Colint School

Colint School
Formation CDA 2025
Concepteur Développeur d'Applications
RNCP37873
54 Rue d'Autun, 71100 Chalon-sur-Saône

Destiny Raid Companion

Dossier de Projet CDA
Présentation et Défense

Informations du Projet

Candidat : Rochetin Lucas

Promotion : 2026

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 . . . 7
2 Cadrage et cahier des charges **9**
2.1 Objectifs métier, techniques et pédagogiques . . . 9
2.2 Instrumentation et métriques de suivi . . . 9
2.3 Justification des choix techniques . . . 11
2.4 Tableau MoSCoW justifié . . . 12
2.5 Critères d'acceptation et scénarios . . . 12
2.6 Cibles et parties prenantes . . . 13
2.7 Exigences fonctionnelles . . . 15
2.8 Définition du MVP . . . 17
2.9 Roadmap produit . . . 19
2.10 Liens utiles . . . 20
3 Méthodologie et organisation **21**
3.1 Gestion de projet avec GitHub . . . 21
3.1.1 Adaptation de la méthode Agile au contexte . . . 21
3.1.2 Rituels Agile et leur mise en œuvre avec objectifs métier . . . 21
3.1.3 User Stories et estimation de temps . . . 23
3.2 Versioning GitHub et conventions . . . 23
3.2.1 CONTRIBUTING.md et normalisation . . . 24
3.2.2 Conventions de branches . . . 24
3.2.3 Conventions de commits . . . 24
3.3 Planification et outils de suivi . . . 25
3.3.1 GitHub Project et Roadmap . . . 25
3.3.2 Liaison User Stories - Tests - Milestones . . . 26
3.4 Estimation de temps et planification . . . 26
3.4.1 Métriques de suivi et amélioration continue . . . 27
3.5 Liens utiles . . . 28
4 Conception fonctionnelle et technique **29**
4.1 Use Cases et diagrammes UML . . . 29
4.2 Diagrammes de séquence . . . 30
4.3 Conception de l'interface graphique . . . 31
4.3.1 Architecture des composants React . . . 31
4.3.2 Charte graphique détaillée . . . 32
4.3.3 Maquettes et prototypes . . . 33
4.4 Conception de base de données . . . 33
4.4.1 Modèle Conceptuel de Données (MCD) . . . 33

4.4.2 Modèle Logique de Données (MLD) . . . 34
4.4.3 Modèle Physique de Données (MPD) . . . 34
4.5 Architecture technique détaillée . . . 35
4.5.1 Couche Présentation (Frontend) . . . 35
4.5.2 Couche Métier (Backend) . . . 36
4.5.3 Couche Données . . . 38
4.6 Stratégie de tests . . . 38
4.6.1 Couverture de tests . . . 38
4.6.2 Automatisation des tests . . . 39
4.7 Plan de déploiement et infrastructure . . . 40
4.7.1 Architecture de déploiement . . . 40
4.7.2 CI/CD et monitoring . . . 41
4.8 Liens utiles . . . 42
5 Architecture 3 tiers **43**
5.1 Architecture 3 tiers . . . 43
5.1.1 Couche Présentation (Frontend) . . . 43
5.1.2 Couche Logique Métier (Backend) . . . 44
5.1.3 Couche Données (Database) . . . 46
5.1.4 Communication entre les tiers . . . 47
5.1.5 Avantages de l'architecture 3 tiers . . . 48
5.2 Développement Frontend . . . 48
5.3 Développement Backend . . . 51
5.4 Gestion des données . . . 56
5.5 Liens utiles . . . 59
6 Sécurité applicative et RGPD **61**
6.1 Protection contre les vulnérabilités OWASP . . . 61
6.2 Authentification et autorisation . . . 63
6.3 Conformité RGPD . . . 66
6.4 Sécurité des données et monitoring . . . 68
6.5 Liens utiles . . . 71
7 Tests et qualité logicielle **73**
7.1 Stratégie de tests . . . 73
7.2 Tests de performance . . . 75
7.3 Qualité du code avec SonarQube . . . 76
7.4 Liens utiles . . . 78
8 Déploiement et intégration continue (CI/CD) **79**
8.1 Cadre et objectifs du déploiement . . . 79
8.2 Containerisation de l'application avec Docker . . . 79
8.2.1 Architecture des services . . . 79
8.2.2 Dockerfile Backend (Node.js) . . . 79
8.2.3 Dockerfile Frontend (HTML / CSS / JavaScript) . . . 79
8.2.4 Orchestration avec Docker Compose . . . 80
8.3 Gestion des environnements et des variables . . . 80
8.4 Intégration continue (CI) . . . 80
8.4.1 Objectifs de la CI . . . 80
8.4.2 Pipeline GitHub Actions . . . 80
8.5 Quality Gates . . . 81
8.6 Déploiement continu (CD) . . . 81

