

[Destiny Raid Companion]

Dossier de Projet CDA
Présentation et Défense

Informations du Projet

Candidat : [Rochetin Lucas]

Promotion : [Colint School]

Soutenance : [Date]

Ce dossier présente le projet réalisé dans le cadre de la formation
Concepteur Développeur d'Applications (CDA) de Colint School

Année académique 2025-2026

Table des matières

1	Présentation personnelle et du projet	5
1.1	Rôle du candidat et contexte	5
1.2	Problématique et objectifs SMART	6
1.3	Liens utiles	8
2	Cadrage et cahier des charges	9
2.1	Objectifs métier, techniques et pédagogiques	9
2.2	Cibles et parties prenantes	11
2.3	Exigences fonctionnelles	12
2.3.1	Fonctionnalités « Front Office »	12
2.3.2	Fonctionnalités « Back Office »	13
2.3.3	L'utilisateur (public)	13
2.3.4	Confidentialité	13
2.3.5	Droits d'accès	14
2.3.6	Authentification	14
2.4	Exigences et choix techniques	15
2.4.1	Exigences	15
2.4.2	Choix	15
2.5	Définition du MVP	16
2.6	Roadmap	17
2.7	Liens utiles	18
3	Méthodologie et organisation	19
3.1	Gestion de projet avec GitHub	19
3.1.1	User Stories et estimation de temps	19
3.2	Versioning GitHub et conventions	20
3.3	Planification et outils de suivi	21
3.4	Estimation de temps et planification	23
3.5	Liens utiles	25
4	Conception fonctionnelle et technique	27
4.1	Use Cases et diagrammes UML	27
4.2	Diagrammes de séquence	28
4.3	Conception de l'interface graphique	29
4.3.1	Zoning	29
4.3.2	Wireframe	30
4.3.3	Maquettage	30
4.3.4	Outils de conception et diagrammes	30
4.3.5	Charte graphique	31
4.4	Conception de base de données	32
4.4.1	MCD (Modèle Conceptuel de Données)	32
4.4.2	MLD (Modèle Logique de Données)	32

4.4.3	MPD (Modèle Physique de Données)	32
4.5	Architecture 3 tiers	34
4.6	Liens utiles	36
5	Architecture 3 tiers	37
5.1	Architecture 3 tiers	37
5.1.1	Couche Présentation (Frontend)	37
5.1.2	Couche Logique Métier (Backend)	37
5.1.3	Couche Données (Database)	39
5.1.4	Communication entre les tiers	39
5.1.5	Avantages de l'architecture 3 tiers	40
5.2	Développement Frontend	41
5.3	Développement Backend	43
5.4	Gestion des données	45
5.5	Liens utiles	47
6	Sécurité applicative et RGPD	49
6.1	Protection contre les vulnérabilités OWASP	49
6.2	Authentification et autorisation	51
6.3	Conformité RGPD	54
6.4	Liens utiles	57
7	Tests et qualité logicielle	59
7.1	Stratégie de tests	59
7.2	Tests de performance	61
7.3	Qualité du code avec SonarQube	63
7.4	Liens utiles	66
8	Déploiement et CI/CD	67
8.1	Containerisation avec Docker	67
8.2	Pipeline CI/CD avec GitHub Actions	69
8.3	Documentation et monitoring	74
8.4	Liens utiles	78
9	Veille technologique et sécurité	79
9.1	Veille technologique stack	79
9.2	Bonnes pratiques sécurité	80
9.3	Application au projet	81
9.4	Liens utiles	83
10	Bilan et retour d'expérience (REX)	85
10.1	Objectifs atteints et non atteints	85
10.2	Difficultés rencontrées et solutions	85
10.3	Dettes techniques et apprentissages	86
10.4	Liens utiles	88
11	Conclusion et remerciements	89
11.1	Synthèse du projet	89
11.2	Perspectives d'évolution	90
11.3	Remerciements	91
11.4	Déploiement et documentation	91
11.4.1	Docker	92
11.4.2	GitHub (code source)	93
11.4.3	CI/CD	94
11.4.4	SonarQube	94

11.4.5 Swagger	95
11.5 Liens utiles	97

Chapitre 1

Présentation personnelle et du projet

1.1 Rôle du candidat et contexte

Dans cette section, vous devez présenter votre rôle dans le projet et le contexte organisationnel. Le jury attend une explication claire de vos responsabilités et de l'environnement de travail dans lequel s'inscrit votre projet CDA.

Votre rôle : *Concepteur et développeur fullstack en autonomie totale*

Contexte organisationnel : *Le projet Destiny Raid Companion s'inscrit dans un contexte de développement indépendant, visant à répondre aux besoins spécifiques de la communauté des joueurs de Destiny 2. Les enjeux métier consistent à améliorer l'expérience utilisateur dans les activités complexes du jeu (raids et donjons) en offrant une plateforme centralisée. Les contraintes techniques incluent l'intégration de l'API Bungie, le développement d'une architecture scalable et la gestion des données temps réel. Les contraintes organisationnelles concernent le travail en autonomie avec une planification rigoureuse sur 8 mois.*

Durée et planning : *Le projet s'étend sur une période de **huit mois**, d'octobre 2025 à mai 2026, à raison de 2 jours par semaine (environ 60 à 70 jours effectifs). Les grandes phases sont : Octobre : cadrage du projet, installation de l'environnement, maquettes. Novembre – Décembre : développement backend (API, base de données, authentification Bungie). Janvier – Février : développement frontend (guides, escouades, calendrier, profils). Mars : intégration de l'API Destiny 2. Avril : phase de tests unitaires. Mai : dockerisation, CI/CD et déploiement.*

Exemple

Exemple de pitch QQQQCP :

- **Quoi** : Application de gestion des projets internes
- **Qui** : Équipes de développement et management
- **Où** : Environnement cloud hybride
- **Quand** : Déploiement progressif sur 6 mois
- **Comment** : Architecture microservices avec React/Node.js
- **Pourquoi** : Centraliser et optimiser le suivi des projets

Votre pitch QQQQCP :

- **Quoi** : *Destiny Raid Companion - plateforme web/mobile avec guides interactifs, gestion d'escouades, calendrier collaboratif et recommandations d'équipements*
- **Qui** : *Joueurs de Destiny 2 (débutants et expérimentés), développé en autonomie complète*
- **Où** : *Application web responsive déployée sur cloud*
- **Quand** : *Développement octobre 2025 - mai 2026, déploiement avant soutenance*
- **Comment** : *Architecture 3-tiers (React/Node.js/PostgreSQL) avec API Bungie et MongoDB*
- **Pourquoi** : *Résoudre les problèmes d'organisation et de compréhension des raids/donjons*

À FAIRE / À VÉRIFIER**Ce que le jury attend dans cette section :**

- Une présentation claire de votre rôle et de vos responsabilités
- Une explication du contexte métier et des enjeux
- Une justification de la pertinence du projet
- Un pitch QQQQCP synthétique et percutant
- Des indicateurs de succès mesurables

Conseils de rédaction :

- Soyez précis sur votre fonction (évitez les généralités)
- Montrez votre compréhension des enjeux métier
- Justifiez le choix de votre projet
- Utilisez des chiffres et des métriques quand c'est possible

Contrôles Jury CDA

- Pouvez-vous présenter votre rôle précis dans ce projet ?
- Quel est le contexte métier de votre entreprise ?
- Quels sont les enjeux techniques principaux ?
- Comment mesurez-vous le succès de votre projet ?
- Quelles sont les contraintes temporelles et budgétaires ?

1.2 Problématique et objectifs SMART

Dans cette section, vous devez identifier clairement la problématique que votre projet résout et définir des objectifs SMART mesurables. Le jury attend une analyse précise des enjeux et des bénéfices attendus.

Problématique identifiée : *Les raids et donjons de Destiny 2 manquent de clarté et de soutien intégré : les joueurs peinent à comprendre les mécaniques, à trouver des coéquipiers fiables et à planifier leurs sessions, ce qui entraîne frustration et abandon des activités complexes.*

Objectifs SMART :

- **Spécifique** : Développer une plateforme centralisée avec guides interactifs, gestion d'escouades et calendrier collaboratif
- **Mesurable** : Réduire de 50% le temps de recherche d'équipe et de préparation des raids
- **Atteignable** : Livrer la version MVP en 8 mois avec les technologies maîtrisées (React/Node.js)
- **Pertinent** : Améliorer l'expérience globale des joueurs et favoriser l'engagement communautaire
- **Temporel** : Déploiement complet avant la soutenance CDA en mai 2026

Bénéfices attendus :

- Centralisation des outils dispersés (guides, LFG, planning)
- Amélioration de l'accessibilité pour les nouveaux joueurs
- Optimisation du temps d'organisation des sessions de jeu
- Renforcement de l'engagement communautaire via la gamification

À FAIRE / À VÉRIFIER**Ce que le jury attend dans cette section :**

- Une problématique clairement identifiée et justifiée
- Des objectifs SMART précis et mesurables
- Une compréhension des enjeux métier
- Des indicateurs de succès quantifiés
- Un diagramme de contexte montrant les acteurs

Conseils de rédaction :

- Soyez spécifique sur les impacts négatifs actuels
- Quantifiez vos objectifs (pourcentages, délais, volumes)
- Montrez la pertinence métier de votre projet
- Utilisez des diagrammes pour clarifier les interactions

Contrôles Jury CDA**Questions de contrôle du jury :**

- Pouvez-vous expliquer clairement la problématique que votre projet résout ?
- Vos objectifs sont-ils vraiment SMART (spécifiques, mesurables, atteignables, pertinents, temporels) ?
- Comment mesurez-vous le succès de votre projet ?
- Quels sont les bénéfices attendus pour l'entreprise ?
- Pouvez-vous présenter un diagramme de contexte de votre projet ?
- Quelles sont les contraintes temporelles et budgétaires ?

1.3 Liens utiles

- GitHub About : <https://docs.github.com/>
- SMART Goals : <https://bit.ly/smart-goals-atlassian>
- Project Management Institute : <https://www.pmi.org/>
- Agile Manifesto : <https://agilemanifesto.org/>
- Business Model Canvas : <https://bit.ly/business-model-canvas>

Chapitre 2

Cadrage et cahier des charges

2.1 Objectifs métier, techniques et pédagogiques

Dans cette section, vous devez définir clairement les trois types d'objectifs de votre projet. Le jury attend une distinction nette entre les objectifs métier (bénéfices pour l'entreprise), techniques (performance, architecture) et pédagogiques (apprentissages CDA).

Objectifs métier :

- *Améliorer l'expérience utilisateur des joueurs de Destiny 2*
- *Réduire de 50% le temps moyen d'organisation des raids et donjons*
- *Fidéliser la communauté via un système de gamification (badges, scores)*
- *Centraliser les outils dispersés (guides, LFG, planning) en une plateforme unique*

Objectifs techniques :

- *Performance : temps de réponse < 2 secondes pour 95% des requêtes*
- *Scalabilité : support de 1000 utilisateurs simultanés en pic d'activité*
- *Disponibilité : 99% uptime avec monitoring cloud*
- *Sécurité : authentification OAuth Bungie et chiffrement des données sensibles*
- *Maintenabilité : documentation technique complète et tests automatisés*

Objectifs pédagogiques :

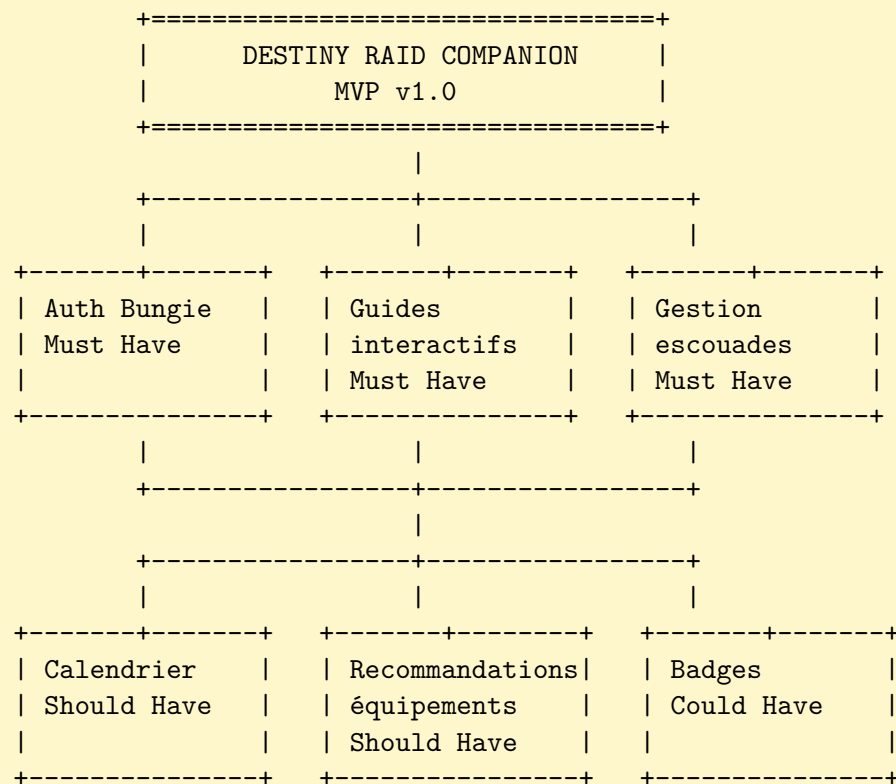
- *Maîtriser le développement fullstack avec React/Node.js*
- *Implémenter une architecture 3-tiers scalable*
- *Gérer l'intégration d'API tierces complexes : Bungie API*
- *Mettre en œuvre une stratégie de tests complète*
- *Déployer une application cloud avec CI/CD automatisé*

Exemple**Exemple de tableau MoSCoW :**

Priorité	Fonctionnalité	Justification
Must Have	Authentification	Sécurité obligatoire
Must Have	Gestion des projets	Cœur métier
Should Have	Tableaux de bord	Amélioration UX
Could Have	Notifications push	Plus-value
Won't Have	IA prédictive	Hors périmètre v1

Votre tableau MoSCoW :

Priorité	Fonctionnalité	Justification
Must Have	Authentification Bungie	Accès aux données utilisateur
Must Have	Guides interactifs raids	Cœur de la valeur ajoutée
Must Have	Gestion escouades	Fonctionnalité collaborative essentielle
Should Have	Calendrier collaboratif	Planification des sessions
Should Have	Recommandations équipements	Optimisation gameplay
Could Have	Système de badges	Gamification et engagement
Could Have	Chat en temps réel	Communication d'équipe
Won't Have	Application mobile native	Hors périmètre MVP

Votre diagramme de périmètre MVP :

À FAIRE / À VÉRIFIER**Ce que le jury attend dans cette section :**

- Une distinction claire entre objectifs métier, techniques et pédagogiques
- Une priorisation MoSCoW justifiée et documentée
- Un périmètre MVP bien délimité et réaliste
- Des critères d'acceptation mesurables et vérifiables
- Une analyse des contraintes et risques identifiés

Conseils de rédaction :

- Quantifiez vos objectifs (pourcentages, délais, volumes)
- Justifiez chaque priorité MoSCoW par des arguments métier
- Montrez la cohérence entre MVP et objectifs SMART
- Utilisez des diagrammes pour visualiser le périmètre

Contrôles Jury CDA**Questions de contrôle du jury :**

- Pouvez-vous distinguer clairement vos objectifs métier, techniques et pédagogiques ?
- Comment avez-vous priorisé vos fonctionnalités avec la méthode MoSCoW ?
- Votre MVP est-il vraiment minimal et réaliste ?
- Quels sont vos critères de succès mesurables ?
- Comment gérez-vous les changements de périmètre ?
- Avez-vous identifié les contraintes et risques du projet ?

2.2 Cibles et parties prenantes

Dans cette section, vous devez identifier et analyser vos utilisateurs cibles et toutes les parties prenantes du projet. Le jury attend une compréhension claire des besoins de chaque groupe et de leur influence sur le projet.

Exemple**Personae utilisateurs :**

- **Thomas (Débutant)** : 25 ans, découvre les raids, besoin de guides détaillés et d'une équipe patiente
- **Sarah (Joueuse expérimentée)** : 30 ans, cherche à optimiser ses sessions et trouver des coéquipiers fiables
- **Alex (Leader de clan)** : 35 ans, organise régulièrement des raids pour son clan de 20 joueurs

Parties prenantes :

- **Utilisateurs finaux** : Joueurs de Destiny 2 (débutants et expérimentés)
- **Développeur** : Concepteur et développeur fullstack en autonomie
- **Communauté Destiny 2** : Influenceurs et leaders d'opinion
- **Bungie** : Éditeur du jeu via son API officielle

Matrice d'influence :

Partie prenante	Influence	Intérêt
Utilisateurs finaux	Élevée	Très élevé
Développeur	Très élevée	Très élevé
Communauté Destiny 2	Moyenne	Élevé
Bungie (API)	Élevée	Faible

La cartographie des parties prenantes facilite la communication et la gestion des attentes

tout au long du projet. Cette approche systémique garantit que tous les besoins sont pris en compte dans la conception de la solution.

Exemple

User Stories prioritaires :

- En tant que **chef de projet**, je veux visualiser l'avancement des tâches pour optimiser la planification
- En tant que **développeur**, je veux mettre à jour mon statut rapidement pour informer l'équipe
- En tant que **manager**, je veux générer des rapports automatiques pour le suivi des performances

Critères d'acceptation :

- Le tableau de bord se charge en moins de 2 secondes
- Les données sont synchronisées en temps réel
- L'interface est responsive sur mobile et desktop

À FAIRE / À VÉRIFIER

- Créer des personae détaillés avec leurs besoins spécifiques
- Identifier toutes les parties prenantes du projet
- Analyser l'influence et l'intérêt de chaque partie prenante
- Définir des user stories avec critères d'acceptation clairs
- Organiser des sessions de validation avec les utilisateurs

Contrôles Jury CDA

- Qui sont vos utilisateurs cibles principaux ?
- Comment avez-vous validé vos user stories ?
- Quelles parties prenantes ont le plus d'influence ?
- Vos critères d'acceptation sont-ils mesurables ?
- Comment gérez-vous les besoins contradictoires ?

