Documentation Technique

Sommaire

1. Stack technique

Composant	Technologie	Version
Frontend	React + TypeScript + Vite	React 19
CSS	Tailwind CSS	4.x
Routage	React Router	v7
Backend	ASP.NET Core Web API	.NET 8
BDD relationnelle	PostgreSQL	18
BDD NoSQL	MongoDB	8.0
Hashage MDP	Argon2id (Isopoh.Cryptography.Argon2)	-
Auth	JWT Bearer Token	-
Email	Brevo API	-
Conteneurisation	Docker + Docker Compose	-
CI/CD	GitHub Actions	-
Déploiement Frontend	Vercel	-
Déploiement Backend	Railway	-
Déploiement MongoDB	MongoDB Atlas	-

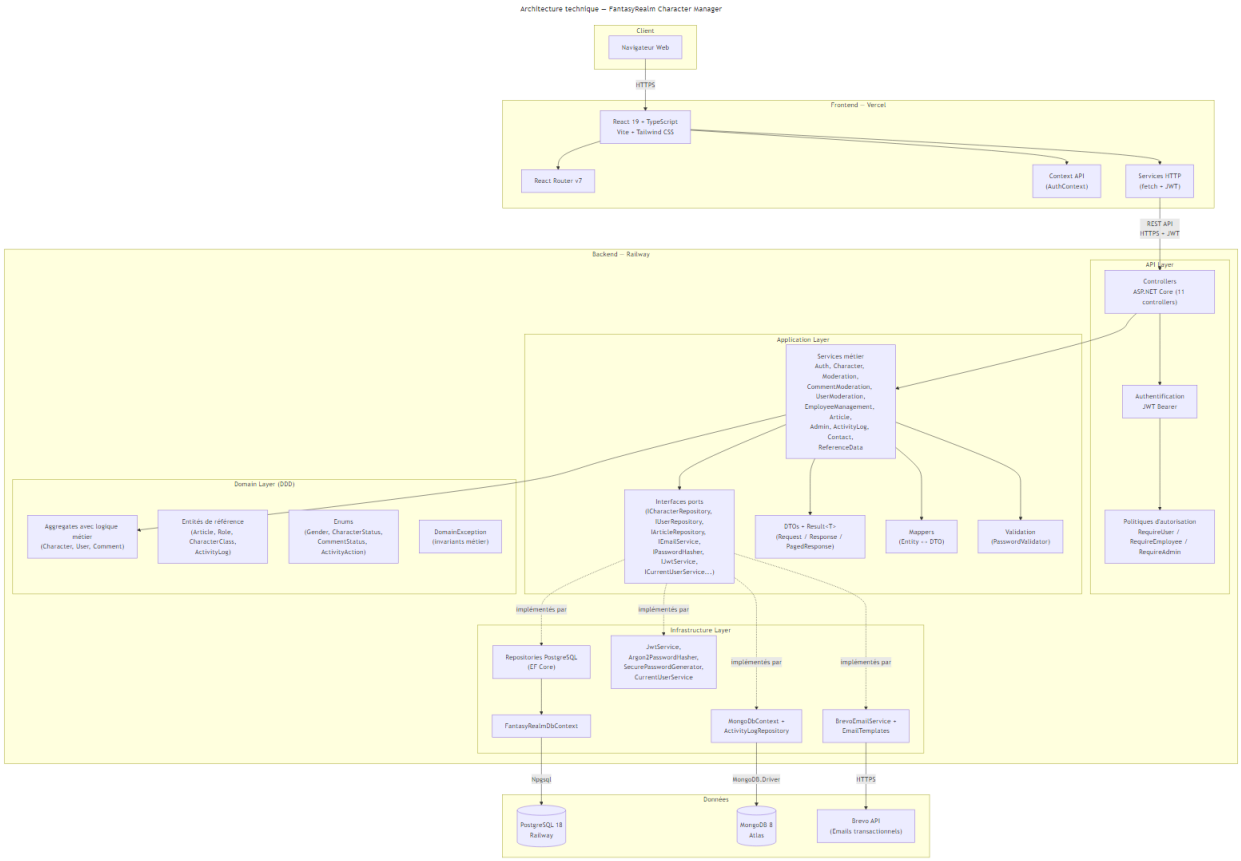
---

2. Architecture

2.1. Vue d'ensemble

L'application suit une architecture client-serveur classique :

- Le **frontend** (React) communique avec le **backend** (.NET API) via des appels REST en HTTPS avec authentification JWT.
- Le backend accède à **PostgreSQL** pour les données relationnelles et à **MongoDB** pour les logs d'activité.
- Les emails transactionnels sont envoyés via l'API **Brevo**.



2.2. Clean Architecture et DDD

Le backend est structuré en 4 couches suivant les principes de Clean Architecture, enrichi de patterns Domain-Driven Design (DDD) :

Domain Layer (aucune dépendance) :

- **Aggregates** : Character, User, Comment — contiennent la logique métier (factory methods, transitions d'état, validation des invariants)
- **Entités de référence** : Article, Role, CharacterClass, ActivityLog
- **Enums** : Gender, CharacterStatus, CommentStatus, ActivityAction
- **DomainException** : exception levée lors de violations métier

Application Layer (dépend de Domain) :

