Contents

Dossier Bloc 2 – Documentation projet

- Protocole de déploiement continu
- Critères de qualité et de performance
- Protocole d'intégration continue
- Architecture logicielle
- Prototype
- Frameworks et paradigmes
- Sécurité
- Accessibilité
- Historique des versions
- Cahier de recettes
- Plan de correction des bogues
- Manuel de déploiement
- Manuel d'utilisation
- Manuel de mise à jour

Protocole de déploiement continu

Ce document décrit le pipeline CI/CD utilisé pour garantir la livraison continue du projet.

Outils utilisés

- GitHub Actions (ou GitLab CI)
- Docker
- Laravel Envoyer (optionnel)

Étapes du pipeline

1. **Lint & Tests** : Lancement automatique des tests unitaires et de l'analyse statique à chaque push.

2. **Build** : Construction des images Docker pour chaque service (Laravel, WordPress, MySQL).
3. **Déploiement** : Déploiement automatique sur l'environnement de staging, puis production après validation.

Sécurité

- Les secrets sont stockés dans les variables d'environnement du pipeline.
- Les accès sont restreints par clé SSH et tokens.

Critères de qualité et de performance

Qualité

- Respect des conventions PSR (PHP)
- Revue de code systématique
- Couverture de tests $> 80\%$
- Utilisation de linters (PHPStan, ESLint)

Performance

- Mise en cache (config, routes, vues)
- Optimisation des requêtes SQL
- Utilisation d'outils de profiling (Laravel Debugbar)
- Tests de charge sur endpoints critiques

Protocole d'intégration continue

- Déclenchement à chaque push sur la branche principale
- Exécution des tests unitaires et d'intégration
- Analyse statique du code
- Génération automatique de la documentation technique
- Notification en cas d'échec ou de succès

Architecture logicielle

Le projet est structuré en microservices Docker : - **Laravel** : Backend API, logique métier, authentification, gestion des utilisateurs et des rôles, API REST pour l'application et le front. - **WordPress** : CMS pour la gestion de contenu éditorial (pages, articles, médias), accessible via /wordpress. - **MySQL** : Base de données partagée, avec schémas séparés pour Laravel et WordPress.

Schéma global

```
[Client Web/Mobile]
```

```
      |
[NGINX Proxy SSL]
      |
[Laravel API] <--> [MySQL] <--> [WordPress]
```

Structure Laravel

- `app/Http/Controllers/` : Contrôleurs REST, gestion des accès, logique métier
- `app/Models/` : Modèles Eloquent (User, Article, etc.)
- `app/Repositories/` : Accès aux données, logique métier réutilisable
- `routes/` : Définition des routes (web, api, admin)
- `resources/views/` : Vues Blade pour l'admin
- `resources/lang/` : Fichiers de traduction (FR/EN)
- `tests/` : Tests unitaires et fonctionnels (PHPUnit)

Structure WordPress

- `wp-content/themes/` : Thème custom pour l'intégration graphique
- `wp-content/plugins/` : Plugins maison pour l'intégration avec Laravel (auth, API, etc.)

Maintenabilité

- Respect du principe SOLID (Single Responsibility, etc.)
- Découplage des composants (services, repositories, middlewares)
- Utilisation de design patterns (Repository, Service, Observer)
- Documentation du code (PHPDoc, README)
- CI/CD pour garantir la qualité et la non-régression

Sécurité & évolutivité

- Authentification centralisée (Sanctum)
- Gestion fine des rôles et permissions
- API versionnée pour permettre l'évolution sans rupture
- Séparation des environnements (dev, staging, prod)

Accessibilité

- Respect des standards d'accessibilité dans les vues Laravel et le thème WordPress
- Tests manuels et automatiques sur les interfaces

Présentation d'un prototype

Le prototype a été réalisé sous Figma pour valider l'ergonomie sur desktop et mobile. Il a servi de base à la conception de l'interface utilisateur et à la validation des parcours principaux avec les parties prenantes.

Choix ergonomiques

- **Navigation claire** : menu principal en haut, accès rapide aux sections clés (Accueil, Articles, Admin, Contact)
- **Contrastes respectés** : couleurs testées pour l'accessibilité (WCAG AA)
- **Responsive design** : chaque écran a été décliné en version mobile et desktop
- **Composants réutilisables** : boutons, formulaires, alertes, modales
- **Feedback utilisateur** : messages d'erreur et de succès visibles et accessibles

Parcours utilisateurs principaux

- Connexion/inscription
- Création et gestion d'articles (Laravel et WordPress)
- Accès à l'espace admin (gestion des utilisateurs, rôles)
- Consultation du contenu public

