



Dossier projet
RNCP 37873
Concepteur développeur d'applications

AI Story



Juanita Afanador Kowalski
Juillet 2024



Sommaire

2. Mon projet face aux compétences attendues

1. Description du Projet et Introduction	
a. Présentation du projet en français.....	4
b. Présentation du projet en anglais.....	5
2. Mon projet face aux Liste de Compétences du Référentiel Attendues	
a. Bloc 1 : Développer une application sécurisée.....	6
i. Configuration de l'Environnement de Travail	
ii. Sécurité et Bonnes Pratiques	
iii. Développement des Interfaces Utilisateur (GUI)	
iv. Développer des composants métier	
v. Contribuer à la gestion d'un projet informatique	
b. Bloc 2 : Concevoir et développer une application sécurisée organisée en couches	10
i. Analyser les besoins et maquetter une application	
ii. Définir l'architecture logicielle d'une application	
iii. Concevoir et mettre en place une base de données relationnelle	
iv. Développer des composants d'accès aux données SQL	
c. Bloc 3 : Préparer le déploiement d'une application sécurisée	13
i. Préparer et exécuter les plans de tests d'une application	
ii. Préparer et documenter le déploiement d'une application	
iii. Contribuer à la mise en production dans une démarche DevOps	
3. Réalisation Technique en détails	
a. Cahier de Charges	
i. Expression des Besoins	14
1. Persona	
2. User Story	
3. Fonctionnalités Essentielles pour le MVP	
ii. Risques et Difficultés	15
1. Analyse des risques et identification des principales difficultés rencontrées	
b. Spécifications Fonctionnelles	
i. Modélisation.....	16
1. Diagramme d'activité à partir de la landing page.	
2. Diagramme d'activité quand l'utilisateur s'est identifié.	
ii. Conception de la Base de Données	18



1. Modèle conceptuel des données (MCD)	
2. Modèle Logique des Données (MLD)	
3. Modèle Physique des Données (MPD)	
iii. Maquette de l'Interface	20
c. Spécifications Techniques du Projet	23
d. Gestion de Projet	25
i. Planning et Suivi	25
ii. Environnement Humain et Technique	25
1. Description des ressources humaines	
2. Description des ressources techniques utilisées	
iii. Objectifs de Qualité	28
e. Réalisation	
i. Contexte	29
ii. Langages Utilisés	29
iii. Réalisation des Services	29
1. Le Front-End	
2. Le Back-End	
3. La base de données	
4. Service tiers: API Open AI	
5. Le déploiement	
f. Tests	
i. Exemple: Test de la page de Profil	53
g. Veille	
i. Activités de veille sur les vulnérabilités de sécurité	56
4. Défis et Recherche	57
5. Conclusion	
a. Axes d'amélioration	58



1. Description du Projet et Introduction

a. Présentation du projet en français

AI Story : Une Aventure Interactive de Crédit d'Histoires Personnalisées avec l'IA

À l'ère du numérique, l'intelligence artificielle (IA) a révolutionné notre façon de créer et d'interagir avec les contenus. "AI Story" illustre parfaitement cette transformation en offrant une plateforme où chacun, adulte comme enfant, peut créer des histoires personnalisées. En tant que maman passionnée par l'éveil créatif et littéraire, j'ai constaté que les récits interactifs enrichissent l'imagination et les choix de mon fils. Inspirée par cette magie et motivée par ma curiosité pour les avancées de l'IA, comme celles offertes par ChatGPT, j'ai conçu ce projet pour fusionner le charme de la narration avec la puissance de la technologie moderne.

AI Story utilise un ensemble de technologies modernes et robustes, incluant Next.js et Tailwind pour le front-end, ainsi que Node.js avec Express pour le back-end, le tout soutenu par une base de données relationnelle. Ce choix stratégique m'a permis de renforcer mes compétences techniques tout en augmentant mon attractivité sur le marché du travail.

J'ai réalisé ce projet de manière autonome, avec le soutien des mentors d'Ada Tech School et de professionnels expérimentés de mon entreprise. En adoptant la méthode Kanban, j'ai structuré le développement de manière efficace, assurant un progrès clair et mesurable à chaque étape : du développement front-end et back-end, à la gestion des données et au déploiement.

Mon objectif est de partager cette plateforme avec mon fils, enrichissant son apprentissage et sa découverte. À terme, AI Story pourrait devenir un service premium, offrant des histoires imprimables et personnalisées avec des illustrations générées par IA, pour captiver une audience plus large. Ce projet est non seulement un voyage personnel dans le monde du développement web, mais il aspire également à redéfinir la façon dont nous concevons et partageons des histoires à l'ère numérique.



b. Présentation du projet en anglais

AI Story: An Interactive Adventure in Personalized Story Creation with AI

In today's digital age, artificial intelligence (AI) has revolutionized how we create and interact with content. "AI Story" embodies this transformation by providing a unique platform where both children and adults can craft their personalized stories. As a mother passionate about nurturing creative and literary development, I've observed the positive impact of interactive stories on my son's imagination and decision-making. Inspired by this wonder and driven by my curiosity about AI advancements, such as those offered by ChatGPT, I designed this project to merge the art of storytelling with the power of modern technology.

AI Story was developed using a robust, contemporary tech stack, including Next.js and Tailwind for the front end, and Node.js with Express for the back end, supported by a relational database. This strategic choice has not only deepened my technical knowledge, but also made me more competitive in the job market.

I managed this project autonomously, with valuable guidance from mentors at Ada Tech School and experienced professionals at my apprenticeship company. Adopting the Kanban method allowed me to structure development efficiently, ensuring clear and measurable progress at every critical phase: front end, back end, data management, and deployment.

My primary goal is to share this platform with my son, thereby enriching his journey of learning and discovery. In the future, AI Story aims to evolve into a premium service, offering printable and customizable stories with AI-generated illustrations to captivate and inspire a broader audience. This project represents not only a personal journey into web development but also aspires to redefine how we conceive and share stories in the digital age.



2. Mon projet face aux Liste de Compétences du Référentiel Attendues

Pour permettre au jury d'apprécier mon projet en un coup d'œil, je confronterai, dans cette partie, les compétences attendues pour la validation du RNCP avec le travail que j'ai réalisé dans le cadre de mon projet.

a. Bloc 1 : Développer une application sécurisée

i. Configuration de l'Environnement de Travail

Pour assurer une productivité optimale et un développement efficace, j'ai opté pour un environnement Mac reconnu pour sa stabilité et sa performance. Comme environnement de développement intégré (IDE), j'ai choisi Visual Studio Code (VSCode), apprécié pour sa légèreté, sa flexibilité, et son large éventail d'extensions qui enrichissent le développement.

Extensions VSCode

L'utilisation de plugins spécifiques dans VSCode a été cruciale pour améliorer ma productivité et la qualité de mon code. Parmi ces extensions, je souligne :

- **Jest** pour l'intégration de tests unitaires et fonctionnels, garantissant la fiabilité de l'application.
- **ESLint et Prettier** pour assurer la cohérence du style de codage et le formatage automatique.
- **GitLens, Git Graph, et GitHub Actions** pour une gestion de version avancée et une intégration continue.
- **Tailwind CSS IntelliSense** pour une conception visuelle rapide et réactive.
- **Rest Client** pour tester les API directement depuis VSCode, simplifiant le processus de développement backend.
- **Remote - SSH** pour manipuler mon serveur distant lors du déploiement directement depuis l'IDE.

Visualisation du projet

Pour avoir une vision globale du projet, formuler comment techniquement réaliser une user story ou comprendre le fonctionnement d'une librairie, j'ai eu besoin de créer plusieurs diagrammes qui m'ont permis de comprendre comment je devais développer les solutions techniques nécessaires. Pour cela j'ai utilisé **Figma, MockUp, Scalidraw et Whimsical**.



Gestion de Version avec Git et GitHub

L'adoption de Git et GitHub a été essentielle pour le versionnement du code. Ces outils m'ont permis de travailler avec flexibilité depuis plusieurs postes, tout en assurant une intégrité et une traçabilité du code. J'ai également utilisé GitHub pour marquer des moments importants dans le développement du projet avec des **tags**, mettant en évidence les différentes versions. De plus, j'ai pu mettre en place des **GitHub Actions** (voir page 54) pour déclencher automatiquement les tests unitaires avant chaque déploiement, évitant ainsi l'introduction de bugs dans les fonctionnalités lors de l'évolution de l'application.

Gestion de Packages et Environnement de Développement

L'utilisation de **GitHub Dependabot** m'a permis de veiller sur les mises à jour nécessaires pour garantir la sécurité de l'application. L'utilisation de **Homebrew et NPM** a facilité la gestion des packages et des outils nécessaires, comme **Node.js et Next.js**, renforçant ainsi l'efficacité de l'environnement de développement et la sécurité de l'application.

Base de Données

Ayant travaillé auparavant dans des projets avec **MongoDB**, j'ai voulu me confronter à la mise en place et au développement d'une base de données relationnelle que je n'avais pas utilisée par le passé. Le choix de **PostgreSQL** pour la base de données reflète une priorité donnée à la robustesse et à la fiabilité fournie par celle-ci, et à ma volonté de la découvrir car son utilisation est assez répandue dans le secteur(voir page 54).

Stack Technique et Intégration d'Intelligence Artificielle

Pour le développement de "AI Story", j'ai sélectionné une stack technique sur laquelle je voulais apprendre et qui allait répondre aux besoins techniques pour mener à bien le projet. Mon choix de **TypeScript** comme langage principal offre un cadre solide pour le développement d'applications complexes avec un typage statique qui facilite la maintenance et la robustesse du code. L'intégration de **Tailwind CSS** a été une évidence pour le design, me permettant de personnaliser rapidement l'application sans compromettre la réactivité ni l'accessibilité.(voir page 54).

Le front-end est construit avec **Next.js** qui utilise **React**. Cette stack technique devient de plus en plus populaire et il me semblait pertinent de la connaître davantage. (voir page 54).



Côté back-end, bien que **Next.js** propose une solution intégrée pour gérer le back-end, j'ai opté pour une séparation entre le back et le front. Pour ce faire, **Node.js couplé à Express** assure une flexibilité et une évolutivité maximales à long terme. ([voir page 54](#)).

Dans le cas de la base de données, je me suis orientée vers **PostgreSQL**, connue pour sa robustesse et richesse en fonctionnalités capable de soutenir l'évolution du projet. ([voir page 54](#)).

Enfin, la principale innovation de ce projet et le cœur de celui-ci est l'intégration de l'**API d'OpenAI**, permettant la fonctionnalité principale du projet : la création d'histoires personnalisées assistée par IA. Avec le boom des IA, j'ai voulu en apprendre davantage et comprendre leur utilisation. ([voir page 54](#)).

ii. Sécurité et Bonnes Pratiques

La sécurité et le respect des bonnes pratiques de développement sont des piliers essentiels de la conception et de l'implémentation de "AI Story". J'ai utilisé, dès le début du projet, la librairie **Dotenv** pour une gestion sécurisée des variables d'environnement, évitant leur exposition sur GitHub et assurant la confidentialité des informations sensibles comme les clés pour la connexion à la base de données ou celles pour l'**API**. Cette méthode prévient toute fuite accidentelle de données critiques et évite d'inscrire en dur les configurations importantes dans le code source, en ligne avec les standards de sécurité les plus stricts.

Pour renforcer la sécurité, j'ai fait appel à plusieurs librairies. Dans le cas de l'authentification, j'ai utilisé des bibliothèques comme **Bcrypt** et **JWT (JSON Web Token)**. Pour protéger le back-end d'une sollicitation excessive, et en particulier l'**API d'OpenAI**, j'ai mis en place un **rate limiter**. Enfin, pour la validation des entrées des utilisateurs et éviter les injections SQL, j'ai employé des bonnes pratiques de validation. ([voir page 54](#)).

3. Développement des Interfaces Utilisateur (GUI)

J'ai conçu l'interface utilisateur avec l'application **MockUp** qui m'a permis de réaliser le wireframe et d'avoir une idée des composants qui devaient être créés. Pour le développement, j'ai choisi de travailler avec le framework **Next.js** qui utilise la bibliothèque **React**. Le but étant d'apprendre davantage sur celui-ci par la pratique, car il devient de plus en plus populaire par sa capacité à effectuer des rendus côté serveur. Au niveau visuel, j'ai voulu travailler avec une palette de couleurs qui garantisse l'accessibilité.



a. Développer des composants métier

Plusieurs composants métier ont été réalisés pour garantir le bon fonctionnement de l'application comme :

- **Formulaire de création de compte utilisateur** : Composant pour la création, y compris la gestion des informations de profil.
- **Formulaire d'authentification** : Composant pour accéder à l'espace personnel de l'utilisateur.
- **Formulaire de création d'histoire personnalisée** : Composant pour la création et la gestion des formulaires permettant aux utilisateurs de soumettre des informations pour générer des histoires personnalisées.
- **Génération d'histoires personnalisées** : Composant pour traiter les données soumises via le formulaire et générer des histoires personnalisées en fonction des préférences et des réponses des utilisateurs.
- **Gestion des histoires** : Composant permettant aux utilisateurs de stocker, visualiser, modifier et supprimer les histoires personnalisées générées.
- **Sécurisation et gestion des sessions** : Composant pour gérer l'authentification, la sécurité des données utilisateur et la gestion des sessions de connexion.

b. Contribuer à la gestion d'un projet informatique

En tant que seule développeuse du projet, j'ai endossé la responsabilité complète de sa gestion. Pour suivre l'évolution du projet, j'ai utilisé l'application **Notion** pour la prise de notes et comme journal et l'application **Trello** pour avoir un tableau en mode Kanban pour décomposer chaque étape en tickets. Cela m'a permis de centraliser les informations, d'avoir une vision globale du travail à réaliser et de réorganiser mon flux de tâches selon l'évolution et les difficultés que je rencontrais. L'utilisation d'un outil de gestion de projet comme **Trello** a été indispensable pour assurer une progression ordonnée et pour ne rien omettre dans le développement et l'amélioration de l'application. J'ai pu également l'utiliser pour donner de la visibilité sur mon travail auprès de mes encadrants à l'école et tuteurs en entreprise. ([voir page 54](#)).



b. Bloc 2 : Concevoir et développer une application sécurisée organisée en couches.

