** pour développer nos composants d'accès aux données SQL. Doctrine est une bibliothèque PHP qui nous permet de gérer les interactions avec une base de données relationnelle en utilisant une approche orientée objet.

C. Développer des composants métier côté serveur (CP7)

Pour réaliser nos composants métier côté serveur, nous avons utilisé le framework Symfony qui permet d'encapsuler la logique métier avec, par exemple, les services, les contrôleurs, les entités Doctrine ou encore les Listeners et Subscribers d'Événements.

Contexte & résumé du projet

1. Présentation de l'entreprise et du service

Vous en rêviez ?! Nous allons le faire ! Suite aux découvertes technologiques de notre ingénieur en chef Melon Usk, le voyage interstellaire est enfin possible ! Souhaitant prendre les devants sur le marché du voyage intergalactique, nous avons développé le site de notre agence de voyage vers différentes planètes de l'univers de la Star Wars (rassurez-vous, pas de voyages prévus sur l'Étoile de la Mort).

Notre but principal est de proposer des expéditions vers les différentes planètes de l'univers Star Wars : envie d'un chaleureux trek sur Tatooine, d'une rafraîchissante promenade sur Hoth ou tout simplement de vous perdre sur Coruscant, c'est sur O'Galaxy qu'il faudra réserver votre voyage pour y aller.

O'Galaxy a pour but de démocratiser le voyage intergalactique jusqu'ici très peu, voire pas du tout, exploité par les agences spatiales. C'est un véritable rêve qui devient réalité pour les nombreux fans de Star Wars.

[À titre de rappel, ce projet est totalement fictif !]

J'ai beaucoup aimé l'idée de ce projet proposé par Guillaume Ribas lors des dernières semaines précédant le début de l'Apothéose. Comme l'univers geek me parlait totalement, j'ai décidé de mettre ce projet en n°1 sur la liste de vœux de projets pour l'Apothéose. De plus, sur ce type de projets, il y a une part de créativité supplémentaire puisque notre limite est notre imagination. Je suis donc très heureux d'avoir participé à la conception du site O'Galaxy avec mes collègues apprenants.



2. Cahier des charges

A. Objectifs du projet

Après la phase de réflexion, et juste avant de commencer à rédiger les documents de conception, voici les objectifs que nous avions définis pour notre projet.

L'objectif du projet O'Galaxy est de créer un site de réservation de voyage fonctionnel pour que les utilisateurs puissent :

- consulter une liste d'expéditions disponibles,
- consulter le détail de ces expéditions,
- se connecter pour effectuer une réservation

...et pour que les administrateurs puissent :

- se connecter à un back office,
- valider les réservations,
- gérer des expéditions,
- gérer les utilisateurs.

B. Cibles du projet

Même si les voyages intergalactiques sont amenés à gagner en popularité après les découvertes récentes de Melon Usk, tout le monde n'a pas les moyens de s'offrir un voyage au-delà des frontières de notre atmosphère. Nos cibles sont donc principalement des membres de familles fortunées et des célébrités.

C. Les User Stories

Une fois les bases du projet posées, nous avons rédigé les user stories du site O'Galaxy. Un **récit utilisateur**, ou **user story**, est une phrase simple qui permet de décrire une fonctionnalité que l'on veut créer et, cette **user story** est rédigée du point de vue de l'utilisateur final.

Par exemple : “*En tant que visiteur, je souhaite consulter la page “Expéditions” afin de connaître toutes les destinations où je peux me rendre.*”

En fin de projet, cela permet de s'assurer que toutes les fonctionnalités requises ont bien été développées.

Vous trouverez le tableau des user stories en Annexe n°1 de ce dossier de projet.

D. Routes

En amont de la phase de développement, nous avons également réfléchi à l'ensemble des routes dont nous avions besoin pour notre site.

Nous avons donc créé un tableau des routes pour notre front office et un tableau des routes pour notre back office. Vous trouverez ces deux tableaux en Annexe n°2 de ce dossier de projet.

Le fait de créer un tableau des routes en amont du développement permet, comme avec les user stories, d'avoir une structure claire et précise en tête. De plus, cela facilite aussi grandement l'arrivée d'éventuelles nouvelles personnes sur le projet.

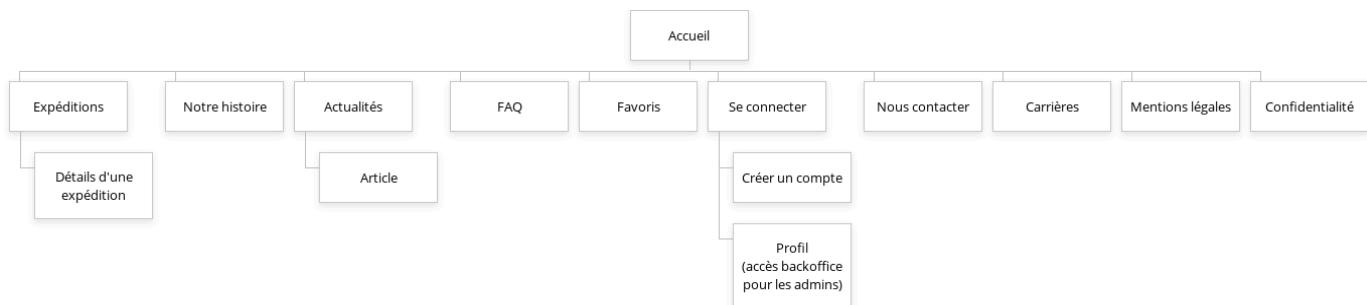
E. Arborescence du site

Une fois les user stories créées, nous avons produit l'arborescence du site O'Galaxy.

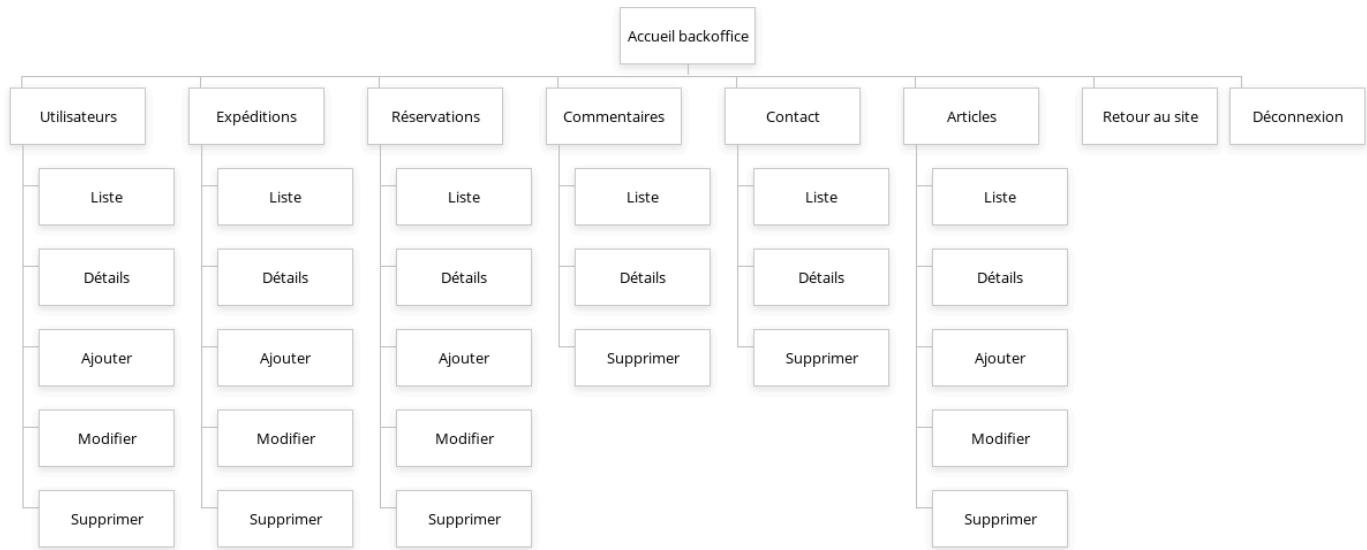
L'arborescence d'un site web est un schéma organisationnel qui décrit la manière dont les différentes pages du site sont organisées entre elles. Nous avons eu besoin, pour notre projet, de deux arborescences : une pour les visiteurs et les utilisateurs du site, mais également une pour les administrateurs du site qui peuvent se connecter à un back office qui permet de gérer le site.

Voici les deux arborescences de notre site :

- l'arborescence du site O'Galaxy :



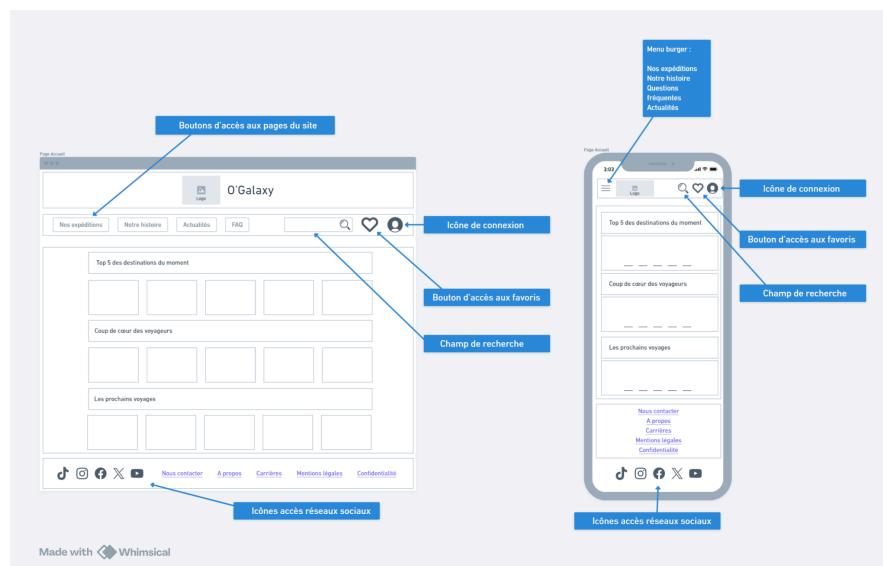
- l'arborescence du back office pour les administrateurs (l'accès se fait via le profil sur le site) :



F. Wireframes

Les user stories et l'arborescence sont définies, ce qui nous a permis de bien cerner les contours de notre projet, nous souhaitons donc maintenant passer à l'étape suivante et proposer un aperçu graphique du futur site grâce à une série de wireframes.

Les **wireframes**, à ne pas confondre avec les maquettes, sont des schémas utilisés lors de la conception d'une interface utilisateur pour définir les zones et les composants que celle-ci doit contenir. À contrario, la maquette est une version aboutie du site que le développeur front-end devra reproduire à l'identique.



Voici un exemple des wireframes desktop et mobile de la page d'accueil de notre projet. Ces wireframes ont été réalisés à l'aide de l'outil Whimsical.

Des exemples de wireframes que j'ai pu réaliser pendant le projet seront également visibles plus loin dans ce dossier dans la partie des réalisations personnelles côté front-end.

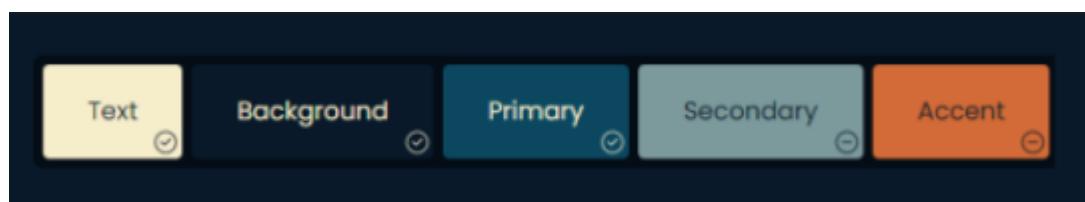
G. Charte graphique et logo

Une fois les wireframes produits, nous nous sommes penchés sur l'aspect graphique du futur site en créant un logo, une palette de couleur et une sélection de deux polices d'écriture.

Logo du site O'Galaxy :



Palette de couleurs pour le design du site :





Polices d'écriture sélectionnées :

- Aldrich - Bunny Font <https://fonts.bunny.net/family/aldrich>

Normal 400

O'Galaxy l'agence qui vous envoie dans l'espaaaaaaace !

- Bruno Ace SC <https://fonts.bunny.net/family/bruno-ace-sc>

Normal 400

O'GALAXY L'AGENCE QUI VOUS ENVOIE DANS L'ESPAAAAACE !

3. Contraintes du projet et livrables attendus

A. Le MVP (Minimum Viable Product)

L'étape suivante de notre projet a été de définir le **MVP** : *Minimum Viable Product*, ou **Produit Minimum Viable** dans la langue de Molière.

En développement web, le MVP est une version simplifiée d'un produit incluant uniquement les fonctionnalités essentielles pour satisfaire les premiers utilisateurs. L'objectif d'un MVP est de pouvoir valider rapidement une idée de produit en utilisant un minimum d'effort et de ressources.

Dans le cadre de notre projet O'Galaxy, notre MVP s'est orienté autour de la réservation d'une expédition par un utilisateur. Cela nécessite donc la possibilité pour un visiteur de se créer un compte et de pouvoir s'authentifier de façon sécurisée, mais aussi la possibilité pour les administrateurs de gérer le site (pouvoir ajouter des expéditions, valider des réservations, gérer les utilisateurs...).

Voici la liste des fonctionnalités que nous avons défini pour notre MVP :

- Accéder à la page d'accueil du site
- Accéder à la page "Expéditions" et aux détails de chaque "Expédition"
- Pouvoir se créer un compte
- Pouvoir se connecter
- Avoir accès à sa page profil avec les expéditions (en attente, validées, annulées)
- Accéder au back office quand on est administrateur du site
- Avoir la possibilité de mettre des "Expéditions" en favoris
- Avoir accès aux différentes pages annexes du site (Notre histoire, FAQ, Carrières, Mentions légales, Confidentialité)
- Avoir la possibilité de consulter les actualités
- En back office, pouvoir gérer (voir, créer, modifier, supprimer) des "Expéditions"
- En back office, pouvoir gérer (voir, créer, modifier, supprimer) les réservations des utilisateurs
- En back office, pouvoir gérer (voir, créer, modifier, supprimer) les utilisateurs
- En back office, pouvoir gérer (créer, modifier, supprimer) les articles

B. Évolutions potentielles du projet O'Galaxy

Une fois le MVP défini, nous avons également pensé en amont aux éventuelles évolutions du projet.