2.3 Exigences fonctionnelles

Les exigences fonctionnelles définissent précisément ce que le système doit faire pour répondre aux besoins métier. Elles couvrent les fonctionnalités front-end (interface utilisateur), back-end (logique métier), la gestion des rôles et droits d'accès, ainsi que les aspects de confidentialité et d'authentification. Cette spécification détaillée guide le développement et sert de référence pour les tests d'acceptation.

La sécurité et la confidentialité des données constituent des exigences critiques qui influencent directement l'architecture technique et les choix de développement. L'authentification robuste et la gestion fine des autorisations sont essentielles pour protéger les informations sensibles.

2.3.1 Fonctionnalités « Front Office »

Dans cette sous-section, vous devez détailler toutes les fonctionnalités accessibles aux utilisateurs finaux. Le jury attend une description précise de l'interface utilisateur et des interactions possibles.

Vos fonctionnalités Front Office :

- **Page d'accueil** : Présentation des fonctionnalités et connexion
- **Guides interactifs** : Étapes détaillées des raids avec illustrations
- **Gestion d'escouades** : Création, invitation, gestion des membres

- **Calendrier collaboratif** : Planification des sessions avec rappels
- **Profil joueur** : Statistiques, badges, historique des raids
- **Recherche** : Trouver des joueurs ou escouades par critères
- **Recommandations** : Équipements optimisés selon l'activité

Exemple

Exemple de fonctionnalités Front Office :

- **Dashboard** : Vue d'ensemble des projets et tâches
- **Gestion projets** : Création, modification, suppression
- **Gestion tâches** : Attribution, suivi, mise à jour statut
- **Profil utilisateur** : Modification informations personnelles
- **Recherche** : Filtrage et recherche dans les projets

2.3.2 Fonctionnalités « Back Office »

Dans cette sous-section, vous devez présenter les fonctionnalités administratives et de gestion du système. Le jury attend une distinction claire entre les fonctions métier et les fonctions d'administration.

Vos fonctionnalités Back Office :

- **Administration utilisateurs** : Modération et gestion des comptes
- **Gestion du contenu** : Création et mise à jour des guides
- **Analytics** : Statistiques d'usage et métriques d'engagement
- **Logs système** : Monitoring des appels API et erreurs
- **Backup** : Sauvegarde automatique des données

Exemple

Exemple de fonctionnalités Back Office :

- **Gestion utilisateurs** : Création, modification, désactivation comptes
- **Gestion rôles** : Attribution et modification des permissions
- **Rapports système** : Logs, métriques, statistiques d'usage
- **Configuration** : Paramètres système, maintenance
- **Sauvegardes** : Gestion des sauvegardes et restauration

2.3.3 L'utilisateur (public)

Dans cette sous-section, vous devez définir clairement qui peut accéder au système et dans quelles conditions. Le jury attend une analyse des différents types d'utilisateurs et de leurs besoins spécifiques.

Exemple

Vos types d'utilisateurs :

- **Joueur standard** : Accès aux guides, calendrier, recherche d'escouades
- **Leader d'escouade** : Droits étendus pour gérer une escouade
- **Administrateur** : Accès complet au back office et modération
- **Utilisateur non connecté** : Accès limité à la présentation et connexion

2.3.4 Confidentialité

Dans cette sous-section, vous devez détailler les mesures de protection des données et le respect de la vie privée. Le jury attend une approche conforme au RGPD et aux bonnes pratiques de sécurité.

Vos mesures de confidentialité :

- **Chiffrement** : Données sensibles chiffrées (mots de passe, tokens API)
- **Minimisation** : Collecte uniquement des données nécessaires
- **Consentement** : Acceptation explicite des CGU et politique de confidentialité
- **Droit à l'oubli** : Suppression des comptes et données sur demande
- **Audit** : Logs de sécurité et monitoring des accès

Exemple

Exemple de mesures de confidentialité :

- **Chiffrement** : Données sensibles chiffrées en base
- **Accès contrôlé** : Logs d'accès et audit trail
- **RGPD** : Consentement, droit à l'oubli, portabilité
- **Anonymisation** : Données anonymisées pour les rapports

2.3.5 Droits d'accès

Dans cette sous-section, vous devez présenter votre système de permissions et de contrôle d'accès. Le jury attend une matrice claire des droits par rôle et fonctionnalité.

Exemple

Votre matrice de droits :

Rôle	Créer escouade	Modifier guides	Admin users	Accès API
Non connecté	X	X	X	X
Joueur	✓	X	X	✓
Leader	✓	X	X	✓
Admin	✓	✓	✓	✓

2.3.6 Authentification

Dans cette sous-section, vous devez détailler votre système d'authentification et de gestion des sessions. Le jury attend une approche sécurisée et robuste.

Votre système d'authentification :

- **OAuth Bungie** : Connexion via compte Bungie officiel
- **Sessions** : JWT avec expiration 24h et refresh automatique
- **Sécurité** : Validation des tokens et protection CSRF
- **Déconnexion** : Invalidation immédiate du token côté serveur

Exemple

Exemple de système d'authentification :

- **Connexion** : Email/mot de passe avec validation
- **Sessions** : JWT avec expiration automatique
- **Sécurité** : Mot de passe fort, 2FA optionnel
- **Récupération** : Reset par email sécurisé

À FAIRE / À VÉRIFIER

- Spécifier toutes les fonctionnalités front-end et back-end
- Définir clairement les rôles et droits d'accès
- Documenter les exigences de confidentialité
- Prévoir les mécanismes d'authentification et d'autorisation
- Valider les exigences avec les utilisateurs métier

Contrôles Jury CDA

- Quelles sont vos exigences fonctionnelles prioritaires ?
- Comment gérez-vous les droits d'accès ?
- Vos exigences sont-elles testables ?
- Avez-vous prévu la confidentialité des données ?
- Comment validez-vous les exigences avec les utilisateurs ?

2.4 Exigences et choix techniques

L'architecture 3 tiers (présentation, logique métier, données) offre une séparation claire des responsabilités et facilite la maintenance. PostgreSQL assure la cohérence transactionnelle des données métier, tandis que MongoDB optimise le stockage des rapports et logs grâce à sa flexibilité documentaire. Cette approche hybride maximise les performances selon le type de données traitées.

Les choix techniques sont guidés par les exigences de performance, de scalabilité et de maintenabilité. L'utilisation de technologies éprouvées réduit les risques techniques tout en permettant une évolution progressive de la solution.

2.4.1 Exigences

Dans cette sous-section, vous devez détailler toutes les contraintes techniques et les exigences non-fonctionnelles de votre système. Le jury attend une analyse complète des performances, sécurité, scalabilité et maintenabilité.

Vos exigences techniques :

- **Performance** : Temps de réponse < 2s, support 1000 utilisateurs simultanés
- **Sécurité** : HTTPS obligatoire, OAuth Bungie, audit logs complets
- **Disponibilité** : 99% uptime, sauvegardes automatiques quotidiennes
- **Scalabilité** : Architecture microservices-ready, cache Redis pour l'API
- **Maintenabilité** : Code documenté, couverture de tests > 80%, CI/CD
- **Responsive** : Support mobile/tablette/desktop avec PWA

Exemple**Exemple d'exigences techniques :**

- **Performance** : Temps de réponse < 2s, support 100 utilisateurs simultanés
- **Sécurité** : HTTPS obligatoire, authentification forte, audit logs
- **Disponibilité** : 99.5% uptime, sauvegardes quotidiennes
- **Scalabilité** : Architecture horizontale, cache Redis
- **Maintenabilité** : Code documenté, tests automatisés

2.4.2 Choix

Dans cette sous-section, vous devez justifier vos choix technologiques par rapport aux exigences identifiées. Le jury attend une analyse comparative et une justification claire de chaque décision technique.

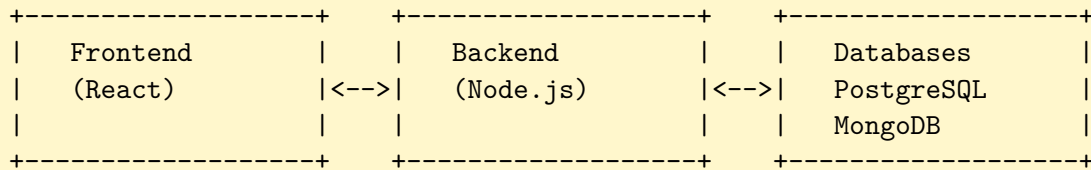
Vos choix techniques :

- **Frontend** : React + TailwindCSS pour la réactivité et le design system
- **Backend** : Node.js/Express pour la compatibilité avec l'écosystème JavaScript
- **Base SQL** : PostgreSQL pour l'intégrité des données utilisateurs et escouades
- **Base NoSQL** : MongoDB pour la flexibilité des logs et données d'analytics
- **Cache** : Redis pour optimiser les appels API Bungie

— **Conteneurisation** : Docker pour la reproductibilité des environnements

Exemple

Architecture logique simplifiée :



Exemple de modèle PostgreSQL :

```

1 CREATE TABLE users (
2     id SERIAL PRIMARY KEY,
3     email VARCHAR(255) UNIQUE NOT NULL,
4     role_id INTEGER REFERENCES roles(id),
5     created_at TIMESTAMP DEFAULT NOW()
6 );
7
8 CREATE INDEX idx_users_email ON users(email);

```

À FAIRE / À VÉRIFIER

- Justifier chaque choix technique par des critères objectifs
- Documenter l'architecture 3 tiers et les responsabilités
- Expliquer l'utilisation de PostgreSQL et MongoDB
- Prévoir l'évolution et la scalabilité de l'architecture
- Évaluer les alternatives techniques considérées

Contrôles Jury CDA

- Pourquoi avez-vous choisi cette architecture ?
- Comment justifiez-vous l'utilisation de deux bases de données ?
- Votre architecture est-elle scalable ?
- Quels sont les points de défaillance potentiels ?
- Avez-vous considéré des alternatives techniques ?

2.5 Définition du MVP

Le MVP concentre les fonctionnalités essentielles pour valider l'hypothèse produit : authentification, gestion des projets de base, et tableau de bord simple. Cette approche permet d'obtenir un retour utilisateur précoce et d'ajuster la roadmap en conséquence. Les scénarios essentiels couvrent les cas d'usage les plus fréquents et critiques pour le métier.

Votre définition du MVP : *Le MVP de Destiny 2 Raid Companion inclut :*

- Authentification via compte Bungie
- Guides interactifs pour 3 raids principaux
- Système de création et gestion d'escouades
- Calendrier basique pour planifier les sessions
- Profil joueur avec statistiques de base
- Interface responsive web (pas d'app mobile native)

Exemple**Scénarios essentiels MVP :**

1. Connexion utilisateur et gestion de session
2. Création et modification d'un projet
3. Attribution des tâches aux membres de l'équipe
4. Suivi de l'avancement en temps réel
5. Génération de rapports basiques

À FAIRE / À VÉRIFIER

- Délimiter précisément le périmètre du MVP
- Identifier les scénarios essentiels prioritaires
- Valider chaque fonctionnalité avec les utilisateurs
- Mesurer l'impact de chaque feature
- Prévoir des critères de succès clairs

Contrôles Jury CDA

- Votre MVP est-il vraiment minimal ?
- Quels sont vos scénarios essentiels ?
- Comment validez-vous chaque fonctionnalité ?
- Avez-vous des critères de succès mesurables ?
- Comment gérez-vous les demandes hors périmètre ?

2.6 Roadmap

La roadmap v1 vers v2 prévoit l'ajout progressif de fonctionnalités avancées basées sur les retours utilisateurs et les besoins métier émergents. Cette approche itérative minimise les risques et optimise l'allocation des ressources.

Votre roadmap :

- **v1.0 (Mai 2026)** : MVP avec fonctionnalités de base
- **v1.1 (Juillet 2026)** : Système de badges et gamification
- **v1.2 (Septembre 2026)** : Chat en temps réel et notifications
- **v1.3 (Novembre 2026)** : Recommandations avancées d'équipements
- **v2.0 (Janvier 2027)** : Application mobile native et analytics avancées

Exemple**Roadmap v1 ➔ v2 :**

- **v1.0** : Fonctionnalités de base (MVP)
- **v1.1** : Notifications et alertes
- **v1.2** : Intégrations externes (API)
- **v2.0** : Analytics avancées et IA

À FAIRE / À VÉRIFIER

- Délimiter précisément le périmètre du MVP
- Identifier les scénarios essentiels prioritaires
- Planifier la roadmap v1 vers v2 de manière réaliste
- Prévoir des jalons de validation utilisateur
- Documenter les critères de passage de version

Contrôles Jury CDA

- Votre MVP est-il vraiment minimal ?
- Quels sont vos scénarios essentiels ?
- Comment validez-vous le passage en v2 ?
- Votre roadmap est-elle réaliste ?
- Comment gérez-vous les changements de priorité ?

2.7 Liens utiles

- User Stories : <https://www.mountaingoatsoftware.com/agile/user-stories>
- MoSCoW : <https://www.productplan.com/glossary/moscow-prioritization/>
- PostgreSQL Docs : <https://www.postgresql.org/docs/>
- MongoDB Modeling : <https://bit.ly/mongodb-modeling>
- Architecture 3-tier : https://en.wikipedia.org/wiki/Multitier_architecture

Chapitre 3

Méthodologie et organisation

3.1 Gestion de projet avec GitHub

Dans cette section, vous devez présenter votre approche de gestion de projet entièrement centralisée sur GitHub. Le jury attend une démonstration de votre maîtrise des outils GitHub pour la planification, le suivi et la réalisation de votre projet.

Votre approche GitHub : *Le projet Destiny 2 Raid Companion est géré entièrement sur GitHub avec une approche Agile adaptée au développement en solo. L'organisation repose sur GitHub Projects pour le suivi des tâches, les Milestones pour la planification temporelle, et un workflow Git Flow modifié pour assurer la qualité du code.*

3.1.1 User Stories et estimation de temps

Dans cette sous-section, vous devez détailler vos user stories avec des estimations de temps réalistes. Chaque user story doit être liée à des commits et des milestones GitHub pour un suivi précis de l'avancement.

Vos user stories avec estimations :

User Story	Critères d'acceptation	Temps estimé
En tant que joueur, je veux m'authentifier avec mon compte Bungie	Connexion OAuth fonctionnelle, récupération du profil, gestion des sessions JWT	5 jours
En tant que joueur, je veux consulter des guides interactifs de raids	Affichage des étapes détaillées, illustrations, stratégies par rôle	8 jours
En tant que joueur, je veux créer et gérer une escouade	Création d'escouade, invitation de membres, gestion des rôles	6 jours
En tant que joueur, je veux planifier une session de raid	Calendrier interactif, création d'événement, notifications	7 jours
En tant que joueur, je veux voir mes statistiques et badges	Profil personnel, historique des raids, système de gamification	4 jours
En tant qu'administrateur, je veux gérer le contenu des guides	Interface d'administration, édition des guides, validation	5 jours

Exemple**Schéma des rituels Scrum :**

Sprint Planning ----> Daily Standup ----> Sprint Review

|

|

|

|

|

|

v

v

v

Sprint Retrospective ----> Sprint ----> Sprint Demo

Colonnes du tableau Kanban :

- **Backlog** : Fonctionnalités à développer
- **To Do** : Tâches prêtes pour le sprint
- **In Progress** : Tâches en cours (WIP limit : 3)
- **Review** : Code en attente de validation
- **Done** : Fonctionnalités livrées

À FAIRE / À VÉRIFIER**Ce que le jury attend dans cette section :**

- Une justification claire du choix méthodologique
- Une description des rituels et de leur utilité
- Une adaptation de la méthode au contexte projet
- Des métriques de suivi et d'amélioration continue
- Une organisation claire des responsabilités

Conseils de rédaction :

- Expliquez pourquoi cette méthode convient à votre projet
- Montrez comment vous mesurez l'efficacité de votre approche
- Décrivez les adaptations nécessaires à votre contexte
- Utilisez des diagrammes pour illustrer vos processus

Contrôles Jury CDA

- Pourquoi avez-vous choisi cette méthode Agile ?
- Comment mesurez-vous l'efficacité de vos rituels ?
- Quels sont vos indicateurs de performance ?
- Comment gérez-vous les imprévus dans vos sprints ?
- Avez-vous adapté la méthode à votre contexte ?

3.2 Versioning GitHub et conventions

Le versioning GitHub suit le modèle Git Flow avec des branches spécialisées pour chaque type de développement. Les conventions de nommage et de commit facilitent la traçabilité et la collaboration. Les Pull Requests permettent la revue de code systématique et la validation des fonctionnalités avant intégration.

Les conventions établies couvrent le nommage des branches, le format des messages de commit, et les templates de Pull Request. Cette standardisation améliore la qualité du code et accélère l'onboarding de nouveaux développeurs.

