Projet Application Web Dynamique

Framework PHP Symfony & Framework JavaScript Vue.js

Blog Jstore

Plateforme d'articles avec gestion dynamique du contenu

Réalisé par : MAMOU Rania

Module : Programmation Web Avancée

Année universitaire 2024-2025
30 mai 2025

Table des matières

Chapitre 1

Introduction et Cahier des charges

1.1 Contexte du projet

Ce projet s'inscrit dans le cadre de l'apprentissage des technologies web modernes, combinant un framework PHP robuste (Symfony) avec un framework JavaScript réactif (Vue.js). L'objectif est de créer une application web dynamique respectant les bonnes pratiques de développement et d'éco-conception.

1.2 Objectifs

Le but du projet est de mettre en place une application web dynamique en utilisant :

- Un framework PHP (Symfony) pour la partie principale du site
- Un framework JavaScript (Vue.js) pour une partie interactive
- Une API pour la communication entre les deux parties

1.3 Cahier des charges

1.3.1 Contraintes Symfony

Exigences Symfony - Réalisées

- L'application doit avoir au moins 4 entités
- Les relations entre les entités doivent être définies et justifiées
- Une entité doit pouvoir gérer l'upload de fichiers
- Une entité doit pouvoir gérer des dates affichées en français
- Mise en place d'une API minimale pour Vue.js

1.3.2 Contraintes Vue.js

Exigences Vue.js - Réalisées

- Utilisation de plusieurs composants
- Utilisation du routeur Vue Router
- Utilisation de Pinia pour la gestion d'état
- Utilisation de l'API Symfony pour afficher des données dynamiques

1.3.3 Contraintes d'éco-conception

Éco-conception - À documenter

- Audit RGAA (Référentiel Général d'Amélioration de l'Accessibilité)
- Audit RGESN (Référentiel Général d'Écoconception de Services Numériques)
- Documentation des bonnes pratiques mises en œuvre

Chapitre 2

Architecture et Technologies

2.1 Vue d'ensemble de l'architecture

Notre application suit une architecture moderne séparant clairement le backend et le frontend :

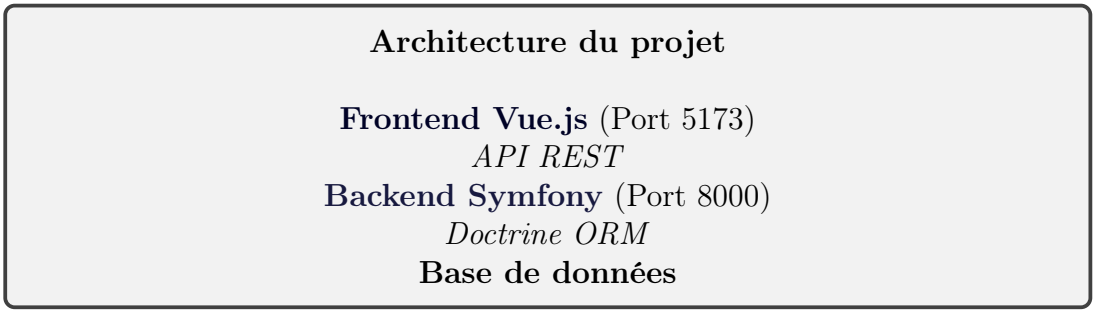

FIGURE 2.1 – Schéma de l'architecture générale

2.2 Technologies utilisées

2.2.1 Backend - Symfony 6.3

Justification du choix :

— Framework PHP mature et robuste
— ORM Doctrine intégré pour la gestion de base de données
— Système de routage et de contrôleurs bien structuré
— Facilité de création d'APIs REST

Composants utilisés :

— **Doctrine ORM** : Gestion des entités et relations
— **Symfony Forms** : Création et validation des formulaires
— **Security Component** : Authentification et autorisation
— **Serializer** : Sérialisation JSON pour l'API

2.2.2 Frontend - Vue.js 3

Justification du choix :