Pour choisir le thème du projet, j'ai essayé de répondre à une problématique que j'ai rencontrée en tant que parent d'un enfant en bas âge : comment utiliser les nouvelles technologies pour introduire le plaisir de la lecture et intéresser l'utilisateur en lui donnant davantage de contrôle dans la création.

Avec l'arrivée des **LLM (Large Language Models)** en ligne grâce à des API, j'ai voulu voir si leur utilisation me permettrait de répondre à ma problématique et de réaliser mon projet. J'ai donc réalisé un **PoC (Proof of Concept)** pour valider la viabilité du projet ([voir page 54](#)). J'ai créé un fichier en **Python** me permettant d'interroger l'**API d'OpenAI** avec un prompt simple et de vérifier si le LLM d'OpenAI était suffisamment performant à ce moment-là. J'avais aussi essayé de le réaliser en **TypeScript**, mais la version du package à ce moment-là avait des problèmes avec le typage, problème qui a été résolu avec la mise à jour suivante. Le résultat du PoC étant satisfaisant, j'ai commencé à structurer mon projet.

i. Analyser les besoins et maquetter une application

Pour commencer le projet, j'ai défini le **Persona** et le **User Story** du projet. Pour saisir pleinement le parcours utilisateur d'AI Story et les exigences techniques qui en découlent, j'ai élaboré plusieurs diagrammes et maquettes : diagramme du parcours de l'utilisateur, de la base de données, de l'authentification et du flux de données. Ceci m'a permis d'avoir de la clarté sur ce que je devais développer, définir le **MVP (minimum viable product)** et dans quel ordre j'allais construire le projet. ([voir page 54](#)).

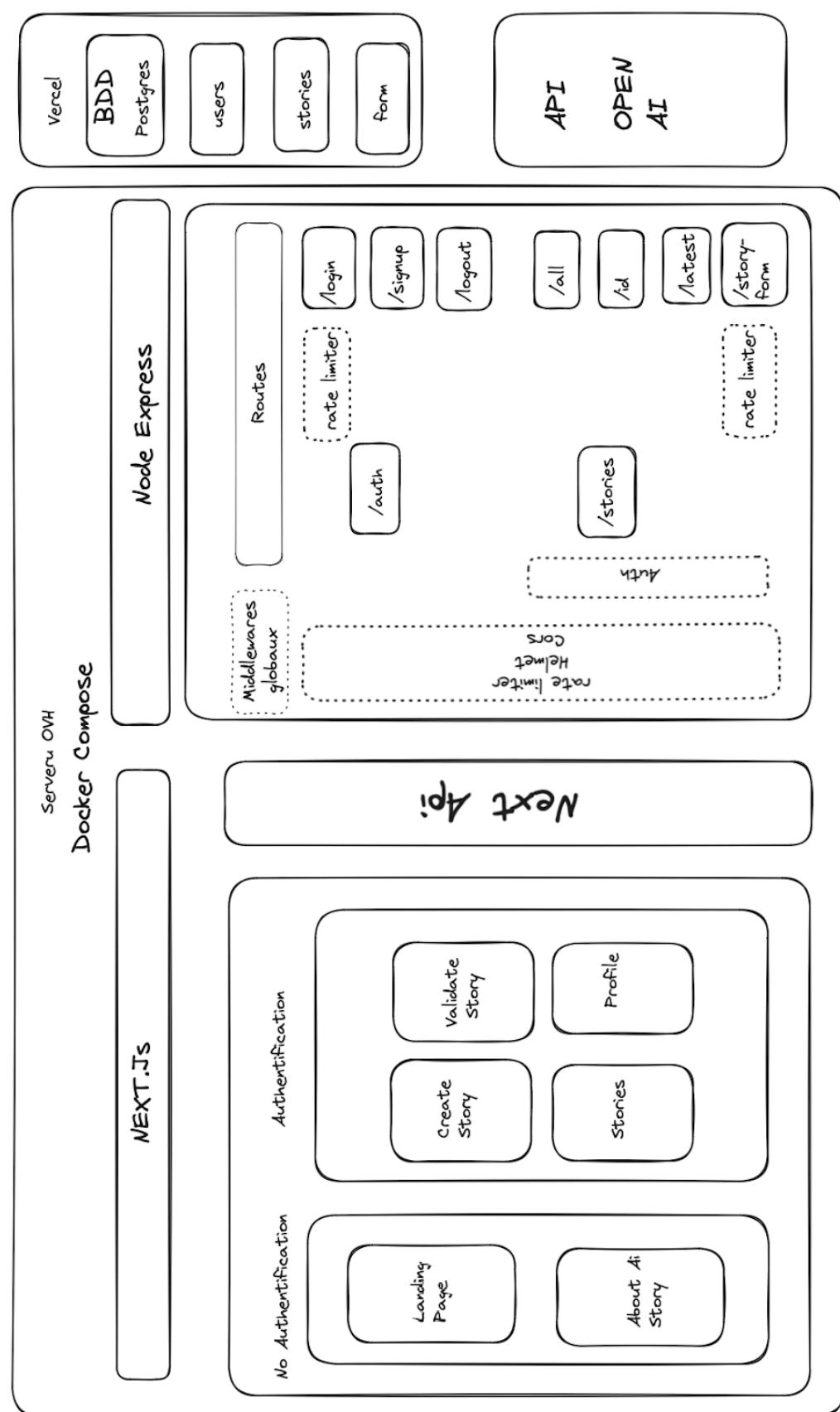
ii. Définir l'architecture logicielle d'une application

Une fois que j'ai pu définir le MVP, j'ai établi la stack technique et l'architecture du projet.



Back End

Front-End





iii. Concevoir et mettre en place une base de données relationnelle

J'ai réalisé un diagramme **UML (Unified Modeling Language)** lorsque j'ai dû définir quelles étaient les données à sauvegarder. Ensuite, j'ai réalisé un script pour créer la base de données avec **PostgreSQL**. ([voir page 54](#)).

iv. Développer des composants d'accès aux données SQL

Pour chaque interaction du back-end avec la base de données dans le cadre du projet, j'ai créé des fonctions dédiées pour effectuer les requêtes nécessaires. Souhaitant approfondir mes connaissances en bases de données relationnelles et en utilisation du **SQL (Structured Query Language)** pour manipuler ces dernières, j'ai décidé de ne pas utiliser d'**ORM (Object-relational mapping)**. À la place, j'ai développé un script pour créer la base de données et j'ai également conçu des fonctions spécifiques pour exécuter des requêtes SQL directement sur la base de données **PostgreSQL**. Cette approche a été choisie spécifiquement pour répondre aux exigences du front-end et pour me permettre d'acquérir une expérience pratique et approfondie de la gestion directe des bases de données. ([voir page 54](#)).

Voici quelques exemples des fonctions utilisées :

```
export const postNewStory = async (openAiStory: string, user_id: string) => {
  const pool = await poolPromise;
  const query =
    "INSERT INTO public.story_created (user_id, story) VALUES ($1, $2)";
  const values = [user_id, openAiStory];

  try {
    const result = await pool.query(query, values);
    return result;
  } catch (error) {
    throw error;
  }
};

export const getAllStories = async (user_id: string): Promise<NewStory> => {
  try {
    const pool = await poolPromise;
    const query =
      "SELECT * FROM public.story_created WHERE user_id = $1 ORDER BY created_at DESC";
    const result = await pool.query(query, [user_id]);
    return result.rows;
  } catch (error) {
    throw error;
  }
};
```



c. Bloc 3 : Préparer le déploiement d'une application sécurisée

i. Préparer et exécuter les plans de tests d'une application

J'ai réalisé des tests unitaires dans les composants principaux et, avant d'envoyer le code sur GitHub, je faisais les tests en local. Toutefois, lors de chaque commit, la totalité des tests était exécutée grâce au script de **GitHub Actions** qui était déclenché. Si un des tests ne passait pas, j'étais directement informée par mail. Une fois l'application redéployée, je vérifiais que les comportements de base de l'application, comme l'authentification et la création des histoires, étaient complètement fonctionnels.

ii. Contribuer à la mise en production dans une démarche DevOps

Pour réaliser le déploiement de l'application, j'ai voulu utiliser les serveurs de l'école **Ada Tech School**. Ce sont des serveurs distants accessibles via **SSH**, et c'était l'occasion de lancer un projet sur ce type de serveur. Pour comprendre l'organisation du serveur partagé, nous avons eu un atelier qui expliquait comment se connecter, la structure du serveur, où trouver la documentation, ainsi que les contraintes pour pouvoir l'utiliser, telles que l'adresse, le mot de passe et les commandes basiques pour naviguer dans le shell du serveur. Par la suite, j'ai lu la documentation dans le serveur et j'ai documenté ces informations pour pouvoir réaliser ultérieurement le déploiement de l'application.

Étant donné que c'est un serveur partagé et que chaque projet utilise des technologies différentes, il est obligatoire d'utiliser **Docker** pour le déploiement. Je voulais également en apprendre davantage sur la technologie derrière Docker et savoir comment créer un fichier **Docker Compose** avec le back-end et le front-end dans le même conteneur. Pour faciliter la manipulation des fichiers, en particulier l'ajout du dossier `.env`, j'ai utilisé une extension sur **Visual Studio Code** qui s'appelle **Remote SSH**. L'extension permet d'utiliser n'importe quelle machine distante avec un serveur SSH comme environnement de développement. J'ai pu ainsi accéder au serveur, voir dans le fichier des ports du serveur en cours d'utilisation. Par la suite, j'ai dû demander à avoir mon URL avec le protocole **https** et non **http** car je savais que **Next.js** risquait de générer un problème de **CORS**. Enfin, j'ai créé un dossier pour y mettre un clone de mon projet, ajouter les variables d'environnement et changer les ports dans les fichiers Docker. J'ai pu ainsi lancer le **Docker Compose** et déployer l'application. ([voir page 54](#)).



3. Réalisation Technique en détails

1. Cahier de Charges

i. Expression des Besoins

Pour définir les besoins essentiels du projet, j'ai défini le **Persona** ou utilisateur idéal.

Ensuite, j'ai défini le contexte d'utilisation au travers du **User Story** et les fonctionnalités essentielles pour répondre aux besoins de mon utilisateur idéal.

1. Persona

Profil : Parent d'un enfant en bas âge

Contexte : Ce parent aime lire des histoires à son enfant. Il préfère les histoires personnalisées où l'enfant devient le héros. Il utilise la technologie pour rendre ces moments plus magiques et interactifs.

2. User Story

En tant que parent, je veux utiliser **AI Story** pour créer et lire des histoires personnalisées pour mon enfant, afin de rendre le moment du coucher spécial et engageant.

3. Fonctionnalités Essentielles pour le MVP

1. **Page d'accueil** : Présentation de la plateforme et des fonctionnalités.
2. **Page d'inscription** : Inscription des nouveaux utilisateurs.
3. **Page d'authentification** : Connexion des utilisateurs existants.
4. **Page de création d'histoire** : Formulaire pour renseigner les informations nécessaires à la création de l'histoire.
5. **Page d'histoires** : Accès à toutes les histoires personnalisées de l'utilisateur.
6. **Modale¹ d'édition de titre** : Modification du titre des histoires.
7. **Modale de visualisation et suppression d'histoire** : Visualisation et possibilité de supprimer une histoire.
8. **Page de validation d'histoire** : Validation ou refus des histoires générées par l'IA.
9. **Page de compte utilisateur** : Accès aux informations du compte utilisateur.

¹ Les fenêtres modales sont des fenêtres qui apparaissent directement à l'intérieur de la fenêtre courante du navigateur, au-dessus de la page web qui les appelle. Source:
<https://www.accede-web.com/notices/interface-riche/fenetres-modales/>



