

DOSSIER DE PROJET

En vue de l'obtention du Titre Professionnel
Développeur Web et Web Mobile

O'Id the door

Réalisé par
HERBET LE FAUCHEUR Tony

Centre de formation
O'clock

Promotion
Excalibur

Référent de formation
CERQUEIRA Fabio

SOMMAIRE

1 - Introduction :	3
1a - Apothéose :	3
1b - O'Id the door :	3
Le principe :	3
Le déroulement d'une partie :	4
Création d'une aventure :	4
2 - Compétences du référentiel couvertes par le projet :	4
2a - Activité type 1 :	4
CP1 - Maquetter une application :	4
CP2 - Réaliser une interface utilisateur web statique et adaptable :	4
CP3 - Développer une interface utilisateur web dynamique :	5
2b - Activité type 2 :	5
CP5 - Créer une base de données :	5
CP6 - Développer les composants d'accès aux données :	5
CP7 - Développer la partie back-end d'une application web ou web mobile :	6
3 - Cahier des charges :	6
3a - MVP :	6
Fonctionnalités du MVP :	6
Fonctionnalités jugées secondaires :	7
3b - User Stories :	8
Les User Stories du MVP :	8
Les User Stories des fonctionnalités secondaires :	9
3c - Maquettes :	9
La page d'accueil :	11
La page d'inscription version desktop :	12
La page d'inscription version mobile :	12
La page des aventures disponibles version desktop :	13
La page des aventures disponibles version mobile :	14
3d - Workflow :	15
3e - MCD :	16
3f - Dictionnaire des données :	17
Story	17
Chapter	18
Party	19
User	20
3g - Endpoints :	21
Liste des Endpoints :	21
Stories	21
Chapters	23

Parties	24
Users	24
Liste des contrôleurs pour les endpoints :	25
Stories	25
Chapters	26
Parties	26
Users	27
Login	27
4 - Spécifications techniques :	27
4a - Versionning :	27
4b - Front :	28
4C - Back :	29
5 - Organisation :	30
5a - L'équipe :	30
5b - Outils :	31
5c - Méthodologie :	32
6 - Réalisations :	34
6a - Sprint 0 :	34
6b - Sprint 1 :	42
6c - Sprint 2 :	51
6d - Sprint 3 :	62
6e - Tests :	67
7 - Recherches :	71
8 - Conclusion :	76
8a - O'ld the door :	76
8b - Remerciements :	77

1 - Introduction :

1a - Apothéose :

Dans le cadre de ma formation chez O'clock, nous avons dû proposer des idées de projet dans le but d'en choisir un sur lequel nous allions travailler pendant un mois comme projet de fin de formation, appelé Apothéose.

1b - O'Id the door :

Le projet auquel j'ai participé s'appelle "O'Id the door", il y a une tradition non officielle et tacite lors d'une formation chez O'clock que les élèves nomment leurs projets en commençant par un "o".

O'Id the door est une idée de Damien TOSCANO qui voulait créer un jeu s'inspirant des escape game et des jeux d'aventures textuelles des années 1980~1990 tel que Zork. Le tout en se basant sur un design rétro qui laisse libre court à l'imagination du joueur et rendra nostalgiques les vétérans du genre.

Le principe :

Un utilisateur inscrit peut choisir une aventure disponible sur le site et y jouer ou créer ses propres aventures pour les autres utilisateurs.

Une aventure disponible contient au minimum un chapitre.

Lors de la partie, l'utilisateur devra déverrouiller tous les chapitres de l'aventure pour la finir et ainsi enregistrer son temps de complétion de celle-ci.

Pour déverrouiller un chapitre et passer au suivant, il devra trouver dans le texte du chapitre un mot clé et un mot serrure.

Le texte de chaque chapitre défile de façon dactylographique pour un effet rétro.

Les aventures ont une moyenne de temps de complétion se basant sur la première complétion de tous les utilisateurs les ayant terminées afin de donner une indication sur la difficulté ainsi que de la durée à prévoir.

Le déroulement d'une partie :

Pour chaque chapitre de l'aventure sélectionnée, l'utilisateur devra remplir un champ pour le mot clé et le mot serrure, ces mots se trouvent dans le texte du chapitre.
Pour chaque tentative ratée, un malus sera appliqué à son temps de complétion.
Un indice sera disponible une fois par chapitre, en contrepartie d'un malus sur le temps de complétion.

Création d'une aventure :

L'utilisateur devra remplir un formulaire afin de nommer et décrire son aventure.
Une fois ce premier formulaire validé, il pourra éditer cette aventure et lui créer son premier chapitre en remplaçant un autre formulaire.
Une aventure contenant au moins un chapitre peut être publiée pour la rendre jouable par les utilisateurs.
L'utilisateur peut ensuite ajouter autant de chapitres qu'il le souhaite et définir l'ordre de ces chapitres.

2 - Compétences du référentiel couvertes par le projet :

2a - Activité type 1 :

CP1 - Maquetter une application :

Nous avons rédigé un cahier des charges lors de la première semaine, il contient des wireframes ou maquettes fonctionnelles, pour les versions mobile et desktop de l'application.
Nous avons également réalisé un schéma du workflow afin de prévoir les cheminement possibles lors de la navigation.
Les différentes fonctionnalités ont été prévues en rédigeant des User stories.

CP2 - Réaliser une interface utilisateur web statique et adaptable :

Le site étant un jeu interactif, il est dans sa quasi totalité dynamique. Cependant pour la page de présentation de l'équipe, nous avons décidé de faire un affichage statique avec nos noms "dessinés" en ASCII Art.

Le site est entièrement adaptable aux différentes tailles d'écran lors de l'utilisation, en effet nous avons utilisé plusieurs media queries CSS afin de prévoir différents breakpoints et ainsi avoir un site responsive.

CP3 - Développer une interface utilisateur web dynamique :

Comme annoncé précédemment, le site est dans sa quasi totalité dynamique.

Pour se faire, nous avons conçu côté back une API (Application Programming Interface). Le front fait appel à cette API afin de recevoir les données nécessaires au fonctionnement du site et afficher les informations dynamiquement.

Ceci est particulièrement vrai et important pour les pages de création d'une aventure ainsi que pour la page de jeu car toutes les aventures ne sont pas les mêmes.

C'est également le cas pour la page de profil d'un utilisateur qui dispose de ses propres informations y compris les aventures dont il est l'auteur.

2b - Activité type 2 :

CP5 - Créer une base de données :

Dans le cahier des charges, nous avons préparé la création de la base de données en réalisant un MCD (Modèle conceptuel des données) ainsi qu'un dictionnaire de données. Nous avons utilisé MySQL comme système de gestion de base de données relationnelles et Adminer comme interface graphique.

Nous avons créé les tables de notre base de données et gérer leurs relations via l'ORM (Object-Relational mapping) Doctrine pour Symfony.

CP6 - Développer les composants d'accès aux données :

L'interaction avec la base de données a été faite en utilisant l'ORM Doctrine dans les fichiers Repository en utilisant des fonctions qui vont générer du DQL (Doctrine Query Language), le langage de requête de Doctrine.

L'accès aux données côté front se fait via des requêtes à notre API utilisant l'architecture REST (Representational state transfer).

Des custom queries ont été réalisées afin de réduire la charge sur notre base de données.

Cela nous a demandé de créer un CRUD (Create, Read, Update, Delete) complet pour chaque table (nommé Entity sur Symfony) de la base de données.

Les utilisateurs auront donc accès à un CRUD complet pour la gestion de leur compte ainsi que des aventures dont ils sont l'auteur (certaines fonctionnalités ne sont pas encore mises en place entièrement).

CP7 - Développer la partie back-end d'une application web ou web mobile :

Le développement de la partie back-end a été fait en utilisant le framework Symfony en suivant l'architecture MVC (Model View Controller).

Via l'utilisation du bundle Security de Symfony nous avons mis en place un système de Voters qui s'assure qu'un utilisateur ne peut avoir accès et ne peut gérer que son compte et ses aventures.

Le système d'authentification utilise des tokens afin que l'utilisateur n'ai pas besoin de s'identifier à chaque requête faite à l'API, ce token est changé dans la base de donnée à chaque connexion et déconnexion de l'utilisateur dans le but de sécuriser son compte.

3 - Cahier des charges :

Le cahier des charges a été écrit par l'équipe entière en suivant les recommandations et la vision du Product Owner Damien TOSCANO.

Plusieurs modifications du cahier des charges ont été apportées pendant le développement au fur et à mesure que le projet devenait plus concret et que des problématiques faisaient surface.

Je reviendrais sur les raisons de ces modifications dans la section Organisation lorsque j'expliquerai notre méthodologie de travail.

3a - MVP :

Le MVP (Minimum viable product) est une version simple et épurée d'un projet qui contient seulement les fonctionnalités jugées nécessaires au fonctionnement et à l'utilisation concrète du projet afin d'avoir un produit viable dès que possible.

Les autres fonctionnalités seront ajoutées par la suite lors du développement.

Fonctionnalités du MVP :

- Une page de jeu sur laquelle les utilisateurs vont pouvoir tester leurs capacités de compréhension des aventures, elle comprendra :
 - Une indication du temps de jeu.
 - La partie textuelle de l'histoire.
 - La partie destinée à tester les combinaisons.

- Un bouton d'indice qui révèle une partie de la combinaison à trouver contre une pénalité de temps.
- La possibilité de créer des aventures.
- La possibilité d'éditer une aventure dont on est l'auteur.
- La possibilité de supprimer une aventure dont on est l'auteur.
- La possibilité de publier et dépublier une aventure dont on est l'auteur.
- Une page d'inscription pour les nouveaux utilisateurs.
- Une page de connexion.
- Une page de profil où l'on pourra retrouver :
 - Les informations personnelles de l'utilisateur.
 - Les aventures jouées par l'utilisateur.
 - Les aventures créées par l'utilisateur.
- Une page listant toutes les aventures jouables sur le site.
- Une page descriptive pour chaque aventure.
- Un système de meilleur temps et de temps moyen pour chaque aventure en guise de niveau de difficulté.
- Une page d'accueil qui explique le concept du site et présente des liens d'inscription et le catalogue des aventures.
- Une page de présentation de l'équipe de développement.
- Une page d'erreur 404 personnalisée.

Fonctionnalités jugées secondaires :

- Un système de catégorie pour les aventures afin d'affiner leur description.
- Un système de recherche par catégorie.
- Un système de favoris sur les aventures.
- Un système de notification lorsqu'un nouveau chapitre d'une aventure favorite est disponible.
- Un système de commentaires sur les aventures, qui comprendra une partie où l'on pourra rédiger un commentaire et une partie permettant de lire les commentaires laissés sur les aventures.
- Une page CGU (Conditions générales d'utilisation).
- Une page de contact.
- Une pagination pour la liste des aventures.
- La possibilité de modifier ses informations sur sa page de profil.
- Une partie administration du site accessible à ceux ayant le rôle adéquat pour pouvoir :
 - La dépublication d'une aventure.
 - La suppression d'une aventure.
 - Contacter un auteur.
 - Consulter la liste de tous les commentaires.
 - Supprimer un commentaire.

3b - User Stories :

User Stories ou récit utilisateur est une liste de phrases décrivant les besoins et/ou attente d'un utilisateur.

Elles permettent de déterminer les fonctionnalités à développer de façon simple et précise.

Nous avons prévus trois types d'utilisateurs :

- Anonyme (un utilisateur qui ne serait pas encore inscrit)
- Utilisateur (un utilisateur inscrit et connecté)
- Administrateur

Les User Stories d'un Administrateur incluent celles d'un Utilisateur et les User Stories d'un Utilisateur incluent celles d'un Anonyme.

Les User Stories du MVP :

- En tant qu'**anonyme**, je peux visiter la page d'accueil, pour en apprendre plus sur le site.
- En tant qu'**anonyme**, je peux regarder la liste des aventures publiées, pour faire mon choix sur mes prochaines parties.
- En tant qu'**anonyme**, je peux aller sur la page d'inscription, pour m'enregistrer dans le système d'authentification du site.
- En tant qu'**anonyme**, je peux aller sur la page de connexion, pour accéder aux fonctions réservées aux utilisateurs.
- En tant qu'**anonyme**, je peux consulter la page "Notre équipe", pour en apprendre davantage sur les créateurs du site.
- En tant qu'**utilisateur**, je peux consulter la page de détail d'une aventure publiée, pour voir toutes ses informations avant de me lancer dans le jeu.
- En tant qu'**utilisateur**, je peux jouer à une aventure publiée, pour tester mes compétences en résolution d'énigme.
- En tant qu'**utilisateur**, en jeu, je peux faire appel à un indice contre une pénalité de temps, pour me débloquer si je suis coincé sur un chapitre de l'aventure.
- En tant qu'**utilisateur**, je peux aller sur ma page de profil, pour voir mes informations, la liste des aventures que j'ai créé et la liste des aventures auxquelles j'ai joué.
- En tant qu'**utilisateur**, je peux écrire une aventure, pour à terme la publier.
- En tant qu'**utilisateur**, je peux modifier une aventure que j'ai écrite, pour la peaufiner.
- En tant qu'**utilisateur**, je peux publier une aventure que j'ai écrite, pour la mettre à disposition des joueurs.
- En tant qu'**utilisateur**, je peux dépublier une aventure que j'ai écrite, pour enlever la possibilité qu'elle soit jouable.
- En tant qu'**utilisateur**, je peux supprimer une aventure que j'ai écrite, pour faire le ménage dans mes aventures.
- En tant qu'**utilisateur**, je veux être informé si je suis sur une page inexistante grâce à une page 404, pour m'aiguiller dans ma navigation.

Les User Stories des fonctionnalités secondaires :

- En tant qu'**anonyme**, je peux consulter la page "CGU", pour lire les conditions générales d'utilisations du site.
- En tant qu'**anonyme**, je peux consulter la page "Contact", pour envoyer un message aux créateurs du site.
- En tant qu'**utilisateur** je peux filtrer les aventures publiées par difficulté, pour trouver les aventures correspondant le mieux à mon profil.
- En tant qu'**utilisateur** je peux filtrer les aventures publiées par catégorie, pour trouver les aventures correspondant le mieux à mon profil.
- En tant qu'**utilisateur**, je peux lire les commentaires sur une aventure, pour me faire une idée de l'expérience que les autres utilisateurs ont eues.
- En tant qu'**utilisateur**, je peux laisser un commentaire sur une aventure publiée, pour donner mon avis sur l'expérience que j'ai eu.
- En tant qu'**utilisateur**, je peux modifier mes informations de profil, pour avoir des informations à jour.
- En tant qu'**utilisateur**, je peux voir mes parties passées, pour me rappeler des parties auxquelles j'ai déjà joué et les scores que j'ai réalisés.
- En tant qu'**utilisateur**, je peux mettre une aventure en favori, pour l'ajouter dans ma liste des favoris et y revenir plus tard.
- En tant qu'**utilisateur**, je peux consulter la liste de mes aventures favorites, pour choisir facilement une partie qui m'intéresse.
- En tant qu'**utilisateur**, je veux recevoir une notification lorsqu'un nouveau chapitre est disponible sur une de mes aventures favorites.
- En tant qu'**administrateur**, je peux dépublier et/ou supprimer une aventure, pour éviter d'avoir du contenu ne correspondant pas aux valeurs du site.
- En tant qu'**administrateur**, je peux contacter un auteur, pour échanger des informations concernant une aventure qu'il a écrite.
- En tant qu'**administrateur**, je peux consulter la liste de tous les commentaires, pour vérifier qu'ils ont des contenus appropriés.
- En tant qu'**administrateur**, je peux supprimer un commentaire, si jamais il n'est pas compatible avec la charte du site.

3c - Maquettes :

Une fois notre MVP et User Stories définis, nous avons pu faire des maquettes fonctionnelles ou wireframes.

Celles-ci servent lors de la conception à définir l'interface utilisateur et les éléments qu'elle contient, leurs emplacements approximatifs ainsi que toutes informations nécessaires à la compréhension sous forme de légendes.

Le but étant de mettre "sur papier" ce qui a été fait précédemment afin que toute l'équipe ait la même vision du projet sans ambiguïté et ainsi faciliter le développement.