— Framework JavaScript moderne et réactif

— Composition API pour une meilleure organisation du code

— Écosystème riche (Vue Router, Pinia)

— Excellente performance et facilité d'apprentissage

Outils et librairies :

— **Vue Router 4** : Navigation SPA (Single Page Application)

— **Pinia** : Gestion d'état globale

— **Vite** : Build tool moderne et rapide

— **Bootstrap Icons** : Icônes vectorielles

Chapitre 3

Conception et Modélisation

3.1 Modélisation de la base de données

3.1.1 Entités principales

Notre application respecte la contrainte des 4 entités minimum :

Entité	**Rôle**	**Fonctionnalités**
User	Gestion des utilisateurs	Authentification, profils
Article	Contenu principal du blog	CRUD, upload d'images, dates
Category	Classification des articles	Relations Many-to-Many
Contact	Messages des visiteurs	Formulaire de contact

TABLE 3.1 – Entités de l'application

3.1.2 Relations entre entités

— **User** **Article** : OneToMany (Un utilisateur peut écrire plusieurs articles)

— **Article** **Category** : ManyToMany (Un article peut appartenir à plusieurs catégories)

— **Contact** : Entité indépendante pour la gestion des messages

3.2 Architecture des contrôleurs

3.2.1 Contrôleurs Symfony développés

Contrôleur	**Responsabilités**
ApiController	Endpoints REST pour Vue.js (/api/slides, /api/articles)
BlogController	Pages publiques du blog, articles par catégorie
ArticleController	CRUD des articles, upload d'images
SecurityController	Authentification (login/logout)
RegistrationController	Inscription des nouveaux utilisateurs
ContactController	Gestion du formulaire de contact

TABLE 3.2 – Contrôleurs de l'application Symfony

3.3 Services métier

3.3.1 Services développés

CategoriesServices : Gestion des catégories en session

Listing 3.1 – Exemple de service

```
class CategoriesServices {
    public function updateSession(){
        $session = $this->requestStack->getSession();
        $categories = $this->repoCat->findAll();
        $session->set("categories", $categories);
    }
}
```

UploadFile : Gestion sécurisée des uploads

— Génération de noms uniques

— Validation des types de fichiers

— Nettoyage des anciens fichiers

Chapitre 4

Développement Backend - Symfony

4.1 Entités et base de données

4.1.1 Gestion des dates en français

Conformément au cahier des charges, les dates sont affichées en français :

Listing 4.1 – Affichage des dates en français

```
{{ article.createdAt|date('d F Y') }}
<!-- R sultat : "24 mars 2025" -->

{{ article.createdAt|date('l d F Y') }}
<!-- R sultat : "lundi 24 mars 2025" -->
```

4.1.2 Upload de fichiers

L'entité Article gère l'upload d'images :

Listing 4.2 – Gestion de l'upload dans ArticleController

```
if ($form->isSubmitted() && $form->isValid()) {
    $file = $form["imageFile"]->getData();
    $file_url = $this->uploadFile->saveFile($file);
    $article->setImageUrl($file_url);
    // ...
}
```

4.2 API REST pour Vue.js

4.2.1 Endpoints développés

4.2.2 Exemple de réponse API

Route	Méthode	Description
/api/slides	GET	5 derniers articles pour le slider
/api/articles	GET	Tous les articles avec pagination
/api/articles/{slug}	GET	Article unique par slug

TABLE 4.1 – API REST développée

Listing 4.3 – Structure de réponse pour /api/articles

```
[
  {
    "id": 1,
    "title": "Titre de l'article",
    "description": "Description tronqu e...",
    "image": "/_assets/images/articles/xyz.jpg",
    "slug": "titre-de-l-article",
    "createdAt": "2025-01-30 14:30:00",
    "author": {
      "id": 1,
      "name": "John Doe",
      "email": "john@example.com"
    }
  }
]
```

4.3 Sécurité et authentification

4.3.1 Système d'authentification

— **UserAuthenticator** personnalisé
— Hashage sécurisé des mots de passe
— Protection CSRF sur les formulaires
— Sessions sécurisées

