



CAHIER DE SPECIFICATIONS TECHNIQUES ET FONCTIONNELLES PRÉSENTE DEVANT LE JURY

POUR L'OBTENTION DU TITRE

DEVELOPPEUR WEB - WEB MOBILE

NIVEAU III (BAC+2)

SENASA Trévor

<https://projet-histoire.herokuapp.com/>

Paris, le XXX



DORANCO ESPACE MULTIMÉDIA

ORGANISME DE FORMATION & ÉTABLISSEMENT D'ENSEIGNEMENT SUPÉRIEUR TECHNIQUE PRIVÉ

10-12 RUE PLANCHAT | 75020 PARIS | TÉL 01 55 25 28 00 | FAX 01 55 25 20 15

look@doranco.fr - www.doranco.fr

Sommaire

Sommaire	2
Remerciements	3
Partie I : Introduction	4
1] Présentation	4
2] "Tutu'Story", un projet de site web d'histoire interactive	4
Partie II : Esquisse du projet	5
1] Personas	5
2] Benchmark concurrentiel	6
3] Use Case	7
4] Diagramme de classe	7
5] Wireframe et maquettes	9
a. Wireframe	9
b. Maquettage	11
c. Choix du style	15
Partie III : Développement du projet	17
1] Langages et framework	17
2] Création du projet et élaboration de la BDD	18
3] Architecture du projet	19
4] Partie Commune	21
a. Le template base.html.twig	21
b. Le template homepage.html.twig	23
c. Inscription et connexion	26
5] Partie Utilisateur	29
a. Structure de story_done	29
b. Page de profil	30
c. Création de personnage	33
d. Affichage des chapitres	36
e. Lecture d'un chapitre	37
f. Validation des choix et des chapitres	42
g. Template d'erreurs	44
6] Partie Administrateur	44
Partie IV : Mise en production et futur	45
1] Mise en production	45
2] Mises à jour futures	45

Remerciements

Je remercie en premier lieu ma famille, pour la patience et le soutien qu'ils m'ont accordés durant cette période de reconversion professionnelle.

Je remercie en outre Pôle emploi ainsi que Doranco qui m'ont fait découvrir certains aspects du monde du développement web qui m'étaient encore inconnus.

Une pensée pour mes collègues de formation qui, je l'espère, continueront d'apprendre et d'enrichir leur savoir dans les années à venir. Antony, pour les longues soirées de tutorat à persévéérer pour comprendre le Javascript, Julien pour le Css chatoyant qui brille de mille feux, Carl et sa volonté inébranlable de se surpasser, Linda pour sa bonne humeur en bouteille et Laetitia avec qui les discussions sur le code étaient toujours un bon moment.

Je remercie en outre Théophile, pour sa confiance pendant le stage à SopraHr Software ainsi que les conseils et le suivi sur mon travail.

Et pour finir, un remerciement pour les conseils avisés de style d'Elise et Soraya lors de la mise en production du site.

Partie I : Introduction

1] Présentation

Depuis ma jeunesse, j'ai toujours été intéressé par le monde informatique sans pour autant savoir son fonctionnement. Il y a peu, j'ai décidé de passer le pas et d'apprendre véritablement à coder en développant davantage mes capacités en algorithmie et logique que j'ai pu acquérir durant mes études.

Ayant réalisé des études en Sciences de la Terre, spécialité hydrogéologie, j'ai dû souvent utiliser des outils et logiciels informatiques. En tant que professeur contractuel en mathématiques, j'ai dû transmettre au mieux mon savoir, tout en accompagnant mes élèves.

Durant cette formation de Développeur Web - Web mobile, j'ai pu concilier mon propre apprentissage avec l'accompagnement d'autres jeunes développeurs.

Au terme de cette formation, j'ai eu la chance d'obtenir un stage à SopraHr Software où j'ai pu développer un composant directement utilisé dans leur projet actuel d'outil RH.

Cependant, étant donné la nature confidentielle de l'ensemble du travail que j'ai eu à produire pour leur compte, c'est un projet personnel dont je discuterai ci-dessous.

2] “Tutu’Story”, un projet de site web d’histoire interactive

Malgré la large place des Sciences en général dans ma vie, développer un intérêt pour la littérature (que ce soit par la lecture ou l'écriture) est tout aussi important à mes yeux.

C'est dans cette optique que m'est venu l'idée de développer un site qui raconterait une histoire. Mais ça ne serait pas seulement une histoire. Ce serait une expérience interactive et évolutive où chaque utilisateur n'aurait pas forcément accès aux mêmes arcs narratifs, en fonction des choix qu'ils feraient.

Chaque utilisateur pourrait créer un personnage (parmi trois) dont la persistance serait acté dans l'univers, tout en lui choisissant un archétype qui pourrait donner quelques variations dans l'histoire.

Chaque personnage disposera de ses chapitres, des accès aux récits déjà lus ainsi que de la liste des tous les choix réalisés depuis de début de l'histoire.

Ces histoires étant écrites par mes soins, elles sont donc ma propriété.

De plus, un indicateur d'alignements sera présent, car les choix impliquent pour la plupart une question morale. Cette moralité sera traduite par un code couleur, qui sera présent sur l'ensemble du site.

Partie II : Esquisse du projet

Comme mentionné plus haut, l'idée de ce site a commencé à germer à mes débuts en tant que développeur. Le concept du site a toujours gardé la même ligne directrice : raconter une histoire tout en laissant des choix possibles dans la trame.

Même si cette idée paraît séduisante de prime abord, elle pouvait aussi devenir effrayante dans l'élaboration future.

Avant de rentrer dans les détails du code, je vais d'abord parler de tout le travail préparatoire réalisé pour la bonne réalisation du projet.

1] Personas

Biographie
Marie-Odile a travaillé en tant qu'infirmière dans l'hôpital public durant de nombreuses années. Une fois à la retraite, elle décide de passer davantage de temps avec sa famille dont ses petits enfants, qui sont adeptes de jeu de rôles. Étant néophyte sur ce domaine, elle souhaite en apprendre davantage pour en faire avec eux et cherche un site sur internet qui pourrait l'aider à comprendre.

Intitulé de poste
Retraitée

Âge
65 ans et plus

Niveau d'études
Baccalauréat ou équivalent

Réseaux sociaux

Secteur d'activité
Santé

Moyen de communication préféré
Téléphone (appel)

Objectifs
Acquérir et comprendre le concept de jeu de rôle
S'initier au jeu de rôle tout en trouvant une histoire qui pourrait l'intéresser

Principaux défis
Comprendre les concepts et ne pas les oublier
Avoir des résumés fréquents sur l'avancée de l'histoire

Illustration 1 : Persona 1 - Marie-Odile de Verrières

Ce premier persona délimite certains points qui devront être appliqués ultérieurement :

- Des notions simples de choix, personnages (côté utilisateur)
- Avoir des résumés des choix passés, pouvoir relire chaque chapitre au besoin.

	<p>Biographie</p> <p>Bobby est un jeune étudiant en informatique et adepte des RPG. Il aime ce genre de jeux où l'histoire est prépondérante et des choix moraux peuvent influer l'histoire. Il aime découvrir plusieurs aspects d'une même histoire en empruntant une route différente de l'ancienne.</p>		
Nom Bobby Watson			
Intitulé de poste Etudiant	<p>Moyen de communication préféré</p> <p>Réseaux sociaux Appels</p>		
Âge Entre 18 et 24 ans	<p>Outils nécessaires au quotidien</p> <p>Téléphone Ordinateur</p>		
Niveau d'études Baccalauréat ou équivalent	<p>Objectifs</p> <p>Trouver des histoires interactives avec un fort impact suite à des choix moraux</p>		
Réseaux sociaux	<p>Principaux défis</p> <p>Forte rejouabilité d'une même histoire Alignement moral présent dans l'histoire</p>		
			
Secteur d'activité Informatique			

Illustration 2 : Persona 2 - Bobby Watson

Ce second persona délimite quant à lui d'autres points :

- Des choix avec un impact fort
- Des choix moraux

2) Benchmark concurrentiel

Ce genre de projets existe principalement dans des formats mobiles même si quelques exemples de site web peuvent subsister. Généralement, les formats mobiles apportent des choix très restreints qui affluent très faiblement sur l'histoire, mais pour forcer l'utilisateur à se servir d'une monnaie virtuelle pour débloquer des images qu'on ne pourrait obtenir autrement.

Certains site web tels que Celestory ou Moiki permettent aux utilisateurs de créer eux même leur histoire pour la partager à d'autres personnes.

Ces projets étant réalisés par des entreprises, ils possèdent davantage de ressources pour fournir de l'art graphique personnalisé afin de garantir une meilleure immersion visuelle.

Afin de se différencier de ce genre de projets, Tutu'Story aura pour vocation de proposer gratuitement des histoires déjà écrites qui apporteront sur le long terme un développement plus mûr et décisif des choix que feront les utilisateurs.

3] Use Case

Les cas d'utilisations sont souvent utilisés pour représenter de manière simple et schématique les différents acteurs qui graviteront avec le projet et les actions qu'ils pourront entreprendre.

Le Use Case ci-dessous est assez simpliste, mais illustre bien la majorité des actions faisables par les deux partis.

Dans notre projet, nous aurons deux rôles : un utilisateur et un administrateur.

Chacun devra être authentifié avant de pouvoir gérer le contenu ou bien créer un personnage ou consulter des histoires déjà écrites.

Un administrateur pourra lui aussi créer ses propres personnages s'il le souhaite

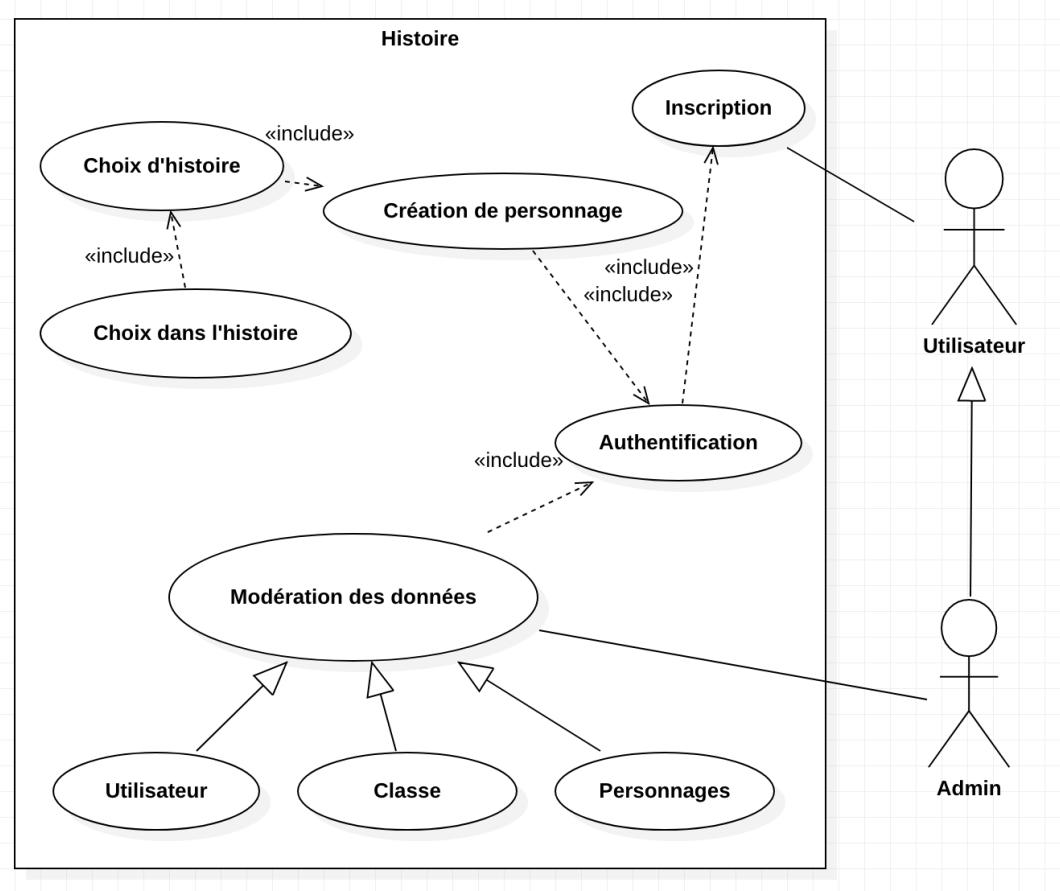


Illustration 3 : Use Case du projet

4] Diagramme de classe

Afin de pouvoir construire le plus facilement possible la base de données, il est plus prudent de réfléchir au préalable à un diagramme des classes, qui permettra de tracer les grandes

lignes de ce qui devrait devenir par la suite les différentes tables qui composeront notre BDD.

Chaque utilisateur pourra obtenir jusqu'à 6 personnages. Ces personnages hériteront lors de leur création des caractéristiques de leur archétype et de leur propre personnage dont ils sont tirés afin de créer leurs propres caractéristiques. Contrairement à celles de l'archétype et du Pj, ces caractéristiques pourraient être modifiées par la suite.

Chaque personnage disposera d'un inventaire, qui contiendra plusieurs emplacements où des objets pourraient s'y trouver.

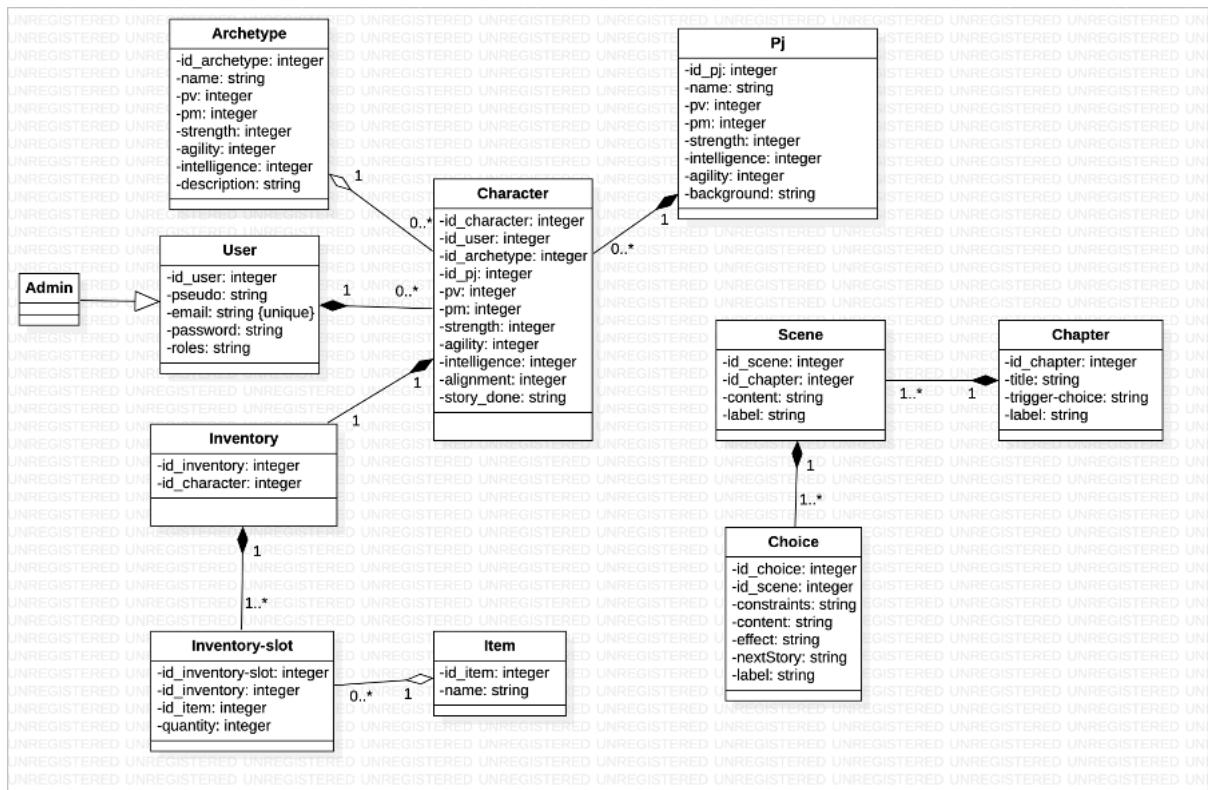


Illustration 4 : Diagramme de classe du projet

Les chapitres, scènes et choix sont à part car n'ont aucun lien nécessaire avec le reste. Les labels de ces trois tables seront les attributs les plus importants, car c'est en fonction de ce label que tous les liens se feront par la suite.