- Accéder à un formulaire de contact pour échanger avec l'équipe du site
- Pouvoir noter une "Expédition" après un voyage
- Pouvoir laisser un commentaire sur une page "Expédition"
- La mise en place des activités et les lier aux différentes expéditions pour permettre un filtre
- Mettre des filtres de sélection sur le climat et les activités (désertique, humide, montagne...)
- Réserver une expédition en passant par un panier et une validation de l'achat (en remplacement du formulaire de contact pour la réservation)

C. Les délais

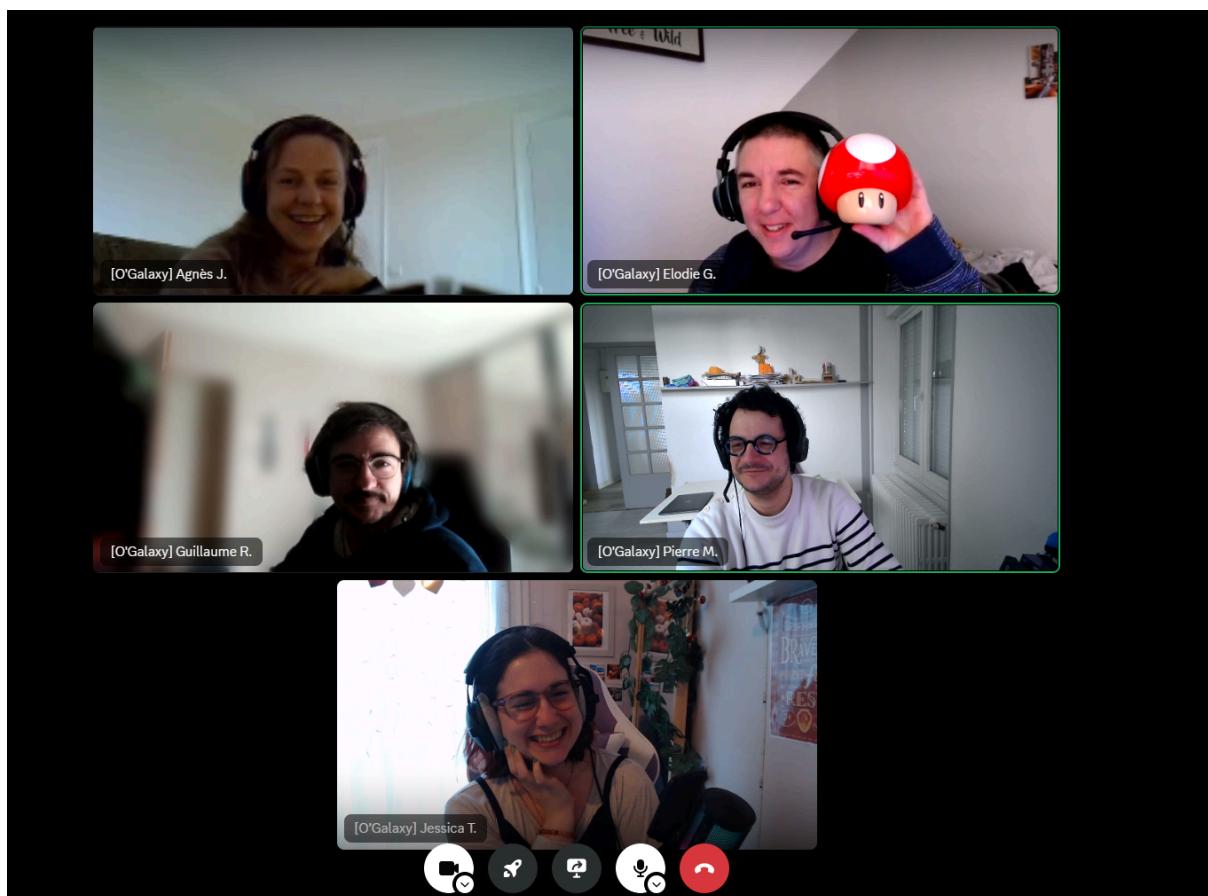
Pour effectuer notre projet, nous disposons de 21 jours répartis en 4 sprints de 5 à 6 jours. Durant chacun de ces sprints, nous avons eu un rendez-vous de suivi avec nos tuteurs O'clock ainsi qu'une rétrospective de sprint où nous devions présenter nos avancées aux tuteurs et à un autre groupe d'Apothéose.

4. L'environnement humain et technique, objectifs de qualité

A. Présentation de l'équipe

Pour réaliser ce projet, j'ai eu le plaisir de travailler avec Guillaume Ribas, Jessica Troilo, Agnès Julien et Elodie Gauthier. La cohésion de groupe s'est faite très rapidement, et nous n'avons recensé aucune tension durant les 21 jours de l'Apothéose. Nous avons vite mis en place des rituels de travail et des rendez-vous réguliers en équipe sur Discord pour discuter des avancées des uns et des autres.

La photo de groupe ci-dessous, que nous avons prise lors d'une réunion Discord, témoigne de cette bonne ambiance. Je les remercie vraiment car ces 21 jours de travail intense ont été très agréables avec eux !



Voici la liste des rôles que nous avons défini en début de projet pour la réalisation du projet O'Galaxy :

- **Product Owner** : Guillaume

Le Product Owner est celui qui définit et priorise les fonctionnalités du produit en développement pour en maximiser la valeur de ce dernier. Il sert d'intermédiaire entre la partie métier et la partie technique du projet.

- **Scrum Master** : Jessica & Pierre

Scrum Master est un animateur de Scrum, un framework Agile léger qui met l'accent sur des itérations limitées dans le temps qu'on appelle "sprints". Les Scrum Masters jouent également le rôle de coachs pour le reste de l'équipe de développement, ou de « leaders-serviteurs », selon la formulation du Guide Scrum.

- **Lead Dev Front** : Jessica, Guillaume & Pierre

Les Lead Developer Front sont les développeurs responsables de superviser l'équipe de développement, guidant les choix techniques et assurant la qualité du code sur la partie front-end de l'application.

- **Lead Dev Back** : Elodie & Agnès

Les Lead Developer Back sont les développeurs responsables de superviser l'équipe de développement, guidant les choix techniques et assurant la qualité du code sur la partie back-end de l'application.

- **Git Master** : Guillaume

Le Git Master désigne une personne experte en Git et qui va être responsable de la gestion des branches, des merges et workflows Git au sein d'une équipe de développement.

- **Référent(e)s technos** : en fonction des tâches

Les référent(e)s technos sont les personnes qui seront garantes de la bonne utilisation de telle ou telle technologie. Par exemple, j'ai pris le rôle de référent Bootstrap car j'ai eu l'occasion d'utiliser de nombreuses fois ce framework front-end lors de projets personnels.

B. Méthodologie de travail

Dans un projet de développement web, la méthodologie est très importante pour tenir les délais et livrer le produit attendu. Nous avons suivi les principes de la méthode Scrum qui permet une certaine souplesse en s'adaptant aux éventuels changements. La méthode Scrum repose sur des itérations courtes appelées “**sprints**” qui durent généralement entre 1 à 4 semaines).

Durant ces “sprints”, des tâches du **product backlog**, la liste des fonctionnalités à développer, sont priorisées par le **product owner** et le **scrum master** s'assure du bon déroulement du sprint.

Cette méthode est aussi très intéressante car elle favorise l'échange entre l'équipe de développement et les parties prenantes durant la période de développement du produit. Ces échanges prennent souvent la forme de réunions régulières :

- **Daily stand-ups** : Ce sont des réunions quotidiennes courtes (environ 10-15 minutes) où les membres de l'équipe partagent ce qu'ils ont fait la veille, ce qu'ils prévoient de faire aujourd'hui, et mentionnent les problèmes qu'ils ont pu rencontrer. Le but est de synchroniser les activités et de pouvoir identifier rapidement les problèmes.
- **Rétrospectives de sprints** : Ce sont des réunions tenues à la fin de chaque sprint, où l'équipe réfléchit sur ce qui a bien fonctionné, ce qui n'a pas fonctionné, et comment améliorer les processus pour les sprints suivants. L'objectif est d'améliorer continuellement la manière de travailler de l'équipe.
- **Revues de sprint** : Ce sont des réunions qui ont lieu à la fin de chaque sprint, où l'équipe présente les travaux complétés aux parties prenantes. Cette réunion est l'occasion de présenter les fonctionnalités développées, de recueillir des avis et de s'assurer que le projet est sur la bonne voie par rapport aux attentes des utilisateurs et des parties prenantes.

C. Description des Sprints

Il y a eu quatre sprints d'une durée de 5 à 6 jours durant notre projet :

- Sprint 0 : mise en place du projet, production des documents de conception
- Sprint 1 : beaucoup de HTML/CSS/Bootstrap côté front-end pour commencer + mise en place de la plupart des fonctionnalités listées dans le MVP
- Sprint 2 : mise en place de l'authentification, de la création de compte, de la fonctionnalité de recherche + harmonisation de l'aspect visuel du site côté front

- Sprint 3 : sprint dédié à la vérification du code, au test des fonctionnalités et à la gestion des questions de sécurité du site

D. Outils utilisés

Pendant la durée du projet O'Galaxy, nous avons utilisé de nombreux outils que nous pouvons mettre dans deux catégories :

- Les outils de code : la machine virtuelle Linux sur laquelle nous codons, l'éditeur de code Visual Studio Code (et ses extensions) avec lequel nous écrivons notre code, le terminal Linux.
- Les outils de gestion de projet & de création graphique : Trello pour l'organisation des sprints et la répartition des tâches, Slack et Discord pour les réunions et le pair-programming, GitHub pour la gestion du versionning de notre projet, Whimsical pour créer nos wireframes, drawDB pour créer le Modèle Physique de Données (MPD), Google Drive pour stocker nos documents de conception et nos images.

E. Choix des technologies

En ce qui concerne les technologies utilisées lors de notre projet :

- Pour la partie front-end nous avons utilisé le moteur de templates Twig intégré avec le framework Symfony. C'est dans ces templates que nous avons écrit notre HTML/CSS pour la partie graphique et visuelle de notre site. Nous avons également utilisé le framework CSS Bootstrap pour faciliter l'écriture de notre CSS grâce aux multiples composants CSS déjà existants. Côté JavaScript, nous avons utilisé JavaScript Vanilla, notamment pour la FAQ (les réponses apparaissent quand on clique sur la question), et certains composants Bootstrap, notamment pour les fenêtres modales.
- Pour la partie back-end de l'application, nous avons utilisé le framework PHP Symfony, et MariaDB comme SGBDR (Système de Gestion de Base de Données Relationnelle).

Réalisations personnelles côté front-end

Lors de ce projet, nous étions une équipe venant entièrement de la spécialisation Symfony, donc plutôt destinée à travailler sur la partie back-end du site. Cependant, nous avons tenu à toutes et tous travailler de manière égale sur le front-end et sur le back-end et ne pas choisir de diviser le groupe en deux avec un groupe ne travaillant que sur le front-end et un autre groupe ne travaillant que sur le back-end. Ainsi, tout le monde a pu appréhender les deux parties du site.

When backend Developer does Frontend



1. Présentation de maquettes de l'application, adaptation web et web mobile

Wireframes

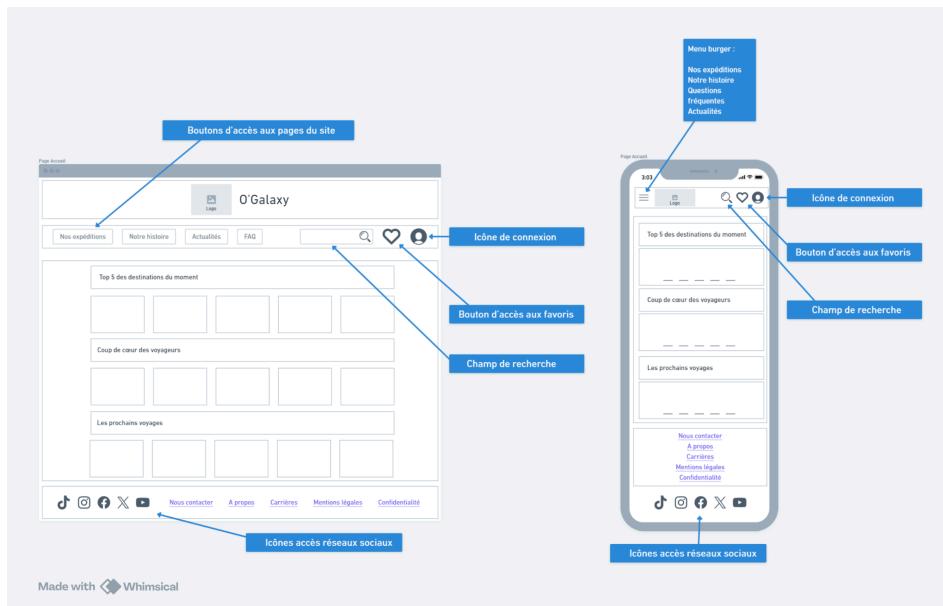
Une fois les user stories définies, nous souhaitons donc maintenant passer à l'étape suivante et proposer un aperçu graphique du futur site grâce à une série de **wireframes**.

Les **wireframes**, à ne pas confondre avec les maquettes, sont des schémas utilisés lors de la conception d'une interface utilisateur pour définir les zones et les composants que cette interface doit contenir. À contrario, la maquette est une version aboutie du site que le développeur front-end devra reproduire à l'identique.

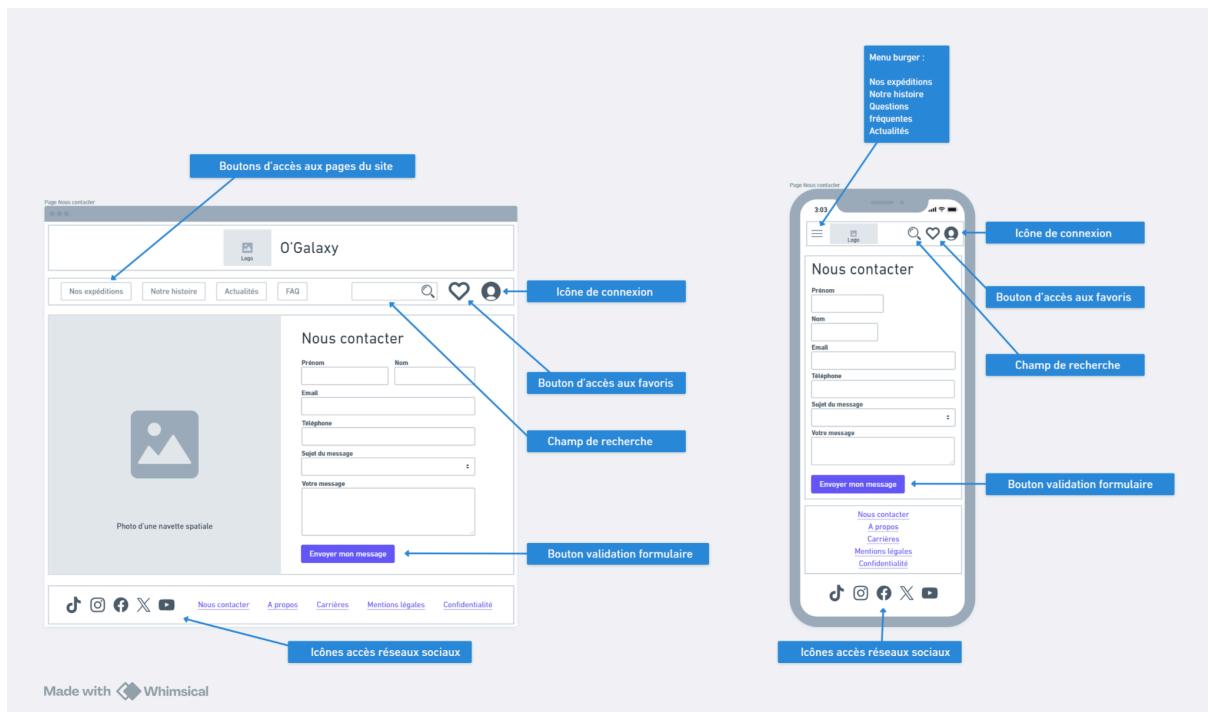
Nos wireframes ont été réalisés grâce à l'outil en ligne [Whimsical](#).

Personnellement, j'étais en charge des wireframes de la page d'"Accueil" et des wireframes de la page "Contact" :

Page “Accueil” - Version desktop et mobile



Page “Contact” - Version desktop et mobile



Sur les wireframes, nous utilisons des **annotations** (encarts bleus avec une flèche) qui peuvent être utiles pour expliquer de façon limpide l'événement lié à un élément.

Exemple : Que se passe-t-il si je clique sur le bouton en forme de cœur ?
L'annotation m'indique que cela m'emmènera sur la page des favoris.

De plus, le site étant créé selon une conception responsive, les wireframes nous permettent de montrer comment sont agencés les différents éléments sur chaque version.

2. Schéma de l'enchaînement des maquettes de l'application

Un schéma de l'enchaînement des maquettes d'un site ou d'une application, souvent appelé **Storyboard** ou **wireflow**, est une représentation visuelle des différentes interfaces utilisateur (UI) de l'application et de la manière dont elles sont interconnectées. Ce schéma aide à comprendre le flux de navigation et l'interaction entre les différentes pages ou écrans de l'application.