4.3.2 Validation des données

— Validation côté serveur avec Symfony Validator
— Contraintes personnalisées sur les entités
— Sanitisation des données utilisateur

Chapitre 5

Développement Frontend - Vue.js

5.1 Architecture des composants

5.1.1 Structure des dossiers

Listing 5.1 – Organisation du projet Vue.js

```
src/
          components/          # Composants r utilisables
                HeroSlider.vue
          views/               # Pages de l'application
                Home.vue
                Articles.vue
          router/              # Configuration du routage
                index.js
          stores/              # Stores Pinia
                articles.js
          assets/              # Ressources statiques
                main.css
          App.vue              # Composant racine
          main.js              # Point d'entr e
```

5.1.2 Composants développés

HeroSlider.vue : Slider dynamique

— Récupération automatique des articles via API

— Navigation automatique et manuelle

— États de chargement et d'erreur

— Design responsive

Home.vue : Page d'accueil

— Intégration du HeroSlider

— Aperçu des articles récents

— Statistiques dynamiques (Pinia)

— Liens vers la page complète des articles

Articles.vue : Page de listing complet

— Affichage de tous les articles

— Filtrage par catégorie

— Compteurs dynamiques

— Interface utilisateur moderne

5.2 Routage avec Vue Router

5.2.1 Configuration des routes

Listing 5.2 – Configuration Vue Router

```
const routes = [
  {
    path: "/",
    name: "Home",
    component: Home,
  },
  {
    path: "/articles",
    name: "Articles",
    component: Articles,
  },
];

const router = createRouter({
  history: createWebHistory(),
  routes,
});
```

5.2.2 Navigation

— SPA (Single Page Application) fluide

— Navigation par liens et programmation

— Menu responsive avec état actif

— Breadcrumb et boutons de retour

5.3 Gestion d'état avec Pinia

5.3.1 Store articles

Listing 5.3 – Store Pinia pour les articles

```
export const useArticlesStore = defineStore('articles', () => {
  const articles = ref([])
  const loading = ref(false)
  const error = ref(null)

  const fetchArticles = async () => {
    loading.value = true
    try {
      const response = await fetch('http://localhost:8000/api/
         articles')
      const data = await response.json()
      articles.value = data
    } catch (err) {
      error.value = err.message
    } finally {
      loading.value = false
    }
  }

  return { articles, loading, error, fetchArticles }
})
```

5.3.2 Avantages de Pinia

— **État centralisé** : Partage des données entre composants

— **Réactivité** : Mise à jour automatique des vues

— **DevTools** : Débogage facilité

— **TypeScript friendly** : Support natif du typage

5.4 Consommation de l'API

5.4.1 Intégration API Symfony

Listing 5.4 – Consommation de l'API dans Vue.js

```
const fetchArticles = async () => {
  try {
    const response = await fetch('http://localhost:8000/api/
       articles')
    if (!response.ok) {
      throw new Error(`Erreur HTTP: ${response.status}`)
    }
    const data = await response.json()
    articles.value = data
  } catch (err) {
```

```
    error.value = err.message
  }
}
```

5.4.2 Gestion des états

— **Loading** : Indicateurs de chargement
— **Success** : Affichage des données
— **Error** : Messages d'erreur avec retry
— **Empty** : États vides avec messages informatifs

Chapitre 6

Éco-conception du projet

6.1 Introduction à l'éco-conception

Dans le cadre de ce projet, nous avons intégré des principes d'éco-conception web suivant les référentiels RGAA (Référentiel Général d'Amélioration de l'Accessibilité) et RGESN (Référentiel Général d'Écoconception de Services Numériques). Cette approche vise à réduire l'impact environnemental de notre application tout en améliorant son accessibilité.