Dans la table Chapter, trigger-choice est une string qui regroupe l'ensemble des labels des scènes qui débloquent un chapitre en particulier.

Dans la table Choice :

- constraints est une string qui contient les contraintes nécessaires pour que ce choix soit possible (bon archétype ? Objet dans l'inventaire ? Etc...)

- effect est une string qui contiendra tous les effets que procureront ce choix : perte de vie, perte ou gain d'alignement, etc...
- nextStory est une string qui contient le label de la scène qui suivra.

L'attribut primordial, celui qui permettra de déterminer tous les choix réalisés par chaque personnage ainsi que de pouvoir relire l'ensemble des histoires se trouvera dans la table Character : story_done.

Ce sera une string qui contiendra toutes les informations nécessaires (en utilisant les labels des chapitres, scènes, choix) pour savoir ce que le personnage a débloqué comme histoire, et donc quelle est celle qu'il pourra lire par la suite.

En effet, étant donné que les choix mènent à des scènes différentes, et parfois à des chapitres différents, il est nécessaire de pouvoir contrôler ce qui est faisable ou impossible. Ceci sera davantage expliqué lors du commentaire du code.

5] Wireframe et maquettes

a. Wireframe

Profil / Accueil

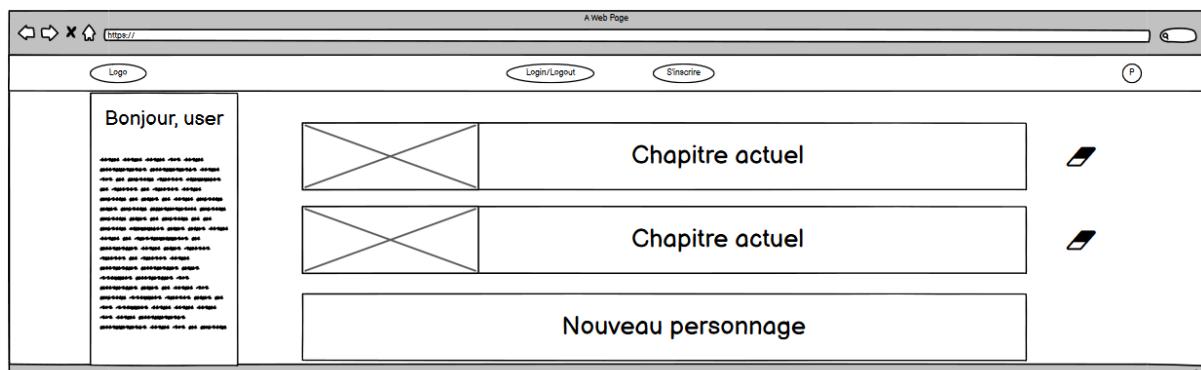


Illustration 5 : Wireframe 1 - Profil

Personnage - histoire

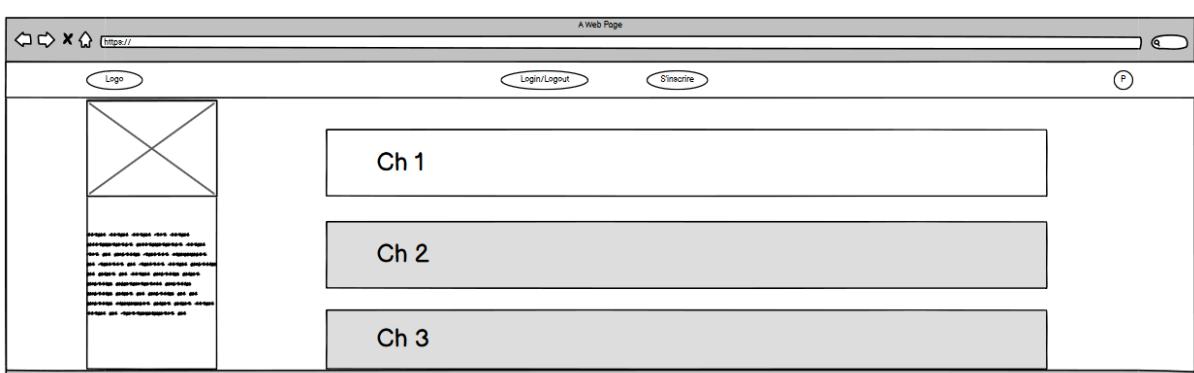
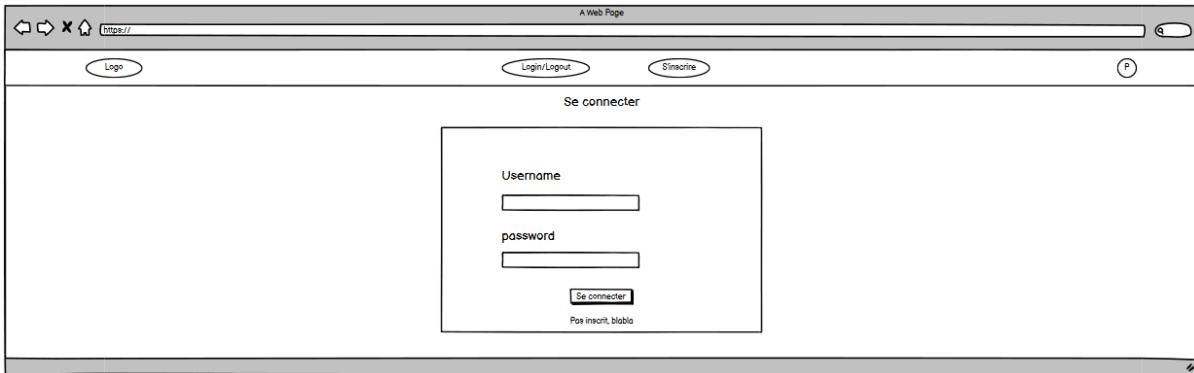


Illustration 6 : Wireframe 2 - Personnage - histoire

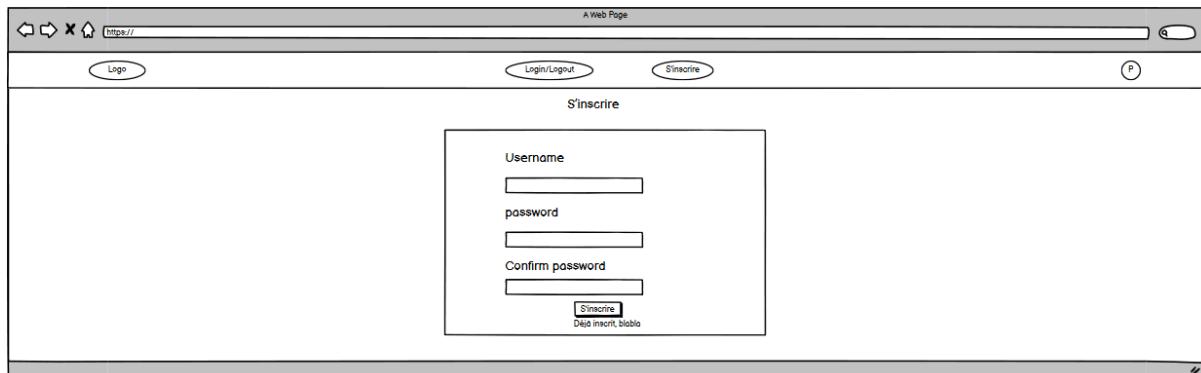
Login



A wireframe of a web browser window titled "A Web Page". The address bar shows "https://". The header contains a "Logo" button, "Login/Logout" and "S'inscrire" buttons, and a profile icon with a "P". Below the header is a "Se connecter" section. It contains two input fields labeled "Username" and "password", a "Se connecter" button, and a message "Pas inscrit, blobo".

Illustration 7 : Wireframe 3 - Login

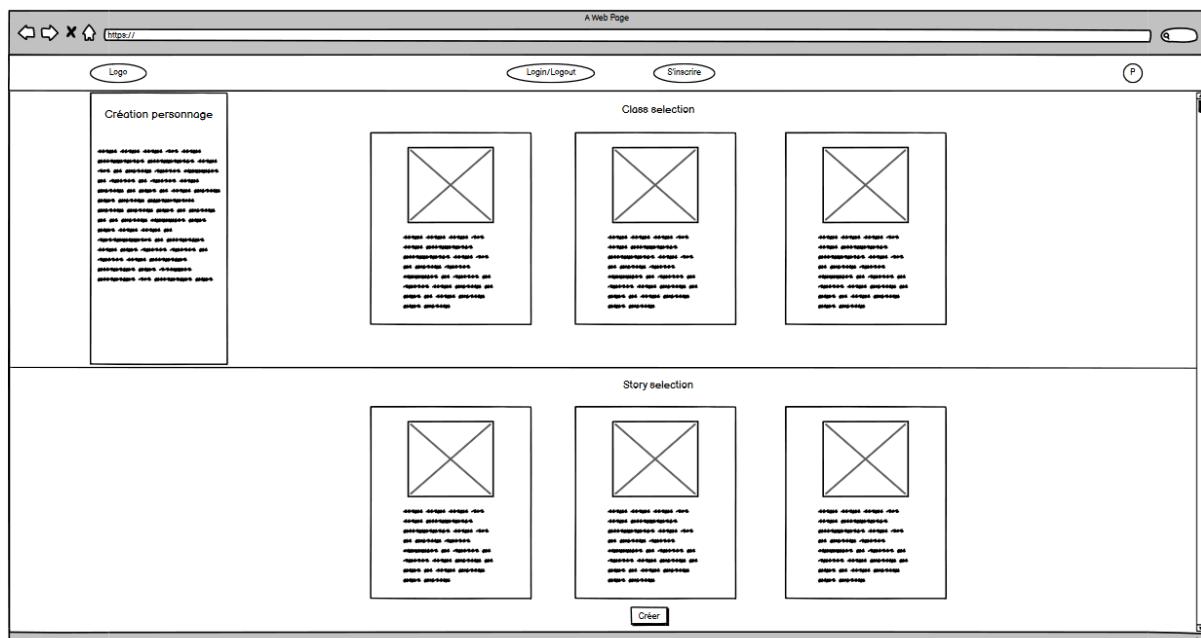
Logout



A wireframe of a web browser window titled "A Web Page". The address bar shows "https://". The header contains a "Logo" button, "Login/Logout" and "S'inscrire" buttons, and a profile icon with a "P". Below the header is a "S'inscrire" section. It contains three input fields labeled "Username", "password", and "Confirm password", a "S'inscrire" button, and a message "Déjà inscrit, blobo".

Illustration 8 : Wireframe 4 - Logout

Character creation



A wireframe of a web browser window titled "A Web Page". The address bar shows "https://". The header contains a "Logo" button, "Login/Logout" and "S'inscrire" buttons, and a profile icon with a "P". The main content area is divided into two sections: "Création personnage" and "Story selection". The "Création personnage" section has a large text area with placeholder text. The "Story selection" section contains three boxes, each with a large "X" and a smaller text area with placeholder text. At the bottom is a "Create" button.

Illustration 9 : Wireframe 5 - Crédit de personnage

Story page

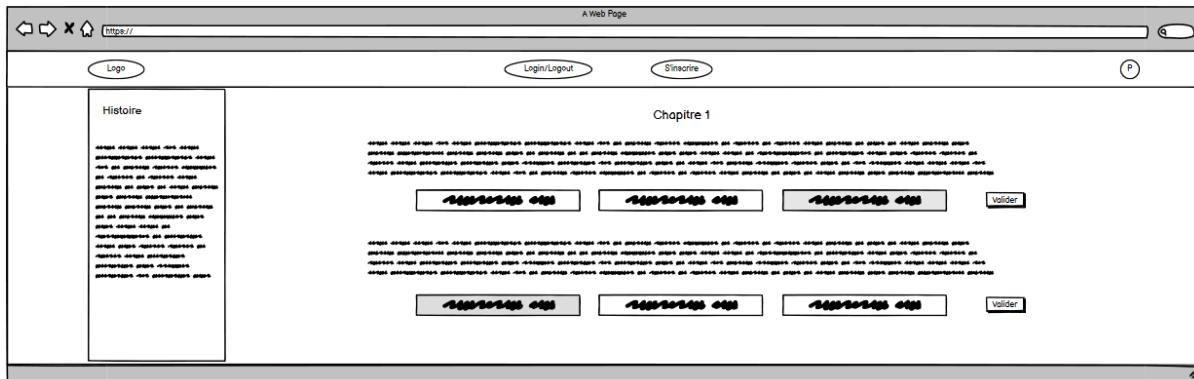


Illustration 10 : Wireframe 6 - Page histoire

Admin tab

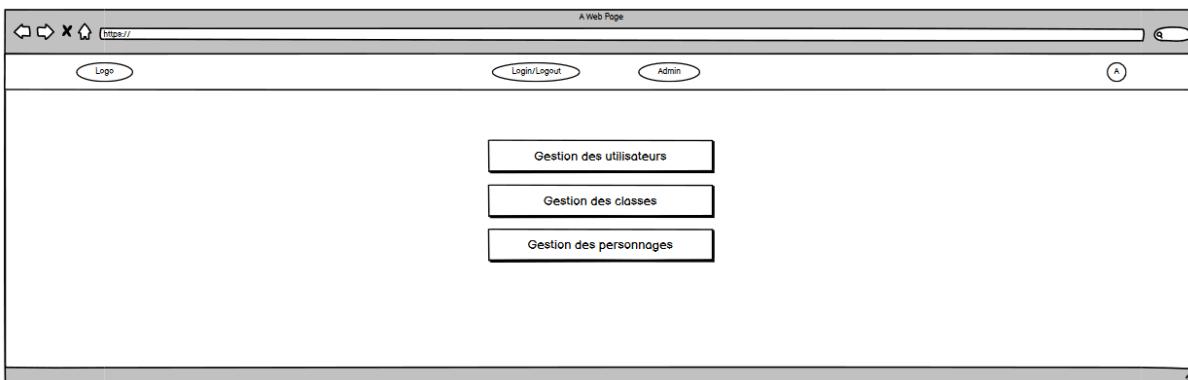


Illustration 11 : Wireframe 7 - Partie admin

Ci-dessus se trouvent tous les wireframe qui recensent la majorité des pages présentes dans le site. Ils ont été réalisés pour que le travail de maquettage ci-dessous prenne le moins de temps possible à être réalisé.

b. Maquettage

L'ensemble des maquettes peut être consulté directement avec le lien suivant sur Figma : [Maquettes du projet](#).