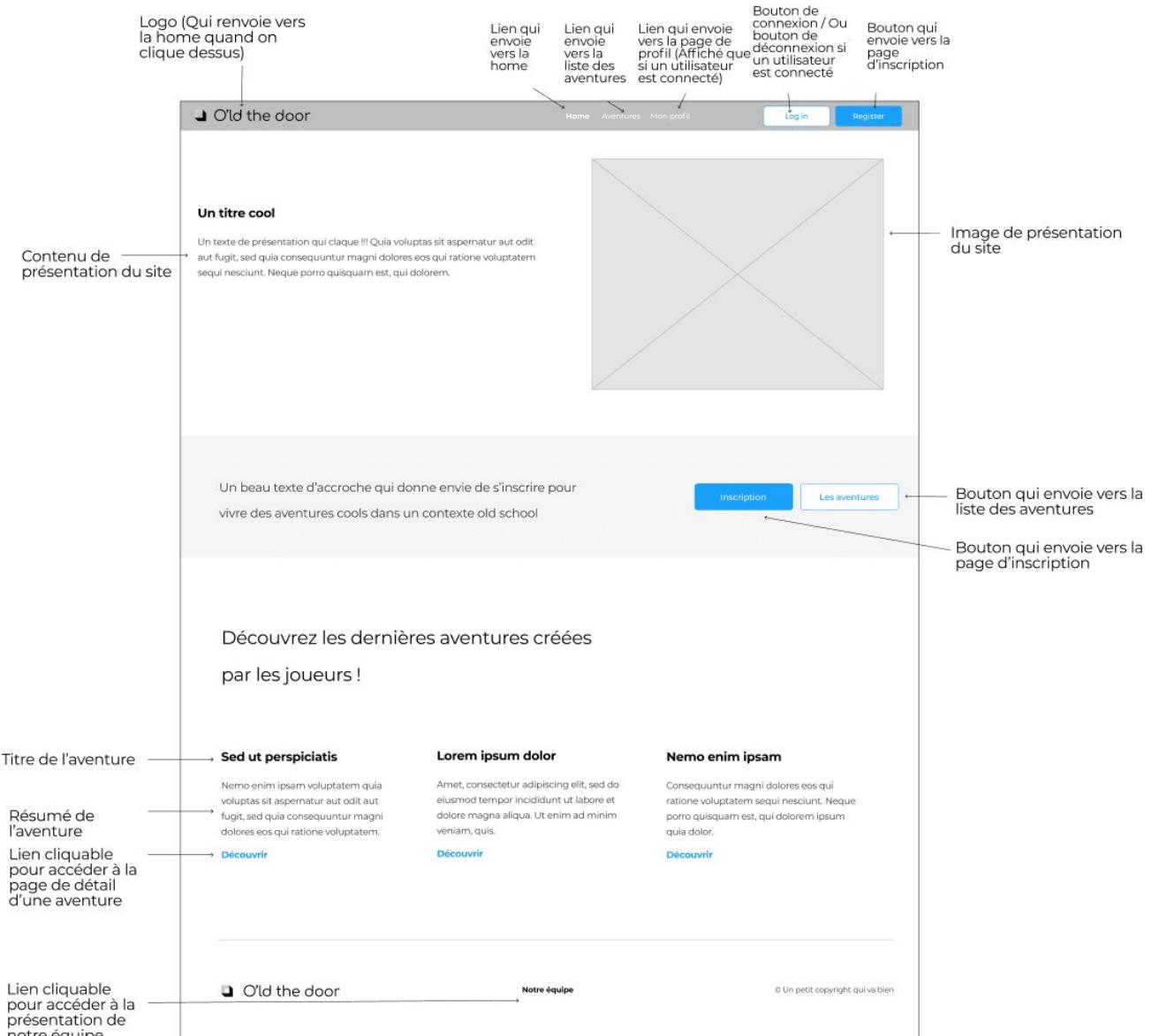
La deuxième étape aurait été de faire des maquettes graphiques pour définir les détails de l'application comme les couleurs, les polices de caractères et toutes autres informations en rapport avec le design qui définissent l'identité visuelle du site. Cependant, nous n'étions à ce moment-là pas sûr du chemin que nous voulions prendre.

Pour la réalisation des wireframes, nous avons utilisé l'outil en ligne Figma qui contient tout le nécessaire à la création de wireframes et est utilisable à plusieurs en simultané.

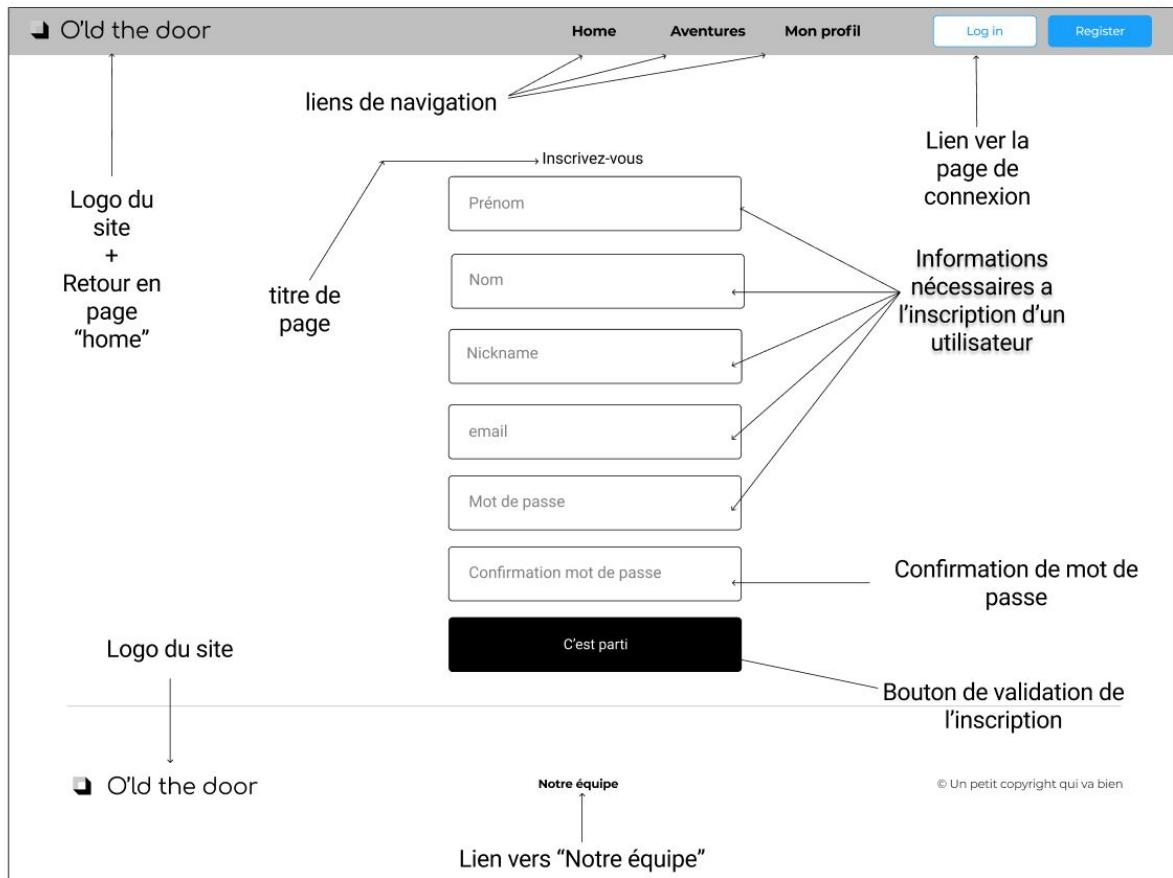
Nous avons tous participé à la réalisation, chacun s'est occupé de quelques pages et nous avons validé en équipe chaque wireframes.

Voici quelques exemples des wireframes réalisées :

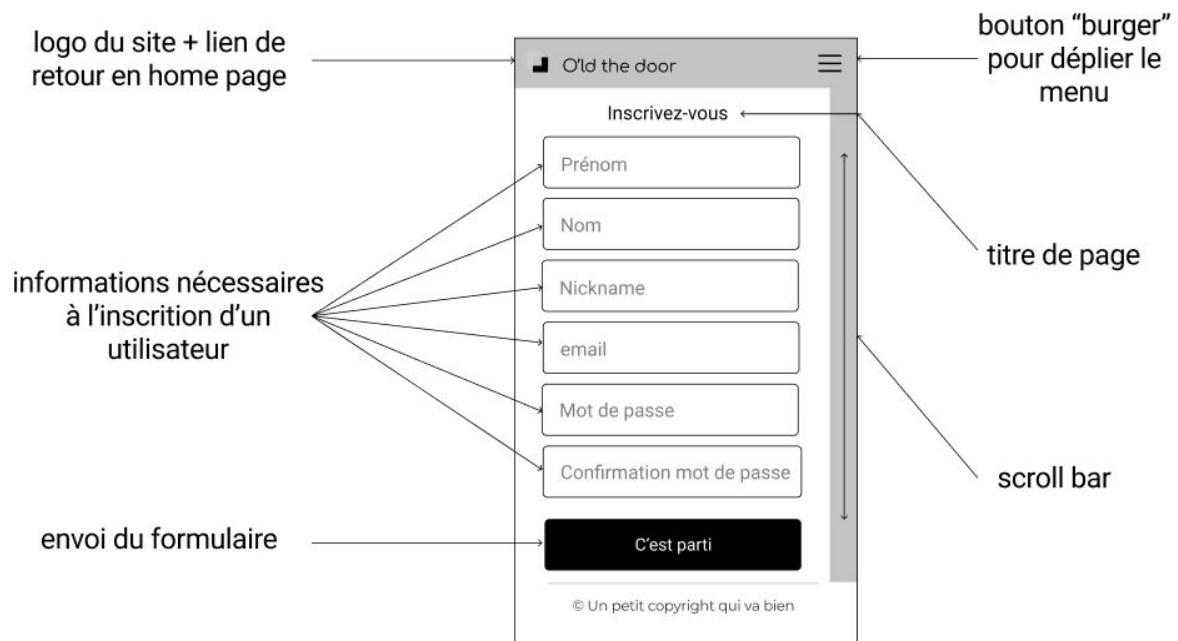
La page d'accueil :



La page d'inscription version desktop :



La page d'inscription version mobile :



La page des aventures disponibles version desktop :

The screenshot shows a desktop browser window displaying a website for "O'ld the door". The header includes the logo, navigation links for "Home", "Aventures", "Mon profil", and buttons for "Log in" and "Register". The main content area features a heading "Voici toutes les aventures de notre catalogue" followed by four adventure cards. Each card contains a title, a synopsis, and a "Découvrir" button. At the bottom, there's a footer with the logo, a link to "Notre équipe", and a copyright notice.

Titre de l'aventure → **Aventure 1**

Synopsis → Ceci est un synopsis ! Quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est.

lien pour accéder à la page de détail d'une aventure → **Découvrir**

→ **Aventure 2**

Ceci est un synopsis ! Quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est.

→ **Découvrir**

→ **Aventure 3**

Ceci est un synopsis ! Quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est.

→ **Découvrir**

→ **Aventure 4**

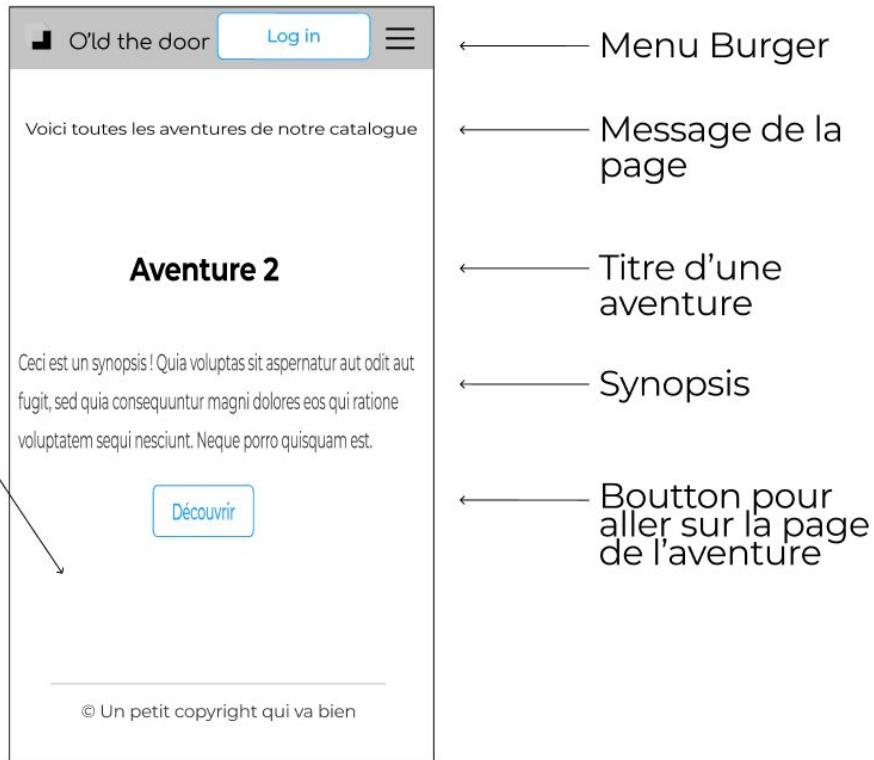
Ceci est un synopsis ! Quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est.

→ **Découvrir**

↑ Lien cliquable pour accéder à la présentation de notre équipe

La page des aventures disponibles version mobile :

Scroll pour voir les autres aventures

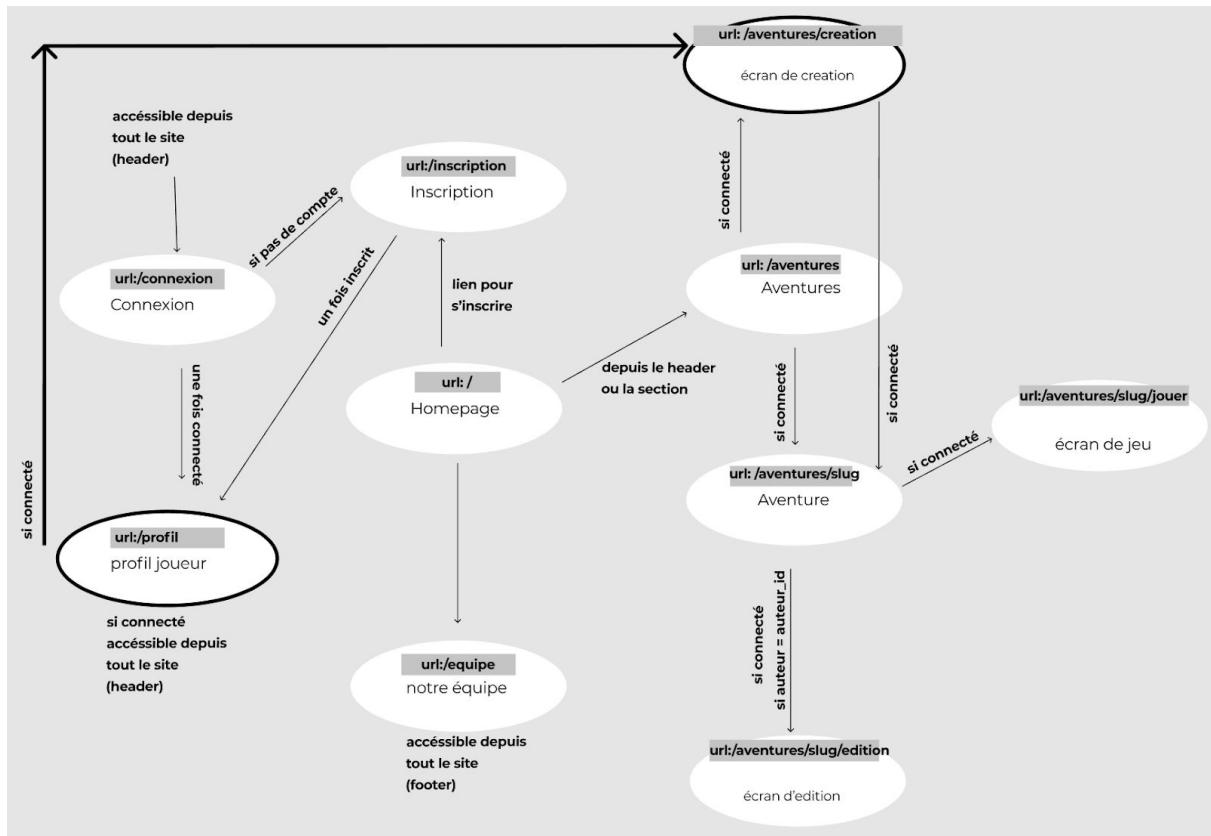


Nous avons pour tout le projet réalisé vingt-deux wireframes pour couvrir toutes les pages de notre MVP.

3d - Workflow :

Toujours en utilisant Figma, nous avons réalisé un schéma du workflow.

Le workflow ou flux de travaux, dans le cadre de la réalisation d'une application web est la représentation de la navigation du site en suivant son arborescence.

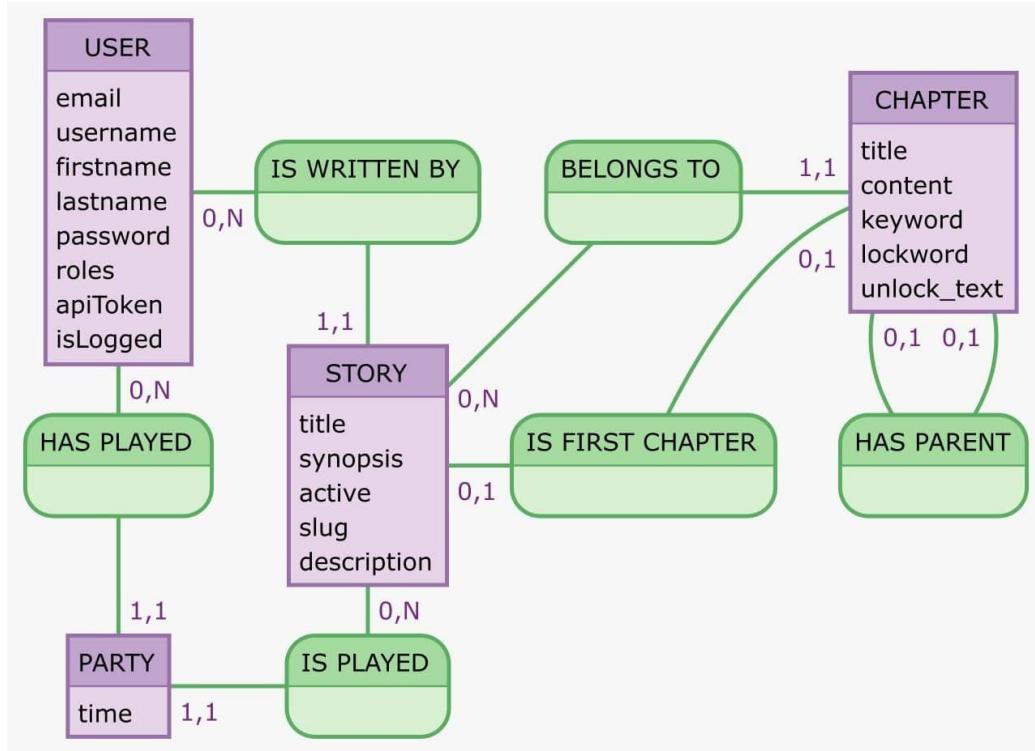


Chaque page est représentée par une forme ovale qui contient son nom ainsi que son url. Les flèches représentées, la possibilité d'aller d'une page à une autre et une légende indique si l'utilisateur doit être inscrit et/ou connecté.