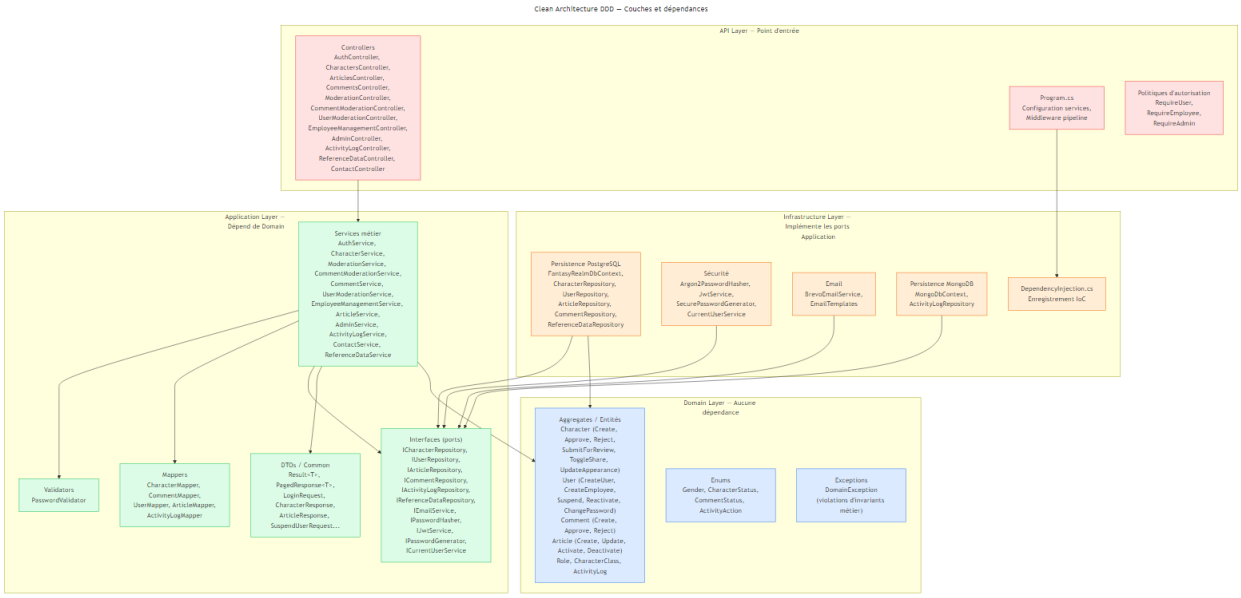
- 12 services métiers (AuthService, CharacterService, ModerationService, etc.)
- Interfaces "ports" (ICharacterRepository, IEmailService, IPasswordHasher, etc.)
- DTOs (Request/Response) avec le pattern Result pour la gestion d'erreurs
- Mappers (Entity vers DTO) et Validators (PasswordValidator)

Infrastructure Layer (implémente les ports Application) :

- Persistence PostgreSQL : Entity Framework Core, DbContext, Repositories
- Persistence MongoDB : MongoDbContext, ActivityLogRepository
- Sécurité : Argon2PasswordHasher, JwtService, SecurePasswordGenerator, CurrentUserService
- Email : BrevoEmailService, EmailTemplates

API Layer (point d'entrée) :

- 12 controllers ASP.NET Core
- Program.cs (configuration services, middleware pipeline)
- Politiques d'autorisation : RequireUser, RequireEmployee, RequireAdmin