Ce travail de maquettage ayant été fait en amont, quelques libertés ont été prises pour modifier le contenu ou bien le fonctionnement de certaines pages.

Une page d'accueil a été réalisée pour tout le monde (authentifié ou pas), expliquant le fonctionnement du site.

Ci-dessous se trouvent des images des principales maquettes pour chacune des pages du site.

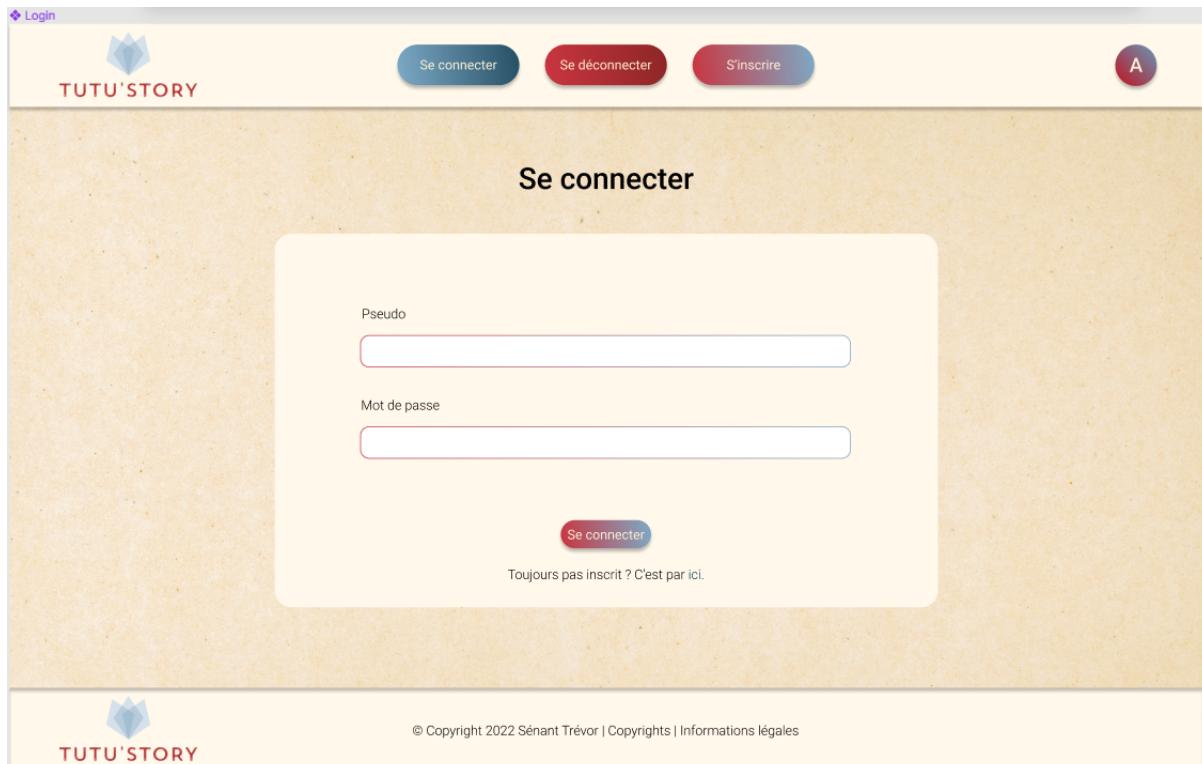


Illustration 12 : Figma 1 - Se connecter

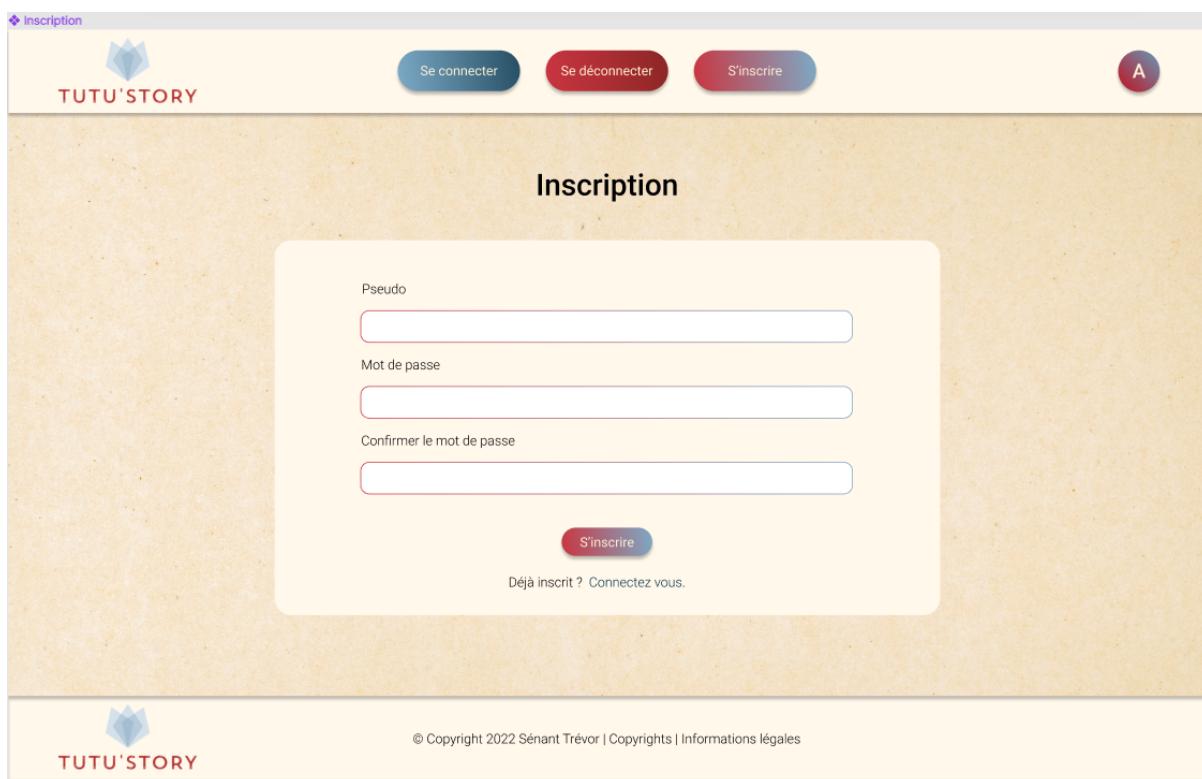


Illustration 13 : Figma 2 - Inscription

❖ Character creation


TUTU'STORY
Se connecter
Se déconnecter
S'inscrire
A

Création de personnage

Chaque personnage dispose déjà d'une histoire personnelle à laquelle il a déjà pu être confronté.

Chacun disposant d'une origine différente, l'histoire pourrait différer en fonction de celui choisi.

Il ne reste plus qu'à vous de compléter leur histoire.

Pour cela, vous pouvez d'abord choisir un archétype de personnage parmi les classes suivantes : guerrier, mage, voleur.

Selection de l'archétype



Mage

Adepte des arts occultes, le mage est capable de matérialiser des forces élémentaires par la force de son esprit. Contrairement aux préjugés, le mage dispose d'une forte endurance afin de pouvoir contrôler au mieux ses pouvoirs sans trop se fatiguer. Il compense cependant sa force physique par sa magie.



Guerrier

Le maniement des armes simples n'a plus aucun secret pour vous. Vous vous êtes entraîné toute votre existence au combat, que ce soit par choix ou par nécessité. Le guerrier dispose d'une grande force physique, mais n'est pas aussi intelligent que peut l'être un mage.



Voleur

Voleur est un raccourci linguistique, car vous ne vous considérez pas comme tel. Vous êtes agile, discret et rapide, certes, mais vous ne subtilisez que (peu) rarement les biens d'autrui. Le voleur est perspicace par empirisme, sachant trouver les bons mots pour se sortir de toutes situations...hasardeuses.

Création de personnage

Chaque personnage dispose déjà d'une histoire personnelle à laquelle il a déjà pu être confronté.

Chacun disposant d'une origine différente, l'histoire pourrait différer en fonction de celui choisi.

Il ne reste plus qu'à vous de compléter leur histoire.

Selection de l'histoire



Agatha Burnwood

Originaire d'un village aux abords d'une ancienne forêt dévastée par les flammes, Agatha s'est retrouvée sans rien du jour au lendemain. Trouvant refuge dans la capitale de l'empire d'Aru, elle doit reconstruire sa vie brique par brique. Saura-t-elle trouver un moyen de recouvrir son ancienne vie, ou cette nouvelle perspective lui fournira-t-elle une raison pour en réchapper ?



Valdwyn d'Aprecourt

Fils illégitime du duc Walter de Port-Vaillant, le célèbre explorateur de Tymalande, Valdwyn a toujours du rester dans l'ombre de sa famille et de sa fratrie. Ayant cependant eu une éducation tel que n'importe quel autre noble du duché, il est capable de comprendre et d'assimiler rapidement son entourage. Vérité passer outre son statut social pour aider ou bien détruire sa famille ?



Alastor

Alastor est un jeune chat errant qui vadrouille aux alentours du temple d'Enkilosh. De nature méfiante envers les humains, si prompt à la violence en le voyant, il reste en retrait de la vie des moines. Cependant, ses sens et sa curiosité le poussent à franchir les barrières qu'il s'était fixé jusque là. Jusqu'où sa témérité va-t-il l'amener et saura-t-il affronter les embûches sur son périple ?

Créer


© Copyright 2022 Sénantr Trevor | Copyrights | Informations légales

Illustration 14 : Figma 3 - Crédit de personnage

Tutu'story | Copyright © Trévor Sénantr 2022

13

Profil

TUTU'STORY

Se connecter Se déconnecter S'inscrire A

Bonjour, user

Cette page de profil recense tous les personnages que vous avez créés.

Le code couleur indique le cheminement global de votre personnage.

Une couleur rouge indique que vous avez préféré des choix qui affectent positivement les autres protagonistes, égoïstes ou bien purement cruels.

Une couleur bleue indique que vous avez préféré des choix qui affectent positivement les autres protagonistes même parfois à l'encontre d'un bénéfice personnel.

Un mélange de couleurs indique que votre personnage est rempli de nuances et qu'il est difficile à cerner.

A l'ombre de Feldrynn

Un choix décisif

De sombres rumeurs

Nouveau personnage

Personnage

Chapitre 1 : Premier chapitre

Chapitre 2 : Second chapitre

Chapitre 3 : Troisième chapitre

Chapitre 4 : Quatrième chapitre

© Copyright 2022 Sénant Trévör | Copyrights | Informations légales

Illustration 15 : Figma 4 - Profil

Personnage

TUTU'STORY

Se connecter Se déconnecter S'inscrire A

Personnage

Ceci est un paragraphe qui résumera brièvement le background du personnage.
Il l'introduira et peut être il y aura une fonction avec un scroll vertical pour avoir un bref résumé des actions passées ?