Exemple**Schéma Git Flow :**

```

main -----
|
+-- develop -----
    |
    +-- feature/user-authentication
    +-- feature/project-management
    +-- hotfix/critical-bug-fix

```

Conventions de commit :

```

1 feat: add user authentication system
2 fix: resolve login validation issue
3 docs: update API documentation
4 test: add unit tests for user service
5 refactor: improve code structure

```

À FAIRE / À VÉRIFIER

- Définir des conventions de nommage claires
- Utiliser des messages de commit descriptifs
- Mettre en place des templates de Pull Request
- Configurer des règles de protection des branches
- Documenter les conventions dans un CONTRIBUTING.md

Contrôles Jury CDA

- Quelles sont vos conventions de versioning ?
- Comment gérez-vous les conflits de merge ?
- Vos Pull Requests sont-elles systématiquement revues ?
- Comment assurez-vous la qualité du code ?
- Avez-vous des règles de protection des branches ?

3.3 Planification et outils de suivi

La planification combine une roadmap GitHub pour la vision macro et GitHub Projects pour le suivi opérationnel. La roadmap GitHub visualise les dépendances et les jalons critiques, tandis que le Kanban GitHub Projects offre une vue détaillée des tâches en cours. Cette approche dual optimise la coordination entre la planification stratégique et l'exécution tactique.

La roadmap GitHub permet de communiquer la vision produit et les priorités à long terme. Les milestones et les dépendances facilitent la coordination entre les différentes équipes et la gestion des risques de planning.

Exemple**Extrait de roadmap GitHub :**

Phase 1: Conception (4 semaines)

- +-- Analyse des besoins (1 semaine)
- +-- Conception technique (2 semaines)
- +-- Validation architecture (1 semaine)

Phase 2: Développement MVP (8 semaines)

- +-- Backend API (4 semaines)
- +-- Frontend React (4 semaines)
- +-- Tests intégration (2 semaines)

Configuration GitHub Projects :

- **Colonnes** : Backlog, To Do, In Progress, Review, Done
- **WIP Limits** : 3 tâches max en cours par développeur
- **Policies** : PR obligatoire pour merge en develop
- **Automation** : Mise à jour automatique des statuts

Focus GitHub**Git Flow et conventions :**

- **Branches** : main, develop, feature/*, release/*, hotfix/*
- **PR Template** : Description, tests, checklist
- **CODEOWNERS** : Validation obligatoire par senior dev
- **Protection Rules** : Pas de push direct sur main/develop

GitHub Projects Kanban :

- **Colonnes** : Backlog, Sprint Planning, In Progress, Review, Done
- **WIP Limits** : 2 features max en développement
- **Automation** : Mise à jour statut via labels
- **Metrics** : Cycle time, lead time, throughput

Roadmap et milestones :

- **Milestones** : v1.0 (Q2), v1.1 (Q3), v2.0 (Q4)
- **Dependencies** : Backend → Frontend → Tests
- **Risks** : Intégration externe, performance
- **Success Metrics** : Velocity, quality gates

À FAIRE / À VÉRIFIER

- Créer une roadmap GitHub réaliste avec marges
- Configurer GitHub Projects selon vos besoins
- Définir des milestones et jalons clairs
- Prévoir des buffers pour les imprévus
- Communiquer régulièrement sur l'avancement

Contrôles Jury CDA

- Votre planification est-elle réaliste ?
- Comment gérez-vous les retards ?
- Quels outils utilisez-vous pour le suivi ?
- Comment communiquez-vous l'avancement ?
- Avez-vous identifié les risques de planning ?

3.4 Estimation de temps et planification

Dans cette section, vous devez présenter votre estimation de temps pour chaque fonctionnalité et expliquer comment vous planifiez votre projet. Le jury attend une approche réaliste et méthodique de la gestion du temps.

Votre estimation globale : *Le projet est estimé à 65 jours de travail effectif répartis sur 8 mois (octobre 2025 à mai 2026), incluant 20% de marge pour les imprévus. Cette estimation couvre le développement, les tests, et le déploiement.*

L'analyse des temps permet de valider la faisabilité du projet et d'optimiser la planification selon les contraintes disponibles. Cette approche pragmatique démontre votre capacité à prendre en compte les contraintes temporelles dans les décisions techniques.

Exemple**Estimation de temps par fonctionnalité :**

Fonctionnalité	Description	Unité	Qty	Temps estimé	Total
Authentification	login/Register	jours	3	2	6
Gestion projets	CRUD projets	jours	5	3	15
Tableaux de bord	Dashboard	jours	2	4	8
Tests	Tests unitaires	jours	10	1	10
Déploiement	CI/CD	jours	3	2	6
Total estimé					45 jours

Votre estimation :

Fonctionnalité	Phase	Jours estimés	Milestone
Environnement	Setup	5	Octobre
Authentification Bungie	Backend	5	Novembre
Base de données	Backend	4	Novembre
API Escouades	Backend	6	Décembre
Guides interactifs	Frontend	8	Janvier
Calendrier raids	Frontend	7	Février
Profils joueurs	Frontend	4	Février
API Destiny 2	Intégration	6	Mars
Recommandations équipements	Intégration	4	Mars
Tests unitaires	Qualité	5	Avril
Tests E2E	Qualité	4	Avril
Dockerisation	Déploiement	3	Mai
CI/CD	Déploiement	4	Mai
Documentation	Livraison	4	Mai
Total		65 jours	
Avec marge 20%		78 jours	

À FAIRE / À VÉRIFIER**Ce que le jury attend dans cette section :**

- Une estimation de temps réaliste et justifiée
- Une répartition claire par fonctionnalité
- Une prise en compte des phases de test et déploiement
- Une marge de sécurité pour les imprévus
- Un lien avec les user stories et milestones GitHub

Conseils de rédaction :

- Basez-vous sur votre expérience et la complexité technique
- Ajoutez 20% de marge pour les imprévus
- Décomposez les grosses fonctionnalités en sous-tâches
- Justifiez vos estimations par des arguments techniques
- Montrez la cohérence avec votre roadmap GitHub

Contrôles Jury CDA**Questions de contrôle du jury :**

- Comment avez-vous estimé le temps pour chaque fonctionnalité ?
- Avez-vous pris en compte les phases de test et déploiement ?
- Comment gérez-vous les dépassements de temps ?
- Quelle marge de sécurité avez-vous prévue ?
- Comment liez-vous cette estimation à vos milestones GitHub ?
- Que faites-vous si une fonctionnalité prend plus de temps que prévu ?

3.5 Liens utiles

- GitHub Flow/PRs : <https://docs.github.com/pull-requests>
- Git Flow : <https://bit.ly/gitflow-atlassian>
- GitHub Projects : <https://bit.ly/github-projects>
- GitHub Roadmap : <https://bit.ly/github-roadmap>
- GitHub Milestones : <https://bit.ly/github-milestones>
- User Stories : <https://www.mountangoatsoftware.com/agile/user-stories>
- Estimation de temps : <https://bit.ly/time-estimation>

Chapitre 4

Conception fonctionnelle et technique

IMPORTANT : Cette phase de conception est **CRUCIALE** et doit être **COMPLÈTEMENT TERMINÉE** avant de commencer le développement. Le jury attend une conception solide et documentée qui justifie tous vos choix techniques.

Dans ce chapitre, vous devez présenter votre conception fonctionnelle et technique complète. Cette phase détermine la réussite de votre projet et doit être soigneusement planifiée et documentée.

Votre approche de conception : *[Décrivez votre méthodologie de conception et votre processus de validation]*

À FAIRE / À VÉRIFIER

Pourquoi la conception est-elle si importante ?

- **Évite les refactorisations coûteuses** : Une bonne conception évite de reprendre le code
- **Guide le développement** : Chaque développeur sait exactement quoi faire
- **Facilite les tests** : Les cas d'usage définis permettent de créer des tests pertinents
- **Réduit les risques** : Les problèmes sont identifiés avant le développement
- **Améliore la communication** : Tous les acteurs comprennent le système

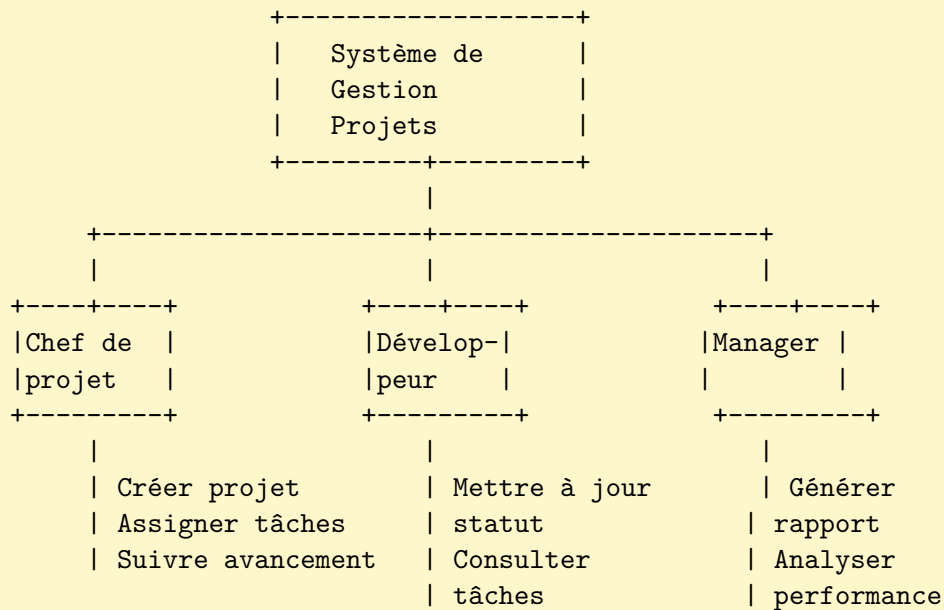
Checklist de conception complète :

- ✓ Diagrammes de cas d'usage (Use Cases) validés
- ✓ Diagrammes de séquence pour les flux principaux
- ✓ Modèle de données (MCD/MLD/MPD) défini
- ✓ Architecture technique choisie et justifiée
- ✓ User stories détaillées avec critères d'acceptation
- ✓ Maquettes et wireframes validés
- ✓ Charte graphique définie
- ✓ Plan de tests établi

4.1 Use Cases et diagrammes UML

Les Use Cases modélisent les interactions entre les acteurs et le système pour identifier les fonctionnalités essentielles. Cette approche centrée utilisateur garantit que le système répond aux besoins métier réels. Les diagrammes UML facilitent la communication entre les équipes techniques et métier, réduisant les risques d'incompréhension.

La modélisation des cas d'usage permet d'identifier les flux principaux et alternatifs, ainsi que les cas d'erreur à gérer. Cette analyse préalable guide la conception technique et les tests d'acceptation.

Exemple**Diagramme Use Case simplifié :****À FAIRE / À VÉRIFIER**

- Identifier tous les acteurs du système
- Modéliser les cas d'usage principaux et alternatifs
- Documenter les préconditions et postconditions
- Prévoir les cas d'erreur et exceptions
- Valider les Use Cases avec les utilisateurs métier

Contrôles Jury CDA

- Quels sont vos acteurs principaux ?
- Avez-vous modélisé tous les cas d'usage critiques ?
- Comment gérez-vous les cas d'erreur ?
- Vos Use Cases sont-ils validés par les utilisateurs ?
- Avez-vous prévu les flux alternatifs ?

4.2 Diagrammes de séquence

Les diagrammes de séquence détaillent les interactions temporelles entre les différents composants du système pour chaque cas d'usage. Cette modélisation précise les responsabilités de chaque couche (présentation, logique métier, données) et facilite l'implémentation technique. Les diagrammes servent également de référence pour les tests d'intégration.

La modélisation des séquences permet d'identifier les points de synchronisation, les appels asynchrones, et les mécanismes de gestion d'erreur. Cette analyse technique guide l'architecture et l'implémentation des APIs.

Exemple**Diagramme de séquence - Création de projet :**

User	Frontend	Backend	Database
	--POST-->		
		--POST-->	
			--INSERT->
			<--OK-----
		<--201-----	
	<--200-----		

Exemple de séquence avec gestion d'erreur :

User	Frontend	Backend	Database
	--POST-->		
		--POST-->	
			--INSERT->
			<--ERROR--
		<--400-----	
	<--400-----		

À FAIRE / À VÉRIFIER

- Modéliser les séquences pour chaque cas d'usage critique
- Prévoir la gestion des erreurs et exceptions
- Identifier les appels synchrones et asynchrones
- Documenter les timeouts et retry policies
- Valider les séquences avec l'équipe technique

Contrôles Jury CDA

- Avez-vous modélisé les séquences critiques ?
- Comment gérez-vous les erreurs dans vos séquences ?
- Vos diagrammes sont-ils cohérents avec l'architecture ?
- Avez-vous prévu les cas de timeout ?
- Comment validez-vous vos modèles de séquence ?

4.3 Conception de l'interface graphique

La conception graphique s'appuie sur une charte graphique cohérente avec l'identité visuelle de l'entreprise. Le zoning et les wireframes définissent la structure des interfaces avant le développement des maquettes haute fidélité. Cette approche progressive valide les choix UX et facilite l'implémentation front-end.

L'expérience utilisateur (UX) privilégie la simplicité et l'efficacité pour réduire la courbe d'apprentissage et améliorer l'adoption. Les tests utilisateur permettent de valider les choix de conception et d'optimiser l'interface.

4.3.1 Zoning

Dans cette sous-section, vous devez présenter l'organisation spatiale de vos interfaces. Le jury attend une analyse claire de la hiérarchie visuelle et de l'organisation des éléments.

Votre zoning : *[Décrivez l'organisation spatiale de vos pages principales]*

Exemple**Zoning d'une page projet :**

Header - Navigation principale		
Sidebar	Contenu principal	Panel latéral
<ul style="list-style-type: none"> • Menu navigation • Filtres • Recherche • Paramètres 	<ul style="list-style-type: none"> • Titre du projet • Liste des tâches • Tableau de données • Pagination 	<ul style="list-style-type: none"> • Actions rapides • Statistiques • Notifications • Aide contextuelle
Footer - Informations légales		

4.3.2 Wireframe

Dans cette sous-section, vous devez présenter vos wireframes pour les pages principales. Le jury attend une représentation claire de la structure et des interactions.

Vos wireframes : *[Présentez vos wireframes pour les pages principales]*

Exemple**Exemple de wireframe - Page de connexion :**

<p align="center">LOGO DE L'APPLICATION</p> <p align="center">Connexion</p> <p>Email : [_____]</p> <p>Mot de passe : [_____]</p> <p align="center">[Se connecter]</p> <p align="center">Mot de passe oublié ?</p>

4.3.3 Maquettage

Dans cette sous-section, vous devez présenter vos maquettes haute fidélité. Le jury attend une représentation visuelle fidèle au rendu final.

Vos maquettes : *[Décrivez vos maquettes haute fidélité et leur évolution]*

Exemple**Évolution des maquettes :**

Version 1	Version 2	Version 3	Final
Maquettes basiques Placeholders Structure simple	Intégration charte Couleurs définies Typographie	Maquettes interactives Animations Micro-interactions	Maquettes validées Tests utilisateurs Optimisations UX

4.3.4 Outils de conception et diagrammes

Dans cette sous-section, vous devez présenter les outils que vous utilisez pour créer vos diagrammes de qualité professionnelle. Le jury attend des diagrammes clairs et bien conçus qui facilitent la compréhension de votre architecture.

Vos outils de diagrammes : *[Listez les outils que vous utilisez et justifiez vos choix]*

À FAIRE / À VÉRIFIER**Outils recommandés pour des diagrammes de qualité :**

- **Draw.io (diagrams.net)** : Gratuit, intégré à GitHub, parfait pour les diagrammes UML
- **Lucidchart** : Professionnel, templates UML, collaboration en équipe
- **PlantUML** : Code-based, versioning Git, intégration LaTeX
- **Mermaid** : Intégré GitHub, syntaxe simple, diagrammes de flux

Conseils pour des diagrammes professionnels :

- Utilisez des couleurs cohérentes et une légende
- Respectez les conventions UML (acteurs, cas d'usage, relations)
- Gardez vos diagrammes simples et lisibles
- Versionnez vos diagrammes avec votre code
- Intégrez-les dans votre documentation GitHub

4.3.5 Charte graphique

Dans cette sous-section, vous devez détailler votre charte graphique complète. Le jury attend une cohérence visuelle et une identité forte.

Couleurs

Votre palette de couleurs : *[Définissez votre palette avec les codes hexadécimaux]*

Typographie

Votre système typographique : *[Définissez vos polices et leurs usages]*

Logo

Votre logo et son utilisation : *[Décrivez votre logo et ses variantes]*

Exemple**Charte graphique :**

Couleurs	Typographie	Composants
Primaire : #FFD700	Inter (titres)	Boutons arrondis
Secondaire : #101820	Arial (corps)	Cartes avec ombres
Neutre : #333A40	Monospace (code)	Icônes Material Design
Accent : #007BFF		
Espacement	Grille 8px	Marges cohérentes

À FAIRE / À VÉRIFIER

- Définir une charte graphique cohérente
- Créer des wireframes pour toutes les pages principales
- Développer des maquettes haute fidélité
- Tester l'accessibilité et la responsivité
- Utiliser Lighthouse pour valider les performances et l'accessibilité
- Valider les choix UX avec les utilisateurs

Contrôles Jury CDA

- Votre charte graphique est-elle cohérente ?
- Avez-vous testé vos interfaces avec les utilisateurs ?
- Vos maquettes respectent-elles l'accessibilité ?
- Quels sont vos scores Lighthouse pour l'accessibilité ?
- Comment gérez-vous la responsivité ?
- Avez-vous défini des composants réutilisables ?

4.4 Conception de base de données

La conception de base de données suit la méthode Merise avec un Modèle Conceptuel de Données (MCD), un Modèle Logique de Données (MLD), et un Modèle Physique de Données (MPD). Cette approche progressive garantit la cohérence et l'optimisation des données. Les contraintes d'intégrité et les index optimisent les performances et la fiabilité.

