Dossier de Projet

NewsWik Articles

stage du 1 septembre au 24 octobre 2025

Préparation au titre de Développeur·se Web et Web Mobile

Thomas SCHMIDT

Remerciements

Je tiens à exprimer toute ma gratitude envers **Alltech Consulting** et **Thibaud FAURE** le directeur d'agence ouest pour m'avoir accueilli et offert l'opportunité de vivre une expérience enrichissante et formatrice. J'ai particulièrement apprécié la confiance qui m'a été accordée ainsi que l'environnement professionnel stimulant, propice à l'apprentissage et à l'épanouissement.

Mes remerciements s'adressent tout spécialement à mon tuteur, **Fabrice Turpeau**, pour son accompagnement bienveillant, sa disponibilité et ses conseils éclairés qui m'ont permis de progresser et de mieux comprendre les enjeux de la mission. Son expertise et son sens du partage ont été des atouts précieux tout au long de mon parcours.

Je souhaite aussi adresser mes sincères remerciements à l'ensemble du staff technique d'Alltech Consulting (**Maxime, David, Damien, Yannick, Orane, Anthony**) . Leur disponibilité, leur professionnalisme et leur esprit d'équipe ont grandement facilité mon intégration et mon apprentissage. Grâce à leurs échanges constructifs, à leur soutien au quotidien et à leur volonté de partager leurs connaissances, j'ai pu progresser dans de bonnes conditions et acquérir des compétences concrètes. Leur sens de la collaboration et leur rigueur technique ont été pour moi une source d'inspiration et de motivation, renforçant mon envie de m'investir pleinement dans ce domaine exigeant et passionnant.

Je remercie également **Jennifer Pie**, pour son écoute, son soutien et son suivi attentif du côté des ressources humaines. Son professionnalisme et sa réactivité ont largement contribué à la qualité de mon intégration et de mon expérience au sein d'Alltech Consulting.

Je remercie également tous les professeurs, coach, et personnels administratifs de l'ENI pour l'aide et le soutien qu'ils m'ont procurés durant ces 8 mois de formation.

Merci à tous pour cette collaboration positive et formatrice !

Table des matières

Introduction

Je suis né en 1982 et à cette époque les ordinateurs n'étaient pas très communs dans les foyers…

Mais à 10 ans j'ai eu mon premier ordinateur, et c'est devenu une passion immédiatement. Ensuite je me suis orienté vers un cursus en électronique (BAC STI électronique puis BTS électronique) pour comprendre comment fonctionnait le cœur de mon ordinateur.

Après l'obtention de mon BTS, j'ai travaillé pendant 22 ans dans le secteur de l'aéronautique, où j'ai eu la chance d'œuvrer sur du matériel civil (équipements divers d'A320 / A340 / A380 / B787…) et militaire (Mirage 2000, Rafale, NH90, Tigre …) de haute technologie et, à certaines occasions, d'intervenir directement sur des bases à l'étranger (Italie, Finlande).

Au fil de ce parcours, j'ai occupé différents postes : technicien de réglage, technicien de réparations, technicien de tests, ainsi que contrôleur d'approbation pour remise en service. Ces expériences m'ont permis d'acquérir une expertise solide dans le domaine du *hardware*, mais aussi de développer des qualités indispensables telles que la rigueur, la polyvalence et la capacité d'adaptation.

Fort de cette expérience, j'ai choisi de donner une nouvelle orientation à ma carrière et de revenir à ma première passion : l'informatique. J'ai décidé de me tourner vers le développement logiciel, et plus particulièrement le développement d'applications web.

C'est dans cette optique que j'ai intégré la formation de Développeur Web/Web Mobile à l'ENI École Informatique, après avoir réussi les tests et l'entretien de sélection. J'ai débuté ce cursus le 17 mars 2025, convaincu qu'il représente le meilleur tremplin pour concrétiser mon projet professionnel.

Mon objectif à présent est de poursuivre mon parcours par une alternance en Concepteur Développeur d'Applications (CDA), afin de consolider mes compétences, gagner en expérience pratique et devenir pleinement opérationnel en tant que développeur web.

Lexique

API : Application Programming Interface
BAC STI : Baccalauréat Sciences et Technologies Industrielles
BSON : Binary JSON
BTS : Brevet de Technicien Supérieur
CDA : Concepteur Développeur d'Applications
CORS : Cross-Origin Resource Sharing
CRUD : Create, Read, Update, Delete
CSS : Cascading Style Sheets
CVE : Common Vulnerabilities and Exposures
DWWM : Développeur Web et Web Mobile
DTO : Data Transfer Object
ES : ECMAScript
ESN : Entreprise de Services du Numérique
HTML : HyperText Markup Language
HTTP : HyperText Transfer Protocol
HTTPS : HyperText Transfer Protocol Secure
IDE : Integrated Development Environment
IT : Information Technology (Technologie de l'Information)
JSON : JavaScript Object Notation
LTS : Long-Term Support
MFE : Micro FrontEnd
MIME : Multipurpose Internet Mail Extensions
NoSQL : Not Only SQL
NPM : Node Package Manager
OWASP : Open Web Application Security Project
PDF : Plateforme Dématérialisée de Formation
REST : Representational State Transfer
SPA : Single-Page Application
SQL : Structured Query Language
SSRF : Server-Side Request Forgery
TLS : Transport Layer Security
UI : User Interface
UML : Unified Modeling Language
URL : Uniform Resource Locator
UX : User Experience
XSS : Cross-Site Scripting

Présentation de l'entreprise

Historique de l'entreprise

Alltech a été fondée en 2015 à Bordeaux par Wilfrid PICQ et Frédéric LASCOMBE, deux professionnels issus du secteur des services numériques (ESN). Forts de plusieurs années d'expérience dans la gestion de projets informatiques et l'accompagnement d'équipes techniques, ils ont souhaité créer une société de services à taille humaine, centrée sur la qualité des relations humaines et la pertinence des missions proposées.

L'objectif d'Alltech est de proposer une approche du consulting IT plus agile et collaborative, valorisant l'écoute, l'expertise technique et l'épanouissement des collaborateurs. Depuis sa création, l'entreprise a connu une croissance régulière tout en conservant ses valeurs d'origine : proximité, transparence et engagement dans la réussite des projets clients.

Présentation des métiers

L 'ADN Alltech Consulting est profondément ancré dans le développement informatique. C'est là que tout a commencé : dans la technique, au plus proche du code, des frameworks, des architectures, des environnements complexes — Là où les projets prennent réellement vie.

Voici les métiers qui structurent un projet IT chez Alltech Consulting :

- Scrum Masters & Chefs de Projet pour orchestrer l'agilité et garantir l'avancement des équipes
- Business Analysts pour faire le pont entre les enjeux métier et les solutions techniques
- UX/UI Designers pour concevoir des expériences utilisateurs intuitives et des interfaces à la fois esthétiques et fonctionnelles
- DevOps pour assurer la stabilité et la mise en production fluide des applications
- Product Owners pour incarner la vision produit
- Mais aussi experts en delivery, qualité, accompagnement au changement, Data Analysts, Data Engineers, Data Scientists…

Clients de référence

Chaque agence Alltech Consulting développe des partenariats solides avec des clients qui lui sont propres, ce qui permet une couverture sectorielle large et des contextes projets très variés :

- Banque & Assurance
- Secteur public
- Industrie
- E-commerce

Chiffres clés

Alltech est aujourd'hui une entreprise solidement implantée dans le paysage du conseil en technologies. Présente dans 11 villes en France et disposant de 4 agences principales situées à Bayonne, Bordeaux, Niort et Nantes, elle compte plus de 180 collaborateurs mobilisés sur plus de 100 projets actifs. Forte de cette croissance continue, l'entreprise a réalisé un chiffre d'affaires de 16 millions d'euros en 2024, confirmant sa place parmi les acteurs dynamiques du secteur des services numériques.

Site de Niort

Je suis en stage à l'agence de Niort, 7 Esplanade de la République.

Elle occupe le 1er, 2eme et 3eme étage d'un immeuble en plein centre de Niort, face à la place de la brèche.

Le 1er étage est un espace détente avec machine à café, fauteuils et tables pour se restaurer.

Le 2eme étage est constitué des bureaux des équipes techniques avec plusieurs salles de réunions dont une équipée pour la visioconférence.

Et le dernier niveau concerne plus particulièrement les métiers transverses comme ressources humaines, commerciaux, direction, mais peut aussi accueillir réunions et équipes techniques.

Equipe PDF

Le projet PDF et les missions de l'équipe

L'équipe dans laquelle je travaille est encadrée par Fabrice Turpeau, responsable du projet PDF (Plateforme Dématérialisée de Formation). Ce projet est le développement d' un logiciel permettant de gérer et de dématérialiser les processus de formation. L'avancement du développement est suivi à l'aide d'un tableau Kanban sur GitHub, qui permet de visualiser les tâches à faire, en cours et terminées. Chaque vendredi après-midi, une réunion d'équipe est organisée afin d'échanger sur l'avancement du projet, les difficultés rencontrées et les pistes d'amélioration.

Organisation et environnement de travail

Les collaborateurs d'Alltech travaillent selon un rythme hybride, avec deux jours par semaine en présentiel à l'agence et le reste du temps en télétravail. La communication est assurée grâce à Microsoft Teams, ce qui permet de garder un lien constant entre les membres de l'équipe. Les développeurs disposent d'un environnement de travail principalement basé sur Windows, avec les outils nécessaires au développement, aux tests et au suivi du projet.

Choix technologiques et méthodes de travail

Les principaux langages et frameworks utilisés au sein du bassin niortais sont Java avec Spring, Kotlin, React et Angular. En tant qu'Entreprise de Services du Numérique (ESN), Alltech Consulting adapte ses technologies aux besoins de ses clients et aux spécificités des projets. L'équipe suit une méthodologie de travail Agile, qui repose sur des itérations courtes, une communication régulière et des réunions de suivi (planifications, points quotidiens et revues de sprint). Cette approche favorise la réactivité et l'amélioration continue du projet PDF.

Liste des compétences du référentiel qui sont couvertes par le projet

Nom et prénom du candidat : SCHMIDT THOMAS

Document complété d'un commun accord entre le stagiaire et le responsable du stage en entreprise à joindre au rapport d'activité.

Compétences Voir le détail dans le référentiel d'emploi, d'activités et de compétences	Cocher les compétences mises en œuvre lors du projet en entreprise (en totalité ou partiellement)
C1 - Installer et configurer son environnement de travail en fonction du projet web ou web mobile	☑
C2 - Maquetter des interfaces utilisateur web ou web mobile	☑
C3 - Réaliser des interfaces utilisateur statiques web ou web mobile	☑
C4 - Développer la partie dynamique des interfaces utilisateur web ou web mobile	☑
C5 - Mettre en place une base de données relationnelle	☑
C6 - Développer des composants d'accès aux données SQL et NoSQL	☑
C7 - Développer des composants métier coté serveur	☑
C8 - Documenter le déploiement d'une application dynamique web ou web mobile	☐

Observations éventuelles :

Du stagiaire : .....

De l'entreprise : Très Bonne Implication de Thomas dans le projet qui lui a été donné de faire

Nom et Prénom du stagiaire : SCHMIDT THOMAS Signature :	Entreprise : Nom du responsable de stage : TURPIN FABRICE Signature :

Configuration de l'environnement de travail

Avant toutes choses il a fallu configurer le PC fourni par l'entreprise et l'espace de travail ainsi qu'installer les IDE utilisés par Alltech Consulting.
IntelliJ et **VScode** ont donc été installés puis configurés.
Chez Alltech Consulting, habituellement, on utilise les IDE :

- **IntelliJ** plutôt pour le Back-End
- **VSCode** plutôt pour le Front-End.

J'ai utilisé **IntelliJ** pour le backend et le frontend.

Une branche **Git** à été ajoutée au projet PDF par mon tuteur afin de pouvoir versionner et partager mon code en ligne.

Installation de logiciels "utilitaires" :

- **UMLet** pour intégrer les diagrammes de cas d'utilisations.
- **MongoDB Compass** pour la visualisation de la Base de données.
- **Postman** pour tester les requêtes vers l'API.

Présentation générale du projet

La plateforme PDF actuellement en construction souhaite se doter de plusieurs widgets pour agrémenter la page d'accueil des collaborateurs.

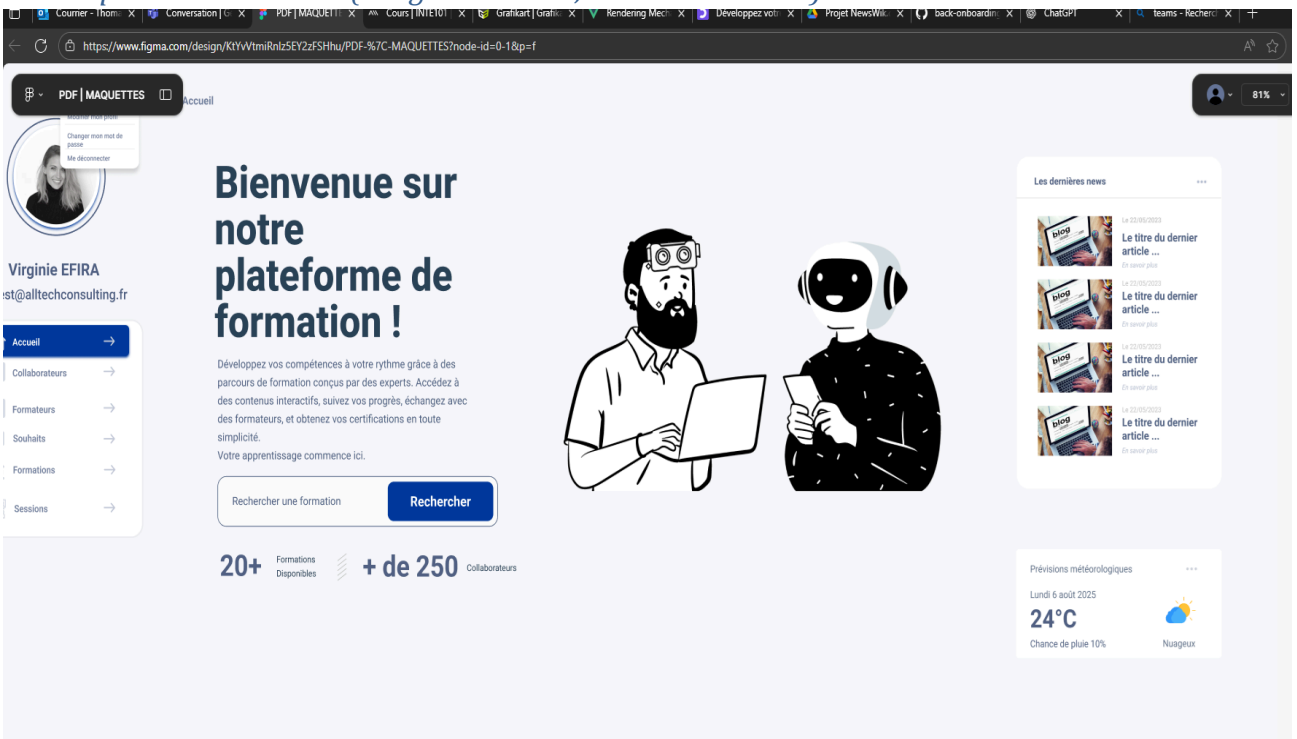
Voici les trois widgets retenus :

- NewsWik Weather qui présente la météo.
- NewsWik Ephemeris qui présente les données du jour (date, Saint du jour, Citations…).
- NewsWik Articles qui présente une sélection de quelques articles.

Le projet qui englobe le développement de ces 3 widgets se nomme NewsWik.

NewsWik Articles est le projet qui m'a été attribué.

Accueil personnalisés PDF (widgets à droite, Articles et Météo) - Annexe 1



Présentation détaillée de NewsWik Articles

Un micro frontend est un module autonome du front-end, développé et maintenu indépendamment, qui peut s'intégrer facilement dans une application principale pour en étendre les fonctionnalités.

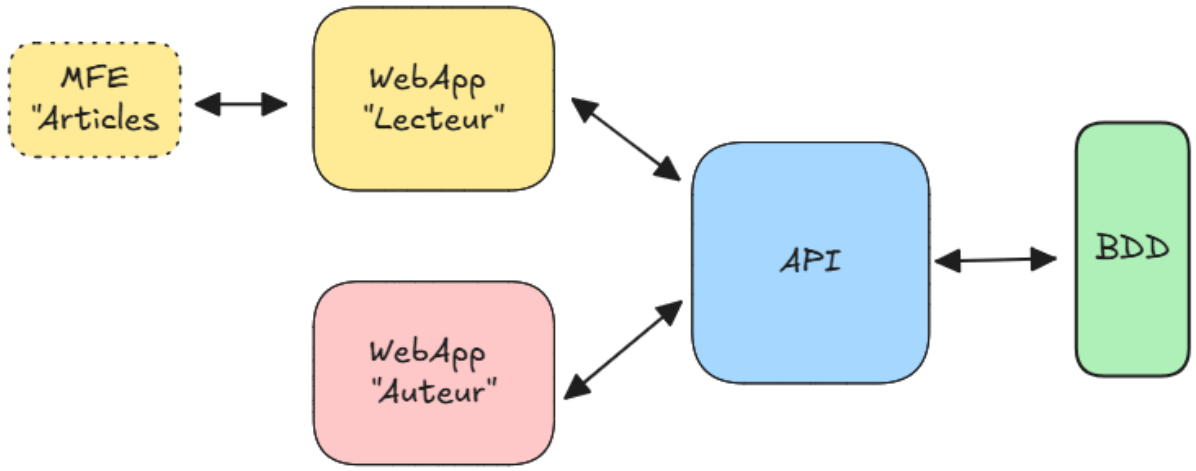
Le module micro frontend Articles présente une sélection de quelques titres d'articles qui seront disponibles à la lecture.