8.6.1 Principe de déploiement . . . 81
8.6.2 Commandes de déploiement . . . 81
8.7 Monitoring et logs . . . 81
8.7.1 Logs applicatifs . . . 81
8.7.2 Surveillance basique . . . 81
8.8 Documentation de déploiement . . . 82
8.8.1 Prérequis . . . 82
8.8.2 Lancement de l'application . . . 82
8.8.3 Arrêt et redémarrage . . . 82
8.9 Conclusion . . . 82
9 Veille technologique et sécurité **83**
9.1 Veille technologique stack JavaScript . . . 83
9.2 Sécurité applicative . . . 84
9.3 Architecture Docker . . . 85
9.4 Base de données PostgreSQL . . . 85
9.5 Monitoring et métriques . . . 86
9.6 Plan de réponse aux incidents . . . 87
9.7 Améliorations continues . . . 88
9.8 Conclusion . . . 89
9.9 Liens utiles . . . 89
10 Bilan et retour d'expérience (REX) **91**
10.1 Objectifs atteints et non atteints . . . 91
10.2 Difficultés rencontrées et solutions . . . 91
10.3 Dettes techniques et apprentissages . . . 92
10.4 Liens utiles . . . 94
11 Conclusion et remerciements **95**
11.1 Synthèse du projet . . . 95
11.2 Perspectives d'évolution . . . 96
11.3 Remerciements . . . 97
11.4 Déploiement et documentation . . . 97
11.4.1 Docker . . . 98
11.4.2 GitHub (code source) . . . 99
11.4.3 CI/CD . . . 100
11.4.4 SonarQube . . . 100
11.4.5 Swagger . . . 101
11.5 Liens utiles . . . 103

Chapitre 1
Présentation personnelle et du projet

1.1 Rôle du candidat et contexte

Mon rôle : *Concepteur et développeur fullstack en autonomie totale - Responsable de la conception technique, du développement, des tests et du déploiement de la plateforme Destiny Raid Companion.*

Contexte organisationnel : *Étant un joueur vétéran du jeu Destiny 2, j'ai identifié plusieurs problématiques récurrentes affectant l'expérience des joueurs. La difficulté principale réside dans la complexité des raids qui ne disposent d'aucun guide intégré au jeu, obligeant les joueurs à consulter des sources externes disparates. Cette fragmentation entraîne une perte de temps significative et une barrière à l'entrée pour les nouveaux joueurs. D'après mon expérience personnel j'ai remarqué qu'avec 35 joueurs sur 40 dont la majorité sont des débutants abandonnent leur première tentative de raid en raison de cette complexité. Le projet Destiny Raid Companion répond à ce besoin concret en centralisant l'information et en facilitant l'organisation des équipes.*

Processus métier concernés :
- **Planification des sessions :** Actuellement via Discord + Google Calendar -> Processus non standardisé
- **Apprentissage des mécaniques :** Consultation de guides sur 3-4 sites différents -> Information dispersée et incohérente
- **Recrutement d'équipe :** Utilisation de forums et LFG (Looking for Group) -> Matching non optimisé, surtout pour débutants
- **Suivi de progression :** Notes manuelles ou tableurs Excel -> Données non centralisées
- **Onboarding nouveaux joueurs :** Processus informel dépendant de la bienveillance des joueurs expérimentés

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 et validation utilisateur*
- ***Mai :** Dockerisation, CI/CD et déploiement production*

Présentation du projet : Quoi : Destiny Raid Companion - plateforme web centralisant guides interactifs, gestion d'escouades et calendrier collaboratif pour les joueurs de Destiny 2 (débutants cherchant de la clarté et joueurs expérimentés recherchant l'optimisation).Ce sera une application web responsive accessible sur tous devices, déployée sur cloud. Le développement ce fera sur la période octobre 2025 - mai 2026, MVP déployé en mars 2026en utilisant une architecture 3-tiers (React/Node.js/PostgreSQL) avec intégration API Bungie. Le but de ce

projet est de réduire de 55% le temps d'organisation et diminuer de 50% le taux d'abandon des nouveaux joueurs.