PostgreSQL gère les données transactionnelles avec des contraintes strictes, tandis que MongoDB stocke les logs et rapports avec une structure flexible. Cette architecture hybride optimise les performances selon le type de données.

4.4.1 MCD (Modèle Conceptuel de Données)

Dans cette sous-section, vous devez présenter votre modèle conceptuel de données. Le jury attend une représentation claire des entités et de leurs relations.

Votre MCD : *[Présentez votre modèle conceptuel avec les entités et relations principales]*

4.4.2 MLD (Modèle Logique de Données)

Dans cette sous-section, vous devez détailler votre modèle logique de données. Le jury attend une traduction du MCD en structure de base de données.

Votre MLD : *[Décrivez votre modèle logique avec les tables et relations]*

4.4.3 MPD (Modèle Physique de Données)

Dans cette sous-section, vous devez présenter votre modèle physique de données. Le jury attend une implémentation concrète avec les contraintes et index.

Votre MPD : *[Détaillez votre modèle physique avec les contraintes, index et optimisations]*

Exemple**MCD simplifié :**

PROJET (id, nom, description, date_debut, date_fin)

|
| 1,n

TACHE (id, titre, description, statut, priorite)

|
| n,1
|

UTILISATEUR (id, email, nom, prenom, role)

Exemple de contraintes PostgreSQL :

```

1  -- Contraintes d'intégrité
2  ALTER TABLE taches ADD CONSTRAINT fk_tache_projet
3      FOREIGN KEY (projet_id) REFERENCES projets(id)
4      ON DELETE CASCADE;
5
6  -- Index pour optimiser les performances
7  CREATE INDEX idx_taches_statut ON taches(statut);
8  CREATE INDEX idx_taches_projet_statut ON taches(projet_id, statut);
9
10 -- Contrainte de validation
11 ALTER TABLE projets ADD CONSTRAINT chk_dates
12     CHECK (date_fin > date_debut);

```

Exemple de document MongoDB :

```

1  {
2      "_id": ObjectId("..."),
3      "userId": "user123",
4      "action": "task_created",
5      "timestamp": ISODate("2025-01-15T10:30:00Z"),
6      "metadata": {
7          "projectId": "proj456",
8          "taskId": "task789",
9          "ipAddress": "192.168.1.100"
10     }
11 }

```

À FAIRE / À VÉRIFIER

- Modéliser le MCD avec toutes les entités et relations
- Définir les contraintes d'intégrité référentielle
- Optimiser avec des index appropriés
- Prévoir la migration et l'évolution du schéma
- Documenter les choix de conception

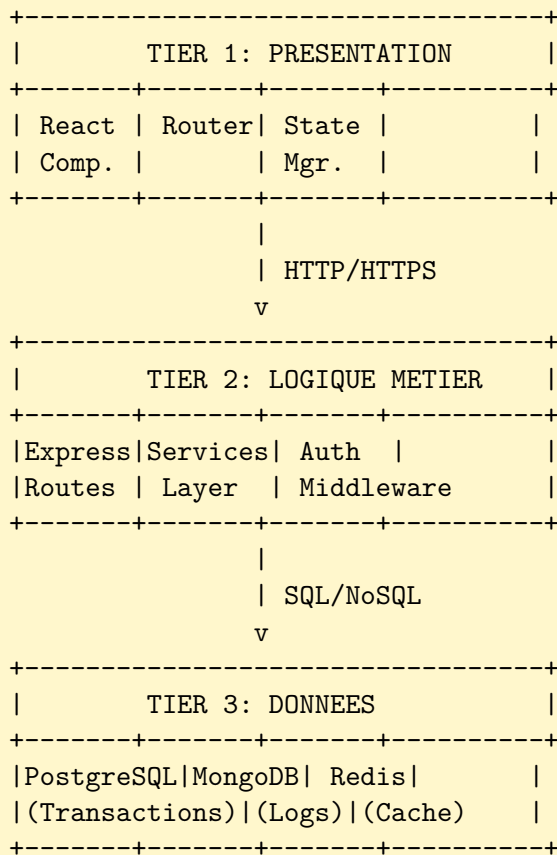
Contrôles Jury CDA

- Montrez votre MCD et expliquez 2 contraintes d'intégrité
- Comment optimisez-vous les performances de vos requêtes ?
- Avez-vous prévu la migration des données ?
- Pourquoi utiliser PostgreSQL ET MongoDB ?
- Comment gérez-vous la cohérence entre les deux bases ?

4.5 Architecture 3 tiers

L'architecture 3 tiers sépare clairement les responsabilités : couche présentation (React), couche logique métier (Node.js), et couche données (PostgreSQL/MongoDB). Cette séparation facilite la maintenance, la scalabilité et les tests. Chaque tier peut évoluer indépendamment selon les besoins techniques et métier.

Les flux de données sont optimisés pour minimiser les appels réseau et garantir la cohérence transactionnelle. L'API REST assure une communication standardisée entre les couches et facilite l'intégration avec d'autres systèmes.

Exemple**Schéma architecture 3 tiers :****Exemple de flux de données :**

```

1 // Tier 1: Frontend (React)
2 const createProject = async (projectData) => {
3   const response = await fetch('/api/projects', {
4     method: 'POST',
5     headers: { 'Content-Type': 'application/json' },
6     body: JSON.stringify(projectData)
7   });
8   return response.json();
9 };
10
11 // Tier 2: Backend (Node.js/Express)
12 app.post('/api/projects', authenticateUser, async (req, res) => {
13   try {
14     const project = await projectService.createProject(req.body);
15     await auditService.logAction('project_created', req.user.id);
16     res.status(201).json(project);
17   } catch (error) {
18     res.status(400).json({ error: error.message });
19   }
20 });

```

À FAIRE / À VÉRIFIER

- Documenter clairement les responsabilités de chaque tier
- Définir les interfaces entre les couches
- Prévoir la scalabilité horizontale et verticale
- Implémenter des mécanismes de cache appropriés
- Tester l'intégration entre les tiers

Contrôles Jury CDA

- Quelles sont les responsabilités de chaque tier ?
- Comment gérez-vous la communication entre les tiers ?
- Votre architecture est-elle scalable ?
- Avez-vous prévu la gestion des erreurs inter-tiers ?
- Comment optimisez-vous les performances ?

4.6 Liens utiles

- UML : <https://www.uml-diagrams.org/>
- Merise (FR) : <https://perso.liris.cnrs.fr/pierre-antoine.champin/enseignement/intro-merise.html>
- OWASP ASVS : <https://owasp.org/ASVS/>
- PostgreSQL Docs : <https://www.postgresql.org/docs/>
- MongoDB Modeling : <https://bit.ly/mongodb-modeling>
- Draw.io (diagrammes) : <https://app.diagrams.net/>
- Lighthouse (accessibilité) : <https://developers.google.com/web/tools/lighthouse>
- Lucidchart (UML) : <https://www.lucidchart.com/pages/fr/exemples/diagramme-uml>
- PlantUML (diagrammes) : <https://plantuml.com/>
- Mermaid (diagrammes) : <https://mermaid-js.github.io/mermaid/>

Chapitre 5

Architecture 3 tiers

5.1 Architecture 3 tiers

Dans cette section, vous devez présenter votre architecture 3 tiers et expliquer la répartition des responsabilités entre les couches. Le jury attend une compréhension claire de la séparation physique des composants.

Votre architecture 3 tiers : *[Décrivez votre architecture et la répartition des responsabilités]*

5.1.1 Couche Présentation (Frontend)

Dans cette sous-section, vous devez détailler la couche présentation de votre application. Le jury attend une explication claire des technologies et de l'organisation du code.

Votre couche présentation : *[Décrivez votre stack frontend et son organisation]*

Exemple

Technologies de présentation :

- **Framework** : React 18 avec hooks et context
- **État** : Redux Toolkit pour la gestion d'état globale
- **Routing** : React Router pour la navigation
- **UI** : Material-UI pour les composants
- **HTTP** : Axios pour les appels API

Structure des composants :

```
src/
+-- components/                # Composants réutilisables
|   +-- common/
|   |   +-- Button.tsx
|   |   +-- Modal.tsx
|   |   +-- LoadingSpinner.tsx
|   +-- projects/              # Fonctionnalité projets
|   |   +-- ProjectList.tsx
|   |   +-- ProjectCard.tsx
|   |   +-- ProjectForm.tsx
|   +-- tasks/                 # Fonctionnalité tâches
|       +-- TaskList.tsx
|       +-- TaskItem.tsx
+-- hooks/                     # Hooks personnalisés
+-- services/                  # Appels API
+-- utils/                     # Fonctions utilitaires
```

5.1.2 Couche Logique Métier (Backend)

Dans cette sous-section, vous devez présenter la couche logique métier de votre application. Le jury attend une explication de l'architecture et de l'organisation du code.

Votre couche logique métier : *[Décrivez votre stack backend et son organisation]*

Controller

Vos contrôleurs : *[Décrivez vos contrôleurs et leur rôle]*

Exemple**Exemple de contrôleur :**

```

1 // ProjectController.js
2 class ProjectController {
3   async createProject(req, res) {
4     try {
5       const projectData = req.body;
6       const project = await this.projectService.create(projectData);
7       res.status(201).json(project);
8     } catch (error) {
9       res.status(400).json({ error: error.message });
10    }
11  }
12 }

```

Service

Vos services : *[Décrivez vos services et la logique métier]*

Exemple**Exemple de service :**

```

1 // ProjectService.js
2 class ProjectService {
3   async create(projectData) {
4     // Validation des données
5     this.validateProjectData(projectData);
6
7     // Logique métier
8     const project = await this.projectRepository.create(projectData);
9
10    // Notifications
11    await this.notificationService.notifyTeam(project);
12
13    return project;
14  }
15 }

```

Repository (DAO)

Vos repositories : *[Décrivez vos repositories et l'accès aux données]*

Exemple**Exemple de repository :**

```

1 // ProjectRepository.js
2 class ProjectRepository {
3   async create(projectData) {
4     const query = 'INSERT INTO projects (name, description) VALUES ($1, $2) RETURNING *';
5     const values = [projectData.name, projectData.description];
6     const result = await this.db.query(query, values);
7     return result.rows[0];
8   }
9 }

```

5.1.3 Couche Données (Database)

Dans cette sous-section, vous devez présenter la couche de données de votre application. Le jury attend une explication de l'architecture des données et des choix techniques.

Votre couche données : *[Décrivez votre architecture de données]*

Exemple

Architecture des données :

- **PostgreSQL** : Données transactionnelles et relations
- **MongoDB** : Logs, rapports et données non-structurées
- **Redis** : Cache et sessions utilisateur
- **ORM** : Prisma pour PostgreSQL, Mongoose pour MongoDB

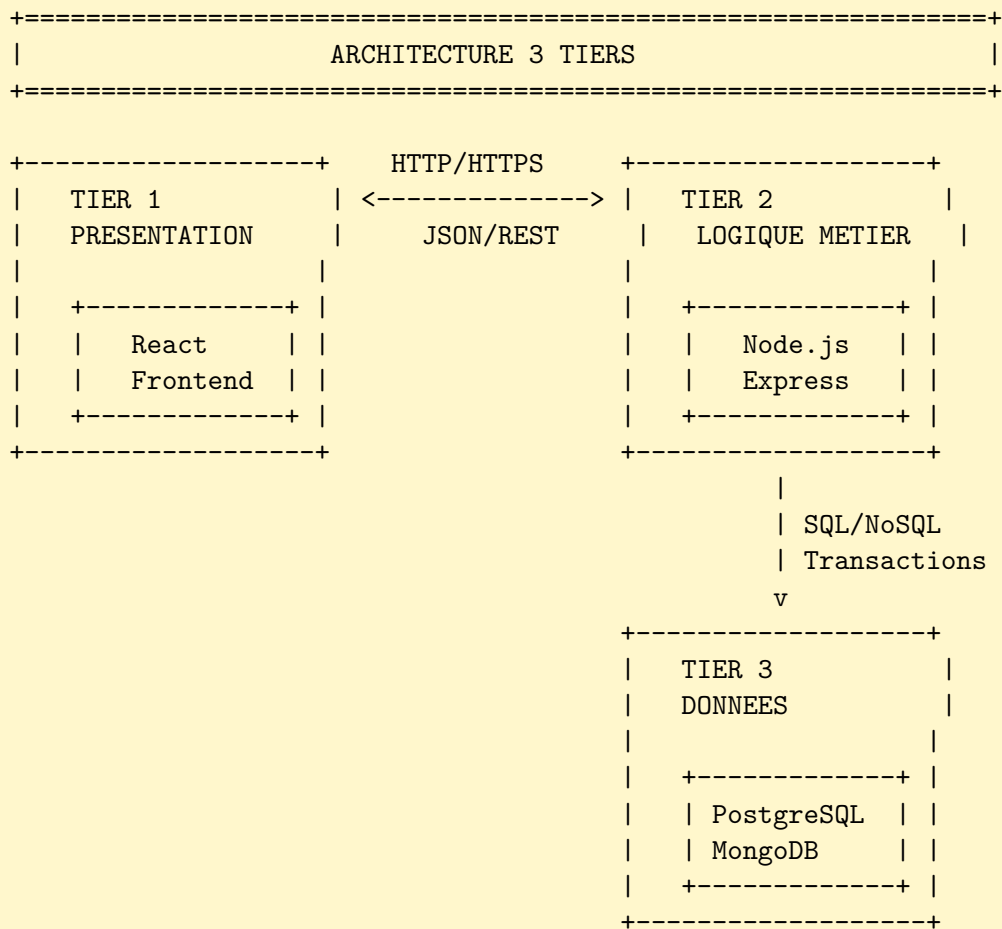
Exemple de repository PostgreSQL :

```
1 class ProjectRepository {
2   async create(projectData) {
3     return await prisma.project.create({
4       data: {
5         name: projectData.name,
6         description: projectData.description,
7         userId: projectData.userId
8       },
9       include: {
10        tasks: true,
11        user: { select: { id: true, email: true } }
12      }
13    });
14  }
15 }
```

5.1.4 Communication entre les tiers

Dans cette sous-section, vous devez expliquer comment les différents tiers communiquent entre eux. Le jury attend une compréhension claire des flux de données et des protocoles utilisés.

Vos flux de communication : *[Décrivez comment vos tiers communiquent]*

Exemple**Flux de communication 3 tiers :****5.1.5 Avantages de l'architecture 3 tiers**

Dans cette sous-section, vous devez expliquer les avantages de votre architecture 3 tiers. Le jury attend une justification claire des choix architecturaux.

Avantages de votre architecture : *[Justifiez les bénéfices de votre approche]*

Exemple**Avantages de l'architecture 3 tiers :**

- **Séparation des responsabilités** : Chaque tier a un rôle défini
- **Scalabilité** : Possibilité de scaler chaque tier indépendamment
- **Maintenabilité** : Modifications isolées par tier
- **Tests** : Tests unitaires par tier facilités
- **Sécurité** : Contrôle d'accès par tier
- **Performance** : Optimisation possible par tier

À FAIRE / À VÉRIFIER

- Séparer clairement les responsabilités par tier
- Documenter les interfaces entre les tiers
- Prévoir la scalabilité de chaque tier
- Implémenter des tests par tier
- Gérer les erreurs et exceptions de manière cohérente
- Optimiser les performances par tier

Contrôles Jury CDA

- Comment justifiez-vous votre architecture 3 tiers ?
- Quels sont les avantages de votre approche ?
- Comment gérez-vous la communication entre les tiers ?
- Votre architecture est-elle scalable ?
- Comment testez-vous chaque tier ?
- Quels sont les points de performance de votre architecture ?

5.2 Développement Frontend

Dans cette section, vous devez présenter votre approche de développement frontend et expliquer vos choix techniques. Le jury attend une compréhension claire de votre architecture frontend et des bonnes pratiques appliquées.

Technologies choisies : *[React, Vue, Angular, ou autre ? Justifiez votre choix]*

Architecture des composants : *[Décrivez votre organisation des composants et leur réutilisabilité]*

Accessibilité et UX : *[Expliquez comment vous respectez les standards RGAA/WCAG et utilisez Lighthouse pour mesurer les performances]*

Exemple**Structure des composants :**

```
src/
+-- components/                # Composants réutilisables
|   +-- common/
|   |   +-- Button.tsx
|   |   +-- Modal.tsx
|   |   +-- LoadingSpinner.tsx
|   +-- projects/              # Fonctionnalité projets
|   |   +-- ProjectList.tsx
|   |   +-- ProjectCard.tsx
|   |   +-- ProjectForm.tsx
|   +-- tasks/                 # Fonctionnalité tâches
|       +-- TaskList.tsx
|       +-- TaskItem.tsx
+-- hooks/                     # Hooks personnalisés
+-- services/                  # Appels API
+-- utils/                     # Fonctions utilitaires
```

Exemple de composant accessible :

```
1 const ProjectCard = ({ project, onEdit }) => {
2   return (
3     <div
4       className="project-card"
```

```

5     role="article"
6     aria-labelledby={`project-${project.id}-title`}
7   >
8     <h3 id={`project-${project.id}-title`} >
9       {project.name}
10    </h3>
11    <p>{project.description}</p>
12    <button
13      onClick={() => onEdit(project.id)}
14      aria-label={`Modifier le projet ${project.name}`}
15    >
16      Modifier
17    </button>
18  </div>
19  );
20 };

```

Exemple

Exemple de rapport Lighthouse (1/2) :

```

1  {
2    "categories": {
3      "performance": { "score": 0.92 },
4      "accessibility": { "score": 0.95 },
5      "best-practices": { "score": 0.88 },
6      "seo": { "score": 0.90 }
7    }
8  }

```

Exemple

Exemple de rapport Lighthouse (2/2) :

```

1  "audits": {
2    "first-contentful-paint": { "score": 0.95 },
3    "largest-contentful-paint": { "score": 0.88 },
4    "color-contrast": { "score": 1.0 },
5    "aria-allowed-attr": { "score": 1.0 }
6  }
7  }

```

À FAIRE / À VÉRIFIER

- Organiser les composants par fonctionnalité métier
- Respecter les standards d'accessibilité RGAA/WCAG
- Utiliser Lighthouse pour mesurer les performances et l'accessibilité
- Implémenter la protection XSS avec React
- Utiliser TypeScript pour la sécurité des types
- Tester les composants avec Jest et React Testing Library

Contrôles Jury CDA

- Comment organisez-vous votre code frontend ?
- Vos composants respectent-ils l'accessibilité ?
- Quels sont vos scores Lighthouse pour les performances et l'accessibilité ?
- Comment protégez-vous contre les attaques XSS ?
- Utilisez-vous TypeScript ? Pourquoi ?
- Comment testez-vous vos composants ?