i. Risques et Difficultés

i. Analyse des risques et identification des principales difficultés rencontrées

Le principal risque que j'ai eu lors de l'idée du projet était de savoir si les capacités de l'**API** étaient suffisamment bonnes pour fournir des histoires personnalisées correctes. Avant de me lancer dans le projet, j'ai suivi le conseil d'une des encadrantes de l'école : **Elisabeth Fainstein**. Réaliser un **PoC (Proof of Concept)**, c'est-à-dire requérir l'**API d'OpenAI** avec un prompt qui demande la génération d'une histoire. Pour cela, j'ai fait un script en **Python** en suivant la documentation de l'**API** avec la version 3.5. J'ai pu faire plusieurs essais en changeant quelques mots du prompt qui correspondent aux variables fournies par le formulaire. Le **PoC** a fonctionné et cela m'a permis de valider la faisabilité de mon projet.

```
import openai

openai.api_key = API_KEY

response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[
        {
            "role": "user",
            "content": "You are a skilled french children author and you are going to write a small story for a 6 years old boy called Mateo. He must be the main character. In this story he would like to meet ninjas. The story must be very simple and 1500 words long."
        }
    ],
    temperature=1,
    max_tokens=2000,
    top_p=1,
    frequency_penalty=0,
    presence_penalty=0
)

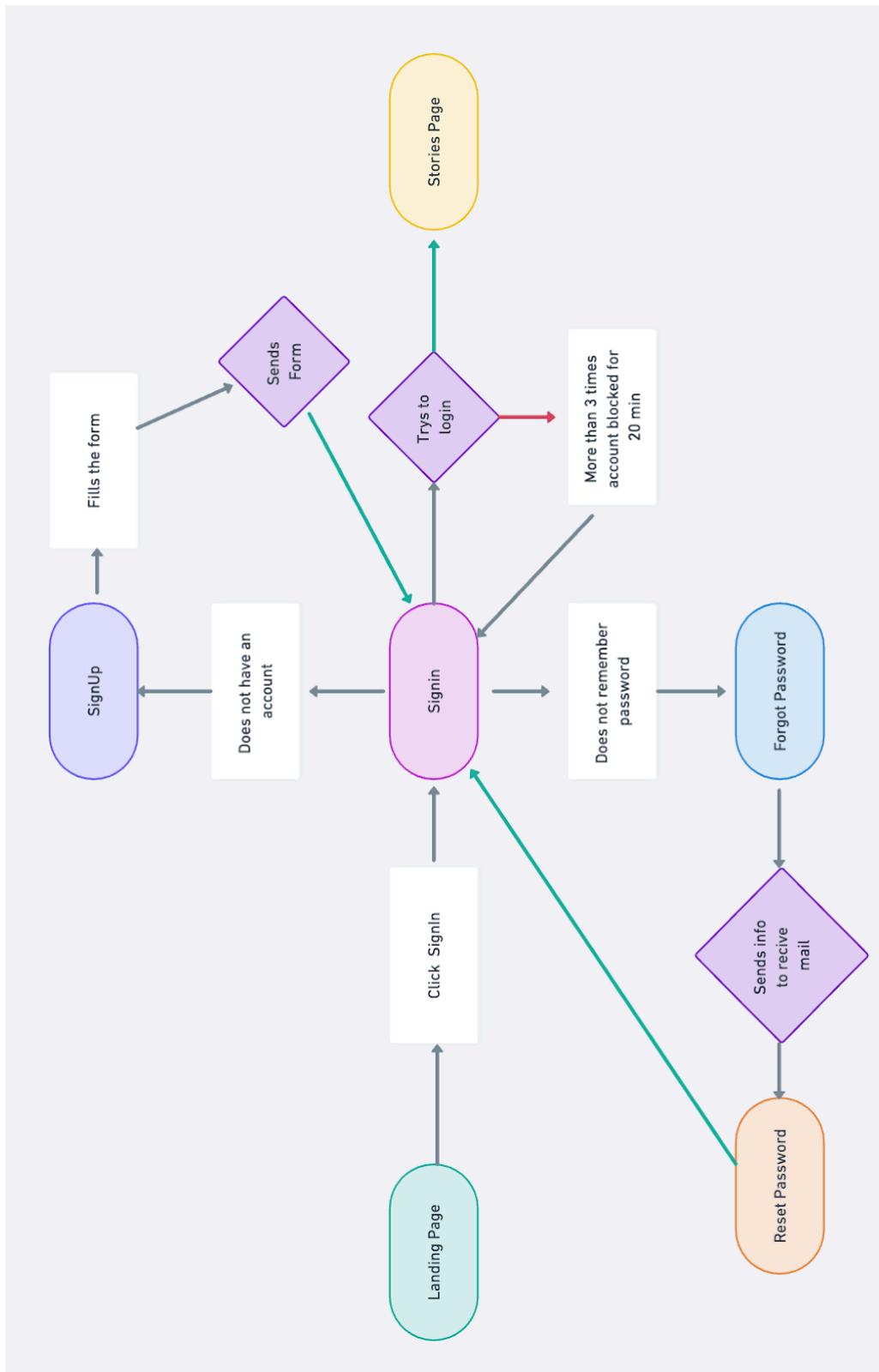
print(response.choices[-1].message.role)
print(response.choices[-1].message.content)
```



2. Spécifications Fonctionnelles

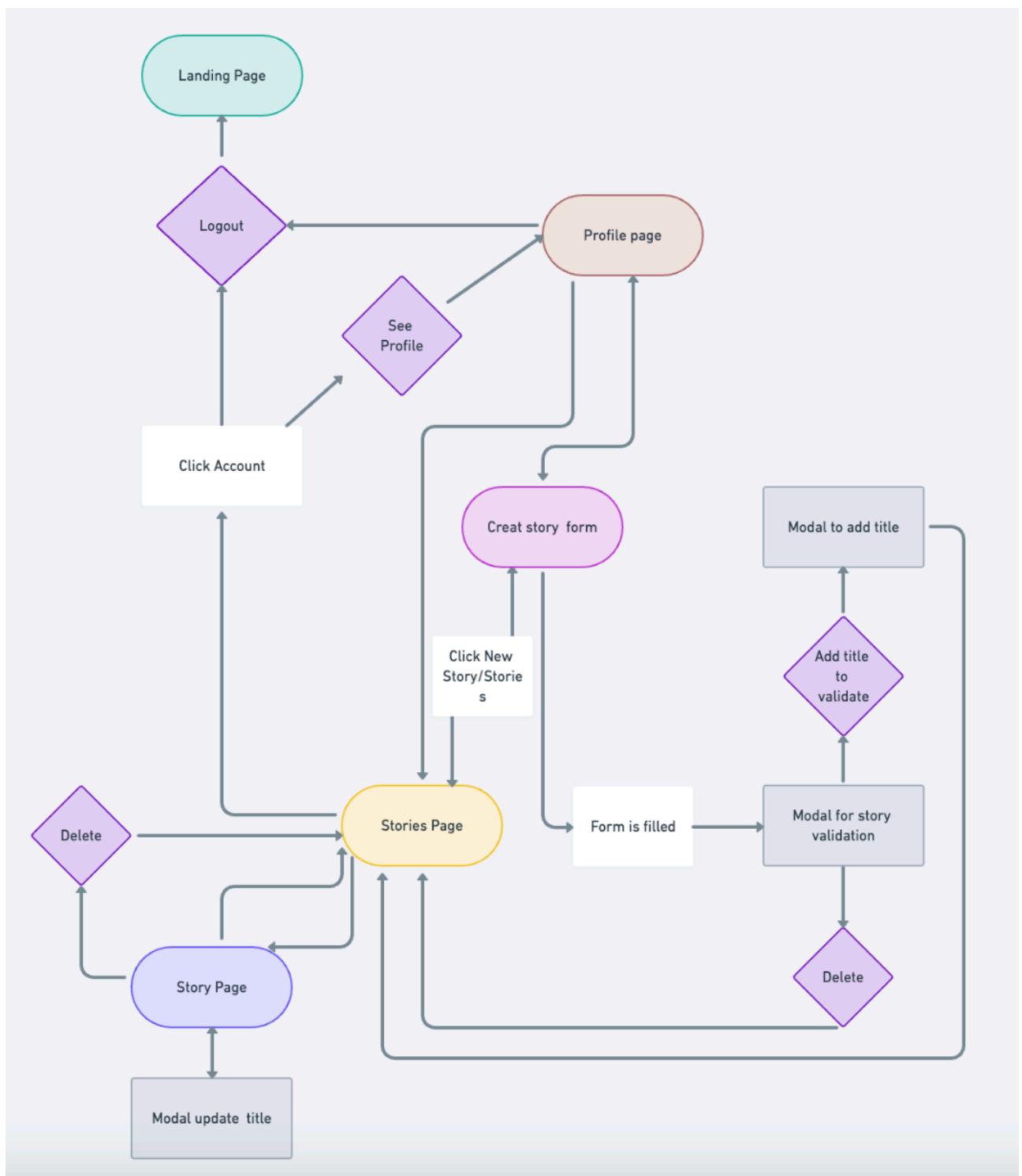
i. Modélisation

i. Diagramme d'activité à partir de la **landing page**.





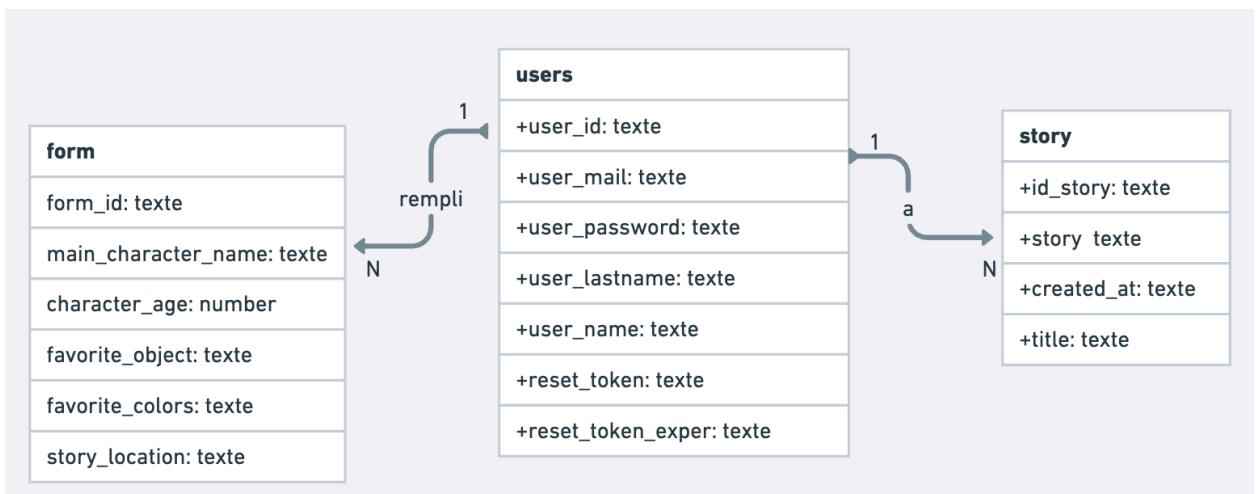
ii. Diagramme d'activité quand l'utilisateur s'est identifié.





ii. Conception de la Base de Données:

1. **Modèle conceptuel des données (MCD)** : Il représente les entités et leurs relations de manière abstraite, sans se soucier de la mise en œuvre physique. Il se concentre sur les concepts métiers et les règles de gestion. Ici, une modélisation de type **UML (Unified Modeling Language)**.



2. **Modèle Logique des Données (MLD)** : Il transforme le **MCD** en une représentation qui peut être implémentée dans un **SGBD**, en respectant les règles de normalisation. Il inclut des détails comme les clés primaires et les clés étrangères, ainsi que les types de données.



3. **Modèle Physique des Données (MPD)** : Il représente la manière dont les données seront effectivement stockées dans la base de données. Il inclut des aspects de performance tels que les index, les contraintes physiques, les partitions, etc.



```
-- PostgreSQL script
▷ Run | New Tab
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

▷ Run | New Tab | Copy
CREATE TABLE users (
    user_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_mail VARCHAR(50) NOT NULL,
    user_password VARCHAR(50) NOT NULL,
    user_lastname VARCHAR(50) NOT NULL,
    user_name VARCHAR(50) NOT NULL,
    reset_token VARCHAR(255),
    reset_token_expiry TIMESTAMP
);

-- Table Form
▷ Run | New Tab | Copy
CREATE TABLE form (
    form_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID NOT NULL,
    main_character_name VARCHAR(50) NOT NULL,
    character_age INT NOT NULL,
    favorite_object VARCHAR(50) NOT NULL,
    favorite_colors VARCHAR(50) NOT NULL,
    story_location VARCHAR(50) NOT NULL,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);

-- Table StoryCreated
▷ Run | New Tab | Copy
CREATE TABLE story_created (
    id_story UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID NOT NULL,
    story TEXT NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    title VARCHAR(255) NOT NULL,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);

-- For performance
▷ Run | New Tab
CREATE INDEX idx_form_user_id ON form(user_id);
▷ Run | New Tab
CREATE INDEX idx_story_created_user_id ON story_created(user_id);
◆◆
```



iii. Maquette de l'Interface

Tout d'abord, pour avoir une vision du nombre d'écrans à réaliser, j'ai créé des wireframes. Ceci m'a permis de visualiser le parcours utilisateur avec les informations essentielles dans chaque écran. Pour cela, j'ai utilisé l'application **MockUp**. Voici le rendu des premières maquettes, ainsi que la version finale.

The wireframe shows a main screen with a calendar grid on the left and a 'Create Story Form' on the right. The form includes fields for language (English), character name (Toto), age (3), favorite object (A teddy bear), and location (Forest). A large circular heatmap is overlaid on the right side of the screen. At the bottom, there's a blue 'Create' button and a footer with 'About AI Story' and copyright information.

The final design features a blue placeholder area with the word 'Visuel' written on it. To its right is a 'Sign in' button. Below this is a 'Description du concept' section. On the right, there's a promotional banner for 'AI Story' with the tagline 'Unleash your creativity with AI Story'. It features a cartoon illustration of two children sitting at a desk with a laptop, looking up at a starry night sky through a window. The banner text describes the platform as an ultimate storytelling tool for both beginners and experienced writers.