Chapitre 1 : Premier chapitre

Chapitre 2 : Second chapitre

Chapitre 3 : Troisième chapitre

Chapitre 4 : Quatrième chapitre

Personnage

© Copyright 2022 Sénant Trévör | Copyrights | Informations légales

Illustration 16 : Figma 5 - Personnage

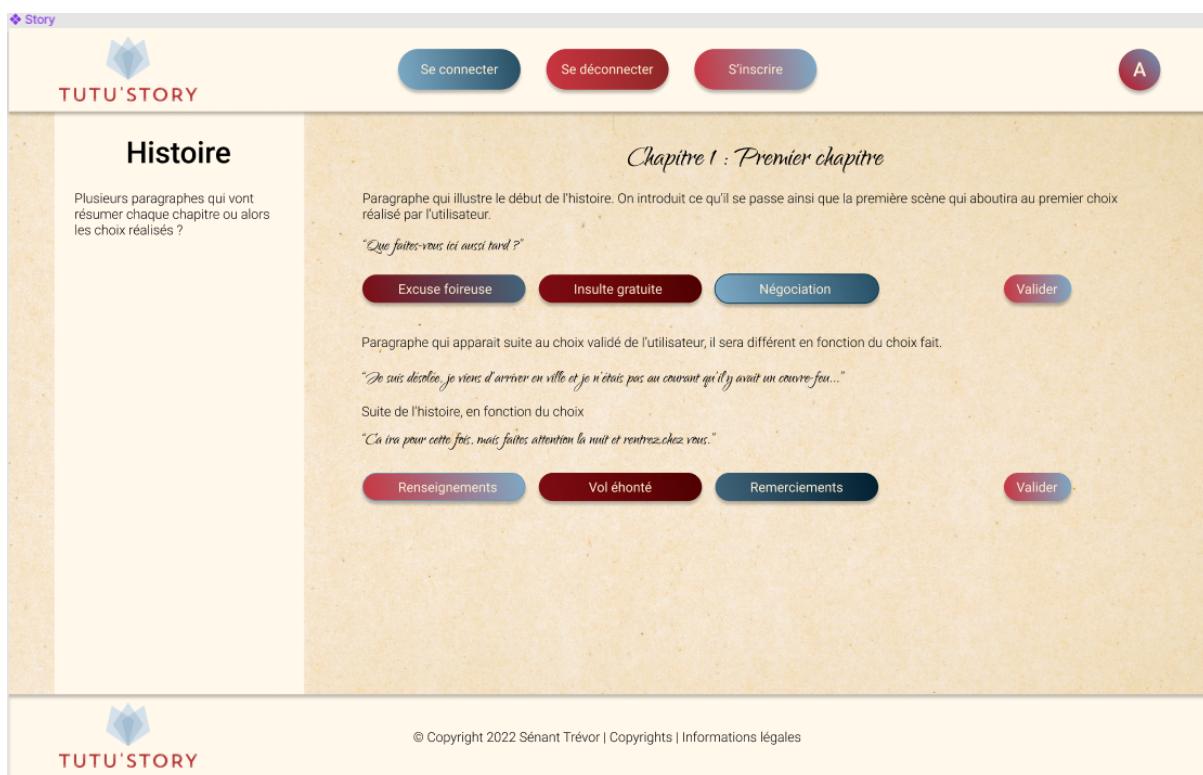


Illustration 17 : Figma 6 - Histoire

c. Choix du style

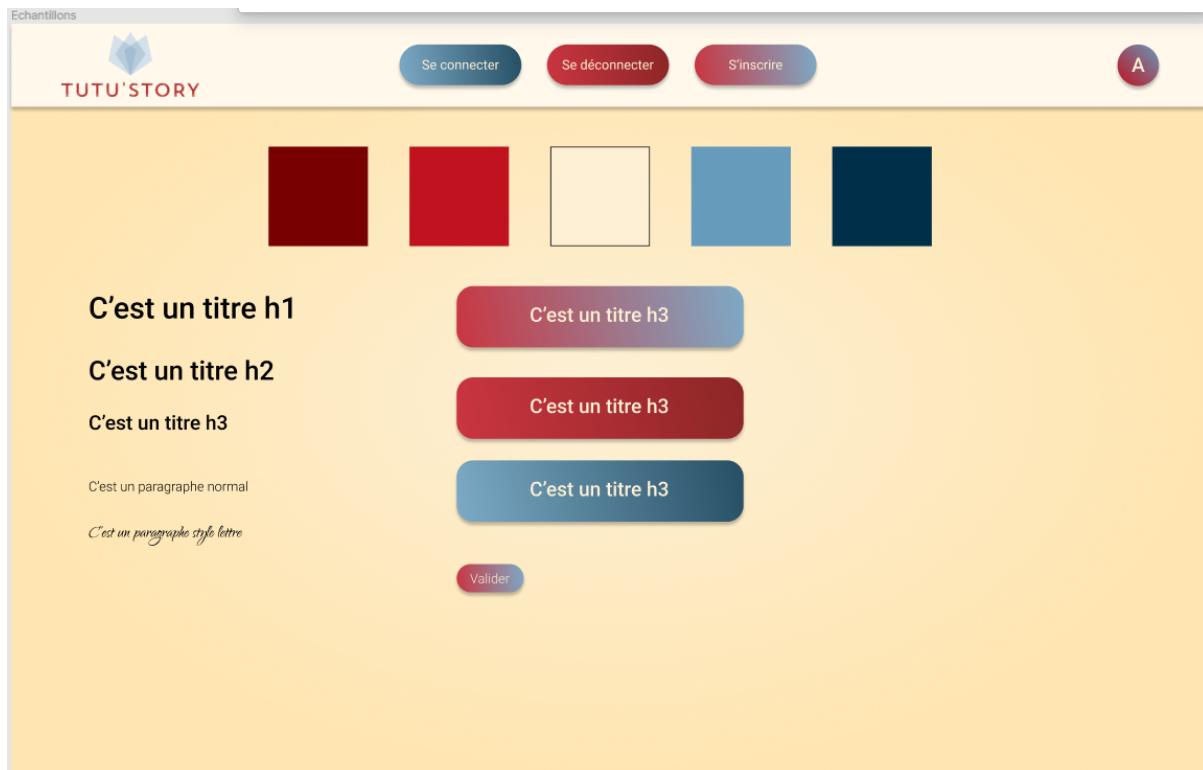


Illustration 18 : Figma 7 - Échantillons

Ci-dessus se trouve une page d'échantillon réalisée avant même tout le reste du maquettage (il y a d'ailleurs l'ancien fond de couleur qui a été remplacé par la suite par une image de parchemin).

La couleur de fond, un parchemin assez clair a été choisi pour illustrer clairement que le but du site est de lire. Le parchemin donne une dimension davantage fantasy, ce qui est le style d'histoire contenu dans ce site.

En ce qui concerne les polices, il me fallait une police qui se charge de tout l'aspect "parcours" du site et une autre avec un style calligraphié pour les noms des chapitres ainsi que pour les éventuels dialogues dans l'histoire.

J'ai donc choisi la police "Roboto" pour tout ce qui est texte général du site et "Qwigley" pour la police calligraphiée.

En ce qui concerne les couleurs choisies, il me fallait un visuel qui tranche entre le bleu et le rouge pour illustrer simplement l'aspect moral des choix ainsi que du personnage incarné.

Dans la majorité des contenus vidéoludiques existants où la notion de moralité et de choix est établie, le rouge est très souvent rattaché à tout ce qui touche aux sentiments forts, brutaux alors que le bleu est lui associé à des choses plus calmes, tempérées.

Ces exemples peuvent être vus dans plusieurs jeux vidéos (Mass Effect, Fable) ou dans certains univers (Star Wars), voire même les deux (Knights of The Old Republic).

J'ai donc décidé d'utiliser cette représentation manichéenne pour l'attribution des couleurs : rouge = méchant, bleu = gentil.

J'ai choisi deux nuances pour chaque afin de créer des gradients de couleurs, plus agréables visuellement.

Tandis que pour les choix sans moralité ou les personnages entre deux, un mélange des deux couleurs primaires donnent une palette de couleur allant du bleu au rouge.

L'ensemble de ces couleurs seront visibles lors :

- Du choix des différents personnages sur la page de profil.
- De la relecture des chapitres déjà réalisés (au niveau des choix)
- De la mini-liste des choix réalisés depuis le début
- Des choix réalisés dans un chapitre en cours
- Dans certains textes destinés à l'utilisateur dans les scènes.

Partie III : Développement du projet

L'ensemble du projet a été réalisé en un peu plus de deux mois. L'ensemble du travail préparatoire (réflexions sur la BDD, persona, maquettage, choix du style, comment gérer certains attributs tels que story_done) a duré entre 2 à 3 semaines.

Le développement du projet lui même en plus de l'écriture des différentes scènes/choix disponibles actuellement a pris un mois entier.

Les finitions ainsi que la mise en production a pris une semaine environ.

1) Langages et framework

Pour la réalisation de ce site, les technologies suivantes ont été utilisées :



HTML (HyperText Markup Language) est un langage qui se sert de balises afin de créer et représenter le contenu d'une page web ainsi que sa structure.



Le CSS (Cascading Style Sheets) permet de créer l'ensemble du style que l'on trouvera dans le site.



Moteur de template pour PHP, utilisé par défaut avec Symfony.



Langage de script côté serveur permettant de faire le lien entre l'utilisateur et la base de données.



Symfony est un ensemble de composants PHP ainsi qu'un framework MVC libre écrit en PHP. Il fournit des fonctionnalités modulables et adaptables, permettant de faciliter et d'accélérer le développement d'un site web.



MySQL (My Structured Query Language) désigne un système de gestion de bases de données relationnelles.



MySQL Workbench est un logiciel de gestion et d'administration de bases de données MySQL.



Heroku est une entreprise créant des logiciels pour serveur qui permettent le déploiement d'applications web.



Visual Studio Code

Visual Studio Code est l'éditeur de code utilisé lors de ce projet

2] Création du projet et élaboration de la BDD

Pour commencer l'élaboration de ce projet, il est nécessaire d'utiliser quelques lignes de commandes. Créer le projet, créer un contrôleur et les différentes entités grâce au diagramme de classe vu précédemment, configurer un accès à la BDD puis opérer les différentes migrations.

Création du projet :

```
composer create-project symfony/website-skeleton:"^5.4"
```

Création du contrôleur :

```
php bin/console make:controller
```

Création des différentes entités :

```
php bin/console make:entity <nom de l'entity>
```

Pour l'entité User, on utilise make:user à la place, étant donné que plusieurs attributs qui seront utilisés par la suite existeront déjà.

Afin de faire le lien entre notre projet Symfony et phpMyAdmin, il est nécessaire de changer quelques configurations dans le .env à la racine de notre projet.

En phase de développement, il suffit de changer la ligne DATABASE_URL en renseignant le user (root), le mot de passe (root), le port associé ainsi que le nom de la future BDD.

En phase de production, il faut utiliser des variables d'environnement afin de cacher les clés secrètes et les mot de passe par mesure de sécurité.

Une fois tout cela réalisé, il ne reste plus qu'à créer une migration et l'effectuer.

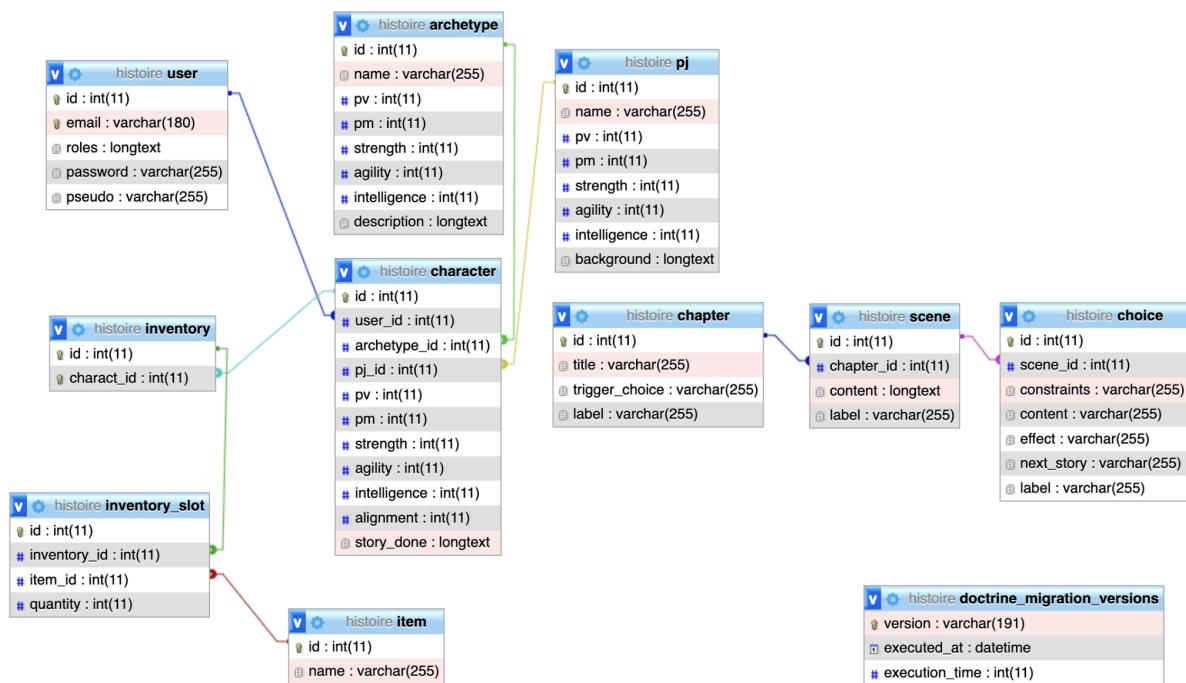
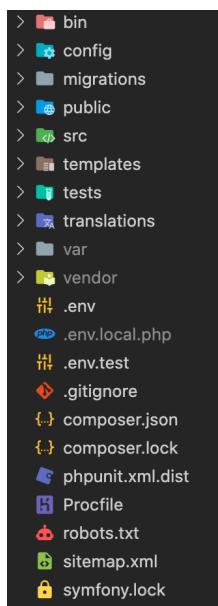


Illustration 19 : Concepteur de la BDD sur phpMyAdmin

3] Architecture du projet



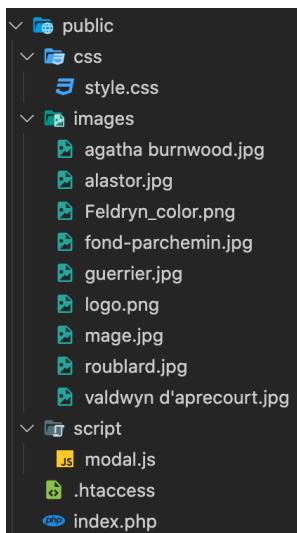
Plusieurs de ces dossiers sont en commun avec la majorité des projets Symfony.

Le fichier Procfile est utilisé par Heroku lors de la mise en production du site.

Le fichier robots.txt et sitemap.xml sont utilisés par les bots google.

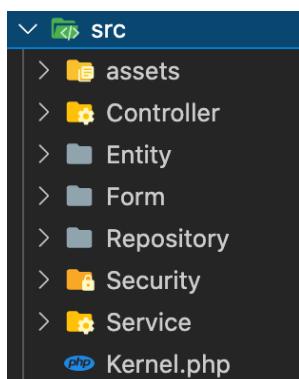
Le dossier config permet de configurer plusieurs packages de Symfony ainsi que les routes

Illustration 20 : Architecture du projet



Le dossier public regroupe le style, les images, le script utilisé pour l'apparition et la disparition d'une modale.

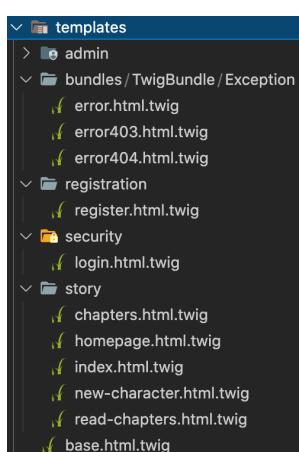
Illustration 21 : Arborescence du dossier public



Le dossier src contient l'ensemble des Controller, Entity, Form et Repository créés avec les lignes de commandes.

Security contient un contrôleur créé lors du formulaire de connexion. Il n'a pas été modifié à part pour renseigner où se connecter après authentification.

Illustration 22 : Arborescence du dossier source



Le dossier template contient tous les templates (vues) du site. Il regroupe ceux liés au côté administrateur, aux templates d'erreurs, d'inscription et de connection (ces deux derniers sont créés automatiquement lors des commandes de formulaires de connection et inscription), les templates côté utilisateur ainsi que le template de base, qui sera utilisé dans presque tous les templates précédents.

Illustration 23 : Arborescence du dossier template

4] Partie Commune

a. Le template base.html.twig

Ce template est, comme l'indique son nom, celui qui contiendra la base de tous les autres templates. C'est lui qui contient la balise <head>, le header et le footer.

```
<!DOCTYPE html>
<html style="scroll-behavior:smooth">

<head>
    <meta charset="UTF-8">
    <meta name="auteur" content="Sénant Trévor">
    <meta name="description" content="Site d'histoire interactive, Tutu'Story permet de se plonger dans un univers d'héroïque | fantaisie et d'en découvrir les multiples facettes en explorant les différentes scènes proposées.">
    <title>{{ block title }}Tutu'Story{{ endblock }}</title>
    <link rel="icon" href="/images/Feldrynn_color.png">
    {# Run `composer require symfony/webpack-encore-bundle` to start using Symfony UX #}
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-1BmE4kWBq78iYhFlvKUhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.3.1/css/all.css"
        integrity="sha384-mzrmE5qonljUremFsqc01SB46JvR0S7bz3I02EmfFsd15uHvIt+Y8vEf7N7fWAU" crossorigin="anonymous">

    {{ block stylesheets }}
    <link rel="stylesheet" href="{{ asset('css/style.css')}}">
    {{ encore_entry_link_tags('app') }}
    {{ endblock }}

    {{ block javascripts }}
    <script src="{{ asset('script/modal.js')}}"></script>
    {{ encore_entry_script_tags('app') }}
    {{ endblock }}
</head>
```

Illustration 24 : Balise <head> du template base.html.twig

On retrouve dans la balise <head> les différentes métadonnées, les liens bootstrap et fontawesome ainsi que les liens pour le style et les éventuels scripts.