Ce schéma peut également être transposé à l'écrit, voici ce que cela pourrait donner pour quelques écrans dans le cas de notre site O'Galaxy :

Écran d'accueil

- **Bouton “Connexion / S'enregistrer”** → Mène à l'écran de connexion.
- **Boutons “Découvrir”** → Mènent sur la page des détails d'une Expédition.

Écran de connexion

- **Champ “Adresse mail”**
- **Champ “Mot de passe”**
- **Bouton “Connexion”** → Mène sur la page d'accueil du site si les informations entrées sont correctes.
- **Bouton “Créer un compte”** → Mène à l'écran de création d'un compte.

Écran de création d'un compte

- **Champ “Ton pseudo, tu inventeras...”**
- **Champ “Ton prénom, tu indiqueras...”**
- **Champ “Ton nom, tu dévoileras...”**
- **Champ “Ton email, tu renseigneras...”**
- **Champ “Ton mot de passe, tu créeras...”**
- **Case à cocher “Les conditions d'utilisation, tu accepteras.”**
- **Bouton “S'inscrire”** → Mène sur la page d'accueil du site si les informations entrées sont correctes.

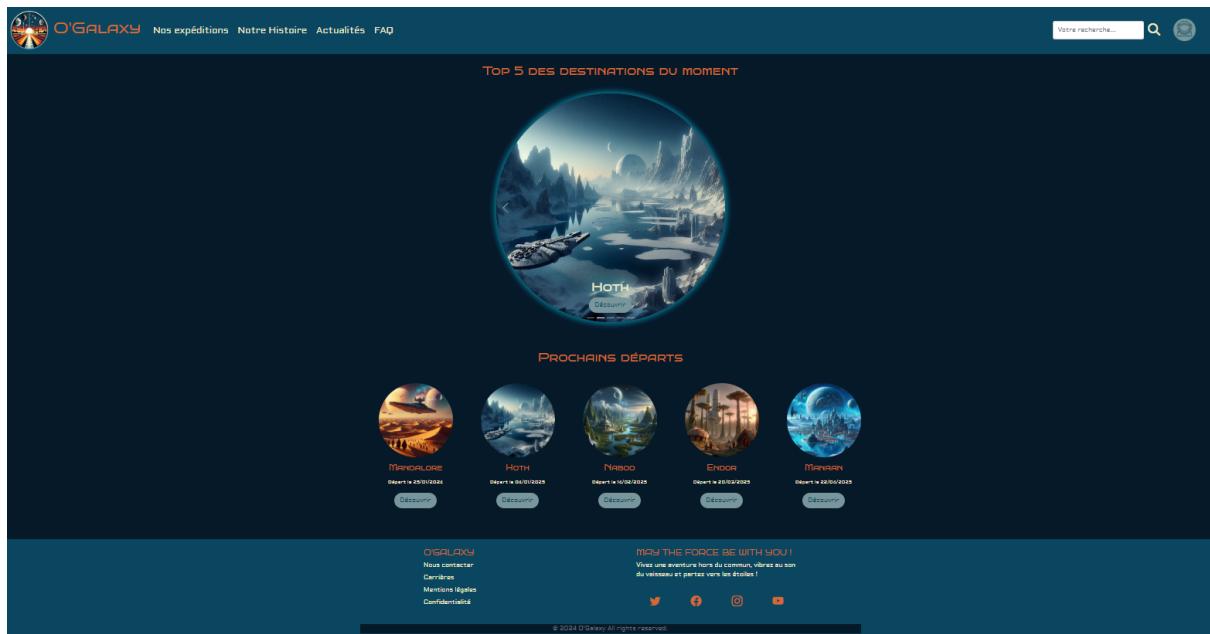
Écran “Nos expéditions”

- **Boutons “En savoir plus”** → Mènent sur la page des détails d'une Expédition.

3. Captures d'écran d'interfaces utilisateur, adaptation web et web mobile

Réalisation de la page d'accueil du site O'Galaxy

Rendu de la page d'accueil sur desktop



Rendu de la page d'accueil sur mobile



Sur les captures d'écran de la page d'accueil du site O'Galaxy, nous avons effectué quelques modifications par rapport aux wireframes que nous avions présentés précédemment. En effet, nous n'avions, à l'origine, pas prévu de mettre un carrousel sur la version desktop pour la section "Top 5 des destinations du moment".

4. Extraits de code d'interfaces utilisateur statiques

Concernant mes réalisations personnelles côté front-end, j'ai eu pour mission de réaliser le template de la page d'accueil du site O'Galaxy.

Pour illustrer cela, je vais me baser sur un exemple concret : la création de la section "Prochains départs" qui affichera dynamiquement la liste des 5 prochaines expéditions qui doivent avoir lieu.



```
<!-- Display the next departure - MOBILE VERSION -->
<section id="next-departure-version-mobile" class="d-block d-lg-none">
  <h2 class="container pt-4 home__bigtitle mt-4 text-center mb-5">
    Prochains départs
  </h2>
  <div id="carouselNextDeparture" class="container d-flex justify-content-center carousel carousel-dark slide">
    <div class="carousel-inner w-75">
      {% for currentHome in expeditionDeparture %}
        <div class="carousel-item {% if loop.first %}active{% endif %}" data-bs-interval="10000">
          
          <div class="carousel-caption">
            <h5 id="home__title">
              {{ currentHome.title }}
            </h5>
            <p class="home__text_dep">
              Départ le {{ currentHome.departure | date ('d/m/Y') }}
            </p>
            <a href="{{ path('app_front_expedition_show', {id:currentHome.id}) }}" class="btn list__btn" role="button"
              aria-label="Bouton pour afficher le slide choisi">
              Découvrir
            </a>
          </div>
        </div>
      {% endfor %}
      <button class="carousel-control-prev white__arrow" type="button" data-bs-target="#carouselNextDeparture"
        data-bs-slide="prev" aria-label="Bouton pour afficher sur le slide précédent">
        <span class="carousel-control-prev-icon" aria-hidden="true"></span>
        <span class="visually-hidden">
          Previous
        </span>
      </button>
      <button class="carousel-control-next white__arrow" type="button" data-bs-target="#carouselNextDeparture"
        data-bs-slide="next" aria-label="Bouton pour afficher sur le slide suivant">
        <span class="carousel-control-next-icon" aria-hidden="true"></span>
        <span class="visually-hidden">
          Next
        </span>
      </button>
    </div>
  </div>
</section>
```

Voici un extrait de code correspondant au carrousel "Prochains départs" sur la version mobile.

```

● ● ●

<!-- Display the next departure - DESKTOP VERSION -->
<section id="next-departure-version-desktop" class="d-none d-lg-block">
    <h2 class="container pt-4 home__bigtitle mt-4 text-center mb-5">
        Prochains départs
    </h2>
    <div
        class="container d-flex gap-4 mt-3">
        <!-- CARD -->
        {% for currentHome in expeditionDeparture %}
            <div id="home__next__border" class="card d-flex mb-3 ps-5" style="width: 30rem;">
                
                <div class="card-body d-flex align-items-center flex-column home__card__bg">
                    <h5 class="card-title home__title mb-3">
                        {{ currentHome.title }}
                    </h5>
                    <p class="card-text home__text__dep">
                        Départ le {{ currentHome.departure | date ('d/m/Y') }}
                    </p>
                    <a href="{{ path('app_front_expedition_show', {id:currentHome.id}) }}" class="btn list__btn" role="button"
                        aria-label="Bouton pour afficher l'expédition choisie">
                        Découvrir
                    </a>
                </div>
            </div>
        {% endfor %}
    </div>
</section>

```

Voici un extrait de code correspondant à la liste des “Prochains départs” sur la version desktop.

Comme nous l'avons expliqué précédemment, nous souhaitons que notre site soit responsive, c'est-à-dire que sa mise en page s'adapte au fait que l'on soit sur mobile ou sur desktop. Sur les wireframes, nous avons vu que la version desktop de cette section “Prochains départs” est une liste horizontale de cartes affichant les 5 prochaines expéditions.

Or, sur mobile, afficher une telle liste reviendrait à faire des cartes minuscules pour que celles-ci s'affichent sur la largeur de l'écran.

Nous avons donc opté pour la création de deux **<section>** qui s'afficheront l'une ou l'autre selon la taille de l'écran. C'est que qu'on voit très bien avec les classes des deux sections :

```

● ● ●

<!-- Display the next departure - MOBILE VERSION -->
<section id="next-departure-version-mobile" class="d-block d-lg-none">
    ...
</section>

<!-- Display the next departure - DESKTOP VERSION -->
<section id="next-departure-version-desktop" class="d-none d-lg-block">
    ...
</section>

```

Côté mobile, nous avons **class="d-block d-lg-none"** :

- **d-block** : C'est la classe utilitaire de Bootstrap qui définit la propriété **display** d'un élément à **block**.
- **d-lg-none** : C'est une classe utilitaire de Bootstrap qui va masquer l'élément sur les écrans d'une largeur égale ou supérieure au breakpoint **lg** de Bootstrap, ce qui correspond à 992 pixels.

Côté desktop, nous avons **class="d-none d-lg-block"** :

- **d-none** : C'est la classe utilitaire de Bootstrap qui définit la propriété **display** d'un élément à **none**. C'est-à-dire que l'élément va être masqué.
- **d-lg-block** : C'est une classe utilitaire de Bootstrap qui va afficher l'élément sur les écrans d'une largeur égale ou supérieure au breakpoint **lg** de Bootstrap, ce qui correspond à 992 pixels.

En ce qui concerne le reste des deux sections, nous avons utilisé des composants Bootstrap :

- Des cartes (cards dans la langue de Shakespeare) pour la version desktop.

Les cartes sont des contenus flexibles et extensibles de Bootstrap, elles contiennent du contenu et des informations : dans notre cas, une image, le nom de l'Expédition et un bouton qui mène vers la page de détail de l'Expédition.

- Un carrousel pour la version mobile du site.

Le carrousel est composé de plusieurs slides qui comportent chacune du contenu et des informations : dans notre cas, une image, le nom de l'Expédition et un bouton qui mène vers la page de détail de l'Expédition.

5. Extraits de code de la partie dynamique d'interfaces utilisateur

Pour illustrer la mise en place d'élément dynamique sur l'interface utilisateur, nous allons prendre l'exemple du carrousel créé pour afficher le "Top 5 des destinations du moment" qui est un exemple concret de dynamisme sur une interface utilisateur.

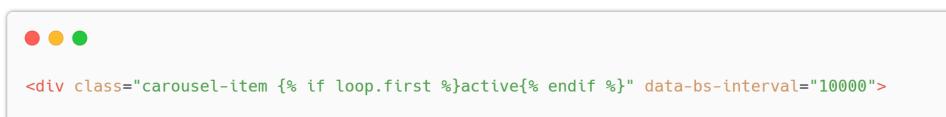
En effet, les **carrousels** ou **sliders** sont des éléments qui permettent de faire défiler des images, du texte, des cartes, etc... Ce sont donc des éléments dynamiques. Le carrousel utilisé pour notre projet provient de [la bibliothèque de composants de Bootstrap](#).



```
<!-- Display top 5 - MOBILE VERSION -->
<section id="destinations-of-the-moment" class="d-block d-lg-none">
    <h2 class="container pt-4 home__bigtitle mt-4 text-center mb-5">
        Top 5 des destinations du moment
    </h2>
    <div id="carouselExampleAutoplaying" class="container d-flex justify-content-center carousel carousel-dark slide">
        <div class="carousel-inner w-75">
            {% for currentHome in expeditionHome %}
                <div class="carousel-item {% if loop.first %}active{% endif %}" data-bs-interval="10000">
                    
                    <div class="carousel-caption">
                        <h5 id="home__title">
                            {{ currentHome.title }}
                        </h5>
                        <p class="home__text__dep">
                            Départ le {{ currentHome.departure | date ('d/m/Y') }}
                        </p>
                        <a href="{{ path('app_front_expedition_show', {id:currentHome.id}) }}" class="btn list__btn" role="button"
                            aria-label="Bouton pour afficher le slide choisi">
                            Découvrir
                        </a>
                    </div>
                </div>
            {% endfor %}
            <button class="carousel-control-prev white__arrow" type="button" data-bs-target="#carouselExampleAutoplaying"
                data-bs-slide="prev" aria-label="Bouton pour afficher sur le slide précédent">
                <span class="carousel-control-prev-icon" aria-hidden="true"></span>
                <span class="visually-hidden">
                    Previous
                </span>
            </button>
            <button class="carousel-control-next white__arrow" type="button" data-bs-target="#carouselExampleAutoplaying"
                data-bs-slide="next" aria-label="Bouton pour afficher sur le slide suivant">
                <span class="carousel-control-next-icon" aria-hidden="true"></span>
                <span class="visually-hidden">
                    Next
                </span>
            </button>
        </div>
    </div>
</section>
```

Voici un extrait de code correspondant au carrousel du “Top 5 des prochaines destinations” sur la version mobile.

Plusieurs éléments dans cet extrait de code nous montrent que nous sommes face à une interface utilisateur dynamique.



```
<div class="carousel-item {% if loop.first %}active{% endif %}" data-bs-interval="10000">
```

Tout d'abord, sur la <div> ci-dessus, on note la présence de l'attribut **data-bs-interval=10000** qui indique que le carrousel défile de manière automatique toutes les 10 secondes sans que l'utilisateur n'ait besoin d'effectuer une interaction. On note également la présence de la classe **active** affichée de manière conditionnelle avec la condition **{% if loop.first %} active {% endif %}**. Dans le composant carrousel de Bootstrap, une slide ne peut être visible par défaut que si elle possède la classe **active**. La condition fait appel à la variable **loop.first** du moteur de template Twig et renvoie **true** si l'itération actuelle est la première itération de la boucle.



Dans la suite de l'extrait de code du carrousel, on peut noter la présence de deux boutons de contrôle possédant les classes **carousel-control-prev** et **carousel-control-next** qui permettent à l'utilisateur de naviguer de façon manuelle entre les différentes diapositives du carrousel.

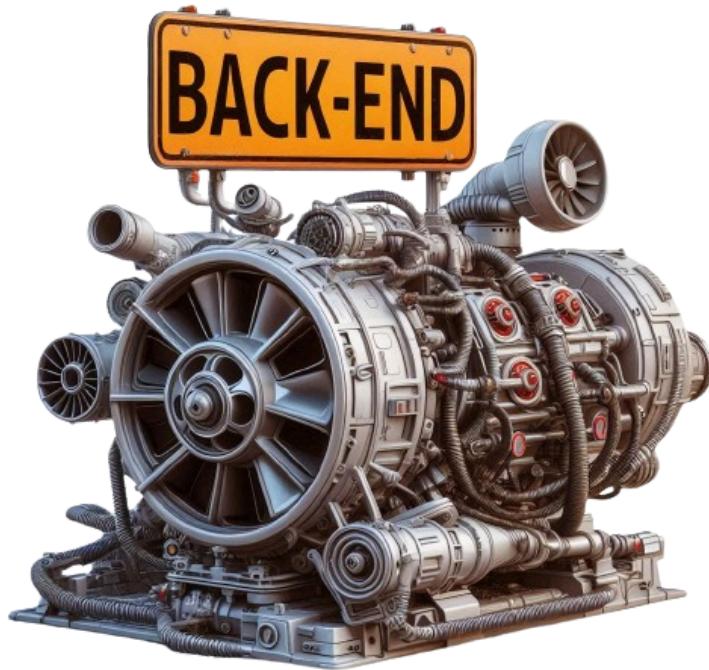
On est donc face à une interface utilisateur dynamique puisque l'utilisateur doit cliquer sur l'un des deux boutons pour modifier l'affichage de la page sur laquelle il se trouve.