Sign up

Sign in Form

Forgot psk

AI Story

Sign up

Sign In

E-mail: nico@nico.com

Password:

Sign In

You do not remember your password? Change it

About AI Story

Created by JuanMak with Next.js, and Tailwind
© 2024 All Rights Reserved

Dernière histoire

Avant dernière histoire

AI Story

Stories New Story Account

KARIM STORY

Il était une fois un petit garçon nommé Karim, qui aimait plus que tout au monde les trains. Sa chambre... était remplie de petites locomotives en bois et de rails colorés. Il passait des heures à imaginer des

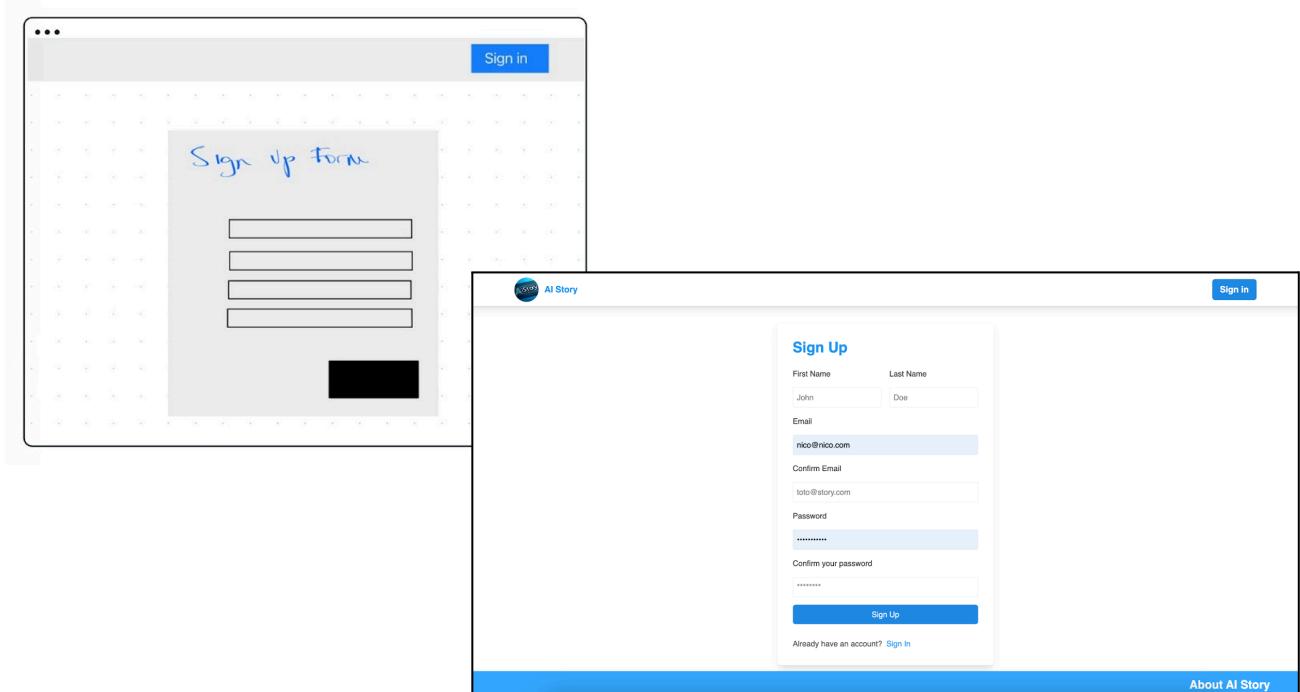
Open Delete

STORY DIANE

Il était une fois, dans la jungle magnifique, un petit garçon nommé Diane. Diane avait un doudou nom... Félix, un lion en peluche qui était son meilleur ami. Chaque jour, Diane et Félix partaient à l'aventure

Open Delete

HISTORIA DE VI



Une fois le parcours défini, j'ai voulu offrir une expérience intuitive et accessible à l'utilisateur. Pour garantir l'accessibilité de l'application, il était important de choisir une palette de couleurs répondant à ce besoin. Pour cela, j'ai effectué plusieurs tests sur différentes couleurs afin de créer une palette de couleurs qui garantirait l'accessibilité. Pour les tests, j'ai utilisé ces deux repositories sur GitHub <https://github.com/toolness/accessible-color-matrix> et <https://github.com/jxnblk/colorable>. Ils sont déployés sur les sites : <https://toolness.github.io/accessible-color-matrix/> et <https://jxnblk.github.io/colorable/demos/matrix/>. Ces outils m'ont permis de tester plusieurs couleurs et de créer une palette adéquate.

Accessible color palette builder





	White text #FFFFFF Aa	Light text #9ED4F8 Aa	Bright text #2196F3 Aa	Medium text #1E88E5 Aa	Dark text #023E8A Aa	Black text #03045E Aa
Black background #03045E	Aa	Aa	Aa	Aa		
Dark background #023E8A	Aa	Aa				
Medium background #1E88E5						Aa
Bright background #2196F3						Aa
Light background #9ED4F8					Aa	Aa
White background #FFFFFF					Aa	Aa

Adoptant une charte graphique épurée et une palette de couleurs contrastées, j'ai amélioré l'accessibilité, notamment avec des effets visuels sur les boutons pour une meilleure interaction, par exemple lors du survol des boutons.

Open

Delete

3. Spécifications Techniques du Projet

Frontend

- i. Langages de Programmation
 - HTML5 : pour la structure des pages web.
 - CSS3 : pour le style et la mise en page.
 - TypeScript : pour les interactions dynamiques et la sécurité typographique.
- ii. Frameworks et Bibliothèques
 - Next.js : pour la construction d'applications React avec rendu côté serveur (SSR) et génération de sites statiques (SSG).
 - React.js : pour la construction d'interfaces utilisateur interactives.
 - Tailwind CSS : pour des composants et des styles réutilisables.



c. Outils et Technologies

- Webpack, Babel : intégrés avec Next.js pour la gestion des modules et la transpilation.
- PostCSS et Autoprefixer : pour le traitement des styles CSS.

d. Tests et Linting

- Jest : pour les tests unitaires.
- ESLint : pour l'analyse statique du code.

Backend

a. Langages de Programmation

TypeScript (Node.js) : pour la robustesse et la sécurité typographique.

b. Frameworks

Express.js : pour la création de l'API RESTful.

c. Base de Données

PostgreSQL : comme base de données relationnelle.

d. APIs et Services

OpenAI API : pour la génération de contenu personnalisée.

e. Sécurité

- jsonwebtoken : pour l'authentification et l'autorisation.
- bcrypt : pour le hachage des mots de passe.
- Helmet : pour la protection de l'application web.
- express-rate-limit et csurf : pour la limitation du taux et la protection contre les CSRF.

f. Gestion de la Configuration

- dotenv : pour la gestion des variables d'environnement.

Spécifications DevOps et Hébergement

a. Environnement de Développement

- Git : pour le contrôle de version.
- Docker : pour la conteneurisation des applications.
- CI/CD (GitHub Actions) : pour l'intégration et le déploiement continu.

b. Serveurs et Hébergement

- Nginx : pour servir les applications web et gérer les proxys inverses.
- Debian : comme système d'exploitation du serveur.



c. Spécifications de Sécurité

- **HTTPS** : pour les communications sécurisées.
- **JWT (JSON Web Tokens)** : pour l'authentification.

Tests et Qualité

- **Tests Unitaires** : Jest (TypeScript).
- **Tests d'Intégration** : à implémenter avec Cypress avec github actions.

Performance et Monitoring

- **Outils de Monitoring** : à implémenter avec Datadog dans une future version.
- **Analyse de Performance** : Lighthouse, Google PageSpeed Insights.

Documentation et Collaboration

- **Outils de Documentation** : Notion pour les notes et journaux.
- **Outils de Collaboration** : Trello pour la gestion de projet en mode Kanban.

4. Gestion de Projet

i. Planning et Suivi

Pour m'organiser dans la réalisation du projet, j'ai décidé de découper mon calendrier d'alternance en cinq périodes pour créer ainsi un rétroplanning et définir les sujets que je comptais aborder dans chacune des périodes.

ii. Environnement Humain et Technique

i. Description des ressources humaines

J'ai réalisé ce projet seule dans le but d'apprendre et de manipuler l'intégralité de la stack technique. Toutefois, je n'ai pas été seule tout au long du processus. J'ai eu la chance de suivre des ateliers au sein de l'école **Ada Tech School**, qui m'ont permis d'aborder différents concepts utiles pour la structuration et le développement du projet. Au sein de l'entreprise où j'ai réalisé mon alternance, j'ai eu une "**buddy**" : la développeuse full stack **Meryem Bennani**. Elle a suivi mon évolution technique dans l'entreprise et dans mon projet. Elle a su répondre à des questions techniques sur le mode de fonctionnement de la stack que j'avais choisi et challenger les choix techniques que j'ai faits, me permettant d'avoir un regard critique au niveau technique sur l'ensemble de mon projet.



ii. Description des ressources techniques utilisées

Pour le développement de "**AI Story**", j'ai sélectionné une stack technique moderne et adaptée à l'exploration des frontières entre développement web et intelligence artificielle (IA), sur laquelle je souhaitais apprendre. Mon choix de **TypeScript** comme langage principal s'inscrit dans cette démarche, offrant un cadre solide pour le développement d'applications complexes avec un typage statique qui facilite la maintenance et la robustesse du code.

Front-end

Pour réaliser des maquettes et le choix de la palette de couleurs j'ai utilisé plusieurs sites internet comme <https://toolness.github.io/accessible-color-matrix/> et <https://jxnblk.github.io/colorable/demos/matrix/>, ainsi que des applications comme **Figma** et **MockUp**. Pour la modélisation du site et de la base de données, j'ai utilisé **Scalidraw** et **Whimsical**.

Le développement du front-end a été fait avec **Next.js**, qui utilise **React**. Cette stack technique devient de plus en plus populaire pour son efficacité dans la création d'interfaces utilisateur dynamiques et réactives. Ces technologies me permettent de tirer parti du rendu côté serveur proposé par Next.js pour améliorer la performance et le référencement, tandis que React facilite le développement d'une UI interactive et moderne. L'intégration de **Tailwind CSS** a été une évidence pour le design, me permettant de personnaliser rapidement l'application sans compromettre la réactivité ni l'accessibilité, grâce à son approche utility-first. Ce dernier consiste à utiliser des classes toutes faites pour ajouter directement des styles aux pages web, plutôt que de créer des styles personnalisés. En théorie, cela rend le développement plus rapide et plus simple. J'ai également fait appel à d'autres packages qui m'ont permis de répondre à des besoins particuliers comme par exemple **react-hook-form** pour la gestion des formulaires.



```
,  
  "dependencies": {  
    "@hookform/resolvers": "^3.3.4",  
    "@types/jsonwebtoken": "^9.0.6",  
    "@types/node": "20.5.9",  
    "@types/react": "18.2.21",  
    "@types/react-dom": "18.2.7",  
    "autoprefixer": "10.4.15",  
    "axios": "^1.6.8",  
    "cookie": "^0.6.0",  
    "eslint": "8.48.0",  
    "eslint-config-next": "13.4.19",  
    "js-cookie": "^3.0.5",  
    "jsonwebtoken": "^9.0.2",  
    "postcss": "8.4.29",  
    "react": "18.2.0",  
    "react-dom": "18.2.0",  
    "react-hook-form": "^7.49.3",  
    "sharp": "^0.33.3",  
    "tailwindcss": "3.3.3",  
    "zod": "^3.22.4"  
  },  
  "devDependencies": {  
    "@babel/preset-typescript": "^7.23.3",  
    "@testing-library/jest-dom": "^6.4.2",  
    "@testing-library/react": "^14.2.1",  
    "@types/cookie": "^0.6.0",  
    "@types/jest": "^29.5.12",  
    "@types/js-cookie": "^3.0.6",  
    "jest": "^29.7.0",  
    "jest-environment-jsdom": "^29.7.0",  
    "next": "^14.1.3",  
    "ts-jest": "^29.1.2",  
    "ts-node": "^10.9.2",  
    "typescript": "^5.4.2"  
  }  
}
```

Back-end

Côté back-end, bien que **Next.js** propose une solution intégrée pour le gérer, j'ai opté pour une séparation entre le back et le front. Pour ce faire, **Node.js** couplé à **Express** ont constitué la base de notre API RESTful, permettant de développer une interface de programmation à la fois efficace et modulaire pour "AI Story". Cette architecture distincte assure une flexibilité et une évolutivité maximale, en adéquation avec les exigences du projet.

La principale innovation de ce projet est l'intégration de l'**API de ChatGPT d'OpenAI**, une percée significative dans le domaine de l'intelligence artificielle. Cette intégration marque mon incursion dans le domaine naissant du **prompt engineering**, enrichissant "AI Story" avec une fonctionnalité unique : la création d'histoires personnalisées assistée par IA. L'application est désormais capable de proposer des suggestions de contenu créatif



adaptées aux préférences et interactions des utilisateurs, inaugurant une expérience utilisateur résolument novatrice et captivante.

J'ai également fait appel à d'autres packages qui m'ont permis de répondre à des besoins particuliers comme **nodemon**, qui permet de relancer le back-end à chaque sauvegarde.

Pour déployer le projet et le rendre accessible, j'ai choisi d'utiliser le serveur proposé par l'école. Il s'agit d'un serveur **SSH** fourni par le prestataire **OVH**. Pour pouvoir l'utiliser, le projet doit être conteneurisé avec **Docker**. J'ai donc décidé de créer un conteneur pour le front-end et un autre pour le back-end, et d'ajouter un fichier **docker-compose** permettant de faire fonctionner ces deux conteneurs ensemble.

```
dependencies": {
    "@types/bcrypt": "^5.0.2",
    "@types/cookie-parser": "^1.4.6",
    "@types/cors": "^2.8.16",
    "@types/csrf": "^1.11.5",
    "@types/dotenv": "^8.2.0",
    "@types/express": "^4.17.21",
    "@types/express-session": "^1.17.10",
    "@types/helmet": "^4.0.0",
    "@types/jsonwebtoken": "^9.0.5",
    "bcrypt": "^5.1.1",
    "cookie-parser": "^1.4.6",
    "cors": "^2.8.5",
    "csurf": "^1.11.0",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "express-rate-limit": "^7.2.0",
    "express-session": "^1.17.3",
    "express-validator": "^7.0.1",
    "helmet": "^7.1.0",
    "jsonwebtoken": "^9.0.2",
    "nodemailer": "^6.9.13",
    "openai": "^4.18.0",
    "pg": "^8.11.3",
    "ts-node": "^10.9.1",
    "typescript": "^5.2.2"
},
"devDependencies": {
    "@types/node": "^20.12.7",
    "@types/nodemailer": "^6.4.14",
    "@types/pg": "^8.10.9",
    "@typescript-eslint/eslint-plugin": "^6.11.0",
    "@typescript-eslint/parser": "^6.11.0",
    "eslint": "^8.53.0",
    "nodemon": "^3.0.1"
}
```

c. Objectifs de Qualité

A titre personnel, j'avais deux objectifs en réalisant ce projet:

- Créer une application qui répondait à l'ensemble du référentiel du RNCP pour valider la certification.
- Me rassurer moi-même sur ma capacité à être développeuse full-stack.
- Construire une application qui allait vivre au-delà de la certification.



Au niveau produit les objectifs sont:

- **Simplicité:** Interface épurée et facile à utiliser.
- **Accessibilité:** Facilement accessible pour les parents et leurs enfants.
- **Personnalisation:** Offrir des histoires personnalisées et interactives.
- **Technologie:** Utiliser les nouvelles technologies pour enrichir l'expérience de lecture.