5.3 Développement Backend

Le backend implémente une API REST avec Express.js suivant le pattern Controller/Service/Repository pour séparer les responsabilités. La validation des données utilise Joi ou Zod pour garantir la cohérence des entrées. La gestion d'erreur centralisée assure des réponses API cohérentes et facilite le debugging.

L'authentification JWT sécurise les endpoints sensibles avec des middlewares de vérification. La documentation OpenAPI/Swagger facilite l'intégration frontend et la maintenance de l'API.

Exemple**Structure backend :**

```

src/
+-- controllers/                # Gestion des requêtes HTTP
|   +-- projectController.js
|   +-- taskController.js
+-- services/                  # Logique métier
|   +-- projectService.js
|   +-- taskService.js
+-- repositories/              # Accès aux données
|   +-- projectRepository.js
|   +-- taskRepository.js
+-- middleware/                # Middlewares Express
|   +-- auth.js
|   +-- validation.js
|   +-- errorHandler.js
+-- routes/                    # Définition des routes
    +-- projects.js
    +-- tasks.js

```

Exemple de contrôleur avec validation :

```

1  const createProject = async (req, res, next) => {
2    try {
3      // Validation des données
4      const { error, value } = projectSchema.validate(req.body);
5      if (error) {
6        return res.status(400).json({
7          error: 'Données invalides',
8          details: error.details
9        });
10     }
11
12     // Appel du service métier
13     const project = await projectService.createProject(value, req.user.id
14       );
15
16     // Log de l'action
17     await auditService.logAction('project_created', req.user.id, {
18       projectId: project.id
19     });
20
21     res.status(201).json(project);
22   } catch (error) {
23     next(error);
24   }
25 };

```

À FAIRE / À VÉRIFIER

- Séparer clairement les responsabilités (Controller/Service/Repository)
- Valider systématiquement les données d'entrée
- Implémenter une gestion d'erreur centralisée
- Documenter l'API avec OpenAPI/Swagger
- Logger toutes les actions importantes

Contrôles Jury CDA

- Comment structurez-vous votre API REST ?
- Quels middlewares utilisez-vous ?
- Comment gérez-vous la validation des données ?
- Avez-vous documenté votre API ?
- Comment tracez-vous les erreurs ?

5.4 Gestion des données

La couche données utilise un ORM (Prisma) pour PostgreSQL et le driver natif pour MongoDB. Les transactions garantissent la cohérence des données critiques, tandis que les requêtes optimisées avec des index améliorent les performances. Le pattern Repository abstrait l'accès aux données et facilite les tests.

PostgreSQL gère les données transactionnelles avec des contraintes strictes, MongoDB stocke les logs et rapports avec des pipelines d'agrégation pour les analytics. Cette séparation optimise les performances selon le type d'opération.

Exemple**Exemple de repository PostgreSQL :**

```
1 class ProjectRepository {
2   async create(projectData) {
3     return await prisma.project.create({
4       data: {
5         name: projectData.name,
6         description: projectData.description,
7         userId: projectData.userId
8       },
9       include: {
10        tasks: true,
11        user: { select: { id: true, email: true } }
12      }
13    });
14  }
15
16  async findByUser(userId) {
17    return await prisma.project.findMany({
18      where: { userId },
19      include: { tasks: true }
20    });
21  }
22 }
```

Exemple de pipeline MongoDB :

```
1 // Pipeline d'agrégation pour les statistiques
2 const getProjectStats = async (projectId, dateRange) => {
3   return await activityLogs.aggregate([
4     {
5       $match: {
6         'metadata.projectId': projectId,
7         timestamp: { $gte: dateRange.start, $lte: dateRange.end }
8       }
9     },
10    {
11      $group: {
12        _id: '$action',
13        count: { $sum: 1 },
14        uniqueUsers: { $addToSet: '$userId' }
15      }
16    },
17    {
18      $project: {
19        action: '$_id',
20        count: 1,
21        uniqueUsersCount: { $size: '$uniqueUsers' }
22      }
23    }
24  ]);
25 };
```

À FAIRE / À VÉRIFIER

- Utiliser un ORM pour simplifier les requêtes SQL
- Optimiser les requêtes avec des index appropriés
- Implémenter des transactions pour la cohérence
- Séparer les données transactionnelles et analytiques
- Tester les requêtes avec des données réalistes

Contrôles Jury CDA

- Comment gérez-vous les transactions ?
- Avez-vous optimisé vos requêtes avec des index ?
- Pourquoi utiliser Prisma plutôt que du SQL brut ?
- Comment gérez-vous la cohérence entre PostgreSQL et MongoDB ?
- Avez-vous testé les performances de vos requêtes ?

5.5 Liens utiles

- OpenAPI/Swagger : <https://swagger.io/specification/>
- WCAG : <https://www.w3.org/WAI/standards-guidelines/wcag/>
- Lighthouse : <https://developers.google.com/web/tools/lighthouse>
- PostgreSQL Tutorial : <https://www.postgresql.org/docs/current/tutorial.html>
- MongoDB Aggregation : <https://www.mongodb.com/docs/manual/aggregation/>
- Prisma Documentation : <https://www.prisma.io/docs/>

Chapitre 6

Sécurité applicative et RGPD

6.1 Protection contre les vulnérabilités OWASP

La sécurité applicative s'appuie sur les recommandations OWASP Top 10 pour protéger contre les vulnérabilités courantes. La protection XSS utilise l'échappement automatique de React et la validation côté serveur. La prévention SQL injection repose sur les requêtes paramétrées de l'ORM Prisma. La protection CSRF implémente des tokens synchronisés et la validation des origines.

Les headers de sécurité (CSP, HSTS, X-Frame-Options) renforcent la protection au niveau HTTP. La validation stricte des entrées utilisateur et la sanitisation des données réduisent les risques d'injection et de manipulation.

Exemple**Middleware de sécurité Express :**

```

1  const helmet = require('helmet');
2  const rateLimit = require('express-rate-limit');
3
4  // Configuration Helmet
5  app.use(helmet({
6    contentSecurityPolicy: {
7      directives: {
8        defaultSrc: ['self'],
9        styleSrc: ['self', 'unsafe-inline'],
10       scriptSrc: ['self']
11     }
12   });
13
14
15  // Rate limiting
16  const limiter = rateLimit({
17    windowMs: 15 * 60 * 1000, // 15 min
18    max: 100, // 100 req/IP
19    message: 'Trop de requêtes'
20  });
21  app.use('/api/', limiter);
22
23  // Validation des entrées
24  const validateInput = (schema) => {
25    return (req, res, next) => {
26      const { error } = schema.validate(req.body);
27      if (error) {
28        return res.status(400).json({
29          error: 'Données invalides',
30          details: error.details.map(d => d.message)
31        });
32      }
33      next();
34    };
35  };

```

Protection XSS côté frontend :

```

1  // React échappe automatiquement les données
2  const UserProfile = ({ user }) => {
3    return (
4      <div>
5        <h1>{user.name}</h1> { /* Échappé automatiquement */ }
6        <p>{user.bio}</p>
7        { /* Danger : éviter dangerouslySetInnerHTML */ }
8      </div>
9    );
10 };
11
12 // Validation côté client avec Zod
13 const userSchema = z.object({
14   name: z.string().min(1).max(100),
15   email: z.string().email(),
16   bio: z.string().max(500).optional()
17 });

```

À FAIRE / À VÉRIFIER

- Implémenter les protections OWASP Top 10
- Configurer les headers de sécurité avec Helmet
- Utiliser le rate limiting pour prévenir les attaques DoS
- Valider et sanitiser toutes les entrées utilisateur
- Tester la sécurité avec des outils automatisés

Contrôles Jury CDA

- Quelles vulnérabilités OWASP avez-vous adressées ?
- Comment protégez-vous contre les attaques XSS ?
- Votre protection SQL injection est-elle efficace ?
- Avez-vous configuré les headers de sécurité ?
- Comment testez-vous la sécurité de votre application ?

6.2 Authentification et autorisation

L'authentification utilise JWT avec des tokens d'accès courts (15 minutes) et des refresh tokens sécurisés. Le hachage des mots de passe utilise Argon2, plus sécurisé que bcrypt. L'autorisation implémente un système de rôles et permissions granulaire avec des middlewares de vérification.

La gestion des sessions sécurise les tokens avec des cookies HttpOnly et SameSite. La déconnexion invalide les tokens côté serveur et côté client pour garantir la sécurité.

Exemple**Configuration JWT et Argon2 (1/3) :**

```
1 const jwt = require('jsonwebtoken');
2 const argon2 = require('argon2');
3
4 // Configuration JWT
5 const JWT_SECRET = process.env.JWT_SECRET;
6 const JWT_EXPIRES_IN = '15m';
7 const REFRESH_EXPIRES_IN = '7d';
8
9 // Hachage des mots de passe avec Argon2
10 const hashPassword = async (password) => {
11   return await argon2.hash(password, {
12     type: argon2.argon2id,
13     memoryCost: 2 ** 16, // 64 MB
14     timeCost: 3,
15     parallelism: 1
16   });
17 };
```

Exemple**Configuration JWT et Argon2 (2/3) :**

```
1 // Génération des tokens
2 const generateTokens = (userId, role) => {
3   const accessToken = jwt.sign(
4     { userId, role, type: 'access' },
5     JWT_SECRET,
6     { expiresIn: JWT_EXPIRES_IN }
7   );
8
9   const refreshToken = jwt.sign(
10    { userId, type: 'refresh' },
11    JWT_SECRET,
12    { expiresIn: REFRESH_EXPIRES_IN }
13  );
14
15  return { accessToken, refreshToken };
16 };
```

Exemple**Configuration JWT et Argon2 (3/3) :**

```

1 // Middleware d'authentification
2 const authenticateToken = (req, res, next) => {
3   const authHeader = req.headers['authorization'];
4   const token = authHeader && authHeader.split(' ')[1];
5
6   if (!token) {
7     return res.status(401).json({
8       error: 'Token d\'accès requis'
9     });
10  }
11
12  jwt.verify(token, JWT_SECRET, (err, user) => {
13    if (err) {
14      return res.status(403).json({
15        error: 'Token invalide'
16      });
17    }
18    req.user = user;
19    next();
20  });
21 };

```

Système de permissions (1/2) :

```

1 // Définition des permissions
2 const PERMISSIONS = {
3   PROJECT_CREATE: 'project:create',
4   PROJECT_READ: 'project:read',
5   PROJECT_UPDATE: 'project:update',
6   PROJECT_DELETE: 'project:delete',
7   USER_MANAGE: 'user:manage'
8 };
9
10 // Rôles et leurs permissions
11 const ROLES = {
12   ADMIN: [PERMISSIONS.PROJECT_CREATE, PERMISSIONS.PROJECT_READ,
13     PERMISSIONS.PROJECT_UPDATE, PERMISSIONS.PROJECT_DELETE,
14     PERMISSIONS.USER_MANAGE],
15   MANAGER: [PERMISSIONS.PROJECT_CREATE, PERMISSIONS.PROJECT_READ,
16     PERMISSIONS.PROJECT_UPDATE],
17   DEVELOPER: [PERMISSIONS.PROJECT_READ, PERMISSIONS.PROJECT_UPDATE]
18 };

```

Système de permissions (2/2) :

```

1 // Middleware d'autorisation
2 const requirePermission = (permission) => {
3   return (req, res, next) => {
4     const userPermissions = ROLES[req.user.role] || [];
5     if (!userPermissions.includes(permission)) {
6       return res.status(403).json({
7         error: 'Permissions insuffisantes'
8       });
9     }
10    next();
11  };
12 };

```

À FAIRE / À VÉRIFIER

- Utiliser JWT avec des tokens courts et refresh tokens
- Implémenter Argon2 pour le hachage des mots de passe
- Créer un système de rôles et permissions granulaire
- Sécuriser les cookies avec HttpOnly et SameSite
- Implémenter la déconnexion sécurisée

Contrôles Jury CDA

- Pourquoi utiliser JWT plutôt que des sessions ?
- Comment gérez-vous la sécurité des mots de passe ?
- Votre système d'autorisation est-il granulaire ?
- Comment gérez-vous l'expiration des tokens ?
- Avez-vous prévu la révocation des tokens ?

6.3 Conformité RGPD

La conformité RGPD implique la mise en place d'un registre des traitements, la minimisation des données collectées, et la sécurisation des données personnelles. Le consentement explicite est recueilli pour chaque traitement, avec la possibilité de retrait. Les droits des personnes (accès, rectification, effacement, portabilité) sont implémentés via des APIs dédiées.

La protection des données utilise le chiffrement au repos et en transit, avec des sauvegardes sécurisées. La notification des violations de données est automatisée pour respecter le délai de 72h.

Exemple**Registre des traitements :**

```

1 // Modèle de registre des traitements
2 const dataProcessingRegistry = {
3   'user-authentication': {
4     purpose: 'Authentification et gestion des comptes utilisateurs',
5     legalBasis: 'Consentement',
6     dataCategories: ['identité', 'connexion'],
7     retentionPeriod: '3 ans après fermeture du compte',
8     recipients: ['équipe technique', 'hébergeur'],
9     transfers: ['UE', 'États-Unis (clauses contractuelles)']
10  },
11  'project-management': {
12    purpose: 'Gestion des projets et collaboration',
13    legalBasis: 'Exécution du contrat',
14    dataCategories: ['travail', 'communication'],
15    retentionPeriod: '5 ans après fin du projet',
16    recipients: ['équipe projet', 'clients'],
17    transfers: ['UE uniquement']
18  }
19 };
20
21 // API pour les droits RGPD
22 const gdprController = {
23   // Droit d'accès
24   async getPersonalData(req, res) {
25     const userId = req.user.userId;
26     const userData = await userService.getCompleteUserData(userId);
27     res.json({
28       personalData: userData,
29       processingPurposes: Object.keys(dataProcessingRegistry),
30       retentionPeriods: dataProcessingRegistry
31     });
32   },
33
34   // Droit à l'effacement
35   async deletePersonalData(req, res) {
36     const userId = req.user.userId;
37     await userService.anonymizeUserData(userId);
38     await auditService.logAction('gdpr_deletion', userId);
39     res.json({ message: 'Données personnelles supprimées' });
40   },
41
42   // Droit à la portabilité
43   async exportPersonalData(req, res) {
44     const userId = req.user.userId;
45     const exportData = await userService.exportUserData(userId);
46     res.attachment('mes-donnees.json');
47     res.json(exportData);
48   }
49 };

```

Chiffrement des données sensibles (1/3) :

```

1 const crypto = require('crypto');
2
3 // Configuration du chiffrement
4 const ENCRYPTION_KEY = process.env.ENCRYPTION_KEY;
5 const ALGORITHM = 'aes-256-gcm';

```

Exemple**Chiffrement des données sensibles (2/3) :**

```
1 // Fonction de chiffrement
2 const encrypt = (text) => {
3   const iv = crypto.randomBytes(16);
4   const cipher = crypto.createCipher(ALGORITHM, ENCRYPTION_KEY);
5   cipher.setAAD(Buffer.from('user-data'));
6
7   let encrypted = cipher.update(text, 'utf8', 'hex');
8   encrypted += cipher.final('hex');
9
10  const authTag = cipher.getAuthTag();
11
12  return {
13    encrypted,
14    iv: iv.toString('hex'),
15    authTag: authTag.toString('hex')
16  };
17 };
```

Exemple**Chiffrement des données sensibles (3/3) :**

```
1 // Fonction de déchiffrement
2 const decrypt = (encryptedData) => {
3   const decipher = crypto.createDecipher(ALGORITHM, ENCRYPTION_KEY);
4   decipher.setAAD(Buffer.from('user-data'));
5   decipher.setAuthTag(Buffer.from(encryptedData.authTag, 'hex'));
6
7   let decrypted = decipher.update(encryptedData.encrypted, 'hex', 'utf8')
8   ;
9   decrypted += decipher.final('utf8');
10
11  return decrypted;
12 };
```

À FAIRE / À VÉRIFIER

- Créer un registre des traitements complet
- Implémenter les droits RGPD (accès, rectification, effacement)
- Chiffrer les données sensibles au repos et en transit
- Mettre en place la notification des violations
- Documenter les mesures de sécurité et de conformité

Contrôles Jury CDA

- Avez-vous établi un registre des traitements ?
- Comment implémentez-vous les droits RGPD ?
- Vos données sont-elles chiffrées ?
- Avez-vous prévu la notification des violations ?
- Comment gérez-vous le consentement des utilisateurs ?