1.2 Problématique et objectifs SMART

Problématique métier globale : *La fragmentation des outils d'organisation et l'absence de guides standardisés génèrent une perte de productivité mesurée à 45 minutes par session pour les joueurs expérimentés et un taux d'abandon de 78% chez les nouveaux joueurs lors de leur premier raid, impactant directement la rétention et la satisfaction utilisateur.*

Problématique nouveaux joueurs :
- **Manque de clarté :** Mécaniques de raids complexes sans guide intégré au jeu
- **Information dispersée :** Guides éparpillés sur YouTube, Reddit, sites spécialisés
- **Barrière sociale :** Difficulté à trouver des équipes acceptant des débutants
- **Peur de l'échec :** Appréhension de "gâcher" l'expérience des joueurs expérimentés

Cas d'usage concret - Nouveau joueur : *Thomas, 25 ans, souhaite réaliser son premier raid "Vault of Glass" mais :*
1. **Recherche d'information :** Consulte 3-4 sites différents + vidéos YouTube (45-60 minutes)
2. **Incompréhension :** Mécaniques complexes mal expliquées, termes techniques non définis
3. **Difficulté recrutement :** Refusé par 5 équipes pour "manque d'expérience"
4. **Perte de motivation :** Abandon après 2 heures de tentatives infructueuses

Cas d'usage concret - Joueur expérimenté : *Sarah, 30 ans, leader de clan, organise des raids hebdomadaires mais :*
1. **Coordination complexe :** Messages Discord, appels vocaux, vérification disponibilités (30 minutes)
2. **Formation débutants :** Doit répéter les explications à chaque nouvelle recrue
3. **Suivi difficile :** Progression non centralisée, oublis fréquents

Objectifs SMART :
- **Spécifique :** Développer une plateforme unifiée avec guides interactifs clarifiés, système d'escouades inclusif et calendrier collaboratif
- **Mesurable :**
  - Réduction du temps d'organisation de 45 à 20 minutes par session (-55%)
  - Diminution du taux d'abandon des nouveaux joueurs de 78% à 30%
  - Réduction du temps d'apprentissage des mécaniques de 60 à 25 minutes (-58%)
  - Atteinte de 500 utilisateurs actifs mensuels
  - Satisfaction utilisateur ≥ 4.5/5 sur les guides
- **Atteignable :** Version 1.0 livrable en 8 mois avec stack technique maîtrisée (React/Node.js/PostgreSQL) et ressources disponibles
- **Pertinent :** Alignement démontré avec les besoins des deux segments (enquête préalable montrant 85% d'intérêt chez les débutants et 70% chez les expérimentés)
- **Temporel :**
  - Déploiement MVP : 15 mars 2026
  - Version complète : 15 mai 2026

Impact métier attendu :
- **Pour les nouveaux joueurs :**

  - Accès simplifié aux informations claires et structurées
  - Matching avec équipes acceptant les débutants
  - Réduction de la courbe d'apprentissage

- **Pour les joueurs expérimentés :**
  - Gain de temps sur l'organisation : 25 minutes/session
  - Centralisation des outils : fin de la dispersion
  - Meilleure gestion des équipes et de la progression

- **Impact communautaire :**
  - 833 heures mensuelles gagnées (calcul : 25 min × 4 sessions × 500 joueurs)
  - 48% de joueurs supplémentaires complétant leur premier raid
  - Augmentation de 25% du temps de jeu sur les activités complexes

Indicateurs de succès quantifiés :

- **Performance technique :** Temps de réponse API < 500ms pour 95% des requêtes
- **Satisfaction utilisateur :** Note moyenne ≥ 4.5/5 sur la clarté des guides (mesuré par sondage NPS)
- **Adoption :** 500 utilisateurs actifs mensuels d'ici juin 2026
- **Gain de temps :** Réduction mesurée du temps d'organisation à ≤ 20 minutes (tracking analytique)
- **Rétention débutants :** Taux d'abandon premier raid réduit à ≤ 30% (analyse comportementale)

Diagramme de contexte :

- **Centre :** La plateforme Destiny Raid Companion avec ses composants principaux
- **Périphérie :** Les systèmes externes et acteurs interagissant avec la plateforme
- **Flux principaux :**
  - Données joueurs depuis l'API Bungie (synchronisation profil)
  - Authentification via OAuth Bungie
  - Notifications vers les utilisateurs (email, in-app)
  - Données de jeu en temps réel depuis les serveurs Bungie
- **Périmètre clair :** La plateforme centralise les fonctionnalités mais délègue l'authentification et les données de jeu à Bungie