5. Réalisation

i. Contexte

J'ai pris la décision consciente de mener le projet "AI Story" en solo, afin de me confronter à l'ensemble de la stack technique choisie et de maîtriser tous les aspects du développement. Cette approche m'a permis de renforcer ma compréhension des différentes technologies du projet et d'assumer pleinement la responsabilité de chaque composant. Mon objectif était de me plonger dans les détails techniques et d'acquérir une expérience pratique exhaustive.

L'une des ambitions principales de ce projet était d'explorer l'intersection entre le développement web et les avancées contemporaines en matière d'intelligence artificielle. À cet effet, j'ai intégré l'**API d'OpenAI** dans "AI Story", me lançant ainsi dans le domaine émergent du **prompt engineering**. Cette intégration a non seulement enrichi l'application d'une fonctionnalité de création d'histoires personnalisées assistée par IA, mais m'a également permis de travailler avec des technologies devenues incontournables pour les développeurs. L'expérience acquise en manipulant cette API a été une occasion précieuse pour comprendre les principes du prompt engineering et d'appliquer des concepts d'IA à un projet concret.

ii. Langages Utilisés

J'ai décidé de réaliser l'ensemble du projet principalement en **TypeScript** pour :

- **Apprentissage :** apprendre un langage qui est dans la continuité de ce que j'ai appris à l'école et qui est de plus en plus utilisé par les entreprises.
- **Typage statique :** il permet de détecter les erreurs lors de la compilation plutôt qu'à l'exécution, améliorant ainsi la fiabilité du code.
- **Complétion de code et IntelliSense :** facilite la saisie de code avec des suggestions intelligentes et des informations contextuelles.



- **Interopérabilité avec JavaScript** : TypeScript est un sur-ensemble de JavaScript, ce qui permet d'utiliser les bibliothèques et le code JavaScript existants.
- **Maintenance et évolutivité** : le typage et la structure de TypeScript facilitent la maintenance et l'évolution des projets à long terme.
- **Support communautaire et outils** : large écosystème et support intégré dans les environnements de développement comme Visual Studio Code.

iii. Réalisation des Services

1. Le Front-End

La stack technique

Le projet se distingue par sa navigation simple et réactive, propulsée par le framework **Next.js**, qui utilise la bibliothèque **React**, et tout a été développé en utilisant **TypeScript**, et **Tailwind** pour une cohérence et une performance optimales.

L'utilisation de React dans Next.js apporte plusieurs avantages parmi lesquels on peut souligner :

- **Composants réutilisables** : Permet de construire des interfaces utilisateurs avec des composants réutilisables, rendant le code plus propre et facile à maintenir.
- **Large écosystème** : La technologie React donne accès à un vaste écosystème de bibliothèques et outils, facilitant l'ajout de fonctionnalités.
- **Communauté active** : React bénéficie d'une large communauté ce qui facilite la résolution des problèmes et l'apprentissage.

Pourquoi ajouter Next.js ? Ce framework joue un rôle clé dans mon projet grâce à sa capacité à effectuer le rendu côté serveur. Cela signifie que l'application envoie le HTML final au client dès leur demande, éliminant ainsi le temps d'attente lié à l'exécution de JavaScript dans le navigateur. Ce processus améliore la vitesse de chargement des pages et optimise le référencement. Next.js offre quatre stratégies de rendu :

- **SSR (Server-Side Rendering)** : Génère le contenu côté serveur à chaque requête, fournissant immédiatement une page entièrement rendue.
- **SSG (Static Site Generation)** : Pré-rend le contenu lors du build, créant des pages statiques pour chaque route. Ces pages sont ensuite servies instantanément à chaque requête.



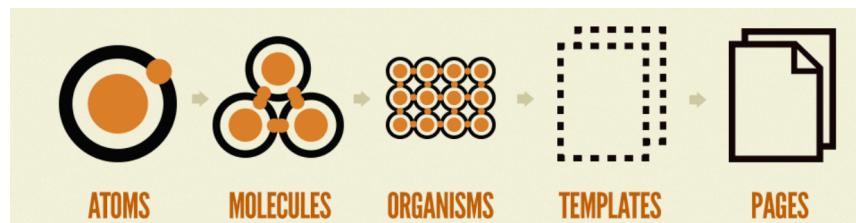
- **ISR (Incremental Static Regeneration)** : Combine SSG et SSR en générant des pages statiquement au build, qui sont rafraîchies à intervalles réguliers, offrant ainsi une mise à jour dynamique sans compromettre la performance.
- **CSR (Client-Side Rendering)** : Effectue le rendu côté client, où le contenu est chargé et affiché par JavaScript après que le squelette initial de la page ait été servi.

Cette approche me permet de créer des interfaces utilisateurs riches et interactives, tout en assurant une excellente performance.

L'architecture

Structuration des composants avec le Atomic Design Pattern

Lors de la réalisation du projet, pour structurer les composants, j'ai essayé de m'inspirer du **Atomic Design Pattern**². C'est une méthodologie de création d'interfaces graphiques qui s'articule autour de cinq composants : Atomes, Molécules, Organismes, Templates et Pages.



Atome => 

Molécule =>

Sign In

E-mail

Password

Sign In

You do not remember your password ? [Change it](#)

² <https://atomicdesign.bradfrost.com/chapter-2/>



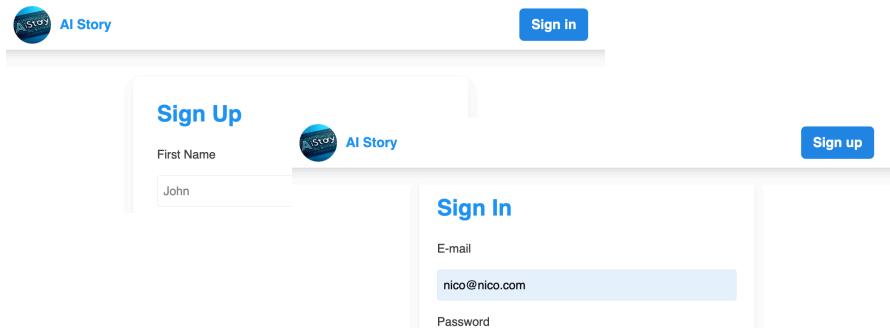
Organisme =>



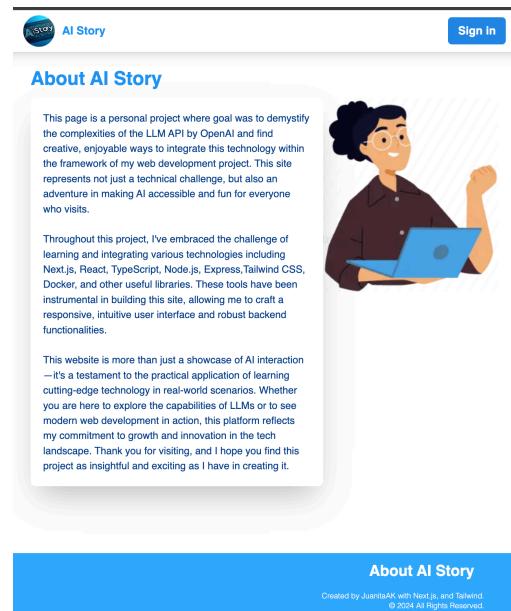
Template => squelette

Evolution navbar selon

la navigation.



Pages =><https://aistory.adahub.fr/about>



J'ai ainsi découpé les composants en **Organismes**, qui sont les fichiers avec des composants permettant de construire les pages, comme les forms, les loaders et les modales. **Templates** pour les fichiers qui regroupent plusieurs organismes pour faire les squelettes, et enfin les **Pages** pour le composant qui allait générer la totalité de la page.



Dans le contexte de **Next.js**, j'ai travaillé sur la version où le routing se fait à partir d'un dossier appelé “**pages**” (frontend>src>pages). Dans ce dossier, j'ai défini le routing et, dans chaque page, j'importe le composant qui a la structure de la page et les données qui seront rendus dans cet URL, qui sera la page finale. Les composants seront importés dans ce fichier depuis le dossier **pages**, qui se trouve dans (frontend>src>components>pages). Ce fichier est l'équivalent du template. Enfin, les composants qui vont structurer le template (organismes) seront importés depuis le dossier (frontend>src>components>organismes).

Réalisation de l'Interface

Pour développer **AI Story**, j'ai d'abord analysé le besoin principal de l'utilisateur afin de concevoir une application répondant précisément à ce besoin et ainsi définir le **MVP (minimum viable product)**. Sur la base de cette analyse, j'ai créé des maquettes qui ont servi de guide pour le développement de l'interface utilisateur. Ces maquettes (voir pages 21-22-23) ont permis de :

- Définir le contenu de chaque écran et le nombre d'écrans nécessaires pour le projet, afin de savoir ce que je devais développer côté front-end et back-end.
- Visualiser le parcours utilisateur et les données avec le contenu de chaque page et à quel moment il était accessible, me permettant de mieux définir les routes côté back-end.
- Anticiper le processus d'authentification sécurisé et les différentes fonctionnalités accessibles selon les actions des utilisateurs qui devaient être codées.

Développement des fonctionnalités

Pour comprendre les fonctionnalités, je vais dans un premier temps détailler le comportement du front-end pour recevoir et afficher des informations. Par la suite, je vais expliquer la manière dont ces informations sont appelées ou envoyées au back-end.

Parmi les différentes fonctionnalités du **MVP** abordées à la page 14, je vais en détailler deux : l'affichage des histoires et la saisie du formulaire d'inscription. Il est important de noter que quatre des fonctionnalités font appel à l'utilisation des formulaires. C'est le cas de l'inscription, de l'authentification, de la création de l'histoire et de la modification du titre.



Les Formulaires :

Pour réaliser l'ensemble des formulaires, j'ai utilisé la bibliothèque **react-hook-form** pour gérer les formulaires et la validation, ainsi que **Zod** pour définir et vérifier les schémas de données. Je vais utiliser le formulaire d'inscription pour montrer comment ils ont été développés.

The screenshot shows a mobile application interface for 'AI Story'. At the top, there is a navigation bar with a circular profile icon, the text 'AI Story', and a 'Sign In' button. Below this is a 'Sign Up' form. The form fields include 'First Name' (John), 'Last Name' (Doe), 'Email' (toto@story.com), 'Confirm Email' (toto@story.com), 'Password' (*****), and 'Confirm your password' (*****). A blue 'Sign Up' button is at the bottom. Below the form, a link says 'Already have an account? [Sign In](#)'. On the right side of the screen, there is a larger, semi-transparent overlay showing the same 'Sign Up' form, which appears to be a modal or a detailed view of the original form.

Importation des modules : J'ai importé les modules nécessaires, comme **createUser** pour envoyer les données d'inscription, **useRouter** pour rediriger après l'inscription, **useForm** de **react-hook-form** pour gérer les formulaires, et **Zod** pour la validation des données.



```
import Link from "next/link";
import { createUser } from "@/services/creatUserFormApi";
import { SubmitHandler, useForm } from "react-hook-form";
import * as z from "zod";
import { zodResolver } from "@hookform/resolvers/zod";
import { useRouter } from "next/router";
import { useState } from "react";
```

Définition des types et schémas:

J'ai défini un type **SignUpFormData** pour les données du formulaire et un schéma **Zod** pour valider ces données, en détaillant le type de données attendu. La validation rigoureuse des entrées utilisateur permet de prévenir les injections SQL et les attaques XSS (Cross-site scripting)³.

Pour information, **Zod** est une bibliothèque de validation de schémas qui permet de :

- **Définir des schémas de validation** : Utilisée pour créer des schémas détaillés qui définissent la structure et les contraintes des données.
- **Valider les données** : Les schémas Zod peuvent être utilisés pour valider les données et fournir des messages d'erreur détaillés si les données ne respectent pas les contraintes définies.

Par exemple, j'ai suivi les conseils de l'**ANSII**⁴ pour réaliser les contraintes de validation du mot de passe. Il doit contenir au moins 8 caractères, une lettre majuscule, une lettre minuscule, un chiffre et un caractère spécial. Pour effectuer cette vérification de schéma, j'ai dû implémenter une expression régulière (Regex) dans **Zod** en détaillant le type de mot de passe à accepter.

```
const passwordRequirements =
  /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}\$/;
```

Cette regex vérifie que le mot de passe répond aux critères suivants :

³ C'est une faille de sécurité qui permet à un attaquant d'injecter dans un site web un code client malveillant. Ce code est exécuté par les victimes et permet aux attaquants de contourner les contrôles d'accès et d'usurper l'identité des utilisateurs.

Source: https://developer.mozilla.org/fr/docs/Glossary/Cross-site_scripting

⁴ L'Agence nationale de la sécurité des systèmes d'information (ANSSI) est l'autorité nationale en matière de cybersécurité.



- **^ et \$** : Début et fin de la chaîne, respectivement.
- **(?=.*[a-z])** : Doit contenir au moins une lettre minuscule.
- **(?=.*[A-Z])** : Doit contenir au moins une lettre majuscule.
- **(?=.*\d)** : Doit contenir au moins un chiffre.
- **(?=.[@\$!%?&])** : Doit contenir au moins un caractère spécial parmi @, \$, !, %, *, ?, &.
- **[A-Za-z\d@\$!%*?&]{8,}** : Doit avoir une longueur minimale de 8 caractères, incluant uniquement les lettres (majuscules et minuscules), chiffres et caractères spéciaux mentionnés.

Le schéma de validation de Zod qui prend en compte la regex:

```
password: z
  .string()
  .min(8, { message: "Password must be at least 8 characters" })
  .regex(passwordRequirements, {
    message:
      "Password must be at least 8 characters long and contain at least 1 uppercase letter,
       lowercase letter, 1 number, and 1 special character (@, $, !, %, *, ?, &).",
  }),
confirm_password: z
  .string()
  .min(8, { message: "Password must be at least 8 characters" }),
})
```

Mise en place du formulaire. J'ai utilisé **useForm** de la bibliothèque **react-hook-form** pour initialiser le formulaire avec la validation basée sur le schéma **Zod**.