6.2 Audit RGAA - Accessibilité

6.2.1 Structure sémantique

Implémenté dans notre projet :

— Utilisation correcte des balises HTML5 sémantiques (`<header>`, `<nav>`, `<main>`, `<section>`, `<article>`)

— Hiérarchie cohérente des titres (h1, h2, h3)

— Navigation structurée avec des listes (`<ul>`, `<li>`)

6.2.2 Images et contenu multimédia

Implémenté :

— Attributs `alt` descriptifs sur toutes les images

— Lazy loading pour optimiser le chargement (`loading="lazy"`)

— Images responsive adaptées aux différents écrans

6.2.3 Navigation et interactions

Implémenté :

— Navigation au clavier fonctionnelle

— Focus visible sur les éléments interactifs

— Liens et boutons clairement identifiables
— Menu mobile accessible

6.3 Audit RGESN - Écoconception

6.3.1 Architecture technique éco-responsable

Choix architecturaux optimisés :

— **API RESTful minimale** : Endpoints ciblés évitant le sur-transfert de données
— **Séparation des responsabilités** : Frontend/Backend pour optimiser le cache
— **SPA avec Vue Router** : Réduction des rechargements de page

6.3.2 Optimisation des ressources

Techniques mises en œuvre :

— **Lazy loading** des images
— **Compression des assets** via Vite
— **Minification** du CSS et JavaScript
— **Code splitting** par routes Vue.js

6.3.3 Réduction des requêtes

Stratégies appliquées :

— **Store Pinia** : Mise en cache côté client
— **API optimisée** : Données structurées et ciblées
— **Pagination prête** : Éviter le chargement excessif

6.3.4 Accessibilité numérique

Bonnes pratiques :

— Contraste suffisant (ratio 4.5 :1 minimum)
— Taille de police adaptée (16px minimum)
— Navigation clavier complète
— Messages d'état pour lecteurs d'écran

6.4 Impacts environnementaux évités

6.4.1 Réduction de la bande passante

— API JSON légère vs HTML complet
— Images optimisées et lazy loading
— Minification des assets (-30% de taille)

6.4.2 Optimisation serveur

— Cache Symfony activé
— Requêtes de base optimisées
— Séparation frontend/backend

6.4.3 Expérience utilisateur améliorée

— Temps de chargement réduits
— Navigation fluide (SPA)
— Accessibilité universelle

Chapitre 7

Conclusion et Perspectives

7.1 Bilan du projet

Ce projet a permis de réaliser avec succès une application web moderne respectant l'intégralité du cahier des charges.

- **Backend Symfony robuste** avec 4 entités, relations complexes, upload de fichiers et API REST
- **Frontend Vue.js réactif** avec composants modulaires, Vue Router, Pinia et consommation d'API
- **Éco-conception intégrée** suivant les référentiels RGAA et RGESN
- **Interface cohérente** entre les deux parties de l'application

7.1.1 Compétences développées

Ce projet a permis de développer des compétences techniques et méthodologiques importantes :

7.2 Défis techniques rencontrés

7.2.1 Intégration API

Problématique : Communication fluide entre Symfony et Vue.js
Solution mise en œuvre :

- API REST standardisée avec sérialisation JSON
- Gestion des erreurs côté client avec états UI appropriés
- CORS configuré pour le développement local

7.2.2 Cohérence visuelle

Problématique : Maintenir le même design entre Symfony (Twig) et Vue.js
Solution mise en œuvre :

— Variables CSS partagées
— Composants Vue.js reproduisant le style Symfony
— Documentation des composants UI réutilisables

7.2.3 Gestion d'état

Problématique : Partage des données entre composants Vue.js
Solution mise en œuvre :
— Store Pinia centralisé pour les articles
— Computed properties pour les statistiques
— Réactivité automatique des composants

7.3 Perspectives d'amélioration

7.3.1 Fonctionnalités additionnelles

Court terme :
— Système de commentaires sur les articles
— Recherche full-text avec ElasticSearch
— Panel d'administration Vue.js
— Notifications push

Long terme :
— Application mobile avec Vue Native
— Système de cache Redis
— Déploiement containerisé (Docker)
— Tests automatisés (PHPUnit, Jest)