Sécurité et accessibilité

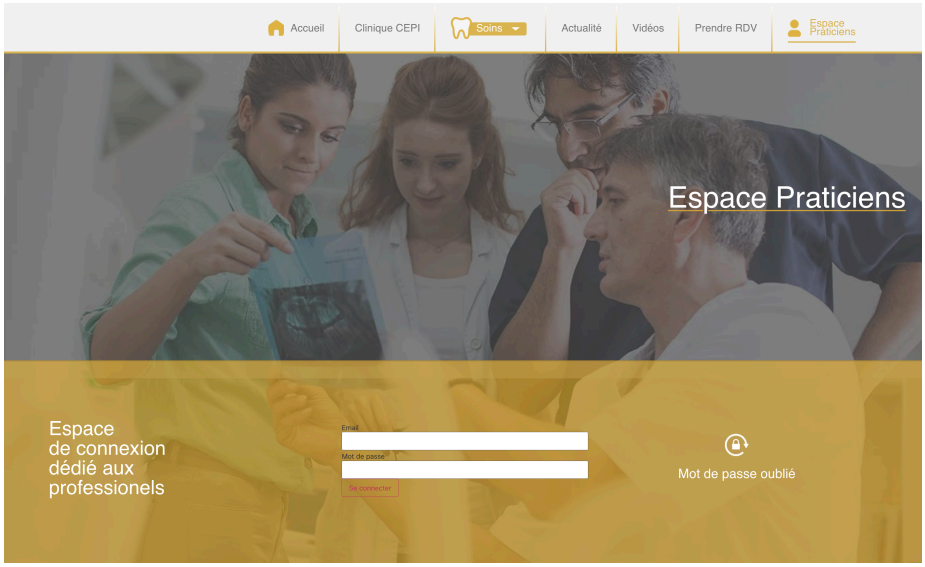
- Les écrans de connexion et d'administration intègrent des mesures de sécurité (masquage des mots de passe, validation côté client et serveur)
- Tous les formulaires sont accessibles au clavier et compatibles lecteurs d'écran

Outils utilisés

- Figma pour la conception des maquettes
- Wave et axe DevTools pour l'audit accessibilité