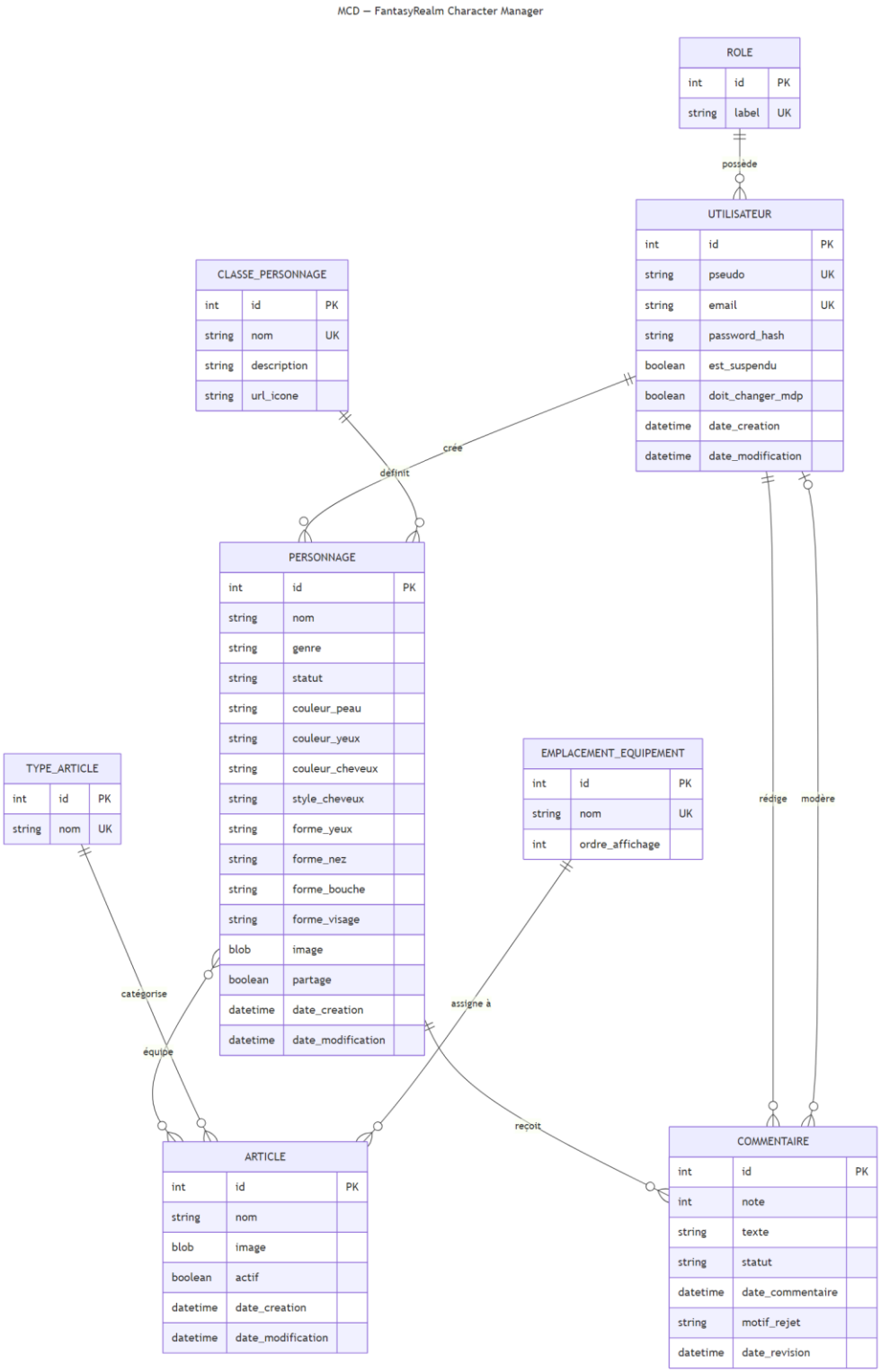
2.3. Principes SOLID

Principe	Application dans le projet
S - Single Responsibility	Chaque service a une responsabilité unique (AuthService, ModerationService, etc.)
O - Open/Closed	Les interfaces permettent d'étendre sans modifier (ex: changer de fournisseur email)
L - Liskov Substitution	Les repositories implémentent des interfaces substituables
I - Interface Segregation	Interfaces spécifiques et granulaires (IEmailService, IPasswordHasher, IJwtService)
D - Dependency	Les couches internes définissent les interfaces, les couches externes

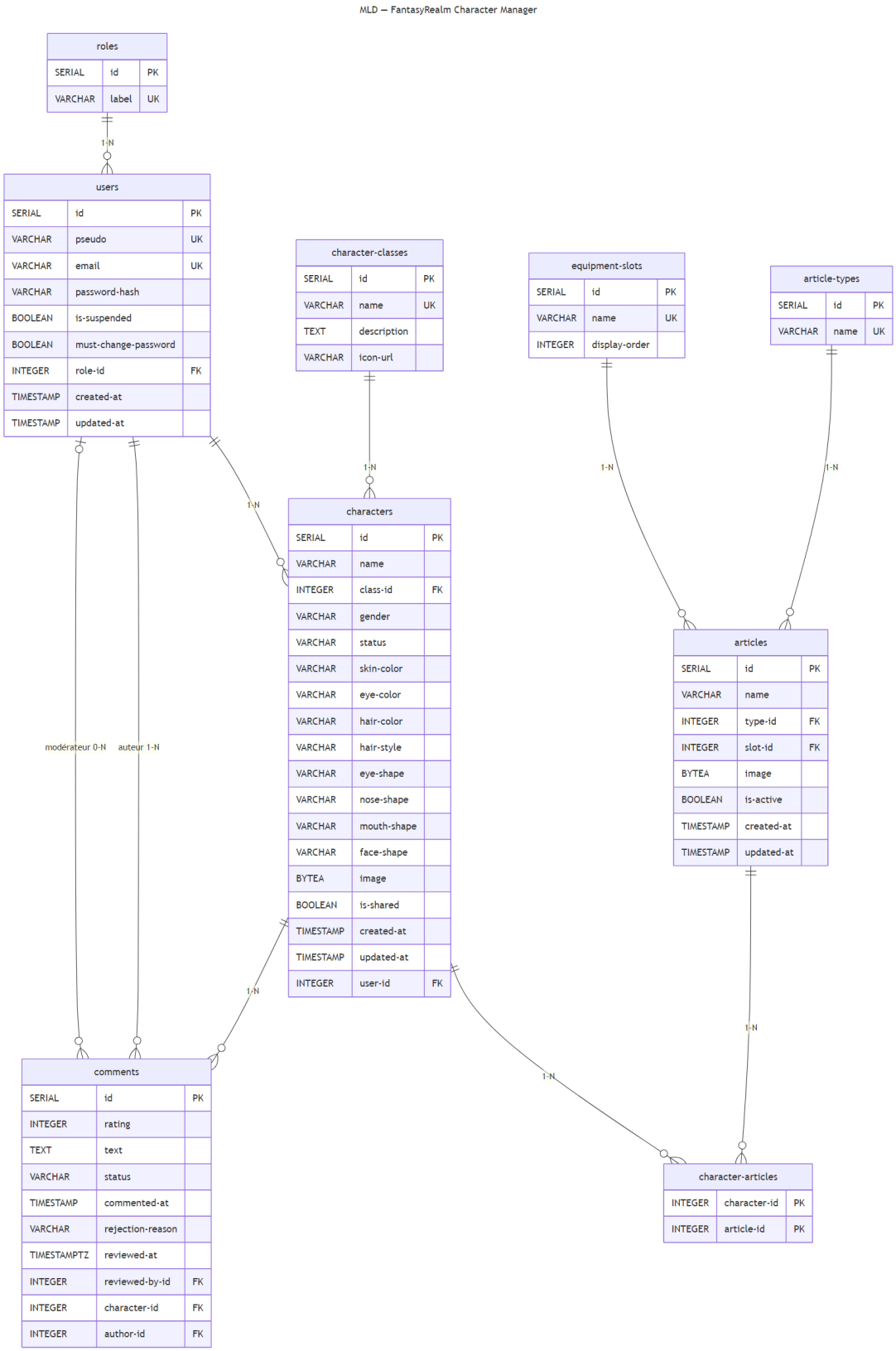
Principe	Application dans le projet
Inversion	les implémentent