L'illustration suivante contient quant à elle le contenu visible du template.

Il est séparé en trois parties distinctes :

- Le header, composé d'une nav où sont regroupés plusieurs liens menant à d'autres pages du site (tel que la connection, la déconnection ou la partie administrateur)
- Une div container, où se trouvera les contenus des différents templates
- Un footer, composé d'un Logo et de quelques textes.

```

<body>
    <header class="shadows">
        <nav>
            <a href="{{path('app_home_page')}}"></a>
            {% if not app.user %}
                <div class="navBoxNoLog">
                    <a href="{{path('app_login')}}" class="blueGradient shadows headerLink">Se connecter</a>
                    <a href="{{path('app_register')}}" class="mixGradient shadows headerLink">S'inscrire</a>
                </div>
                <div class="emptyBox"></div>
            {% else %}
                <div class="navBoxLog">
                    {% for role in app.user.roles %}
                        {% if role == '[ROLE_ADMIN]' %}
                            <a class="blueGradient headerLink" href="{{path('app_admin')}}">Back Office</a>
                        {% endif %}
                        {% endfor %}
                        <a href="{{path('app_logout')}}" class="redGradient headerLink">Se déconnecter</a>
                    </div>
                    <a href="{{path('app_story')}}" class="noHoverShadow headerLink">
                        <div class="profilCercle mixGradient shadows">
                            {{app.user.pseudo|striptags|slice(0,1)|capitalize}}
                        </div>
                    </a>
                {% endif %}
            </div>
        </nav>
    </header>
    <div class="container">
        {% block body %}
        {% endblock %}
    </div>
    <footer>
        
        <div class="footerText">
            <p>© Copyright 2022 Sénant Trévor –
                <a href="{{path('app_copyrights')}}">Copyrights</a> | <a href="{{path('app_infos')}}">Informations légales</a>
            </p>
        </div>
    </footer>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
        integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYs0g+OMhuP+IlRH9sENB00LRn5q+8nbTov4+1p"
        crossorigin="anonymous">
    </script>
</body>
</html>

```

Illustration 25 : Balise <body> du template base.html.twig

La navigation du header contient différentes briques, qui apparaissent ou non en fonction de certaines conditions.

Les boutons ‘Se connecter’ et ‘S’inscrire’ n’apparaissent que si l’utilisateur n’est pas connecté. Sinon, le bouton ‘Se déconnecter’ sera présent.

De plus, une vérification du rôle se fait si l’utilisateur connecté dispose des droits administrateurs. Si c’est le cas, un bouton d’accès au Back Office sera présent.

Enfin, un petit cercle contenant la première lettre du pseudo utilisé par l’utilisateur lors de la création de son compte sera présent.

b. Le template homepage.html.twig

```
/**  
 * @Route("/", name="app_home_page")  
 */  
  
public function homepage()  
{  
    return $this->render('story/homepage.html.twig');  
}
```

Ce template est accessible via la route basique du site web.

Dans le StoryController, ce n'est qu'une fonction qui retourne un render du template, permettant de le visualiser.

Illustration 26 : La route app_home_page

```
{% extends 'base.html.twig' %}  
  
{% block title %}Accueil{% endblock %}  
  
{% block body %}  
<div class="bigBox">  
    <div class="columnBox">  
        <h1>Bienvenue</h1>  
        <div class="textBox">  
            <p>Tutu'story est un site qui contient des histoires interactives, dans le style des anciens livres "où vous êtes le héros".</p>  
            <p>Il vous sera possible de suivre l'histoire de trois protagonistes aux origines et histoires différentes.</p>  
            <ul>  
                <li>Agatha Burnwood, une femme résidant dans un village dans les contrées sauvages de l'empire d'Arutl.</li>  
                <li>Valdwyn d'Aprecourt, un bâtard affilié à la noblesse de Port-Vaillant.</li>  
                <li>Alastor, un chat errant dans les alentours du temple d'Enkilosh.</li>  
            </ul>  
            <p>Il vous sera possible d'avoir un archétype de personnage, qui vous donnera accès à des choix et des scènes exclusives.</p>  
            <p>Plus vous avancerez dans l'histoire, plus votre personnage sera marqué par les événements. Quelle sera la voie que vous lui choisirez ?</p>  
            <ul>  
                <li><span class="redText"> Egoïste, brutale et sanguinaire ?</span></li>  
                <li><span class="blueText">Conciliante, vertueuse et juste ?</span></li>  
                <li><span class="mixColorText">Ou bien, un peu des deux ?</span></li>  
            </ul>  
            <p>Il ne reste plus qu'à vous de le découvrir.</p>  
            <div class="buttonBox"><a href="{{path('app_story')}}" class="button mixGradient shadows">Poursuivre</a></div>  
        </div>  
    </div>  
    <div class="contentBox">  
        {% for classes,messages in app.flashes %}  
        <div class="text-center alert alert-{{ classes }}>  
            {% for message in messages %}  
            {{ message }} <br>  
            {% endfor %}  
        </div>  
        {% endfor %}  
        <div class="textBox">  
            <h2 class="text-center qwigleyTitle">Feldryn</h2>  
              
            <p>Ce site est encore en construction et a été réalisé dans le cadre d'un projet personnel pour l'obtention d'un diplôme.</p>  
            <p>Certaines histoires ne sont pas encore finies et/ou complètes.</p>  
            <p>Certaines scènes ou choix peuvent heurter la sensibilité de personnes non averties.</p>  
        </div>  
    </div>  
</div>  
{% endblock %}
```

Illustration 27 : Le template homepage.html.twig

Dans chacun des templates secondaires, il est nécessaire d'utiliser un **extends** du template de base pour que le contenu du template secondaire puisse être assimilé à celui de base. Celui-ci se trouvera donc dans un container.

Les classes BigBox, columnBox, textBox et contentBox sont des classes qui apparaissent régulièrement dans chacun des templates secondaires. Elles sont utilisées pour donner la même structure et apparence aux différentes pages du site. En outre, elles permettent au site d'être responsive.

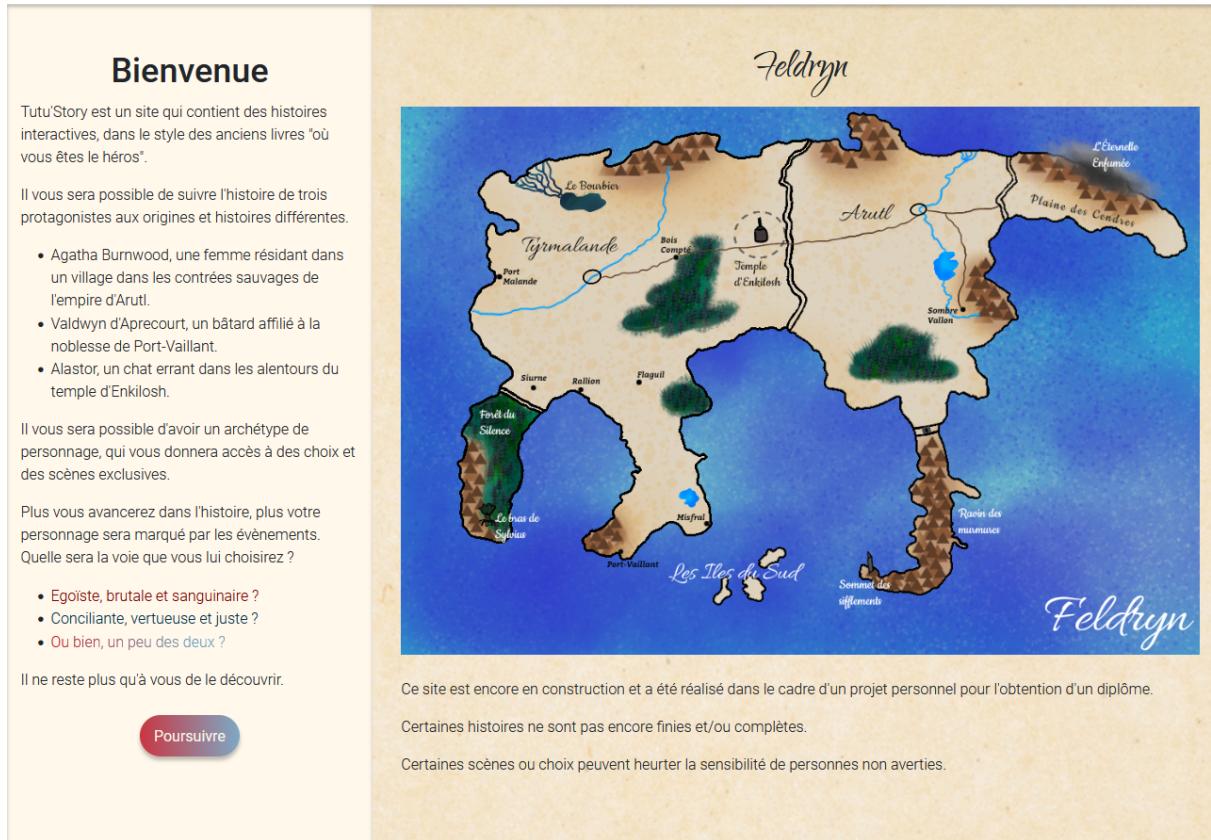


Illustration 28 : Visuel du template en grand écran

La majorité du visuel des templates secondaires disposent des mêmes structures et visuels. Ils sont tous découpés en deux parties distinctes. Une colonne sur la gauche qui dispose d'un scroll vertical indépendant et où plusieurs informations changent en fonction du template, et la partie où se trouve le contenu principal du template sur la droite.

C'est pourquoi je ne détaillerai pas en détail le style et le visuel de chacun des autres templates, étant donné qu'ils disposent tous du même comportement en fonction de la largeur de l'écran sur lequel le site est consulté.

Ci dessous, voici le même template en version mobile. On voit que la disposition en colonnes a fait place à une succession de lignes, permettant une lecture de l'ancien panel de gauche ainsi que de l'ancien contenu juste en dessous.

Il existe d'autres points d'arrêt entre la version mobile et le grand écran, où certaines images sont redimensionnées pour éviter des problèmes visuels.



TUTU'STORY

[Se connecter](#) [S'inscrire](#)

Bienvenue

Tutu'story est un site qui contient des histoires interactives, dans le style des anciens livres "où vous êtes le héros".

Il vous sera possible de suivre l'histoire de trois protagonistes aux origines et histoires différentes.

- Agathe Burnwood, une femme résidant dans un village dans les contrées sauvages de l'empire d'Aruti.
- Valdwyn d'Agrecourt, un bâtard affilié à la noblesse de Port-Vallant.
- Aleator, un chat errant dans les alentours du temple d'Enklash.

Il vous sera possible d'avoir un archétype de personnage, qui vous donnera accès à des choix et des actions exclusives.

Plus vous avancerez dans l'histoire, plus votre personnage sera marqué par les événements. Quelle sera la voie que vous lui choisirez ?

- Egoliste, brutale et sanguinaire ?
- Conciliante, vertueuse et juste ?
- Ou bien, un peu des deux ?

Il ne reste plus qu'à vous de le découvrir.

[Poursuivre](#)



Feldrynn

Ce site est encore en construction et a été réalisé dans le cadre d'un projet personnel pour l'obtention d'un diplôme.

Certaines histoires ne sont pas encore finies et/ou complètes.

Certaines actions ou choix peuvent heurter la sensibilité de personnes non averties.

Illustration 29 : Visuel du template en version mobile

En outre, dans chacun des templates secondaires se trouve une boucle for qui tourne sur app.flashes. Cette boucle permet de restituer un message que l'on envoie en cas de réussite ou erreur dans le controller. Lors de la création de ce message, on envoie deux arguments à la fonction addFlash : le nom d'une classe Bootstrap et le contenu du message.

Ainsi, chaque template pourra restituer des cadres d'erreurs dont la couleur changera en fonction de classe Bootstrap renseignée. Si plusieurs messages d'erreurs sont présents, ils seront chacun dans un cadre en fonction de leur couleur.

c. Inscription et connexion

Comme mentionné précédemment, les formulaires et templates d'inscription et de connexion ont été créés grâce à une ligne de commande spécifique. Ils disposent en outre chacun d'un controller qui a été très peu modifié (à part pour renseigner sur quelle route être redirigé après une connexion).

La majorité des changements opérés furent dans le Form de l'inscription ainsi que l'élaboration de son template.

```
class RegistrationFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('pseudo')
            ->add('email')
            ->add('majeur', CheckboxType::class, [
                'mapped' => false,
                'constraints' => [
                    new IsTrue([
                        'message' => 'Vous devez bien préciser que vous êtes majeur.',
                    ]),
                ],
            ])
            ->add('plainPassword', RepeatedType::class, [
                'type' => PasswordType::class,
                'invalid_message' => 'Les mots de passe ne correspondent pas.',
                'first_options' => ['label' => 'Mot de passe', 'attr' => ['class' => 'borderColor']],
                'second_options' => ['label' => 'Confirmer le mot de passe', 'attr' => ['class' => 'borderColor']],
                'mapped' => false,
                'attr' => ['autocomplete' => 'disabled'],
                'constraints' => [
                    new NotBlank([
                        'message' => 'Veuillez renseigner un mot de passe.',
                    ]),
                    new Length([
                        'min' => 6,
                        'minMessage' => 'Votre mot de passe doit contenir au moins {{ limit }} caractères.',
                        // max length allowed by Symfony for security reasons
                        'max' => 4096,
                    ]),
                ],
            ]),
        );
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'attr' => [
                'novalidate' => 'novalidate'
            ],
            'data_class' => User::class,
        ]);
    }
}
```

Illustration 30 : RegistrationFormType.php

Pour ce formulaire, on constate qu'il y a quatre champs nécessaires, dont l'un est dédoublé. Le mot de passe étant un RepeatedType, il y aura donc deux champs distincts qui devront posséder la même valeur au risque de provoquer une erreur.