Lorsque l'utilisateur clique sur l'article, il est redirigé vers l'application "lecteur" qui permet de visionner la photo taille réelle, lire l'article. Des informations supplémentaires sont disponibles comme la date de création, la date de modification, le nom de l'auteur.

Les articles présentés sont écrits par des collaborateurs qui ont accès à une autre application Web côté Auteur. Les articles peuvent être aussi modifiés, publiés ou dépubliés.

Une API commune au module "lecteur" et "auteur" s'occupe de fournir les données en provenance de la base de données.

Représentation graphique des différents modules



Synoptique fourni par Alltech Consulting en Annexe 2 pour plus de détails.

Spécification du projet

Les contraintes techniques de ce projet était :

- utiliser le langage Java et le framework Spring pour la partie Backend
- utiliser javascript et le framework vueJs pour la partie Frontend
- utiliser la base de données MongoDb pour le stockage

Définition des User Stories

La définition des User Stories permet de bien capter le besoin du client. Dans notre projet On peut déjà définir qu'un "Utilisateur" peut être "Lecteur" ou "Auteur".

En tant que Lecteur :

- Je veux avoir une prévisualisation des articles sur mon espace de travail, afin de pouvoir sélectionner rapidement l'article qui m'intéresse.
- Je veux voir la liste des articles disponibles, afin de choisir ceux qui m'intéressent.
- Je veux pouvoir filtrer les articles par catégorie, auteur ou date, afin de trouver rapidement un contenu spécifique.
- Je veux rechercher un article par mot-clé, afin d' accéder rapidement à une information précise.
- Je veux lire le contenu complet d'un article, afin de profiter de sa lecture sans distraction.

En tant qu'Auteur :

- Je veux pouvoir créer un nouvel article, afin de publier du contenu sur la plateforme.
- Je veux pouvoir enregistrer un article en brouillon, afin de le finaliser plus tard.
- Je veux modifier un article existant, afin de corriger ou améliorer mon contenu.
- Je veux pouvoir publier un article, afin d'afficher son contenu aux lecteurs.
- Je veux pouvoir dé-publier un article, afin de ne pas l'afficher aux lecteurs.
- Je veux pouvoir ajouter une image à mon article afin de le rendre plus attractif.
- Je veux voir la liste de mes articles (publiés, dé-publiés et brouillons), afin de gérer efficacement ma production.

Définition des Use Case

J'ai défini les **use case** ou cas d'utilisations avec l'aide de mon tuteur Fabrice Turpeau.

Cas d'utilisation : Utilisateur

Voir en annexe 3 le diagramme UML.
Un utilisateur peut être lecteur et/ou auteur selon l'interface qu'il utilise.
Dans ces deux cas, il a été décidé que les accès sécurisés seraient gérés par l'entité supérieure à savoir la plateforme PDF.

Cas d'utilisation : Lecteur

Voir annexe 4 le diagramme UML.
L'utilisateur qui est lecteur peut en utilisant l'interface de lecture :

- cas N°1 : Lire un article en accédant à la liste des articles puis à son contenu complet.
- cas N°2 : Rechercher un article en tapant en barre de recherche (un mot, un auteur)
- cas N°3 : Trier / Filtrer les articles en renseignant des critères (par date, ordres alphabétique, auteur, articles)

Cas d'utilisation : Auteur

Voir annexe 5 le diagramme UML.
L'Utilisateur qui est Auteur peut en utilisant l'interface auteur :

- cas N°1 : Créer un article en renseignant les différents champs du formulaire. En fonction des différentes vérifications métiers l'article est sauvegardé en tant que brouillon ("DRAFT") avec une date et heure de création actuelles.
- cas N°2 : Publier un article, en vérifiant le statut préalable de "brouillon" ou "dépublié".
- cas N°3 : Dé-publier un article, en vérifiant le statut préalable "publié".
- cas N°4 : Modifier un article en récupérant les informations déjà saisies, en les rectifiant au besoin et en enregistrant ces nouvelles informations.

Maquettage

Wireframes

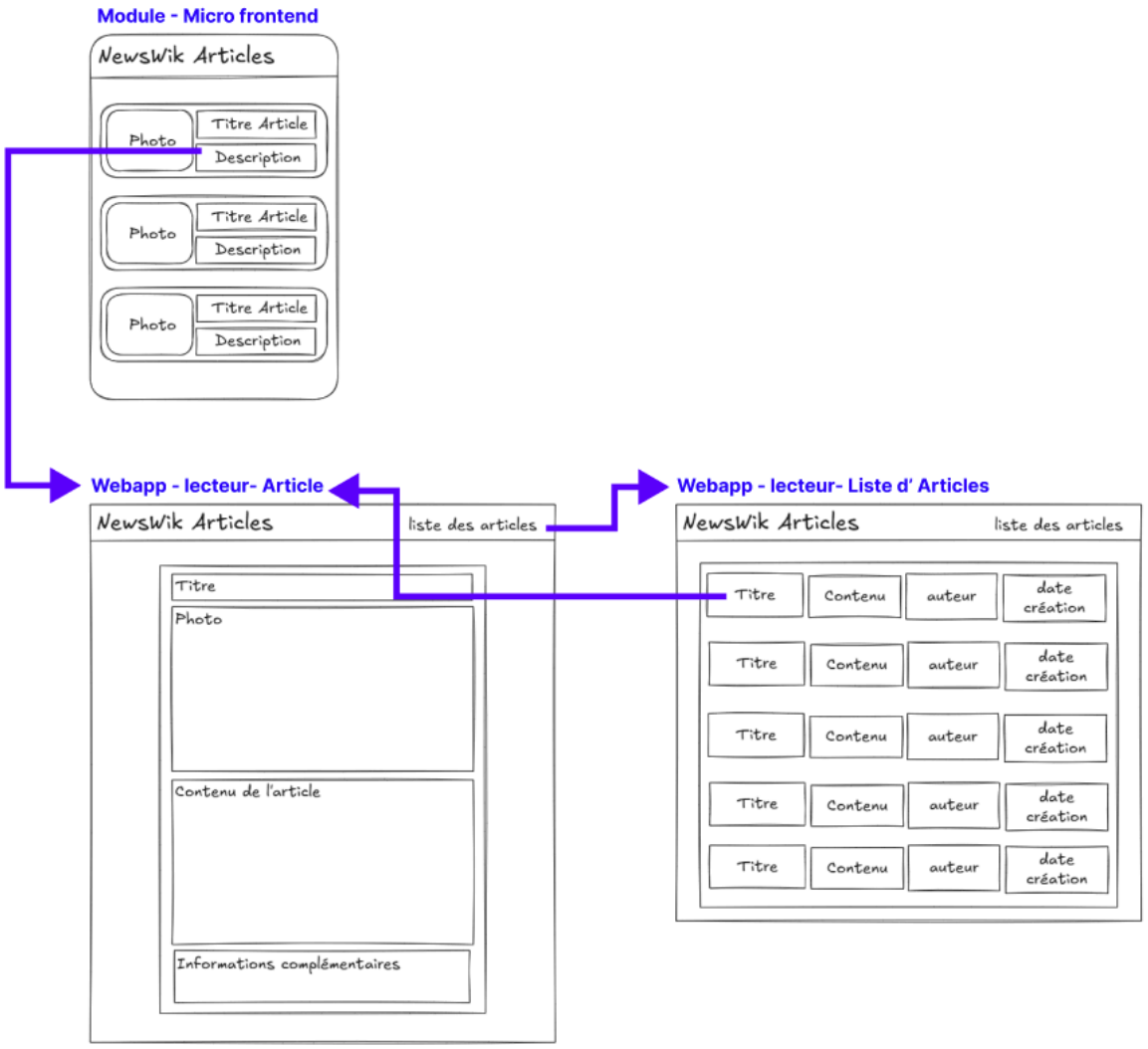
Voici quelques Wireframes que j'ai dessiné pour le prototypage du projet avec l'équipe.

Module "Lecteur" Mfe-preview

Le module micro frontend(mfe) est le widget qui sera inséré dans une page "hôte" au même titre que les widget météo et éphéméride. Il contient une sélection d'articles miniaturisés avec pour chacun la photo, le titre et la description.

Application "Lecteur"

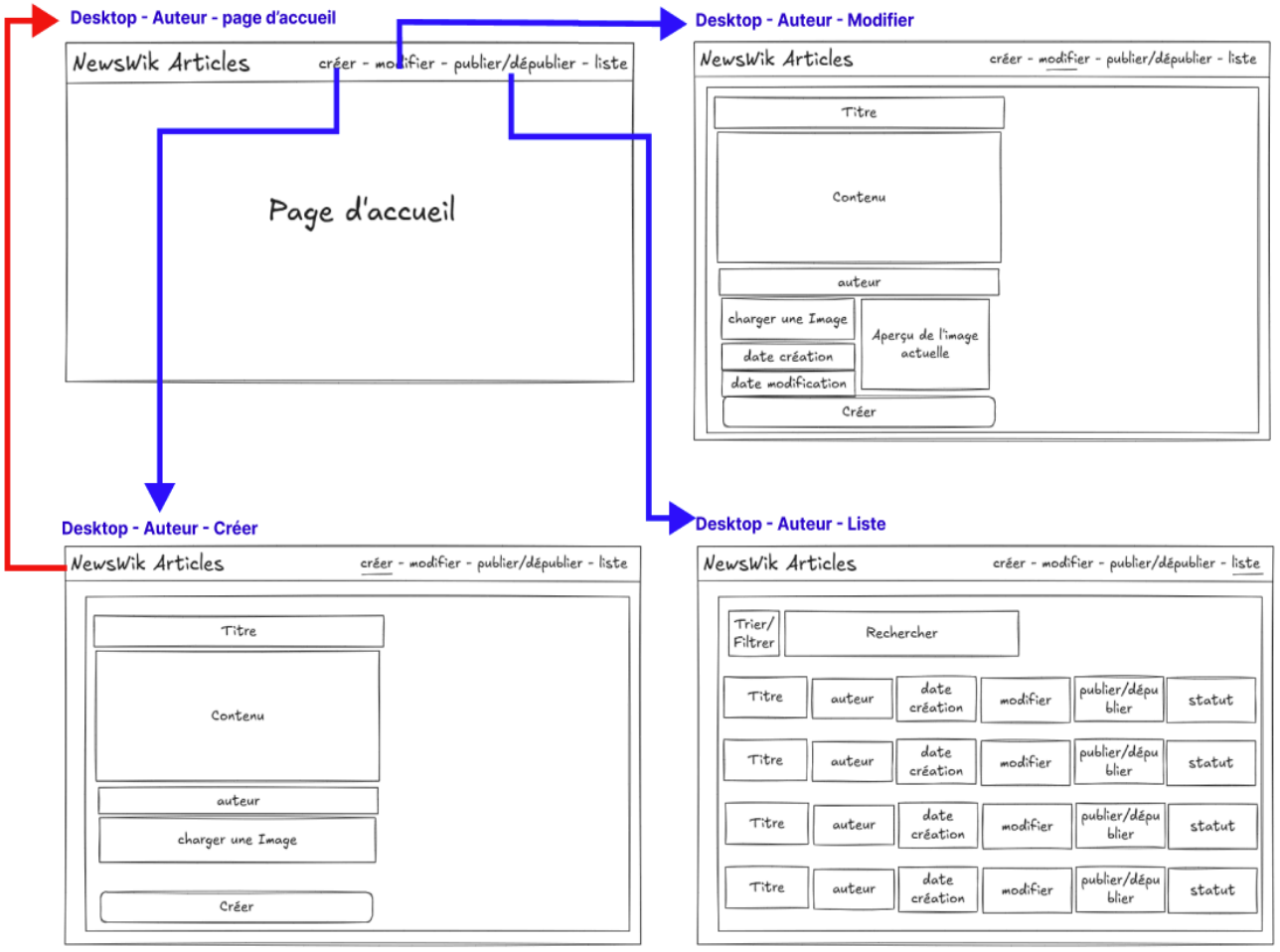
On accède à l'application lecteur par le module micro frontend. Cette application permet ensuite de lire l'article complet et d'accéder à d'autres articles.



Application "Auteur"

L'application auteur permet la navigation dans les différentes pages :

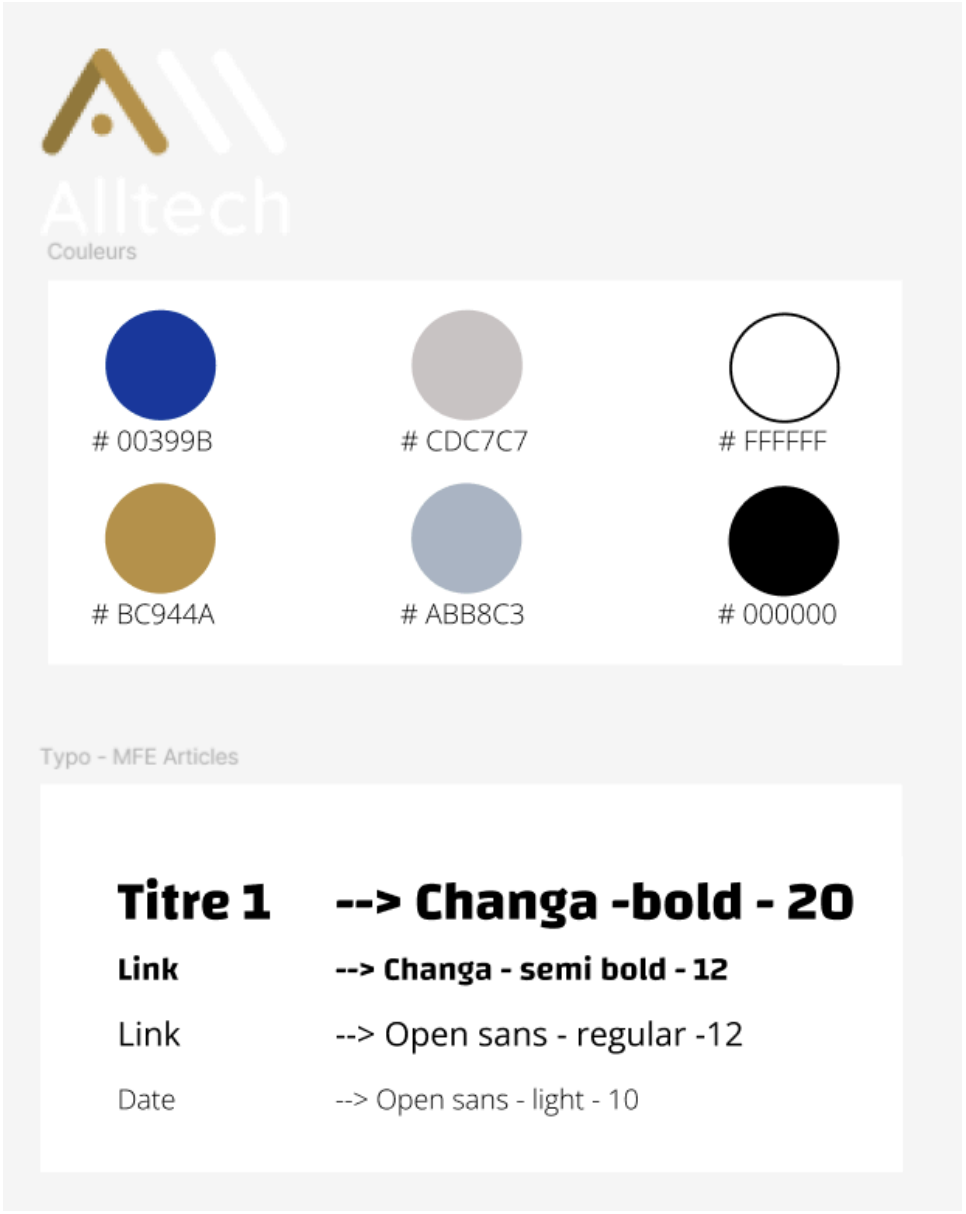
- page d'accueil
- créer un article
- modifier un article
- publier / dépublier un article
- l'accès à la liste de tous les articles qui est aussi un moyen d'accéder à toutes les fonctionnalités précitées



Identité graphique

Voici la charte graphique que j'ai mise en place sous **Figma**.
J'ai récupéré les couleurs dominantes que j'ai trouvées sur le site internet de l'entreprise Alltech Consulting.