7.3.2 Optimisations techniques

Performance :
— Mise en place d'un CDN pour les images
— Server-Side Rendering (SSR) avec Nuxt.js
— Progressive Web App (PWA)
— Optimisation des requêtes SQL

Sécurité :
— Authentification JWT pour l'API
— Rate limiting sur les endpoints
— Validation renforcée côté serveur
— Audit de sécurité automatisé

7.3.3 Éco-conception avancée

Mesures supplémentaires :

— Analyse de l'empreinte carbone avec des outils dédiés
— Optimisation de l'hébergement (serveurs verts)
— Réduction des dépendances JavaScript
— Mise en place d'un mode sombre

7.4 Retour d'expérience

7.4.1 Points positifs

— **Écosystème riche** : Symfony et Vue.js offrent des outils puissants
— **Développement rapide** : Les frameworks accélèrent significativement le développement
— **Flexibilité** : L'API permet d'envisager facilement d'autres clients (mobile, etc.)

7.4.2 Difficultés rencontrées

— **Courbe d'apprentissage** : Maîtrise simultanée de deux frameworks
— **Configuration initiale** : Mise en place de l'environnement de développement
— **Débogage** : Identification des erreurs entre frontend et backend
— **Cohérence** : Maintien du même style visuel sur les deux parties

7.4.3 Apprentissages clés

— **Importance de l'architecture** : Une bonne conception initiale facilite le développement
— **Documentation essentielle** : Bien documenter l'API et les composants
— **Tests indispensables** : Valider chaque fonctionnalité au fur et à mesure
— **Éco-conception intégrée** : Penser durable dès la conception

Annexe A

Annexes

A.1 Structure des fichiers

A.1.1 Arborescence Symfony

Listing A.1 – Structure du projet Symfony

```
blog_app/
          config/                 # Configuration
          migrations/             # Migrations de base de donn es
          public/                 # Point d'entr e web
                _assets/           # Assets compil s
          src/
                Controller/        # Contr leurs
                      ApiController.php
                      BlogController.php
                      ArticleController.php
                      SecurityController.php
                      RegistrationController.php
                      ContactController.php
                Entity/            # Entit s Doctrine
                      User.php
                      Article.php
                      Category.php
                      Contact.php
                Form/              # Formulaires Symfony
                      ArticleType.php
                      ContactType.php
                      RegistrationFormType.php
                Repository/        # Repositories Doctrine
                Security/          # Authentification
                Services/          # Services m tier
                    CategoriesServices.php
                    UploadFile.php
          templates/             # Templates Twig
                base.html.twig
```

```
                blog/
                article/
                security/
                registration/
        var/                    # Cache et logs
```

A.1.2 Arborescence Vue.js

Listing A.2 – Structure du projet Vue.js

```
mon-blog-vue/
        public/                 # Assets statiques
                index.html
        src/
                components/       # Composants r utilisables
                        HeroSlider.vue
                views/            # Pages de l'application
                        Home.vue
                        Articles.vue
                router/           # Configuration Vue Router
                        index.js
                stores/           # Stores Pinia
                        articles.js
                assets/           # Ressources (CSS, images)
                        main.css
                App.vue           # Composant racine
                main.js           # Point d'entr e
        package.json            # D pendances npm
        vite.config.js          # Configuration Vite
```

A.2 Configuration de l'environnement

A.2.1 Prérequis

Symfony :

— PHP 8.1+

— Composer

— Symfony CLI

— Base de données (MySQL)

Vue.js :

— Node.js 16+

— NPM

A.2.2 Installation

Backend Symfony :

Listing A.3 – Installation du backend

```
# Cloner le projet
git clone [repository-url] blog_app
cd blog_app

# Installer les d pendances
composer install

# Configuration de la base de donn es
#  diter  .env.local avec vos param tres de DB

# Cr er la base et les tables
php bin/console doctrine:database:create
php bin/console doctrine:migrations:migrate

# Charger les donn es de test (optionnel)
php bin/console doctrine:fixtures:load

# D marrer le serveur
symfony server:start
```