La gestion des erreurs et la sécurité est gérée en outre par Doctrine directement dans l'Entity User.

```
/**  
 * @ORM\Column(type="string", length=180, unique=true)  
 * @Assert\NotBlank(message="Renseignez un email.")  
 * @Assert\Email(message="L'email {{ value }} n'est pas un email valide.")  
 */  
private $email;  
  
/**  
 * @ORM\Column(type="json")  
 */  
private $roles = [];  
  
/**  
 * @var string The hashed password  
 * @ORM\Column(type="string")  
 */  
private $password;  
  
/**  
 * @ORM\Column(type="string", length=255)  
 * @Assert\NotBlank(message="Renseignez un pseudo.")  
 * @Assert\Length(  
 *     min = 4,  
 *     max = 20,  
 *     minMessage = "Votre pseudo doit contenir au moins {{ limit }} caractères.",  
 *     maxMessage = "Votre pseudo doit contenir moins de {{ limit }} caractères."  
 */  
private $pseudo;
```

Illustration 31 : Les contraintes dans l'entity User

Lors du remplissage des champs, l'utilisateur sera donc obligé de remplir l'ensemble des champs tout en respectant les différentes contraintes et sécurité mises en place, sous peine d'être face à de nombreuses erreurs qui ne lui permettront pas de valider son inscription et/ou sa connexion.

Ceci permet aussi en outre de protéger le site d'éventuelles injection SQL.

Lors de l'inscription, Symfony utilise de lui-même une méthode de hashage des mots de passe (transformer la string renseignée par l'utilisateur en la cryptant en une autre string qui sera entreposée dans la BDD). Le mot de passe renseigné par l'utilisateur n'est jamais stocké quelque part.

S'inscrire

Pseudo
Renseignez un pseudo.

Email
Renseignez un email.

Mot de passe
Veuillez renseigner un mot de passe.

Confirmer le mot de passe

Je suis majeur
 Vous devez bien préciser que vous êtes majeur.

S'inscrire

Déjà inscrit ? Connectez vous.

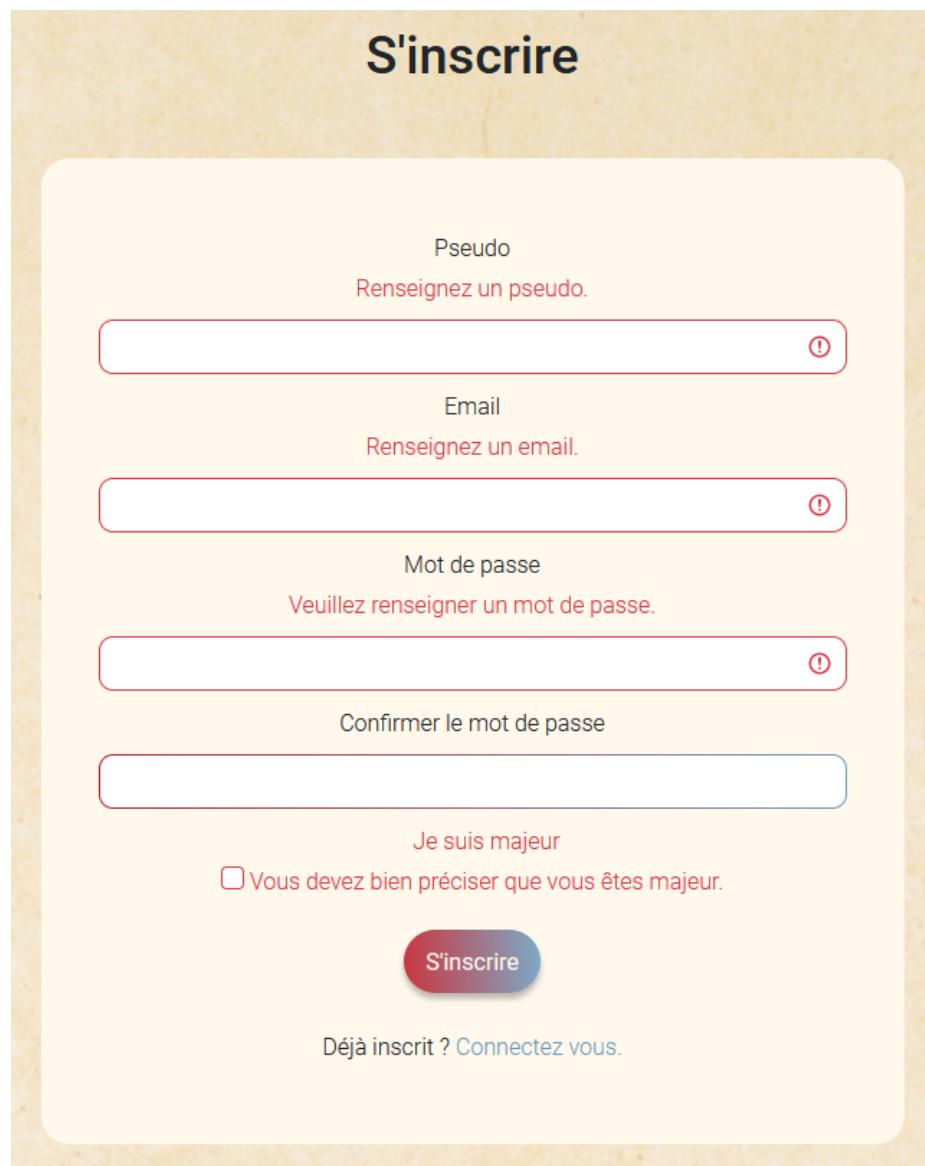


Illustration 32 : Exemple de l'apparition des erreurs lors d'une mauvaise inscription

5] Partie Utilisateur

a. Structure de story_done

Avant de parler plus avant du code qui suivra, il est nécessaire de faire un aparté sur l'attribut se trouvant dans l'entity Character mentionné plus tôt : story_done.

Une histoire dispose de cohérences, il est impossible de passer du chapitre 1 au 5 pour ensuite revenir au 2. story-done est une string qui permettra de stocker au fil du temps l'ensemble des chapitres, scènes et choix réalisés pour chaque personnage. Mais comment ça marche ?

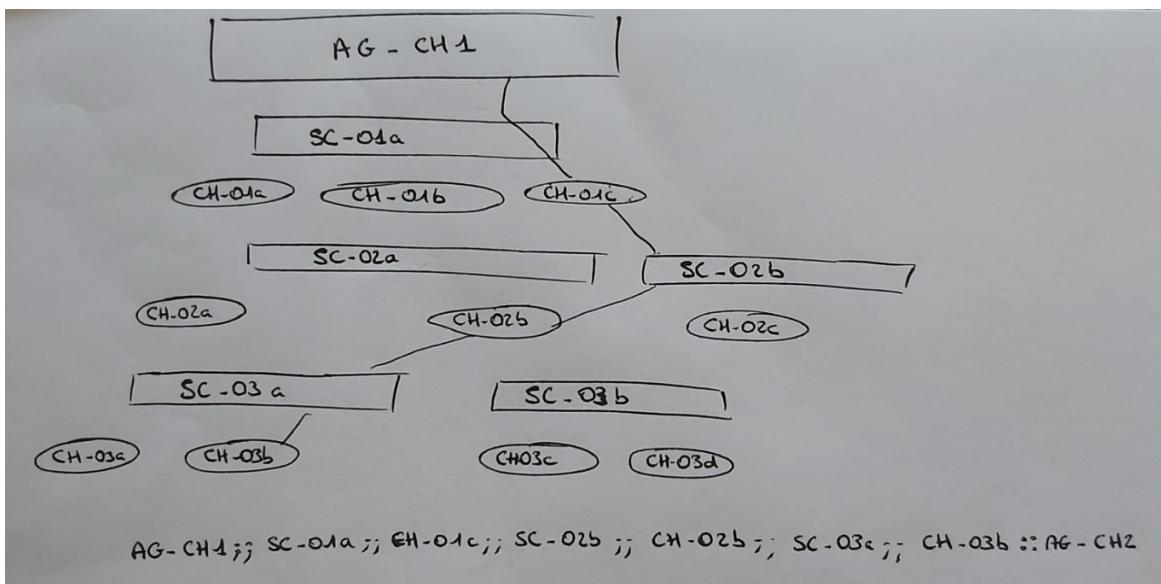


Illustration 33 : Représentation schématique du chapitre 1 d'Agatha Burnwood

Ci-dessus, on constate que le chapitre 1 d'Agatha dispose d'une succession de 3 scènes et choix. Le trait entre chaque cadre indique les choix et les scènes débloquées par l'utilisateur, ce qui est traduit en bas par cette chaîne de caractères composés de plusieurs labels.

Chaque chapitre commence par les deux premières lettres du protagoniste (AG pour Agatha, VA pour Valdwyn et AL pour Alastor). Tous les contenus de chaque chapitres sont séparés par des ";;". Au sein même d'un seul chapitre, tous les éléments sont séparés par des ";;".

Ces différents marqueurs nous permettront ainsi pour la suite de décomposer cette string en des tableaux multidimensionnels (séparation des chapitres d'une part, puis séparation de chaque élément au sein d'un même chapitre).

Ceci nous permettra donc de pouvoir récupérer les données liées à chaque chapitre lu par l'utilisateur tout en gardant en mémoire ses choix.

b. Page de profil

```
/**  
 * @Route("/story", name="app_story")  
 */  
public function index(CharacterRepository $charaRepo, ChapterRepository $chapterRepo, RequestStack $rs): Response  
{  
    if($this->getUser())  
    {  
        $characters = $charaRepo->getAllCharactersByUser($this->getUser());  
        $creation = true;  
        if(count($characters) >= 6){  
            $creation = false;  
        }  
        $titles = [];  
        foreach ($characters as $character) {  
            $chapters = explode('::',$character->getStoryDone());  
            $chapter = explode(';', end($chapters));  
            $title = $chapterRepo->findChapterByLabel($chapter[0])[0]->getTitle();  
            $position = strpos($title,":");  
            $title = substr($title,$position+2);  
            array_push($titles,$title);  
        }  
        return $this->render('story/index.html.twig', [  
            'data' => [  
                'characters' => $characters,  
                'titles' => $titles,  
            ],  
            'creation' => $creation  
        ]);  
    }  
    else  
    {  
        return $this->redirectToRoute('app_login');  
    }  
}
```

Illustration 34 : Controller, route app_story

La première chose que l'on réalise sur cette route, est de vérifier si l'utilisateur est connecté. Ceci est une mesure de sécurité rajoutée suite à une complication lors de la mise en production, normalement il est possible de configurer l'accès à des parties du site rapidement via le security.yaml se trouvant dans config/packages.

Si l'utilisateur n'est pas authentifié, il sera donc redirigé automatiquement vers la page de connexion.

Une limite de 6 personnages par compte a été instaurée afin de ne pas surcharger les bases de données. C'est pourquoi on récupère l'ensemble des personnages de l'utilisateur pour ensuite compter le nombre d'éléments récupérés. Par défaut, la création de personnage est en "true", si le nombre atteint 6, alors il sera en "false". Ce boolean sera utilisé dans le template correspondant.

Sur cette route, on utilise une boucle foreach afin de tourner sur l'ensemble des personnages. Ensuite, on "explode" l'attribut story-done par deux fois (ce qui permet de créer les tableaux comme mentionné plus tôt).

On précise au second explode que l'on ne souhaite créer un tableau que du dernier élément (donc, le dernier chapitre où l'utilisateur s'est arrêté).

Le label du chapitre étant toujours à la première position, on le récupère pour obtenir son titre grâce au Repository de Chapter.

Les deux lignes suivantes permettent juste de récupérer le titre du chapitre, en éclipsant le reste.

Dans le template index.html.twig, on récupère deux variables : ‘data’ et ‘creation’, envoyées pendant le render.

Voici ci-dessous des parties du template.

```
<h2>Limite de personnages : {{ data.characters|length }} / 6 </h2>
{% for character in data.characters %}
<div class="textBox mt-4">
    <h3> {{ character.archetype.name }}</h3>
</div>
<div class="flexBar">
    <a href="{{path('app_chapters', {id: character.id })}}" class="linkChapter">
        <div class="chapterBar shadows {% if character.alignment < -50 %}>
            redGradient
        {% elseif character.alignment > 50 %}>
            blueGradient
        {% else %}>
            mixGradient
        {% endif %}">
            
            <div class="titleBox">
                <h2 class="">
                    {{ data.titles[ loop.index0 ] }}
                </h2>
            </div>
        </div>
    </a>
    <i class="fa fa-eraser" id="delete-{{character.id}}"></i>
</a>
</div>
<hr>
```

Illustration 35 : template index.html.twig

Dans ce template, on boucle sur l'ensemble des personnages récupérés afin d'afficher les détails de chacun, les uns au-dessus des autres.

Au niveau de la classe de ce qui contient l'ensemble des données, nous avons une condition qui permet, en fonction de l'alignement du personnage, de changer la couleur du fond :

- Alignement > 50 : gradient bleu
- Alignement < -50 : gradient rouge
- Intervalle entre les deux : gradient mixte

Au niveau de l'image, on utilise l'attribut name du pj que l'on récupère dans les données pour aller chercher l'image stockée et qui porte le même nom. On ajoute le filtre |lower afin d'éviter tout souci de casse.

De plus, nous avons une balise `<i>` qui est une icône fontAwesome d'effaceur, il possède une id qui sera utilisée dans le script Javascript suivant.

```

<div class="modalBox"></div>
<div class="modale">
    <p>Confirmation de la suppression du personnage</p>
    <a href="" class="redGradient deleteLink button">Suppression</a>
    <span class="blueText modalText">Annuler</span>
</div>

```

Illustration 36 : fin du template index.html.twig

```

document.addEventListener("DOMContentLoaded", () => {
    console.log("Page chargée");
    document.addEventListener("click", (e) => {
        let modalBox = document.querySelector(".modalBox");
        let modale = document.querySelector(".modale");
        if (e.target.id.indexOf("delete") !== -1) {
            modalBox.style.display = "block";
            modale.style.display = "block";
            let idSuppr = e.target.id.substring(e.target.id.indexOf("-") + 1);
            let link = document.querySelector(".deleteLink");
            link.href = `/story/delete/${idSuppr}`;
        }
        if (
            e.target.className == "modalBox" ||
            e.target.className == "modale" ||
            e.target.className == "blueText modalText"
        ) {
            modalBox.style.display = "none";
            modale.style.display = "none";
        }
    });
});

```

Illustration 37 : script pour l'apparition/disparition d'une modale

Si l'utilisateur clic sur l'icône fontAwesome dont l'id possède “delete”, alors il entrera dans la première condition du script, qui changera le style des modales afin qu'elles apparaissent. modalBox est un filtre d'opacité sur l'ensemble de la fenêtre de navigation alors que modale est le cadre de la modale elle même.

Il récupère ensuite l'id du personnage en se débarrassant grâce au substring à tout ce qu'il se trouve avant le ‘-’ de l'id, lui compris. Ceci permet de rendre le lien de suppression de la modale dynamique, lui permettant de supprimer n'importe quel personnage.

Si l'utilisateur clic sur la modale, le cadre opaque en arrière plan ou sur “annuler”, la modale disparaîtra.