Nous pouvons donc voir, par exemple, qu'un utilisateur inscrit et donc connecté peut depuis sa page de profil accéder à la page de création d'une aventure ainsi que depuis la page des aventures qui elle même est accessible depuis le header.

3e - MCD :

Le MCD ou modèle conceptuel de données est la représentation des données ainsi que leurs relations.



Nous pouvons voir ici que notre base de données contiendra quatre entités ou tables représentées en jaune avec leurs propriétés ou champs inscrits en dessous.
Les relations entre nos entités sont représentées en vert et un mot indique la nature de cette relation.

Les annotations entre nos entités et relations servent à déterminer le type de relation entre nos entités, elles s'appellent des cardinalités.

Prenons comme exemple les entités User et Party ainsi que leur relation :

- Un User a comme propriétés email, firstname, lastname, password, roles, apiToken et isLoggedIn.
- Une Party a comme propriété time.
- Un User peut jouer à aucune Party ou jouer à plusieurs Party (0..N).
- Une Party doit être jouée par un et seulement un User (1..1).

Le schéma du MCD a été réalisé sur l'outil en ligne Mocodo.

3f - Dictionnaire des données :

Suite à la création du MCD, nous avons pu réaliser un dictionnaire de données, celui-ci sert à référencer toutes les informations nécessaires à la conception d'une base de données relationnelle.

Ces informations sont, le nom du champ, le type, les spécificités ainsi qu'une brève description.

Prenons comme exemple le champ title de la table Story :

Le champ title est de type varchar(128), c'est-à-dire une chaîne de caractères d'une longueur maximale de 128 caractères. Ce champ ne peut pas être null, c'est-à-dire qu'il doit forcément contenir une chaîne de caractères. Il représente le titre de l'histoire.

Ne pouvant prévoir à l'avance la taille d'un titre d'une histoire que choisirait l'auteur, nous avons opté pour un varchar(128) afin d'avoir suffisamment de place pour des longs titres et pour en économiser pour des petits titres contrairement à un char(128) qui lui, serait toujours égal à 128 avec un remplissage à blanc.

Story

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	L'identifiant de l'histoire
title	VARCHAR(128)	NOT NULL	Le titre de l'histoire
synopsis	TEXT	NOT NULL	Le résumé de l'histoire
active	BOOL	NOT NULL, DEFAULT 0	1 = Active, 0 = Non active, permet de gérer la visibilité d'une histoire

slug	VARCHAR(255)		Slug du titre
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	La date de création de l'histoire
updated_at	DATETIME	NULL	La date de la dernière mise à jour de l'histoire
firstChapter	Entity	NULL	Par quel chapitre commence l'histoire
author	Entity	NULL	Quel utilisateur a créé l'histoire
description	TEXT	NOT NULL	La description complète de l'histoire

Chapter

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	L'identifiant du chapitre
title	VARCHAR(128)	NOT NULL	Le titre du chapitre
content	TEXT	NOT NULL	Le contenu texte du chapitre
keyword	VARCHAR(128)	NOT NULL	L'élément clé du chapitre

lockword	VARCHAR(128)	NOT NULL	L'élément serrure du chapitre
unlock_text	TEXT	NOT NULL	Le texte à afficher à la réussite de la combinaison du chapitre
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	La date de création du chapitre
updated_at	DATETIME	NULL	La date de la dernière mise à jour du chapitre
parentChapter	Entity	NULL	Le chapitre parent du chapitre en cours
forStory	Entity	NULL	L'histoire à laquelle appartient le chapitre

Party

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	L'identifiant de la partie
time	SMALLINT	NOT NULL, UNSIGNED	Le temps réalisé en secondes
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	La date de création de la partie
updated_at	TIMESTAMP	NULL	La date de la dernière mise à jour de la partie

forStory	Entity	NULL	L'histoire correspondante à la partie
player	Entity	NULL	L'utilisateur ayant fait la partie

User

Champ	Type	Spécificités	Description
id	INT	PRIMARY KEY, NOT NULL, UNSIGNED, AUTO_INCREMENT	L'identifiant de l'utilisateur
email	VARCHAR(128)	NOT NULL, UNIQUE	L'email de l'utilisateur
firstname	VARCHAR(64)	NOT NULL	Le prénom de l'utilisateur
lastname	VARCHAR(64)	NOT NULL	Le nom de l'utilisateur
username	VARCHAR(64)	NOT NULL	Le surnom de l'utilisateur
password	VARCHAR(255)	NOT NULL	Le mot de passe de l'utilisateur
roles	VARCHAR(64)	NULL	Les rôles de l'utilisateur
api_token	VARCHAR(255)	UNIQUE, NULL	Le token de connexion de l'utilisateur

isLoggedIn	BOOLEAN	NULL, DEFAULT 0	Statut connecté de l'utilisateur
------------	---------	-----------------	----------------------------------

3g - Endpoints :

Pour la mise en place d'une API, nous avons dû définir les endpoints de celle-ci. Les endpoints sont les parties variables de l'URL utilisée pour communiquer avec l'API. Nous avons défini tous les endpoints qui nous semblaient nécessaire en indiquant la méthode HTTP, les données à transmettre ainsi qu'une brève description. Le but de cette préparation était de se projeter dans le projet et anticiper les besoins et problèmes de la communication front-back.

Liste des Endpoints :

Stories

EndPoint	Méthode HTTP	Données(s) à transmettre	Description
/api/v0/stories	GET		Récupérer toutes les histoires
/api/v0/stories?last=[int]	GET	Paramètre last en indiquant le nombre d'histoires voulus	Récupérer le nombre d'histoires voulus
/api/v0/stories/?author_id=[id]	GET	Paramètre author_id indiquant les histoires écrites par une personne en particulier	Récupère les histoires d'un auteur
/api/v0/stories/[id]	GET		Récupérer une histoire selon son id

<code>/api/v0/stories/[sluggedTitle]</code>	GET		Récupérer une histoire selon son slug
<code>/api/v0/stories/[id]/time</code>	GET		Récupérer les meilleurs temps et temps moyens pour une partie
<code>/api/v0/stories/count</code>	GET		Renvoyer le nombre histoires actives pour afficher sur la home dans le texte
<code>/api/v0/stories</code>	POST	<code>title, synopsis , active , first chapter, author</code>	Créer une histoire
<code>/api/v0/stories/[id]</code>	PUT	<code>id, title, synopsis , active , first chapter</code>	Modifier un histoire selon son id
<code>/api/v0/stories/[id]/active</code>	PUT	Get parameter set=true or set=false	Activer / Désactiver une histoire
<code>/api/v0/stories/[id]</code>	DELETE		Supprimer une histoire selon son id

Chapters

EndPoint	Méthod e HTTP	Données(s) à transmettre	Description
/api/v0/chapters	GET	Id de l'histoire en cours en paramètre get Paramètre facultatif : non_parent pour recevoir que les chapitres qui ne sont pas encore parents	Récupérer tous les chapitres d'une histoire
/api/v0/chapters/[id]	GET		Récupérer un chapitre selon son id
/api/v0/chapters/[id]/child	GET		Récupérer le chapitre enfant d'un chapitre selon son id
/api/v0/chapters	POST	title , text , key , lock , unlock text	Créer un chapitre
/api/v0/chapters/[id]	PUT	title , text , key , lock , unlock text	Modifier un chapitre
/api/v0/chapters/[id]	DELETE		Supprimer un chapitre

Parties

EndPoint	Méthode HTTP	Données(s) à transmettre	Description
<code>/api/v0/parties ?user_id=[id]</code>	GET	Id de l'utilisateur connecté	Récupérer toutes les parties de l'utilisateur connecté
<code>/api/v0/parties</code>	POST	time	Créer un partie

Users

EndPoint	Méthode HTTP	Données(s) à transmettre	Description
<code>/api/v0/users/[id]</code>	GET		Récupérer un utilisateur en fonction de son id
<code>/api/v0/users</code>	POST	Email, firstname, lastname, password, role	Créer un utilisateur
<code>/api/v0/user/login</code>	POST	Email, Password	Vérifier si un user est présent en base
<code>/api/v0/user/logout</code>	GET		Déconnecte le user

Nous avons également fais une liste des contrôleurs pour ces endpoints avec comme informations supplémentaires le nom de la route, le nom du contrôleur et le nom de la méthode afin que toute l'équipe ai à disposition ces informations lors du développement.

Liste des contrôleurs pour les endpoints :

Stories

URL	Nom de la route	Méthode HTTP	Contrôleur	Méthode
/api/v0/stories	api_v0_stories_list	GET	StoryController	list
/api/v0/stories/?last=[int]	api_v0_stories_list	GET	StoryController	list
/api/v0/stories/?author_id=[id]	api_v0_stories_list	GET	StoryController	list
/api/v0/stories/[id]	api_v0_stories_show	GET	StoryController	show
/api/v0/stories/[sluggedTitle]	api_v0_stories_showBySlug	GET	StoryController	showBySlug
/api/v0/stories/[id]/time	api_v0_stories_time	GET	StoryController	getTime
/api/v0/stories/[id]/count	api_v0_stories_count	GET	StoryController	count
/api/v0/stories	api_v0_stories_add	POST	StoryController	add
/api/v0/stories/[id]	api_v0_stories_edit	PUT	StoryController	edit
/api/v0/stories/[id]/active	api_v0_stories_active	PUT	StoryController	active
/api/v0/stories/[id]	api_v0_stories_delete	DELETE	StoryController	delete

Chapters

URL	Nom de la route	Méthode HTTP	Contrôleur	Méthode
/api/v0/chapters/?story_id=[id]	api_v0_chapters_list	GET	ChapterController	list
/api/v0/chapters/?story_id=[id] & non_parent=true	api_v0_chapters_list	GET	ChapterController	list
/api/v0/chapters/[id]	api_v0_chapters_show	GET	ChapterController	show
/api/v0/chapters/[id]/child	api_v0_chapters_show_child	GET	ChapterController	showChild
/api/v0/chapters	api_v0_chapters_add	POST	ChapterController	add
/api/v0/chapters/[id]	api_v0_chapters_edit	PUT	ChapterController	edit
/api/v0/chapters/[id]	api_v0_chapters_delete	DELETE	ChapterController	delete

Parties

URL	Nom de la route	Méthode HTTP	Contrôleur	Méthode
/api/v0/parties?user_id=[id]	api_v0_parties_list	GET	PartyController	list
/api/v0/parties	api_v0_parties_add	POST	PartyController	add

Users

URL	Nom de la route	Méthode HTTP	Contrôleur	Méthode
/api/v0/users/[id]	api_v0_users_show	GET	UserController	show
/api/v0/users	api_v0_users_add	POST	UserController	add

Login

URL	Nom de la route	Méthode HTTP	Contrôleur	Méthode
/api/v0/login	api_v0_login	POST	LoginController	login

4 - Spécifications techniques :

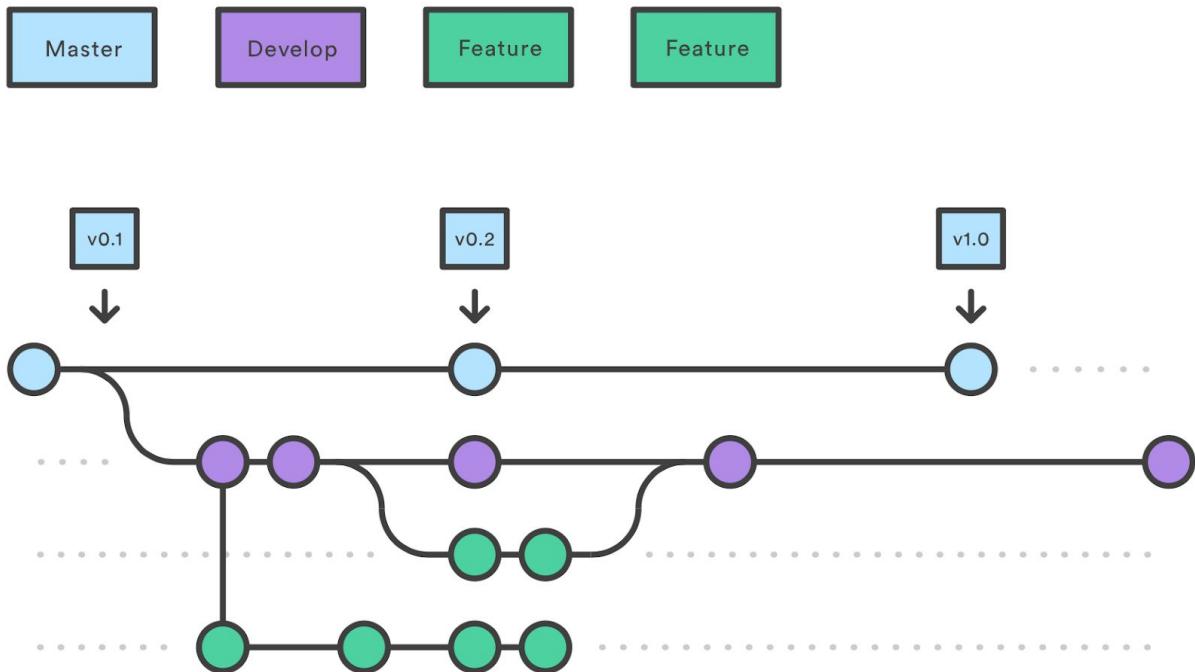
4a - Versionning :

Nous avons choisi d'utiliser un versionning inspiré et simplifié de la méthode Gitflow, cette méthode se base sur la création de plusieurs branches qui ont chacune un but distinct.

- Une branche de développement (develop) qui est notre branche principale.
- Une branche de version (release/vN°) qui est créée depuis develop à chaque nouvelle version.
- Une branche pour chaque fonctionnalité (feature) qui est créée depuis develop puis merge sur develop via une pull request lorsque la fonctionnalité est finie.

Pour les commits, nous avons également choisi de les préfixer avec un “b-” ou “f-” selon si la feature était côté back ou front.

Les branches peuvent être également préfixées.



4b - Front :

Nous étions trois développeurs front-end ayant fait la spécialisation React lors de la formation, c'est donc tout naturellement que nous avons choisi de faire le front en utilisant React.

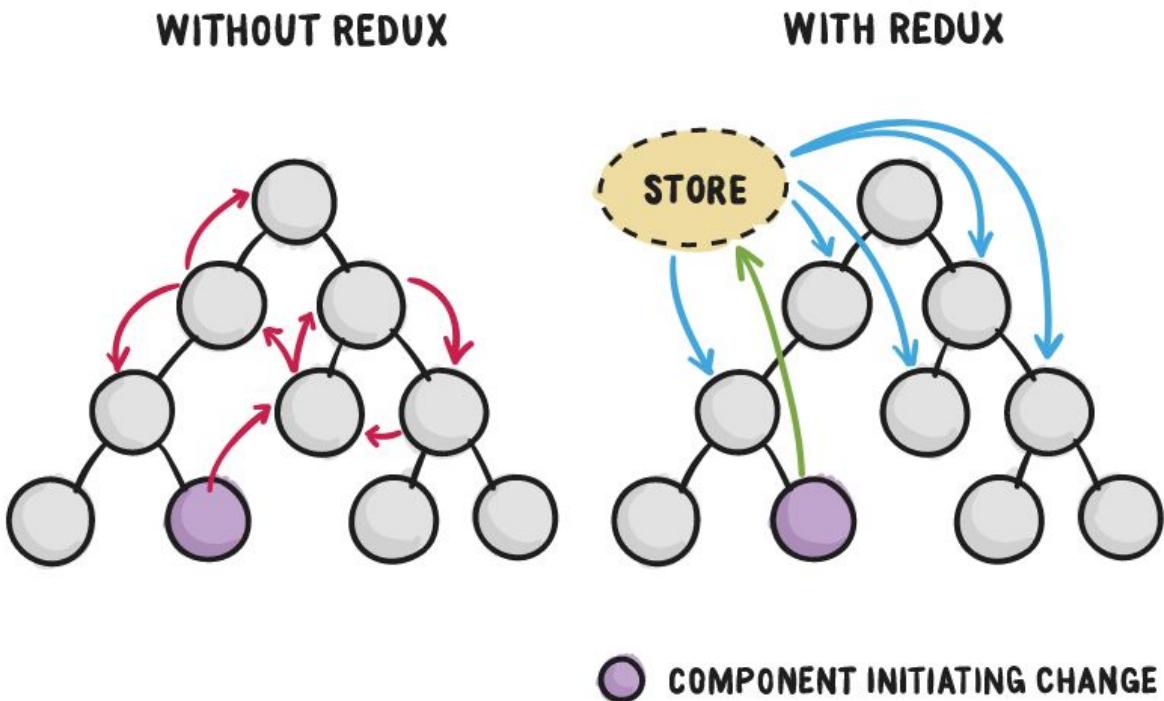
Nous avons également utilisé ESLint avec la configuration AirBnB qui permet d'analyser le code et afficher des erreurs si les règles de la configuration ne sont pas respectées dans le but d'avoir un code homogène.

React est une bibliothèque JavaScript qui permet de créer des interfaces utilisateurs dynamiques, en découplant les éléments de chaque page en “composants” écrit en JSX (JavaScript Syntax eXtension). Le JSX pourrait être décrit comme un mélange entre JavaScript et HTML.

Les données nécessaires pour l'affichage dynamique sont stockées dans le “state” et les composants se basent sur ce state et ses changements pour l'affichage en comparant le DOM (Document Object Model) avec un DOM virtuel créé par React qui permet de ne rafraîchir que les composants où les données ont été changées.

Nous avons également utilisé React Router qui permet la gestion des routes côté front. Pour la gestion des données, nous avons choisi d'utiliser Redux qui permet de stocker les données du state dans un store afin que tous les composants puissent avoir accès au state

ce qui facilite le déplacement de données entre composants.