3. Modèle de données

3.1. Modèle Conceptuel (MCD)



3.2. Modèle Logique (MLD)



3.3. MongoDB - Logs d’activité

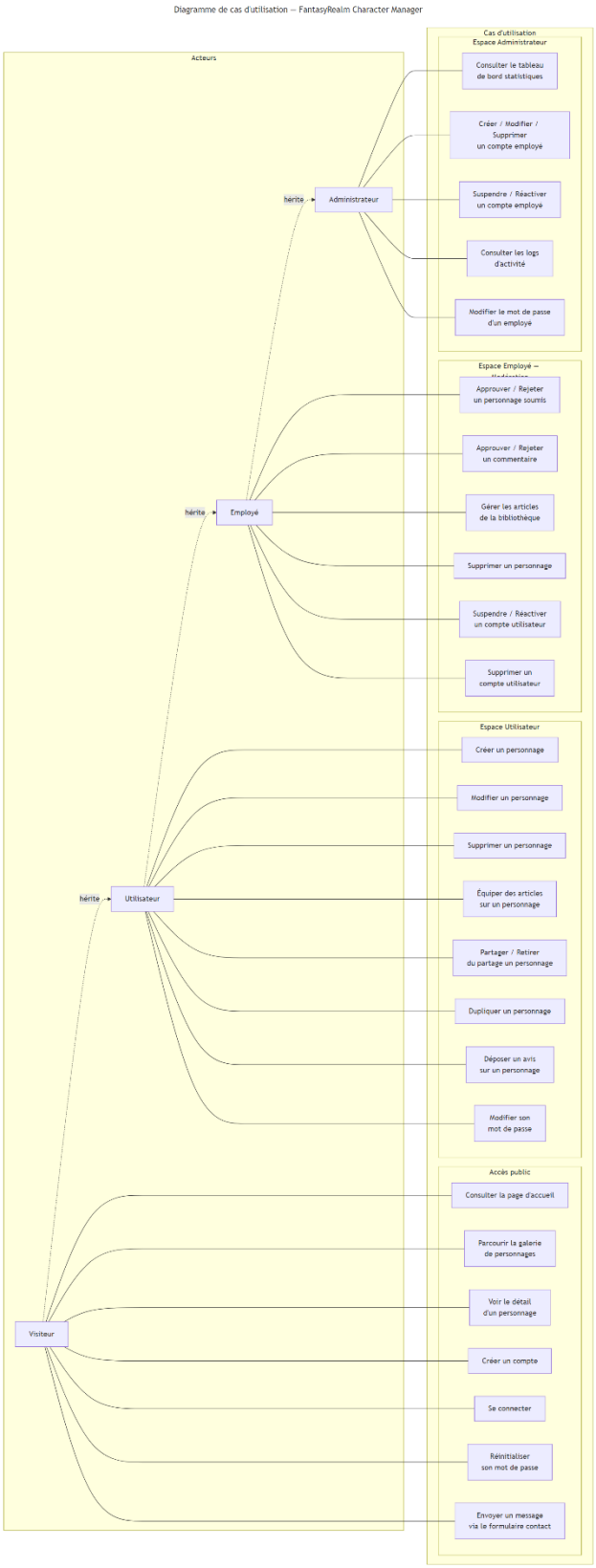
Les logs d’activité sont stockés dans MongoDB (base NoSQL) car leur structure est flexible et ils n’ont pas de relations avec les autres entités. Exemple de document :


```
{
  "_id": "ObjectId(...)",
  "action": "CharacterApproved",
  "target_type": "Character",
  "target_id": 42,
  "target_name": "Aragorn",
  "details": null,
  "user_id": 2,
  "user_pseudo": "employe1",
  "timestamp": "2026-02-19T10:30:00Z"
}
```


Les 16 types d’actions journalisées : CharacterApproved, CharacterRejected, CommentApproved, CommentRejected, ArticleCreated, ArticleUpdated, ArticleDeleted, UserSuspended, UserReactivated, UserDeleted, EmployeeCreated, EmployeeSuspended, EmployeeReactivated, EmployeeDeleted, EmployeePasswordReset, PasswordChanged.