Comme autre exemple, nous pourrions également citer JavaScript Vanilla qui permet de rendre les interfaces utilisateur dynamiques en manipulant le DOM (Document Object Model). En effet, en réagissant à des événements (ex : clic sur un bouton, passage de la souris sur un élément, etc...), JavaScript va permettre de mettre à jour le contenu d'une page web sans que cela ne nécessite un rechargeement complet de la page.

Pour rappel, le DOM, [Document Object Model](#), est une interface de programmation (ou API, Application Programming Interface, dans la langue de Shakespeare) qui permet à des scripts d'examiner et de modifier le contenu du navigateur web.

Réalisations personnelles côté back-end

Nous allons maintenant parler des réalisations que j'ai pu effectuer du côté back-end de notre projet O'Galaxy. Une partie qui, je dois l'avouer, m'a plus intéressé de façon globale sur ce projet puisque j'ai effectué la spécialisation Symfony après la phase de socle commun lors de mon cursus de formation chez O'clock.



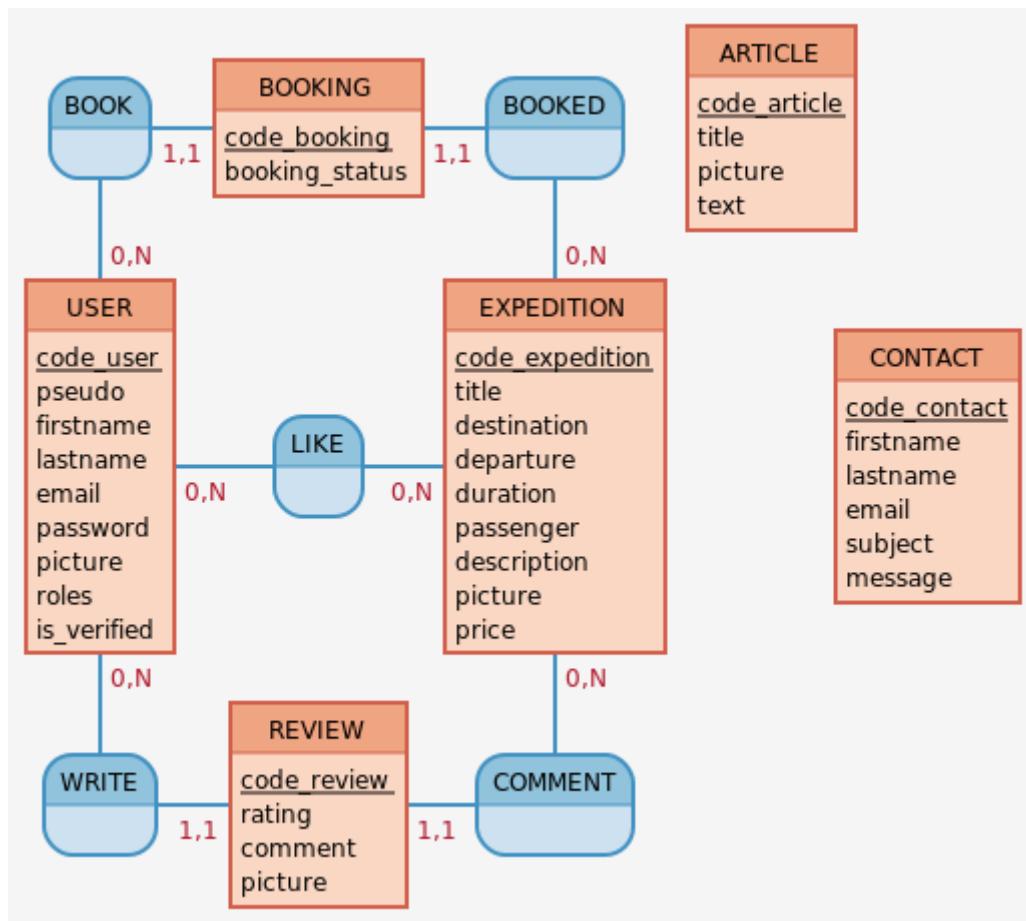
1. Présentation de la base de données

A. Le MCD, Modèle Conceptuel de Données

Pour visualiser au mieux notre projet et comprendre les relations entre les différentes tables de la base de données à créer, nous avons créé un MCD : un Modèle Conceptuel de Données.

Le modèle conceptuel de données est une représentation des données qui permet de décrire facilement les données utilisées par le système d'information et leurs relations.

Pour créer ce MCD, nous avons utilisé l'outil en ligne Mocodo.net. À noter, ce site est dédié à l'apprentissage des bases de données relationnelles. Si nous avions utilisé des données confidentielles, il aurait fallu que nous installions directement sur notre machine et pas en ligne.



Les informations y sont représentées de façon logique grâce à des règles et des diagrammes codifiés :

- Les entités : 1 rectangle correspond à un objet.
exemple : Le rectangle **EXPEDITION** est une entité.
- Les propriétés : la liste des données liées à une entité.
exemple : L'entité **EXPEDITION** a 9 propriétés → `code_expedition`, `title`, `destination`, `departure`, `duration`, `passenger`, `description`, `picture` et `price`
- Les relations : comment les entités sont reliées entre elles.
exemple : Le rectangle bleu aux bords arrondis **booked** décrit la relation existante entre les deux entités **EXPEDITION** & **BOOKING**.
- Les cardinalités : ce sont des caractères (0, 1, N) qui fonctionnent par couple et qui sont présents de chaque côté d'une relation.
exemple : Les cardinalités de chaque côté de la relation **booked** sont 1,1 et 0,N. Cela peut se lire de la façon suivante : une **EXPEDITION** peut être réservée par 0 ou plusieurs **BOOKING** & un **BOOKING** ne peut être lié qu'à une seule **EXPEDITION**.

B. Le MLD, Modèle Logique de Données

Une fois le MCD défini, nous avons mis en place un Modèle Logique de Données, ou MLD, qui est la représentation des données d'un système d'information. À partir du MCD, le MLD introduit donc les éléments, les relations et les détails contextuels qui vont être essentiels à la structuration des données.

Voici le MLD correspondant à notre MCD :

ARTICLE (code_article, title, picture, text)
BOOKING (code_booking, booking_status, #code_user, #code_expedition)
CONTACT (code_contact, firstname, lastname, email, subject, message)
EXPEDITION (code_expedition, title, destination, departure, duration, passenger, description, picture, price)
LIKE (code_like, #code_user, #code_expedition)
REVIEW (code_review, rating, comment, picture, #code_user, #code_expedition)
USER (code_user, pseudo, firstname, lastname, email, password, picture, roles, is_verified)

Côté vocabulaire et conventions :

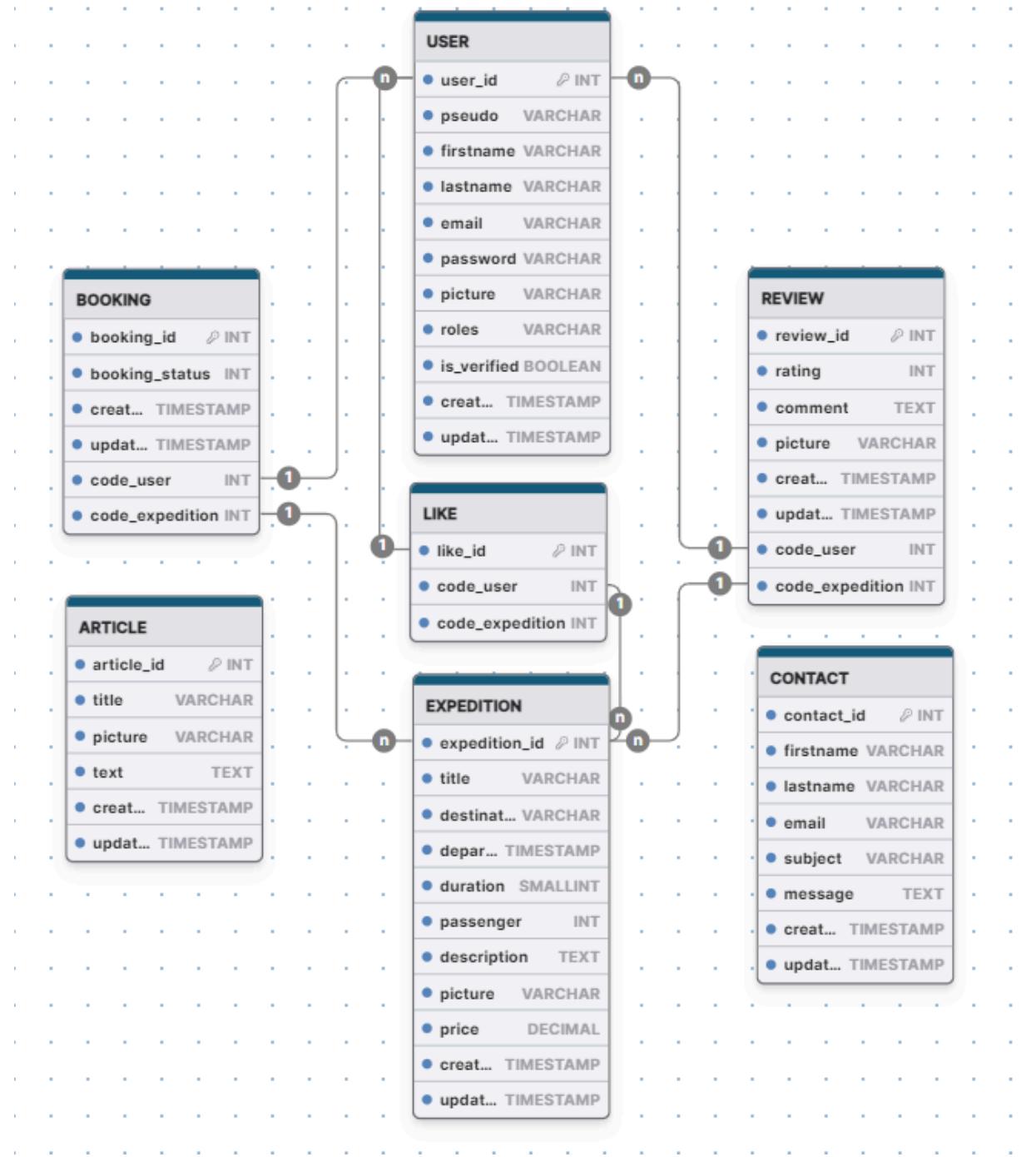
- On note que les entités du MCD deviennent des tables dans le MLD.
- Les clés principales, appelées clés primaires, sont soulignées.
- Les clés étrangères, c'est à dire une clé qui fait référence à la clé primaire d'une autre table, sont représentées en étant préfixées du symbole # dièse.

C. Le MPD, Modèle Physique de Données

Après avoir mis par écrit notre MLD, nous sommes passés à l'étape suivante : la création du Modèle Physique, ou MPD. Le MPD permet de construire la structure finale de la base de données où l'on pourra visualiser les différents liens entre les éléments. Le vocabulaire évolue par rapport au MCD :

- Les entités deviennent des tables.
- Les propriétés deviennent des champs ou des attributs.
- Les propriétés qui se trouvent au milieu d'une relation sont à l'origine d'une nouvelle table ou vont dans la table appropriée en fonction des cardinalités de la relation.
- Les identifiants deviennent des clés : chaque table doit au minimum avoir une clé primaire.
- Les relations et les cardinalités deviennent des champs ou attributs : elles deviennent des clés étrangères, c'est-à-dire des clés qui font référence à la clé primaire d'une autre table.

Voici notre MPD réalisé grâce à l'app [drawDB](#) :



D. Dictionnaire de données

Le dictionnaire de données est un document qui regroupe l'ensemble des entités, des attributs, des relations et des contraintes d'intégrité d'un système. Il fournit des informations détaillées sur chaque table et sur chaque champ de la base de données. Ce document est donc important et sert :

- à documenter la base de données,
- à standardiser la base de données,
- à maintenir la base de données,
- à assurer la qualité des données,
- à aider les développeurs à écrire des requêtes SQL efficaces,
- à définir les niveaux de sécurité et d'accès des différentes parties de la base de données.

Le dictionnaire de données de notre projet est consultable en Annexe n°3 de ce dossier de projet.

E. Script de création de la base de données

Une fois le MPD réalisé, l'outil [drawDB](#) permet d'exporter le MPD en fichier SQL pour créer directement notre base de données via l'outil “Requête SQL” d'Adminer.

Le code SQL fourni montre deux grandes étapes dans la création de la base de données :

1. Dans un premier temps, on note la création des tables avec **CREATE OR REPLACE TABLE**.
2. Dans un second temps, les clés étrangères (**FOREIGN KEY**) sont créées en altérant les tables précédemment créées avec **ALTER TABLE**.

Le script de création de la base de données est consultable en Annexe n°4 de ce dossier de projet.

2. Extraits de code de composants métier

Pour illustrer la création d'un composant métier, nous allons nous intéresser à la gestion des utilisateurs et à l'authentification que j'ai réalisées côté back-end lors de notre projet O'Galaxy.

La gestion et l'authentification des utilisateurs peut être considérée à la fois comme un composant technique, car c'est un indispensable des applications et des sites : proposer un espace personnel aux utilisateurs, mais également comme un composant métier dans la mesure où ce composant peut intégrer des règles de gestion spécifiques à notre site. Par exemple, c'est le cas pour nous, puisque certains utilisateurs ont des rôles et permissions spécifiques.

Dans cette partie, je vais montrer les différentes étapes qui m'ont permis de mettre en place ce composant métier du site O'Galaxy.

A. Installation du security-bundle et du maker-bundle

Tout d'abord, j'ai dû m'assurer que nous avions bien installé deux paquets nécessaires à la création de ce composant :

- Le **security-bundle** de Symfony qui permet d'assurer la sécurité de notre site. En effet, ce bundle permet de gérer l'authentification, les autorisations, les pare-feux, l'encodage des mots de passe, la gestion des utilisateurs, les contrôles d'accès (ACL, access control list) et la sécurité des requêtes avec notamment une protection contre les attaques CSRF (Cross-Site Request Forgery) et XSS (Cross-Site Scripting).

L'installation de ce bundle s'effectue avec la commande :

composer require symfony/security-bundle

- Le **maker-bundle** de Symfony qui permet de faciliter le développement en générant du code de façon automatique pour des tâches comme la création d'entités, la gestion des migrations, la création de contrôleurs, la création de formulaires, etc...