Nous avons également utilisé Axios qui permet la communication avec une API de façon simple, ainsi que la bibliothèque Typiste, qui est un composant React permettant l'ajout d'un effet dactylographique d'un texte et nous a permis de gagner du temps lors du développement. Le CSS a été fait en utilisant SASS principalement pour sa syntaxe et l'utilisation pratique de variables qui nous a permis de tester rapidement plusieurs palettes de couleurs quand nous n'étions pas sûrs de la direction à prendre pour le design.

4C - Back :

Les deux développeurs back-end ayant fait la spécialisation Symfony, ils ont également choisi tout naturellement d'utiliser Symfony pour le projet.

Le framework (un ensemble d'outils et composants permettant le développement) PHP Symfony a été utilisé dans sa version complète qui comprend plusieurs bundle (ensemble de fichier qui ajoute des fonctionnalités) afin de gagner du temps sur la configuration en contrepartie d'un alourdissement du projet.

Symfony a été utilisé afin de créer une architecture "MVC" et la mise en place d'une API REST.

L'ORM (Object-Relational Mapping) Doctrine a été utilisé pour interagir avec la base de données ce qui permet une interaction sous forme d'objet et en utilisant un langage proche du SQL (Structured Query Language, le langage de manipulation des données d'une base de données) le DQL.

Nous avons utilisé le bundle de Doctrine, Fixtures lors du développement pour créer de "fausses" données dans notre base de données et ainsi pouvoir l'utiliser rapidement et facilement.

Le bundle Nelmio CORS a été utilisé pour la gestion des CORS (Cross Origin Ressources Sharing) afin de permettre au front la communication avec le back via les headers lors d'une requête HTTP.

Le bundle Security qui grâce à ces multiples fonctionnalités a permis :

- D'encoder les mots de passe des utilisateurs dans la base de données pour qu'ils ne soient pas en "clair".
- D'ajouter un système d'authentification via token lors d'une requête API.
- De faire une vérification des droits d'accès en fonction du rôle et si l'utilisateur est connecté ainsi qu'une vérification plus poussée pour les fonctionnalités d'ajout, de modification et de suppression via des Voters pour être sûr que l'utilisateur a bien le rôle nécessaire et est connecté.

La génération des tokens a en plus été faite via le bundle Polyfill UUID de Symfony afin de s'assurer qu'ils soient uniques.

5 - Organisation :

5a - L'équipe :

Nous étions cinq personnes pour la réalisation de ce projet, deux développeurs back-end et trois développeurs front-end.

- Damien TOSCANO, il est la personne qui a eu l'idée du projet, il était donc cohérent qu'il soit le Product Owner, à savoir la personne en charge du projet et des décisions liées au cahier des charges. Il était également développeur back-end.
- Maxence ROYER, il lui a été confié le rôle de lead développeur back, il était en charge du développement de la partie back-end et devait veiller à son bon fonctionnement ainsi qu'apporter soutien et conseil sur celle-ci.
- Hugo LIGIER, développeur front-end et Git master, il avait comme rôle de superviser le versionning du projet ainsi que faire une première révision du code lors des pull request.
- Jukka PASANEN, développeur front-end et Scrum master, il devait rédiger le journal de bord de l'équipe et y noter l'avancée du projet ainsi qu'animer les réunions d'équipe appelées daily scrum pour faciliter la transmission d'informations au sein de l'équipe.
- Tony HERBET LE FAUCHEUR, j'ai été choisi pour être lead développeur front, j'avais en charge le développement de la partie front-end et devait veiller à son bon fonctionnement ainsi qu'apporter soutien et conseil sur celle-ci.

Ce projet étant le premier de l'équipe, nos rôles se sont souvent chevauchés et toute l'équipe a aidé sur quasiment tous les aspects du développement à l'exception de l'écriture de code back-end par l'équipe front-end.

En effet, lorsque l'équipe front-end était en retard, l'équipe back-end est venue aider, ce qui nous a permis de rattraper notre retard.

5b - Outils :

Nous avons utilisé git pour la gestion des versions du projets et la plateforme github pour héberger le projet.

Pour le déploiement du projet, nous avons utilisé une instance de machine virtuelle EC2 (Elastic Compute Cloud) via AWS (Amazon Web Services).

Nous y avons installé les logiciels nécessaires à la mise en ligne du projet à savoir Apache (serveur web), PHP, git, MySQL et composer.

Une fois les logiciels nécessaires installés, nous avons pu y mettre le repository github du projet et utilisé la dernière branche de release/v.

Nous avons fait une release par semaine qui a été déployée par l'équipe back-end à chaque fois.

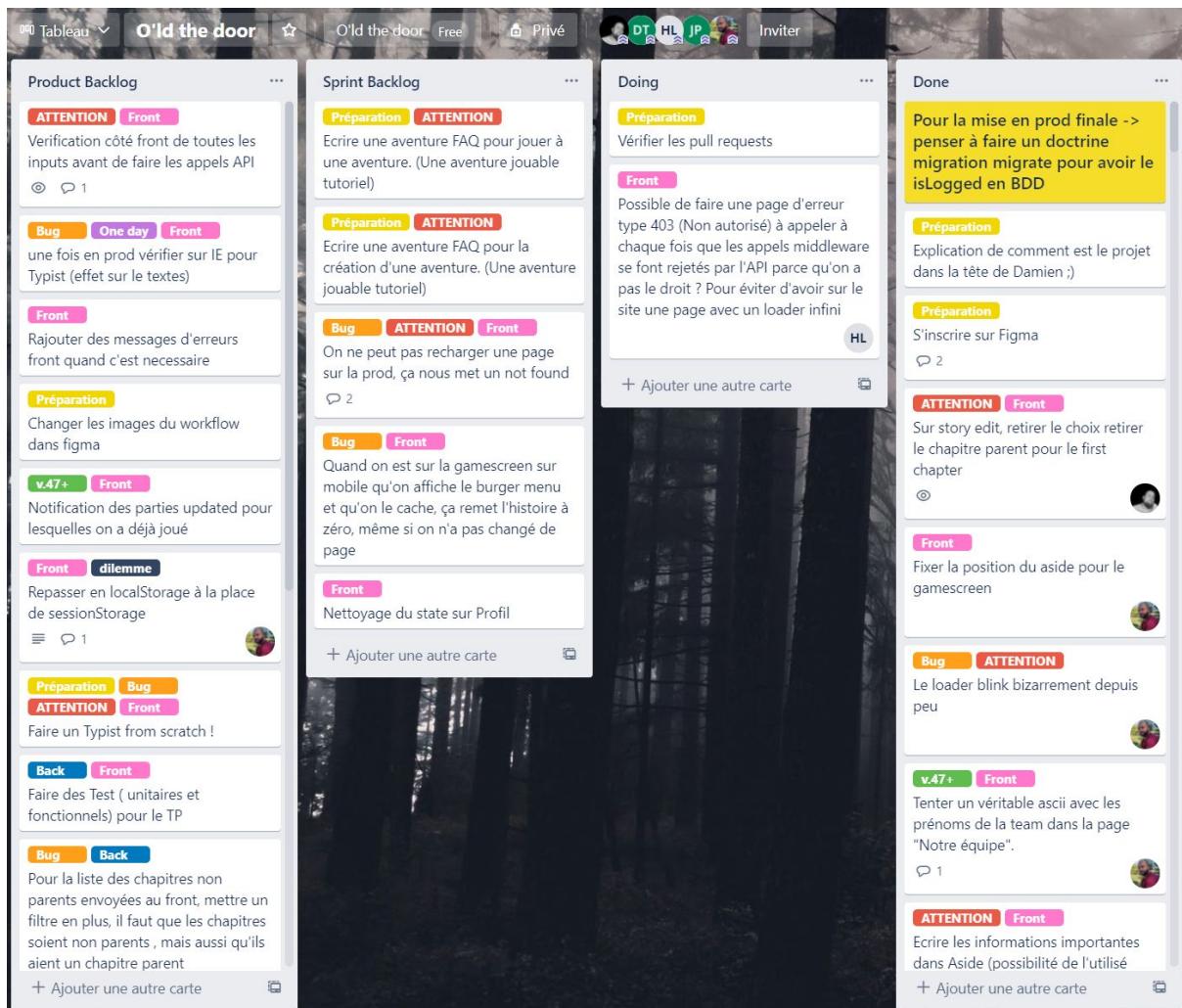
Pour cette présentation, je me suis occupé de faire une nouvelle release avec l'aide et les explications de l'équipe back-end.

Pour aider à l'organisation du travail en équipe, nous avons utilisé l'outil en ligne Trello qui permet d'organiser et décomposer le projet en tâches.

Les tâches peuvent être assignées à des membres du projet, elles peuvent avoir des étiquettes qui aide à visualiser le(s) domaine(s) auxquels elles appartiennent, des commentaires et peuvent être déplacées selon leurs avancements et/ou la phase de développement suivant la méthodologie de travail utilisée (je reviendrais sur ce point plus bas).

Voici les différentes étiquettes que nous avons instaurées :

- Front, tout ce qui est en rapport avec le front-end.
- Back, tout ce qui est en rapport avec le back-end.
- Attention, lorsqu'une tâche nécessite une attention particulière car sensible ou défectueuse.
- Dilemme, lorsqu'un choix important est à faire et que nous n'avons pas encore tranché.
- Bug, lorsqu'un bug a été repéré ou que la tâche est associée à une autre tâche ayant un bug.
- Préparation, une tâche qui nécessite une préparation en amont, généralement en dehors du code.
- v++, fonctionnalités qui seront ajoutées lors d'une version future encore non définie.



5c - Méthodologie :

Nous avons travaillé selon la méthodologie Scrum qui se base sur les principes de la méthode agile.

Elle consiste à diviser le développement du projet en “sprint” qui peuvent avoir une durée variable, pour l’Apothéose la durée d’un sprint était d’une semaine.

A chaque fin de sprint, nous devions présenter le projet et son évolution. Dans ce but, nous avons fait quatre releases qui ont été déployées à chaque fois.

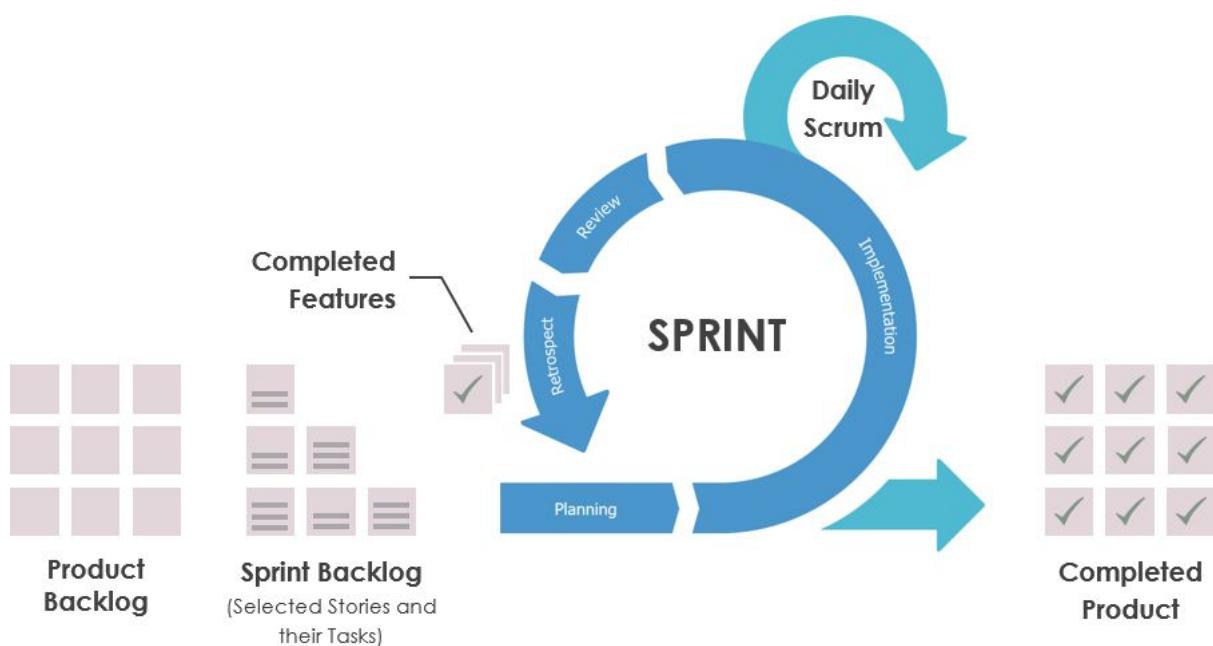
Le but de cette méthodologie est de travailler de concert avec le client, en l’impliquant dans chaque évolution du projet et en accueillant ses remarques et demandes de modification de façon positive. Cela a pour effet d’avoir un projet qui évolue rapidement et régulièrement en ajoutant des fonctionnalités à grande valeur ajoutée.

Lors de notre première réunion, nous avons défini toutes les tâches que nous aurions à réaliser lors du développement, le Product backlog. Il est amené à changer au fur et à mesure de l'avancement du développement et des demandes reçues.

Au début de chaque sprint, nous avons défini les objectifs de celui-ci en remplissant sur trello le Sprint Backlog via les tâches du Product backlog.

Tous les matins, nous avions une réunion appelée Daily Scrum, animée par le Scrum Master afin de transmettre toutes les informations nécessaires au bon déroulement du projet. Nous parlions des tâches que chacun allait effectuer dans la journée, des problèmes que nous avions rencontré la veille, demander de l'aide ou poser des questions si nécessaire.

A la fin de chaque sprint, nous avons fait un Sprint review afin de montrer l'avancement du projet et revenir sur ce qui a été fait depuis le dernier Sprint Review puis un Sprint Retrospective afin d'analyser le dernier sprint et modifier l'organisation des prochains sprints ou de la méthode de travail si besoin.



Nous avons fait quatre sprints :

1. Nous avions comme consigne l'élaboration du projet et la réalisation du cahier des charges sans impliquer de code.
2. Une fois le premier sprint validé, nous avons pu commencer le développement de notre MVP.
3. Finir ce qui avait été commencé lors du sprint 2 et continuer le développement du MVP.
4. Finir ce qui avait été commencé lors du sprint 3 et polir le projet sans développer de nouvelles fonctionnalités majeures.

Une journée de travail type commençait à 9h par un Daily Scrum sur discord puis nous nous séparions entre back et front si besoin et commençons à coder.

Une pause déjeuner d'environ une heure avait lieu aux alentours de midi puis nous continuons de coder en étant sur discord jusqu'à 17h, l'heure à laquelle les parents du groupe devaient se déconnecter de discord.

Avant de se déconnecter, nous faisions une dernière réunion pour revenir très rapidement

sur la journée et annoncer le cas échéant ce que nous allions faire dans la soirée afin que tout le monde puisse travailler sur quelque chose de différent.
C'est souvent lors de ces heures de travail en soirée ou le week-end que nous testions des designs, modifications ou fonctionnalités qui n'avaient jusqu'alors pas été envisagés et que nous présentions lors du prochain Daily scrum.

6 - Réalisations :

Je vais dans cette section détailler mes réalisations personnelles lors du projet pour chaque sprint ainsi qu'expliquer ce qu'a fait l'équipe lorsque cela est nécessaire.

6a - Sprint 0 :

Ce sprint avait pour but la conception du projet et la rédaction du cahier des charges dans son entièreté.

Damien TOSCANO avait déjà réalisé quelques wireframes et rédigé les fonctionnalités principales et la description de l'application comme support de proposition du projet.

Nous avons donc repris ce qui avait déjà été fait et peaufiné le concept en équipe en donnant les points de vue côté back et front et ainsi rédigé le cahier des charges dans son intégralité.

Les wireframes initiales ayant été faites sur l'outil en ligne Figma, nous avons donc décidé de continuer son utilisation malgré la méconnaissance de l'outil par le reste de l'équipe car c'était une bonne occasion pour apprendre son fonctionnement.

Cela s'est révélé être un petit challenge à part entière car nous avions l'habitude d'utiliser un autre outil en ligne, whimsical qui est plus simple et facile d'accès car il contient beaucoup moins de fonctionnalités et est moins avancé.

Nous nous sommes donc répartis les pages du projet pour en faire deux chacun et le reste en groupe. Toutes les pages ont été revues par toute l'équipe avant de les valider.

En plus de la page des aventures qui est disponible dans la partie Maquettes du Cahier des charges de ce dossier, je me suis occupé de la page d'une aventure.

Page d'une aventure version desktop :

Description de l'aventure

Bouton pour commencer l'aventure

Un super beau titre d'aventure

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Purus in massa tempor nec feugiat nisl pretium fusce. A condimentum vitae sapien pellentesque habitant morbi tristique. Consequat semper viverra nam libero. Mauris pharetra et ultrices neque ornare aenean euismod elementum nisi. Vel fringilla est ullamcorper eget nulla facilisi etiam dignissim diam. Sociis natoque penatibus et magnis dis parturient montes nascetur. Tellus in hac habitasse platea dictumst vestibulum rhoncus est pellentesque. Praesent elementum facilisis leo vel fringilla. Scelerisque eleifend donec pretium vulputate sapien nec. Pretium aenean pharetra magna ac placerat. Dolor morbi non arcu risus quis varius. Suscipit tellus mauris a diam maecenas sed enim ut. Ut enim blandit volutpat maecenas. Tellus cras adipiscing enim eu turpis egestas pretium aenean. Etiam non quam lacus suspendisse faucibus. Integer quis auctor elit sed vulputate mi sit amet mauris.