Illustration 38 : Affichage de la page profil

c. Création de personnage

La création de personnage utilise un form composé de deux attributs : l'archétype et pj. Ces deux attributs font écho aux deux autres tables où ils vont chercher les données. Le formulaire aura donc la forme de deux menus déroulants où l'utilisateur pourra choisir indépendamment l'archétype du personnage et le personnage en lui-même.

```
class CharacterCreationFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('archetype', EntityType::class, [
                'class' => Archetype::class,
                'choice_label' => 'name'
            ])
            ->add('pj', EntityType::class, [
                'class' => Pj::class,
                'choice_label' => 'name'
            ])
    }
}
```

Illustration 39 : CharacterCreationFormType.php

```
/*
 * @Route("/story/new", name="app_new_character")
 */

public function newCharacter(Request $request, EntityManagerInterface $manager, ArchetypeRepository $aRepo,
    PjRepository $pjRepo, CharacterService $characService, CharacterRepository $charaRepo)
{
    if($this->getUser())
    {
        $characters = $charaRepo->getAllCharactersByUser($this->getUser());
        if(count($characters) < 6){
            $character = new Character;
            $character->setUser($this->getUser());
            $character->setInventory(new Inventory);
            $character->setAlignment(0);
            $form = $this->createForm(CharacterCreationFormType::class, $character);
            $form->handleRequest($request);
            if($form->isSubmitted() && $form->isValid()) {
                $characService->characterCreation($character);
                $manager->persist($character);
                $manager->flush();
                $this->addFlash('success','Personnage créé avec succès.');
                return $this->redirectToRoute('app_story');
            }
            return $this->render("story/new-character.html.twig", [
                'formCharacter' => $form->createView(),
                'archetypes' => $aRepo->findAll(),
                'histoires' => $pjRepo->findAll()
            ]);
        } else {
            $this->addFlash('danger','Nombre limite de personnage atteint');
            return $this->redirectToRoute('app_story');
        }
    } else
    {
        $this->addFlash('danger','Connexion requise');
        return $this->redirectToRoute('app_login');
    }
}
```

Illustration 40 : Controller, route app_new_character

Afin de contrer ceux qui utiliseraient directement l'url, on précise en premier lieu que la condition pour laquelle le contenu de cette route s'exécutera est que l'utilisateur possède strictement moins de 6 personnages.

Ensuite, on opère comme à la création de n'importe quel formulaire de Symfony, on instancie la classe Character et on set quelques données initiales avant de le créer pour ensuite envoyer tout ce dont nous avons besoin via un render.

Une fois le formulaire validé et valide, on utilise un service créé afin d'alléger le code. Ce service permet, en utilisant l'archétype et le Pj validé avec le formulaire, de créer les caractéristiques du personnage.

De plus, c'est ici que l'on initie pour la première fois story_done en y inscrivant le label du premier chapitre du personnage associé ainsi que la première scène correspondante.

Une fois que tout est en ordre, on utilise l'EntityManagerInterface pour persister et sauvegarder les données en BDD

```

<?php

namespace App\Service\Character;

class CharacterService
{
    public function characterCreation($character){
        $character->setPv($character->getArchetype()->getPv() + $character->getPj()->getPv());
        $character->setPm($character->getArchetype()->getPm() + $character->getPj()->getPm());
        $character->setStrength($character->getArchetype()->getStrength() + $character->getPj()->getStrength());
        $character->setAgility($character->getArchetype()->getAgility() + $character->getPj()->getAgility());
        $character->setIntelligence($character->getArchetype()->getIntelligence() + $character->getPj()->getIntelligence());
        if($character->getPj()->getName() == "Agatha Burnwood"){
            $character->setStoryDone('AG-CH1;;AG-SC-01a');
        } else if ($character->getPj()->getName() == "Valdwyn d'Aprecourt") {
            $character->setStoryDone('VA-CH1;;VA-SC-01a');
        } else if ($character->getPj()->getName() == "Alastor") {
            $character->setStoryDone('AL-CH1;;VA-SC-01a');
        }
    }
}

```

Illustration 41 : CharacterService.php

```

<h2>Sélection de l'archétype</h2>
{{ form_start(formCharacter)}}
<div class="characterBox">
    {% for archetype in archetypes %}
        <div class="characterCard">
            
            <h3>{{ archetype.name }}</h3>
            <p>{{ archetype.description }}</p>
        </div>
    {% endfor %}
</div>

{{ form_row(formCharacter.archetype, {
    label: "Choisissez un archétype",
    attr: {
        class: "my-2"
    }
})}}
<h2>Sélection de l'histoire</h2>
<div class="characterBox">
    {% for histoire in histoires %}
        <div class="characterCard">
            
            <h3>{{ histoire.name }}</h3>
            <p>{{ histoire.background }}</p>
        </div>
    {% endfor %}
</div>

{{ form_row(formCharacter.pj, {
    label: "Choisissez le personnage principal",
    attr: {
        class: "my-2"
    }
})}}
<button type="submit" class="button shadows mixGradient">Créer</button>
{{ form_end(formCharacter)}}

```

Illustration 42 : Partie du template new-character.html.twig

Pour ce formulaire, on l'initie grâce au “form_start” pour le terminer avec un “form_end”. On fait apparaître les menus déroulants grâce à des “form_row” où on a configuré quelques paramètres comme l'ajout de labels.

Ce formulaire se retrouve caché entre des cards représentant chaque personnage et archétype.

d. Affichage des chapitres

```
/**  
 * @Route("/story/chapters/{id}", name="app_chapters")  
 */  
  
public function showChapters(Character $character = null, CharacterRepository $characRepo, ChapterRepository $chapterRepo)  
{  
    if($this->getUser())  
    {  
        if($character){  
            $characters = $characRepo->findBy(array('user' => $this->getUser(), 'id' => $character->getId()));  
            if($characters){  
                $stories = explode('::',$character->getStoryDone());  
                $chapters = [];  
                foreach($stories as $story){  
                    $story = explode(';', $story);  
                    array_push($chapters,$chapterRepo->findBy(array('label' => $story[0])));  
                }  
  
                return $this->render('story/chapters.html.twig', [  
                    'chapters' => $chapters,  
                    'character' => $character  
                ]);  
            }  
        }  
  
        $this->addFlash('danger','Ce personnage est indisponible');  
        return $this->redirectToRoute('app_story');  
    }  
    else  
    {  
        $this->addFlash('danger','Connexion requise');  
        return $this->redirectToRoute('app_login');  
    }  
}
```

Illustration 43 : Controller, route app_chapters

La sécurité supplémentaire que l'on rajoute ici est de bien vérifier que l'id du personnage que l'on soumet dans l'url de la page est bien l'id d'un personnage appartenant à l'utilisateur actuellement connecté.

Si c'est le cas, on répète la méthode de création de tableaux multidimensionnels pour cette fois ci ne récupérer que le premier élément de chaque (les labels de chaque chapitre), afin de pouvoir récupérer leurs données et de les push un à un dans un tableau vide.

Ce tableau sera ensuite envoyé en même temps que les données relatives au personnage dans le template chapters.html.twig.

```

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

{% block title %}Chapitres{% endblock %}

{% block body %}


 }})

# {{character.pj.name}}



{{character.pj.background}}



{% for classes,messages in app.flashes %}


{% for message in messages %}
{{ message }} <br>
{% endfor %}


{% endfor %}
{% for chapter in chapters %}


## {{ chapter\[0\].title }}


{% endfor %}


{% endblock %}

```

Illustration 44 : Template chapters.html.twig

Ce template est assez simpliste car, outre le fait qu'il est construit comme les autres, il n'y a qu'une boucle qui récupère les données de chaque chapitre ainsi qu'un lien qui permet de se rendre au template le plus important du site : celui qui gère la lecture de l'histoire.

e. Lecture d'un chapitre

La route app_read_chapters est la plus complexe pour le moment, car elle devra gérer en un seul template :

- La lecture initiale d'un chapitre
- La poursuite d'une lecture à n'importe quel moment du chapitre
- La relecture du chapitre après clôture

En premier lieu, on apporte une sécurité en vérifiant si l'id du personnage dans l'url correspond bien à l'un des personnages de l'utilisateur actuellement connecté.

Ensuite, on définit plusieurs variables pour une utilité future :

- \$content : on y stockera toutes les scènes et choix à faire apparaître.
- \$lastContent : les choix pas encore validés par l'utilisateur.
- \$isClosed : un booléen pour définir si le chapitre est clos ou pas.
- \$nextChapter : le prochain chapitre.

- \$storyContent : l'ensemble des choix réalisés depuis le début ainsi que les chapitres associés.

```

/**
 * @Route("/story/read/{id}/{chapter}", name="app_read_chapters")
 */

public function readChapters(Character $character = null, Chapter $chapter = null, SceneRepository $sceneRepo, ChoiceRepository $choiceRepo,
                             CharacterRepository $characRepo, ChapterRepository $chapterRepo, ChoicesService $choicesService)
{
    if($this->getUser())
    {
        $characters = $characRepo->findBy(array('user' => $this->getUser()));
        foreach($characters as $userCharacter){
            if($character && $userCharacter->getId() == $character->getId() && $chapter){
                $stories = explode(':::', $character->getStoryDone());
                $nextChapter = [];
                $isClosed = true;
                $content = [];
                $index = 0;
                $storyContent = [];
                foreach($stories as $iteration => $story){
                    $index= $iteration + 1;
                    $story = explode(';;', $story);
                    array_push($storyContent, $chapterRepo->findChapterByLabel($story[0]));
                    foreach($story as $key => $values){
                        if($story[0] == $chapter->getLabel()){
                            if($iteration == count($stories) - 1){
                                $isClosed = false;
                            } else {
                                $nextChapter = explode(';;', $stories[$index]);
                                $nextChapter = $chapterRepo->findBy(array('label' => $nextChapter[0]))[0];
                            }
                            if(strpos($values, '-SC-')){
                                array_push($content, $sceneRepo->findBy(array('label' => $values)));
                            } else if(strpos($values, '-CH-')){
                                array_push($content, $choiceRepo->findBy(array('label' => $values)));
                            }
                        }
                        if(strpos($values, '-CH-')){
                            array_push($storyContent, $choiceRepo->findBy(array('label' => $values)));
                        }
                    }
                }
                if($content) [
                    $lastChoices = $choiceRepo->findBy(array('scene' => end($content)[0]), array('label' => 'ASC'));
                    $lastContent = $choicesService->checkConstraints($lastChoices, $character);
                ]
            }
        }
    }
}

```

Illustration 45 : Première partie de la route app_read_chapters

On retrouve des similitudes sur les autres routes, avec l'utilisation des explode et des push. Les labels des scènes étant les seuls à contenir "-SC-" et les labels de choix "-CH-", il est donc possible de les retrouver grâce à ça.

Pour \$lastContent, on utilise un service qui permet de vérifier les contraintes des choix.

```

public function checkConstraints($lastChoices, $character){
    $lastContent = [];
    foreach($lastChoices as $choice){
        if($choice->getConstraints() != ""){
            $constraints = explode(";;", $choice->getConstraints());
            foreach($constraints as $constraint){
                $constraint = explode(':::', $constraint);
                if($constraint[0] == "archetype" && $constraint[1] == strtolower($character->getArchetype()->getName())){
                    array_push($lastContent, $choice);
                }
            }
        } else {
            array_push($lastContent, $choice);
        }
    }
    return $lastContent;
}

```

Illustration 46 : Fonction checkConstraints dans le service ChoicesService.php

Ce service permet de vérifier si les contraintes des choix sont respectées ou non. Pour le moment, il ne vérifie que si l'archétype du personnage est correct. Mais plus tard il serait possible de rajouter des choix qui se débloquent en fonction d'un objet dans l'inventaire ou d'un alignement spécifique.

Le template associé à cette route est séparé en deux grandes parties (comme tous les autres templates secondaires).

```
<div class="columnBox">
    <h1>Histoire</h1>
    <div class="choiceContent">
        {% for story in storyContent %}
        {% if '-CH-' in story[0].label %}
        {% set effectsChoices = story[0].effect|split(';;') %}
        {% for itemsChoices in effectsChoices %}
        {% set effectChoice = itemsChoices|split('::') %}
        {% if (effectChoice[0] == "") or (effectChoice[0] == 'alignment') %}
        <p> <span class="redText"
        {% if effectChoice[0] == 'alignment' %}
        {% if effectChoice[1] < 0 %}
        redText
        {% elseif effectChoice[1] > 0 %}
        blueText
        {% else %}
        mixColorText
        {% endif %}
        {% else %}
        mixColorText
        {% endif %}>
            {{story[0].content}}</span></p>
        {% endif %}
        {% endfor %}
        {% else %}
        <hr>
        <h3 class="text-center qwigley">{{story[0].title}}</h3>
        {% endif %}
        {% endfor %}
    </div>
</div>
```

Illustration 47 : Partie latérale gauche du template de lecture

Comme vu plus haut, on retrouve ici une condition en allant chercher les effets d'un choix, et si ceux-ci contiennent un effet d'alignement. La couleur changera en fonction d'un gain ou d'une perte d'alignement.

Tout ceci permet d'obtenir une liste des chapitres et des choix qui donnent un bon visuel de la progression du personnage.

Chapitre 1 : Une vie tranquille

Hésiter

Partir vers la forêt

Chapitre 2 : Des flammes dévorantes

Récupérer de l'eau

Vérifier maison

Attaquer

Brûler vif

Chapitre 3 : Fuite à travers les arbres

Illustration 48 : Visuel de la partie latérale

Avant de pouvoir parler de la partie contenu, il est nécessaire d'expliquer l'élaboration d'une scène et la manière dont elle est écrite en BDD.

Plus tôt, il a été mentionné qu'une histoire était une succession de textes descriptifs, de dialogues et parfois de messages adressés directement à l'utilisateur.

Afin de pouvoir différencier l'un ou l'autre, tout en imbriquant l'ensemble dans l'attribut de la scène, nous allons utiliser la même logique que pour story_done.

Chaque message différent sera séparé par des “::”. Les dialogues commencent toujours par des double guillemets et les messages hors histoire commenceront soit pas & ou par @ en fonction de la couleur que nous voudrions lui donner.