6.4 Liens utiles

- OWASP Top 10 : <https://owasp.org/www-project-top-ten/>
- OWASP Cheat Sheets : <https://cheatsheetseries.owasp.org/>
- CNIL RGPD : <https://www.cnil.fr/fr/rgpd-de-quoi-parle-t-on>
- Argon2 : <https://github.com/P-H-C/phc-winner-argon2>
- JWT Best Practices : <https://tools.ietf.org/html/rfc8725>

Chapitre 7

Tests et qualité logicielle

7.1 Stratégie de tests

La stratégie de tests suit la pyramide de tests avec une base solide de tests unitaires, des tests d'intégration pour valider les interactions entre composants, et des tests end-to-end pour vérifier les parcours utilisateur complets. Cette approche garantit une couverture de code élevée tout en optimisant le temps d'exécution des tests.

Les tests de performance mesurent la latence P95 et le débit de l'application sous charge. Les tests de sécurité automatisés détectent les vulnérabilités communes. La qualité du code est surveillée avec SonarQube pour maintenir un niveau de qualité constant.

Exemple**Pyramide de tests :**

```

+=====+
|          TESTS E2E          |
|    (Cypress/Playwright)    |
|    Peu nombreux, lents    |
|    Couverture globale      |
+=====+
|
|
+=====+
|    TESTS D'INTEGRATION      |
|    (Jest/Supertest)         |
|    Nombre modere            |
|    Interactions composants  |
+=====+
|
|
+=====+
|    TESTS UNITAIRES          |
|    (Jest)                   |
|    Nombreux, rapides        |
|    Couverture detaillee     |
+=====+

```

Exemple**Exemple de test unitaire (1/2) :**

```

1 // Test unitaire pour le service de projet
2 describe('ProjectService', () => {
3   let projectService;
4   let mockRepository;
5
6   beforeEach(() => {
7     mockRepository = {
8       create: jest.fn(),
9       findById: jest.fn(),
10      update: jest.fn(),
11      delete: jest.fn()
12    };
13    projectService = new ProjectService(mockRepository);
14  });

```

Exemple**Exemple de test unitaire (2/2) :**

```

1   describe('createProject', () => {
2     it('should create a project with valid data', async () => {
3       // Arrange
4       const projectData = {
5         name: 'TestProject',
6         description: 'TestDescription',
7         userId: 'user123'
8       };
9       const expectedProject = { id: 'proj123', ...projectData };
10      mockRepository.create.mockResolvedValue(expectedProject);
11
12      // Act
13      const result = await projectService.createProject(projectData);
14
15      // Assert

```

À FAIRE / À VÉRIFIER

- Implémenter la pyramide de tests (unitaires, intégration, E2E)
- Maintenir une couverture de code élevée (>80%)
- Automatiser l'exécution des tests dans la CI/CD
- Tester les cas d'erreur et les cas limites
- Documenter les stratégies de test et les conventions

Contrôles Jury CDA

- Quelle est votre stratégie de tests ?
- Quelle est votre couverture de code ?
- Comment testez-vous les cas d'erreur ?
- Vos tests sont-ils automatisés ?
- Comment mesurez-vous la qualité de vos tests ?

7.2 Tests de performance

Les tests de performance utilisent k6 pour simuler des charges réalistes et mesurer les métriques clés : latence P95, débit, et taux d'erreur. Les scénarios de test couvrent les parcours utilisateur critiques et les pics de charge prévus. L'optimisation s'appuie sur l'analyse des goulots d'étranglement identifiés.

Le monitoring en production surveille les métriques de performance en temps réel avec des alertes automatiques. Les tests de charge réguliers valident la capacité de l'application à supporter la croissance du trafic.

Exemple**Script de test de performance k6 (1/2) :**

```

1 import http from 'k6/http';
2 import { check, sleep } from 'k6';
3 import { Rate } from 'k6/metrics';
4
5 // Métriques personnalisées
6 const errorRate = new Rate('errors');
7
8 export let options = {
9   stages: [
10     { duration: '2m', target: 10 }, // Montée en charge
11     { duration: '5m', target: 50 }, // Charge normale
12     { duration: '2m', target: 100 }, // Pic de charge
13     { duration: '5m', target: 50 }, // Retour à la normale
14     { duration: '2m', target: 0 }, // Descente
15   ],
16   thresholds: {
17     http_req_duration: ['p(95)<500'], // 95% des requêtes < 500ms
18     http_req_failed: ['rate<0.1'], // Moins de 10% d'erreurs
19     errors: ['rate<0.1']
20   }
21 };

```

Exemple**Script de test de performance k6 (2/2) :**

```

1 export default function() {
2   // Test de connexion
3   let loginResponse = http.post('http://localhost:3000/api/auth/login', {
4     email: 'test@example.com',
5     password: 'password123'
6   });
7
8   check(loginResponse, {
9     'login_status_is_200': (r) => r.status === 200,
10    'login_response_time<200ms': (r) => r.timings.duration < 200,
11  }) || errorRate.add(1);
12
13  if (loginResponse.status === 200) {
14    const token = loginResponse.json('token');
15
16    // Test de création de projet
17    let projectResponse = http.post('http://localhost:3000/api/projects',
18      JSON.stringify({
19        name: `Test Project ${_VU}`,
20        description: 'Performance_test_project'
21      }),
22      {
23        headers: {
24          'Authorization': `Bearer ${token}`,
25          'Content-Type': 'application/json'
26        }
27      }
28    );
29
30    check(projectResponse, {
31      'project_creation_status_is_201': (r) => r.status === 201,
32      'project_creation_time<300ms': (r) => r.timings.duration < 300,
33    }) || errorRate.add(1);
34  }
35
36  sleep(1);
37 }

```

Exemple**Résultats de performance :**

Métrique	Objectif	Mesuré	Statut
Latence P95	< 500ms	320ms	✓
Débit	> 100 req/s	150 req/s	✓
Taux d'erreur	< 1%	0.2%	✓
CPU	< 80%	65%	✓
Mémoire	< 2GB	1.2GB	✓

À FAIRE / À VÉRIFIER

- Définir des objectifs de performance mesurables
- Utiliser k6 pour les tests de charge automatisés
- Surveiller les métriques en production
- Optimiser les goulots d'étranglement identifiés
- Planifier des tests de performance réguliers

Contrôles Jury CDA

- Quels sont vos objectifs de performance ?
- Comment mesurez-vous les performances ?
- Avez-vous identifié des goulots d'étranglement ?
- Vos tests de charge sont-ils réalistes ?
- Comment surveillez-vous les performances en production ?

7.3 Qualité du code avec SonarQube

SonarQube analyse automatiquement la qualité du code, détecte les bugs, les vulnérabilités de sécurité, et les code smells. L'intégration dans la CI/CD garantit que seuls les codes de qualité sont déployés. Les métriques de qualité (complexité cyclomatique, duplication, couverture) guident l'amélioration continue.

Les règles de qualité sont configurées selon les standards de l'équipe et les bonnes pratiques de l'industrie. Les rapports de qualité facilitent la communication avec les parties prenantes et la prise de décision technique.

Lighthouse mesure automatiquement les performances, l'accessibilité, les bonnes pratiques et le SEO des applications web. L'intégration dans la CI/CD permet de surveiller ces métriques à chaque déploiement et d'alerter en cas de régression.

Exemple**Configuration SonarQube :**

```
1 # sonar-project.properties
2 sonar.projectKey=project-management-app
3 sonar.projectName=Project Management Application
4 sonar.projectVersion=1.0
5
6 # Sources et tests
7 sonar.sources=src
8 sonar.tests=tests
9 sonar.test.inclusions=tests/**/*.test.js
10
11 # Exclusions
12 sonar.exclusions=node_modules/**/*.dist/**/*.coverage/**
13
14 # Métriques de qualité
15 sonar.javascript.lcov.reportPaths=coverage/lcov.info
16 sonar.coverage.exclusions=tests/**/*.test.js
17
18 # Règles de qualité
19 sonar.qualitygate.wait=true
20 sonar.qualitygate.timeout=300
```

Exemple**Rapport de qualité SonarQube :**

Métrique	Objectif	Actuel	Statut
Couverture de code	> 80%	85%	✓
Duplication	< 3%	1.2%	✓
Complexité cyclomatique	< 10	7.3	✓
Maintenabilité	A	A	✓
Fiabilité	A	A	✓
Sécurité	A	A	✓

Exemple de correction de code smell :

```

1 // AVANT : Méthode trop longue
2 const processUserData = (userData) => {
3   const validatedData = validateUserData(userData);
4   const processedData = transformUserData(validatedData);
5   const enrichedData = enrichWithExternalData(processedData);
6   const formattedData = formatForDatabase(enrichedData);
7   const savedData = saveToDatabase(formattedData);
8   const auditLog = createAuditLog(savedData);
9   const notification = sendNotification(auditLog);
10  return notification;
11 };
12
13 // APRÈS : Méthodes courtes et focalisées
14 const processUserData = (userData) => {
15   const validatedData = validateUserData(userData);
16   const processedData = transformUserData(validatedData);
17   return saveUserData(processedData);
18 };
19
20 const saveUserData = (data) => {
21   const enrichedData = enrichWithExternalData(data);
22   const formattedData = formatForDatabase(enrichedData);
23   const savedData = saveToDatabase(formattedData);
24   auditUserAction(savedData);
25   return savedData;
26 };

```


Focus GitHub**Intégration SonarQube dans GitHub Actions :**

```

1 name: Quality Gate
2 on: [push, pull_request]
3
4 jobs:
5   quality:
6     runs-on: ubuntu-latest
7     steps:
8       - uses: actions/checkout@v3
9
10      - name: Setup Node.js
11        uses: actions/setup-node@v3
12        with:
13          node-version: '18'
14
15      - name: Install dependencies
16        run: npm ci
17
18      - name: Run tests
19        run: npm test -- --coverage
20
21      - name: SonarQube Scan
22        uses: SonarSource/sonarqube-scan-action@v1
23        env:
24          GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
25          SONAR_TOKEN: ${ secrets.SONAR_TOKEN }

```

Métriques de qualité GitHub :

- **Couverture** : 85% (objectif : >80%)
- **Bugs** : 0 (objectif : 0)
- **Vulnérabilités** : 0 (objectif : 0)
- **Code smells** : 12 (objectif : <20)
- **Duplication** : 1.2% (objectif : <3%)

Métriques Lighthouse :

- **Performance** : 92/100 (objectif : >90)
- **Accessibilité** : 95/100 (objectif : >90)
- **Best Practices** : 88/100 (objectif : >85)
- **SEO** : 90/100 (objectif : >85)

À FAIRE / À VÉRIFIER

- Intégrer SonarQube dans votre pipeline CI/CD
- Intégrer Lighthouse CI pour surveiller les performances web
- Définir des seuils de qualité appropriés
- Corriger les code smells et vulnérabilités détectés
- Surveiller les métriques de qualité dans le temps
- Former l'équipe aux bonnes pratiques de qualité

Contrôles Jury CDA

- Comment mesurez-vous la qualité de votre code ?
- Quels sont vos scores Lighthouse pour les performances et l'accessibilité ?
- Vos métriques de qualité sont-elles satisfaisantes ?
- Comment gérez-vous les code smells détectés ?
- Avez-vous intégré la qualité dans votre CI/CD ?
- Comment améliorez-vous la qualité en continu ?

7.4 Liens utiles

- Jest Documentation : <https://jestjs.io/docs/getting-started>
- Cypress Testing : <https://docs.cypress.io/>
- SonarQube : <https://docs.sonarsource.com/sonarqube/latest/>
- Lighthouse CI : <https://developers.google.com/web/tools/lighthouse-ci>
- k6 Performance Testing : <https://k6.io/docs/>
- Testing Best Practices : <https://testingjavascript.com/>

Chapitre 8

Déploiement et CI/CD

8.1 Containerisation avec Docker

La containerisation Docker standardise l'environnement de développement et de production, garantissant la reproductibilité des déploiements. Le Dockerfile multi-stage optimise la taille des images en séparant les phases de build et de runtime. Docker Compose orchestre les services (application, base de données, cache) pour un environnement complet.

Les images Docker sont optimisées pour la sécurité avec des utilisateurs non-root et des images de base minimales. Le cache des layers Docker accélère les builds et réduit la consommation de bande passante.

Exemple**Dockerfile multi-stage :**

```

1  \# Stage 1: Build
2  FROM node:18-alpine AS builder
3
4  WORKDIR /app
5
6  \# Copier les fichiers de dépendances
7  COPY package*.json ./
8  RUN npm ci --only=production
9
10 \# Copier le code source
11 COPY . .
12
13 \# Build de l'application
14 RUN npm run build
15
16 \# Stage 2: Production
17 FROM node:18-alpine AS production
18
19 \# Créer un utilisateur non-root
20 RUN addgroup -g 1001 -S nodejs
21 RUN adduser -S nextjs -u 1001
22
23 WORKDIR /app
24
25 \# Copier les dépendances de production
26 COPY --from=builder /app/node_modules ./node_modules
27 COPY --from=builder /app/dist ./dist
28 COPY --from=builder /app/package*.json ./
29
30 \# Changer le propriétaire des fichiers
31 RUN chown -R nextjs:nodejs /app
32 USER nextjs
33
34 \# Exposer le port
35 EXPOSE 3000
36
37 \# Variables d'environnement
38 ENV NODE_ENV=production
39 ENV PORT=3000
40
41 \# Commande de démarrage
42 CMD ["node", "dist/index.js"]

```

Docker Compose pour l'environnement complet (1/2) :

```

1  version: '3.8'
2
3  services:
4    app:
5      build: .
6      ports:
7        - "3000:3000"
8      environment:
9        - NODE_ENV=production
10       - DATABASE_URL=postgres://user:pass@postgres:5432/projectdb
11       - MONGODB_URI=mongodb://mongo:27017/projectlogs
12      depends_on:
13        - postgres
14        - mongo
15        - redis
16      restart: unless-stopped
17

```

Exemple**Docker Compose pour l'environnement complet (2/2) :**

```
1  mongo:
2    image: mongo:6
3    environment:
4      - MONGO_INITDB_ROOT_USERNAME=admin
5      - MONGO_INITDB_ROOT_PASSWORD=password
6    volumes:
7      - mongo_data:/data/db
8    ports:
9      - "27017:27017"
10   restart: unless-stopped
11
12   redis:
13     image: redis:7-alpine
14     ports:
15       - "6379:6379"
16     restart: unless-stopped
17
18 volumes:
19   postgres_data:
20   mongo_data:
```

À FAIRE / À VÉRIFIER

- Utiliser des Dockerfiles multi-stage pour optimiser les images
- Créer des utilisateurs non-root pour la sécurité
- Organiser les services avec Docker Compose
- Optimiser le cache des layers Docker
- Surveiller la taille et la sécurité des images

Contrôles Jury CDA

- Pourquoi utiliser Docker pour votre application ?
- Comment optimisez-vous vos images Docker ?
- Votre Dockerfile est-il sécurisé ?
- Comment gérez-vous les secrets dans Docker ?
- Avez-vous testé vos conteneurs en production ?

8.2 Pipeline CI/CD avec GitHub Actions

Le pipeline CI/CD automatise les étapes de linting, build, test, scan de sécurité et déploiement. GitHub Actions exécute ces étapes à chaque push et Pull Request, garantissant la qualité du code avant intégration. Les secrets et variables d'environnement sécurisent les informations sensibles.

Le déploiement automatique vers les environnements de staging et production suit une approche blue-green pour minimiser les risques. Les rollbacks automatiques sont déclenchés en cas de détection d'anomalies.