---

4. Cas d’utilisation



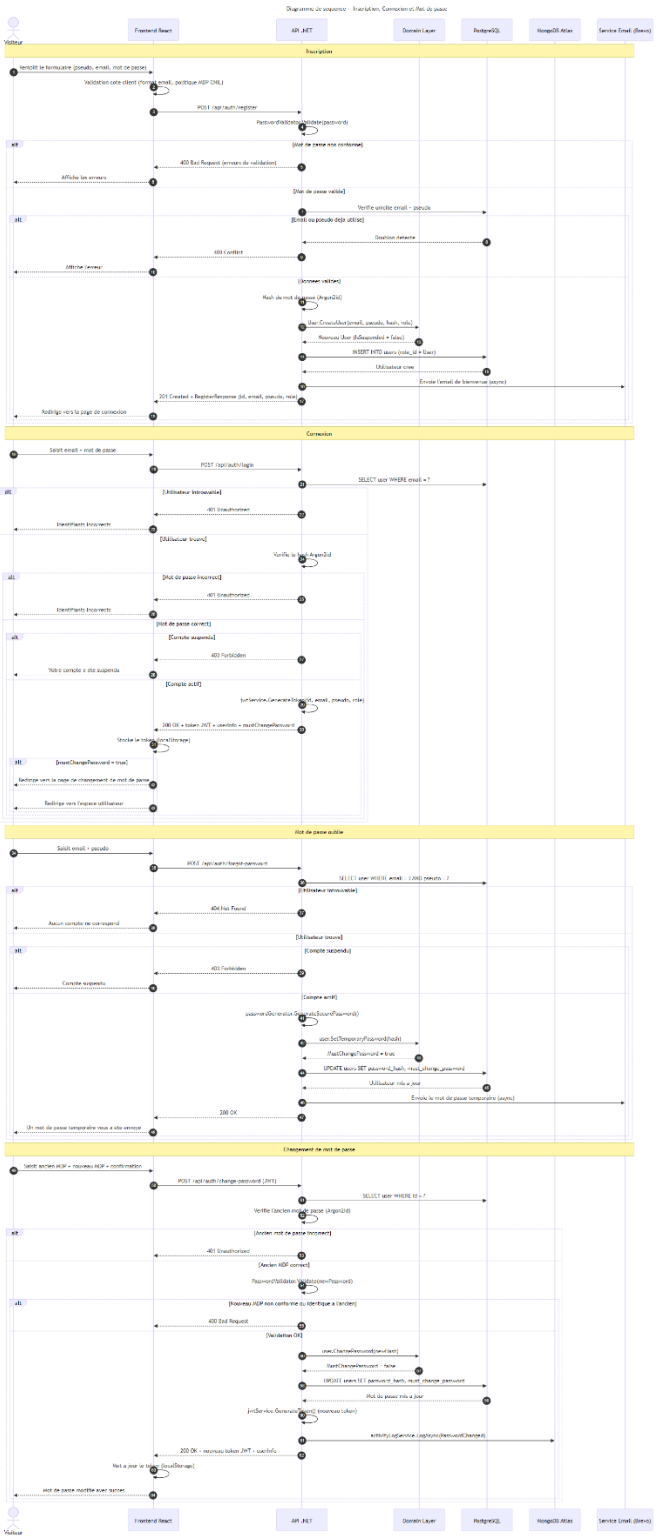
L’application distingue 4 profils :

1. **Visiteur** : consulte la galerie, s’inscrit, se connecte, utilise le formulaire de contact
2. **Utilisateur** : crée/modifie/supprime ses personnages, les équipe, les partage, dépose des avis
3. **Employé** : modère les personnages et commentaires, gère les comptes utilisateurs
4. **Administrateur** : consulte les statistiques, gère les employés, consulte les logs d’activité