L'installation de ce bundle s'effectue avec la commande :

composer require symfony/maker-bundle --dev

B. Configuration du système de sécurité

Je dois ensuite m'assurer de la bonne configuration du fichier **security.yaml** situé dans le dossier **config/packages**. Dans ce fichier (voir capture p. 33), voici le rôle des différentes sections :

- **password_hashers** configure les hashers de mot de passe utilisés pour sécuriser les mots de passe des utilisateurs
- **providers** permet de définir les fournisseurs de données utilisés pour charger les utilisateurs, ici c'est la classe User qui va permettre d'identifier un utilisateur en se basant sur la propriété email
- **firewalls** permet de configurer les pare-feux de sécurité pour le site
- **access_control** permet de contrôler quels rôles ont accès aux différentes parties de notre site
- **when@test** contient des configurations spécifiques qui sont appliquées lors de l'exécution de tests

Pour le moment, la section qui nous intéresse le plus est **providers** puisque c'est elle qui va définir d'où sont chargés les utilisateurs : dans notre cas, ils seront chargés depuis l'**entité User** et c'est l'email de l'utilisateur (**property**) qui sera utilisé pour rechercher un utilisateur. La configuration de cette section est essentielle pour que Symfony sache où trouver les informations utilisateurs nécessaires à l'authentification et aux autorisations.

```
● ● ●

security:
    # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
    # https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider
    providers:
        # used to reload user from session & other features (e.g. switch_user)
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            lazy: true
            provider: app_user_provider
            custom_authenticator: App\Security\LoginFormAuthenticator
            logout:
                path: app_logout
            remember_me:
                secret: '%kernel.secret%'
                lifetime: 604800
                path: /
        # Easy way to control access for large sections of your site
        # Note: Only the *first* access control that matches will be used
        access_control:
            - { path: ^/expedition/favorites, roles: ROLE_USER }
            - { path: ^/admin, roles: [ROLE_ADMIN] }

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
                time_cost: 3 # Lowest possible value for argon
                memory_cost: 10 # Lowest possible value for argon
```

C. Création de l'entité User

J'utilise ensuite le maker-bundle de Symfony pour créer l'entité User avec la commande : **php bin/console make:user**

Voici ce qui apparaît dans le terminal quand on lance cette commande :

```
$ php bin/console make:user
The name of the security user class (e.g. User) [User]:
> User

Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:
> yes

Enter a property name that will be the unique "display" name for the user (e.g. email,
username, uuid) [email]:
> email

Will this app need to hash/check user passwords? Choose No if passwords are not needed or will
be checked/hashed by some other system (e.g. a single sign-on server).

Does this app need to hash/check user passwords? (yes/no) [yes]:
> yes

created: src/Entity/User.php
created: src/Repository/UserRepository.php
updated: src/Entity/User.php
updated: config/packages/security.yaml
```

Dans un premier temps, le maker nous demande comment on souhaite nommer la classe sécurisée User et nous propose (la proposition est entre les crochets), par défaut, de l'appeler **User**. Parfait, c'est ce que qu'on choisit de manière logique.

Le maker nous demande ensuite si l'on souhaite stocker les informations utilisateurs en base de données via Doctrine (l'ORM de Symfony dont nous parlerons plus tard dans le dossier de projet) : **oui**, nous voulons stocker ces informations en base de données.

Nous devons ensuite choisir la propriété qui sera utilisée par Symfony pour retrouver un utilisateur dans la base de données. Ici nous rentrons la propriété **email**.

Enfin, le maker nous demande si nous souhaitons hasher les mots de passe des utilisateurs : ici aussi nous choisissons **oui** car cela permet d'assurer la sécurité des données et d'éviter des attaques courantes par force brute ou par dictionnaire.

C'est bon, notre entité est créée et l'on note plusieurs choses qui apparaissent dans le terminal une fois qu'on a répondu à la dernière question du maker :

- Création d'un fichier **src/Entity/User.php** : création de l'entité User.
- Création d'un fichier **src/Repository/UserRepository.php** : création du fichier qui permettra d'effectuer des interactions avec la base de données.
- Mise à jour du fichier **src/Entity/User.php** : mise à jour du fichier de l'entité pour importer la classe **UserRepository.php**.
- Mise à jour du fichier **config/packages/security.yaml** : mise à jour du providers si ce n'était pas déjà configuré auparavant.



```
<?php

namespace App\Entity;

use DateTimeImmutable;
use Doctrine\ORM\Mapping as ORM;
use App\Repository\UserRepository;
use Doctrine\Common\Collections\Collection;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;
use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface;
use Symfony\Component\Validator\Constraints as Assert;

#[ORM\Entity(repositoryClass: UserRepository::class)]
#[ORM\UniqueConstraint(name: 'UNIQ_IDENTIFIER_EMAIL', fields: ['email'])]
#[UniqueEntity(fields: ['email'], message: 'Il existe déjà un compte relié à cette adresse')]
#[ORM\HasLifecycleCallbacks]
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    ...
}
```

J'ai donc mon **entité User** qui est créée et qui implémente bien **UserInterface** et **PasswordAuthenticatedUserInterface**.

UserInterface est une interface fournie par Symfony qui représente un User dans le système de sécurité de Symfony. Cette interface permet d'avoir accès à des méthodes pour retrouver des informations sur un utilisateur comme **getUsername()** ou **getRoles()**.

PasswordAuthenticatedUserInterface est une interface pour les classes d'utilisateurs qui peuvent être authentifiées via un mot de passe dans le composant Security de Symfony. Cette interface définit une méthode **getPassword()** que les classes d'utilisateurs doivent implémenter pour récupérer le mot de passe de l'utilisateur.

Ce mot de passe est ensuite comparé avec le mot de passe saisi par l'utilisateur lors de la tentative d'authentification, généralement après avoir été hashé pour des raisons de sécurité.

D. Mise à jour de la base de données

Quand on crée une entité sur Symfony, il est ensuite nécessaire d'effectuer une migration pour mettre à jour cette dernière.

Pour cela, j'ouvre un terminal et je tape les deux commandes suivantes :

```
php bin/console make:migration  
php bin/console doctrine:migrations:migrate
```

La première commande est utilisée pour créer une migration. La migration est un script qui contient les requêtes SQL permettant de mettre à jour la structure de notre base de données.

La seconde commande, quant à elle, permet d'exécuter le script SQL contenu dans la migration afin de mettre à jour la structure de notre base de données.

E. Création du formulaire d'authentification

Maintenant que j'ai une classe **User** qui permet à mes utilisateurs d'avoir un identifiant et un mot de passe. Il faut qu'un utilisateur ayant déjà un compte sur mon site puisse se connecter.

Pour cela, nous allons à nouveau ouvrir le terminal pour entrer la commande suivante :

```
php bin/console make:auth
```

Cette commande va permettre de générer de façon automatique un système d'authentification sécurisé pour notre site.

Concrètement, cette commande crée un contrôleur d'authentification ainsi que des templates Twig pour le formulaire de connexion.

```

● ● ●

$ php bin/console make:auth
What style of authentication do you want? [Empty authenticator]:
[0] Empty authenticator
[1] Login form authenticator
> 1

The class name of the authenticator to create (e.g. AppCustomAuthenticator):
> LoginFormAuthenticator

Choose a name for the controller class (e.g. SecurityController) [SecurityController]:
> SecurityController

Do you want to generate a '/logout' URL? (yes/no) [yes]:
> yes

created: src/Security/LoginFormAuthenticator.php
updated: config/packages/security.yaml
created: src/Controller/SecurityController.php
created: templates/security/login.html.twig

Success!

Next:
- Customize your new authenticator.
- Finish the redirect "TODO" in the
App\Security\LoginFormAuthenticator::onAuthenticationSuccess() method.
- Review & adapt the login template: templates/security/login.html.twig.

```

Ici, le maker de Symfony nous demande plusieurs choses :

- Le type d'authentification souhaité : je choisis l'option 1, **Login form authenticator**
- Le nom de classe de l'authentificateur que l'on veut créer dans le dossier **src/Security** : je choisis un nom clair et logique → **LoginFormAuthenticator**
- Si l'on souhaite générer une URL /logout : **oui**, nous le souhaitons

Comme pour la création de l'entité User que l'on a vu précédemment, plusieurs choses apparaissent dans le terminal une fois qu'on a répondu à la dernière question du maker :

- Création du fichier **LoginFormAuthenticator.php** dans le dossier **src/Security** : c'est la classe qui va s'occuper de gérer le processus d'authentification, ce qui implique la récupération des informations d'identification de l'utilisateur, la vérification de ces informations d'identification et la création de la session utilisateur. Dans la méthode **onAuthenticationSuccess()** de cet Authenticator, il ne faut pas oublier, comme l'indique le message dans le terminal, d'indiquer la redirection voulue quand l'authentification de l'utilisateur est validée.

```

● ● ●

public function onAuthenticationSuccess(Request $request, TokenInterface $token, string
$firewallName): ?Response
{
    if ($targetPath = $this->getTargetPath($request->getSession(), $firewallName)) {
        return new RedirectResponse($targetPath);
    }

    // For example:
    return new RedirectResponse($this->urlGenerator->generate('app_home'));
    // throw new \Exception('provide a valid redirect inside '.__FILE__);
}

```

- Mise à jour du fichier **security.yaml** dans le dossier **config/packages** : ici, la mise à jour permet d'ajouter quelques lignes à la section pare-feu pour indiquer que nous avons bien un authenticator ainsi qu'un chemin vers lequel renvoyer en cas de déconnexion (voir capture p. 33).
- Création du fichier **SecurityController.php** dans le dossier **src/Controller** : il s'agit d'un contrôleur dans lequel vont être générées les méthodes **login** et **logout**.
- Création du template **login.html.twig** dans le dossier **templates/security** : c'est dans ce template que va être créé le formulaire de connexion. C'est cette vue qui sera affichée quand on cliquera sur un bouton / un lien qui mène à la route **/connexion**, associée à la méthode **login** dans le **SecurityController**.

F. Affichage du formulaire de connexion sur le site O'Galaxy

Maintenant que tout est en place pour valider la connexion d'un utilisateur, il faut effectuer quelques petites customisations sur le template Twig qui a été créé afin de rendre notre formulaire de connexion visuellement cohérent avec le reste du site O'Galaxy.

```

{% extends 'base.html.twig' %}

{% block title %}Connexion{% endblock %}

{% block body %}
<!-- Display the login -->
<section id="login" aria-labelledby="to-log-in">
<div class="container col-12 col-lg-6 my-4">
    <h3 id="list__bigtitle" class="display-4">Se connecter</h3>
    <form method="post" novalidate>
        {% if error %}
            <div class="alert alert-danger" role="alert">{{ error.messageKey|trans(error.messageData, 'security') }}</div>
        {% endif %}

        {% if app.user %}
            <div class="login__color mb-3">
                Bienvenue {{ app.user.userIdentifier }}, <a href="{{ path('app_logout') }}" role="button"
                    aria-label="Bouton pour se déconnecter">Déconnexion</a>
            </div>
        {% endif %}

        <h2 class="h3 mb-3 font-weight-normal login__color">Veuillez vous connecter</h2>
        <div class="col-8">
            <label for="inputEmail" class="login__color">Adresse mail</label>
            <input type="email" value="{{ last_username }}" name="email" id="inputEmail" class="form-control mb-3"
                autocomplete="email" required autofocus>
            <label for="inputPassword" class="login__color">Mot de passe</label>
            <input type="password" name="password" id="inputPassword" class="form-control mb-3"
                autocomplete="current-password" required>

            <input type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}>
        </div>
        <div class="checkbox mt-3 mb-3">
            <label class="login__color">
                <input type="checkbox" name="_remember_me" role="button"
                    aria-label="Bouton pour mémoriser sa connexion"> Se souvenir de moi</label>
            </div>
            <button class="btn login__btn" type="submit" aria-label="Bouton pour se connecter">Connexion</button>
        </div>
        <h2 class="h3 mt-5 mb-3 font-weight-normal login__color">Je ne possède pas de compte</h2>
        <button class="btn login__btn" onclick="window.location.href='{{ path('app_register') }}'" aria-label="Bouton pour se créer un compte">Créer un compte</button>
    </div>
</section>
{% endblock %}

```

Ce qui permet alors d'afficher cet écran à l'utilisateur quand il se rend sur l'url **/connexion**.

The screenshot shows the 'Se connecter' (Log in) page of the O'Galaxy website. At the top, there is a navigation bar with the logo 'O'GALAXY', links for 'Nos expéditions', 'Notre Histoire', 'Actualités', and 'FAQ', a search bar with placeholder 'Votre recherche...', and a magnifying glass icon. The main section has a dark background with orange text. It says 'SE CONNECTER' in large letters, followed by 'Veuillez vous connecter'. There are two input fields: 'Adresse mail' and 'Mot de passe', each with a red '...' button to the right. Below them is a checkbox labeled 'Se souvenir de moi'. A blue 'Connexion' button is centered below the inputs. To the left of the inputs, there is a link 'Je n'ai pas de compte' and a blue 'Créer un compte' button. At the bottom of the page, there is a footer with links to 'O'GALAXY', 'Nous contacter', 'Carrières', 'Mentions légales', and 'Confidentialité'. To the right, it says 'MAY THE FORCE BE WITH YOU!' and 'Vivez une aventure hors du commun, vibrez au son du vaisseau et partez vers les étoiles !'. It features social media icons for Twitter, Facebook, Instagram, and YouTube. A small copyright notice at the bottom left reads '© 2024 O'Galaxy All rights reserved.'

G. Création du formulaire d'enregistrement

Mon utilisateur peut se connecter, mais maintenant, nous voudrions qu'il puisse se créer un compte et s'enregistrer directement depuis le site.

Pour cela, j'utilise à nouveau le maker de Symfony en lançant la commande :

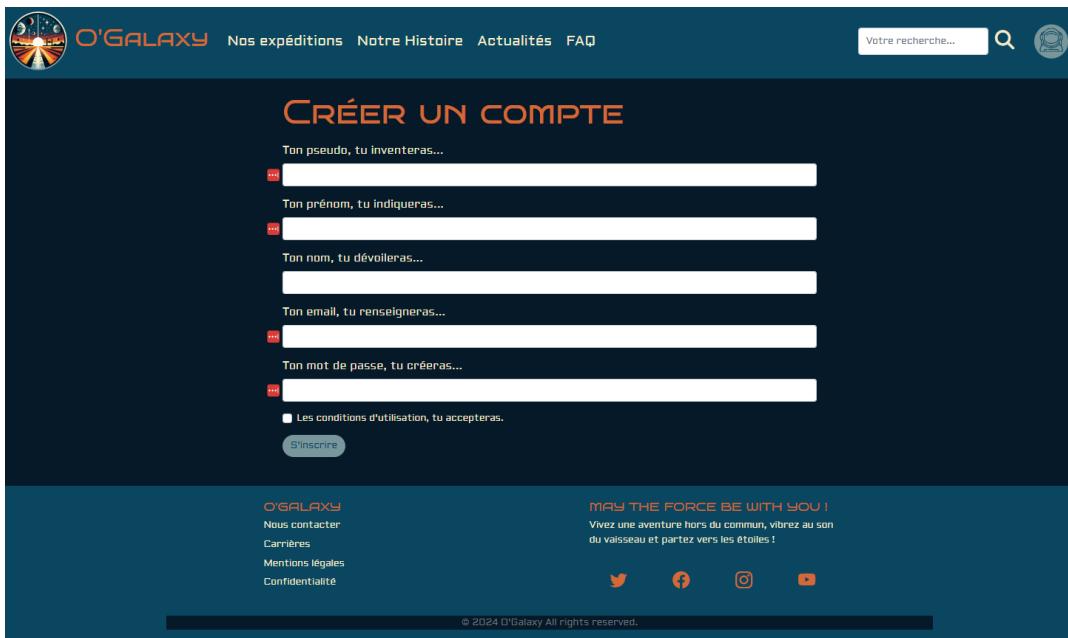
```
php bin/console make:registration-form
```

Cette commande va nous permettre de générer un système d'inscription des utilisateurs dans notre site en Symfony.

Je ne vais pas rentrer dans le détail comme pour l'authenticator mais globalement, voici ce que fait cette commande :

1. Elle génère un contrôleur d'inscription des utilisateurs que l'on va appeler **RegistrationController**. Celui-ci contient les méthodes nécessaires pour afficher un formulaire d'inscription et peut traiter les formulaires soumis.
2. Elle crée un formulaire d'inscription qui est nommé **RegistrationFormType** où sont définis les champs nécessaires pour qu'un utilisateur s'inscrive (email, mot de passe par exemple).
3. Elle inclut une logique de sécurité avec l'encodage des mots de passe.

4. Elle configure le contrôleur **RegistrationController** et le formulaire **RegistrationFormType** pour interagir avec l'**entité User** créée préalablement. Les données soumises via ce formulaire seront correctement validées et persistées en base de données si elles sont correctes.
5. Elle génère un template Twig **register.html.twig** dans le dossier **templates/registration** pour afficher le formulaire d'inscription aux utilisateurs quand ils se rendent sur l'url **/inscription**.