Charte graphique



Maquettes

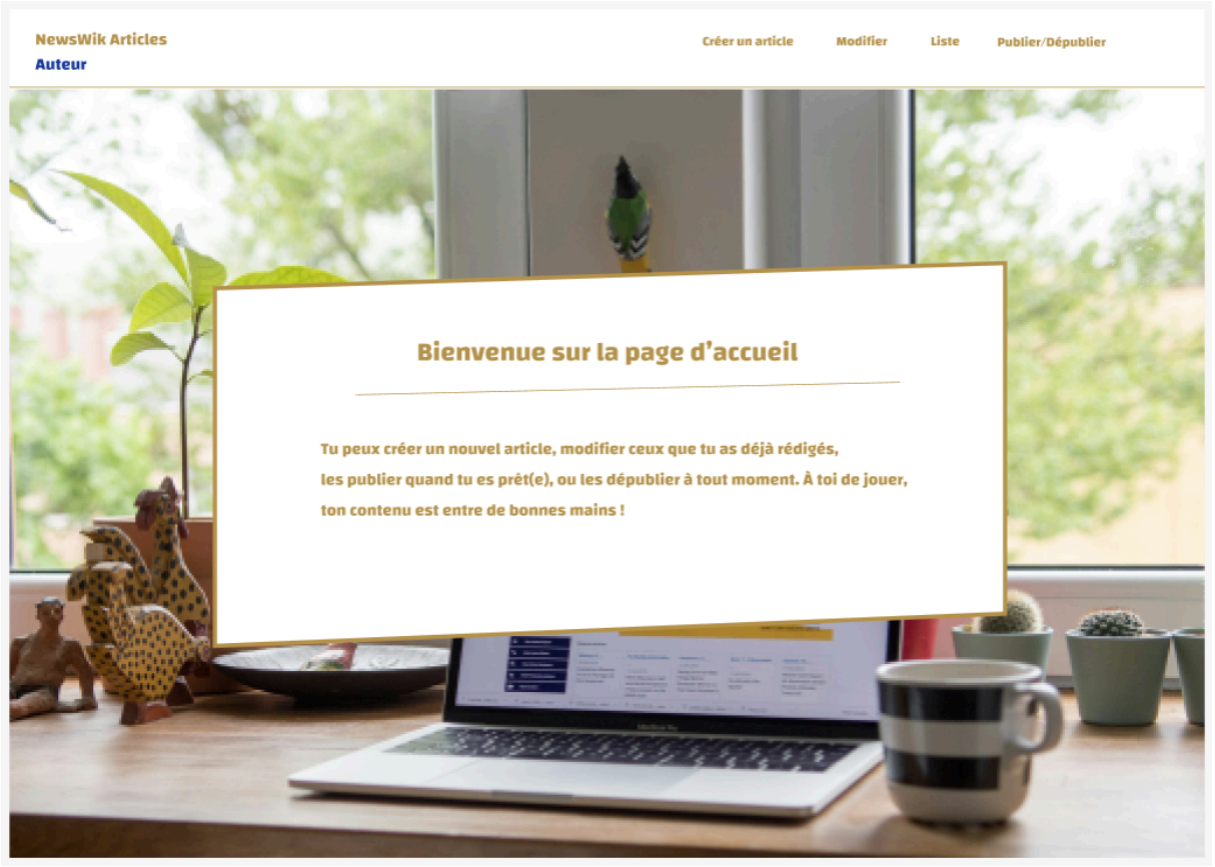
Voici quelques-unes des maquettes qui ont toutes été réalisées sous **Figma**.

Maquette(mobile) - mfe-preview

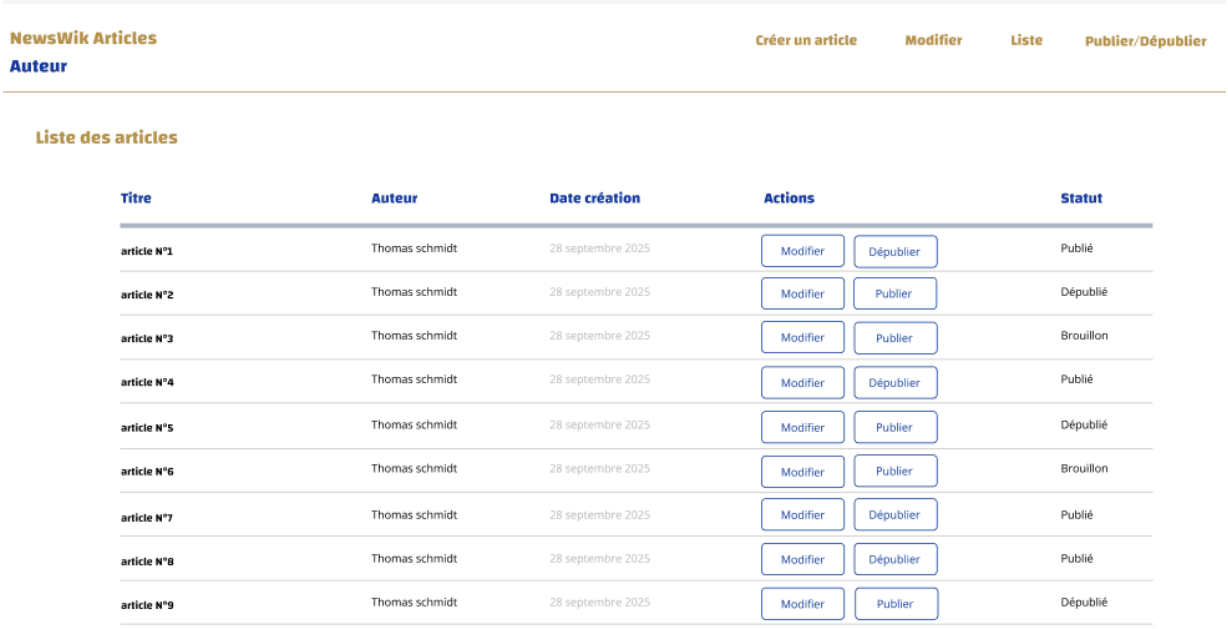
En annexe 7 un aperçu du mode "inspecter" de Microsoft Edge présente l'interface et son code.

Maquette(mobile) - lecture d'un article

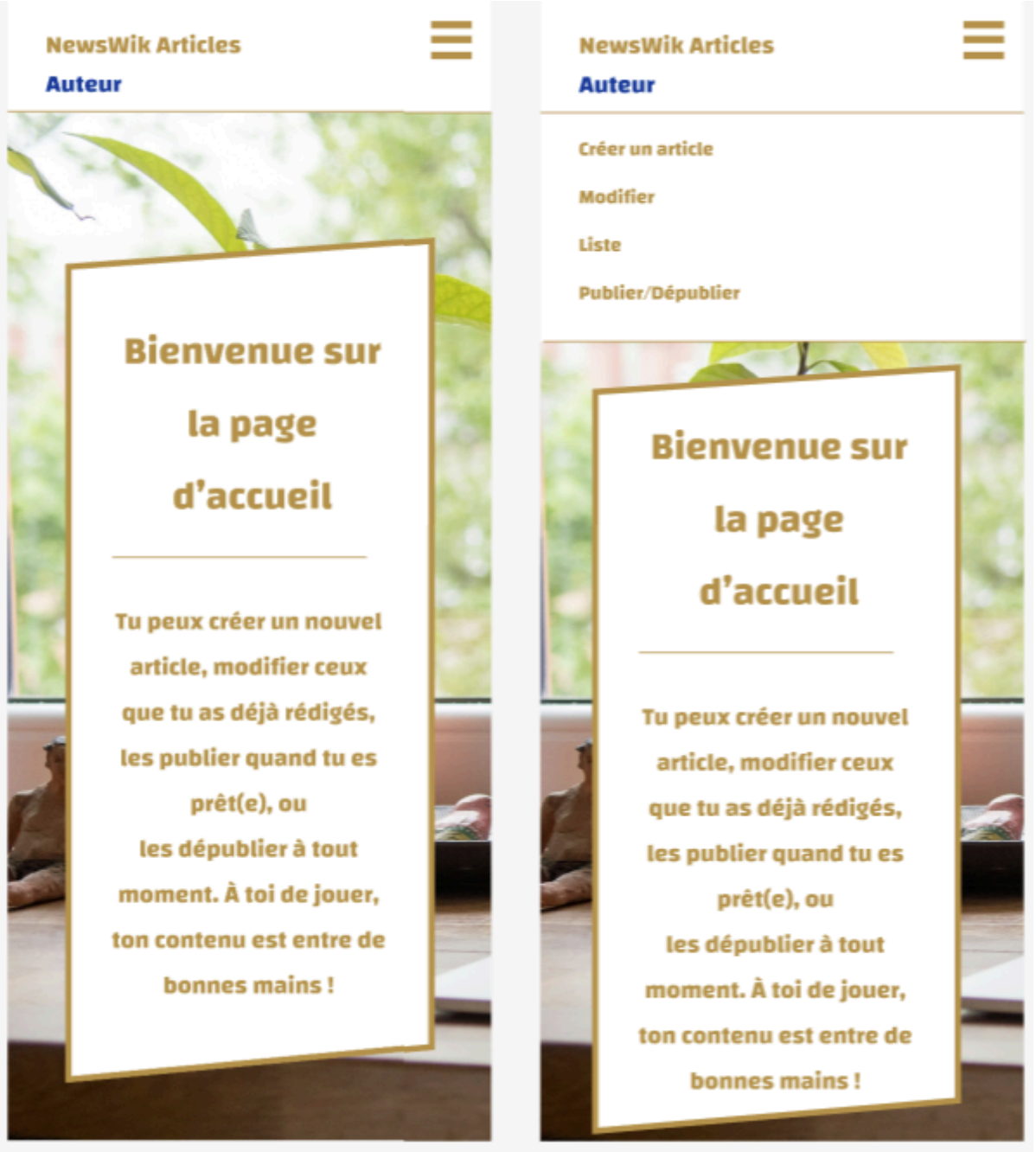
Maquette(Desktop) - Auteur - Accueil



Maquette(Desktop)- Auteur - Liste



Maquette(mobile) - Auteur - Accueil & menu déplié



Architecture du projet

L'architecture du projet repose sur une séparation claire entre les couches Front-End et Back-End, favorisant la modularité, la maintenabilité et l'évolution de l'application.
Voir Annexe 6 Découpage back/front.

Back-End

La partie Back-End, dédiée à la gestion de l'API, est développée en Java 21, une version LTS (Long-Term Support) garantissant stabilité et support à long terme.
Elle s'appuie sur le framework Spring, qui facilite la création de services RESTful et la gestion des dépendances. Le gestionnaire de build Gradle est utilisé pour la compilation, la gestion des dépendances du projet.
Il s'agit d'un développement en couches (Controller/Service/Dto/Model/Repository). Pour une meilleur :

- séparation des responsabilités,
- testabilité
- lisibilité et maintenance
- facilité d'évolution (changer la bdd par exemple)

Pour le stockage c'est MongoDb qui a été choisie car c'est une base de données orientée documents (NoSQL). Chaque article est stocké sous forme de document JSON (ou BSON) ce qui permet une structure souple et modifiable. MongoDB est optimisé pour les opérations de lecture rapide et la recherche textuelle.
Les données peuvent être hétérogènes (texte long, contenu HTML, Markdown, images intégrées, commentaires imbriqués…) mais MongoDB les gère naturellement, sans devoir créer de multiples tables et relations complexes (contrairement à un modèle SQL).

Front-End

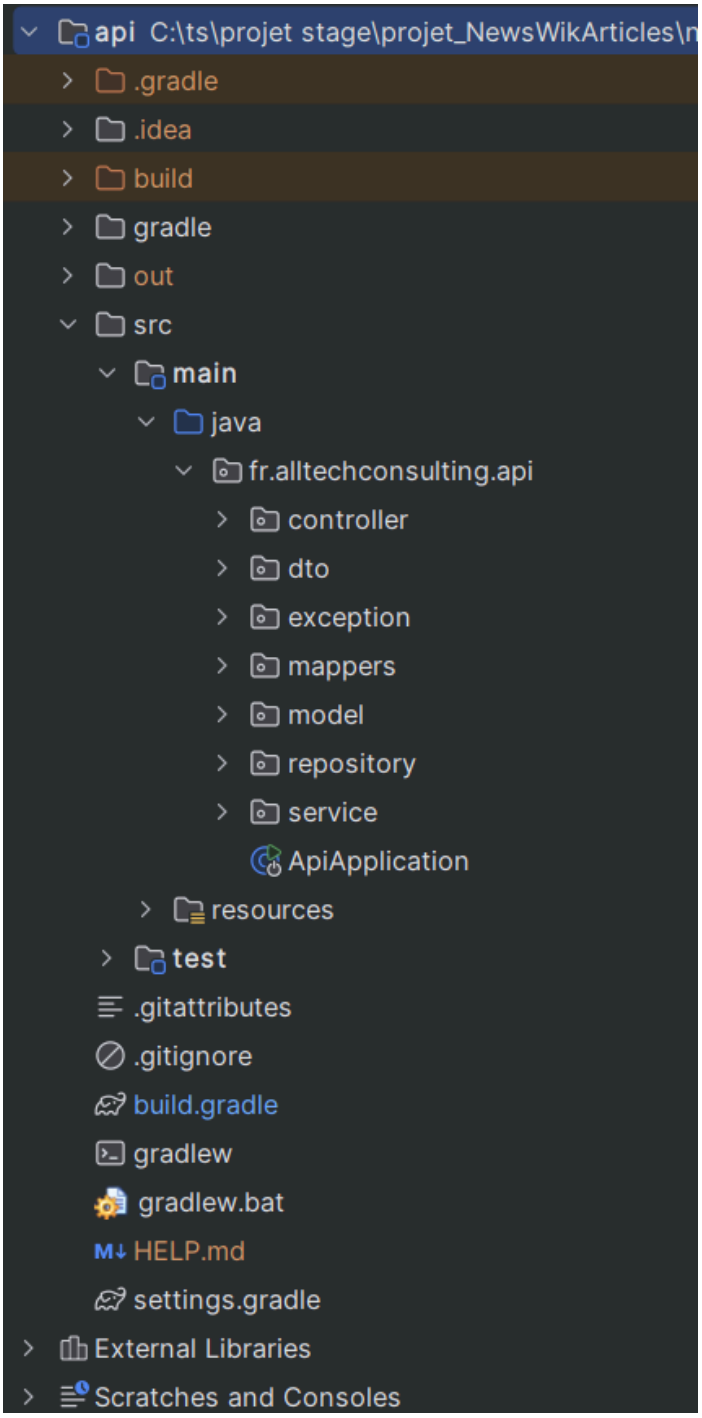
Les interfaces utilisateur — interface lecteur et interface auteur — sont développées avec le framework Vue.js (version 3.5.18), exécuté dans un environnement Node.js.

Le framework **PicoCSS** est utilisé pour la mise en forme des composants graphiques et pour garantir un design responsive, assurant ainsi une bonne adaptabilité sur différents types d'écrans (ordinateurs, tablettes, smartphones). Mais des spécificités sur chaque page sont possibles grâce au concept de **vue.js** qui autorise du **CSS, HTML** & **JS** sur chaque fichier **.vue** .

Partie Back-End du projet

La partie Backend comporte la base de données et l'API dont voici l'arborescence du projet.

Arborescence du projet



Base de données

Il a été fait le choix de la simplicité pour la base de données.
MongoDB étant orienté documents (NoSQL), on ne parle pas de tables ni de relations strictes comme en SQL, mais plutôt de collections et documents imbriqués.

L'enregistrement de l'image depuis le formulaire du frontend se fait en base64 puis est codée en "binary" pour le stockage en bdd.

Les articles sont enregistrés dans la base de données MongoDB au sein d'une collection appelée **posts**.
Chaque document de cette collection représente un article unique, contenant notamment :

- le contenu de l'article,
- le nom de l'auteur,
- diverses métadonnées (date de création, catégories, etc.),
- ainsi que le lien ou les données de l'image associée.

Cette collection est utilisée par notre application de gestion d'articles pour stocker et gérer l'ensemble des publications.

Exemple d'un article en base de données


```
{
  "_id": {
    "$oid": "68f0f2749a7d294e30d56027"
  },
  "name": "Le climat change rapidement",
  "content": "Le changement climatique est devenu l'un des défis les plus urgents de no
  "author": "Alice Dupont",
  "date_created": {
    "$date": "2025-10-16T13:26:12.362Z"
  },
  "date_updated": {
    "$date": "2025-10-16T13:26:12.744Z"
  },
  "image": {
    "$binary": {
      "base64": "iVBORw0KGgoAAAANSUhEUgAAA4EAAAH7CAYAAAByjJpCAAAAAXNSR0IArs4c6QAAAARnQU
      "subType": "00"
    }
  },
  "imageContentType": "image/png",
  "status": "DRAFT",
  "_class": "fr.alltechconsulting.api.model.Article"
}
```


Ce fichier JSON correspond à la représentation BSON stockée dans MongoDB.
Les marqueurs comme **$oid**, **$date**, ou **$binary** sont utilisés par MongoDB pour encoder les types natifs :
$oid → identifiant unique (ObjectId)
$date → valeur de type Date
$binary → donnée binaire encodée en base64
Le champ **_class** indique qu'on utilise Spring Data MongoDB : c'est ajouté automatiquement. La métadonnée fait le lien entre la collection MongoDB et la classe Java Article.

Détails du document

Champ	Type	Description
_id	**ObjectId**	Identifiant unique généré automatiquement par MongoDB pour ce document.
name	**String**	Titre de l'article. Ici : *"Le climat change rapidement"*.
content	**String**	Contenu principal de l'article, sous forme de texte enrichi.
author	**String**	Nom de l'auteur ayant rédigé l'article. Ici : *Alice Dupont*.
date_created	**Date**	Date et heure de création de l'article (au format ISO).
date_updated	**Date**	Date de dernière mise à jour de l'article.
image	**Binary (BSON)**	Donnée binaire encodée en base64, représentant l'image associée à l'article.
imageContentType	**String**	Type MIME de l'image (ici image/png). Permet d'indiquer comment interpréter la donnée binaire.
status	**String**	État de publication de l'article : **"DRAFT", "PUBLISHED"**, "**UNPUBLISHED**".
_class	**String**	Métadonnée ajoutée par l'ORM Spring Data pour MongoDB, indiquant la classe Java associée **(fr.alltechconsulting.api.model.Article)**.