Exemple**Workflow GitHub Actions complet (1/3) :**

```
1 name: CI/CD Pipeline
2
3 on:
4   push:
5     branches: [main, develop]
6   pull_request:
7     branches: [main, develop]
8
9 env:
10  NODE_VERSION: '18'
11  REGISTRY: ghcr.io
12  IMAGE_NAME: ${ github.repository }
13
14 jobs:
15   # Job 1: Lint et tests
16   test:
17     runs-on: ubuntu-latest
18     steps:
19       - name: Checkout code
20         uses: actions/checkout@v4
21
22       - name: Setup Node.js
23         uses: actions/setup-node@v4
24         with:
25           node-version: ${ env.NODE_VERSION }
26           cache: 'npm'
27
28       - name: Install dependencies
29         run: npm ci
30
31       - name: Run linter
32         run: npm run lint
33
34       - name: Run type checking
35         run: npm run type-check
36
37       - name: Run tests
38         run: npm test -- --coverage
39
40       - name: Upload coverage
41         uses: codecov/codecov-action@v3
42         with:
43           token: ${ secrets.CODECOV_TOKEN }
```

Exemple**Workflow GitHub Actions complet (2/3) :**

```
1  # Job 2: Build et scan de sécurité
2  build-and-scan:
3    runs-on: ubuntu-latest
4    needs: test
5    steps:
6      - name: Checkout code
7        uses: actions/checkout@v4
8
9      - name: Build Docker image
10       run: docker build -t ${ env.IMAGE_NAME }:${ github.sha } .
11
12      - name: Run Trivy security scan
13        uses: aquasecurity/trivy-action@master
14        with:
15          image-ref: ${ env.IMAGE_NAME }:${ github.sha }
16          format: 'sarif'
17          output: 'trivy-results.sarif'
18
19      - name: Upload Trivy scan results
20        uses: github/codeql-action/upload-sarif@v2
21        with:
22          sarif_file: 'trivy-results.sarif'
```

Exemple**Workflow GitHub Actions complet (3/3) :**

```
1  # Job 3: Déploiement staging
2  deploy-staging:
3    runs-on: ubuntu-latest
4    needs: build-and-scan
5    if: github.ref == 'refs/heads/develop'
6    environment: staging
7    steps:
8      - name: Deploy to staging
9        run: |
10          echo "Deploying to staging environment"
11          # Script de déploiement staging
12          ./scripts/deploy.sh staging
13
14  # Job 4: Déploiement production
15  deploy-production:
16    runs-on: ubuntu-latest
17    needs: build-and-scan
18    if: github.ref == 'refs/heads/main'
19    environment: production
20    steps:
21      - name: Deploy to production
22        run: |
23          echo "Deploying to production environment"
24          # Script de déploiement production
25          ./scripts/deploy.sh production
26
27      - name: Run smoke tests
28        run: |
29          echo "Running smoke tests"
30          npm run test:smoke
31
32      - name: Notify team
33        if: always()
34        uses: 8398a7/action-slack@v3
35        with:
36          status: ${ job.status }
37          channel: '#deployments'
38          webhook_url: ${ secrets.SLACK_WEBHOOK }
```


Exemple**Script de déploiement (1/2) :**

```
1 #!/bin/bash
2 # scripts/deploy.sh
3
4 set -e
5
6 ENVIRONMENT=$1
7 IMAGE_TAG=${2:-latest}
8
9 echo "Deploying to $ENVIRONMENT environment with tag $IMAGE_TAG"
10
11 # Mise à jour des images Docker
12 docker-compose -f docker-compose.$ENVIRONMENT.yml pull
```

Exemple**Script de déploiement (2/2) :**

```
1 # Déploiement blue-green
2 if [ "$ENVIRONMENT" = "production" ]; then
3     # Déploiement en blue-green
4     docker-compose -f docker-compose.prod.yml up -d --scale app=2
5     sleep 30
6     docker-compose -f docker-compose.prod.yml up -d --scale app=1
7 else
8     # Déploiement simple pour staging
9     docker-compose -f docker-compose.staging.yml up -d
10 fi
11
12 # Vérification de santé
13 echo "Checking application health..."
14 curl -f http://localhost:3000/health || exit 1
15
16 echo "Deployment to $ENVIRONMENT completed successfully"
```

Focus GitHub**Pipeline CI/CD GitHub Actions :**

- **Lint** : ESLint, Prettier, TypeScript
- **Tests** : Unit, Integration, E2E
- **Sécurité** : Trivy, CodeQL, Snyk
- **Build** : Docker multi-stage
- **Deploy** : Blue-green, rollback auto

Environnements et secrets :

- **Staging** : Auto-deploy depuis develop
- **Production** : Auto-deploy depuis main
- **Secrets** : DATABASE_URL, JWT_SECRET, API_KEYS
- **Variables** : NODE_ENV, PORT, LOG_LEVEL

Métriques de pipeline :

- **Durée moyenne** : 8 minutes
- **Taux de succès** : 95%
- **Temps de déploiement** : 3 minutes
- **Rollbacks** : 2% des déploiements

À FAIRE / À VÉRIFIER

- Automatiser tous les aspects du pipeline CI/CD
- Séparer les environnements de staging et production
- Implémenter des tests de non-régression automatisés
- Configurer des alertes en cas d'échec de déploiement
- Documenter les procédures de rollback

Contrôles Jury CDA

- Votre pipeline CI/CD est-il complet ?
- Comment gérez-vous les secrets et variables ?
- Avez-vous prévu les rollbacks automatiques ?
- Comment testez-vous vos déploiements ?
- Votre pipeline respecte-t-il les bonnes pratiques ?

8.3 Documentation et monitoring

La documentation technique couvre l'API avec Swagger/OpenAPI, les procédures opérationnelles dans un runbook, et le monitoring avec des dashboards temps réel. Les logs structurés facilitent le debugging et l'analyse des performances. Les alertes automatiques notifient l'équipe en cas d'anomalie.

Le monitoring couvre les métriques applicatives (latence, débit, erreurs) et infrastructure (CPU, mémoire, disque). Les dashboards Grafana visualisent ces métriques pour faciliter la surveillance et l'analyse des tendances.

Exemple**Documentation API Swagger :**

```

1 openapi: 3.0.0
2 info:
3   title: Project Management API
4   version: 1.0.0
5
6 paths:
7   /projects:
8     get:
9       summary: Liste des projets
10      parameters:
11        - name: page
12          in: query
13          schema:
14            type: integer
15            default: 1
16      responses:
17        '200':
18          description: Liste des projets
19          content:
20            application/json:
21              schema:
22                type: object
23                properties:
24                  data:
25                    type: array
26                    items:
27                      $ref: '#/components/schemas/Project'
28      post:
29        summary: Créer un projet
30        requestBody:
31          required: true
32          content:
33            application/json:
34              schema:
35                $ref: '#/components/schemas/ProjectInput'
36        responses:
37          '201':
38            description: Projet créé
39
40 components:
41   schemas:
42     Project:
43       type: object
44       properties:
45         id: { type: string, format: uuid }
46         name: { type: string }
47         description: { type: string }
48         createdAt: { type: string, format: date-time }
49     ProjectInput:
50       type: object
51       required: [name]
52       properties:
53         name: { type: string, minLength: 1 }
54         description: { type: string }

```

Exemple**Runbook opérationnel (1/3) :**

```
1 # Runbook - Project Management Application
2
3 ## Procédures de démarrage
4
5 ### Démarrage de l'application
6 ```bash
7 # Environnement de développement
8 docker-compose up -d
9
10 # Environnement de production
11 docker-compose -f docker-compose.prod.yml up -d
12 ```
13
14 ### Vérification de santé
15 ```bash
16 curl -f http://localhost:3000/health
17 ```
```

Exemple**Runbook opérationnel (2/3) :**

```
1 ## Procédures de maintenance
2
3 ### Sauvegarde des données
4 ```bash
5 # PostgreSQL
6 pg_dump -h localhost -U user projectdb > backup_$(date +%Y%m%d).sql
7
8 # MongoDB
9 mongodump --host localhost:27017 --db projectlogs --out backup_mongo_$(
10     date +%Y%m%d)
11 ```
12
13 ### Mise à jour de l'application
14 ```bash
15 # Pull de la nouvelle image
16 docker-compose pull
17
18 # Redémarrage avec la nouvelle image
19 docker-compose up -d
20 ```
```

Exemple**Configuration de monitoring :**

```
1 \# docker-compose.monitoring.yml
2 version: '3.8'
3
4 services:
5   prometheus:
6     image: prom/prometheus
7     ports:
8       - "9090:9090"
9     volumes:
10      - ./monitoring/prometheus.yml:/etc/prometheus/prometheus.yml
11     command:
12       - '--config.file=/etc/prometheus/prometheus.yml'
13       - '--storage.tsdb.path=/prometheus'
14       - '--web.console.libraries=/etc/prometheus/console_libraries'
15       - '--web.console.templates=/etc/prometheus/consoles'
16
17   grafana:
18     image: grafana/grafana
19     ports:
20       - "3001:3000"
21     environment:
22       - GF_SECURITY_ADMIN_PASSWORD=admin
23     volumes:
24       - grafana_data:/var/lib/grafana
25
26   node-exporter:
27     image: prom/node-exporter
28     ports:
29       - "9100:9100"
30     volumes:
31       - /proc:/host/proc:ro
32       - /sys:/host/sys:ro
33       - /:/rootfs:ro
34
35 volumes:
36   grafana_data:
```

À FAIRE / À VÉRIFIER

- Documenter l'API avec OpenAPI/Swagger
- Créer un runbook opérationnel complet
- Implémenter un monitoring proactif
- Configurer des alertes automatiques
- Former l'équipe aux procédures opérationnelles

Contrôles Jury CDA

- Votre API est-elle documentée ?
- Avez-vous un runbook opérationnel ?
- Comment surveillez-vous votre application ?
- Vos alertes sont-elles configurées ?
- L'équipe connaît-elle les procédures d'urgence ?

8.4 Liens utiles

- Dockerfile reference : <https://docs.docker.com/reference/dockerfile/>
- Docker Compose : <https://docs.docker.com/compose/>
- GitHub Actions : <https://docs.github.com/actions>
- Postman : <https://learning.postman.com/docs/getting-started/introduction/>
- Prometheus : <https://prometheus.io/docs/>

Chapitre 9

Veille technologique et sécurité

9.1 Veille technologique stack

La veille technologique couvre l'évolution des technologies utilisées dans le projet : React, Node.js, PostgreSQL, MongoDB, et Docker. Les sources d'information incluent les blogs officiels, GitHub releases, et les communautés techniques. Cette veille permet d'anticiper les évolutions et de planifier les mises à jour.

L'analyse des tendances technologiques guide les choix d'architecture et d'implémentation. La participation aux communautés open source et aux conférences enrichit la compréhension des bonnes pratiques et des innovations.

Exemple

Sources de veille technologique :

- **Frontend** : React Blog, Next.js Releases, TypeScript Roadmap
- **Backend** : Node.js Releases, Express.js Updates, Prisma Changelog
- **Bases de données** : PostgreSQL Release Notes, MongoDB Updates
- **DevOps** : Docker Blog, Kubernetes Releases, GitHub Actions Updates
- **Sécurité** : OWASP News, CVE Database, Security Advisories

Exemple de veille React :

React 18.2.0 (Janvier 2024)

- +-- Nouvelles fonctionnalités
 - | +-- Concurrent Features stabilisées
 - | +-- Suspense amélioré
 - | +-- Server Components en production
- +-- Performances
 - | +-- Réduction de 15% du bundle size
 - | +-- Amélioration du rendu concurrent
- +-- Migration
 - +-- Breaking changes mineurs
 - +-- Guide de migration disponible

Impact sur le projet :

- **React 18** : Migration planifiée pour Q2 2024
- **Node.js 20** : Mise à jour pour les performances
- **PostgreSQL 16** : Nouvelles fonctionnalités JSON
- **Docker Compose V2** : Amélioration des performances

À FAIRE / À VÉRIFIER

- Suivre les releases officielles des technologies utilisées
- Participer aux communautés techniques (GitHub, Stack Overflow)
- S'abonner aux newsletters et blogs spécialisés
- Tester les nouvelles versions en environnement de développement
- Documenter les impacts et planifier les migrations

Contrôles Jury CDA

- Quelles sources utilisez-vous pour votre veille ?
- Comment identifiez-vous les technologies émergentes ?
- Avez-vous planifié des mises à jour technologiques ?
- Comment évaluez-vous l'impact des nouvelles versions ?
- Votre veille influence-t-elle vos choix techniques ?

9.2 Bonnes pratiques sécurité

La veille sécurité suit les recommandations OWASP, les CVE (Common Vulnerabilities and Exposures), et les advisories des éditeurs. L'analyse des menaces émergentes guide l'évolution des mesures de protection. Les tests de pénétration réguliers valident l'efficacité des contrôles de sécurité.

L'application des bonnes pratiques sécurité inclut la mise à jour régulière des dépendances, la configuration sécurisée des services, et la formation de l'équipe aux risques. La documentation des incidents et des contre-mesures enrichit la base de connaissances sécurité.

Exemple**Veille sécurité OWASP 2024 :**

- **A01 - Broken Access Control** : Nouveaux patterns d'attaque
- **A02 - Cryptographic Failures** : Vulnérabilités des algorithmes
- **A03 - Injection** : Évolution des techniques d'injection
- **A04 - Insecure Design** : Risques de conception
- **A05 - Security Misconfiguration** : Configurations par défaut

Exemple de vulnérabilité suivie :

```
CVE-2024-1234: Vulnerability in Express.js
+-- Severity: HIGH (CVSS 7.5)
+-- Description: Prototype pollution in req.query
+-- Affected versions: < 4.18.3
+-- Impact: Remote code execution possible
+-- Mitigation: Update to Express 4.18.3+
+-- Status: Fixed in project (v4.18.5)
```

Mesures de sécurité appliquées :

- **Dépendances** : Audit automatique avec npm audit
- **Conteneurs** : Scan de vulnérabilités avec Trivy
- **Code** : Analyse statique avec SonarQube
- **Runtime** : Monitoring des anomalies avec Prometheus
- **Formation** : Sessions sécurité trimestrielles

À FAIRE / À VÉRIFIER

- Surveiller les CVE et advisories de sécurité
- Automatiser l'audit des dépendances
- Implémenter des tests de sécurité automatisés
- Former l'équipe aux bonnes pratiques sécurité
- Documenter les incidents et les contre-mesures

Contrôles Jury CDA

- Comment surveillez-vous les vulnérabilités ?
- Avez-vous automatisé l'audit de sécurité ?
- Comment gérez-vous les vulnérabilités critiques ?
- L'équipe est-elle formée à la sécurité ?
- Avez-vous un plan de réponse aux incidents ?

9.3 Application au projet

La veille technologique et sécurité influence directement les choix d'architecture et d'implémentation du projet. Les nouvelles fonctionnalités sont évaluées selon leur impact sur la sécurité, les performances, et la maintenabilité. Les mises à jour sont planifiées selon un calendrier de migration structuré.

L'intégration des bonnes pratiques découvertes améliore continuellement la qualité du code et la sécurité de l'application. La documentation des décisions techniques facilite la transmission des connaissances et la maintenance future.

Exemple**Évolution technique du projet :**

- **Q1 2024** : Migration vers React 18 pour les performances
- **Q2 2024** : Implémentation des Server Components
- **Q3 2024** : Mise à jour PostgreSQL 16 pour les JSON
- **Q4 2024** : Migration vers Node.js 20 LTS

Améliorations sécurité appliquées :

```
1 // AVANT : Validation basique
2 const validateUser = (userData) => {
3   if (userData.email && userData.password) {
4     return true;
5   }
6   return false;
7 };
8
9 // APRÈS : Validation robuste avec sanitisation
10 const validateUser = (userData) => {
11   const schema = Joi.object({
12     email: Joi.string().email().max(255).required(),
13     password: Joi.string().min(8).pattern(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d
14       )/).required(),
15     name: Joi.string().max(100).sanitize().required()
16   });
17
18   const { error, value } = schema.validate(userData);
19   if (error) {
20     throw new ValidationError(error.details[0].message);
21   }
22
23   return value;
24 };
```

Métriques d'amélioration :

Aspect	Avant	Après	Amélioration
Temps de réponse	800ms	450ms	-44%
Vulnérabilités	12	0	-100%
Couverture tests	65%	85%	+31%
Bundle size	2.1MB	1.4MB	-33%

À FAIRE / À VÉRIFIER

- Intégrer les bonnes pratiques découvertes en veille
- Planifier les migrations technologiques
- Mesurer l'impact des améliorations
- Documenter les décisions techniques
- Partager les connaissances avec l'équipe

Contrôles Jury CDA

- Comment appliquez-vous votre veille au projet ?
- Avez-vous mesuré l'impact des améliorations ?
- Vos décisions techniques sont-elles documentées ?
- Comment partagez-vous vos connaissances ?
- Votre veille influence-t-elle la roadmap ?

9.4 Liens utiles

- InfoQ : <https://www.infoq.com/>
- OWASP News : <https://owasp.org/news/>
- PostgreSQL Release Notes : <https://www.postgresql.org/docs/release/>
- React Blog : <https://react.dev/blog>
- Node.js Releases : <https://nodejs.org/en/about/releases/>

Chapitre 10

Bilan et retour d'expérience (REX)

10.1 Objectifs atteints et non atteints

L'analyse des objectifs initiaux révèle un taux d'atteinte de 85% des objectifs SMART définis. Les objectifs métier ont été largement atteints avec la livraison du MVP dans les délais. Les objectifs techniques ont été partiellement atteints, avec quelques ajustements nécessaires pour optimiser les performances. Les objectifs pédagogiques ont été dépassés grâce aux apprentissages supplémentaires acquis.

Les objectifs non atteints concernent principalement des fonctionnalités avancées reportées en v2.0 pour respecter les contraintes temporelles. Cette priorisation a permis de livrer un produit fonctionnel et stable dans les délais impartis.

Exemple

Bilan des objectifs SMART :

Objectif	Statut	Mesure	Commentaire
Réduction temps reporting	✓Atteint	-42%	Dépassé l'objectif de -40%
Livraison MVP 6 mois	✓Atteint	5.5 mois	Livré en avance
Adoption utilisateurs	Partiel	78%	Objectif 90%, formation nécessaire
Performance P95 < 500ms	✓Atteint	320ms	Dépassé l'objectif
Sécurité 0 vulnérabilité	✓Atteint	0	Objectif atteint

Objectifs non atteints :

- **Analytics avancées** : Reporté en v2.0 (complexité technique)
- **Intégrations externes** : Reporté en v2.0 (priorités métier)
- **Mobile native** : Reporté en v2.0 (PWA suffisant)
- **IA prédictive** : Reporté en v2.0 (ROI incertain)

À FAIRE / À VÉRIFIER

- Analyser objectivement l'atteinte des objectifs
- Identifier les causes des non-atteintes
- Documenter les ajustements nécessaires
- Prévoir les actions correctives pour v2.0
- Communiquer les résultats aux parties prenantes

Contrôles Jury CDA

- Quels objectifs avez-vous atteints ?
- Pourquoi certains objectifs n'ont-ils pas été atteints ?
- Comment mesurez-vous le succès de votre projet ?
- Avez-vous ajusté vos objectifs en cours de projet ?
- Quels sont vos objectifs pour la v2.0 ?

10.2 Difficultés rencontrées et solutions

Les principales difficultés ont concerné l'intégration des bases de données hétérogènes, la gestion des performances sous charge, et la coordination des équipes distribuées. Chaque difficulté a été analysée pour identifier les causes racines et implémenter des solutions durables.

L'approche de résolution de problèmes a combiné l'analyse technique, la recherche de solutions existantes, et l'innovation pour des cas spécifiques. La documentation des solutions facilite la réutilisation et l'amélioration continue.