3. Extraits de code de composants d'accès aux données

Pour illustrer la création de composants d'accès aux données, je vais reprendre l'exemple des utilisateurs.

J'ai créé mon **entité User** et je souhaite maintenant accéder aux données de cette entité de ma base de données pour que les administrateurs du site O'Galaxy puissent gérer les utilisateurs dans le back office.

Pour afficher ces données dans le back office, il va procéder à plusieurs étapes pour les récupérer puis les afficher.

A. Création de la route dans le UserController

Je vais donc dans mon fichier **UserController.php** du dossier **src/Controller/Back** afin de créer la route qui permettra d'afficher les données et la méthode ***list()*** liée à cette route.

```

<?php

namespace App\Controller\Back;

use App\Entity\User;
use App\Form\UserType;
use App\Repository\UserRepository;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

#[Route('/admin/utilisateur', name: 'app_admin_user_')]
class UserController extends AbstractController
{
    /**
     * Method to show the list of all users
     *
     * @param UserRepository $userRepository
     * @return Response
     */

    #[Route('/', name: 'list', methods: ['GET'])]
    public function list(UserRepository $userRepository): Response
    {
        $allUsers = $userRepository->findAll();

        return $this->render('back/user/list.html.twig', [
            'userList' => $allUsers
        ]);
    }
}

```

Le framework Symfony fonctionne sur le design pattern (patron de conception dans la langue de Molière) MVC Modèle-Vue-Contrôleur. Notre contrôleur **UserController** va donc demander au modèle **UserRepository** les données qu'il souhaite envoyer à la vue **list.html.twig** située dans le dossier **templates/back/user**.

C'est ce qui est fait avec la ligne :

```
$allUsers = $userRepository->findAll();
```

Ici, je crée une variable **\$allUsers** qui est un tableau dans lequel je vais stocker l'ensemble des utilisateurs qui se trouvent dans la base de données. En effet, je fais appel à la méthode **findAll()** du **UserRepository** qui est une méthode standard fournie par l'**ORM Doctrine** pour récupérer tous les enregistrements d'une entité.

Pour rappel, l'**ORM (Object Relational Mapping) Doctrine** est une bibliothèque PHP qui nous permet de gérer les interactions avec une base de données relationnelle en utilisant une approche orientée objet.

Les données stockées dans la variable **\$allUsers** sont ensuite passées à la vue comme l'indique la fin de la méthode **list()**. Les données sont encapsulées dans un tableau associatif et seront accessibles dans la vue via la clé **userList**.

```
return $this->render('back/user/list.html.twig', [
    'userList' => $allUsers
]);
```

B. Création d'un template Twig pour afficher la vue

Une fois que les données sont passées à la vue, je vais pouvoir les afficher dans le template **list.html.twig** située dans le dossier **templates/back/user** que j'ai créé au préalable (voir Annexe n° 5).

Pour afficher les données dynamiquement dans le template, je place une boucle for au sein de mon code html pour aller chercher les données stockées dans la variable **\$allUsers**.

#	Pseudo	Prénom	Nom	Mail	Voir/Modifier/Supprimer
1	daphney.mcglynn	Mathias	Fritsch	nicolas.norman@gmail.com	
2	elyse53	Christelle	Hessel	flatley.johnathan@yahoo.com	
3	laverne.dooley	Carole	Crona	zane.schinner@waker.com	
4	greenhort.hal	Thad	Wuckert	mertz.tillman@hotmail.com	
5	reynals67	Bo	Johns	nkirin@hotmail.com	
6	wiza.jula	Kalley	Breitenberg	skozev@yahoo.com	
7	pzbonicak	Johnson	McGlynn	wchristiansen@gmail.com	
8	lince	Elisa	Bogdan	fmarquart@hotmail.com	
9	catherine.koeppin	Anabelle	Herman	krystina.lind@yahoo.com	
10	koss.michel	Gladys	Ratke	hyatt.gabe@kuuize.org	
11	Liam	Liam	Iourtfaizer	liam@iourtfaizer.com	
12	Leia	Leia	Situations	pierre.morin@oclock.school	

Grâce à cela, nous pouvons désormais afficher la liste complète des utilisateurs aux administrateurs de notre site O'Galaxy quand ils se rendent sur l'url **/admin/utilisateurs**.

Présentation des éléments de sécurité

1. La sécurité, composante essentielle pour tout site web

La sécurité est un élément essentiel quand on développe un site web, que ce soit avec le framework Symfony ou tout autre framework ou langage permettant de développer un site web ou une application.

La sécurité permet tout d'abord de **protéger les données sensibles** comme les données personnelles, les informations de paiement, les identifiants de connexion.

La protection des données des utilisateurs peut être une conformité légale, comme en Europe avec le Règlement Général de Protection des Données (RGPD). Le non-respect de ce règlement pouvant entraîner de lourdes sanctions financières.

La sécurité d'un site web permet également de se prémunir contre les attaques courantes qui existent sur internet : attaques par injection SQL, attaques Cross-Site Scripting (XSS), attaques Cross-Site Request Forgery (CSRF). Un site sécurisé permet d'éviter ces attaques connues et de protéger les données des utilisateurs.

Assurer une bonne sécurité sur son site web permet également de maintenir la confiance des utilisateurs. Un site peu sécurisé peut vite nuire à la réputation d'une entreprise, et donc par extension à son chiffre d'affaires. En sécurisant son site web, on s'assure de ne pas subir de pertes financières directes, avec par exemple des vols de fonds via des attaques de hackers, ou indirectes, avec la perte de clients.

2. Avec Symfony, des outils robustes pour la sécurité

En utilisant le framework Symfony pour développer notre site web, nous avons accès à des outils robustes pour assurer la sécurité de notre site O'Galaxy. Passons en revue quelques éléments que nous avons mis en place lors de la phase de développement de notre projet.

A. Authentification et autorisations

Comme nous avons pu le voir dans les réalisations personnelles côté back-end, l'authentification permet de vérifier l'identité des utilisateurs d'un site web. Dans le framework Symfony, c'est le [**Security Component**](#) que nous avons utilisé pour gérer cet aspect de sécurité. Ce bundle Symfony fournit les éléments nécessaires pour mettre en place l'authentification et les autorisations sur notre site web.

La configuration du Security Component se fait dans le fichier **security.yaml** qui se situe dans le dossier **config/packages** (voir capture p. 33).

L'authentification se configure via la mise en place de providers qui vont avoir pour mission de récupérer les informations de connexion de nos utilisateurs.

Les autorisations se configurent par l'attribution de rôles aux utilisateurs et via la configuration de la section **access_control** dans le fichier **security.yaml**.

B. Protection contre les attaques CSRF, les attaques par injection SQL et les attaques XSS

Les attaques CSRF (Cross-Site Request Forgery) permettent à un hacker de réaliser des actions sur un site web via un utilisateur authentifié et, bien évidemment, sans le consentement de ce dernier.

Pour se prémunir de ce type d'attaques, Symfony utilise des **tokens CSRF** pour protéger les formulaires. Pour expliquer brièvement, un **token CSRF** est un jeton unique et secret qui est placé dans chaque formulaire. Quand un utilisateur soumet le formulaire, Symfony va venir vérifier ce jeton pour être certain que la requête vient bien du site et pas d'un site externe, ce qui indiquerait une tentative d'attaque. La **protection CSRF** est activée par défaut pour tous les formulaires dans Symfony quand on utilise le **composant Form**.

Pour les attaques par injections SQL, nous utilisons l'ORM Doctrine qui utilise par défaut des **requêtes préparées** lors de l'exécution de requêtes SQL. En faisant cela, l'ORM sépare les données des instructions SQL ce qui permet d'empêcher d'éventuels hackers de mener des attaques en injectant du code SQL malveillant.

Enfin pour se protéger des attaques par Cross-Site Scripting (XSS), l'utilisation du moteur de template Twig permet d'échapper automatiquement les sorties par défaut, c'est-à-dire que les données affichées dans le template sont traitées pour supprimer les caractères considérés comme dangereux.

Par exemple, imaginons que nous avons une variable {{ user.name }} dans notre template Twig pour afficher le nom de l'utilisateur de manière dynamique. Un hacker arrive et tente de faire passer la balise <script> à la place de la variable pour exécuter du code JavaScript malveillant. Et bien l'échappement automatique de Twig va venir transformer la balise en <script> ce qui va empêcher l'input du hacker d'être exécuté comme du code JavaScript et cela va faire tomber son attaque à l'eau.

C. Hasher les mots de passe

En utilisant le **Security Component** et le **maker de Symfony** pour créer notre formulaire d'enregistrement des utilisateurs, Symfony va créer un fichier **RegistrationController** dans le dossier **src/Controller** qui utilise le service **UserPasswordHasherInterface** pour encoder les mots de passe quand les utilisateurs s'enregistrent. Quand le formulaire d'enregistrement est soumis, c'est le mot de passe hashé qui est stocké en base de données.

Quand un utilisateur se connecte, Symfony vérifie le mot de passe en comparant le mot de passe saisi et le mot de passe hashé qui a été stocké en base de données lors de l'enregistrement.

D. Validation des données

Pour valider nos données, nous avons utilisé le **composant Validator** de Symfony qui permet de valider et filtrer les données dans un site web.

Par exemple, dans l'entité User (visible en Annexe n°6), nous avons utilisé de nombreuses contraintes prédéfinies de Symfony pour valider les différents types de données.

Par exemple, pour l'email, nous avons utilisé les contraintes NotBlank et Email. Si je fais une erreur volontairement en ne mettant pas un email mais le mot "ogalaxy", voici ce qui se passe :

The screenshot shows a dark-themed web form titled "CRÉER UN COMPTE". The fields and their current values are:

- Ton pseudo, tu inventeras...: Pierrotlalune
- Ton prénom, tu indiqueras...: Pierrot
- Ton nom, tu dévoileras...: La Lune
- Ton email, tu renseigneras...: ogalaxy (highlighted with a red border)
- Ton mot de passe, tu crées...: (empty field)

A validation message appears above the email field: "Veuillez inclure "@" dans l'adresse e-mail. Il manque un symbole "@" dans "ogalaxy"."

At the bottom of the form, there is a checkbox labeled "Les conditions d'utilisation, tu accepteras." followed by a "S'inscrire" button.

Jeu d'essai

Pour s'assurer du bon fonctionnement de notre site, nous pouvons créer ce qu'on appelle un jeu d'essai. Pour tester que notre site fonctionne, nous allons nous mettre dans la peau d'une nouvelle utilisatrice qui souhaite se créer un compte pour effectuer une réservation.

Notre nouvelle utilisatrice est une influenceuse très connue dans le fandom Star Wars puisqu'il s'agit ni plus, ni moins que de Leïa Situations.

The image shows two screenshots of the O'Galaxy website. The left screenshot displays an expedition to Tatooine, featuring a circular image of the双子星 (Tatooine) with two suns, a detailed description of the planet, and booking information (November 2025 departure, 7 weeks stay, 36,000 € price). The right screenshot shows the 'SE CONNECTER' (Log In) page, which includes fields for email and password, a 'Se souvenir de moi' (Remember me) checkbox, and a 'Connexion' (Log In) button. Below it is a 'Je n'ai pas de compte' (I don't have an account) section with a 'Créer un compte' (Create an account) button.

En arrivant sur le site, elle est très intéressée par l'expédition prévue sur Tatooine en novembre 2025. Elle clique donc sur le bouton "**Pour réserver, merci de vous connecter**". Comme le bouton le laissait entendre, elle arrive sur la page de connexion du site puisqu'il faut être connecté à son compte pour pouvoir effectuer une réservation. N'ayant pas encore de compte sur O'Galaxy, elle clique sur le bouton "**Créer un compte**". Je lui prête gentiment mon mail car sa boîte mail ne fonctionnait plus.

The image shows the 'CRÉER UN COMPTE' (Create an account) form. It consists of several input fields with placeholder text: 'Ton pseudo, tu inventeras...' (Your pseudonym, you will invent it...) with 'Leïakouza', 'Ton prénom, tu indiqueras...' (Your first name, you will indicate it...) with 'Leïa', 'Ton nom, tu dévoileras...' (Your last name, you will reveal it...) with 'Situations', 'Ton email, tu renseigneras...' (Your email, you will enter it...) with 'pierre.morin@oclock.school', and 'Ton mot de passe, tu créeras...' (Your password, you will create it...) with a redacted password. Below the fields is a checked checkbox for 'Les conditions d'utilisation, tu accepteras.' (The terms of use, you will accept.) and a 'S'inscrire' (Sign up) button.

C'est bon, l'inscription fonctionne puisque j'ai bien reçu un mail de confirmation à l'adresse pierre.morin@oclock.school.

Elle clique alors sur le lien “**Je confirme mon email**” qui la renvoie sur son compte O’Galaxy. Un message flash lui indique que son compte a bien été vérifié ! Son compte est donc créé.

Confirmation de la création de votre compte O'Galaxy 

De O'Galaxy Mail Bot
À pierre.morin@oclock.school
Date Aujourd'hui 14:47
[Résumé](#) [En-têtes](#) [Texte en clair](#)

Hello futur astronaute !



Vous êtes sur le point de rejoindre la communauté O'Galaxy !

Nous sommes ravis de vous accueillir parmi nous.

Pour confirmer votre inscription, merci de cliquer sur le lien suivant :
[Je confirme mon email.](#)
Le lien expirera dans 1 heure.

Vers l'infini et au-delà !



Votre compte a bien été vérifié !

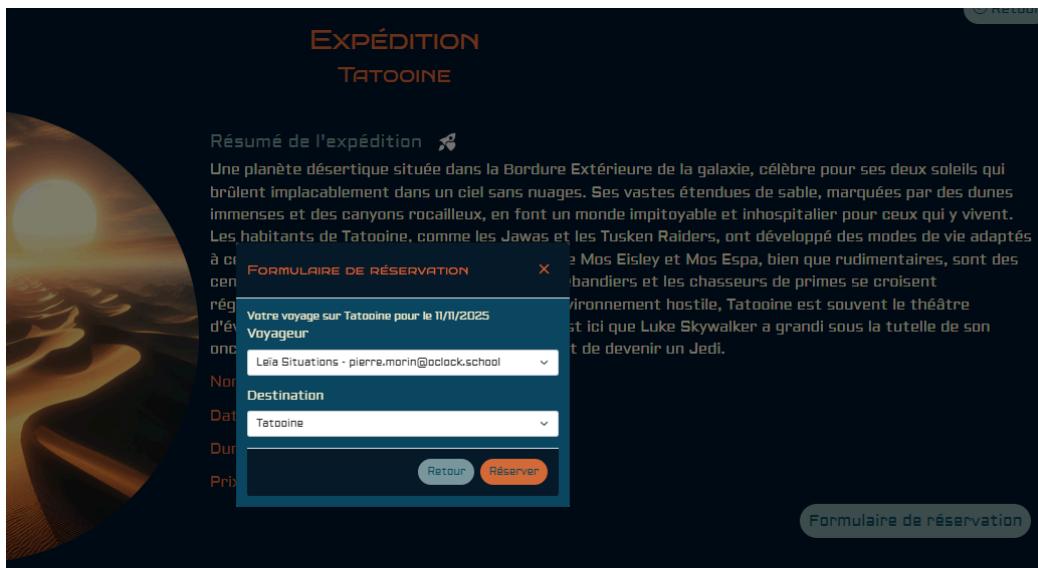
Leïa Situations
pierre.morin@oclock.school
[Modifier](#)

MES RÉSERVATIONS EN ATTENTE

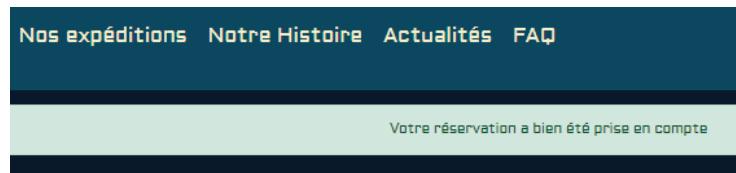
MES RÉSERVATIONS CONFIRMÉES

MES RÉSERVATIONS ANNULÉES

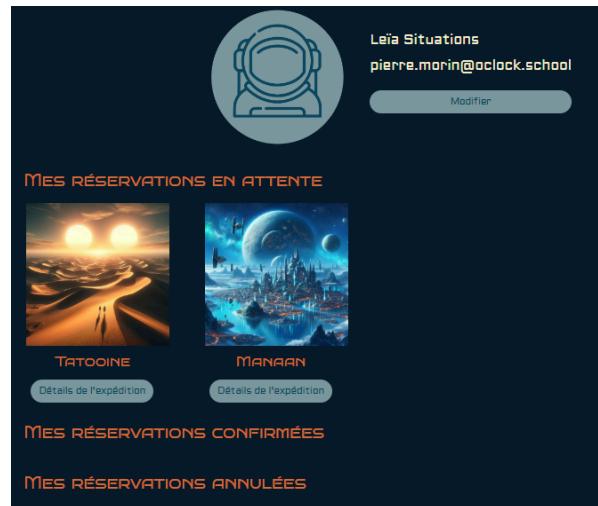
Le but premier de son inscription étant de réserver son voyage sur Tatooine, elle retourne sur ladite page expédition et, étant connectée, clique sur le bouton “**Formulaire de réservation**” (voir capture p. 48). Les informations de mail et de destination étant correctes, elle clique sur le bouton “**Réserver**”.