Stockage de l'image

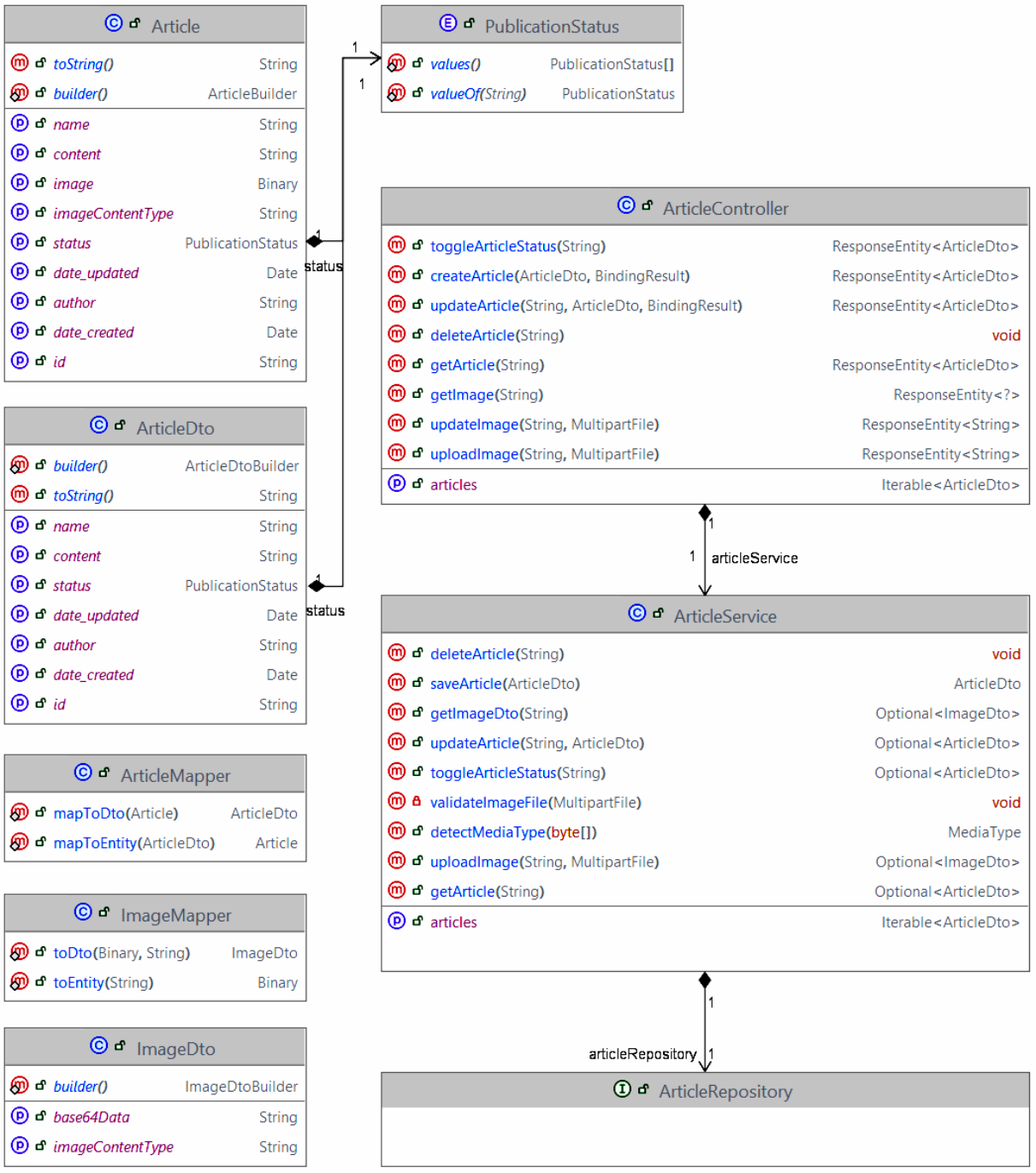
L'image associée à chaque article est stockée directement dans le document **MongoDB** sous forme binaire, plutôt que sur un serveur distinct avec uniquement un chemin d'accès enregistré dans la base de données (méthode plus courante).
Cette approche permet de récupérer facilement l'article complet avec son image intégrée.
Cependant, elle peut augmenter la taille des documents lorsque les images sont volumineuses.
Dans notre cas, cet inconvénient a été compensé par l'utilisation d'appels d'API qui séparent les champs texte et image de chaque article, optimisant ainsi les échanges et les performances de l'application.

L' API

Diagramme des classes

Voici un diagramme de classes qui permet une vue d'ensemble (mis en forme par IntelliJ) avec les méthodes et propriétés.Il montre aussi les relations entre les classes.

Diagramme de classes de l'API



La couche Model

Les types de champs de la collection **MongoDB** sont définis à partir de l'entité **Article.java**, située dans le package model de l'**API**.
Ainsi, la structure des documents stockés dans la collection correspond directement à la modélisation de la classe **Java**, garantissant la cohérence entre la base de données et le code de l'application.

La classe Article.java

```java
package fr.alltechconsulting.api.model;

import com.fasterxml.jackson.annotation.JsonIgnore;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;
import lombok.Getter;
import lombok.Setter;
import lombok.Builder;
import lombok.ToString;
import org.bson.types.Binary;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;
import java.util.Date;

@Getter
@Setter
@ToString
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Document(collection = "posts")
public class Article {

    @Id
    private String id;
    @Indexed(unique = true)
    private String name;
    private String content;
    private String author;
    private Date date_created;
    private Date date_updated;

    @JsonIgnore
    private Binary image;
    private String imageContentType;

    private PublicationStatus status = PublicationStatus.DRAFT;

}
```

La couche Repository

L'accès au données se fait par la couche repository et précisément par l'interface "ArticleRepository.java"

L'interface ArticleRepository.java

```java
package fr.alltechconsulting.api.repository;

import fr.alltechconsulting.api.model.Article;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ArticleRepository extends MongoRepository<Article, String> {
}
```

Détails de l'interface

L'annotation **@Repository** indique à Spring que c'est un élément de la couche persistance.

L'interface hérite de **MongoRepository** :

- Le premier paramètre générique est une entité de type "Article"
- Le second paramètre se réfère au type de l'identifiant (**@Id** de la classe Article).

En héritant de **MongoRepository**, cette interface a donc accès à tout un panel de méthodes sans avoir besoin de les implémenter comme par exemple :

- **save(Article article)** Sauvegarde ou met à jour un article
- **findById(String id)** Recherche d'un article par son identifiant
- **findAll()** Renvoi tous les articles
- **deleteById(String id)** Supprime un article par son identifiant
- et d'autres encore…

Il serait néanmoins possible de faire un **@Override** de ces méthodes afin de les modifier mais celles de base nous conviennent parfaitement.

La couche Service

La couche service va maintenant pouvoir utiliser les méthodes héritées de "MongoRepository<>"
Cette couche contient les besoins métiers de l'application.
La couche service fait appel aux **mappers** pour convertir les objets selon les besoins de l'application.
Ces **mappers** permettent de créer ou de transformer les objets DTO (Data Transfer Objects) à partir des entités du modèle, ou inversement, afin de faciliter les échanges entre les différentes couches de l'application.
Il y a donc une séparation entre modèle métier et modèles de transfert.

Extrait de la classe ArticleService.java

```java
public class ArticleService {

    @Autowired
    private ArticleRepository articleRepository;

    public Optional<ArticleDto> getArticle(String id) {
        return articleRepository.findById(id)
                .map(article -> ArticleMapper.mapToDto(article));
    }

    public Iterable<ArticleDto> getArticles() {
        return articleRepository.findAll()
                .stream()
                .map(article -> ArticleMapper.mapToDto(article))
                .collect(Collectors.toList());
    }

    public void deleteArticle(String id) {
        articleRepository.deleteById(id);
    }

    public ArticleDto saveArticle(ArticleDto articleDto) {
        Article article = articleRepository.save(ArticleMapper.mapToEntity(articleDto));
        return ArticleMapper.mapToDto(article);
    }

    public Optional<ArticleDto> updateArticle(String id, ArticleDto articleDto) {
        Optional<Article> optionalArticle = articleRepository.findById(id);

        if (optionalArticle.isEmpty()) {
            return Optional.empty();
        }

        Article currentArticle = optionalArticle.get();

        if (articleDto.getName() != null) {
            currentArticle.setName(articleDto.getName());
        }
        if (articleDto.getContent() != null) {
            currentArticle.setContent(articleDto.getContent());
        }
        if (articleDto.getAuthor() != null) {
            currentArticle.setAuthor(articleDto.getAuthor());
        }
        if (articleDto.getStatus() != null) {
            currentArticle.setStatus(articleDto.getStatus());
        }

        currentArticle.setDate_updated(new Date());

        Article savedArticle = articleRepository.save(currentArticle);
        return Optional.of(ArticleMapper.mapToDto(savedArticle));
    }
}
```

Le package Dto

Les attributs de l'entité Article sont répartis dans deux DTO, **ArticleDto.java** et **ImageDto.java** . L' **ArticleDto** contient, sur les attributs name, content et author, des contraintes de validations matérialisées par **@NotBlank** et **@Size** de la bibliothèque "**jakarta.validation.constraints.NotBlank**" ou "**.Size**" .

Extrait de la classe ArticleDto.java

```
@Getter
@Setter
@ToString
@Builder
@AllArgsConstructor @NoArgsConstructor
public class ArticleDto {

    private String id;

    @NotBlank(message = "Le titre est requis")
    @Size(min = 2, max = 50, message = "Le titre doit contenir entre {min} et {max} caractères")
    private String name;

    @NotBlank(message = "Le contenu est requis")
    @Size(min = 2, max = 5000, message = "Le contenu doit contenir entre {min} et {max} caractères")
    private String content;

    @NotBlank(message = "L'auteur est requis")
    @Size(min = 2, max = 50, message = "Le nom de l'auteur doit contenir entre {min} et {max}
caractères")
    private String author;

    private Date date_created;
    private Date date_updated;

    private PublicationStatus status = PublicationStatus.DRAFT;
}
```

Le package Mapper

Selon le sens du flux de données, la classe **ArticleMapper** permet soit de construire un DTO à partir d'un objet Article grâce à la méthode **mapToDto()**, soit de convertir un DTO en entité **Article** via la méthode **mapToEntity()**.
Cette dernière opération est notamment utilisée pour transmettre les données à la couche **repository**, en vue de leur enregistrement dans la base de données.

Extrait de la classe ArticleMapper.java

```
public class ArticleMapper {

    public static ArticleDto mapToDto(Article article) {
        ArticleDto articleDto = ArticleDto.builder()
                .id(article.getId())
                .name(article.getName())
                .content(article.getContent())
                .author(article.getAuthor())
                .date_created(article.getDate_created())
                .date_updated(article.getDate_updated())
                .status(article.getStatus())
                .build();
        return articleDto;
    }

    public static Article mapToEntity(ArticleDto articleDto)
{       Article article = Article.builder()
                .id(articleDto.getId())
                .name(articleDto.getName())
                .content(articleDto.getContent())
                .author(articleDto.getAuthor())
                .date_created(articleDto.getDate_created())
                .date_updated(articleDto.getDate_updated())
                .status(articleDto.getStatus())
                .build();
        return article;
    }

}
```

La couche Controller

ArticleController est la façade de l'application, elle expose les opérations **CRUD** (Create Read Update Delete) en s'appuyant sur la couche Service.
La couche Controller reçoit les requêtes **HTTP** de l'extérieur. Elle est vulnérable aux attaques et doit être sécurisée par l'annotation "**@CrossOrigin**" qui verrouille l'accès aux utilisateurs externes (politiques **CORS**, Cross Origin Resource Sharing). Ici deux utilisateurs autorisés (**localhost:5173** et **localhost:5174**).
En plus l'annotation **@CrossOrigins** ne permet que quelques types de méthodes (**GET**, **POST**, **PATCH**, **PUT**, **DELETE**).
Dès le démarrage de l'application, l'injection de dépendances par constructeur permet de réduire le couplage entre les classes.

Extrait de la classe ArticleController.java

```
@CrossOrigin(origins = {"http://localhost:5173", "http://localhost:5174"},
        methods = {
                RequestMethod.GET,
                RequestMethod.POST,
                RequestMethod.PATCH,
                RequestMethod.PUT,
                RequestMethod.DELETE},
        allowedHeaders = "*",
        allowCredentials = "true")
@RestController
public class ArticleController {

    private final ArticleService articleService;

    // Injection par constructeur
    public ArticleController(ArticleService articleService) {
        this.articleService = articleService;
    }


    /**
     * CREATE - Add a new article
     *
     * @param articleDto an object article
     * @return The article object saved
     */
    @PostMapping("/article")
    public ResponseEntity<ArticleDto> createArticle(@Valid @RequestBody ArticleDto articleDto,
BindingResult result) {
        if (result.hasErrors()) {
            return ResponseEntity.badRequest().build();
        }
        ArticleDto savedArticle = articleService.saveArticle(articleDto);
        return ResponseEntity.status(HttpStatus.CREATED).body(savedArticle);
    }
```

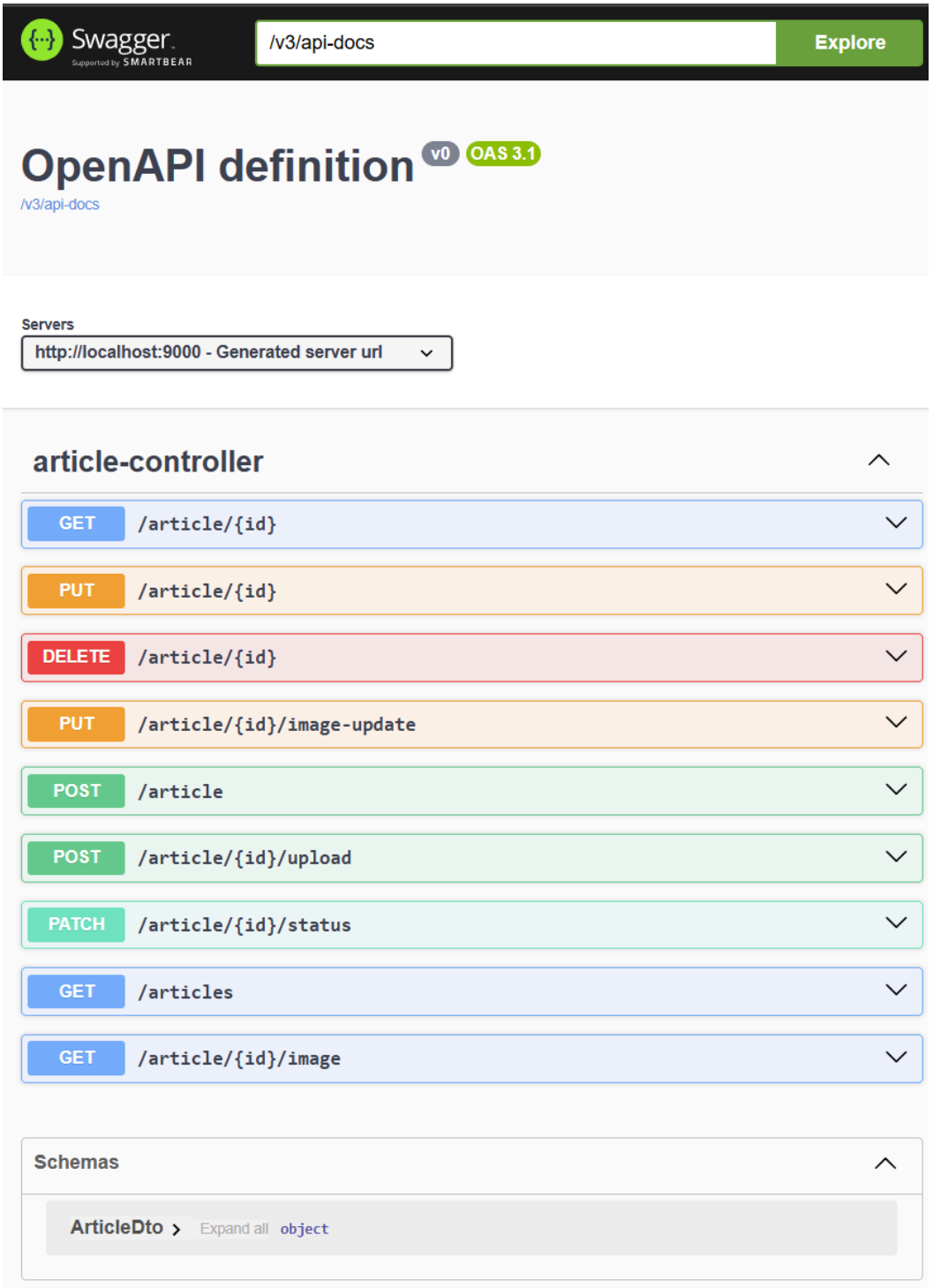
En annexe 9 la classe ArticleController.java affiche toutes les opérations du CRUD.