Ainsi, en utilisant le filtre twig |split('::'), nous allons récupérer chaque portion de texte qui se différencient, permettant de leur appliquer le style adéquat.

```

<div class="contentBox">
    {% for classes, messages in app.flashes %}
        <div class="text-center alert alert-{{ classes }}>
            {% for message in messages %}
                {{ message }} <br>
            {% endfor %}
        </div>
    {% endfor %}
    <h2 class="qwigleyTitle">{{ content[0][0].chapter.title }}</h2>
    {% for item in content %}
        {% if '-CH-' in item[0].label %}
            {% set effects = item[0].effect|split(';;') %}

            {% for items in effects %}
                {% set effect = items|split('::') %}
                {% if (effect[0] == "") or (effect[0] == 'alignment') %}
                    <button class="button noHoverShadow noCursor
                {% if effect[0] == 'alignment' %}
                    {% if effect[1] < 0 %}
                        redGradient
                    {% elseif effect[1] > 0 %}
                        blueGradient
                    {% else %}
                        mixGradient
                    {% endif %}
                {% else %}
                    mixGradient
                {% endif %}">
                    {{ item[0].content }}
                </button>
                {% endif %}
            {% endfor %}
            {% else %}
                <div class="textBox" {% if loop.last %} id="ancre" {% endif %}>
                    {% set texts = item[0].content|split('::') %}
                    {% for text in texts %}
                        {% if text|first == "" %}
                            <span class="qwigleyP">{{text|nl2br}}</span>
                        {% elseif text|first == "@" %}
                            <span class="blueText italic">{{text[1:]|nl2br}}</span>
                        {% elseif text|first == "&" %}
                            <span class="redText italic">{{text[1:]|nl2br}}</span>
                        {% else %}
                            <span>{{text|nl2br}}</span>
                        {% endif %}
                    {% endfor %}
                </div>
            {% endif %}
        {% endfor %}
    {% endfor %}

```

Illustration 49 : Partie contenu du template `read-chapters.html.twig`

A la fin de ce template, nous avons aussi la liste des boutons qui évolueront en fonction de l'état actuel du chapitre. S'il est considéré clôt, un bouton 'Chapitre suivant' apparaîtra.

Si la dernière scène ne propose aucun choix, on propose de fermer le chapitre.

Sinon, on fait apparaître l'ensemble des choix sans style particulier.

Nous pouvons constater ici exactement la même logique de style en ce qui concerne l'apparence des choix déjà réalisés par l'utilisateur.

Comme mentionné précédemment, on peut voir ici les différentes conditions permettant de changer le style du contenu de l'attribut `content`.

On ajoute aussi une condition dans le cas d'un dernier tour de boucle pour ajouter une `id="ancre"`.

Cela permet à chaque validation d'un nouveau choix de scroll automatiquement au niveau de la dernière zone de texte ajoutée.

En ce qui concerne les textes hors histoire, on ne récupère que le contenu du texte et non le marqueur & ou @, afin d'obtenir un meilleur visuel côté utilisateur.

```

<div class="buttonBox">
  {% if isClosed %}
    <a href="{{path('app_read_chapters', { id: character.id, chapter: nextChapter.id, _fragment: 'ancre' })}}" class="button shadows mixGradient">Chapitre suivant</a>
  {% else %}
    {% if choices is empty %}
      <a href="{{path('app_endChapter', {character: character.id})}}" class="button shadows mixGradient">Fin du chapitre</a>
    {% else %}
      {% for choice in choices %}
        <a href="{{path('app_choices', {id: choice.id, character: character.id})}}" class="button shadows mixGradient">{{ choice.content }}</a>
      {% endfor %}
      {% endif %}
      {% endif %}
</div>

```

Illustration 50 : Fin du template read-chapters.html.twig

f. Validation des choix et des chapitres

Afin de contrer les utilisateurs abusant de l'url, il est nécessaire de mettre en place de nombreux dispositifs dans les routes de validation des choix et des chapitres. Cela consiste en des conditions vérifiant l'id du personnage, du choix ainsi que des différents choix possibles par le personnage en question en fonction de son story_done. (Illustration 51)

Une fois les vérifications opérées, il est nécessaire par la suite de vérifier si le choix en question respecte les contraintes mises en place au préalable.

Une fois que toutes les vérifications sont terminées, on vérifie s'il y a plusieurs effets pour ce choix en particulier. En fonction de ces effets, on opère les changements adéquats avant de sauvegarder en BDD.

Pour terminer, on complète story_done du personnage avec le label du choix suivi de la scène associée à ce choix. Etant donné que la lecture d'un chapitre dépend de story_done, le mettre à jour changera donc le template de lecture des chapitres.

En ce qui concerne la vérification des chapitres, cela s'opère lors du choix de l'utilisateur de clôturer le chapitre.

Il faut vérifier au préalable que le chapitre est en état pour être clôturé. On vérifie donc si le trigger d'un chapitre correspond au dernier label d'une scène de story_done (Illustration 52)

Si c'est le cas, on récupère le label du prochain chapitre ainsi que de la scène qui le suit. Puis on met à jour story_done.

```

public function validateChoice(Choice $choice = null, Character $character = null, EntityManagerInterface $manager, CharacterRepository $characRepo, ChoiceRepository $choiceRepo,
SceneRepository $sceneRepo, ChapterRepository $chapterRepo, ChoicesService $choiceService, ItemRepository $itemRepo, InventorySlotRepository $inventorySlotRepo){
if($this->getUser())
{
    $characters = $characRepo->findBy(array('user' => $this->getUser()));
    foreach($characters as $userCharacter){
        if($choice && $character && $userCharacter->getId() == $character->getId())[<
            $storyDone = $character->getStoryDone();
            $story = explode(':', $storyDone);
            $lastStoryLabel = explode(';', end($story));
            $nextStory = $sceneRepo->findBy(array('label' => end($lastStoryLabel)));
            $possibleChoices = $choiceRepo->findBy(array('scene' => end($lastStory)), array('label' => 'ASC'));
            foreach($possibleChoices as $possibleChoice){
                if($choice->getLabel() == $possibleChoice->getLabel()){
                    $validateChoice = $choiceService->checkValidationConstraintChoice($choice, $character);
                    if($validateChoice){
                        $actualChoice = $choice->getLabel();
                        $nextStory = $choice->getNextStory();
                        $effects = $choice->getEffect();
                        if($effects){
                            $seffect = explode(';', $effects);
                            foreach($seffect as $key => $values){
                                $array = explode('::', $values);
                                if($array[0] == 'alignment'){
                                    $character->setAlignment($character->getAlignment() + $array[1]);
                                } else if ($array[0] == 'pv'){
                                    $character->setPv($character->getPv() + $array[1]);
                                } else if($array[0] == 'objet'){
                                    $objetDetails = explode(' ', $array[1]);
                                    $item = $itemRepo->findBy(array('name' => $objetDetails[0]));
                                    $itemSlot = $inventorySlotRepo->findBy(array('item' => $item, 'inventory' => $character->getInventory()));
                                    if(!$itemSlot){
                                        $itemSlot = new InventorySlot();
                                        $itemSlot->setInventory($character->getInventory());
                                        $itemSlot->setItem($item[0]);
                                        $itemSlot->setQuantity($objetDetails[1]);
                                    } else {
                                        $itemSlot[0]->setQuantity($itemSlot[0]->getQuantity() + $objetDetails[1]);
                                    }
                                    $manager->persist($itemSlot);
                                    $manager->flush();
                                }
                            }
                        }
                    }
                }
            }
        }
        $storyDone .= ";$actualChoice;$nextStory";
        $chapter = $chapterRepo->findBy(array('label' => $lastStoryLabel[0]));
        $character->setStoryDone($storyDone);
        $manager->persist($character);
        $manager->flush();
        return $this->redirectToRoute('app_read_chapters', array('id' => $character->getId(), 'chapter' => $chapter[0]->getId(), '_fragment' => 'ancre'));
    }
}

```

Illustration 51 : Partie principale de la route app_choices

```

/**
 * @Route("/story/validate/chapter/{character}", name="app_endChapter")
 */

public function validateChapter(Character $character = null, ChapterRepository $chapterRepo, EntityManagerInterface $manager,
SceneRepository $sceneRepo, CharacterRepository $characRepo){

    if($this->getUser())
    {
        $characters = $characRepo->findBy(array('user' => $this->getUser()));
        foreach($characters as $userCharacter){
            if($character && $userCharacter->getId() == $character->getId())[<
                $storyDone = $character->getStoryDone();
                $chapter = explode(':', $storyDone);
                $chapter = explode(';', end($chapter));
                $trigger = $chapterRepo->findTrigger(end($chapter));
                if($trigger){
                    $chapterData = $trigger[0];
                    $nextChapter = $chapterData->getLabel();
                    $nextScene = $sceneRepo->findBy(array('chapter' => $chapterData))[0]->getLabel();
                    $storyDone .= ":$nextChapter;$nextScene";
                    $character->setStoryDone($storyDone);
                    $manager->persist($character);
                    $manager->flush();
                    return $this->redirectToRoute('app_chapters', ['id' => $character->getId()]);
                }
            }
        }
        $this->addFlash('danger', "Ce chapitre ne peut être clôturé.");
        return $this->redirectToRoute('app_story');
    }
    else
    {
        $this->addFlash('danger', "Connexion requise");
        return $this->redirectToRoute('app_login');
    }
}

```

Illustration 52 : Controller, route app_endChapter

g. Template d'erreurs

Plusieurs templates d'erreurs ont été créés en cas d'erreur 404 ou 403, ainsi qu'un template général d'erreur.

6] Partie Administrateur

La partie administrateur est composée de plusieurs templates qui fonctionnent de la même manière :

- Un menu permettant de naviguer entre les scènes, choix, utilisateur, etc...
- Chaque sous-catégorie dispose d'un affichage sous forme de tableau avec un filtre permettant de trier rapidement ce que l'on souhaite obtenir (surtout pour retrouver des scènes via leur label)
- Chaque élément peut être modifié (uniquement les choses qui ne casseront pas la logique ou l'architecture de l'histoire)
- La visualisation de chaque élément se fait sous la forme d'un formulaire où certains champs sont grisés (généralement les labels)

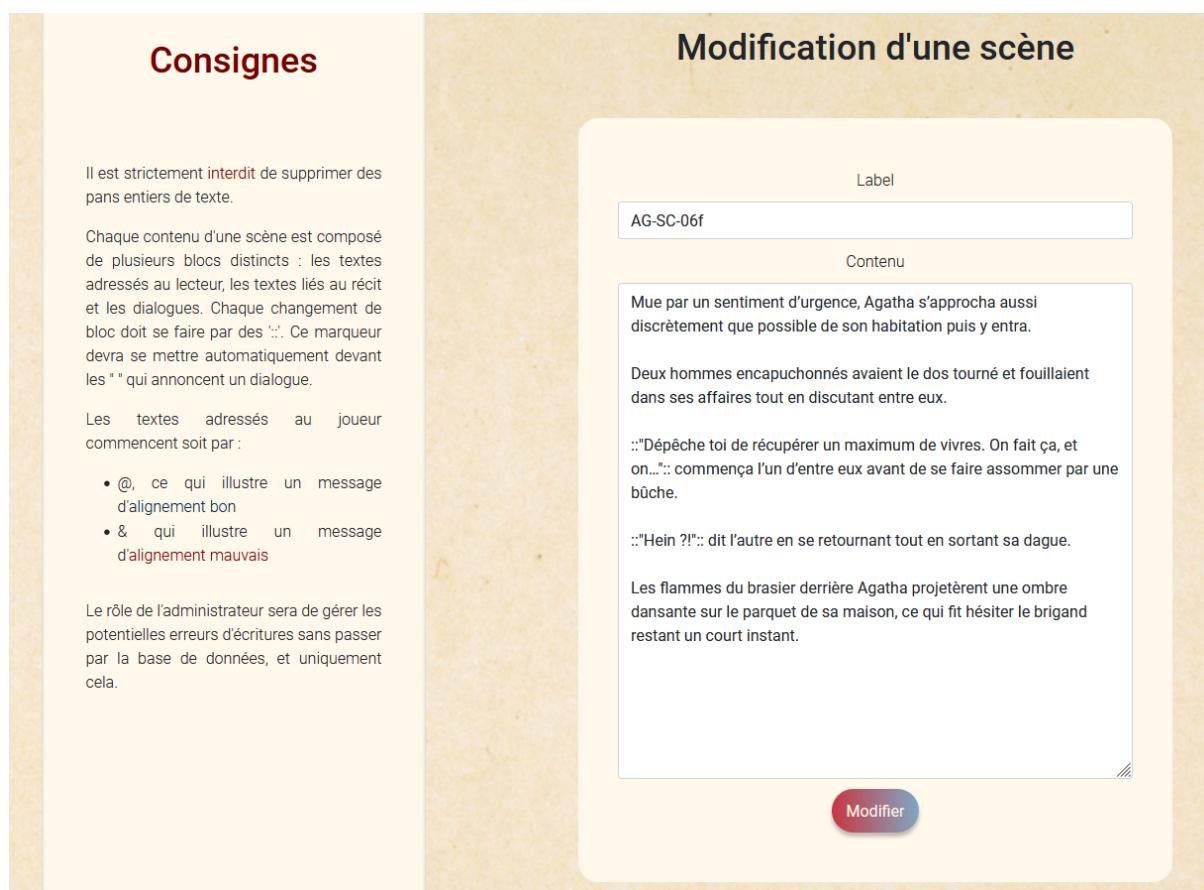


Illustration 53 : Exemple du visuel de la modification d'une scène

On peut remarquer sur la précédente illustration la structure du texte expliquée plus tôt.

Partie IV : Mise en production et futur

1] Mise en production

La mise en production a été réalisée via Heroku. Afin d'éviter un soucis de restrictions de requêtes avec la BDD, il a été nécessaire de fournir un abonnement de 7.00 €. De plus, un add on ClearDB MySQL en version gratuite a été utilisé pour faire le lien avec notre BDD. Comme expliqué au début du rapport, des variables d'environnements ont été utilisées pour dissimuler les codes d'accès et les clés afin d'éviter un accès extérieur à la BDD.

Suite à des problèmes avec la BDD fournie par Heroku (qui n'acceptait pas le format Json pour l'attribut rôle du User), des modifications drastiques ont dû être opérées dans la majorité des routes afin de vérifier si l'utilisateur disposait des droits ou non pour accéder à certaines parties du site.

De plus, la méthode getRoles a nécessité quelques ajustements afin que tout refonctionne comme il le faut.

Malgré la présence de certificats de sécurité, il existe encore des problèmes pour lier le site à des mots clés Google. Ceci devra être réalisé dans un avenir proche, afin de vérifier la stabilité du site ainsi que son SEO.

2] Mises à jour futures

Pour le moment, seulement Agatha dispose d'une partie de son histoire. Il sera donc nécessaire d'en écrire davantage pour elle-même et commencer celle des autres. Le processus d'écriture couplé avec la logique très spéciale de hiérarchisation du récit nécessite un temps très important.

La gestion des labels permettant la fonctionnalité de l'ensemble du site, il est nécessaire d'opérer un travail minutieux de préparation et d'écriture, ce qui peut vite devenir laborieux.

Par exemple, le chapitre 2A d'Agatha dispose pour lui tout seul de 23 scènes différentes.

A l'avenir, il faudra mettre en place des choix qui seront disponibles en fonction d'objets présents dans l'inventaire ou de l'alignement actuel du personnage. Peut être même un choix si un choix spécifique a été réalisé précédemment ? Un système plus poussé de l'inventaire du personnage, afin d'avoir une trace de ce qu'il contient.

Ce projet étant vaste, il était nécessaire de définir des limites. Ces limites pourront désormais être repoussées au fil du temps, permettant la création d'une histoire dynamique.