Pour information, **react-hook-form** est une bibliothèque utilisée pour gérer les formulaires dans les applications React. Elle permet de :

- **Faciliter la gestion des formulaires** : En fournissant des hooks pour enregistrer les entrées et gérer les soumissions de formulaire.
- **Optimiser les performances** : En réduisant les rendus inutiles lors de l'interaction avec le formulaire.
- **Gérer la validation** : En intégrant facilement des schémas de validation comme **Zod**.

Le formulaire capture des informations telles que le prénom, le nom, l'email, la confirmation de l'email, le mot de passe et la confirmation du mot de passe. En résumé, **react-hook-form** et **Zod** travaillent ensemble pour fournir une gestion efficace des



formulaires avec une validation robuste, assurant que les données saisies par l'utilisateur sont correctes et sécurisées avant d'être envoyées au serveur.

Soumission du formulaire:

Lorsque le formulaire est soumis, j'appelle la fonction `createUser` pour envoyer les données au serveur. Si l'inscription réussit, l'utilisateur est redirigé vers la page de connexion. En cas d'erreur, un message d'erreur est affiché.

```
try {
  await createUser(data);
  await router.push(`./login`);
} catch (error) {
  setResponse(response);
  setError("root", {
    message: "Something is wrong with your informations",
  });
  if (error instanceof z.ZodError) {
    console.error("Validation Errors:", error.errors);
  } else {
    console.error("Other Errors:", error);
    setResponse(String(error));
  }
}
```

Affichage des erreurs: J'ai configuré le formulaire pour afficher les messages d'erreur à côté des champs concernés si la validation échoue.

```
<label
  className="text-m font-medium text-neutral-700"
  htmlFor="password"
>
  Password
</label>
<input
  className="w-full p-2 border rounded focus:ring focus:ring-hover"
  type="password"
  id="password"
  placeholder="*****"
  {...register("password")}
/>
{errors.password && (
  <div className="text-red-500">{errors.password.message}</div>
)}
```

Le catalogue d'histoires:

L'une des principales fonctionnalités est le catalogue d'histoires. Une fois l'utilisateur identifié, il est redirigé vers <https://aistory.adahub.fr/stories>. Si c'est la première session, l'utilisateur reçoit un message l'invitant à créer sa première histoire. Dans le cas contraire, il voit son catalogue d'histoires déjà réalisées. Pour réaliser cela, j'ai créé la fonction `StoriesContainer`.



Avant de faire la condition, il a fallu faire :

- **Importations**
 - **NoStoryCard** : Un composant qui est affiché lorsque la liste des histoires est vide.
 - **StoryTitleCard** : Un composant qui affiche le titre et les détails d'une histoire.
 - **girl** : Une image importée représentant une fille regardant quelque chose.
 - **Image** : Un composant de **Next.js** utilisé pour gérer les images.

```
You, last month | 2 authors (Juanita Afanador and one other)
import { NoStoryCard } from "../../organismes/storiesCard/NoStoryCard";
import { StoryTitleCard } from "../../organismes/storiesCard/StoryTitleCard";
import girl from "../../../../../public/girl_watching.png";
import Image from "next/image";

export type Story = [
  id_story: string;
  story: string;
  user_id?: string; Juanita Afanador, 3 months ago • feat: authentication
  created_at?: Date | string;
  title?: string;
];
```

- **La définition du type Story**

- **Story** : Un type TypeScript définissant la structure d'une histoire, incluant **id_story**, **story**, **user_id**, **created_at** et **title**.

- **La fonction StoriesContainer**

StoriesContainer est un composant React qui affiche une liste d'histoires ou un message indiquant qu'il n'y a pas d'histoires disponibles. Il utilise des composants importés et des images pour créer l'interface utilisateur.

Le **Paramètre** qu'elle prend c'est “stories” qui est un tableau d'objets de type **Story**.



```
const StoriesContainer = ({ stories }: { stories: Story[] }) => {
  if (!stories || stories.length === 0) {
    return (
      <div className="stories flex flex-col justify-center items-center">
        <NoStoryCard />
      </div>
    );
  }
  return (
    <div className="stories flex flex-col m-3">
      <div className="flex flex-row md:gap-3">
        <h2 className="md:w-full text-2xl font-bold text-left md:text-4xl mt-8 mb-3 ml-3
          md:ml-9 leading-tight text-nav-font"> You, last month • chore: Update foot
          Stories
        </h2>
        <Image
          src={girl}
          alt="women watching"
          width="40"
          className="md:hidden z-30"
        />
      </div>

      <div className=" md:grid md:grid-cols-8">
        <div className="md:col-start-1 col-span-5 z-0">
          {stories.map((story: Story, index) => (
            <StoryTitleCard key={index} {...story} />
          )))
        </div>
        <Image
          src={girl}
          alt="women watching"
          width="300"
          className="invisible sticky top-20 md:visible md:grid md:col-start-6
          md:col-span-3 z-30"
        />
      </div>
    </div>
  );
},
```

Le Comportement

Vérification des Histoires

- Si le tableau **stories** est vide ou non défini, le composant retourne une **div** contenant le composant **NoStoryCard**, indiquant qu'il n'y a pas d'histoires à afficher.

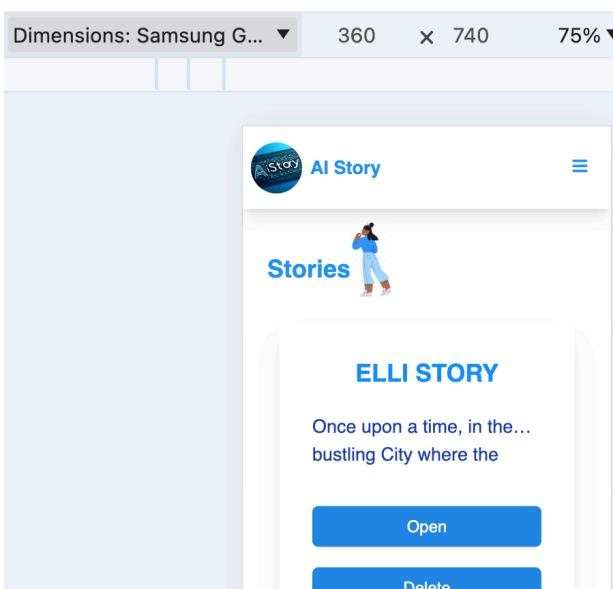
Affichage des Histoires

Si le tableau **stories** contient au moins un élément, on utilise la méthode **map** pour parcourir chaque histoire dans le tableau **stories** et afficher un composant **<StoryTitleCard/>** pour chaque histoire.

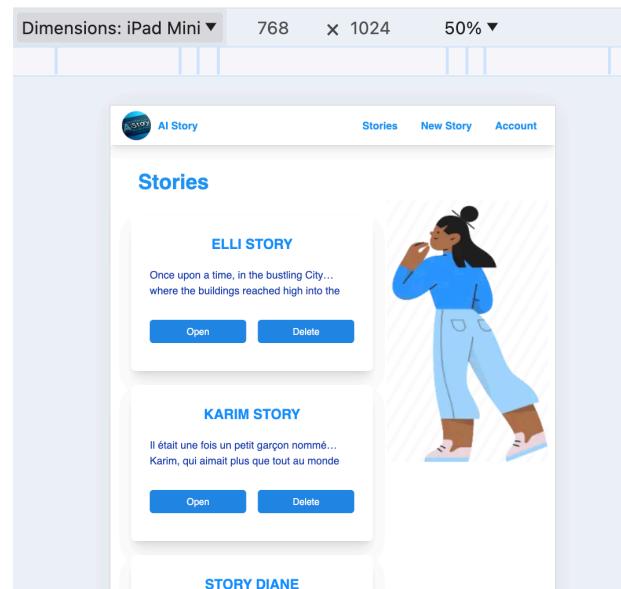


- **Titre des Histoires** : Affiche le titre "Stories" avec une mise en page gérée avec Tailwind.
- **Images** : Affichage des images selon la taille de l'écran. Pour les petits écrans, comme ceux des téléphones mobiles, une petite image <Image/> d'une fille est affichée à côté du titre. Pour les écrans de taille moyenne ou plus, comme ceux des tablettes ou des ordinateurs, une version plus grande de la même image <Image/> est affichée à côté des composants <StoryTitleCard/>.

Écran de type mobile



Écran type tablette



La récupération et l'envoie des données avec un Back For Front (BFF):

Une des particularités de **Next.js** est qu'il intègre une API directement dans le framework. J'ai utilisé cette fonctionnalité pour envoyer depuis l'API de Next les données au back-end en **Node.js**. Pour cela, dans le dossier de routing **pages**, il y a un sous-dossier nommé **api**. À l'intérieur de ce dossier, nous effectuons les différentes opérations de gestion des données CRUD (Create, Read, Update, Delete).

Comme premier exemple, on peut voir l'implémentation de la fonction **createUser** vue précédemment dans le formulaire d'inscription. Cette fonction asynchrone utilise la méthode **POST** pour envoyer les données renseignées par l'utilisateur vers le back-end de **Next.js** avec la méthode **fetch** vers "**/api/auth/signup**", qui est l'emplacement du



gestionnaire de **Next.js**. En cas d'erreur, il gère les erreurs et renvoie une réponse appropriée à l'utilisateur.

```
export const createUser = async (user: SignUpFormData) => {
  const response = await fetch("/api/auth/signup", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(user),
  });

  if (response.status === 409) {
    throw new Error("User already exists 😊");
  }

  if (response.status === 500) {
    throw new Error("Failed to create user... 😞 please try again later");
  }

  return response.json();
}
```

À son tour, le gestionnaire d'API, à travers la fonction asynchrone handler, traite les requêtes POST pour l'inscription en envoyant les données reçues à l'API du back-end. Il gère les erreurs et renvoie une réponse appropriée à l'utilisateur.

```
import { NextApiRequest, NextApiResponse } from "next";
import axios, { AxiosError } from "axios";
const BACKEND_BASE_URL = process.env.BACKEND_BASE_URL as string;
const SIGNUP_API_URL = (BACKEND_BASE_URL + process.env.SIGNUP_API) as string;

const handler = async (req: NextApiRequest, res: NextApiResponse) => {
  if (req.method === "POST") {
    try {
      const response = await axios.post(SIGNUP_API_URL, req.body);
      res.status(200).json(response.data);
    } catch (error) {
      if (error instanceof AxiosError) {
        res
          .status(error.status || 500)
          .json({ error: "Internal Server Error" });
      }
      res.status(500).json({ error: "Internal Server Error" });
    }
  }
};

export default handler;
```



Nous procémons de manière similaire pendant la création des histoires. Lors de la soumission du formulaire, nous appelons la fonction **createStory**. Celle-ci envoie une requête au **BFF(api/stories)** de **Next.js**.

```
export const createStory = async (story: StoryFormData) => {
  const response = await fetch("/api/stories", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(story),
  });

  if (!response.ok) {
    throw new Error("Failed to create story... 😞 please try again later");
  }

  return response.json();
};
```

La fonction asynchrone **handler** est un gestionnaire d'API qui analyse le type de requête HTTP reçue (**POST**, **GET**, **PATCH**, **DELETE**) et exécute l'opération CRUD correspondante sur les données des histoires. Pour cela, elle utilise **Axios** pour communiquer avec le back-end en **Node.js**, qui est sécurisé par un jeton d'authentification. En gérant les erreurs, elle assure une interaction fiable et sécurisée entre le front-end et le back-end.

```
const handler = async (req: NextApiRequest, res: NextApiResponse) => {
  if (req.method === "POST") {
    try {
      const AuthToken = req.cookies["Auth-Token"];
      const response = await axios.post(STORIES_API_URL, req.body, {
        headers: { Authorization: `Bearer ${AuthToken}` },
      });
      res.status(200).json(response.data);
    } catch (error) {
      if (error instanceof AxiosError) {
        res
          .status(error.status || 500)
          .json({ error: "Internal Server Error" });
      }
      res.status(500).json({ error: "Internal Server Error" });
    }
  }
};
```



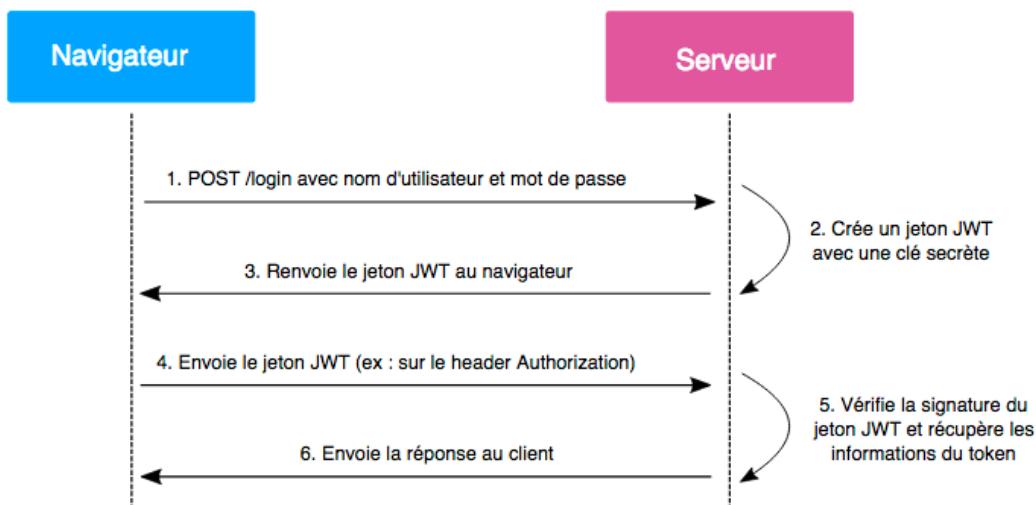
L'authentification:

Un des principaux enjeux était de garantir la protection des comptes utilisateurs et de leurs données. Pour cela, j'ai utilisé des **JSON Web Tokens (JWT)** stockés dans des cookies. Le package **JWT** améliore la sécurisation et l'optimisation de l'authentification et de la gestion des sessions pour l'ensemble de l'application. Les tokens **JWT** identifient les utilisateurs de manière sécurisée et sans friction, permettant une navigation fluide tout en protégeant leur contenu et leurs données personnelles. Dans le back-end, ils permettent également de maintenir l'état de connexion des utilisateurs à travers différentes requêtes, sans nécessité de stockage de session côté serveur, simplifiant la gestion des sessions et augmentant la sécurité en minimisant les risques d'exposition des données de session.