Endpoints

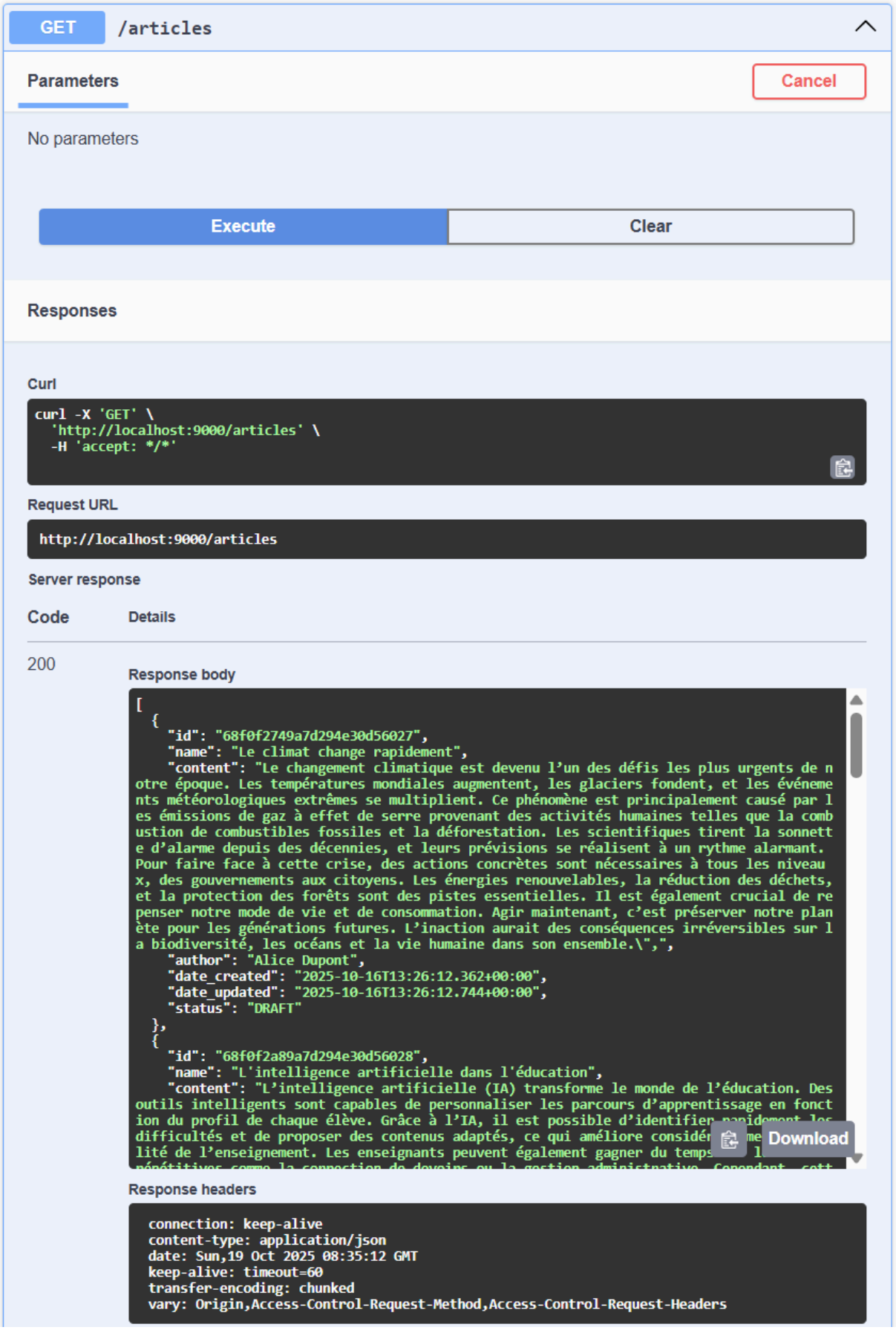
Les ***Endpoints*** sont les points d'entrées qui permettent d'interroger la base de données par l'intermédiaire de l'API.
J'utilise **Swagger** et la dépendance **'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.8.6'** pour générer la documentation qui servira à l'utilisateur

de l'application (programmeur côté frontend par exemple). De cette façon, il pourra utiliser l'API correctement et éventuellement tester des requêtes.

Documentation des EndPoints de l'API



Exemple Requête GET/articles



Tests de l'API

Les tests de l'API peuvent être réalisés de deux manières complémentaires :

Tests simulés (automatisés) : à l'aide de l'environnement de test de **Spring Boot**, notamment avec les annotations **@SpringBootTest** et l'outil **MockMvc**. Cela permet d'écrire des programmes de test qui simulent des appels **HTTP** et vérifient automatiquement les réponses, les statuts, et les comportements attendus. Ce type de test est utile pour automatiser la validation des fonctionnalités et s'intègre facilement dans un processus d'intégration continue (**CI**).

Tests réels : en lançant l'**API** localement, puis en utilisant un outil comme **Postman** pour envoyer des requêtes **HTTP** manuelles (**GET**, **POST**, **PUT**, **DELETE**, etc.) et observer les réponses. Ce type de test permet de valider le comportement global de l'**API** dans des conditions proches de la production.

Détails du test simulé : requête POST ("/article")

1- Je définis le contenu d'une requête (**payload**) **POST** de création d'article.
2- Je simule l'envoi de la requête avec :
mockMvc.perform(post("/article).contentType("application/json").content(payload)
3- Je contrôle le retour de l'API :

- **.andExpect(status().isCreated())** est le status Http 201 attendu.
- **.andExpect(content().contentTypeCompatibleWith("application/json"))** assure la compatibilité entre les types de contenu.
- **.andExpect(jsonPath("$.name").value("Title test"))** pour vérifier le contenu du champ name.
- **.andExpect(jsonPath("$.id").exists())** je vérifie que l'id est bien renvoyé.

Extrait du test de la requête POST ArticleControllerTest.java

```
@Test
@DisplayName("POST /article enregistre un article Test")
public void testCreateArticle() throws Exception {
    String payload = """
                {
                  "id": "1",
                  "name": "Title test",
                  "content": "Content test",
                  "author": "Author test",
                  "date_created": "2025-10-19T08:48:01.224Z",
                  "date_updated": "2025-10-19T08:48:01.224Z",
                  "status": "DRAFT"
                },
            """;

    // Vérification directe sur la réponse du POST
    mockMvc.perform(post("/article")
                    .contentType("application/json")
                    .content(payload))
            .andExpect(status().isCreated())
            .andExpect(content().contentTypeCompatibleWith("application/json"))
            .andExpect(jsonPath("$.name").value("Title test"))
            .andExpect(jsonPath("$.content").value("Content test"))
            .andExpect(jsonPath("$.author").value("Author test"))
            .andExpect(jsonPath("$.id").exists());
```

Détails du test simulé : la requête GET (“/articles”)

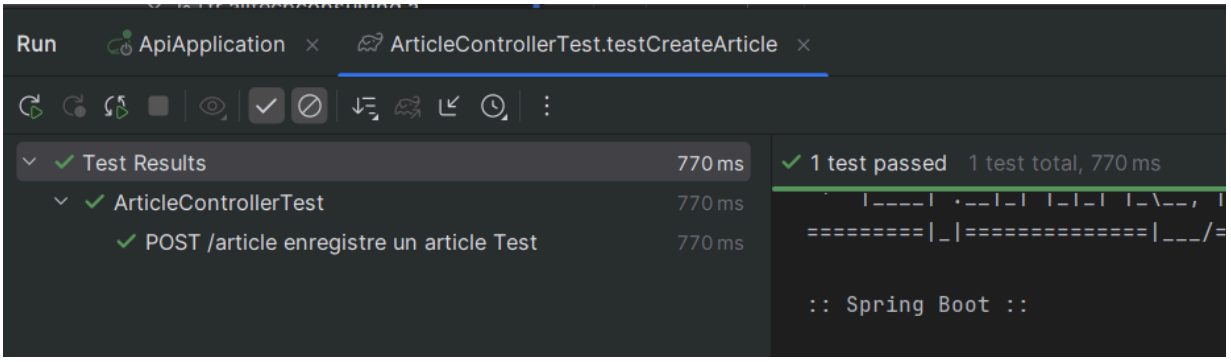
1- Je lance une requête **GET** (“articles”) lecture de tous les articles.
2- Je surveille le statut de retour “**isOk()**” qui est le statut **HTTP** 200.
3- Le type de retour doit être de type **json**
4- Je cherche dans la liste qu’il y a bien un champ avec l’expression :

- jsonPath **"$[?(@.name == 'Title test' && @.content == 'Content test' && @.author == 'Author test')]"**

Extrait du test de la requête GET ArticleControllerTest.java

```
        // Verifications dans la liste d'articles
        mockMvc.perform(get("/articles"))
                .andExpect(status().isOk())
                .andExpect(content().contentTypeCompatibleWith("application/json"))
                .andExpect(jsonPath("$[?(@.name == 'Title test' && @.content == 'Content test' &&
@.author == Author test')]").exists());
    }
```

Résultats du test POST sous IntelliJ



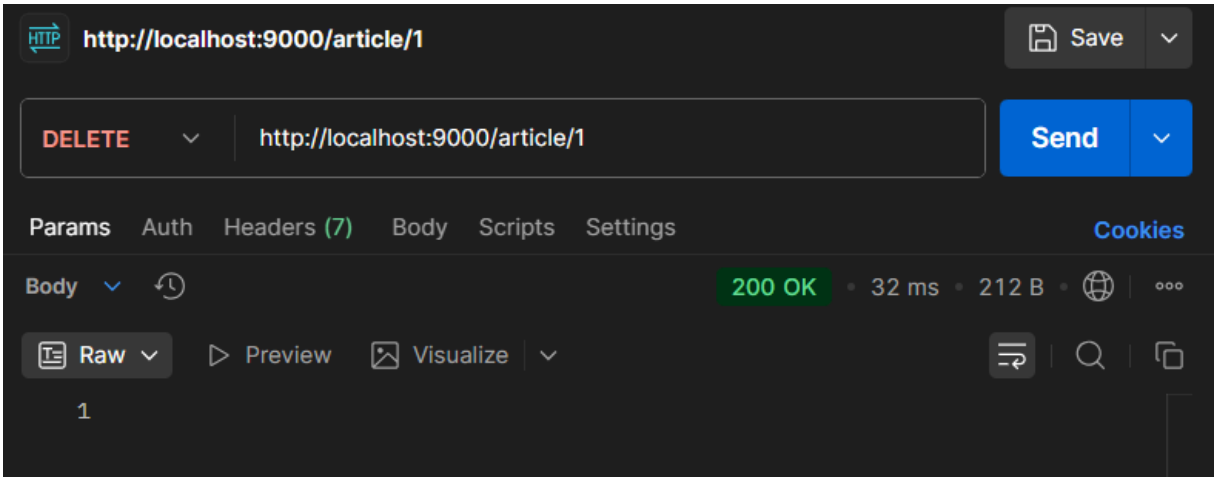
Test réels avec Postman

Il faut lancer l'API puis le logiciel **Postman**.

Exemple de test de la Requête DELETE

sur **http://localhost:9000/article/1** va nous permettre de supprimer l'article précédemment créé. Je constate un status **200** (OK) qui nous indique que la requête s'est effectuée correctement.

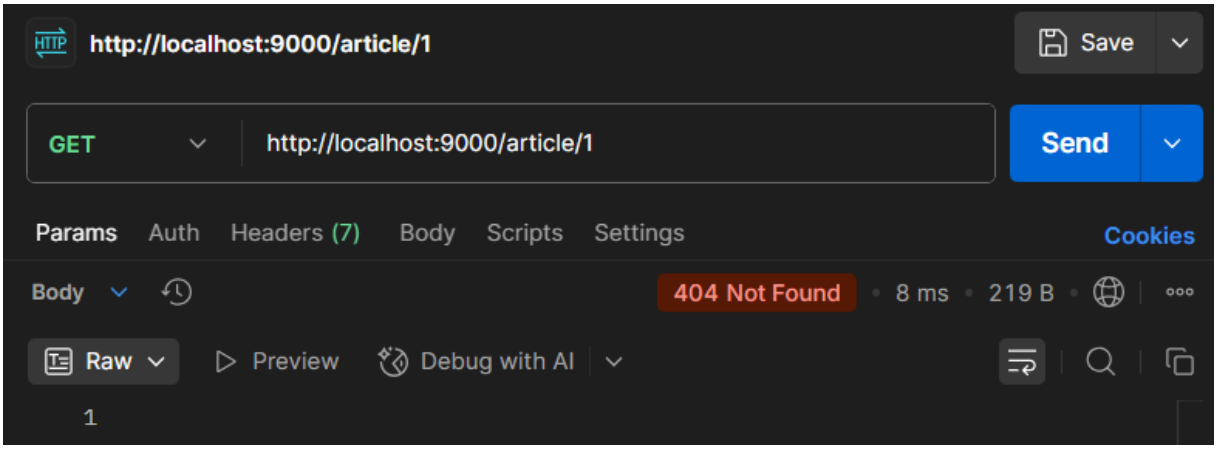
Résultat de la requête Delete sur Postman



Exemple de test de la Requête GET

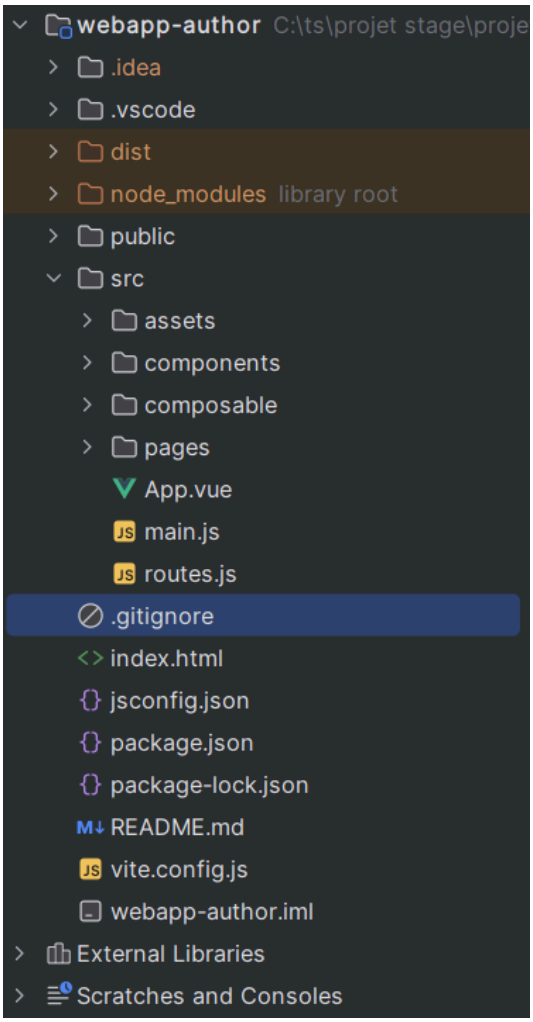
sur **http://localhost:9000/article/1** va nous permettre de voir si l'article est toujours présent en bdd Je constate le statut **404** (NOT FOUND). Il nous indique que la requête n'a pas trouvée ce qui a été demandé.

Résultat de la requête Get sur Postman



Partie Front-End du projet

Arborescence du projet



Il y a deux parties Frontend sur mon projet, mais à partir de maintenant nous ne parlerons que de l'application webapp-author car c'est celle qui regroupe le plus de fonctions.
Elle est globalement programmée en javascript, CSS et .vue .

Exemple de fichier VueJs

Une des particularités de **VueJs** est que son fichier ".vue" contient :

- de l'**HTML** entre les balises **<template></template>**
- du **javascript** ou du **Typescript** entre les balises **<script></script>**
- du **CSS** entre les balises **<style></style>**

La page de création d'article (CreatePage.vue) utilise un *component* qui s'appelle **<Postform />**. Ce *component* contient les champs du formulaire de création d'articles que j'ai regroupé dans un composant pour une meilleure lisibilité du code et d'une potentielle réutilisation future.

*Extrait de **CreatePage.vue***

```
<template>
  <Header/>
  <div class="container">
    <h1>Créer un article</h1>
    <PostForm/>
  </div>
</template>

<script setup>
import PostForm from "@/components/PostForm.vue";
import Header from "@/components/Header.vue";
</script>

<style scoped>
</style>
```

En annexe 8 un aperçu du mode “inspecter” de Microsoft Edge présente l’interface et son code.

Règles de validation

Afin d'optimiser les performances et la fiabilité du système, des règles de validation (**required, minlength, maxlength** et **accept**) sont appliquées dès le niveau du formulaire utilisateur. Ces règles, alignées sur celles du **backend**, permettent de limiter les erreurs côté API, de réduire la consommation de ressources serveur et d’améliorer la fluidité de l’application.

*Extrait du fichier src/pages/**PostForm.vue***

```
<template>
  <form @submit.prevent="onSubmit">
    <label for="title"><h2>Titre de l'article :</h2></label>
    <input type="text" id="title" v-model="post.title" required minlength="2" maxlength="50"/>

    <label for="content"><h2>Contenu de l'article :</h2></label>
    <textarea rows="6" id="content" v-model="post.content" required minlength="2" maxlength="5000"/>

    <label for="author"><h2>Auteur :</h2></label>
    <input type="text" id="author" v-model="post.author" required minlength="2" maxlength="50"/>

    <fieldset>
      <label for="postPicture"><h2>Choisir une photo pour l'article :</h2></label>
      <input type="file" id="postPicture" name="postPicture" accept="image/jpeg, image/png"
@change="onFileChange">
    </fieldset>

    <button type="submit">Créer</button>
  </form>
</template>
```

Ensuite dans le script js on découpe en deux étapes la soumission du formulaire (création de l’article, puis upload de l’image) permet de séparer les responsabilités : les données textuelles sont envoyées en JSON, tandis que l’image est envoyée sous forme de multipart/form-data. Cette organisation respecte les bonnes pratiques REST et rend l’API plus modulaire et maintenable.

Extrait du script js de Postform.vue, création "textuel" de l'article

```
    // Étape 1 : Créer l'article (JSON)
    const articleData = {
      name: post.value.title,
      content: post.value.content,
      author: post.value.author,
      date_created: new Date().toISOString(),
      status: "DRAFT"
    };

    const res = await axios.post('http://localhost:9000/article', articleData,
{     headers: {
        'Content-Type': 'application/json'
      }
    });

    console.log('Article créé:', res.data);
    const articleId = res.data.id;
```

J'utilise cette première étape pour récupérer l'id de l'article nouvellement créé. Cet id permettra ensuite d'attribuer l'image au bon article grâce à la requête POST **/article/${articleId}/upload** .