Commencer

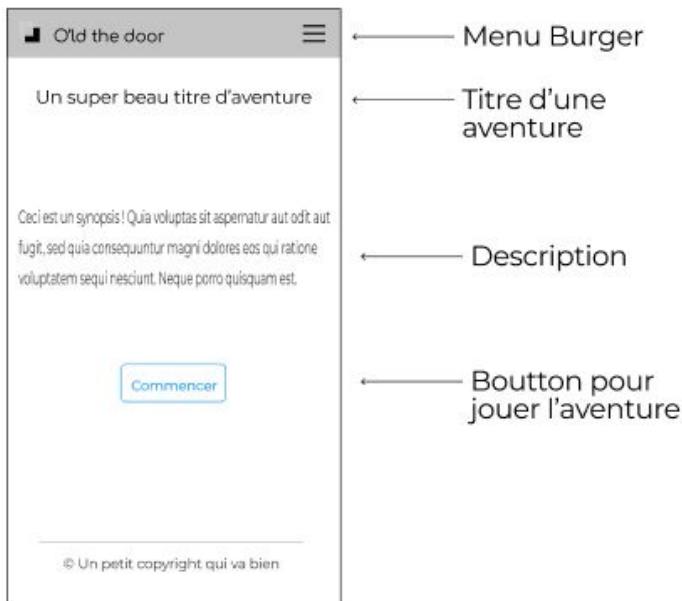
■ O'ld the door

Notre équipe

© Un petit copyright qui va bien

Lien cliquable pour accéder à la présentation de notre équipe

Page d'une aventure version mobile :



Nous avons également mis en place tous les outils nécessaires au bon déroulement du projet comme le Trello, des google docs et sheets afin de noter et mettre à disposition de toute l'équipe toutes informations utiles et nécessaires à l'aboutissement du projet.

Une fois le cahier des charges “terminé”, nous avons eu le feu vert du référent de promotion pour commencer à coder.

Nous avons créé les dossiers front et back dans notre repository github pour le projet et mis en place nos architectures initiales (Symfony et React avec leurs bibliothèques).

Côté front, nous nous sommes répartis les pages du site pour commencer à créer nos composants en version statique.

Je me suis occupé de faire la page d'accueil, la page d'aventures (le “catalogue”), le composant AdventureSmall (le composant qui est utilisé sur la page d'accueil et la page d'aventures), la page d'équipe et la page d'inscription.

Le lendemain, nous avons, en équipe, discuté du design du site et fait des recherches pour trouver de l'inspiration.

Je me suis également occupé de trouver la police d'écriture et de la mettre en place.

Une fois toutes nos pages créées, j'ai pu mettre en place la base de la navigation en utilisant la bibliothèque react-router. Cela m'a également demandé de créer les composants Nav et Header. J'ai par la suite ajouté les nouvelles routes au fur et à mesure.

Les liens sont mis en place via le composant “Link” de react-router. Cela permet, lorsqu'un utilisateur clique dessus et change l'URL, d'interpréter la nouvelle URL et de changer l'affichage dynamiquement tout en restant sur la même page. C'est le principe d'une SPA (Single-page application ou application web monopage).

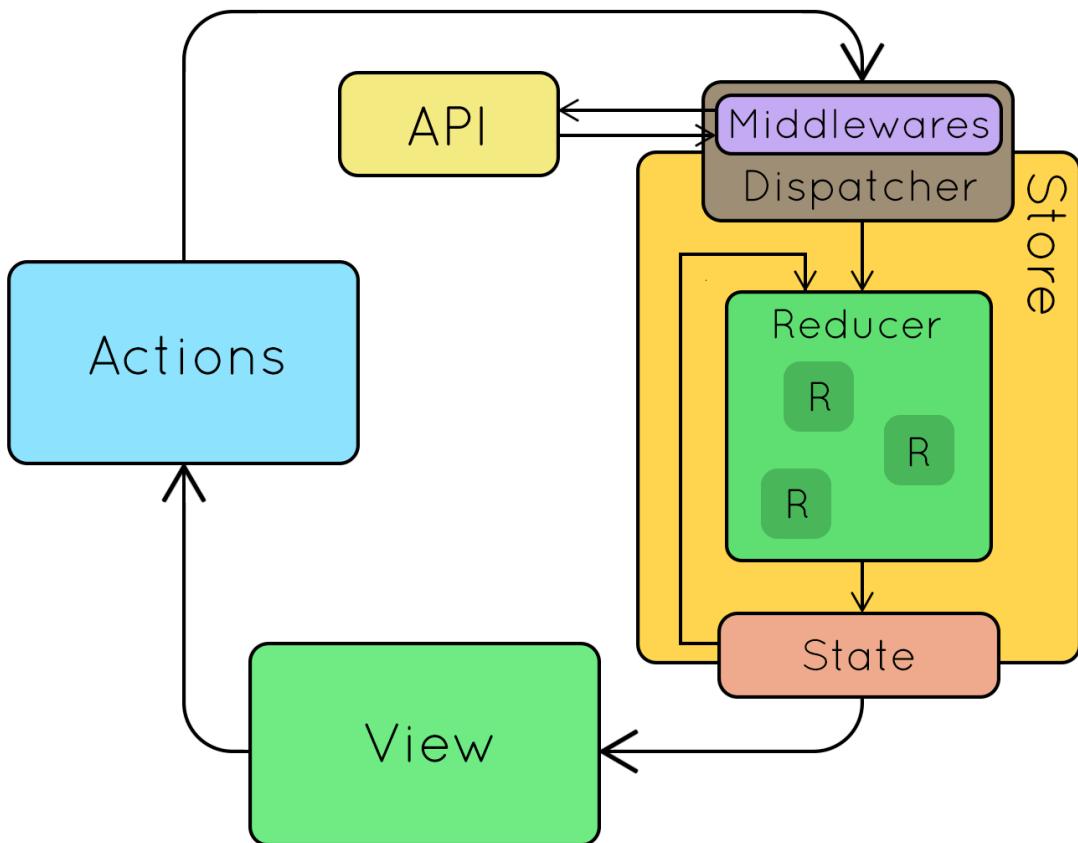
Nos routes sont définies via le composant “Route” de react-router et nécessitent un “path” qui est la partie de l'URL utilisée par les “Link”. Elles sont englobées par le composant “Switch” qui fait un rendu de la première Route qui correspond à l'URL, il est donc important de mettre les Routes dans un ordre précis.



```
1 const App = ({ burgerMenuOpen, isLoggedIn }) => (
2   <div className="app">
3     <Header />
4     {burgerMenuOpen && <Nav />}
5     {!burgerMenuOpen && (
6       <Switch>
7         <Route exact path="/">
8           <Home />
9         </Route>
10        <Route exact path="/connexion">
11          <Connexion />
12        </Route>
13        <Route exact path="/inscription">
14          <Register />
15        </Route>
16        <Route exact path="/aventures/creation">
17          <Aside data={asideData.storyCreate} />
18          {isLoggedIn !== true && (<Redirect to="/connexion" />)}
19          {isLoggedIn === true && (<StoryCreate />)}
20        </Route>
21        <Route exact path="/aventures/:slug/jouer">
22          <Aside data={asideData.gameScreen} />
23          {isLoggedIn !== true && (<Redirect to="/connexion" />)}
24          {isLoggedIn === true && (<GameScreen />)}
25        </Route>
26        <Route exact path="/aventures/:slug/edition">
27          <Aside data={asideData.storyEdit} />
28          {isLoggedIn !== true && (<Redirect to="/connexion" />)}
29          {isLoggedIn === true && (<StoryEdit />)}
30        </Route>
31        <Route exact path="/aventures/:slug">
32          {isLoggedIn !== true && (<Redirect to="/connexion" />)}
33          {isLoggedIn === true && (<Adventure />)}
34        </Route>
35        <Route exact path="/aventures">
36          <Adventures />
37        </Route>
38        <Route exact path="/equipe">
39          <Team />
40        </Route>
41        <Route exact path="/profil">
42          <Profil />
43        </Route>
44        <Route>
45          <PageError404 />
46        </Route>
47      </Switch>
48    )}
49    <Footer />
50  </div>
51 );
```

Une fois les routes terminées pour le moment, je me suis occupé de mettre en place l'architecture redux à savoir les containers, le store, les reducers, les actions et les middlewares.

Voici un gif qui schématisé le fonctionnement de redux :



Avec cette architecture, notre View est un composant avec un container associé qui lui permet d'être connecté avec le store pour avoir accès aux informations du state dont il a besoin ainsi qu'aux actions.

Lorsqu'un événement est activé dans la View, l'action qui en découle est envoyée au store. Elle va ainsi parcourir tous les middlewares jusqu'à trouver un "cas" pour cette action. Si un "cas" est présent, il est résolu et fait une requête API ainsi que toutes autres actions nécessaires. Une fois la réponse de notre API reçue, l'action est envoyée au reducer avec toutes autres actions qui pourraient être dans notre "cas" du middleware. L'action va parcourir tous les "cas" du reducer de la même façon qu'avec le middleware et modifier le state si besoin est. Notre View se basant sur l'état du state, si un changement se produit l'affichage est mis à jour pour les parties concernées.

Une fois redux mis en place, j'ai pu commencer par une fonctionnalité simple, le burger menu pour la barre de navigation sur mobile.

Pour réaliser le burger menu, j'ai créé un composant BurgerMenu qui ne sert que de bouton via l'EventListener "onClick" de react sur un icône qui change selon un booléen lors d'un

clic. Les icônes sont importées depuis la bibliothèque react-feather qui permet de les utiliser sous forme de composant.

```
● ● ●
1 import React from 'react';
2 import PropTypes from 'prop-types';
3 import { FolderPlus, FolderMinus } from 'react-feather';
4
5 import './burgerMenu.scss';
6
7 const BurgerMenu = ({ toggleBurgerMenuOpen, burgerMenuOpen }) => (
8   <div
9     className="burgerMenuContainer"
10    onClick={toggleBurgerMenuOpen}
11   >
12     <span className="menu">Menu</span>&nbsp; {burgerMenuOpen ? <FolderMinus /> : <FolderPlus />}
13   </div>
14 );
15
16 BurgerMenu.propTypes = {
17   burgerMenuOpen: PropTypes.bool.isRequired,
18   toggleBurgerMenuOpen: PropTypes.func.isRequired,
19 };
20
21 export default BurgerMenu;
22
```

(On peut également voir l'utilisation de la bibliothèque "prop-types" qui permet de forcer le type des arguments passés à notre composant et définir s'ils sont obligatoires.)

Ce composant BurgerMenu se trouve dans un composant Header qui s'occupe, comme son nom l'indique, du header de notre application et qui contient également le composant Nav qui est la barre de navigation.

Sur l'image plus haut de notre composant App que j'ai utilisé pour montrer les routes, on peut voir ligne 4 et 5 que le composant Nav est également utilisé en dehors du composant Header.

Cela permet, en se basant sur le booléen d'ouverture du BurgerMenu, d'afficher soit notre Nav soit le composant de la page sur laquelle nous sommes.

En ajoutant à cela des règles CSS, nous sommes sûrs de n'avoir toujours qu'un Nav de visible car si BurgerMenu n'est pas disponible, la Nav qui se trouve dans App ne serait jamais affichée.

Des règles CSS en fonction de la taille d'écran permettent donc d'avoir une Nav qui prend tout l'écran lorsque l'on clique sur le BurgerMenu ou d'être visible dans le Header quand le BurgerMenu n'est plus affiché.

CSS du BurgerMenu :

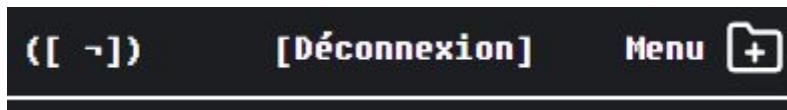
```
1 @import 'src/styles/vars';
2
3 .burgerMenuContainer {
4   position: absolute;
5   right: 0.5em;
6   display: flex;
7   justify-content: center;
8   align-items: center;
9
10  .menu {
11    &:hover {
12      color: $backgroundColor;
13      background-color: #fff;
14    }
15  }
16 }
17
18
19 @media only screen and (min-width: 768px) {
20   .burgerMenuContainer {
21     display: none;
22   }
23 }
24
```

CSS de la Nav :

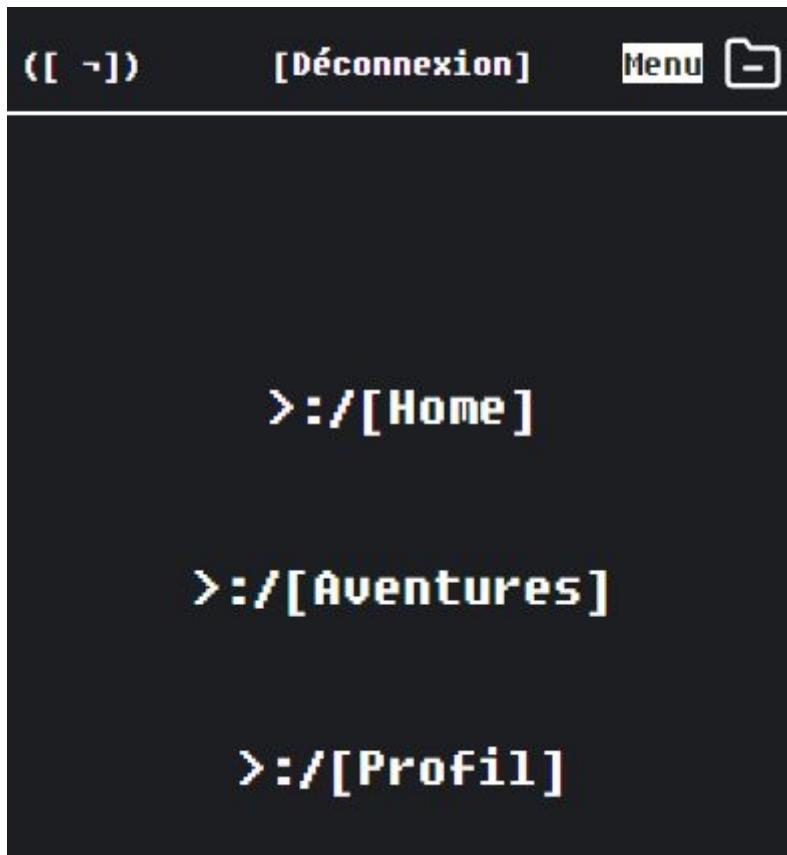
```
1 @import 'src/styles/vars';
2
3 .nav {
4   display: flex;
5   flex-direction: column;
6   align-items: center;
7   padding-top: $baseSize * 5;
8   padding-bottom: 100vh;
9   color: #fff;
10
11 .nav-item {
12   padding: $baseSize;
13   font-size: 1.5em;
14   &:hover {
15     background-color: #fff;
16     color: $backgroundColor;
17   }
18 }
19 }
20
21 @media only screen and (min-width: 768px) {
22   .nav {
23     margin-bottom: auto;
24     display: flex;
25     flex-direction: row;
26     padding-top: 0;
27     padding-bottom: 0;
28
29     .nav-symbol {
30       display: none;
31     }
32
33     .nav-item {
34       padding: 0em;
35       margin-left: $baseSize;
36       font-size: 1em;
37     }
38   }
39 }
```

Nav pour mobile :

BurgerMenu fermé :



BurgerMenu ouvert :



Nav pour Desktop :



6b - Sprint 1 :

J'ai commencé par mettre en place un début de design en m'occupant du CSS sur une grande partie du site, principalement le flexbox et l'emplacement de tous les éléments. Le design a été changé à plusieurs reprises tout au long du développement car nous n'étions pas satisfaits et avons eu du mal à trouver un juste milieu entre simplicité et effets donnant un aspect rétro.

Je me suis ensuite occupé de la page 404 en m'inspirant du célèbre “blue screen of death” en y ajoutant un easter egg à l'intention de notre référent de promo.

J'ai essayé trois bibliothèques différentes pour avoir un effet dactylographique sur le texte et j'ai retenu Typist. Je l'ai donc ajouté au projet et utilisé aux endroits voulus déjà mis en place et j'ai expliqué à l'équipe front son fonctionnement pour qu'ils puissent également l'utiliser.

J'ai modifié le composant Footer en intégrant le lien de la page “L'équipe” et en l'ajoutant à nos routes déjà existantes.

Sur la page d'accueil, le “catalogue” (à savoir la page aventures), ainsi que sur la page de profil pour les aventures dont l'utilisateur est l'auteur, nous voulions afficher des informations sommaires. J'ai donc créé un composant AdventureSmall qui permet d'afficher pour une aventure, le nom de son auteur, son synopsis, sa date de création ainsi que le lien pour aller sur la page de cette aventure.

En utilisant les ressources que nous avions mis de côté pour l'inspiration du design du site, j'ai créé un composant Loader qui sert à être affiché lorsque l'utilisateur déclenche une action qui requiert une requête API afin de l'informer que son action a bien été prise en compte et est en attente d'une réponse de l'API.



Dans chacun des composants représentant une page dans lesquels une requête API peut être faite, nous avons fait un affichage conditionnel qui se base sur un booléen “loading”.

Lorsqu'un composant requiert dès son premier affichage une requête API, nous avons utilisé le hook d'effet “useEffect” qui permet d'indiquer à React que nous voulons exécuter quelque chose après le premier

affichage et après chaque modification (il est également possible de paramétrier le useEffect pour que l'exécution ne se fasse que lorsque que des modifications précises du state ont eu lieu).

Prenons l'exemple de la page d'accueil dont je me suis occupé, lors de son affichage nous demandons à React d'exécuter trois fonctions callback appelées “action creators” :

- fetchAdventuresHome, qui fait une requête API pour avoir les trois dernières aventures créées et publiées sur le site. Ces aventures sont utilisées par le composant AdventureSmall pour afficher les informations.
- fetchAdventuresActiveNumber, qui fait une requête API pour avoir le nombre d'aventures publiées sur le site, donc jouables. Cette information est utilisée dans le texte de présentation du site.
- displayLoader, qui change le booléen “loading” en true afin d'afficher le Loader.



```
1  useEffect(() => {  
2    fetchAdventuresHome();  
3    fetchAdventuresActiveNumber();  
4    displayLoader();  
5  }, []);
```

```

1 import { connect } from 'react-redux';
2
3 import {
4   fetchAdventuresHome,
5   fetchAdventuresActiveNumber,
6 } from 'src/actions/adventures';
7 import { displayLoader } from 'src/actions/utils';
8
9 import Home from 'src/components/Home';
10
11 // === mapStateToProps
12 const mapStateToProps = (state) => ({
13   adventuresActiveNumber: state.adventures.adventuresActiveNumber,
14   adventuresHome: state.adventures.adventuresHome,
15   loading: state.utils.loading,
16   isLoggedIn: state.user.isLoggedIn,
17 });
18
19 // === mapDispatchToProps
20 const mapDispatchToProps = (dispatch) => ({
21   // Loader
22   displayLoader: () => {
23     dispatch(displayLoader());
24   },
25
26   // Adventures
27   fetchAdventuresHome: () => {
28     dispatch(fetchAdventuresHome());
29   },
30   fetchAdventuresActiveNumber: () => {
31     dispatch(fetchAdventuresActiveNumber());
32   },
33 });
34
35 export default connect(mapStateToProps, mapDispatchToProps)(Home);

```

Le container du composant Home (la page d'accueil) contient dans “mapDispatchToProps” les actions creators qui sont utilisées dans le UseEffect.