Un message flash apparaît sur la page pour lui indiquer que sa réservation est bien prise en compte.



Elle fait de même avec l'expédition prévue sur Manaan en avril 2025 et se rend sur sa page profil pour voir si les réservations sont bien prises en compte. Et c'est bien le cas, elle peut voir que les deux réservations sont bien “**En attente**” sur son profil.



Si l'on prend un compte administrateur pouvant se connecter au back office, on vérifie deux choses :

- Si une utilisatrice du nom de Leïa Situations existe bien :

UTILISATEURS

Liste des utilisateurs					
#	Prénom Nom	Elisa	Bogdan	fmarquardt@hotmail.com	Voir/Modifier/Supprimer
9	catherine.koelpin	Anabelle	Herman	krystina.lind@yahoo.com	
10	koss.michel	Gladys	Ratke	hyatt.gabe@kunze.org	
11	Liam	Liam	Iourfazer	liam@iourfazer.com	
13	Leïakouza	Leïa	Situations	pierre.morin@oclock.school	

- Si deux réservations au nom de Leïa Situations ont bien été effectuées :

RÉSERVATIONS

Liste des réservations				
#	Prénom Nom	Destination	Statut	Voir/Modifier/Supprimer
1	Bo Johns	Manaan	Validée	
3	Leïa Situations	Tatooine	En attente	
4	Leïa Situations	Manaan	En attente	

C'est bien le cas ! Nous avons donc un site fonctionnel sur les fonctionnalités que nous avons choisies de tester. Dernière petite chose, pour s'assurer que les autorisations sont bien en place : Leïa Situations veut s'essayer au hacking et tente de forcer l'url **/admin** pour accéder au back office :

The screenshot shows a Symfony error page with a red header bar containing the text "AccessDeniedException > AccessDeniedHttpException" and "HTTP 403 Forbidden". Below the header, there is a large red area with the text "Access Denied." and a cartoon ghost icon with a speech bubble saying "Exception!". At the bottom, there are navigation links for "Exceptions 2", "Logs 1", and "Stack Traces 2". A detailed stack trace is visible in a grey box at the bottom:

```

Symfony\Component\HttpKernel\Exception\AccessDeniedHttpException
▶ Show exception properties

```

Leïa tombe alors sur une page lui indiquant une erreur 403 : Access Denied. Le serveur a compris la requête mais ne lui autorise pas l'accès car elle n'a que le ROLE_USER et pas le ROLE_ADMIN nécessaire pour accéder à cette partie du site.

Veille sectorielle et veille sécurité

Quand on travaille dans le développement web, il est important de garder un œil sur les actualités du secteur, sur les actualités des frameworks qu'on utilise, et sur les actualités côté sécurité puisque de nouvelles failles peuvent voir le jour régulièrement.

1. Veille technologique et sectorielle

Côté veille technologique, j'ai beaucoup utilisé les documentations officielles des langages et des frameworks que j'ai pu utiliser tout au long du projet :

- MDN HTML : <https://developer.mozilla.org/fr/docs/Web/HTML>
- MDN CSS : <https://developer.mozilla.org/fr/docs/Web/CSS>
- MDN JavaScript : <https://developer.mozilla.org/fr/docs/Web/JavaScript>
- Documentation de Bootstrap :
<https://getbootstrap.com/docs/5.3/getting-started/introduction/>
- Documentation de Symfony : <https://symfony.com/doc/current/index.html>
- Documentation de PHP : <https://www.php.net/manual/fr/index.php>
- Documentation SQL : <https://sql.sh/>

Je me suis également créé une veille dédiée au développement web sur X (ex-Twitter) : <https://twitter.com/i/lists/1714391429454155942>. Cette veille regroupe des experts français et étrangers dans le domaine du développement web.

Enfin, je me suis également tenu au courant de l'actualité du secteur grâce :

- à des chaînes YouTube comme [Grafikart](#), [Melvynx](#), [Nouvelle Techno](#), etc...
- à des sites comme le blog de [CodeGarage.fr](#), le blog [SymfonyCasts.com](#), le site [dev.to](#), etc...

2. Veille sécurité

La sécurité du site O'Galaxy étant également un point important de notre projet, il était important de mettre en place une veille dédiée à la sécurité afin d'être au point sur les attaques les plus courantes : attaques XSS, attaques CSRF, attaques par injections SQL, attaques DDoS.

Pour cela, je me suis documenté sur des sites comme...

- La doc' Security de Symfony pour notre projet :
<https://symfony.com/doc/current/security.html>

- Blog de l'ANSSI : <https://cyber.gouv.fr/actualites>
- Les "Alertes de sécurité" du CERT-FR : <https://www.cert.ssi.gouv.fr/>
- Le blog jesuisundev.com : <https://www.jesuisundev.com/securite-web/>
- Le site Café Cyber : <https://www.cafe-cyber.oteria.fr/>
- Le site The Hacker News : <https://thehackernews.com/>

...mais aussi grâce à de nombreuses chaînes YouTube :

- Micode : <https://www.youtube.com/@Micode>
- Underscore_ : https://www.youtube.com/@Underscore_
- Pour1nf0 : <https://www.youtube.com/@Pour1nfo>
- Hafnium : <https://www.youtube.com/@HafniumSecuriteInformatique>

Enfin, le site Hacksplaining m'a bien aidé à comprendre les principes de base de ces attaques.

Conclusion

Pour conclure ce dossier de projet, je dois dire que la création du site O'Galaxy en PHP avec le framework Symfony a été une expérience enrichissante qui m'a conforté dans mon projet de reconversion professionnelle. Cette période d'un mois m'a permis de pouvoir exprimer tout ce que j'ai pu apprendre durant la phase socle (développeur web et web mobile PHP fullstack) et la phase de spécialisation (framework Symfony).

Il est vrai que lorsqu'on était dans les périodes de cours théoriques, il était parfois frustrant de devoir attendre la fin de journée pour avoir le challenge quotidien et pouvoir coder à partir des choses apprises en cours plus tôt dans la journée. Là, nous pouvions coder le matin et l'après-midi sans restriction et, personnellement, ça m'a procuré un bien fou !

Avec ce projet O'Galaxy, j'ai pu consolider mes compétences dans plusieurs langages (HTML, CSS, PHP) et sur plusieurs frameworks (Bootstrap, Symfony) en me confrontant aux problèmes que les développeurs web rencontrent au quotidien. Mais de manière tout aussi importante, j'ai amélioré certains soft skills comme le travail en équipe, la gestion de projet ou encore la communication verbale.

Merci du fond du cœur à tous les professeurs de l'école O'clock qui nous ont ouvert la porte vers le monde du développement web lors de cette formation : Yannick Kuhn, Jean-Christophe Guinez, Benjamin Bailly, Solène Goumy, Baptiste Delphin, Renaud Charpentier, Gregory Baltide, Pierrick Marie.

Un grand merci également à Guillaume, Loïc & Ludovic qui ont été de super tuteurs pendant l'Apothéose. Et bien évidemment, mille merci aussi à notre super tutrice Clara qui a pris soin de nous tout au long de cette formation !

Mes derniers mots seront pour l'équipe avec qui j'ai travaillé sur ce super projet ! Un ÉNORME merci à Guillaume, Jessica, Agnès et Élodie. Sans vous quatre, ce projet n'aurait pas du tout été le même, j'ai aimé notre symbiose de groupe, j'ai aimé travailler avec chacun(e) d'entre vous et si c'était à refaire, je remonterai dans la navette immédiatement !



Annexes

Annexe n°1 : Tableau des user stories

[O'GALAXY] USERS STORIES				
ID	THÈME	EN TANT QUE	JE SOUHAITE	AFIN DE...
1	Accueil	Visitor	Consulter la page d'accueil	Connaître les destinations phares
2	Expédition	Vistor	Consulter la liste de toutes les expéditions	Connaître l'ensemble des destinations
3	Expédition	Visitor	Consulter une expédition en particulier	Connaître les détails d'une expédition
4	Authentification	Visitor	Un formulaire d'inscription	Créer un compte client
5	Mentions légales	Visitor	Afficher les mentions légales	Connaître les informations légales du site
6	Carrières	Visitor	Afficher la page carrières	Connaître les offres d'emploi proposées par l'entreprise
7	Confidentialité	Visitor	Afficher la page de confidentialité	Connaître les règles de confidentialité du site
8	Actualités	Visitor	Afficher les actualités du site	Connaître les actualités de l'entreprise
9	Actualité	Visitor	Afficher la page d'un article des actualités	Voir le contenu d'un article
10	FAQ	Visitor	Afficher la FAQ	Connaître les questions fréquemment posées
11	À propos	Visitor	Afficher les informations "À propos"	Connaître l'équipe et l'entreprise
12	Contact	Visitor	Avoir accès au formulaire de contact	Poser des questions sur les expéditions / activités
13	Musique	User	Avoir accès à la page cachée du site	Pouvoir écouter l'hymne O'Galaxy
14	Authentification	User	Un formulaire de connexion	Me connecter à mon compte
15	Expédition	User	Réserver une ou plusieurs expéditions	Partir en voyage
16	Réservation	User	Avoir accès à un formulaire de réservation	Réserver une expédition
17	Favoris	User	Mettre des voyages en favoris	Toujours garder la liste de mes expéditions préférées
18	Avis	User	Noter une expédition effectuée	Montrer ma joie ou mon mécontentement par rapport à l'expédition effectuée

19	Avis	User	Mettre un commentaire sur mes expéditions effectuées	Faire partager mon expérience avec les autres
20	Administration	Admin	Avoir accès à mon compte et au back office	Gérer mon site
21	Administration	Admin	Créer / Modifier / Supprimer des expéditions et activités	Tenir à jour le site
22	Administration	Admin	Gérer les Users	Autoriser / restreindre l'accès à certaines parties du site
23	Administration	Admin	Visualiser les réservations des utilisateurs	Gérer les disponibilités et les finances
24	Administration	Admin	Consulter la page d'accueil du back office	Avoir un suivi du site et lire le mot du Président
25	Administration	Admin	Valider / Modifier / Supprimer les réservations	Tenir à jour les réservations
26	Administration	Admin	Valider / Modifier / Supprimer les utilisateurs	Tenir à jour la liste des utilisateurs

Annexe n°2 : Tableau des routes

Routes Front Office

URL	HTTP Method	Controller	Method	HTML Title	Comment
/	GET	HomeController	index	Home Page	Display the homepage of O'Galaxy
/inscription	GET	RegistrationController	register	Register Page	Display the form to create an account
/verification/email	GET	RegistrationController	verifyUserEmail	-	Display a flash message when the account is verify
/connexion	GET	SecurityController	login	Login Page	Display the form to login
/deconnexion	GET	SecurityController	logout	-	Logout the user
/expedition	GET	ExpeditionController	list	Expedition List	Display the expeditions list
/expedition/{id}	GET POST	ExpeditionController	show	Expedition details	Display the expedition details
/commentaire/ajouter/{id}	GET	ReviewController	add	Add review Page	Display the form to add review on an expedition
/recherche	GET	SearchController	search	Result Search Page	Display the search result for expeditions or news
/profil	GET POST	ProfileController	show	Profile Page	Display the profil page of the user
/favoris	GET	FavoriteController	list	Favorite Page	Display the favorite page
/favoris/ajouter/{id}	GET	FavoriteController	add	-	Add expeditions to Favorite page
/favoris/supprimer/{id}	GET	FavoriteController	remove	-	Remove expeditions to Favorite page
/favoris/vide	GET	FavoriteController	empty	-	Empty list of favorites
/actualites	GET	MiscController	news	News Page	Display the list of news
/{id}/actualites	GET	MiscController	show	News details	Display the news details
/carrieres	GET	MiscController	careers	Careers Page	Display the Careers page
/confidentialite	GET	MiscController	confidentiality	Confidentiality Page	Display the confidentiality page
/contact	GET POST	MiscController	contact	Contact Page	Display the form to contact O'Galaxy
/faq	GET	MiscController	faq	FAQ Page	Display the faq page
/mentions-legales	GET	MiscController	legalMentions	Legal Notice Page	Display the legal notice page
/notre-histoire	GET	MiscController	index	About Us Page	Display the about us page
/notre-album	GET	MiscController	album	Album Page	Display the album page

Routes Back Office

URL	HTTP Method	Controller	Method	HTML Title	Comment
/admin	GET	MainController	homeAdmin	Home Page	Display the homepage of bacck office
/admin/utilisateur	GET	UserController	list	User List	Display the users list
/admin/utilisateur/creer	GET POST	UserController	create	Add User	Display the form for add an user
/admin/utilisateur/{id}/modifier	GET POST	UserController	update	Update User	Display the form for update an user
/admin/utilisateur/{id}/supprimer	POST	UserController	delete	Delete User	Button to delete an user
/admin/utilisateur/{id}	GET	UserController	show	User details	Display the user details
/admin/utilisateur/{id}/reservation	GET	UserController	userBooking	User booking	Display the user booking
/admin/commentaire	GET	ReviewController	list	Reviews List	Display the reviews list
/admin/commentaire/{id}	GET	ReviewController	show	Review details	Display the review details
/admin/commentaire/{id}/supprimer	POST	ReviewController	delete	Delete review	Button to delete a review
/admin/expedition	GET	ExpeditionController	list	Expeditions List	Display the expeditions list
/admin/expedition/creer	GET POST	ExpeditionController	create	Add Expedition	Display the form to an expedition
/admin/expedition/{id}/modifier	GET POST	ExpeditionController	update	Update Expedition	Display the form to update an expedition
/admin/expedition/{id}/supprimer	POST	ExpeditionController	delete	Delete Expedition	Button to delete an expedition
/admin/expedition/{id}	GET	ExpeditionController	show	Expedition details	Display the expedition details
/admin/contact	GET	ContactController	list	Contact Message List	Display the message list
/admin/contact/{id}	GET	ContactController	show	Contact Message Details	Display the message details
/admin/contact/{id}/supprimer	POST	ContactController	delete	Delete contact message	Button to delete a message
/admin/reservation	GET	BookingController	list	Booking List	Display the booking list
/admin/reservation/creer	GET	BookingController	create	Add Booking	Display the form to add booking
/admin/reservation/{id}/modifier	GET POST	BookingController	update	Update Booking	Display the form to update booking
/admin/reservation/{id}/supprimer	POST	BookingController	delete	Delete Booking	Button to delete a booking
/admin/reservation/{id}	GET	BookingController	show	Booking details	Display the booking details
/admin/article	GET	ArticleController	list	Article List	Display the news list
/admin/article/creer	GET	ArticleController	create	Add News	Display the form to add a news
/admin/article/{id}/modifier	GET POST	ArticleController	update	Update news	Display the form to update news
/admin/article/{id}/supprimer	POST	ArticleController	delete	Delete news	Button to delete a news
/admin/article/{id}	GET	ArticleController	show	News details	Display the news details