Extrait du script js de Postform.vue, envoi de l'image par requête POST

```
    // Étape 2 : Envoyer l'image si elle existe
    if (postPicture.value && articleId) {
      try {
        const formData = new FormData();
        formData.append("image", postPicture.value);

        await axios.post(`http://localhost:9000/article/${articleId}/upload`, formData, {
          headers: {
            'Content-Type': 'multipart/form-data'
          }
        });
        console.log('Image uploadée avec succès');
        toast.success('Image uploadée avec succès')

      } catch (e) {
        console.error('Erreur:', e);
        toast.error(e.response?.data || "Erreur lors du téléchargement de l'image")
      }
    }
```

Interactions utilisateur

Grâce à l'import de la bibliothèque **vue-toastification** on utilise des ***popup*** pour informer l'utilisateur de la bonne, ou mauvaise, prise en compte des champs du formulaire.
On remarque que des logs ont été insérés pour le débogage de l'application.

Requêtes HTTP

Pour les requêtes **HTTP** nous utilisons la bibliothèque **axios()** plutôt que **fetch()** pour sa plus grande concision dans le code, sa gestion automatique des en-têtes et sa meilleure gestion des erreurs. Cela permet une lisibilité du code améliorée.

Design Responsive

Le composant **Header**, présent sur l'ensemble des pages de l'application, a été conçu de manière responsive.
Deux modes d'affichage sont prévus : un affichage "desktop" pour les écrans larges (supérieur à 901 pixels de largeur), et un affichage "mobile" pour les écrans plus petits (< ou = à 900 pixels de largeur).
À l'aide de la règle **@media** dans la section CSS du fichier **Header.vue**, une cassure de mise en page (ou breakpoint) est définie à 900 pixels de largeur. En dessous de ce seuil, les liens de navigation tels que "Créer un article" ou "Modifier un article" sont masqués et remplacés par un bouton "burger", afin d'optimiser l'affichage et l'expérience utilisateur sur mobile.

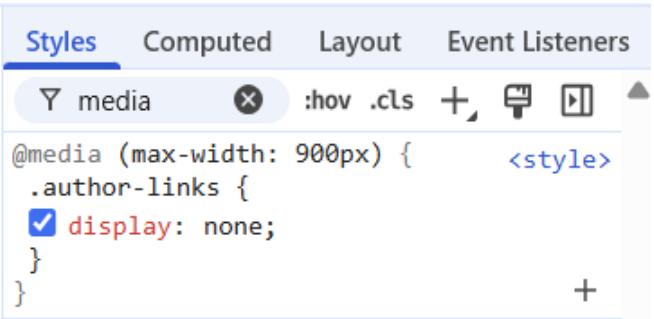
Extrait du CSS du fichier Header.vue, partie @media

```
/* Responsive: <= 900px */
@media (max-width: 900px) {
  .header nav {
    gap: 1rem;
  }

  .author-links {
    display: none; /* on cache les liens dans la barre */
  }

  .burger {
    display: block; /* on affiche le bouton burger */
  }

etc....
```

On peut tester le design responsive en utilisant des outils de développement d'un navigateur (par exemple : Chrome). Ce qui permet de réduire artificiellement les dimensions de l'écran d'affichage de l'application web et ainsi tester les ***breakpoints***. Des valeurs pré-enregistrées sont disponibles pour se mettre à la place d'un utilisateur d'un iphone SE, d'une tablette Ipad-mini…

Par exemple pour un mobile Iphone-SE qui fait 375px largeur, la règle **@media** est prise en compte et s'affiche donc dans l'onglet style de l'outil de développement de **Chrome**.

Onglet "Styles" des éléments bouton burger



Veille Technologique

Dans le cadre de la veille technologique, je me suis intéressé au document de référence **OWASP Top Ten** proposé par l'OWASP. Ce guide présente les dix principaux risques de sécurité des applications web, identifiés par la communauté et par l'analyse de vulnérabilités terrain.
Pour mon projet — une application front-end en Vue.js et un backend API en Spring Boot/Java — ce document permet de cadrer les aspects sécurité dès la conception et le développement : il invite à porter une attention particulière à des problématiques comme le contrôle d'accès, l'injection, les mauvaises configurations et l'exposition de composants obsolètes. En intégrant les bonnes pratiques OWASP dans l'architecture (par exemple validation des entrées, gestion des erreurs, sécurisation des endpoints, mise à jour des dépendances), on renforce la fiabilité et la robustesse de l'application. Cette démarche proactive s'inscrit pleinement dans une philosophie « shift-left », c'est-à-dire anticiper la sécurité dès les premiers stades du développement, et non uniquement en fin de projet ou après un audit.

Voici le top 10 des risques de sécurité des applications Web établit en 2021 vis a vis de mon projet :

A01-2021 – Broken Access Control
Le contrôle d'accès est centralisé au sein de la plateforme de formation PDF, qui gère l'authentification et l'autorisation des utilisateurs.
Mon application Newswik-Articles s'intègre à cette plateforme, qui applique la politique de contrôle d'accès.

A02-2021 – Cryptographic Failures
Les échanges entre le frontend Vue.js et le backend Spring Boot se font actuellement sur un réseau interne sécurisé, sans exposition extérieure. Aucune donnée sensible (mot de passe, token, données personnelles) n'est transmise ou stockée par l'application.
Toutefois, pour renforcer la sécurité conformément aux bonnes pratiques OWASP, une transition vers HTTPS pour tous les échanges (y compris internes) est possible.
Cette évolution passera par la configuration de Spring Security pour activer TLS côté backend, et l'envoi systématique des requêtes HTTPS depuis le frontend Vue.js.

A03-2021 – Injection
Mon application **Vue.js** utilise uniquement des liaisons textuelles sécurisées {{ }} , ce qui évite l'exécution de contenu potentiellement injecté (**XSS**).
Aucune génération de code **HTML** par **JavaScript** n'est effectuée dans le **DOM**.
Le backend utilise **Spring Boot** avec **MongoRepository** sans surcharge de méthodes ni requêtes personnalisées. Cela garantit l'utilisation de requêtes paramétrées générées automatiquement, évitant les **risques d'injection NoSQL.**
Les données en entrée sont encapsulées dans des **DTO** validés, empêchant l'injection de contenu malformé ou inattendu.

A04-2021 – Insecure Design

L'architecture de l'application repose sur une séparation claire des responsabilités entre le frontend Vue.js, le backend Spring Boot, et la plateforme de contrôle d'accès (PDF).
Le design respecte les principes de sécurité dès la conception :

- Utilisation de DTOs pour encapsuler et valider les données entrantes,
- Absence de génération dynamique de contenu HTML dans le DOM,
- Aucun stockage ou manipulation de secrets dans le frontend.

  Bien que les communications internes ne soient pas encore systématiquement chiffrées, une transition vers HTTPS avec TLS côté backend est possible pour renforcer la sécurité.

A05-2021 – Security Misconfiguration

L'API Spring Boot est configurée pour accepter uniquement les requêtes provenant des frontends autorisés, localisés sur les ports localhost:5173 et localhost:5174 (configuration CORS restreinte).
Seules les méthodes HTTP nécessaires (GET, POST, PATCH, PUT, DELETE) sont autorisées, les autres étant bloquées par défaut.

Des protections supplémentaires (comme les headers de sécurité HTTP, la désactivation des consoles de debug, et la gestion fine des logs) sont possibles pour renforcer la sécurité de la configuration.

A06-2021 – Vulnerable and Outdated Components

Les dépendances utilisées dans le frontend Vue.js et le backend Spring Boot sont maintenues à jour régulièrement.
Côté frontend, la commande npm audit est utilisée pour identifier les vulnérabilités connues, et les mises à jour sont effectuées dès que possible (npm audit fix). Les packages non utilisés sont supprimés du projet.
Côté backend, les dépendances Gradle sont surveillées et IntelliJ Ultimate a une surveillance des CVE (Common Vulnerabilities and Exposures).

Deux failles sont détectées :

1- **CVE-2025-11226** : Elle concerne l'utilisation de Spring avec Logback mais non utilisé dans mon projet. J'ai donc demandé d'ignorer l'alerte en la justifiant par sa non applicabilité dans mon projet.

2- **CVE-2025-48924** : Elle concerne une dépendance apache (commons-lang en version antérieur à 3.18.0)

J'ai vérifié les dépendances avec la commande
" ./gradlew dependencies " dans le terminal.
J'étais en **3.17.0** donc effectivement antérieur à 3.18.0

```
org.apache.commons:commons-lang3:3.17.0
```

J'ai donc mis à jour et vérifié dans mon build.gradle la version **3.18.0**

```
org.apache.commons:commons-lang3:3.17.0 -> 3.18.0
```

A07-2021 - Identification and authentication failures
Ce projet n'implémente pas de mécanismes d'authentification, de gestion de sessions ou de tokens.Le contrôle d'accès est géré par la plateforme globale en dehors du périmètre de cette application.
Par conséquent, cette vulnérabilité n'est pas applicable à notre projet.

A08-2021 – Software and Data Integrity Failures
Côté frontend, les dépendances npm sont installées exclusivement depuis le registre officiel, avec un contrôle d'intégrité automatique via le fichier package-lock.json. Les mises à jour sont régulièrement effectuées grâce aux outils **npm audit** et **npm audit fix**.
Côté backend, les dépendances Gradle sont gérées via le plugin io.spring.dependency-management, avec des vérifications régulières des versions et une utilisation exclusive de dépôts officiels (Maven Central).
Ces mesures garantissent la fiabilité, la provenance et la mise à jour sécurisée des composants utilisés, réduisant ainsi les risques liés à la chaîne d'approvisionnement logicielle.

A09-2021 – Security Logging and Monitoring Failures
Actuellement, cette application ne met pas en place de mécanismes de journalisation et de surveillance des événements de sécurité.
Cette fonctionnalité pourrait être déléguée à un système centralisé ou à la plateforme globale qui gère l'ensemble de la supervision et des alertes.

A10-2021 – Server-Side Request Forgery (SSRF)
Comme vu précédemment nous avons restreint les types de requêtes au niveau du controller de l'API ce qui limite les possibilités d'attaques.

L'utilisation de DTO permet de contrôler strictement la structure et le type des données reçues par l'API. Grâce aux annotations de validation (ex : @NotNull, @Pattern, @Size, etc.), seules les données conformes au schéma attendu sont acceptées.
Cela limite fortement la possibilité pour un attaquant de fournir des URLs malicieuses ou des paramètres imprévus qui pourraient déclencher des requêtes dangereuses côté serveur.

Ainsi, la validation côté backend via les DTO et la limitation des requêtes constituent 2 lignes, contribuant à la prévention des vulnérabilités SSRF.

Recherche à partir de site anglophone

Dans le cadre de mon utilisation de **Vue.js**, j'ai mis en place un système de navigation entre les différentes pages de l'application. Pour cela, j'ai utilisé **Vue Router**, l'outil de routage officiel du framework. Afin de bien comprendre son fonctionnement, j'ai consulté la documentation officielle disponible à l'adresse suivante : https://router.vuejs.org/guide/ . Cette ressource est considérée comme la référence la plus fiable, car elle est maintenue par l'équipe qui développe Vue.js.
D'une manière générale je privilégie l'utilisation de documentation officielle des langages ou framework utilisés. Si je ne trouve pas je recherche sur :

- stackoverflow.com
- developer.mozilla.org

Extrait du site Vue

Traduction du "Getting Started" du guide disponible à l'adresse https://router.vuejs.org/guide .

Commencer
Regarder un cours vidéo gratuit sur Vue Router
Vue Router est la solution officielle de routage côté client pour Vue.
Le routage côté client est utilisé par les applications en page unique (SPAs) pour lier l'URL du navigateur au contenu visionné par l'utilisateur. Au fur et à mesure que les utilisateurs naviguent dans l'application, l'URL est mise à jour en conséquence, mais la page n'a pas besoin d'être rechargée depuis le serveur.
Vue Router est construit sur le système des composants vue. Vous configurez les routes pour dire au routeur Vue quels composants afficher pour chaque chemin d'URL.
Ce guide supposera que vous êtes déjà familier avec Vue lui-même. Vous n'avez pas besoin d'être un expert de Vue, mais vous pourrez occasionnellement avoir besoin de vous référer à la documentation du cœur de Vue pour plus d'information concernant certaines fonctionnalités.
Un exemple
Pour présenter certaines des idées principales, nous allons étudier cet exemple
Exemple de l'aire de jeu Vue
Commençons en regardant à la racine du composant, "App.vue"
App.vue
…{ exemple de code }...
Ce modèle utilise deux composants fournis par Vue Router, RouterLink et RouterView.
Au lieu d'utiliser les balises classiques <a>, nous utilisons le composant personnalisé RouterLink pour créer des liens. Ceci permet à Vue Router de changer l'URL sans recharger la page, de gérer la génération d'URL, l'encodage, et diverses autres caractéristiques. Nous irons dans plus de détails concernant RouterLink plus loin dans les chapitres du guide.
Le composant RouterView informe Vue Router où faire l'affichage de la route composant actuel. C'est le composant qui correspond au chemin d'URL actuel. Il n'a pas besoin d'être dans App.vue, vous pouvez le mettre n'importe où pour l'adapter à votre mise en page, mais il doit être mis quelque part, autrement Vue Router ne fera le rendu de rien du tout.

L'exemple ci-dessous utilise aussi {{ $route.fullPath }}. Vous pouvez utiliser $route dans votre modèle de composant pour accéder à un objet qui représente la route actuelle.

Création de l'instance de routeur

L'instance de routeur est créé en appelant la fonction createRouter():

…{ exemple de code}...

L'option "routes" définit les routes en elles-mêmes, en mappant les chemins d'URL aux composants. Le composant spécifié par l'option "component" est celui qui sera rendu par le <Routerview> dans notre App.vue précédente. Ces composants de routage sont quelquefois appelés "views", bien qu'il ne s'agisse que de composants Vue normaux.

Il est intéressant de noter que si vous voulez utiliser des composants fonctionnels comme composants de routage, vous devez leur donner un "displayName" pour qu'ils puissent être différenciés des routages à chargement "lazy".

…{ exemple de code }…

Routes comportent de nombreuses autres options que nous verrons plus tard dans le guide, mais pour l'instant nous avons juste besoin d'un "path" ("chemin") et un "component" ("composant").

L'option "history" contrôle comment les routes sont mappées sur l'URL et vice versa. Pour l'exemple de l'aire de jeu nous avions utilisé "createMemoryHistory()", qui ignore complètement l'URL du navigateur et utilise sa propre URL interne à la place. Ça fonctionne bien pour l'Aire de Jeux, mais ça n'est pas ce que vous voudriez dans une application réelle. En règle générale vous souhaiteriez utiliser "createWebHistory()" à la place, ou peut-être "createWebHashHistory()". Nous couvrirons ce sujet de manière plus détaillée dans le guide des modes d'histoires ("History modes).

Enregistrer le plugin routeur

Une fois notre instance de routeur créée, nous avons besoin de l'enregistrer comme un plugin en appelant "use()" dans notre application :

…{ exemples de code }...

Ou de manière équivalente :

…{ exemples de code }...

Comme avec la plupart des plugins Vue, l'appel à "use()" doit être fait avant l'appel de "mount()".

Si vous êtes curieux de ce que ces plugins font, quelques unes de leurs responsabilités comprennent :

1- enregistrement global des composants RouterView" et RouterLink".

2- Ajout les propriétés globales $router et $route.

3- Activation des composables "useRouter()" and "useRoute()".

4- Déclenche le routeur pour lancer la route initiale.

Accéder au routeur et à la route en cours

Vous allez probablement vouloir accéder au routeur à partir d'un autre endroit dans votre application.

Si vous exportez l'instance du routeur depuis un module ES, vous pouvez importer l'instance de routeur directement où vous en avez besoin. Dans certains cas c'est la meilleure approche, mais nous avons d'autres options si nous sommes à l'intérieur d'un composant.

Dans le "template" du composant, l'instance de routeur est exposé ainsi "$router". C'est similaire à la propriété "$route" que nous avons vu plus tôt, mais notez le "r" supplémentaire à la fin.

Si nous utilisons les Options d'API, nous pouvons y accéder avec les deux mêmes propriétés comme "this.$router" et "this.$route" dans notre code javascript. Le composant "HomeView.vue" dans l'exemple de l'aire de jeu accède au routeur de cette manière :
…{ exemple de code}...
Cette méthode appelle "push()", qui est utilisé pour la navigation programmatique. Nous en apprendrons plus à ce sujet plus tard.

Bilan et perspectives du projet

Le projet NewsWik Articles a pu être intégré dans la plateforme PDF à la fin de ma période de stage. Le logiciel PDF devrait faire surface dans sa première version avant la fin de cette année.

Le projet pourrait être amélioré en y ajoutant plus de fonctions à l'UI, afin d'améliorer l'UX :

- Ajouter un thème sombre et clair
- Choisir les préférences utilisateur en utilisant des tags sur les articles.
- Choisir d'alimenter la base d'articles avec une API externe.
- etc …

Les possibilités d'améliorations sont nombreuses, la première brique de l'application est maintenant posée. La modularité de MongoDB pourrait permettre d'ajouter un certain nombre de champs supplémentaires.

En tout cas les 3 applications réalisées pendant le stage remplissent les spécifications demandées.

Conclusion

Avant de commencer mon stage, je n'étais pas sûr de moi ni de mes capacités à suivre le rythme. Mais, j'ai pu constater que, même avec un niveau encore débutant, j'ai réussi à développer une partie fonctionnelle d'un projet plus large. Cela a été très motivant et valorisant.