Une fois les informations reçues, le composant Home aura l'accès à leur “lecture” depuis le state car nous les avons mises dans le “mapStateToProps”.

fetchAdventuresHome et fectchAdventuresActiveNumber permettent d'envoyer (dispatch) les actions du même nom.

```
1 export const fetchAdventuresActiveNumber = () => ({
2   type: FETCH_ADVENTURES_ACTIVE_NUMBER,
3 });
4
5 export const fetchAdventuresHome = () => ({
6   type: FETCH_ADVENTURES_HOME,
7 });
```

```
1 // == Loader
2 export const displayLoader = () => ({
3   type: DISPLAY_LOADER,
4 });
5
6 export const hideLoader = () => ({
7   type: HIDE_LOADER,
8 });
```

Les actions creators ont un type unique qui permet aux middlewares et reducers de "reconnaître" ces actions et peuvent avoir des arguments lorsque d'autres informations doivent être transmises comme nous le verrons plus bas.

```

1 const adventuresMiddleware = (store) => (next) => (action) => {
2   switch (action.type) {
3     case FETCH_ADVENTURES_HOME:
4       // API request for the last three adventures
5       axios.get(`${baseURL}/api/v0/stories?last=3`)
6         .then((response) => {
7           // dispatch to save the Adventures used in Home
8           store.dispatch(saveAdventuresHome(response.data[0]));
9           // dispatch to hide the loader
10          store.dispatch(hideLoader());
11        })
12        .catch((error) => {
13          console.warn(error);
14        });
15
16        next(action);
17        break;
18
19     case FETCH_ADVENTURES_ACTIVE_NUMBER:
20       // API request for the number of active adventures
21       axios.get(`${baseURL}/api/v0/stories/count`)
22         .then((response) => {
23           // dispatch to save the Adventure selected
24           store.dispatch(saveAdventuresActiveNumber(response.data.storyNumber));
25           // dispatch to hide the loader
26           store.dispatch(hideLoader());
27         })
28         .catch((error) => {
29           console.warn(error);
30         });
31        next(action);
32        break;

```

Dans le middleware dédié aux Aventures, nous pouvons voir les deux cas qui nous concerne.

Les deux actions font une requête à notre API et lorsque la réponse est reçue, elles dispatch une nouvelle action qui permet de sauvegarder les informations dans le state ainsi que l'action qui permet de “désactiver” le loader.

```

1 export const saveAdventuresHome = (adventuresHome) => ({
2   type: SAVE_ADVENTURES_HOME,
3   adventuresHome,
4 });
5
6 export const saveAdventuresActiveNumber = (adventuresActiveNumber) => ({
7   type: SAVE_ADVENTURES_ACTIVE_NUMBER,
8   adventuresActiveNumber,
9 });

```

Les actions ont cette fois des arguments, appelés “payload” qui sont les réponses de nos requêtes API et vont être utilisés pour mettre à jour notre state dans le reducer dédié aux aventures.



```
1 import {
2   SAVE_ADVENTURES_HOME,
3   SAVE_ADVENTURES_CATALOG,
4   SAVE_ADVENTURE_SELECTED,
5   SAVE_ADVENTURES_ACTIVE_NUMBER,
6   SAVE_ADVENTURE_TIMER,
7   CLEAR_ADVENTURE_TIMER,
8 } from 'src/actions/adventures';
9
10 const initialState = {
11   adventuresHome: [],
12   adventuresCatalog: [],
13   adventureSelected: {
14     id: '',
15     title: '',
16     description: '',
17     createdAt: '',
18     author: {
19       username: '',
20       id: 0,
21     },
22     active: false,
23     slug: '',
24   },
25   adventuresActiveNumber: 0,
26   adventureTimer: {
27     average: '',
28     best: '',
29     number: '',
30   },
31 };
32
33 const adventures = (state = initialState, action = {}) => {
34   switch (action.type) {
35     case SAVE_ADVENTURES_HOME:
36
37       return {
38         ...state,
39         adventuresHome: action.adventuresHome,
40       };
41
42     case SAVE_ADVENTURES_ACTIVE_NUMBER:
43       return {
44         ...state,
45         adventuresActiveNumber: action.adventuresActiveNumber,
46       };
47
48     default: return state;
49   }
50 };
51
52 export default adventures;
```

(J'ai retiré les autres "case" qui ne sont pas utiles pour cet exemple afin de gagner de la place sur l'image.)

On peut voir ici que nos deux actions sont bien des “case” de notre reducer et vont donc être exécutées.

Les deux “case” fonctionnent de la même façon, le spread operator va “déverser” l'état du state actuel puis va changer la valeur de la propriété par celle du même nom dans le payload de notre action puis retourner ce nouveau state.

Une fois le nouveau state sauvegardé, notre composant ayant accès à ces informations va refaire un affichage avec les nouvelles valeurs.

La const “initialState” sert à avoir des valeurs par défaut dans notre state ainsi que définir la “forme” initiale qu'il aura.

Pour le Loader, c'est beaucoup plus simple, lors du useEffect, l'action displayLoader est dispatch et l'action est exécutée dans le reducer Utils.

Lorsque nos requêtes API reçoivent une réponse, l'action hideLoader est dispatch et est exécutée dans le reducer Utils. Nous n'avons pas besoin de payload, nous voulons juste changer la valeur d'un booléen.

Voici le reducer Utils dans son entièreté, on y trouve toutes les actions et changements de state qui sont de simples changements de valeur de booléen et ne rentrent pas dans un autre reducer qui a un rôle précis comme celui des Aventures.

On y voit également les “case” pour le BurgerMenu dont j'ai parlés dans l'explication du Sprint 0.



```
1 import {
2   TOOGLE_BURGER_MENU,
3   TOOGLE_BURGER_MENU_FROM_NAV,
4   DISPLAY_LOADER,
5   HIDE_LOADER,
6   REDIRECT_ON,
7   REDIRECT_OFF,
8   TOGGLE_ASIDE_OPEN,
9 } from 'src/actions/utils';
10
11 const initialState = {
12   burgerMenuOpen: false,
13   loading: true,
14   asideOpen: false,
15   redirect: false,
16 };
17
18 const utils = (state = initialState, action = {}) => {
19   switch (action.type) {
20     case TOOGLE_BURGER_MENU:
21       // return a new state
22       return {
23         ...state,
24         // reverse the value of burgerMenuOpen
25         burgerMenuOpen: !state.burgerMenuOpen,
26       };
27
28     case TOOGLE_BURGER_MENU_FROM_NAV:
29       // return a new state
30       return {
31         ...state,
32         // reverse the value of burgerMenuOpen
33         burgerMenuOpen: false,
34       };
35   }
36 }
```



```
1  case DISPLAY_LOADER:
2      return {
3          ...state,
4          // change loading to false to hide the loader
5          loading: true,
6      };
7
8  case HIDE_LOADER:
9      return {
10         ...state,
11         // change loading to false to hide the loader
12         loading: false,
13     };
14
15 case REDIRECT_ON:
16     return {
17         ...state,
18         redirect: true,
19     };
20
21 case REDIRECT_OFF:
22     return {
23         ...state,
24         redirect: false,
25     };
26
27 case TOGGLE_ASIDE_OPEN:
28     return {
29         ...state,
30         asideOpen: !state.asideOpen,
31     };
32
33 default: return state;
34 }
35 };
36
37 export default utils;
```

J'ai également ajouté la bibliothèque Moment.js pour la gestion et manipulation des dates et j'ai commencé le composant StoryCreate avec Hugo. Ce composant est la page de création d'une nouvelle aventure.

6c - Sprint 2 :

Pour ce sprint, je me suis entièrement consacré à l'une de nos principales fonctionnalités, à savoir la création et la modification d'une aventure.

Le reste de l'équipe front s'est occupé de la fonctionnalité de jeu et de tout ce qui était en rapport avec l'utilisateur, à savoir l'inscription, la connexion et la page de profil.

Nous nous sommes entraînés quand cela était nécessaire et l'équipe back est également venue en renfort car nous étions en retard de notre côté.

Je me suis rendu compte que nous allions avoir beaucoup d'input, j'ai donc décidé de commencer par la création de deux composants réutilisables : Field et FieldArea.

Le composant Field est un simple input avec un label qui nécessite de lui passer les propriétés (props) nécessaires à son utilisation tel que name, value, type mais surtout un identifier qui permet d'identifier l'input qui est utilisé et ainsi de n'avoir qu'un action creator par composant l'utilisant.

FieldArea fonctionne de la même façon mais est un textarea au lieu d'un input pour quand cela est nécessaire.

Pour la création et la modification d'une aventure dans tous ses aspects, je suis passé par plusieurs étapes et plusieurs façons d'agencer mon code. La version finale est la suivante :

- Un composant StoryCreate qui est la page de création d'une aventure.
- Un composant Adventure qui est la page d'une aventure qui contient son titre, sa description et toutes les autres informations disponibles selon le rôle de l'utilisateur.
- Un composant StoryEdit qui est la page de modification d'une aventure et qui contient entre autres deux composants importants :
 - Un composant AdventureEdit qui permet de modifier les informations du composant Adventure.
 - Un composant ChapterEdit qui permet la création d'un chapitre pour l'aventure et la modification des chapitres déjà existants.

Ces pages et leurs sous composants ne sont accessibles que si l'utilisateur est inscrit et connecté et dans le cas d'une modification que s'il en est l'auteur.

Je vais me concentrer sur StoryEdit pour les explications détaillées car cela a été le plus compliqué et a nécessité la création d'un autre composant lors du Sprint 3, Architecture.

Lorsque l'on arrive sur la page de StoryEdit, le hook d'effet useEffect déclenche l'action creator "fetchAdvEditSelected" qui va être un "case" de notre middleware, le fonctionnement est le même que celui décrit dans le Sprint 1.



```
1  case FETCH_ADV_EDIT_SELECTED: {
2    const { apiToken } = store.getState().user.user;
3    axios.get(`#${baseURL}/api/v0/stories/${action.slug}`, {
4      headers: { 'X-AUTH-TOKEN': apiToken },
5    })
6      .then((response) => {
7        // dispatch to save the Adventure to edit selected
8        store.dispatch(saveAdventureEditSelected(response.data[0]));
9        if (response.data[0].firstChapter !== null) {
10          store.dispatch(fetchAdvEditChapters());
11        }
12      })
13      .catch((error) => {
14        console.warn(error);
15      }).finally(() => {
16        store.dispatch(hideLoader());
17      });
18    next(action);
19    break;
20  }
```

Ici, notre “case” va, dans un premier temps, chercher et stocker l’apiToken de l’utilisateur connecté qui a été mis dans le state lors sa connexion et qui lui avait été assigné lors de la création de son compte.

Ensuite, en utilisant Axios, nous faisons une requête à notre API afin d’avoir les informations de l’aventure en lui fournissant l’apiToken.

Une fois la réponse reçue, nous utilisons l’action creator “saveAdventureEditSelected” pour enregistrer les informations dans le state et si cette aventure a un premier chapitre existant, nous utilisons l’action creator “fetchAdvEditChapters” afin de faire une requête API pour voir les chapitres existants.

Le fonctionnement est le même, cette fois nous cherchons l’id de l’aventure que nous venons d’enregistrer dans le state puis nous faisons une requête pour avoir les chapitres de l’aventure.

Une fois les chapitres reçus, on les enregistre dans le state et on utilise l’action creator qui fait la requête nécessaire pour avoir les chapitres qui n’ont pas de parent et une fois reçu, on enregistre l’information dans le state.

```

1   case FETCH_ADV_EDIT_CHAPTERS: {
2     const { idStory } = store.getState().storyEdit;
3     axios.get(`${baseURL}/api/v0/chapters?story_id=${idStory}`)
4       .then((response) => {
5         store.dispatch(displayLoader());
6         store.dispatch(saveAdvEditChapters(response.data[0]));
7         store.dispatch(fetchParentChapterPossibleOptions());
8       })
9       .catch(() => {
10       store.dispatch(clearStoryEdit());
11     });
12     next(action);
13     break;
14   }
15
16 case FETCH_PARENT CHAPTER_POSSIBLE_OPTIONS: {
17   const { idStory } = store.getState().storyEdit;
18   axios.get(`${baseURL}/api/v0/chapters/?story_id=${idStory}&non_parent=true`)
19     .then((response) => {
20       store.dispatch(saveParentChapterPossibleOptions(response.data[0]));
21     })
22     .catch(() => {
23
24     })
25     .finally(() => {
26       store.dispatch(hideLoader());
27     });
28     next(action);
29     break;
30   }

```

Je vais profiter de cet exemple et de la première requête pour expliquer le fonctionnement côté back.

Lors de notre “case” “FETCH_ADV_EDIT_SELECTED”, la requête API utilise la route qui se trouve dans notre StoryController.

StoryController :

```

1 /**
2  * Return one story with slug parameter
3  *
4  * @Route("/api/v0/stories/{slug}" , name="api_v0_stories_slug", methods={"GET"})
5  */
6 public function showBySlug (Story $story , ObjectNormalizer $normalizer)
7 {
8     $this->denyAccessUnlessGranted( 'showBySlug', $story);
9
10    $serializer = new Serializer([new DateTimeNormalizer(), $normalizer]);
11
12    $normalizedStory = $serializer->normalize($story, null, ['groups' => 'api_story_detail']);
13
14    //Return all stories
15    return $this->json([
16        $normalizedStory
17    ]);
18 }

```

Cette fonction, showBySlug, a dans sa route le “slug” ce qui permet, en lui donnant Story en argument, de retrouver grâce au ParamConverter la Story qui contient ce slug. Story étant une Entity qui est une représentation de notre table Story de la base de données sous la forme d'un objet, elle contient donc la propriété (le champ) slug. Si une Story ayant ce slug est bien trouvée, elle est mise dans la variable \$story.

La ligne 8 vérifie que l'utilisateur a bien accès à cette \$story grâce au Voter. Les Voters servent à vérifier les permissions de l'utilisateur. La fonction denyAccessUnlessGranted utilise nos Voters si il y en a.

Le Voter exécute la fonction “supports” en premier lieu pour vérifier qu'il doit bien être utilisé. Si “supports” retourne true alors la fonction “voteOnAttribute” est exécutée.

StoryVoter :

```
1 class StoryVoter extends Voter
2 {
3     protected function supports($attribute, $subject)
4     {
5         // replace with your own logic
6         // https://symfony.com/doc/current/security/voters.html
7         return in_array($attribute, ['edit', 'add', 'delete', 'active', 'showBySlug'])
8             && $subject instanceof \App\Entity\Story;
9     }
10
11    protected function voteOnAttribute($attribute, $subject, TokenInterface $token)
12    {
13        $user = $token->getUser();
14
15        // ... (check conditions and return true to grant permission) ...
16        switch ($attribute) {
17
18            case 'showBySlug':
19
20                // if the story is active or if connected user is author
21                if($subject->getActive() || $subject->getAuthor()->getId() === $user->getId())
22                {
23                    return true;
24                }
25
26                break;
27        }
28
29        return false;
30    }
31 }
```

Dans la fonction supports qui a reçu nos deux arguments showBySlug (\$attribute) et \$story (\$subject), on vérifie que \$attribute fait bien partie des cas de figure que l'on a définis et \$subject aussi.

Si les deux sont bons, on retourne donc true et true et on passe à la fonction suivante voteOnAttribute.

On commence par stocker dans \$user l'utilisateur en se servant du \$token pour le retrouver ensuite on regarde si un cas de figure correspond (j'ai retiré les cas qui ne nous concernent pas ici).

Dans le cas du showBySlug, on vérifie si la Story est active ou si l'id de l'auteur de cette Story est bien l'id de l'utilisateur qu'on a retrouvé via le token. Si c'est bien le cas, on retourne true et on peut donc continuer dans notre fonction showBySlug du StoryController.

A la ligne 10, on instancie un nouveau Serializer qui nous servira à transformer notre objet en format JSON.

A la ligne 12, on demande à ce nouveau Serializer de prendre les propriétés de notre \$story et leurs valeurs mais seulement celles qui sont dans le “groups” “api_story_detail”.

Les “groups” sont définis dans nos Entity au-dessus de chaque propriété.

Exemple de propriété de l'Entity Story :

```
1  /**
2   * @ORM\Column(type="string", length=255, nullable=true, unique=true)
3   * @Groups("api_story_detail")
4   * @Groups("api_party_detail")
5   */
6  private $slug;
7
8  /**
9   * @ORM\OneToOne(targetEntity=Chapter::class, cascade={"persist", "remove"})
10  * @JoinColumn(name="chapter_id", referencedColumnName="id", onDelete="CASCADE")
11  * @Groups("api_story_detail")
12  */
13 private $firstChapter;
14
```

On peut voir que \$slug et \$firstChapter sont tous les deux dans le groupe “api_story_detail” mais seulement \$slug est dans le groupe “api_party_detail”.

Puis la fonction retourne les informations au format JSON, ce qui est donc la réponse de notre requête API FETCH_ADV_EDIT_SELECTED.

Reprendons donc le côté front, une fois que toutes les requêtes initiales pour StoryEdit ont été faites, que les informations ont été enregistrées dans le state et que le Loader est désactivé, l'affichage est enfin disponible.