J'ai comparé l'utilisation des **Cookies** avec celle du **Local Storage**. J'ai décidé d'utiliser les cookies car cela me semblait un choix plus sécurisé et c'était aussi une opportunité d'apprendre à les utiliser comme méthode d'authentification.

Fonctionnalité	Local Storage	Cookies
Capacité de stockage	Environ 5-10 Mo	Environ 4 Ko
Accessibilité	Accessible uniquement via JavaScript	Accessible via JavaScript et/ou serveur
Envoi au serveur	Non	Oui, envoyé avec chaque requête HTTP
Durée de vie	Persiste jusqu'à suppression	Peut être configuré pour expirer
Sécurité	Moins sécurisé pour les données sensibles	Plus sécurisé avec `HttpOnly`, `Secure`, etc.

Pour réaliser l'implémentation côté front-end, lorsque l'utilisateur saisit ses identifiants dans le formulaire, ils sont envoyés au **BFF**. La fonction **handler** envoie à son tour une requête **HTTP POST** avec les identifiants de l'utilisateur au back-end. Une fois la réponse reçue du back-end, le **handler** traite la réponse pour stocker un token d'authentification dans un cookie et l'utiliser à chaque requête au back-end. Enfin, la fonction gère les erreurs pour fournir des messages appropriés en cas d'échec de l'authentification. Désormais, à chaque fois qu'il y aura une demande au back-end, si le cookie existe, il sera envoyé dans le header de la requête.



5

Comme vous avez pu peut-être le constater à la page 42, lors de la fonction `createStory`, le **handler** du **BFF** pour la méthode **POST** fournit le token dans le header comme un cookie au back-end pour montrer que l'utilisateur est identifié et ainsi obtenir une réponse.

Pour l'implémentation côté back-end, j'ai fait appel à **Bcrypt**⁶ et **JWT**. **Bcrypt** protège les mots de passe et les informations sensibles des utilisateurs par un processus de hachage⁷ et de salage⁸, bloquant les attaques par force brute et rendant les données indéchiffrables même en cas d'accès non autorisé à la base.

Lorsque le back-end reçoit la requête du front-end avec les identifiants de l'utilisateur, il effectue les étapes suivantes :

1. Il recherche dans la base de données l'utilisateur correspondant à l'email fourni.
2. Il compare le mot de passe fourni (en texte clair) avec le mot de passe haché stocké dans la base de données en utilisant **bcrypt.compare()**.

⁵ Source: <https://www.vaadata.com/blog/fr/jetons-jwt-et-securite-principes-et-cas-dutilisation>.

⁶ Algorithme de hachage de mot de passe fort qui est résistant aux attaques par force brute. Source: <https://academy.visiplus.com/ressources/definition/bcrypt>

⁷ Le **hachage** permet de ne pas stocker les mots de passe en clair dans la base mais uniquement de stocker une empreinte de ces derniers grâce à un algorithme dédié. Source: <https://www.cnil.fr/fr/definition/hachage#:~:text=L'utilisation%20d'une%20fonction,de%20calculer%20les%20dites%20empreintes>.

⁸ Le **salage** est une méthode permettant de renforcer la sécurité des informations qui sont destinées à être hachées (par exemple des mots de passe) en y ajoutant une donnée supplémentaire afin d'empêcher que deux informations identiques ne conduisent à la même empreinte. Source: [https://fr.wikipedia.org/wiki/Salage_\(cryptographie\)](https://fr.wikipedia.org/wiki/Salage_(cryptographie))



```
try {
  const user = await getUserByMail(user_mail);
  if (user) {
    const isValid = await bcrypt.compare(password, user.user_password);
    if (!isValid) {
      return res.status(401).json({ message: "Invalid credentials" });
    }
  }
}
```

3. Si la comparaison est validée, on génère un JWT avec des informations sensibles du client, un secret et la validité du cookie. Le secret permet de vérifier l'authenticité du token lors des requêtes ultérieures.

```
const token = jwt.sign(
  { userId: user.user_id, name: user.user_name, mail: user_mail },
  JWT_SECRET,
  {
    expiresIn: AUTH_TOKEN_EXPIRY_IN_SECONDS,
  }
);
```

4. Enfin, nous configurons un cookie HTTP dans le header pour l'authentification et envoyons une réponse JSON contenant l'ID de l'utilisateur.

```
res.header(
  "set-cookie",
  `Auth-Token=${token}; SameSite=Lax; Secure; path=/; expires=${new Date().getTime() + AUTH_TOKEN_EXPIRY_IN_SECONDS * 1000}
`);
res.status(200).json({ userId: user.user_id });
```

Lors de la configuration, on définit :

- **SameSite=Lax** : Indique que le cookie doit être envoyé pour les requêtes de première partie et les navigations de troisième partie (liens), mais pas pour les requêtes de type POST provenant de sites externes. Cela aide à protéger contre les attaques CSRF (Cross-Site Request Forgery).
- **Secure** : Indique que le cookie doit être transmis uniquement via des connexions HTTPS sécurisées.
- **path=/** : Spécifie que le cookie est accessible sur l'ensemble du site (toutes les pages du domaine).



- **expires=\${new Date().getTime() + AUTH_TOKEN_EXPIRY_IN_SECONDS * 1000}** : Définit la date d'expiration du cookie. Le temps actuel (en millisecondes) est additionné à la durée de validité du token (en secondes, convertie en millisecondes). Cela fixe une date après laquelle le cookie sera automatiquement supprimé par le navigateur dans les 24 heures.

2. Le back-end

J'ai décidé de développer le back-end en **Node.js** avec **Express** pour plusieurs raisons :

Node.js :

- **Code unifié pour le front-end et le back-end:** Node.js utilise le même langage de programmation côté serveur et client, simplifiant le développement et réduisant le temps d'apprentissage.
- **Asynchrone et non bloquant:** L'architecture de Node.js est basée sur des opérations d'entrée/sortie non bloquantes, ce qui rend les applications rapides et efficaces.
- **Écosystème riche:** npm, le gestionnaire de paquets de Node.js, est l'un des plus grands écosystèmes de bibliothèques au monde, fournissant des modules et des packages qui simplifient le développement.
- **Scalabilité:** Node.js est conçu pour être léger et efficace, idéal pour des applications en temps réel traitant un grand nombre de connexions simultanées.

Express :

- **Simplicité et modularité:** Express permet de développer des applications web avec les outils nécessaires pour gérer les opérations courantes comme gérer les requêtes HTTP, les routes et les middlewares, tout en restant simple et sans complexité inutile.
- **Middleware flexible:** le système de middleware d'Express simplifie l'enrichissement des applications web avec des fonctionnalités variées et modulaires
- **Routing performant:** Express simplifie la gestion des chemins d'accès (routes) des applications web, même lorsque ces chemins sont variés et complexes
- **Documentation et support communautaire:** l'excellente documentation et la communauté active d'Express rendent son apprentissage et son utilisation plus faciles grâce à la disponibilité de nombreux tutoriels et forums.



- **Compatibilité avec les autres modules Node.js:** Express facilite l'intégration avec d'autres modules et frameworks, offrant ainsi une flexibilité et une extensibilité accrues.

L'architecture :

L'architecture de ce projet peut être décrite comme une combinaison d'architecture modulaire avec des principes **MVC** et **RESTful**.

- **Architecture MVC (Modèle Vue Contrôleur)** : Cette architecture se manifeste par des routes définies de manière modulaire (**userRouter** et **storyRouter**), ce qui permet de séparer les besoins. En plus, l'utilisation de plusieurs middlewares (**helmet**, **cors**, **globalLimiter**, **auth**) avec des responsabilités bien définies.

```
import express, { Request, Response } from "express";
import helmet from "helmet";
import * as dotenv from "dotenv";
import cors from "cors";
import { corsOptions } from "./middlewares/cors";
import { globalLimiter } from "./middlewares/limiter";
import { auth } from "./middlewares/auth";

import userRouter from "./routes/userRouter";
import storyRouter from "./routes/storyRouter";
dotenv.config();

const app = express();
const port = process.env.PORT;

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

app.use(helmet(), cors(corsOptions), globalLimiter);

app.use("/auth", userRouter);

app.use("/stories", auth, storyRouter);

app.get("/", (req: Request, res: Response) => {
  res.status(200).send("This is the backend server for the app.");
});

app.listen(port, () => {
  const date = new Date().toISOString();
  const mode = process.env.PORT || "development";
  console.log(`[${date}] Server started in ${mode} mode on port ${port}.`);
});
```



- **Architecture RESTful** : Cette architecture est visible par l'utilisation des endpoints **REST (Representational State Transfer)**, c'est-à-dire des points d'accès dans l'API comme les routes “/auth” et “/stories”, où se déroulent des opérations **CRUD**. L'utilisation des middlewares directement dans ces routes montre une gestion des permissions et des sécurités, alignée avec les principes **RESTful**.

```
import express from "express";
import * as StoriesControllers from "../controllers/storiesController";
import { postStoryForm } from "../controllers/storiesFormControllers";
import { storyLimiter } from "../middlewares/limiter";
const router = express.Router(); Juanita Afanador, 3 months ago • 1

// get last story
router.get("/latest", StoriesControllers.getLatestStory);

// get one story
router.get("/:id", StoriesControllers.getStory);

//delete story
router.delete("/:id", StoriesControllers.deletingStory);

//patch story
router.patch("/:id", StoriesControllers.patchingStory);

//post story-form
router.post("/", storyLimiter, postStoryForm);

//get all stories
router.get("/", StoriesControllers.getStories);

export default router;
```

Veiller sur la sécurité du serveur

Au niveau du back-end, plusieurs packages tels que **CORS**, **JWT**, **Helmet** et **express-rate-limit** ont été intégrés dans le but de protéger le serveur, le compte des utilisateurs et l'accès à l'API d'OpenAI.

- Le package **CORS**, par exemple, avait pour but de donner seulement accès au back-end à l'URL où se trouve le front-end.
- **express-rate-limit** était utilisé pour limiter le nombre de requêtes sur l'API d'OpenAI par utilisateur, limiter le nombre de tentatives de connexion sur un compte précis et enfin le nombre de requêtes directement sur le serveur.



```
import rateLimit from "express-rate-limit";

export const storyLimiter = rateLimit({
  max: 10,
  windowMs: 24 * 60 * 60 * 1000,
  message: "Too many requests from this IP, please try again after in 24 hours",
});

export const userLimiter = rateLimit({
  max: 3,
  windowMs: 20 * 60 * 1000,
  message:
    "Too many requests to identify, please try again after in 20 minutes",
});

export const globalLimiter = rateLimit({
  max: 100,
  windowMs: 3 * 60 * 1000,
  message:
    "Too many requests from this IP, please try again after in 15 minutes",
});
```

3. La base de données

Le choix de **PostgreSQL** comme base de données s'est porté sur plusieurs raisons :

- **Apprentissage** : C'était l'occasion d'apprendre à utiliser ce type de base de données qui est très répandu.
- **Richesse des fonctionnalités** : **PostgreSQL** est reconnu pour offrir un ensemble riche de fonctionnalités avancées permettant une grande flexibilité et des performances optimales.
- **Fiabilité et conformité aux standards** : **PostgreSQL** est réputé pour sa stabilité et sa fiabilité, avec une conformité stricte aux standards **SQL (Structured Query Language)**, garantissant l'intégrité des données et permettant une migration plus facile vers ou depuis d'autres bases de données relationnelles.
- **Extensibilité et communauté active** : Comme le reste de la stack technique, **PostgreSQL** est hautement extensible. De plus, il existe une communauté active et un large éventail de ressources contribuant à la formation et à la résolution rapide des problèmes.

Je suis consciente cependant que ce choix implique certaines problématiques. Les principaux inconvénients d'utiliser une base de données SQL comme **PostgreSQL** incluent :



- **Complexité de gestion** : La gestion de **PostgreSQL** peut être complexe et nécessite des compétences spécialisées pour la maintenance.
- **Schémas rigides** : Les schémas rigides peuvent limiter la flexibilité et rendre les modifications structurelles difficiles.
- **Performances** : Les performances peuvent être impactées dans des scénarios de forte concurrence ou de requêtes complexes, augmentant ainsi les coûts en matériel et en expertise.

Pour réaliser la base de données, j'ai écrit un script en SQL pour **PostgreSQL** que j'ai utilisé sur **pgAdmin** et **Vercel** pour créer la base de données (voir le modèle physique des données à la page 19).

Pour le développement, j'ai installé **PostgreSQL** sur mon ordinateur et utilisé **pgAdmin** pour la visualisation des données. Pour réaliser les appels à la base de données en local, j'ai dû suivre un protocole différent de celui utilisé pour la version déployée sur **Vercel**.

Sur la version locale, il est nécessaire de fournir plusieurs informations comme l'utilisateur, l'hôte, le port et le mot de passe. Pour la version déployée, **Vercel** fournit une URL qui permet de réaliser la connexion sur leur serveur directement. Lors du développement, j'ai travaillé sur la version locale. Le but étant de pouvoir faire des tests sans avoir à me soucier de l'intégrité des données en local et de garantir des données non corrompues en production.