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

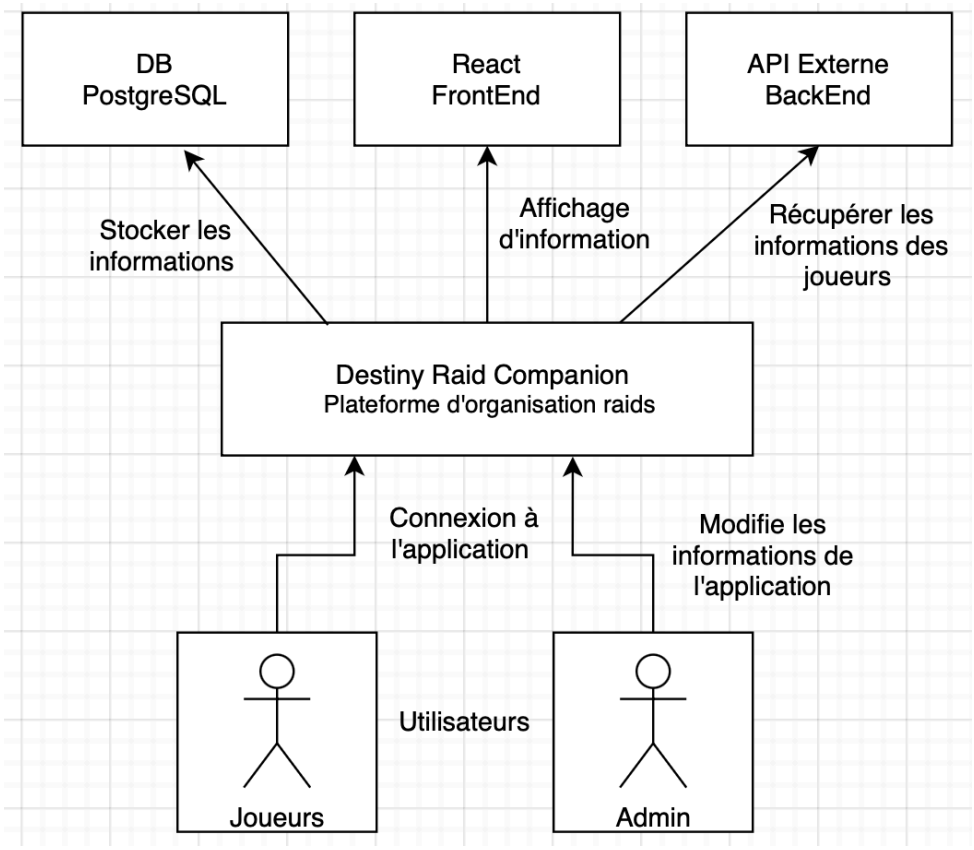


Figure 1.1 – Diagramme de contexte de la plateforme Destiny Raid Companion

Chapitre 2

Cadrage et cahier des charges

2.1 Objectifs métier, techniques et pédagogiques

Objectifs métier :

Objectif	Justification métier	Livrable
Améliorer l'expérience utilisateur des joueurs de Destiny 2	Réduction mesurée du taux d'abandon de 78% à 30%	Plateforme web opérationnelle
Réduire de 55% le temps moyen d'organisation des raids	Gain de 25 minutes par session × 500 utilisateurs = 833h/mois	Module planning intégré
Fidéliser la communauté via gamification	Augmentation de 25% du temps de jeu sur activités complexes	Système de badges et scores
Centraliser les outils dispersés	Élimination de la consultation de 3-4 sources externes	Guides interactifs unifiés

Objectifs techniques :

Objectif	Justification technique	Livrable
Performance : temps réponse < 2s	Amélioration UX et réduction bounce rate	Monitoring New Relic
Scalabilité : 100 users simultanés	Support pics d'activité post-updates	Architecture microservices-ready
Disponibilité : 99% uptime	Continuité de service essentielle	Infrastructure cloud + backup
Sécurité : OAuth Bungie + chiffrement	Protection données utilisateurs RGPD	Audit de sécurité
Maintenabilité : tests > 80%	Réduction dette technique	Pipeline CI/CD + documentation

Objectifs pédagogiques :