Sur la page est présent un menu select.



```
1 <label htmlFor="storyEdit-form-editChoice">
2   Editer :
3   <select
4     className="storyEdit-form-editChoice"
5     id="storyEdit-form-editChoice"
6     onChange={handleEditOption}
7     defaultValue=""
8   >
9     <option disabled value="">
10    Choix
11    </option>
12    <option value="Nouveau Chapitre">
13      Nouveau Chapitre
14    </option>
15    <option value={initialTitle}>
16      Aventure: {initialTitle}
17    </option>
18    {chapters.map((chapter) => (
19      <option
20        key={chapter.id}
21        value={chapter.id}
22      >
23        Chapitre: {chapter.title}
24      </option>
25    )))
26  </select>
27 </label>
```

Depuis ce menu, il est possible de :

- Créer un nouveau chapitre.
- Modifier l'aventure directement (les informations visibles sur le composant AdventureSmall et celles visibles sur la page de l'aventure).
- Modifier les chapitres existants.

En fonction du choix, la value est enregistrée dans le state (event -> action creator -> reducer) et un affichage conditionnel permet de voir le composant adéquat.

```
● ● ●
1  feditOption === initialTitle && (
2    <form
3      className="advEdit-form"
4      onSubmit={handleAdvEditSubmit}
5    >
6      <AdventureEdit />
7      <button type="submit" className="storyEdit-Button">Enregistrer les modifications sur l'aventure</button>
8    </form>
9  ){}
10
11 <ChapterEdit {...chapterEdit} id={`${chapterEdit.id}`}>
```

Dans le cas de la modification de l'aventure, on affiche le composant AdventureEdit qui contient simplement des Field et FieldArea avec les informations déjà pré-remplies car nous les avons dans le state.

Dans le cas de la création ou la modification d'un chapitre, on affiche le composant ChapterEdit qui comme AdventureEdit a une condition pour son affichage mais directement dans le composant cette fois.

La gestion du choix est faite via l'événement onChange sur le menu select.

```
● ● ●
1  const handleEditOption = (event) => {
2    setEditOption(event.target.value);
3    // Reset error display
4    setValidationAdvEditFalse();
5    setValidationChapEditFalse();
6    // Condition to not do the get request if it's a new chapter or the adventure
7    // And clear the state of chapterEdit
8    if (event.target.value === 'Nouveau Chapitre' || event.target.value === initialTitle || event.target.value
9     === '') {
10      clearChapterEdit();
11    }
12    else {
13      fetchChapterEditSelected(event.target.value);
14    }
15  };
16
```

On commence par enregistrer le choix dans le state puis on vérifie le choix, si c'est un nouveau chapitre, l'aventure ou une valeur vide, on nettoie le state en retirant les informations à la référence ChapterEdit.

Si c'est un autre choix, donc un chapitre déjà existant, on fait une requête API pour avoir les informations.

En utilisant la valeur de editOption du state, on peut donc comme pour la modification d'une aventure plus haut, faire un affichage conditionnel pour le composant ChapterEdit en le comparant à l'id que l'on lui passe en props ou à la valeur "Nouveau Chapitre".

On déverse également les informations du chapitre que l'on veut modifier (dans le cas d'une modification) dans le composant ChapterEdit afin que les Field et FieldArea soient déjà pré-remplis.

Prenons le cas de la modification d'un chapitre déjà existant :

```
● ● ●

1 {editOption === id && (
2   <form
3     className="newChapter-form"
4     onSubmit={handleChapterEditSubmit}
5   >
6     <div className="chapterEdit">
7       {firstChapter.id !== parseInt(editOption, 10) &&
8         {chapterEdit.parentChapter !== null &&
9           <span>Chapitre parent actuel : {chapterEdit.parentChapter.title}<br /></span>
10        )}
11       Choisir le Chapitre parent :
12       <select
13         className="chapterEdit-parentChapter"
14         id="chapterEdit-parentChapter"
15         onChange={handleParentChapterChoice}
16         defaultValue={chapterEdit.parentChapter !== null && chapterEdit.parentChapter.id}
17       >
18         <option
19           value={chapterEdit.parentChapter !== null && chapterEdit.parentChapter.id}
20         >
21           Chapitres parent possible
22         </option>
23         <option value="">
24           Retirer le choix actuel
25         </option>
26         {parentChapterOption.map((chapter) => {
27           // Condition to prevent the option to select a chapter as his own parent
28           if (chapter.id !== parseInt(editOption, 10)) {
29             return (
30               <option
31                 key={chapter.id}
32                 value={chapter.id}
33               >
34                 {chapter.title}
35               </option>
36             );
37           }
38         })}
39       </select>
40     </label>
41   )
42   <Field
43     className="chapterEdit-chapterTitle"
44     identifier="title"
45     placeholder="Le coffre"
46     value={title}
47     changeField={updateField}
48     label="Titre :"
49   />
50   <FieldArea
51     className="chapter>Edit-chapterText"
52     identifier="content"
53     placeholder="Vous voyez devant vous un bureau sur lequel est posé un coffre fermé par un cadenas, ainsi
54     qu'une clé accrochée à un clou sur le mur"
55     value={content}
56     changeField={updateField}
57     label="Contenu :"
58   />
59   ...
60   ... et cetera
```

Si l'option choisie est bien celle de l'id du Chapitre, le form est disponible et les champs sont pré-remplis.

Si le chapitre est le premier de l'aventure, toute la partie pour le choix du chapitre parent ne s'affiche pas (ligne 7 à 42).

Si ce n'est pas le premier chapitre alors un menu select est disponible afin de choisir un chapitre parent.

Si le chapitre a déjà un parent, il est possible de lui retirer son parent (ligne 24) et le chapitre parent actuel est visible (ligne 9).

Si des chapitres n'ont pas de parent et sont donc disponibles, une option est créée pour chacun d'eux sauf pour le chapitre qui est en train d'être modifié pour empêcher qu'il puisse être son propre parent (ligne 27).

Prenons comme exemple la modification d'un chapitre pour voir une requête API dans l'autre sens cette fois.

Lorsque le form est submit, la fonction handleChapterEditSubmit est exécutée.

```
● ● ●
```

```
1 const handleChapterEditSubmit = (event) => {
2   event.preventDefault();
3   // Reset error display
4   setValidationErrorChapEditFalse();
5   // Check the length of title and content
6   if ((title.length > 3) && (content.length) > 50 && (unlockText.length) > 0) {
7     // Check if keyword and lockword are in the content
8     if (checkWordInContent(keyword, content) && checkWordInContent(lockword, content)) {
9       submitChapterEditForm();
10    }
11    else {
12      setErrorKeyLockTrue();
13    }
14  }
15  else {
16    setValidationErrorChapEditTrue();
17  }
18};
```

On commence par empêcher le comportement par défaut pour gérer nous même l'envoi.

On vérifie si les informations saisies sont bien conformes à ce qui est demandé.

On exécute les fonctions qui permettent de vérifier que les mots clé et serrure sont bien dans le texte du chapitre afin qu'il soit vraiment jouable. Si c'est bien le cas, on envoie notre form grâce à submitChapterEditForm, sinon on affiche des messages d'erreurs appropriés.

Comme à chaque fois, le container de notre composant nous permet de déclencher l'action creator que l'on souhaite, ici submitChapterEditForm qui va passer par nos middlewares ou un "case" a été créé pour lui.

Je commence par aller chercher dans le state, les informations dont j'ai besoin à propos de l'aventure et du chapitre que je souhaite modifier.

Je vais également chercher l'apiToken de l'utilisateur.

J'utilise ensuite Axios pour faire une requête PUT en fournissant toutes les informations nécessaires y compris un header contenant mon apiToken puis je redirige l'utilisateur sur la page de l'aventure.

```
1 case SUBMIT CHAPTER EDIT FORM: {
2   const {
3     id,
4     title,
5     content,
6     keyword,
7     lockword,
8     unlockText,
9     parentChapterChoice,
10    } = store.getState().chapterEdit;
11  const {
12    idStory,
13  } = store.getState().storyEdit;
14  const {
15    apiToken,
16  } = store.getState().user.user;
17  axios.put(`$baseURL}/api/v0/chapters/${id}`, {
18    title,
19    content,
20    keyword,
21    lockword,
22    unlockText,
23    forStory: idStory,
24    parentChapter: parentChapterChoice,
25  },
26  {
27    headers: { 'X-AUTH-TOKEN': apiToken },
28  })
29  .then(() => {
30    store.dispatch(redirectOn());
31  })
32  .catch((error) => {
33    console.warn(error);
34  })
35  next(action);
36  break;
37 }
```

Utilisons également cet exemple pour voir le fonctionnement côté back :

```
● ● ●

1  /**
2  * Edit an existing chapter in DB
3  *
4  * @Route("/api/v0/chapters/{id}", name="api_v0_chapters_edit", methods={"PUT"}, requirements={"id": "\d+"})
5  *
6  * @return Chapter
7  */
8 public function edit(Chapter $chapter, Request $request, ObjectNormalizer $normalizer, KeyLockWordChecker
9 $checker)
10 {
11     //check if user is logged and also the author of the related story
12     $this->denyAccessUnlessGranted('edit', $chapter);
13
14     //Create the associating form to send request data in the chapter in parameter
15     //With the csrf option desactivated as we are on an API
16     $form = $this->createForm(ChapterType::class, $chapter, ['csrf_protection' => false]);
17
18     //Extract the json content of the request
19     $jsonText = $request->getContent();
20
21     //Transform this Json in array
22     $jsonArray = json_decode($jsonText, true);
23
24     //We submit this data array to the form
25     $form->submit($jsonArray);
26
27     //Verify if the form is valide
28     if ($form->isValid())
29     {
30         /* **** Lock and key verification start */
31
32         //Get values
33         //Get values
34         $keyWord = $chapter->getKeyword();
35         $lockWord = $chapter->getLockword();
36         $content = $chapter->getContent();
37
38         // Call the service to check if words are in the chapter content
39         if ($checker->checkWords($keyWord, $lockWord, $content) === false)
40         {
41             // If it is not ok, send an error
42             return $this->json(['message' => "keyword and lockword should be in the chapter content"], 400);
43         }
44         /* Lock and key verification end */
45         /* **** */
46
47         //Set an updated date
48         $chapter->setUpdatedAt(new \DateTime());
49
50         //If it is valid, we flush
51         $em = $this->getDoctrine()->getManager();
52         $em->flush();
53
54         //Then, we return 200 HTTP code with the Chapter Object updated
55
56         //Serialize the data
57         $serializer = new Serializer([new DateTimeNormalizer(), $normalizer]);
58
59         $normalizedChapter = $serializer->normalize($chapter, null, ['groups' => 'chapter_details']);
60
61         //And return it
62         return $this->json($normalizedChapter, 200);
63     }
64     //If it is not valid
65     //We display errors
66     //With a 400 Bad request HTTP code
67     return $this->json((string) $form->getErrors(true, false), 400);
68 }
```

On commence encore une fois par une vérification des accès de l'utilisateur (ligne 11) de la même façon que précédemment.

On crée un form suivant le modèle des chapitres (ligne 15) puis après avoir extrait les informations de la requête, on les envoie au form (ligne 18 à 24).

Ensuite, on vérifie que le form soit bien valide avec la fonction de Symfony (ligne 27). Si ce n'est pas le cas, on renvoie un JSON avec le code 400 (ligne 67). Si c'est le cas, on vérifie si les mots clé et serrure sont bien dans le texte en utilisant une fonction écrite par l'équipe back (ligne 34 à 43). Si ce n'est pas le cas, on renvoie un JSON avec un message et le code 400 (code HTTP pour indiquer que la requête n'a pas été comprise). Si c'est bon, on modifie la date de mise à jour pour le chapitre (ligne 48) et on utilise les fonctions de Doctrine afin d'envoyer nos modifications à la base de données. Puis, on renvoie en JSON les informations du chapitre avec un code 200 (code HTTP pour indiquer la réussite) (ligne 57 à 62). Le renvoi des informations n'est pas nécessaire pour le front mais peut être utile lors du développement.

6d - Sprint 3 :

Lors de ce sprint, nous avions comme directive de ne pas coder de nouvelles fonctionnalités majeures.

J'ai commencé par ajouter des actions creator partout où cela était nécessaire pour "nettoyer" le state, afin d'éviter qu'un utilisateur, qui n'a pas validé l'envoi d'un formulaire ou qui a changé l'option d'un menu select, retrouve les informations qu'il avait entrées en revenant sur la page/le choix.

J'ai créé une fonction qui permet de vérifier que les mots clé et serrure sont bien dans le texte d'un chapitre.



```
1 // Check if keyword and lockword are in the content
2 export const checkWordInContent = (word, content) => {
3   const regex = /[.,;!?"'*\/']+/gi;
4
5   // Content
6   const contentLowerCase = content.toLowerCase();
7   const contentReplace = contentLowerCase.replace(regex, ' ');
8   const contentTrim = contentReplace.trim();
9   const contentSplit = contentTrim.split(' ');
10
11  // Word
12  const wordTrim = word.trim();
13  const wordLowerCase = wordTrim.toLowerCase();
14
15  // Check
16  const checked = contentSplit.includes(wordLowerCase);
17
18  return checked;
19};
```

J'ai ajouté des placeholder où cela manquait, j'ai réglé quelques soucis de CSS.
J'ai ajouté une animation sur la page de jeu pour que la "led" clignote.

Puis, après avoir discuté avec l'équipe, nous avons conclu que le site n'était pas très user-friendly et qu'il manquait des indications importantes, nous avons donc décidé d'ajouter des petites fonctionnalités.

La première, un "Aside" qui affichait des explications sur les pages de création et de modification d'une aventure ainsi que sur la page de jeu.

La deuxième, une arborescence, une "Architecture", un sommaire des chapitres sur la page d'édition afin de voir facilement les chapitres ayant des parents et l'ordre des chapitres.

Pour le Aside, j'ai commencé par créer un objet "asideData" dans mon fichier Utils. Dans cet objet, j'ai créé trois autres objets, un pour chaque page où l'Aside serait utilisé avec un titre et une description. Les descriptions ont été écrites avec toute l'équipe.

J'ai ensuite créé un composant Aside que j'ai importé et utilisé dans le composant principal App (visible également sur le screenshot des routes du Sprint 0) sur les routes contenant les pages nécessitant un Aside. J'ai importé asideData dans mon composant App et passé les

informations nécessaires au composant Aside.



```
1 <Route exact path="/aventures/creation">
2   <Aside data={asideData.storyCreate} />
3   {isLoggedIn !== true && (<Redirect to="/connexion" />)}
4   {isLoggedIn === true && (<StoryCreate />)}
5 </Route>
6 <Route exact path="/aventures/:slug/jouer">
7   <Aside data={asideData.gameScreen} />
8   {isLoggedIn !== true && (<Redirect to="/connexion" />)}
9   {isLoggedIn === true && (<GameScreen />)}
10 </Route>
11 <Route exact path="/aventures/:slug/edition">
12   <Aside data={asideData.storyEdit} />
13   {isLoggedIn !== true && (<Redirect to="/connexion" />)}
14   {isLoggedIn === true && (<StoryEdit />)}
15 </Route>
```

Le composant Aside :



```
1 const Aside = ({ toggleAsideOpen, asideOpen, data }) => (
2   <aside className="aside">
3     <div className="aside-icon" onClick={toggleAsideOpen}>
4       {asideOpen && (<XCircle size={30} />)}
5       {!asideOpen && (<Info size={30} />)}
6     </div>
7     {asideOpen && (
8       <div className="aside-content">
9         <p className="aside-content-title">
10           {data.title}
11         </p>
12         <p className="aside-content-text">
13           {data.content}
14         </p>
15       </div>
16     )}
17   </aside>
18 );
```

L'événement onClick sur un icône permet d'activer ou désactiver l'affichage des informations et l'icône change en fonction.

L'utilisation du z-index dans le CSS permet d'avoir le composant Aside toujours visible "au dessus" du composant de la page.

Pour le composant Architecture, cela a été un peu plus compliqué. Comme énoncé dans le Sprint 2, j'utilise Architecture dans le composant StoryEdit.



```
1 /* display the adventure architecture if there is one */
2 {storyEdit.chapters.length > 0 && (
3   <>
4     <Architecture
5       titleStory={storyEdit.title}
6       firstChapter={storyEdit.firstChapter.title}
7       firstChapterId={storyEdit.firstChapter.id}
8       chapters={storyEdit.chapters}
9     />
10    </>
11  )}
```

L'affichage d'Architecture ne se fait que si l'aventure a des chapitres, car Architecture sert à visualiser le sommaire de l'aventure ce qui n'est donc pas nécessaire s'il n'y a pas de chapitre. Je passe toutes les props nécessaires au fonctionnement d'Architecture qui sont déjà accessibles dans StoryEdit. La seule information qui est gérée par Architecture et son container est le booléen "structureView" et ses actions creator qui sont utilisés pour gérer l'ouverture et la fermeture du sommaire.

La première chose que j'ai du faire dans le composant Architecture était d'organiser les informations reçues par StoryEdit car les chapitres ne sont pas "rangés" selon leur ordre de lecture lors d'une partie.

```

1 // Array to put the chapter in
2 chaptersDisplay = [];
3 // Start with the firstChapter
4 previousChapterMapped[0] = firstChapterId;
5 // Loop for every chapters
6 for (let i = 0, indexChapters = 1; i < chapters.length; i++) {
7   // Map to push in an array only the chapter that's the previousChapter's child
8   chapters.map((chapter) => {
9     if (chapter.parentChapter === null) {
10       if (chapter.parentChapter.id === previousChapterMapped[0]) {
11         indexChapters++;
12         previousChapterMapped[0] = chapter.id;
13         chaptersDisplay.push(
14           <div className="architecture-row" key={chapter.id}>
15             <span>#{indexChapters}: {chapter.title}</span>
16           </div>,
17         );
18       }
19     }
20   });
21 }

```

Je commence par créer un tableau vide (ligne 2) et je mets le premier chapitre que j'ai reçu de StoryEdit dans un tableau previousChapterMapped à l'index 0 (ligne 4).