Annexe n°3 : Dictionnaire de données

Table : `USER`

Colonne	Type	Description	Clé primaire	Auto-incrément	Non null
user_id	INT	Identifiant unique de l'utilisateur	Oui	Oui	Oui
pseudo	VARCHAR(255)	Pseudonyme de l'utilisateur	Non	Non	Oui
firstname	VARCHAR(255)	Prénom de l'utilisateur	Non	Non	Oui
lastname	VARCHAR(255)	Nom de famille de l'utilisateur	Non	Non	Oui
email	VARCHAR(255)	Adresse email de l'utilisateur	Non	Non	Oui
password	VARCHAR(255)	Mot de passe de l'utilisateur	Non	Non	Oui
picture	VARCHAR(255)	URL de la photo de profil de l'utilisateur	Non	Non	Non
roles	VARCHAR(255)	Rôles de l'utilisateur (ex. : admin, utilisateur)	Non	Non	Oui
is_verified	BOOLEAN	Indique si l'utilisateur est vérifié	Non	Non	Oui
created_at	TIMESTAMP	Date de création de l'utilisateur	Non	Non	Non
updated_at	TIMESTAMP	Date de mise à jour de l'utilisateur	Non	Non	Non

Table : `BOOKING`

Colonne	Type	Description	Clé primaire	Auto-incrément	Non null
booking_id	INT	Identifiant unique de la réservation	Oui	Oui	Oui
booking_status	INT	Statut de la réservation (ex. : en attente, confirmé, annulé)	Non	Non	Oui
created_at	TIMESTAMP	Date de création de la réservation	Non	Non	Non
updated_at	TIMESTAMP	Date de mise à jour de la réservation	Non	Non	Non
code_user	INT	Identifiant de l'utilisateur ayant fait la réservation	Non	Non	Non
code_expedition	INT	Identifiant de l'expédition réservée	Non	Non	Non

Table : `EXPEDITION`

Colonne	Type	Description	Clé primaire	Auto-incrémentation	Non null
expedition_id	INT	Identifiant unique de l'expédition	Oui	Oui	Oui
title	VARCHAR(255)	Titre de l'expédition	Non	Non	Oui
destination	VARCHAR(255)	Destination de l'expédition	Non	Non	Oui
departure	TIMESTAMP	Date et heure de départ de l'expédition	Non	Non	Oui
duration	SMALLINT	Durée de l'expédition en jours	Non	Non	Oui
passenger	INT	Nombre de passagers prévus	Non	Non	Oui
description	TEXT(65535)	Description détaillée de l'expédition	Non	Non	Oui
picture	VARCHAR(255)	URL de l'image de l'expédition	Non	Non	Oui
price	DECIMAL	Prix de l'expédition	Non	Non	Oui
created_at	TIMESTAMP	Date de création de l'expédition	Non	Non	Non
updated_at	TIMESTAMP	Date de mise à jour de l'expédition	Non	Non	Non

Table : `REVIEW`

Colonne	Type	Description	Clé primaire	Auto-incrémentation	Non null
review_id	INT	Identifiant unique de l'avis	Oui	Oui	Oui
rating	INT	Note donnée par l'utilisateur (ex. : 1 à 5)	Non	Non	Oui
comment	TEXT(65535)	Commentaire de l'utilisateur	Non	Non	Oui
picture	VARCHAR(255)	URL de l'image associée à l'avis	Non	Non	Non
created_at	TIMESTAMP	Date de création de l'avis	Non	Non	Non
updated_at	TIMESTAMP	Date de mise à jour de l'avis	Non	Non	Non
code_user	INT	Identifiant de l'utilisateur ayant laissé l'avis	Non	Non	Non
code_expedition	INT	Identifiant de l'expédition évaluée	Non	Non	Non

Table : `LIKE`

Colonne	Type	Description	Clé primaire	Auto-incrémentation	Non null
like_id	INT	Identifiant unique du "like"	Oui	Oui	Oui
code_user	INT	Identifiant de l'utilisateur ayant donné le "like"	Non	Non	Non
code_expedition	INT	Identifiant de l'expédition aimée	Non	Non	Non

Table : `ARTICLE`

Colonne	Type	Description	Clé primaire	Auto-incrémentation	Non null
article_id	INT	Identifiant unique de l'article	Oui	Oui	Oui
title	VARCHAR(255)	Titre de l'article	Non	Non	Oui
picture	VARCHAR(255)	URL de l'image associée à l'article	Non	Non	Non
text	TEXT(65535)	Contenu de l'article	Non	Non	Oui
created_at	TIMESTAMP	Date de création de l'article	Non	Non	Non
updated_at	TIMESTAMP	Date de mise à jour de l'article	Non	Non	Non

Table : `CONTACT`

Colonne	Type	Description	Clé primaire	Auto-incrémentation	Non null
contact_id	INT	Identifiant unique du contact	Oui	Oui	Oui
firstname	VARCHAR(255)	Prénom du contact	Non	Non	Oui
lastname	VARCHAR(255)	Nom de famille du contact	Non	Non	Oui
email	VARCHAR(255)	Adresse email du contact	Non	Non	Oui
subject	VARCHAR(255)	Sujet du message	Non	Non	Oui
message	TEXT(65535)	Contenu du message	Non	Non	Oui
created_at	TIMESTAMP	Date de création du message	Non	Non	Non
updated_at	TIMESTAMP	Date de mise à jour du message	Non	Non	Non

Annexe n°4 : Script de création de la base de données

```
CREATE OR REPLACE TABLE `USER` (
    `user_id` INT NOT NULL AUTO_INCREMENT UNIQUE,
    `pseudo` VARCHAR(255) NOT NULL,
    `firstname` VARCHAR(255) NOT NULL,
    `lastname` VARCHAR(255) NOT NULL,
    `email` VARCHAR(255) NOT NULL,
    `password` VARCHAR(255) NOT NULL,
    `picture` VARCHAR(255),
    `roles` VARCHAR(255) NOT NULL,
    `is_verified` BOOLEAN NOT NULL,
    `created_at` TIMESTAMP,
    `updated_at` TIMESTAMP,
    PRIMARY KEY(`user_id`)
);
CREATE OR REPLACE TABLE `EXPEDITION` (
    `expedition_id` INT NOT NULL AUTO_INCREMENT UNIQUE,
    `title` VARCHAR(255) NOT NULL,
    `destination` VARCHAR(255) NOT NULL,
    `departure` TIMESTAMP NOT NULL,
    `duration` SMALLINT NOT NULL,
    `passenger` INT NOT NULL,
    `description` TEXT(65535) NOT NULL,
    `picture` VARCHAR(255) NOT NULL,
    `price` DECIMAL NOT NULL,
    `created_at` TIMESTAMP,
    `updated_at` TIMESTAMP,
    PRIMARY KEY(`expedition_id`)
);
CREATE OR REPLACE TABLE `BOOKING` (
    `booking_id` INT NOT NULL AUTO_INCREMENT UNIQUE,
    `booking_status` INT NOT NULL,
    `created_at` TIMESTAMP,
    `updated_at` TIMESTAMP,
    `code_user` INT,
    `code_expedition` INT,
    PRIMARY KEY(`booking_id`)
);
CREATE OR REPLACE TABLE `REVIEW` (
    `review_id` INT NOT NULL AUTO_INCREMENT UNIQUE,
    `rating` INT NOT NULL,
    `comment` TEXT(65535) NOT NULL,
    `picture` VARCHAR(255),
    `created_at` TIMESTAMP,
    `updated_at` TIMESTAMP,
    `code_user` INT,
    `code_expedition` INT,
    PRIMARY KEY(`review_id`)
);
```

```

CREATE OR REPLACE TABLE `LIKE` (
    `like_id` INT NOT NULL AUTO_INCREMENT UNIQUE,
    `code_user` INT,
    `code_expedition` INT,
    PRIMARY KEY(`like_id`)
);
CREATE OR REPLACE TABLE `ARTICLE` (
    `article_id` INT NOT NULL AUTO_INCREMENT UNIQUE,
    `title` VARCHAR(255) NOT NULL,
    `picture` VARCHAR(255),
    `text` TEXT(65535) NOT NULL,
    `created_at` TIMESTAMP,
    `updated_at` TIMESTAMP,
    PRIMARY KEY(`article_id`)
);
CREATE OR REPLACE TABLE `CONTACT` (
    `contact_id` INT NOT NULL AUTO_INCREMENT UNIQUE,
    `firstname` VARCHAR(255) NOT NULL,
    `lastname` VARCHAR(255) NOT NULL,
    `email` VARCHAR(255) NOT NULL,
    `subject` VARCHAR(255) NOT NULL,
    `message` TEXT(65535) NOT NULL,
    `created_at` TIMESTAMP,
    `updated_at` TIMESTAMP,
    PRIMARY KEY(`contact_id`)
);
ALTER TABLE `USER`
ADD FOREIGN KEY(`user_id`) REFERENCES `BOOKING`(`code_user`)
ON UPDATE NO ACTION ON DELETE NO ACTION;
ALTER TABLE `EXPEDITION`
ADD FOREIGN KEY(`expedition_id`) REFERENCES `BOOKING`(`code_expedition`)
ON UPDATE NO ACTION ON DELETE NO ACTION;
ALTER TABLE `REVIEW`
ADD FOREIGN KEY(`code_user`) REFERENCES `USER`(`user_id`)
ON UPDATE NO ACTION ON DELETE NO ACTION;
ALTER TABLE `REVIEW`
ADD FOREIGN KEY(`code_expedition`) REFERENCES `EXPEDITION`(`expedition_id`)
ON UPDATE NO ACTION ON DELETE NO ACTION;
ALTER TABLE `LIKE`
ADD FOREIGN KEY(`like_id`) REFERENCES `USER`(`user_id`)
ON UPDATE NO ACTION ON DELETE NO ACTION;
ALTER TABLE `LIKE`
ADD FOREIGN KEY(`code_user`) REFERENCES `EXPEDITION`(`expedition_id`)
ON UPDATE NO ACTION ON DELETE NO ACTION;

```

Annexe n°5 : Templates list.html.twig du dossier templates/back/user

```
● ● ●

{% extends 'back/backoffice.html.twig' %}

{% block title %}
    Backoffice - Utilisateurs
{% endblock %}

{% block body %}
<!-- Display the list of users-->
<section id="list-users" aria-labelledby="list-user-complete">
    <div class="container my-4">
        <h3 class="display-4 ms-3 page__title"> Utilisateurs </h3>
        <a class="btn add__items ms-3" href="{{ path('app_admin_user_create') }}" role="button"
            aria-label="Ajouter un utilisateur à la liste">
            Ajouter un utilisateur
        </a>
        {% for message in app.flashes('success') %}
            <div class="alert alert-success col-md-12 ms-3" role="alert">
                {{ message }}
            </div>
        {% endfor %}

        <div class="row mt-3">
            <div class="col-12 ms-3">
                <div class="card table__bg">
                    <div class="card-header table__bg">
                        <caption>Liste des utilisateurs</caption>
                    </div>
                    <div class="card-body">
                        <table role="presentation" class="table table-hover">
                            <thead class="text-center table__header">
                                <tr>
                                    <th scope="col">
                                        #
                                    </th>
                                    <th scope="col">
                                        Pseudo
                                    </th>
                                    <th scope="col">
                                        Prénom
                                    </th>
                                    <th scope="col">
                                        Nom
                                    </th>
                                    <th scope="col">
                                        Mail
                                    </th>
                                    <th scope="col">
                                        Voir/Modifier/Supprimer
                                    </th>
                                </tr>
                            </thead>
                            <tbody>
                                {% for currentUser in userList %}
                                    <tr class="text-center">
                                        <th scope="row">
                                            {{ currentUser.id }}
                                        </th>
                                        <td>
                                            {{ currentUser.pseudo }}
                                        </td>
                                        <td>
                                            {{ currentUser.firstname }}
                                        </td>
                                        <td>
                                            {{ currentUser.lastname }}
                                        </td>
                                        <td>
                                            {{ currentUser.email }}
                                        </td>
                                        <td>
                                            <a href="{{ path('app_admin_user_show', {id: currentUser.id}) }}"
                                                class="btn btn-sm show__btn" role="button"
                                                aria-label="Bouton pour afficher l'utilisateur choisi">
                                                <i class="bi bi-eye" aria-hidden="true"></i>
                                            </a>
                                            <a href="{{ path('app_admin_user_update', {id: currentUser.id}) }}"
                                                class="btn btn-sm update__btn" role="button"
                                                aria-label="Bouton pour modifier l'utilisateur choisi">
                                                <i class="bi bi-pencil-square" aria-hidden="true"></i>
                                            </a>
                                            <div class="btn-group">
                                                <form method="post" action="{{ path('app_admin_user_delete', {id: currentUser.id}) }}"
                                                    onsubmit="return confirm('Voulez-vous supprimer ce compte utilisateur ?');">
                                                    <input type="hidden" name="token" value="{{ csrf_token('delete') ~ currentUser.id }}"/>
                                                    <button type="submit" class="btn btn-sm delete__btn" aria-label="Bouton pour supprimer l'utilisateur choisi">
                                                        <i class="bi bi-trash" aria-hidden="true"></i>
                                                    </button>
                                                </form>
                                            </div>
                                        </td>
                                    </tr>
                                {% endfor %}
                            </tbody>
                        </table>
                    </div>
                </div>
            </div>
        </section>
    {% endblock %}
```

Annexe n°6 : Entité User du dossier src/Entity

```
● ● ●

<?php

namespace App\Entity;

use DateTimeImmutable;
use Doctrine\ORM\Mapping as ORM;
use App\Repository\UserRepository;
use Doctrine\Common\Collections\Collection;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;
use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface;
use Symfony\Component\Validator\Constraints as Assert;

#[ORM\Entity(repositoryClass: UserRepository::class)]
#[ORM\UniqueConstraint(name: 'UNIQ_IDENTIFIER_EMAIL', fields: ['email'])]
#[UniqueEntity(fields: ['email'], message: 'Il existe déjà un compte relié à cette adresse')]
#[ORM\HasLifecycleCallbacks]
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[Assert\NotBlank]
    #[Assert\Email([
        'message' => 'Votre adresse mail "{{ value }}" n\'est pas valide.',
    ])]
    #[ORM\Column(length: 180)]
    private ?string $email = null;

    /**
     * @var list<string> The user roles
     */
    #[ORM\Column]
    private array $roles = [];

    /**
     * @var string The hashed password
     */
    #[ORM\Column]
    private ?string $password = null;

    #[Assert\Length(
        min: 3,
        max: 30,
        minMessage: "Votre pseudo doit contenir au moins {{ limit }} caractères.",
        maxMessage: "Votre pseudo doit contenir au maximum {{ limit }} caractères."
    )]
    #[ORM\Column(length: 30)]
    private ?string $pseudo = null;

    #[Assert\NotBlank]
    #[Assert\Length(
        min: 2,
        max: 50,
        minMessage: 'Votre prénom doit contenir au moins {{ limit }} caractères',
        maxMessage: 'Votre prénom doit contenir au plus {{ limit }} caractères',
    )]
    ...

}
```