Le projet sur lequel j'ai travaillé m'a vraiment intéressé, et j'ai beaucoup appris, aussi bien sur le plan technique qu'organisationnel. Ce stage m'a permis d'enrichir mes connaissances et de les mettre en pratique dans un cadre professionnel concret.
Même si j'avais déjà une expérience du monde de l'entreprise, ce stage m'a permis de découvrir un nouvel environnement, dans le domaine du développement. J'ai été très bien accueilli, et l'ambiance bienveillante m'a permis de m'intégrer rapidement et de travailler en confiance.

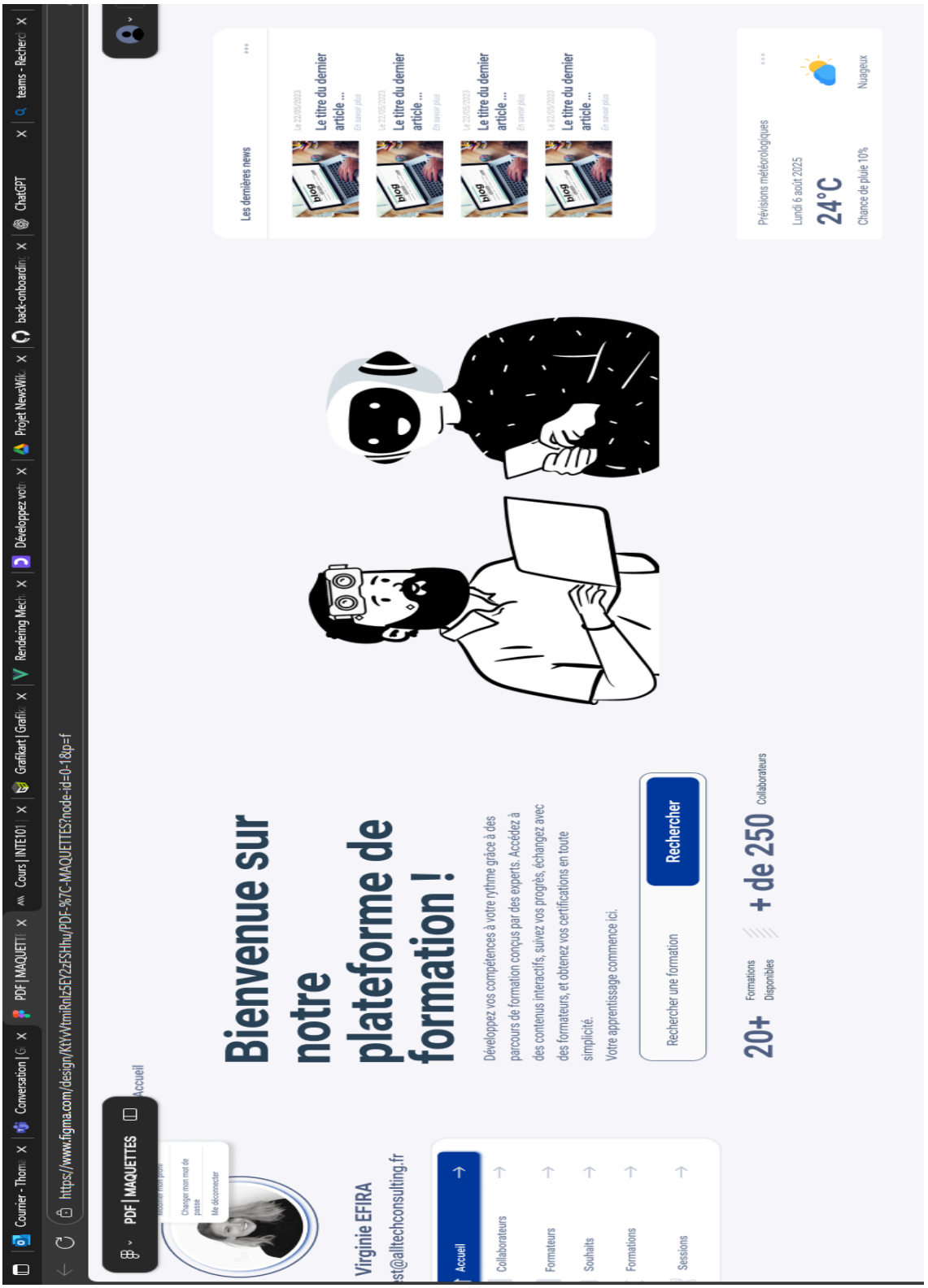
La formation DWWM de l'ENI m'a apporté des bases solides, qui m'ont permis d'appréhender le stage avec sérieux.
Pendant la formation, le rythme était très soutenu et parfois complexe à suivre.
J'ai remarqué que l'école était bien reconnue dans le milieu professionnel.

Enfin, ce stage m'a confirmé que rien ne vaut la pratique pour progresser. C'est pourquoi je souhaite poursuivre ma reconversion par une formation en alternance CDA (Concepteur Développeur d'Applications), tout en continuant à développer et capitaliser mes compétences à travers des projets personnels.

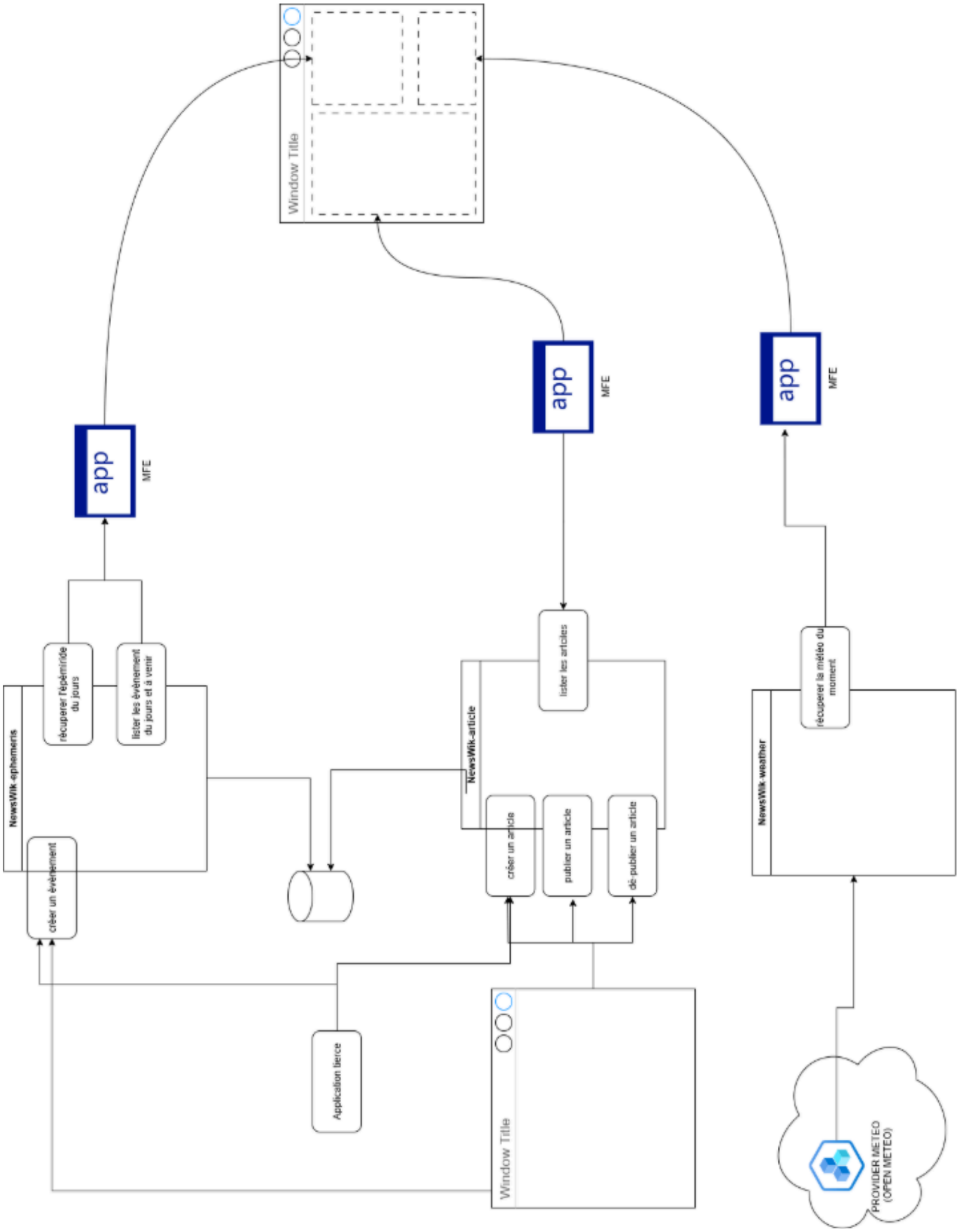
Annexes

Projet NewsWik - Articles

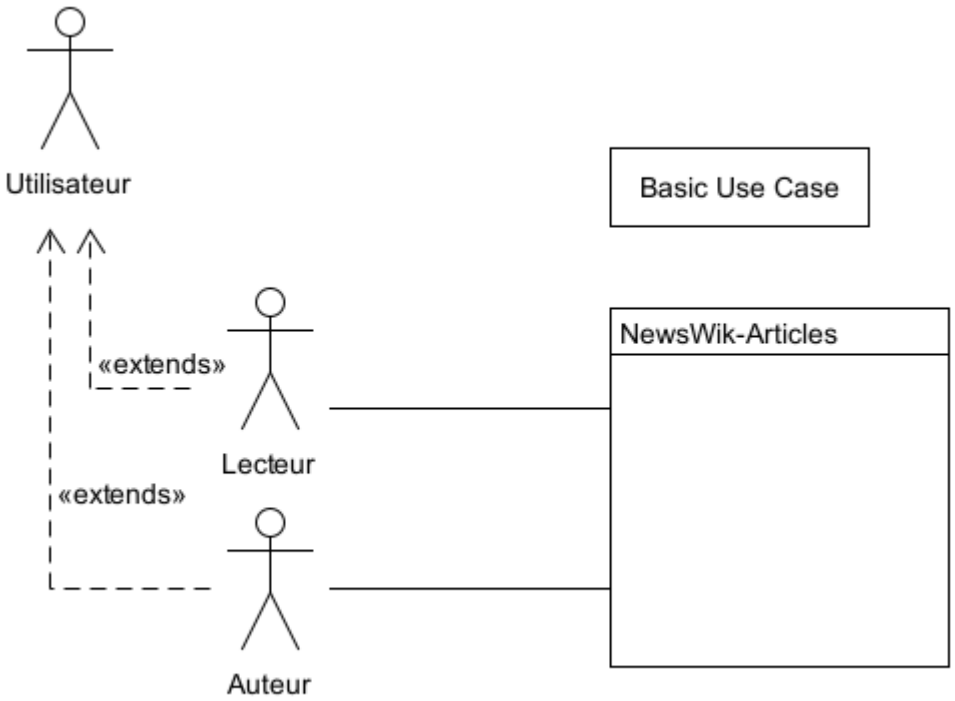
Annexe 1 - Espace de travail collaborateur



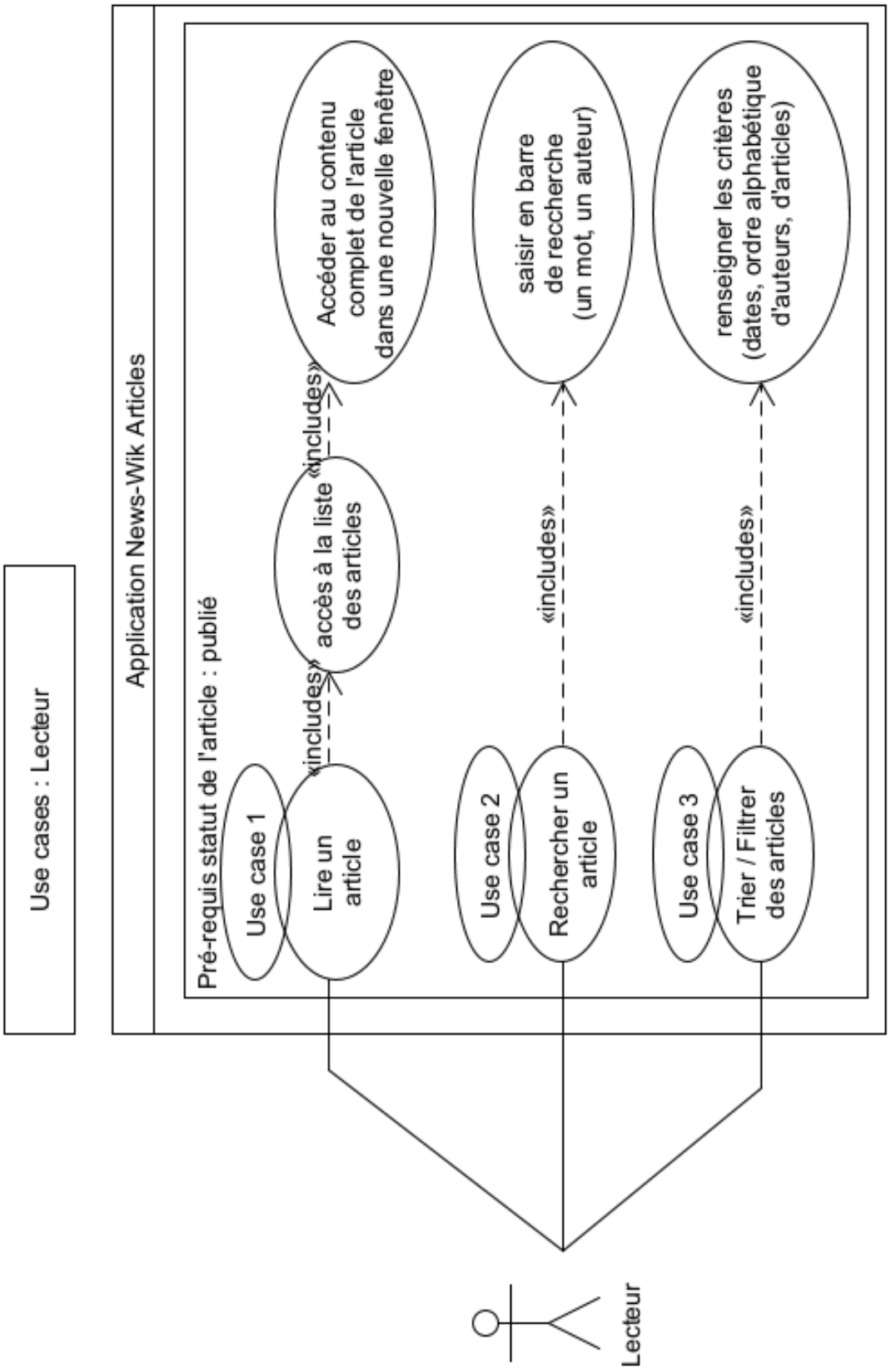
Annexe 2 - Synoptique du projet NewsWik



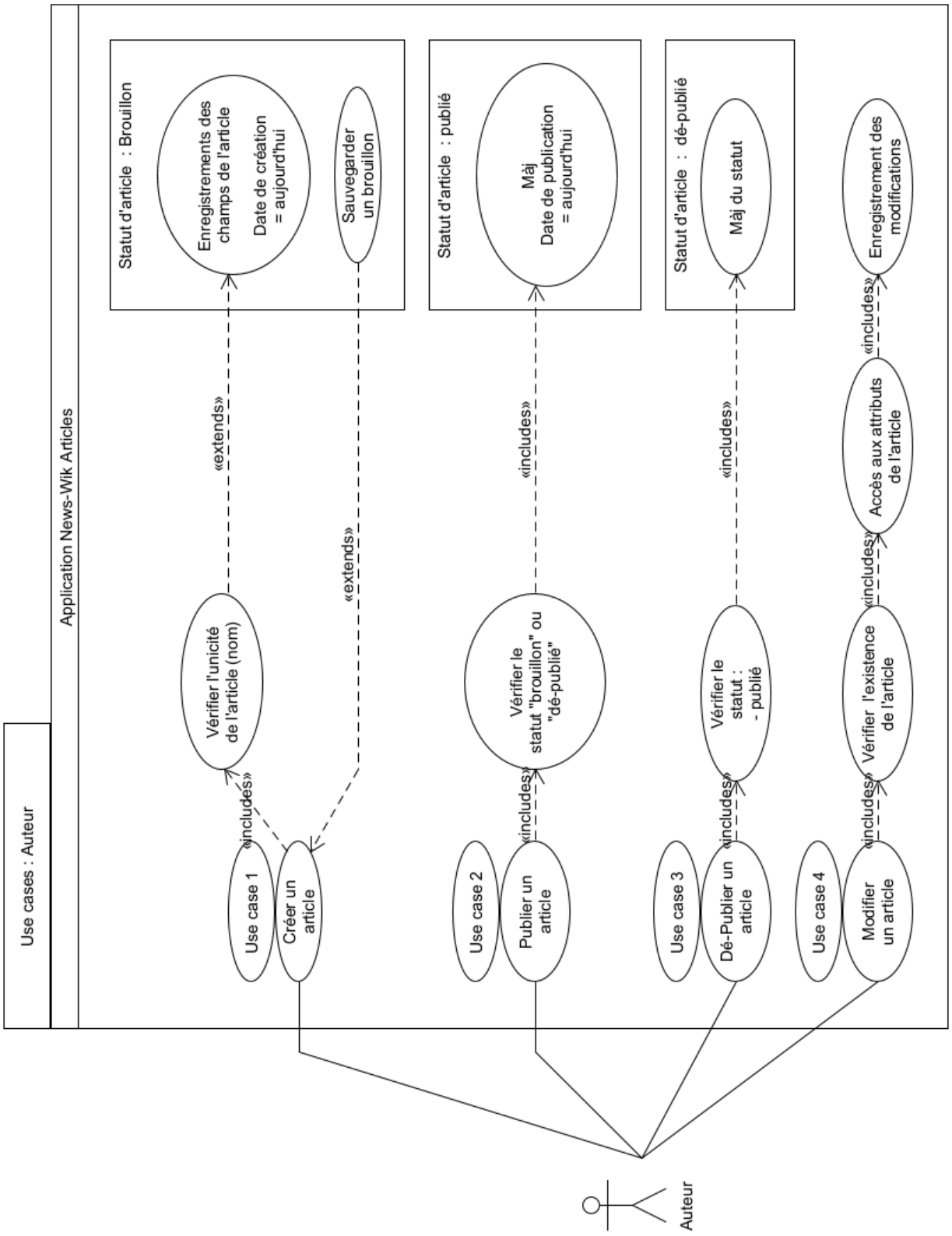
Annexe 3 - Diagramme UML - Cas d'utilisation Utilisateur