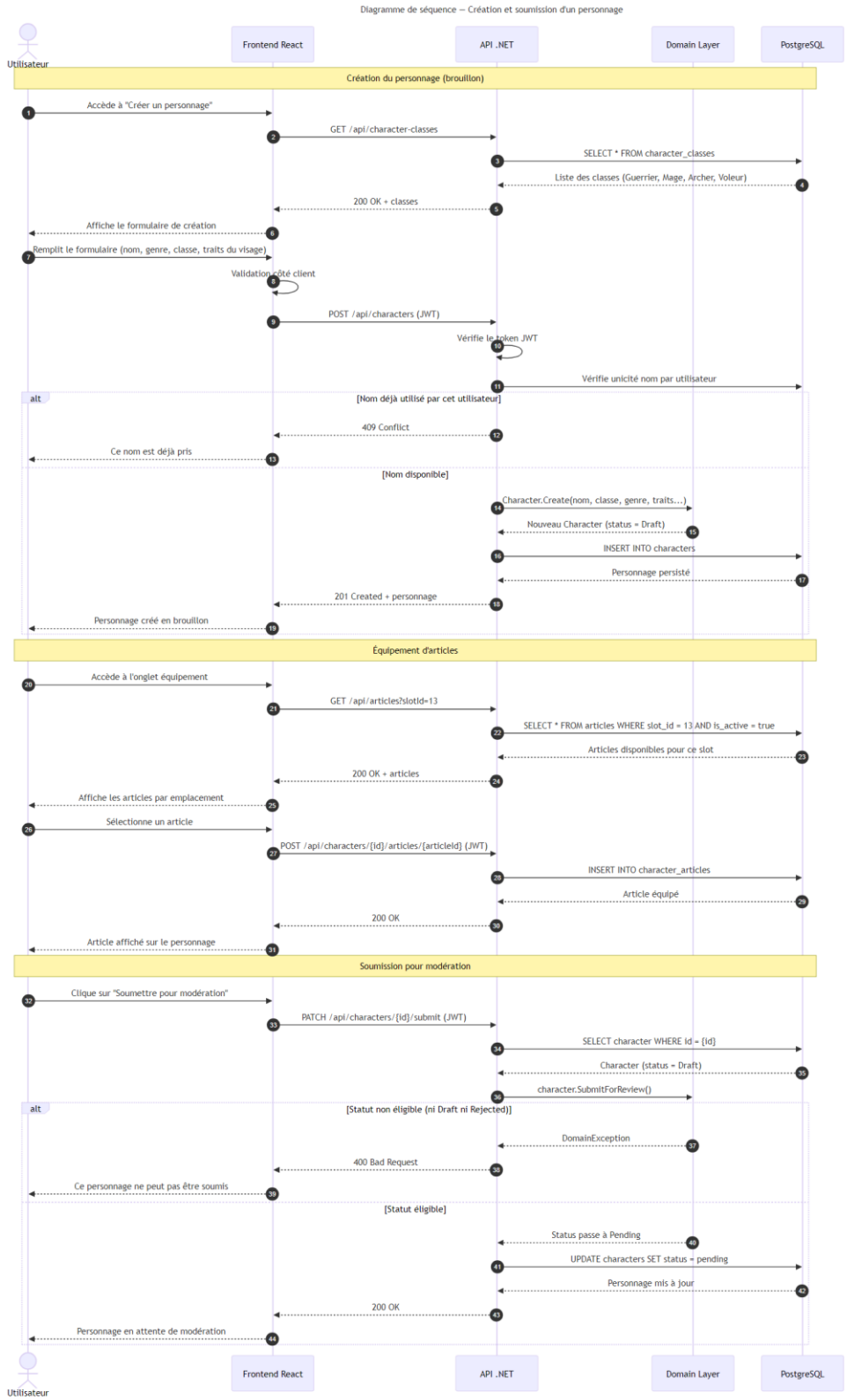
---

5. Diagrammes de séquence

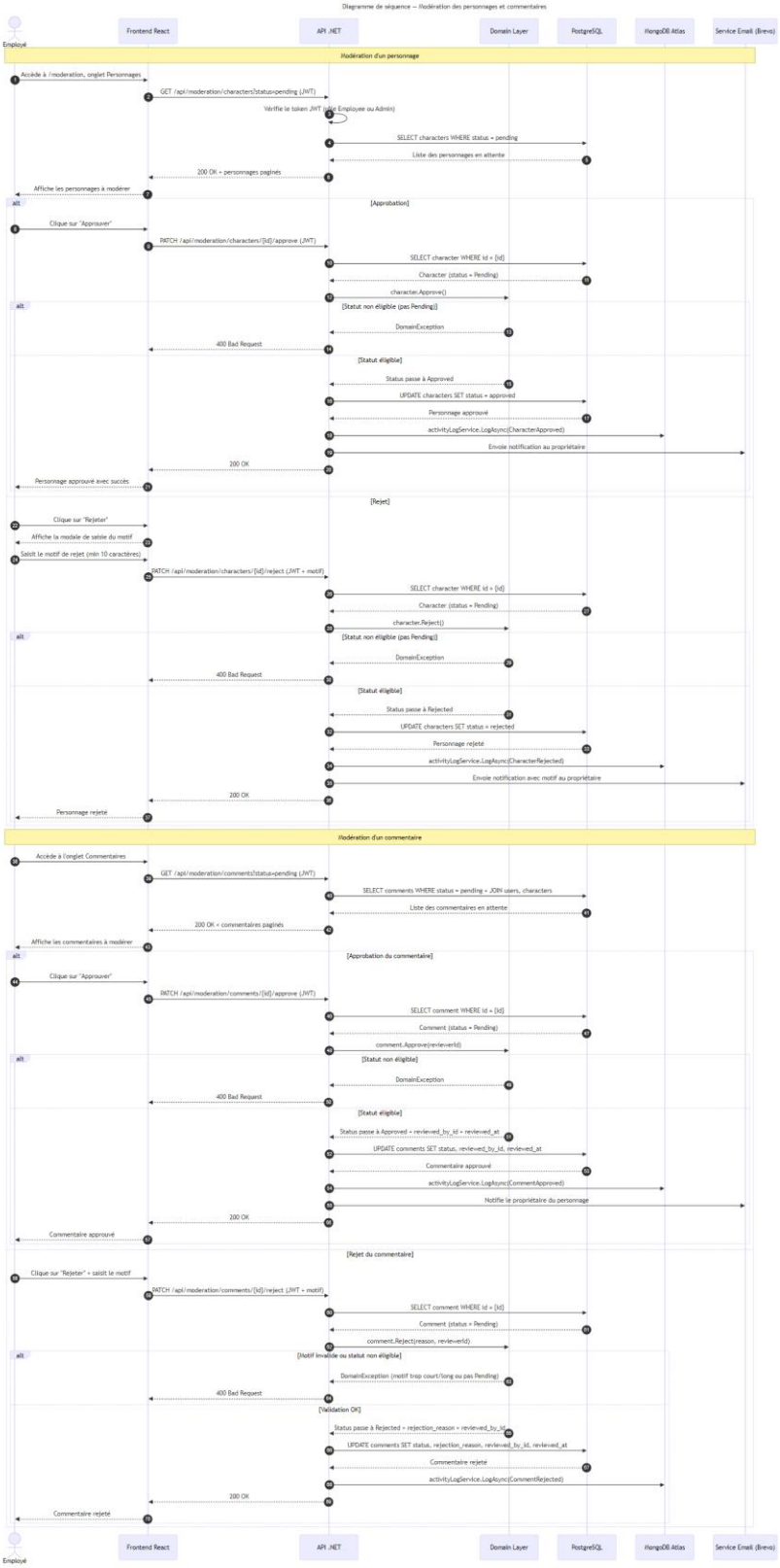
5.1. Inscription et Connexion



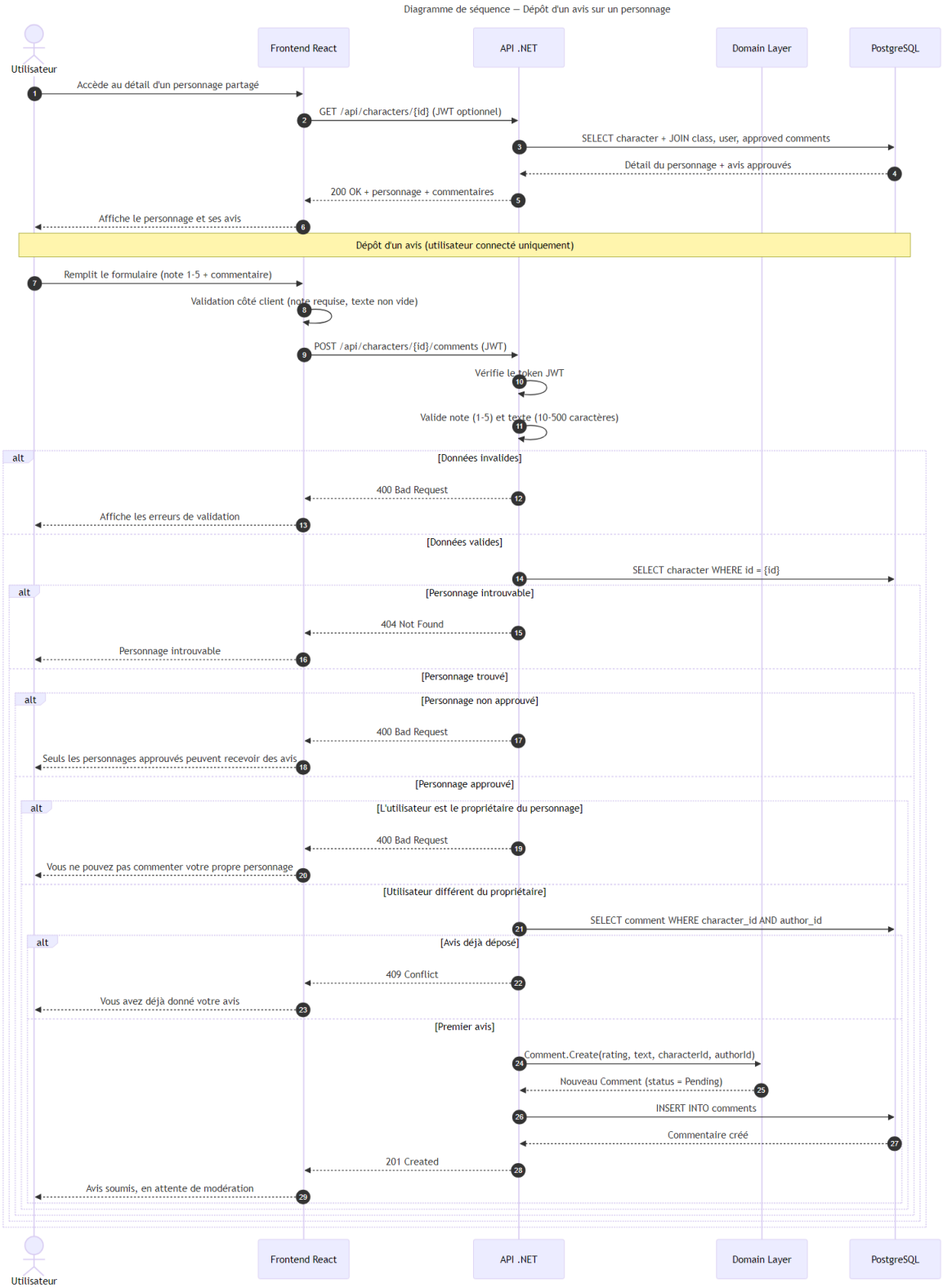
5.2. Création d’un personnage



5.3. Modération



5.4. Dépôt d’un avis



6. Endpoints API REST

Authentification (api/auth)

Méthode	Route	Description	Auth
POST	/register	Inscription	Non
POST	/login	Connexion (retourne JWT)	Non
POST	/forgot-password	Demande de réinitialisation MDP	Non
POST	/change-password	Changement de mot de passe	Oui
GET	/me	Informations utilisateur courant	Oui

Personnages (api/characters)

Méthode	Route	Description	Auth
POST	/	Créer un personnage	User
GET	/mine	Mes personnages	User
GET	/	Galerie publique (filtres, pagination)	Non
GET	/{id}	Détail d'un personnage	Non
PUT	/{id}	Modifier un personnage	User
DELETE	/{id}	Supprimer un personnage	User
PATCH	/{id}/submit	Soumettre pour modération	User
PATCH	/{id}/share	Partager / Retirer du partage	User
POST	/{id}/duplicate	Dupliquer un personnage	User
GET	/check-name	Vérifier disponibilité du nom	User

Commentaires (api/comments)

Méthode	Route	Description	Auth
POST	/api/characters/{id}/comments	Déposer un avis	User
GET	/api/characters/{id}/comments	Avis approuvés d'un personnage	Non
GET	/api/characters/{id}/comments/mine	Mon avis sur ce personnage	User
DELETE	/api/comments/{id}	Supprimer mon avis	User

Modération Personnages (api/moderation/characters)

Méthode	Route	Description	Auth
GET	/	Personnages en attente (paginé)	Employee
PATCH	/{id}/approve	Approuver	Employee

Méthode	Route	Description	Auth
PATCH	/{id}/reject	Rejeter (motif obligatoire)	Employee

Modération Commentaires (api/moderation/comments)

Méthode	Route	Description	Auth
GET	/	Commentaires en attente (paginé)	Employee
PATCH	/{id}/approve	Approuver	Employee
PATCH	/{id}/reject	Rejeter (motif obligatoire)	Employee

Modération Utilisateurs (api/moderation/users)