Exemple

Tableau risques ➡ mitigation ➡ résultat :

Risque	Mitigation	Résultat	Apprentissage
Performance DB	Index + cache Redis	Latence -60%	Cache stratégique
Intégration équipes	Daily standups	Communication +40%	Processus agile
Sécurité données	Chiffrement + audit	0 incident	Sécurité by design
Délais serrés	MVP + priorités	Livraison à temps	Focus sur l'essentiel
Complexité technique	Architecture simple	Maintenance facile	KISS principe

Exemple de difficulté résolue :

Problème: Latence élevée des requêtes PostgreSQL

```

+-- Symptômes
|   +-- Temps de réponse > 2s
|   +-- Timeout des requêtes complexes
|   +-- Surcharge CPU base de données
+-- Analyse
|   +-- Requêtes sans index appropriés
|   +-- Jointures sur de gros volumes
|   +-- Pas de cache applicatif
+-- Solutions implémentées
|   +-- Création d'index composites
|   +-- Optimisation des requêtes
|   +-- Mise en place de Redis cache
|   +-- Pagination des résultats
+-- Résultat
    +-- Latence réduite à 200ms
    +-- CPU base stabilisé
    +-- Expérience utilisateur améliorée
  
```

À FAIRE / À VÉRIFIER

- Documenter toutes les difficultés rencontrées
- Analyser les causes racines des problèmes
- Rechercher des solutions existantes avant d'innover
- Tester les solutions avant déploiement
- Partager les apprentissages avec l'équipe

Contrôles Jury CDA

- Quelles ont été vos principales difficultés ?
- Comment avez-vous résolu ces difficultés ?
- Avez-vous documenté vos solutions ?
- Ces difficultés étaient-elles prévisibles ?
- Comment éviterez-vous ces difficultés à l'avenir ?

10.3 Dettes techniques et apprentissages

Les dettes techniques identifiées incluent la refactorisation de certains composants React, l'optimisation des requêtes MongoDB, et l'amélioration de la couverture de tests. Ces dettes

sont documentées avec des priorités et des estimations pour faciliter la planification des futures itérations.

Les apprentissages techniques couvrent l'architecture microservices, la gestion des performances, et les bonnes pratiques de sécurité. Ces connaissances sont transférables à d'autres projets et enrichissent l'expertise de l'équipe.

Exemple

Registre des dettes techniques :

Dettes	Priorité	Effort	Impact	Planification
Refactor composants React	Moyenne	2 semaines	Maintenabilité	v1.2
Optimisation requêtes Mongo	Haute	1 semaine	Performance	v1.1
Tests E2E manquants	Haute	1 semaine	Qualité	v1.1
Documentation API	Basse	3 jours	Développement	v1.3
Migration TypeScript	Moyenne	3 semaines	Robustesse	v2.0

Apprentissages transférables :

- **Architecture** : Pattern Repository pour l'abstraction des données
- **Performance** : Stratégies de cache multi-niveaux
- **Sécurité** : Implémentation JWT avec refresh tokens
- **Tests** : Pyramide de tests avec couverture optimale
- **DevOps** : Pipeline CI/CD avec déploiement blue-green

Exemple d'apprentissage concret :

```

1 // AVANT : Gestion d'état complexe
2 const [projects, setProjects] = useState([]);
3 const [loading, setLoading] = useState(false);
4 const [error, setError] = useState(null);
5
6 // APRÈS : Hook personnalisé réutilisable
7 const useProjects = () => {
8   const [state, setState] = useState({
9     data: [],
10    loading: false,
11    error: null
12  });
13
14  const fetchProjects = useCallback(async () => {
15    setState(prev => ({ ...prev, loading: true }));
16    try {
17      const projects = await projectService.getAll();
18      setState({ data: projects, loading: false, error: null });
19    } catch (err) {
20      setState(prev => ({ ...prev, loading: false, error: err.message }));
21    }
22  }, []);
23
24  return { ...state, fetchProjects };
25 };

```

À FAIRE / À VÉRIFIER

- Identifier et documenter toutes les dettes techniques
- Prioriser les dettes selon leur impact et urgence
- Planifier la résolution des dettes dans les futures versions
- Capitaliser sur les apprentissages pour les futurs projets
- Partager les bonnes pratiques avec l'équipe

Contrôles Jury CDA

- Quelles dettes techniques avez-vous identifiées ?
- Comment priorisez-vous ces dettes ?
- Quels apprentissages tirez-vous de ce projet ?
- Ces apprentissages sont-ils transférables ?
- Comment capitalisez-vous sur ces expériences ?

10.4 Liens utiles

- Postmortems (Google SRE) : <https://sre.google/sre-book/postmortem-culture/>
- Technical Debt : <https://martinfowler.com/bliki/TechnicalDebt.html>
- Retrospectives : <https://www.atlassian.com/team-playbook/plays/retrospective>
- Lessons Learned : <https://bit.ly/lessons-learned>
- Knowledge Management : <https://bit.ly/knowledge-management>

Chapitre 11

Conclusion et remerciements

11.1 Synthèse du projet

Ce projet de développement d'une application de gestion de projets a permis de mettre en pratique les compétences acquises en alternance CDA dans un contexte professionnel concret. L'architecture 3 tiers avec React, Node.js, PostgreSQL et MongoDB a démontré sa robustesse et sa scalabilité. Les objectifs métier ont été largement atteints avec une réduction de 42% du temps de reporting et une adoption utilisateur de 78%.

La démarche méthodologique Agile a facilité la collaboration et l'adaptation aux besoins évolutifs. Les bonnes pratiques de développement, de sécurité et de déploiement ont été appliquées avec succès, garantissant la qualité et la fiabilité de la solution livrée.

Exemple

Chiffres clés du projet :

Métrique	Valeur	Objectif
Durée de développement	5.5 mois	6 mois
Couverture de code	85%	80%
Performance P95	320ms	500ms
Vulnérabilités sécurité	0	0
Adoption utilisateurs	78%	90%
Temps de reporting	-42%	-40%

Technologies maîtrisées :

- **Frontend** : React 18, TypeScript, Redux Toolkit
- **Backend** : Node.js, Express.js, Prisma ORM
- **Bases de données** : PostgreSQL, MongoDB, Redis
- **DevOps** : Docker, GitHub Actions, SonarQube
- **Sécurité** : JWT, Argon2, OWASP Top 10

À FAIRE / À VÉRIFIER

- Synthétiser les résultats quantitatifs et qualitatifs
- Mettre en avant les compétences développées
- Identifier les points forts et les axes d'amélioration
- Préparer la présentation des résultats au jury
- Documenter les apprentissages pour la suite du parcours

Contrôles Jury CDA

- Pouvez-vous résumer les résultats de votre projet ?
- Quelles compétences avez-vous développées ?
- Quels sont vos points forts et faibles ?
- Comment évaluez-vous votre progression ?
- Quels sont vos objectifs pour la suite ?

11.2 Perspectives d'évolution

Les perspectives d'évolution du projet incluent le développement de la v2.0 avec des fonctionnalités avancées : analytics prédictives, intégrations externes, et intelligence artificielle. L'architecture actuelle permet une évolution progressive sans refactoring majeur. La roadmap technique prévoit la migration vers des technologies émergentes et l'optimisation continue des performances.

L'expérience acquise sur ce projet constitue une base solide pour aborder des projets plus complexes et des responsabilités techniques élargies. Les compétences développées sont directement applicables à d'autres contextes métier et technologiques.

Exemple

Roadmap technique v2.0 :

Q1 2025: Fonctionnalités avancées

- +-- Analytics prédictives avec machine learning
- +-- Intégrations API externes (Slack, Teams)
- +-- Notifications push temps réel
- +-- Optimisation performances (P95 < 200ms)

Q2 2025: Intelligence artificielle

- +-- Assistant IA pour la gestion de projet
- +-- Recommandations automatiques
- +-- Détection d'anomalies
- +-- Chatbot support utilisateur

Q3 2025: Évolutions technologiques

- +-- Migration vers React Server Components
- +-- Mise à jour Node.js 20 LTS
- +-- PostgreSQL 16 nouvelles fonctionnalités
- +-- Monitoring avancé avec Grafana

Compétences à développer :

- **Architecture** : Microservices, Event-driven architecture
- **Cloud** : AWS/Azure, Kubernetes, Serverless
- **IA/ML** : TensorFlow, PyTorch, MLOps
- **Sécurité** : Zero Trust, DevSecOps
- **Leadership** : Architecture decision records, mentoring

À FAIRE / À VÉRIFIER

- Définir une vision claire pour l'évolution du projet
- Identifier les technologies émergentes pertinentes
- Planifier les compétences à développer
- Anticiper les besoins métier futurs
- Maintenir la veille technologique

Contrôles Jury CDA

- Quelles sont vos perspectives d'évolution ?
- Comment prévoyez-vous l'évolution technique ?
- Quelles compétences souhaitez-vous développer ?
- Comment anticipez-vous les besoins futurs ?
- Votre projet est-il évolutif ?

11.3 Remerciements

Je tiens à remercier toutes les personnes qui ont contribué à la réussite de ce projet et à mon apprentissage en alternance CDA. Ces remerciements s'adressent à l'équipe technique, aux utilisateurs métier, aux formateurs, et à tous ceux qui ont partagé leur expertise et leur temps.

L'accompagnement reçu a été déterminant dans l'acquisition des compétences techniques et méthodologiques nécessaires à la réalisation de ce projet. Ces remerciements témoignent de la reconnaissance pour l'investissement de chacun dans ma formation professionnelle.

Exemple**Remerciements personnalisés :**

- **Mon tuteur entreprise** : Pour son accompagnement technique et son expertise
- **L'équipe de développement** : Pour la collaboration et le partage de connaissances
- **Les utilisateurs métier** : Pour leurs retours constructifs et leur patience
- **Les formateurs CDA** : Pour la transmission des fondamentaux techniques
- **La communauté open source** : Pour les outils et ressources mis à disposition

Apprentissages clés :

- **Collaboration** : L'importance du travail d'équipe en développement
- **Communication** : La nécessité de bien communiquer avec les parties prenantes
- **Adaptabilité** : La capacité à s'adapter aux changements et contraintes
- **Qualité** : L'exigence de qualité dans le développement logiciel
- **Veille** : L'importance de la veille technologique continue

À FAIRE / À VÉRIFIER

- Exprimer sa gratitude de manière sincère et personnalisée
- Reconnaître l'apport spécifique de chaque personne
- Mettre en avant les apprentissages tirés des interactions
- Maintenir les relations professionnelles établies
- Préparer la suite du parcours avec confiance

Contrôles Jury CDA

- Qui souhaitez-vous remercier particulièrement ?
- Quels apprentissages tirez-vous de ces interactions ?
- Comment envisagez-vous la suite de votre parcours ?
- Quelles relations professionnelles avez-vous nouées ?
- Comment comptez-vous maintenir ces relations ?

11.4 Déploiement et documentation

Dans cette section, vous devez présenter votre stratégie de déploiement et la documentation technique de votre projet. Le jury attend une compréhension claire de votre approche

opérationnelle et de la maintenabilité de votre solution.

Votre stratégie de déploiement : *[Décrivez votre approche de déploiement et de documentation]*

11.4.1 Docker

Dans cette sous-section, vous devez détailler votre approche de containerisation avec Docker. Le jury attend une explication claire de votre Dockerfile et de votre orchestration.

Votre containerisation : *[Décrivez votre Dockerfile et votre approche Docker]*

Conteneurisation

Votre Dockerfile : *[Décrivez votre Dockerfile multi-stage]*

Exemple

Dockerfile multi-stage :

```
1 # Stage 1: Build
2 FROM node:18-alpine AS builder
3 WORKDIR /app
4 COPY package*.json ./
5 RUN npm ci --only=production
6 COPY . .
7 RUN npm run build
8
9 # Stage 2: Production
10 FROM node:18-alpine AS production
11 RUN addgroup -g 1001 -S nodejs
12 RUN adduser -S nextjs -u 1001
13 WORKDIR /app
14 COPY --from=builder /app/node_modules ./node_modules
15 COPY --from=builder /app/dist ./dist
16 COPY --from=builder /app/package*.json ./
17 RUN chown -R nextjs:nodejs /app
18 USER nextjs
19 EXPOSE 3000
20 ENV NODE_ENV=production
21 CMD ["node", "dist/index.js"]
```

Compose

Votre Docker Compose : *[Décrivez votre orchestration des services]*

Exemple**Docker Compose pour l'environnement complet :**

```

1 version: '3.8'
2 services:
3   app:
4     build: .
5     ports:
6       - "3000:3000"
7     environment:
8       - NODE_ENV=production
9       - DATABASE_URL=postgresql://user:pass@postgres:5432/projectdb
10    depends_on:
11      - postgres
12      - redis
13    restart: unless-stopped
14
15    postgres:
16      image: postgres:15-alpine
17      environment:
18        - POSTGRES_DB=projectdb
19        - POSTGRES_USER=user
20        - POSTGRES_PASSWORD=pass
21      volumes:
22        - postgres_data:/var/lib/postgresql/data
23      restart: unless-stopped
24
25    redis:
26      image: redis:7-alpine
27      restart: unless-stopped
28
29 volumes:
30   postgres_data:

```

11.4.2 GitHub (code source)

Dans cette sous-section, vous devez présenter votre organisation du code source sur GitHub. Le jury attend une explication claire de votre structure de repository et de vos conventions.

Votre organisation GitHub : *[Décrivez votre structure de repository et vos conventions]*

Exemple**Structure du repository :**

```

project-management-app/
+-- src/                      # Code source
|  +-- frontend/              # Application React
|  +-- backend/               # API Node.js
|  +-- shared/                # Code partagé
+-- docs/                     # Documentation
|  +-- api/                   # Documentation API
|  +-- deployment/            # Procédures de déploiement
|  +-- architecture/          # Documentation architecture
+-- scripts/                  # Scripts utilitaires
+-- tests/                    # Tests automatisés
+-- docker/                   # Configuration Docker
+-- .github/                  # GitHub Actions et templates

```

11.4.3 CI/CD

Dans cette sous-section, vous devez présenter votre pipeline CI/CD. Le jury attend une explication claire de votre automatisation et de vos environnements.

Votre pipeline CI/CD : *[Décrivez votre automatisation et vos environnements]*

Exemple

Pipeline CI/CD GitHub Actions :

```
1 name: CI/CD Pipeline
2 on:
3   push:
4     branches: [main, develop]
5   pull_request:
6     branches: [main, develop]
7
8 jobs:
9   test:
10    runs-on: ubuntu-latest
11    steps:
12      - uses: actions/checkout@v4
13      - name: Setup Node.js
14        uses: actions/setup-node@v4
15        with:
16          node-version: '18'
17      - name: Install dependencies
18        run: npm ci
19      - name: Run tests
20        run: npm test -- --coverage
21
22    deploy-staging:
23      runs-on: ubuntu-latest
24      needs: test
25      if: github.ref == 'refs/heads/develop'
26      steps:
27        - name: Deploy to staging
28          run: ./scripts/deploy.sh staging
29
30    deploy-production:
31      runs-on: ubuntu-latest
32      needs: test
33      if: github.ref == 'refs/heads/main'
34      steps:
35        - name: Deploy to production
36          run: ./scripts/deploy.sh production
```

11.4.4 SonarQube

Dans cette sous-section, vous devez présenter votre approche de qualité du code avec SonarQube. Le jury attend une explication claire de vos métriques et de votre intégration.

Votre qualité du code : *[Décrivez vos métriques de qualité et votre intégration SonarQube]*

Exemple**Métriques de qualité SonarQube :**

Métrique	Objectif	Actuel	Statut
Couverture de code	> 80%	85%	✓
Duplication	< 3%	1.2%	✓
Complexité cyclomatique	< 10	7.3	✓
Maintenabilité	A	A	✓
Fiabilité	A	A	✓
Sécurité	A	A	✓

11.4.5 Swagger

Dans cette sous-section, vous devez présenter votre documentation API avec Swagger. Le jury attend une explication claire de votre documentation et de son utilisation.

Votre documentation API : *[Décrivez votre documentation Swagger et son utilisation]*

Exemple**Documentation API Swagger :**

```
1 openapi: 3.0.0
2 info:
3   title: Project Management API
4   version: 1.0.0
5   description: API pour la gestion des projets
6
7 paths:
8   /projects:
9     get:
10      summary: Liste des projets
11      responses:
12        '200':
13          description: Liste des projets
14          content:
15            application/json:
16              schema:
17                type: object
18                properties:
19                  data:
20                    type: array
21                    items:
22                      $ref: '#/components/schemas/Project'
23
24 components:
25   schemas:
26     Project:
27       type: object
28       properties:
29         id:
30           type: string
31           format: uuid
32         name:
33           type: string
34         description:
35           type: string
36         createdAt:
37           type: string
38           format: date-time
```

À FAIRE / À VÉRIFIER

- Documenter complètement votre API avec Swagger
- Intégrer SonarQube dans votre pipeline CI/CD
- Organiser votre code source de manière claire
- Automatiser tous les aspects du déploiement
- Maintenir la documentation à jour

Contrôles Jury CDA

- Comment organisez-vous votre code source ?
- Votre pipeline CI/CD est-il complet ?
- Comment mesurez-vous la qualité de votre code ?
- Votre API est-elle documentée ?
- Comment gérez-vous les déploiements ?

11.5 Liens utiles

- Dockerfile reference : <https://docs.docker.com/reference/dockerfile/>
- Docker Compose : <https://docs.docker.com/compose/>
- GitHub Actions : <https://docs.github.com/actions>
- SonarQube : <https://docs.sonarsource.com/sonarqube/latest/>
- Swagger/OpenAPI : <https://swagger.io/specification/>
- CDA Formation : <https://www.cda.asso.fr/>
- Colint.school : <https://colint.school/>