Puis, je fais une boucle for en créant une variable indexChapters initialement à 1 car elle servira de numérotation du chapitre. La boucle sera itérée un nombre de fois égale au nombre de chapitres présents dans l'aventure en utilisant .length sur le tableau de chapters que j'ai reçu de StoryEdit (ligne 6).

Pour chaque itération de ma boucle, je vais faire un map des chapters, qui permet de créer un nouveau tableau en appliquant des modifications au tableau pour chaque entrée (ligne 8).

La première modification que j'effectue pour chacun des chapitres de mon tableau de chapitres est de regarder si le chapitre en question a un parent. Si ce n'est pas le cas, je ne fais rien d'autre et je passe donc à l'itération suivante de mon map (ligne 9).

Si le chapitre a bien un parent, je compare l'id de son chapitre parent au previousChapterMapped[0] qui est le premier chapitre de mon aventure (ligne 10), ce qui me permet de trouver le deuxième chapitre de mon aventure. Je cherche ici le deuxième chapitre car le premier n'a pas de "parentChapter", c'est également pour cela que indexChapters commence à 1.

Une fois que le deuxième chapitre est trouvé, je passe donc dans ma condition.

J'incrémente l'indexChapters qui passe donc à 2.

Je remplace le previousChapterMapped[0] par l'id du deuxième chapitre ce qui permettra à la prochaine itération de trouver le troisième chapitre lors de la condition ligne 10.

Je pousse ensuite dans mon tableau chaptersDisplay le JSX avec les informations du chapitre que je pourrais utiliser directement lors de l'affichage.

Comme je ne passe dans cette condition (ligne 10) qu'une fois, je passe ensuite à une autre itération de ma boucle for (ligne 6) ce qui me permet de faire cette manipulation pour tous les chapitres et ainsi de les ranger dans l'ordre dans mon tableau chaptersDisplay.

Maintenant que nous avons tous nos chapitres dans l'ordre, regardons comment les afficher.

```
● ● ●

1 <section className="architecture">
2   {/* Adventure */}
3   {!structureView && <span className="architecture-link" onClick={showStoryStructure}>Sommaire&ampnbsp<PlusSquare
/></span>}
4   {structureView && <span className="architecture-link" onClick={hideStoryStructure}>Sommaire&ampnbsp<MinusSquare
/></span>}
5   {structureView && (
6     <div className="architecture-details">
7       {firstChapter !== null && (
8         <div className="architecture-row">
9           <span>#1: {firstChapter}</span>
10        </div>
11      )}
12
13     {/* Chapter with a parent */}
14     {chaptersDisplay}
15     {/* Display the chapters without parent category only when we have some */}
16     {(chapters.length - chaptersDisplay.length) > 1 && <div className="architecture-subtitle">Chapitres sans
parents</div>}
17     {/* Chapter without a parent */}
18     {chapters.map((chapter) => {
19       if ((chapter.parentChapter == null) && (chapter.id !== firstChapterId)) {
20         return (
21           <div key={chapter.id}>
22             <span>-- {chapter.title}</span>
23           </div>
24         );
25       }
26     })}
27   </div>
28 })
29 </section>
```

L'affichage du sommaire est soumis à la condition que l'utilisateur clique sur un "bouton". On vérifie que l'aventure possède bien un premier chapitre (ligne 7) et si c'est le cas, on l'affiche directement car nous savons déjà qu'il est le #1 et nous l'avons reçu séparément par StoryEdit (ligne 9).

Puis, pour les chapitres qui ont des parents et que nous avons ordonnés plus haut nous n'avons besoin que de mettre chaptersDisplay (ligne 14).

Enfin pour les autres chapitres existants qui n'ont pas de parent, j'utilise une nouvelle fois un map (ligne 18) et je vérifie si le chapitre n'a bien pas de parent et qu'il n'est pas le premier chapitre car celui-ci n'a jamais de parent (ligne 19). Pour chaque chapitre, je renvoie du JSX avec le titre du chapitre.

6e - Tests :

L'équipe back a réalisé des tests unitaires et fonctionnels sur une branche prévue à cet effet mais n'ayant pas encore vu leurs fonctionnements sur Symfony, je préfère ne pas m'avancer sur leurs explications.

Côté front, nous avons appris lors de la spécialisation React de notre formation, l'utilisation du framework Mocha.js qui permet d'exécuter des tests et celle de la bibliothèque

d'assertion Chai qui permet d'encapsuler la logique à tester.

Nous avons également vu Enzyme qui permet de tester les composants React.

Nous avions initialement prévu de réaliser des tests unitaires, nous avions même ajouté mocha, chai et enzyme à nos dépendances de développement mais n'ayant fait que très peu de tests lors de la spécialisation, nous avions besoin de temps pour revoir leur utilisation et leur syntaxe. Étant en retard à mi parcours, nous avons décidé de ne pas en faire et de se contenter de tests manuels réalisés au fur et à mesure.

Exemple de test manuel :

Lors de la création d'une aventure, je dois remplir un formulaire qui nécessite un nombre minimum de caractères dans les champs :

The screenshot shows a mobile application interface for creating an adventure. At the top, there is a navigation bar with '[Déconnexion]' (Logout) and 'Menu' with a plus sign icon. Below the navigation bar, the title 'Créer une aventure' is displayed. The form consists of three main input fields:

- Titre :** L'éénigme mystérieuse ...
- Synopsis : (50 caractères min.)** Il était une fois, dans une contrée lointaine ...
- Description : (50 caractères min.)** L'histoire commence dans une grange abandonnée ...

At the bottom right of the form area, there is a large button with the text 'C'est parti !' (Let's go!).

Si je remplis mon formulaire avec moins de caractères et que je l'envoie en cliquant sur le bouton "C'est parti !" je dois rester sur cette page et un message d'erreur doit apparaître :

The screenshot shows a mobile application interface with a dark theme. At the top, there are navigation icons: a back arrow, a user profile icon, [Déconnexion] (Logout), Menu, and a plus sign icon. Below the header, the title "Créer une aventure" is displayed. The form fields are as follows:

- Titre :** Un titre de test
- Synopsis : (50 caractères min.)** Un texte de synopsis trop court.
- Description : (50 caractères min.)** Le texte de description lui est suffisamment long car il contient au moins 50 caractères.

A red error message box contains the text: Les informations envoyées ne correspondent pas aux prérequis. At the bottom, a large green button with white text says C'est parti !

Je modifie mon formulaire en respectant cette fois le nombre de caractères :

The screenshot shows a mobile application interface for creating an adventure. At the top, there is a navigation bar with icons for back, forward, and search, followed by '[Déconnexion]' and 'Menu'. A blue circular icon with an 'i' is on the right.

Créer une aventure

Titre :
Un titre de test

Synopsis : (50 caractères min.)
Je modifie donc mon texte de synopsis pour qu'il soit d'au moins 50 caractères.

Description : (50 caractères min.)
Le texte de description lui est suffisamment long car il contient au moins 50 caractères.

Les informations envoyées ne correspondent pas aux prérequis

C'est parti !

Cette fois lorsque j'envoie mon formulaire, qui est maintenant valide, je dois être redirigé vers la page de l'aventure que je viens de créer :



C'est bien le cas, mon test est donc réussi.

7 - Recherches :

Lors du développement du projet, j'ai dû faire beaucoup de recherches seul ou en groupe, que ce soit pour trouver comment résoudre un problème ou trouver la réponse à un objectif.

Pendant le sprint 0, nous avons effectué des recherches dans le but de trouver de l'inspiration et voir ce qui était possible et avons trouvé différents codepen ou articles notamment de css-tricks.

Pour le choix de la police d'écriture, j'ai cherché celles utilisées par les vieilles consoles comme la Comodore ou l'Amstrad ainsi que celles des terminaux et du "blue screen of death".

J'ai également cherché s'il existait des bibliothèques qui pourraient nous aider et nous faire gagner du temps. J'ai trouvé trois bibliothèques pour l'effet dactylographique ainsi qu'un composant minuteur pour React que nous n'avons finalement pas utilisé et que Jukka a codé lui-même.

Pendant les autres sprints, j'ai rencontré divers problèmes que j'ai pu régler généralement en relisant ce que nous avions fait en cours, la documentation appropriée et bien évidemment mon code. Voici quelques exemples de problèmes rencontrés qui ont nécessité des recherches Google.

L'un d'eux était la vérification des titres afin qu'ils ne contiennent pas que des chiffres.

J'ai cherché "js check if only digit in a string" et le premier lien était un stackoverflow pour la même question et la meilleure réponse était simplement "how about : let isnum = `^\\d+$/`.test(val);".

J'ai cherché "js .test" car je n'en avais pas souvenir et j'ai lu la documentation sur MDN.

En voici un extrait :

"The **test()** method executes a search for a match between a regular expression and a specified string. Returns true or false."

Traduction :

"La méthode **test()** exécute une recherche pour une correspondance entre une expression régulière et une chaîne de caractères spécifique. Elle renvoie vrai ou faux."

J'ai donc compris la réponse de stackoverflow et après avoir vérifié avec l'équipe et sur une "cheat sheet" regex ce que l'expression voulait dire j'ai pu l'utiliser dans mon code.



```
1 if ((title.length > 3 && !( /\\d+$/ .test(title)))
```

Un autre problème que j'ai eu était le formatage des dates. N'arrivant pas à faire ce que je voulais, j'ai fini par chercher "is it possible to format date in js" et dans un article de css-tricks et dans des réponses sur stackoverflow, j'ai souvent vu la mention de la bibliothèque Moment.js. Afin de gagner du temps, j'ai décidé de l'utiliser ainsi que la version react qui permet son utilisation en composant.

Lors de la première utilisation, j'ai eu le message d'erreur suivant dans la console du navigateur : "Deprecation warning: moment construction falls back to js Date".

J'ai donc recherché le message d'erreur et en lisant ce que j'ai trouvé, j'ai compris que le problème venait du format de la date, j'ai donc consulté la documentation.

En voici un extrait :

"Moment can parse most standard date formats. Use the parse attribute to tell moment how to parse the given date when non-standard."

Traduction :

"Moment peut analyser la plupart des formats de date standard. Utiliser l'attribut parse pour indiquer à Moment comment analyser la date donnée quand elle n'est pas standard."

J'ai compris que le format de la date que l'on recevait du back n'avait pas un format standard et j'ai donc ajouté l'attribut parse pour indiquer à Moment le format.



```
1 <Moment format="DD/MM/YYYY" parse="YYYY-MM-DD HH:mm">
2   {adventureSelected.createdAt}
3 </Moment>
```

Pour finir, un des problèmes que j'ai eu et cela très tôt dans le projet a été de tout le temps avoir le footer en bas de page.

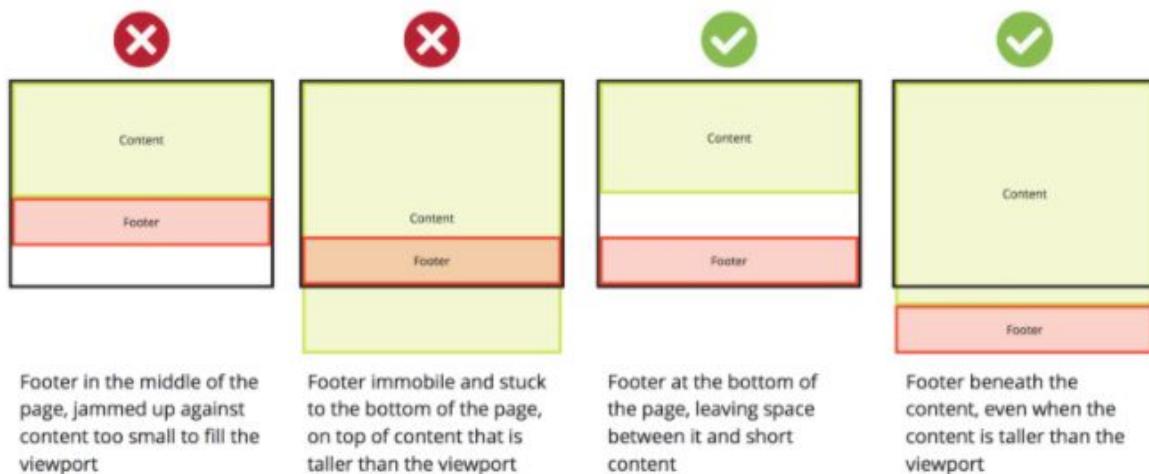
J'ai donc cherché "css footer always bottom" et j'ai trouvé un article nommé "How to keep your footer where it belongs?"

A footer is the last element on the page. At a minimum it is at the bottom of the viewport, or lower if the page content is taller than the viewport. Simple, right??

When working with dynamic content that includes a footer, a problem sometimes occurs where the content on a page is not enough to fill it. The footer, rather than staying at the bottom of the page where we would want it to stay, rises up and leaves a blank space beneath it.

For a quick fix, you can absolutely position the footer at the bottom of the page. But this comes with its own downside. If the content grows larger than the viewport, the footer will remain 'stuck' to the bottom of the viewport, whether we want it to or not.

This shows the behaviour we don't and do want:



Traduction :

Un footer (pied de page) est le dernier élément d'une page. Il est au minimum en bas de la zone d'affichage, ou plus bas si le contenu de la page est plus grand que sa zone d'affichage. Simple n'est-ce pas??

Quand on travaille avec des contenus dynamiques qui incluent un pied de page, un problème peut se produire quand le contenu de la page n'est pas suffisant pour la remplir. Le pied de page, plutôt que de rester en bas de la page où l'on voudrait qu'il reste, remonte et laisse un espace vide en dessous.

Pour une solution rapide, vous pouvez utiliser une position absolue sur le pied de page. Mais cela vient avec son lot d'inconvénients. Si le contenu devient plus grand que la zone d'affichage, le pied de page va rester "bloqué" en bas de la zone d'affichage, qu'on le veuille ou pas.

Cela montre les comportements que l'on veut et ne veut pas:

Pied de page au milieu de la page, coincé contre le contenu trop petit pour remplir la zone d'affichage	Pied de page immobile et bloqué au bas de la page, superposé au contenu qui est trop grand pour la zone d'affichage	Pied de page en bas de la page, laissant un espace entre lui même et le petit contenu	Pied de page en dessous du contenu, même lorsque le contenu est plus grand que la zone d'affichage
---	---	---	--

J'ai donc utilisé les conseils et règles CSS proposés dans la suite de l'article.

J'ai décidé d'utiliser cet exemple pour la traduction requise dans ce dossier. En effet, dans mes notes, je n'avais pas assez précisé les problèmes que j'avais eus qui ont nécessité la documentation de React ou d'une autre bibliothèque. Cela m'avait semblé naturel d'aller lire la doc en premier lieu et je n'ai pas eu l'idée de noter les passages que j'ai lu, contrairement aux fois où j'ai dû faire une recherche Google.

J'ai préféré utiliser ces trois petits exemples car ce sont de réelles recherches que j'ai effectuées et je ne voulais pas choisir un passage aléatoirement de la documentation React par exemple.

8 - Conclusion :

8a - O'Id the door :

Ce mois de travail fut intense mais j'y ai pris du plaisir tout du long, c'était mon premier vrai projet en équipe et une expérience riche en tous points.

Le moment le plus important pour moi a été le sprint 0. Se concentrer sur le cahier des charges et la conception du projet m'a permis de mettre beaucoup de choses en ordre dans ma tête vis-à-vis du code et du fonctionnement de React et Redux. Non seulement parce que j'ai du réfléchir en amont plutôt qu'au fur et à mesure et en prenant tous les aspects du projet, mais aussi parce que l'équipe back n'avait aucune connaissance sur React donc nous avons dû leur expliquer son fonctionnement afin que l'équipe entière soit au courant de tous les aspects du projet. L'inverse s'est également produit et j'ai appris énormément de l'équipe back.

Pour les points négatifs, je n'ai pas assez commenté mon code, sur le moment je trouvais qu'il n'y en avait pas besoin à certains endroits mais revenant sur le projet, il est clair que c'était le cas.

Beaucoup de code demande une réécriture, il y a trop de répétitions et de lignes de code inutiles qui peuvent être optimisées.

L'équipe entière était d'accord pour ne pas toucher au projet jusqu'à ce que tout le monde ait passé le titre professionnel et maintenant que c'est le cas, nous allons pouvoir nous y remettre. J'attends cela avec plaisir car le projet me tient à cœur et il y a encore beaucoup de choses que j'aimerais faire dessus comme les tests unitaires et fonctionnels, retirer des bibliothèques pour améliorer la sécurité en faisant les fonctionnalités moi-même et ajouter toutes celles que nous n'avons pas eues le temps de faire.

8b - Remerciements :

Bien évidemment, je commence par remercier O'clock et toute son équipe (j'ai trop peur d'oublier des noms donc je m'abstiens d'en mettre) chez qui la bonne humeur et l'humour règnent.

La formation a été une magnifique expérience que j'ai pu suivre dans les meilleures conditions possibles malgré ma condition (j'ai un handicap au genou qui m'empêche de me déplacer facilement).

Merci à ma promotion, Excalibur, où l'entraide était le mot d'ordre.

Un grand merci aux ambulancières, un groupe soudé de la promotion avec qui j'échange encore tous les jours.

Merci à toute l'équipe d'O'Id the door sans qui ce projet n'aurait pas pu voir le jour.

Merci à tous mes amis qui m'ont aidé pour la réalisation des dossiers pour le titre professionnel, que ce soit pour de la relecture, des explications, des avis où les projets sur lesquels nous avons travailler ensemble. Fidia EL BOUANANI, Maxence ROYER, Damien TOSCANO, Tomy BLONDEAU, Kérim IDRIS et Farid AIT-GACEM.

Et enfin merci au Fabigeon d'être le majestueux oiseau qu'il est.