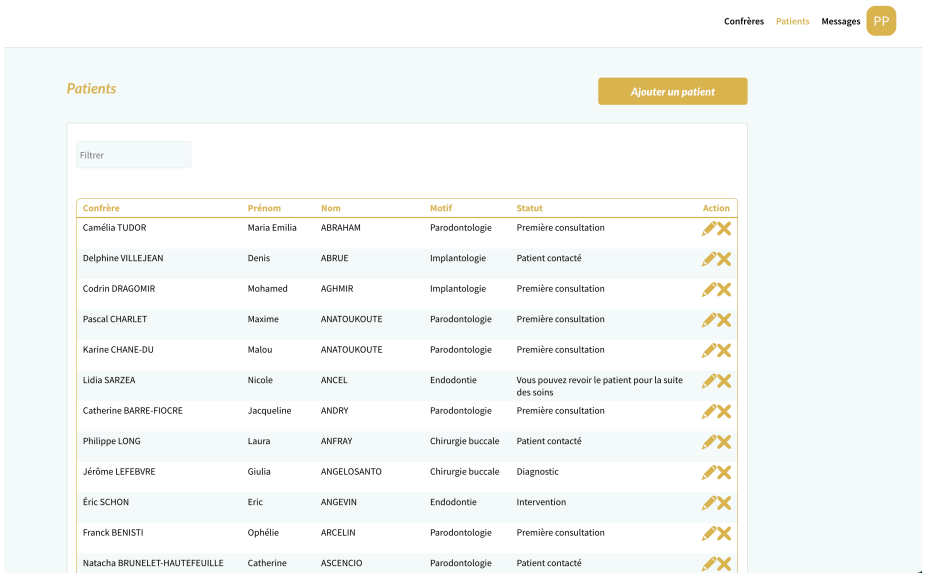
Exemples de captures d'écran

Page de connexion



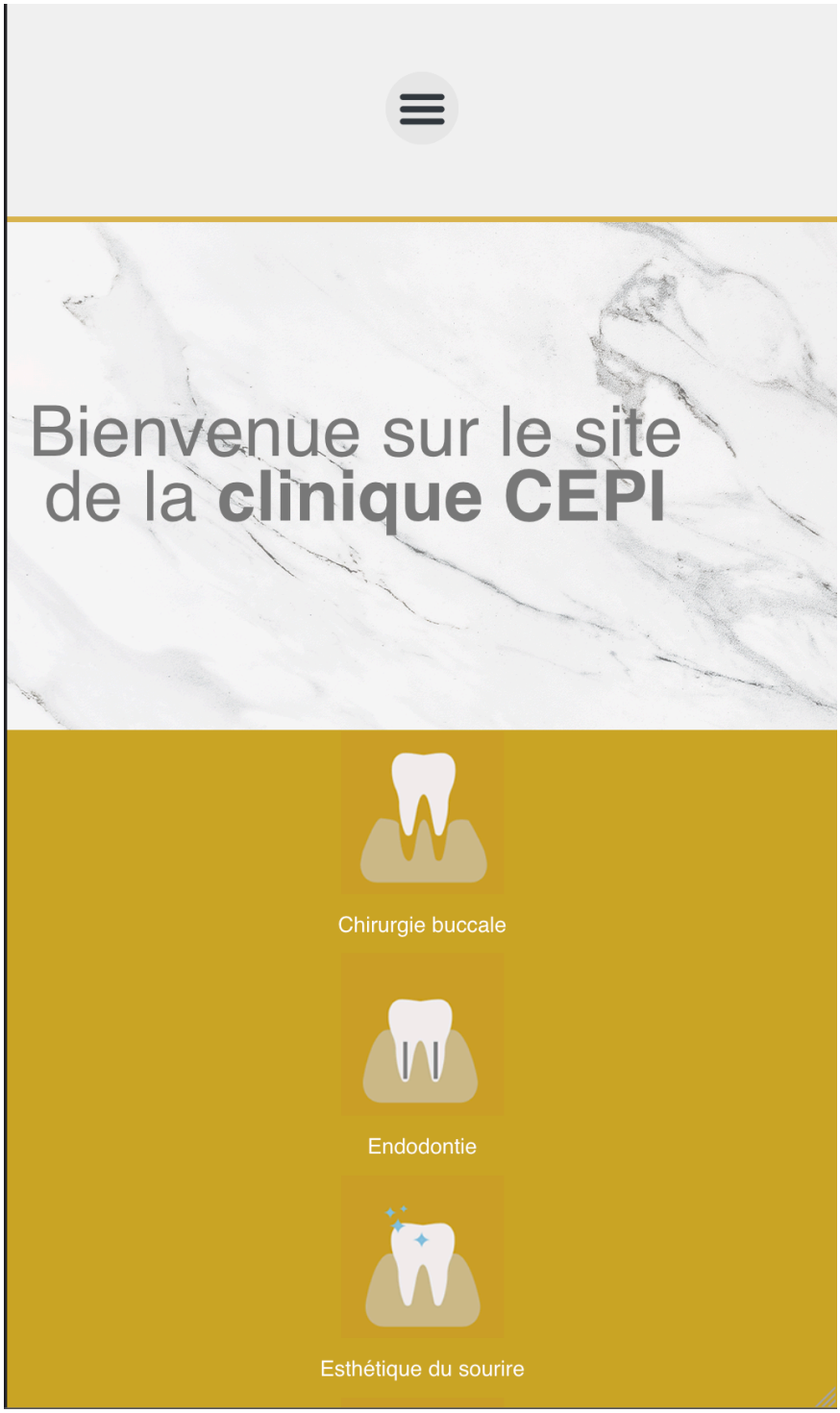
Écran de connexion avec validation et message d'erreur accessible

Tableau de bord admin



Vue d'ensemble de l'espace d'administration avec gestion des utilisateurs

Version mobile



Accueil optimisé pour smartphone

Pour plus de détails, voir la maquette complète en annexe ou dans le dossier `docs/captures/`.

Utilisation de Framework et paradigmes

- **Laravel** (MVC, injection de dépendances, Eloquent ORM)
- **WordPress** (hooks, plugins)
- Paradigmes : POO, SOLID, DRY, KISS
- Utilisation de migrations, seeders, factories pour la gestion des données

Mesures de sécurité mises en œuvre

Authentification et gestion des accès

- Authentification forte via Laravel Sanctum (API tokens, cookies sécurisés)
- Gestion des rôles et permissions (admin, éditeur, utilisateur simple) via middlewares personnalisés
- Accès restreint à l'admin Laravel et WordPress selon le rôle

Protection des données et des sessions

- Hashage des mots de passe avec bcrypt (paramétrage des rounds)
- Sécurisation des sessions (cookies httpOnly, secure, SameSite=strict)
- Chiffrement des données sensibles dans la base

Sécurité applicative

- Protection CSRF sur tous les formulaires Laravel
- Validation stricte des entrées utilisateurs (form requests, validation côté serveur)
- Protection XSS : échappement systématique dans les vues, désactivation de l'upload de scripts
- Headers de sécurité : CORS, HSTS, X-Frame-Options, X-Content-Type-Options