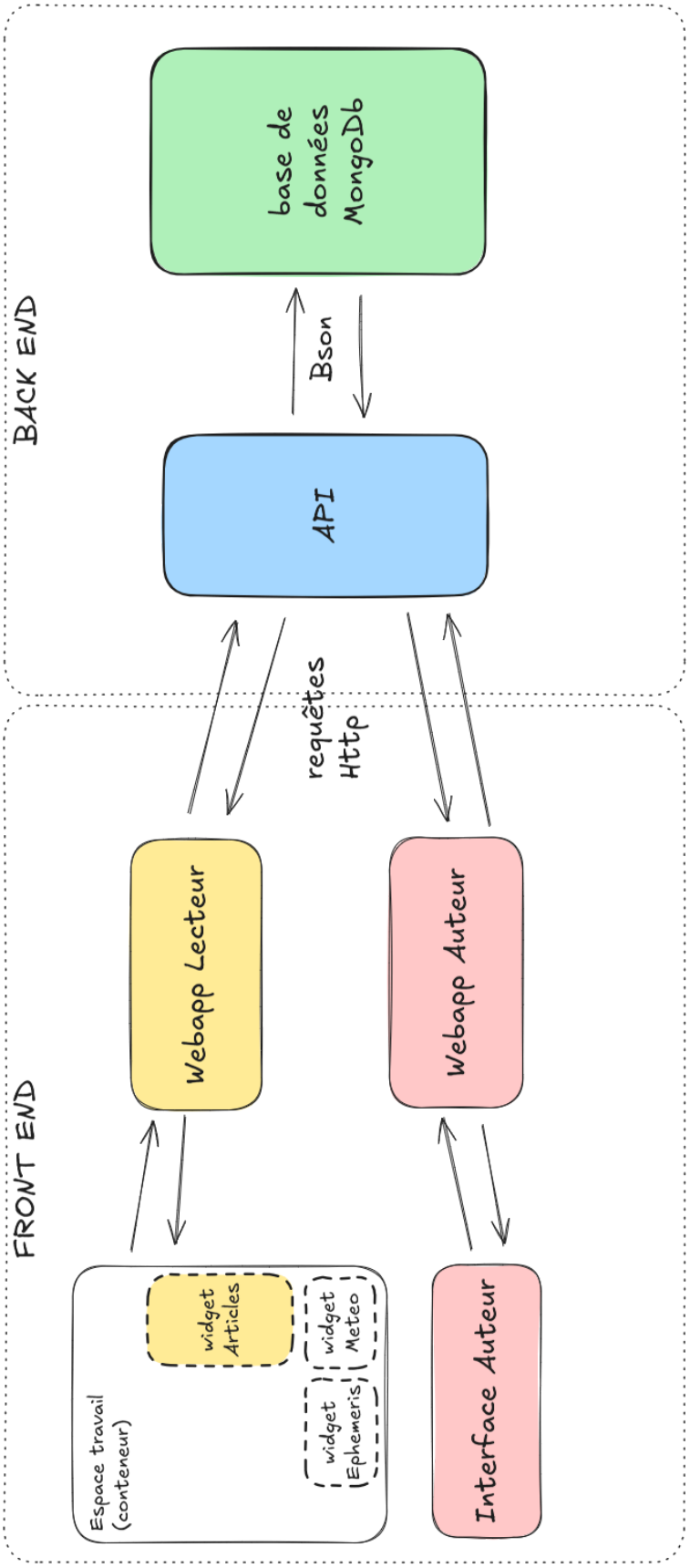
Annexe 4 - Diagramme UML - Cas d'utilisation Lecteur



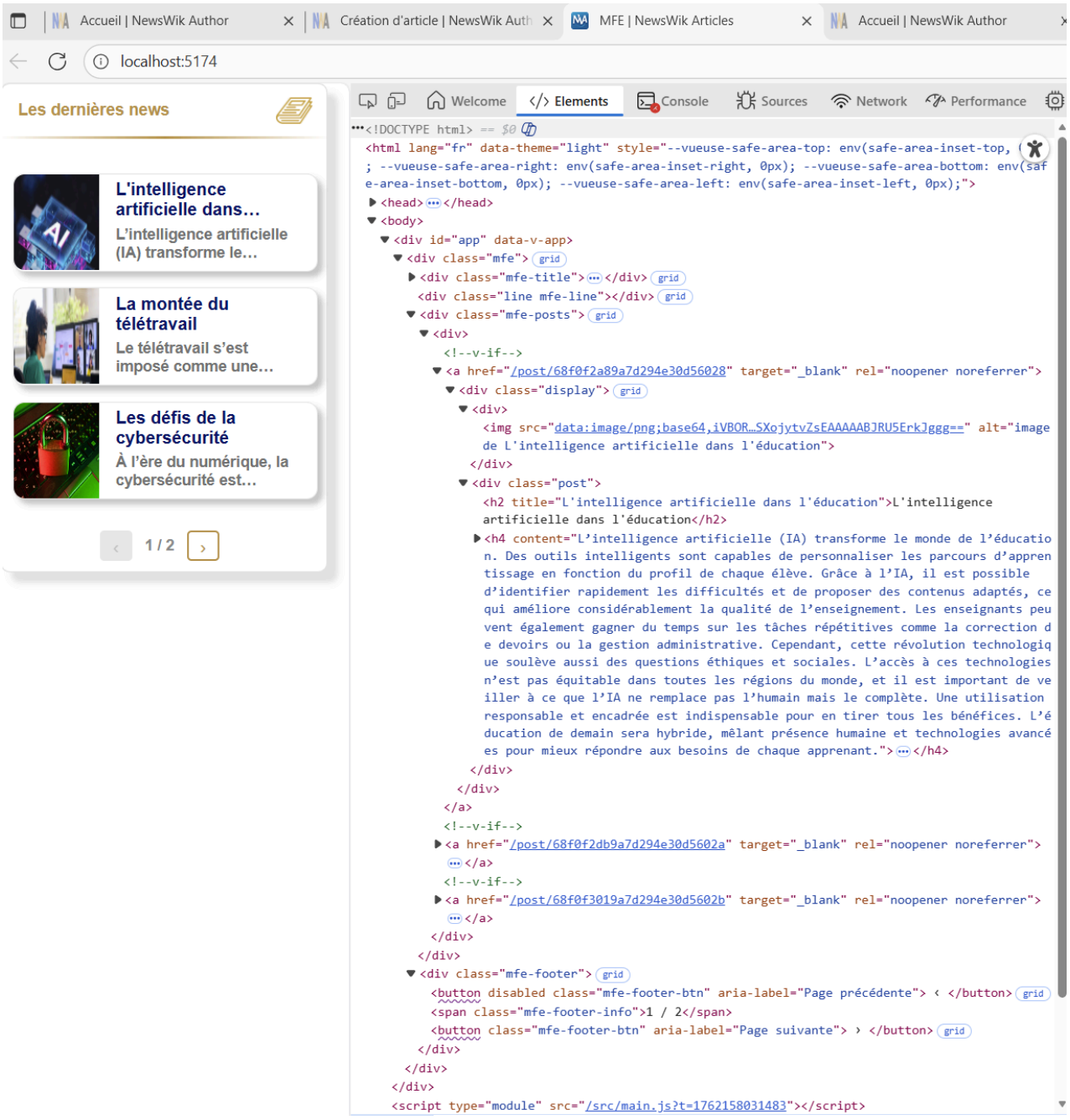
Annexe 5 - Diagramme UML - Cas d'utilisation Auteur



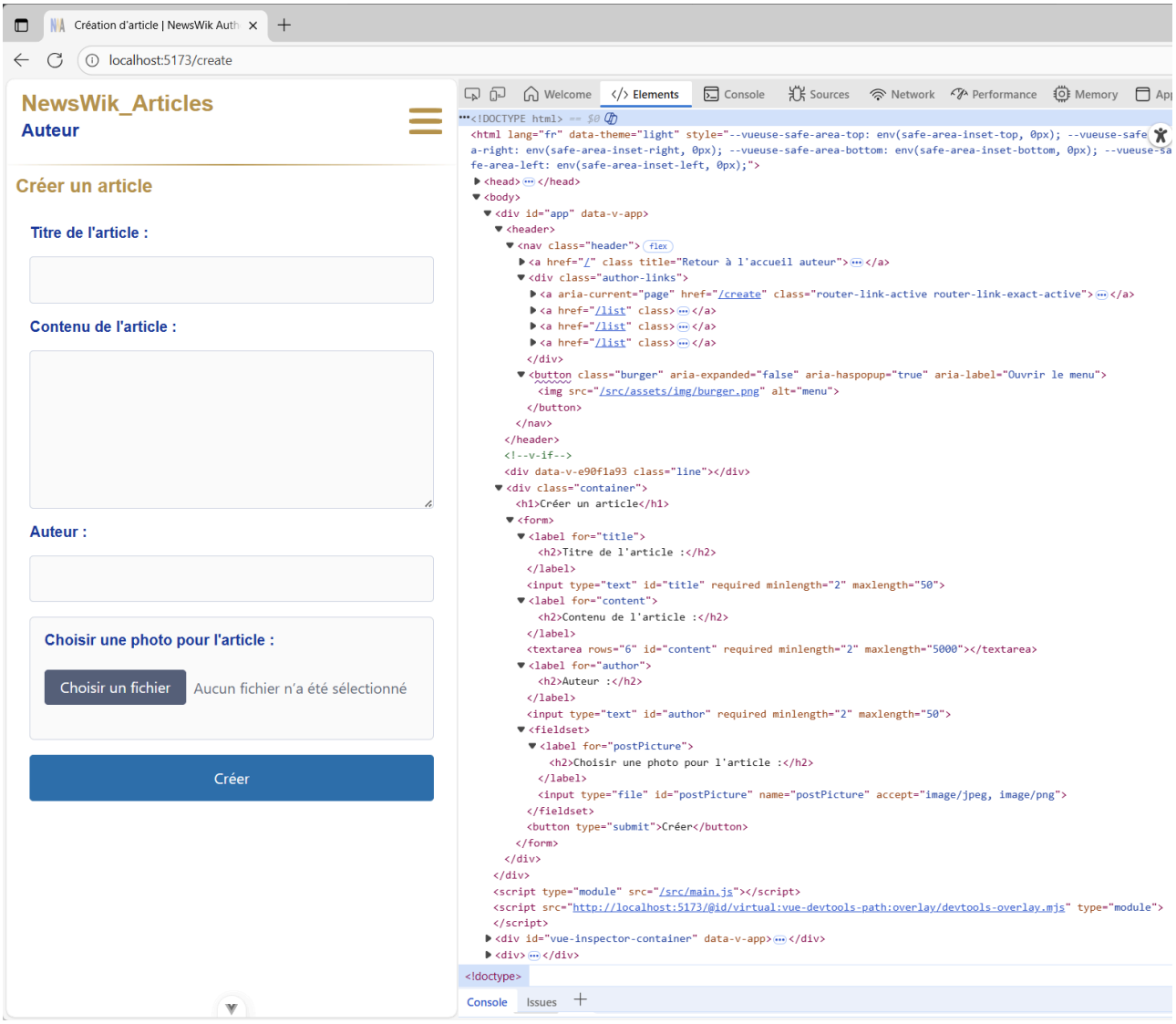
Annexe 6 - Découpage Back et Front



Annexe 7 - MFE-Preview - Interface utilisateur(lecteur) et code



Annexe 8 - page Create - interface (Auteur) et code



Annexe 9 - ArticleController.java

```
@CrossOrigin(origins = {"http://localhost:5173", "http://localhost:5174"},
        methods = {
                RequestMethod.GET,
                RequestMethod.POST,
                RequestMethod.PATCH,
                RequestMethod.PUT,
                RequestMethod.DELETE},
        allowedHeaders = "*",
        allowCredentials = "true")
@RestController
public class ArticleController {

    private final ArticleService articleService;

    // Injection par constructeur
    public ArticleController(ArticleService articleService) {
        this.articleService = articleService;
    }


/** CREATE - Add a new article **/
    @PostMapping("/article")
    public ResponseEntity<ArticleDto> createArticle(@Valid @RequestBody ArticleDto articleDto,
BindingResult result) {
        if (result.hasErrors()) {
            return ResponseEntity.badRequest().build();
        }
        ArticleDto savedArticle = articleService.saveArticle(articleDto);
        return ResponseEntity.status(HttpStatus.CREATED).body(savedArticle);
    }

/** READ - GET one article **/
    @GetMapping("/article/{id}")
    public ResponseEntity<ArticleDto> getArticle(@PathVariable("id") final String id) {
        return articleService.getArticle(id)
                        .map(ResponseEntity::ok)
                        .orElseGet(()-> ResponseEntity.notFound().build());
    }

/** READ - GET all Articles **/
    @GetMapping("/articles")
    public Iterable<ArticleDto> getArticles() {
        return articleService.getArticles();
    }
...
```

```
...

/** UPDATE - Update an existing article **/
    @PutMapping("/article/{id}")
    public ResponseEntity<ArticleDto> updateArticle(
            @PathVariable("id") final String id,
            @Valid @RequestBody ArticleDto articleDto,
            BindingResult result) {
        if (result.hasErrors()) {
            return ResponseEntity.badRequest().build();
        }
        Optional<ArticleDto> updatedArticle = articleService.updateArticle(id, articleDto);

        return updatedArticle
                .map(ResponseEntity::ok)
                .orElseGet(() -> ResponseEntity.notFound().build());
    }

/** DELETE - Delete an article **/
    @DeleteMapping("/article/{id}")
    public void deleteArticle(@PathVariable("id") final String id) {
        articleService.deleteArticle(id);
    }

/** PATCH - modify the status into published or unpublished **/
    @PatchMapping("/article/{id}/status")
    public ResponseEntity<ArticleDto> toggleArticleStatus(@PathVariable("id") final String id)
{
        Optional<ArticleDto> toggledArticle = articleService.toggleArticleStatus(id);

        return toggledArticle
                .map(ResponseEntity::ok)
                .orElseGet(() -> ResponseEntity.notFound().build());
    }

/** Image Endpoints */
/** READ - reading a picture **/
    @GetMapping("/article/{id}/image")
    public ResponseEntity<?> getImage(@PathVariable("id") String id) {
        if (id.trim().isEmpty()) {
            return ResponseEntity.badRequest().body("L'identifiant de l'article est requis");
        }
        Optional<ImageDto> optionalImageDto = articleService.getImageDto(id);

        return optionalImageDto
                .map(ResponseEntity::ok)
                .orElseGet(() -> ResponseEntity.notFound().build());
    }
...
```

```
...

/** UPLOAD - uploading picture **/
    @PostMapping(value = "/article/{id}/upload", consumes =
MediaType.MULTIPART_FORM_DATA_VALUE)
    public ResponseEntity<String> uploadImage(
            @PathVariable("id") final String id,
            @RequestParam("image") MultipartFile image) {

        if (id.trim().isEmpty()) {
            return ResponseEntity.badRequest().body("L'identifiant de l'article est requis");
        }

        try {
            Optional<ImageDto> uploadedArticle = articleService.uploadImage(id, image);

            if (uploadedArticle.isEmpty()) {
                return ResponseEntity.notFound().build();
            }

            return ResponseEntity.ok("Image ajoutée avec succès");

        } catch (IllegalArgumentException e) {
            return ResponseEntity.badRequest().body(e.getMessage());
        } catch (IOException e) {
            return ResponseEntity.internalServerError()
                    .body("Erreur lors du traitement de l'image: " + e.getMessage());
        }
    }

/** Update picture **/
    @PutMapping(value = "/article/{id}/image-update", consumes =
MediaType.MULTIPART_FORM_DATA_VALUE)
    public ResponseEntity<String> updateImage(
            @PathVariable("id") final String id,
            @RequestParam("image") MultipartFile image) {
        if (id.trim().isEmpty()) {
            return ResponseEntity.badRequest().body("L'identifiant de l'article est requis");
        }
        if (image == null || image.isEmpty()) {
            return ResponseEntity.badRequest().body("Le fichier image est requis");
        }

        try {
            Optional<ImageDto> updatedArticle = articleService.uploadImage(id, image);

            if (updatedArticle.isEmpty()) {
                return ResponseEntity.notFound().build();
            }

            return ResponseEntity.ok("Image mise à jour avec succès");

        } catch (IllegalArgumentException e) {
            return ResponseEntity.badRequest().body(e.getMessage());
        } catch (IOException e) {
            return ResponseEntity.internalServerError()
                    .body("Erreur lors du traitement de l'image: " + e.getMessage());
        }
    }
}
```