Frontend Vue.js :

Listing A.4 – Installation du frontend

```
# Cloner le projet
git clone [repository-url] mon-slider-vue
cd mon-blog-vue

# Installer les d pendances
npm install

# D marrer le serveur de d veloppement
npm run dev
```

A.3 API Documentation

A.3.1 Endpoints disponibles

A.3.2 Format de réponse

Exemple pour /api/articles :

Listing A.5 – Format de réponse API

```
[
  {
```

Endpoint	Méthode	Description	Réponse
/api/slides	GET	Articles pour le slider	JSON Array
/api/articles	GET	Tous les articles	JSON Array
/api/articles/{slug}	GET	Article par slug	JSON Object
/api/categories	GET	Liste des catégories	JSON Array

TABLE A.1 – API REST disponible

```
    "id": 1,
    "title": "Article sur l' conomie  fran aise",
    "description": "Description tronqu e   150 caract res...",
    "image": "/_assets/images/articles/abc123.jpg",
    "slug": "article-economie-francaise",
    "createdAt": "2025-01-30 14:30:00",
    "updatedAt": null,
    "author": {
      "id": 1,
      "name": "Jean Dupont",
      "email": "jean.dupont@example.com"
    }
  }
]
```

A.4 Tests et validation

Symfony (Backend) :

- http://localhost:8000/ - Page d'accueil
- http://localhost:8000/api/articles - API articles
- http://localhost:8000/login - Connexion
- http://localhost:8000/account - Administration

Vue.js (Frontend) :

- http://localhost:5173/ - Page d'accueil Vue.js
- http://localhost:5173/articles - Listing des articles

A.5 Ressources et références

A.5.1 Documentation officielle

- **Symfony** : https://symfony.com/doc
- **Vue.js** : https://vuejs.org/guide/
- **Vue Router** : https://router.vuejs.org/
- **Pinia** : https://pinia.vuejs.org/
- **Doctrine ORM** : https://www.doctrine-project.org/

A.5.2 Éco-conception

— **RGAA** : https://accessibilite.numerique.gouv.fr/
— **RGESN** : https://ecoresponsable.numerique.gouv.fr/
— **GreenIT** : https://www.greenit.fr/
— **EcoIndex** : https://www.ecoindex.fr/

A.5.3 Outils utilisés

Outil	Version	Usage
Symfony	6.3	Framework backend
Vue.js	3.4	Framework frontend
Vite	5.0	Build tool
Pinia	2.1	State management
Doctrine	3.0	ORM
Bootstrap Icons	1.11	Icônes

TABLE A.2 – Stack technique

Bibliographie

[1] Symfony Documentation Team. *The Symfony Framework Documentation*. SensioLabs, 2025. https://symfony.com/doc

[2] Vue.js Core Team. *Vue.js Guide*. Vue.js, 2025. https://vuejs.org/guide/

[3] DINUM. *Référentiel Général d'Amélioration de l'Accessibilité*. Gouvernement français, 2023. https://accessibilite.numerique.gouv.fr/

[4] DINUM, ADEME, Institut du Numérique Responsable. *Référentiel Général d'Éco-conception de Services Numériques*. Gouvernement français, 2022. https://ecoresponsable.numerique.gouv.fr/

[5] Eduardo San Martin Morote. *Pinia Documentation*. Vue.js Ecosystem, 2025. https://pinia.vuejs.org/

[6] Doctrine Team. *Doctrine ORM Documentation*. Doctrine Project, 2025. https://www.doctrine-project.org/

[7] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, 2000.

[8] W3C Web Accessibility Initiative. *Web Content Accessibility Guidelines (WCAG) 2.1*. World Wide Web Consortium, 2018. https://www.w3.org/WAI/WCAG21/

[9] Frédéric Bordage. *Green IT : les clés pour des projets informatiques plus respectueux de l'environnement*. Eyrolles, 2019.