Connexion pour le développement

```
const { Pool } = require("pg");
import * as dotenv from "dotenv";
dotenv.config();

//Dev database
const poolPromise = (async () => {
  const pool = new Pool({
    user: process.env.DB_USER,
    host: process.env.DB_HOST || "localhost",
    port: process.env.DB_PORT,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_NAME,
  });
  return pool;
})();
export { poolPromise };
```

Connexion pour la production

```
//Prod database
const poolPromise = new Pool({
  connectionString: process.env.POSTGRES_URL,
});

export { poolPromise };
```

Pour répondre aux besoins du front-end, j'ai implémenté plusieurs fonctions pour requêter ou enregistrer des données.



```
"""
export const postNewStory = async (openAiStory: string, user_id: string) => {
  const pool = await poolPromise;
  const query =
    "INSERT INTO public.story_created (user_id, story) VALUES ($1, $2)";
  const values = [user_id, openAiStory];

  try {
    const result = await pool.query(query, values);
    return result;
  } catch (error) {
    console.error(error);
    throw error;
  }
};

export const getAllStories = async (user_id: string): Promise<NewStory> => {
  try {
    const pool = await poolPromise;
    const query =
      "SELECT * FROM public.story_created WHERE user_id = $1 ORDER BY
      created_at DESC";
    const result = await pool.query(query, [user_id]);
    return result.rows;
  } catch (error) {
    console.error(error);
    throw error;
  }
};

export const getStoryById = async (user_id: string): Promise<NewStory> => {
  const pool = await poolPromise;
  const query = "SELECT * FROM public.story_created WHERE user_id = $1";
  const result = await pool.query(query, [user_id]);
  return result.rows;
};
```

4. Service tiers: API Open AI

Quand j'ai commencé le projet, la seule entreprise qui permettait un accès facile à son API avec une documentation claire était **OpenAI**. J'ai essayé de faire des demandes d'accès à d'autres API, comme l'API **Claude d'Anthropic**, sans succès à ce moment-là. Depuis, l'écosystème de la **GenAI (Generative AI)** a beaucoup évolué, plusieurs acteurs sont apparus et l'offre s'est étendue. J'ai cependant voulu rester avec la stack technique avec laquelle j'avais commencé et qui répondait à mes besoins avant d'explorer d'autres modèles.

Le PoC était réalisé en Python (voir la page 10), mais le projet était en TypeScript. J'ai donc dû revoir la documentation pour comprendre comment réaliser l'appel dans le contexte du projet.



Dans le cadre d'un projet utilisant **Node.js** et **Express**, il est nécessaire, dans un premier temps, d'instancier la classe **OpenAI** en utilisant l'objet contenant la clé API de l'utilisateur.

```
import OpenAI from "openai";
import * as dotenv from "dotenv";
dotenv.config();

const openai = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY,
});

export default openai;
```

```
export const postStoryForm = async (
): Promise<void> => {
  if (!req.body) {
    res.status(400).send({
      message: "No information has been sent.",
    });
  }
  try {
    const storyForm: storyForm & { userId: string } = req.body;
    await postStoryFormInput(storyForm);
    const {
      language,
      main_character_name,
      character_age,
      favorite_object,
      story_location,
      favorite_colors,
    } = storyForm;
    const completion = await openai.chat.completions.create({
      messages: [
        {
          role: "user",
          content: `You are going to play the role of Roald Dahl to write a very imaginative story for a ${character_age} old child called ${main_character_name}. ${main_character_name} should himself be the main character of the story. ${main_character_name} loves very much ${favorite_object} and his favorite place is ${story_location}. The story must be entirely written ${language}. The story will subtly emphasize the values of courage, creativity, honesty and resilience if possible, without being too on the nose about it. The story should not have sexism, gender bias, racism or any other form of discrimination. Avoid any illegal or inappropriate content creation. You will write a 1000 words story (without title) in the style so characteristic of Roald Dahl. The story should have a single twist adapted to a ${character_age} year old to ensure the story is not boring -- Your goal is for ${main_character_name} to be delighted and to think this is one of the best short story he ever heard. Feel free to add your own imaginative elements in the story so that it will not be too repetitive if you need to do this exercice multiple times. Remember, this is a SHORT story, don't be too ambitious. Also do not write genericities like "He had many adventures", "Together they faced many dangers"-- This is super boring, write a sharp story with a beginning, and end, and a single twist, but write it well so it reads as a great story to read to a kid before going to bed.`,
        },
      ],
      model: "gpt-3.5-turbo",
    });

    const openAiStory = completion.choices[0].message.content?.trim();
    const storyResult = await postNewStory(openAiStory ?? "", req.body.userId);
    res.status(201).json(storyResult);
  } catch (error) {
    console.error(error);
    res
      .status(500)
      .send("Internal Server Error. There is a problem with the GenAI API");
  }
}
```

Dans la fonction **postStoryForm** on gère une requête **POST** pour générer une histoire personnalisée en utilisant l'API d'OpenAI. Elle fonctionne comme suit :



1. **Récupération des données du formulaire** : Extrait les informations soumises telles que le nom de l'enfant, son âge, son objet préféré, son lieu favori, ainsi que son identifiant.
2. **Préparation de la requête** : Crée un prompt détaillé pour l'API OpenAI, demandant de générer une histoire dans le style de Roald Dahl.
3. **Appel à l'API OpenAI** : Envoie le prompt préparé via la méthode `chat.completions.create` en utilisant le modèle **gpt-3.5-turbo** pour générer l'histoire.
4. **Validation de la requête** : Vérifie si le corps de la requête est vide et renvoie une erreur 400 si c'est le cas.
5. **Génération de l'histoire** : Utilise l'API d'OpenAI pour générer l'histoire.
6. **Sauvegarde de l'histoire** : Enregistre l'histoire générée dans la base de données avec la fonction `postNewStory`.
7. **Réponse** : Renvoie un statut 201 avec le résultat de l'enregistrement de l'histoire si tout s'est bien passé, sinon renvoie une erreur 500 en cas de problème.

5. Le déploiement

Dockerisation

Pour réaliser le déploiement, j'ai utilisé le serveur **OVH** proposé par l'école. Il s'agit d'un serveur **SSH** accessible à l'ensemble des apprenants en alternance. Chacun travaillant sur des projets divers avec des stacks techniques différentes, l'utilisation de **Docker** devient indispensable pour permettre à chaque projet d'être déployé sur le même serveur. Pour une gestion plus facile du serveur, j'ai utilisé le plugin **Remote - SSH** de **VSCode**, qui permet de naviguer dans le serveur comme on le ferait dans un projet avec **VSCode**.

Pour dockeriser le projet, j'ai dû conteneuriser le front-end et le back-end. Dans le cas du front-end, les applications **Next.js** sont idéalement déployées sur les

```
services:  
  backend:  
    build:  
      dockerfile: Dockerfile  
      context: ./backend  
    ports:  
      - "5000:5000"  
    volumes:  
      - ./backend:/var/app  
      - /var/app/node_modules  
    env_file:  
      - ./backend/.env.docker  
  frontend:  
    build:  
      context: ./frontend  
      target: dev  
      dockerfile: Dockerfile  
    volumes:  
      - ./frontend:/var/app  
      - /app/node_modules  
      - /app/.next  
    env_file:  
      - ./frontend/.env.docker  
    ports:  
      - "3000:3000"
```



serveurs de **Vercel**, l'entreprise créatrice du framework. Pour cela, il est nécessaire de donner accès au projet sur **GitHub** et de fournir les variables d'environnement nécessaires au déploiement. Cependant, le site de **Next.js** fournit de la documentation pour conteneuriser l'application.

Côté back-end, il existe plusieurs sources de documentation expliquant comment conteneuriser un projet avec **Node.js** et **Express**. Pour orchestrer les conteneurs, j'ai dû me familiariser avec la création du fichier **docker-compose** en fonction de l'arborescence du projet et intégrer les variables d'environnement nécessaires au déploiement.

6. Tests

Pour éviter d'avoir des bugs et m'assurer que les fonctionnalités essentielles marchent correctement, j'ai réalisé des tests unitaires pour différentes fonctionnalités.

```
home:frontend/ (mainx) $ npm run test-unit
> my-story@0.1.0 test-unit
> jest

PASS  src/components/pages/Home/__tests__/_LandingPage.test.tsx
PASS  src/components/organismes/Layout/__tests__/_Footer.test.tsx
PASS  src/components/organismes/storiesCard/__tests__/_StoryCard.test.tsx
PASS  src/components/organismes/storiesCard/__tests__/_NoStoryCard.test.tsx
PASS  src/components/organismes/storiesCard/__tests__/_StoryTitleCard.test.tsx
PASS  src/components/pages/Profile/__tests__/_Profile.test.tsx
PASS  src/components/pages/Stories/__tests__/_StoriesContainer.test.tsx

Test Suites: 7 passed, 7 total
Tests:       13 passed, 13 total
Snapshots:   0 total
Time:        2.967 s
Ran all test suites.
```

J'ai aussi mis en place un GitHub Action pour lancer l'ensemble des test unitaires à chaque merge pour garantir qu'à chaque modification ceci impactait pas les fonctionnalités.



JuanitaAK / my-story

Type ⌘ to search

Code Issues Pull requests Actions Projects Security 7 Insights Settings

All workflows

Showing runs from all workflows

80 workflow runs

✓ chore: update test footer main Deploy to test environment #77: Commit 528a250 pushed by JuanitaAK

✗ chore: Update footer layout and styling main Deploy to test environment #76: Commit 0d52b18 pushed by JuanitaAK

✗ chore: Update footer layout and styling main Deploy to test environment #75: Commit 72cafb4 pushed by JuanitaAK

Exemple : Test de la page de Profile

J'ai testé le bon fonctionnement de la page de profil de l'utilisateur et sa capacité à afficher correctement les données reçues par le composant.

Juanita Afanador, 4 months ago | 1 author (Juanita Afanador)

```
1 name: "⚡ Deploy to test environment"
2
3 on:
4   push:
5     branches:
6       - main
7
8   concurrency:
9     group: main
10  cancel-in-progress: true      Juanita Afanador
11
12 #matrix for frontend and backend packages
13
14 jobs:
15   build-test:
16     name: Build and test
17     runs-on: ubuntu-latest
18     strategy:
19       matrix:
20         package: [frontend, backend]
21     steps:
22       - name: Checkout
23         uses: actions/checkout@v2
24       - name: Install dependencies
25         run: |
26           echo "Building and testing ${{ matrix.package }}"
27           cd ${{ matrix.package }} && npm install
28       - name: Run tests
29         run: |
30           echo "Running tests for ${{ matrix.package }}"
31           cd ${{ matrix.package }} && npm run test-unit
```

Deploy to test environment: Some jobs were not successful

View workflow run

✗ Deploy to test environment / Build and test (frontend) Failed in 23 seconds

✓ Deploy to test environment / Build and test (backend) Succeeded in 9 seconds



```
Juanita Afanador, 2 months ago · Author (Juanita Afanador)
import { screen, render } from "@testing-library/react";
import "@testing-library/jest-dom";
import { ProfilePageProps } from "../Profile";
import Profile from "../Profile";

const userData = {
  userId: "111111111111",
  name: "Doe",
  mail: "toto@gmail.com",
};

Run | Debug
describe("Profile", () => [
  const props: ProfilePageProps = {
    ...userData,
  };
  Run | Debug
  it("should render the user's name and email", () => {
    render(<Profile profile={props} />);
    const name = screen.getByText("Doe");

    expect(name).toBeInTheDocument();
  });
  Juanita Afanador, 3 months ago • feat: Update package.json
  Run | Debug
  it("should render the user's email", () => {
    const userData = {
      firstName: "John",
      lastName: "Doe",
      email: "john.doe@example.com",
    };

    render(<Profile profile={props} />);
    const email = screen.getByText("toto@gmail.com");

    expect(email).toBeInTheDocument();
  });
])

```

7. Veille

J'ai suivi de près l'évolution de l'intelligence artificielle et l'interaction avec le modèle d'OpenAI. Au début de mon projet, le typage dans le package d'OpenAI n'était pas exhaustif. Cependant, suite à une mise à jour d'OpenAI, j'ai pu intégrer un package qui facilitait considérablement l'implémentation. De plus, j'ai surveillé les évolutions des packages et pris en compte les dépendances sur **GitHub** pour effectuer les mises à jour nécessaires en cas de problèmes de sécurité.



4. Défis et Recherche

Plusieurs étapes de ce projet ont représenté des défis de recherche, car j'avais choisi d'utiliser plusieurs technologies que je n'avais pas manipulées auparavant. Cela incluait la gestion de bases de données relationnelles, l'API d'**OpenAI**, l'utilisation de **cookies** pour l'authentification sans recourir à un service tiers, le déploiement sur un serveur **SSH**, la dockerisation d'un projet full stack, ainsi que la mise en place d'un **Docker Compose** permettant de lancer l'ensemble des conteneurs. Ces défis m'ont permis de me familiariser avec ces différentes technologies, d'acquérir de nouvelles compétences et de mener à bien mon projet.

5. Conclusion

La réalisation de cette application illustre non seulement mon intérêt pour les technologies de pointe, mais aussi mon ambition de rester à la pointe des évolutions dans le secteur du développement web. Travailler avec l'API de ChatGPT m'a permis d'assimiler et d'appliquer des concepts d'intelligence artificielle de manière pratique, renforçant ainsi mes compétences en tant que développeuse full stack. Cela a également approfondi ma compréhension de l'intégration des solutions IA dans les applications web modernes. La réalisation de ce projet m'a permis de comprendre les enjeux d'un projet full-stack et de prendre le temps d'approfondir des concepts et de découvrir des technologies qui m'étaient inconnues au début de ce projet.

a. Axes d'amélioration

À long terme, je souhaite apporter plusieurs améliorations à ce projet :

- **Utilisation des images générées par IA** : Intégrer des images générées par IA pour enrichir visuellement les histoires et offrir une expérience plus immersive aux utilisateurs.
- **Amélioration de l'UI** : Revoir et améliorer l'interface utilisateur pour une expérience plus fluide et agréable, en appliquant les principes de l'Atomic Design de manière plus exhaustive.



- **Optimisation du SEO** : Améliorer le référencement naturel pour attirer plus de trafic organique et augmenter la visibilité de l'application.
- **Landing page améliorée** : Développer une landing page optimisée avec une option de vente pour des contenus plus élaborés et une partie freemium pour attirer davantage d'utilisateurs.

En somme, ce projet a été une expérience enrichissante et formatrice, me permettant de naviguer entre différentes technologies et d'acquérir des compétences précieuses. Les défis rencontrés ont été des opportunités d'apprentissage et de croissance, et je suis enthousiaste à l'idée de continuer à améliorer cette application et à explorer de nouvelles innovations dans le développement web.