Méthode	Route	Description	Auth
GET	/	Liste des utilisateurs (recherche, filtre)	Employee
GET	/count	Nombre total d'utilisateurs	Employee
PATCH	/{id}	Suspendre / Réactiver	Employee
DELETE	/{id}	Supprimer un compte	Employee

Administration (api/admin)

Méthode	Route	Description	Auth
GET	/stats	Statistiques de la plateforme	Admin

Gestion Employés (api/admin/employees)

Méthode	Route	Description	Auth
GET	/	Liste des employés (recherche, filtre)	Admin
GET	/count	Nombre total d'employés	Admin
POST	/	Créer un employé	Admin
PATCH	/{id}	Suspendre / Réactiver	Admin
POST	/{id}/reset-password	Réinitialiser le MDP	Admin
DELETE	/{id}	Supprimer un employé	Admin

Logs d'activité (api/admin/activity-logs)

Méthode	Route	Description	Auth
GET	/	Logs paginés (filtres: action, dates)	Admin

Données de référence (api/)

Méthode	Route	Description	Auth
GET	/character-classes	Classes de personnages	Non
GET	/equipment-slots	Emplacements d'équipement	Non

Contact (api/contact)

Méthode	Route	Description	Auth
POST	/	Envoyer un message de contact	Non

---

7. Sécurité

7.1. Authentification JWT

Le backend génère un token JWT signé (clé symétrique) contenant l'identifiant, le pseudo, l'email et le rôle de l'utilisateur. Le token a une durée de vie limitée. Le frontend le stocke dans le localStorage et l'envoie dans l'en-tête Authorization: Bearer <token> à chaque requête.

7.2. Politiques d'autorisation

Trois politiques sont définies dans le middleware ASP.NET Core :

- **RequireUser** : rôles User, Employee ou Admin
- **RequireEmployee** : rôles Employee ou Admin
- **RequireAdmin** : rôle Admin uniquement

7.3. Hashage des mots de passe — Argon2id

Les mots de passe sont hachés avec Argon2id (recommandé par l'OWASP). Cet algorithme est résistant aux attaques par GPU/ASIC grâce à son coût en mémoire configurable.

7.4. Validation des mots de passe (CNIL)

Conformément aux recommandations de la CNIL, le mot de passe doit contenir :

- Au moins 8 caractères
- Au moins une majuscule
- Au moins une minuscule
- Au moins un chiffre
- Au moins un caractère spécial

La validation est effectuée côté frontend (temps réel) et côté backend (PasswordValidator).

7.5. Protection CORS

Seules les origines autorisées (domaine Vercel du frontend) peuvent accéder à l'API. En développement, localhost:5173 est autorisé.

7.6. Règles métiers avec DDD

Les règles métier sont protégées au niveau du Domain Layer via des DomainException :

- Un personnage ne peut être approuvé que s'il est en statut Pending
- Un compte ne peut être suspendu que s'il n'est pas déjà suspendu
- Un motif de rejet doit contenir entre 10 et 500 caractères
- etc.

---

8. Conformité

8.1. RGPD

- **Minimisation des données** : seuls email, pseudo et mot de passe (haché) sont collectés
- **Droit à l'effacement** : suppression du compte et des données associées (FK CASCADE)
- **Journalisation** : les actions de modération sont tracées dans MongoDB
- **Pseudonymisation** : utilisation de pseudos plutôt que de noms réels

8.2. RGAA (Accessibilité)

- Navigation au clavier fonctionnelle
- Liens d'évitement
- Balisage sémantique HTML
- Attributs ARIA sur les composants interactifs
- Contrastes de couleurs respectés (ratio 4.5:1 minimum)
- Hiérarchie des titres logique (h1, h2, h3…)
- Labels associés aux champs de formulaires
- Messages d'erreur clairs et accessibles

8.3. CNIL

- Politique de mot de passe conforme (voir section 7.4)
- Pas de stockage de mot de passe en clair

---

9. Tests

Backend

- Framework : **xUnit** + **Moq** + **FluentAssertions**
- Tests unitaires sur les services métier
- Exécution dans le pipeline CI avant chaque déploiement

Frontend

- Framework : **Vitest** + **React Testing Library** + **jest-axe**
- Tests unitaires des composants et pages
- Tests d'accessibilité automatisés (jest-axe)
- Exécution dans le pipeline CI avant chaque déploiement

---

10. Structure du projet

```
fantasyrealm-character-manager/
├── src/
│   ├── frontend/              # React 19 + TypeScript + Vite
│   │   ├── src/
│   │   │   ├── components/       # 10 dossiers de composants
│   │   │   ├── pages/            # 16 pages
│   │   │   ├── services/         # Appels API (fetch + JWT)
│   │   │   ├── context/          # AuthContext
│   │   │   ├── types/            # Types TypeScript
│   │   │   └── utils/            # Utilitaires
│   │   └── package.json
│   │
│   └── backend/               # .NET 8
│       ├── src/
│       │   ├── FantasyRealm.Api/          # 12 Controllers
│       │   ├── FantasyRealm.Application/  # 12 Services, DTOs, Mappers
│       │   ├── FantasyRealm.Domain/       # Aggregates, Enums, Exceptions
│       │   └── FantasyRealm.Infrastructure/ # EF Core, MongoDB, Argon2, Brevo
│       └── tests/
│           └── FantasyRealm.Tests.Unit/    # Tests xUnit
│
├── infra/                     # Docker Compose (5 services)
├── database/
│   ├── sql/                   # Scripts PostgreSQL
│   └── mongodb/               # Scripts MongoDB
└── .github/workflows/         # CI/CD GitHub Actions
```