Sécurité du déploiement

- Variables d'environnement dans `.env` (jamais versionnées)
- Accès SSH restreint, clés déployées via CI/CD
- Mises à jour régulières des dépendances (composer, npm, plugins WP)

Surveillance et audit

- Logs centralisés (fichiers, Sentry, etc.)
- Alertes sur les erreurs critiques

- Revue de code systématique et analyse statique (PHPStan, SonarQube)

Tests de sécurité

- Tests unitaires sur les middlewares et la configuration (voir `tests/Unit/SecurityTest.php`)
- Scénarios de recette pour les accès non autorisés et la gestion des sessions

Accessibilité

Principes appliqués

- Respect des standards RGAA/WCAG 2.1 AA sur toutes les pages Laravel et WordPress
- Navigation clavier complète (tabulation logique, focus visible)
- Contrastes vérifiés (>4.5:1) sur tous les textes et boutons
- Utilisation d'attributs ARIA pour les composants dynamiques (menus, modales)
- Formulaires accessibles : labels explicites, messages d'erreur lisibles, champs obligatoires signalés

Outils utilisés

- Extension Wave, axe DevTools pour l'audit accessibilité
- Tests manuels avec NVDA/VoiceOver
- Vérification automatique via CI (pa11y, axe-core)

Actions spécifiques

- Thème WordPress custom conforme RGAA (navigation, couleurs, structure sémantique)
- Composants Laravel (boutons, alertes, modales) testés au clavier et avec lecteur d'écran
- Documentation interne sur les bonnes pratiques d'accessibilité pour les développeurs et contributeurs

Recette accessibilité

- Scénarios de test dans le cahier de recette (navigation sans souris, lecture des messages d'erreur, etc.)
- Correction continue des retours utilisateurs en situation de handicap

Historique des versions

- v1.0 : Initialisation du projet, structure Docker, base Laravel/WordPress
- v1.1 : Ajout de l'authentification et des tests unitaires

- v1.2 : Optimisation des performances et accessibilité
- v1.3 : Déploiement continu et documentation complète

Plan de correction des bogues

Procédure de gestion des bugs

1. **Signalement**
   - Le bug est signalé via une issue GitHub (ou GitLab) ou un formulaire interne accessible depuis l'application (menu "Support").
   - Le signalement doit inclure : description, capture d'écran, étapes de reproduction, gravité, environnement concerné (prod, test, local).
2. **Qualification**
   - Un membre de l'équipe vérifie la reproductibilité du bug et l'associe à une version.
   - Le bug est priorisé (bloquant, majeur, mineur) et assigné à un développeur.
3. **Correction**
   - Le développeur crée une branche dédiée (`fix/nom-bug`) et corrige le bug.
   - Un test unitaire ou fonctionnel est ajouté ou mis à jour pour éviter la régression.
4. **Revue de code**
   - La correction est relue par un autre membre de l'équipe (pull request obligatoire).
   - Vérification de la conformité aux standards et de la couverture de tests.
5. **Déploiement sur environnement de test**
   - La branche est fusionnée sur `develop` ou `staging`.
   - Les tests automatiques sont relancés (CI/CD).
   - Le bug est validé par le demandeur ou le PO.
6. **Déploiement en production**
   - Après validation, la correction est déployée sur la branche `main`.
   - Un tag de version est créé et l'historique des versions est mis à jour.

Suivi et traçabilité

- Tous les bugs sont tracés dans l'historique des versions (`docs/historique_versions.md`).
- Un tableau de bord (GitHub Projects ou équivalent) permet de suivre l'état d'avancement.
- Les corrections majeures sont documentées dans le changelog.

Manuel de déploiement

1. Cloner le dépôt

2. Copier le fichier `.env.example` en `.env` et configurer les variables
3. Lancer `docker-compose up -d`
4. Exécuter les migrations Laravel : `docker-compose exec laravel php artisan migrate --seed`
5. Accéder à l'application via http://localhost

Pour la production, adapter les variables d'environnement et sécuriser les accès.

Manuel d'utilisation

- Connexion utilisateur
- Navigation dans l'interface
- Création/modification de contenu (WordPress)
- Gestion des utilisateurs (Laravel)
- Déconnexion

Des captures d'écran sont disponibles dans le dossier `/docs/`.

Manuel de mise à jour

1. Récupérer les dernières modifications : `git pull`
2. Mettre à jour les dépendances :
   - Laravel : `docker-compose exec laravel composer install`
   - Node : `docker-compose exec laravel npm install && npm run prod`
   - WordPress : via l'interface admin
3. Appliquer les migrations si besoin : `docker-compose exec laravel php artisan migrate`
4. Redémarrer les services si nécessaire : `docker-compose restart`