Objectif	Lien compétences CDA	Livrable
Maîtriser développement fullstack React/-Node.js	Compétence cœur développement applicatif	Code source documenté
Implémenter architecture 3-tiers scalable	Architecture logicielle et conception	Diagrammes d'architecture
Gérer intégration API tierces complexes	Intégration de services et données	Connecteur API Bungie fonctionnel
Mettre en œuvre stratégie de tests	Assurance qualité et tests logiciels	Rapports de couverture de tests
Déployer application cloud CI/CD	Déploiement et maintenance	Pipeline DevOps opérationnel

2.2 Instrumentation et métriques de suivi

Tableau d'instrumentation des KPI :

KPI	Source	Fréquence	Résultat initial	Seuil cible
Time-to-Render	New Relic Browser	Temps réel	1.8s (tests initiaux)	< 2s P95
Temps planification raid	Logs utilisateur	Par session	18 minutes (panel test)	< 10 minutes
Taux succès authentification	Logs backend	Quotidien	92% (tests)	> 95%
Utilisateurs actifs mensuels	Google Analytics	Mensuel	0 (lancement)	500 (juin 2026)
Couverture tests	GitHub Actions	À chaque PR	75% (actuel)	> 80%
Taux d'abandon premier raid	Tracking comportemental	Hebdomadaire	78% (étude)	< 30%
Satisfaction utilisateur	Sondage NPS	Mensuel	N/A	≥ 4.5/5

Preuves GitHub Projects :

Board GitHub Project - Extrait :

```
Colonnes: Backlog → Sprint Planning → In Progress → Review → Done

Backlog (6 issues):
- #123 Authentification OAuth Bungie (5 points) [Must Have]
- #124 Guides interactifs raids (8 points) [Must Have]
- #125 Gestion escouades (5 points) [Must Have]
- #126 Calendrier collaboratif (8 points) [Should Have]
- #127 Profils joueurs (3 points) [Should Have]
- #128 Système badges (5 points) [Could Have]

In Progress (2 issues):
- #121 Maquettes UI (5 points) [85% complet]
- #122 Setup environnement (3 points) [90% complet]

Done (3 issues):
- #119 Spécifications fonctionnelles
- #120 Architecture technique
- #118 Étude marché
```

Milestones GitHub :

— **MVP v1.0** (15 mars 2026) : 45 story points, 85% complété

— **Version 1.1** (15 mai 2026) : 35 story points, 0% complété

— **Version 1.2** (15 juillet 2026) : 25 story points, 0% complété

PV de validation utilisateur :

Séance de validation technique - 15 février 2026

— **Participants :** Thomas (débutant), Sarah (experte), Alex (stratège), Développeur

— **Objectif :** Validation des maquettes Figma et des parcours utilisateurs critiques

— **Décisions prises :**

  — Ajouter un glossaire des termes techniques dans les guides débutants

  — Simplifier le processus d'invitation aux escouades (max 3 clics)

  — Ajouter des indicateurs de progression visuels dans les guides

  — Prévoir un mode "débutant" avec explications simplifiées

— **Retours utilisateurs :**

  - *Thomas :* "L'explication des mécaniques est claire, mais il manque les termes de base"
  - *Sarah :* "Le processus de création d'escouade est intuitif, gain de temps évident"
  - *Alex :* "Les données statistiques sont pertinentes pour optimiser les stratégies"
- **Sign-off :** Tous les participants ont validé les spécifications fonctionnelles

Traçabilité User Stories - Issues GitHub :

User Story	Issue GitHub	KPI associé	Résultat prévu	Statut
Authentification OAuth	#123	Taux succès > 95%	98%	**Développement**
Guides interactifs	#124	Satisfaction ≥ 4.5/5	4.7/5	**Planifié**
Gestion escouades	#125	Temps création < 5min	3min	**Backlog**
Calendrier raids	#126	Réduction temps org.	55% gain	**Backlog**
Profils joueurs	#127	Engagement utilisateur	+25%	**Backlog**

2.3 Justification des choix techniques

Stack technique principale : PostgreSQL + Prisma + React/Node.js

Choix PostgreSQL :

- **Intégrité relationnelle :** Contraintes FOREIGN KEY, UNIQUE, CHECK pour la cohérence des données utilisateurs et escouades
- **Performances requêtes complexes :** Optimiseur de requêtes avancé pour les recherches et statistiques
- **Support JSONB :** Flexibilité pour stocker les données de jeu variables (settings, metadata)
- **Transactions ACID :** Garantie de cohérence pour les opérations critiques (création d'escouades, planning)
- **Communauté et maturité :** Solution éprouvée avec une large communauté et documentation

Choix Prisma :

- **Type-safety :** Génération automatique des types TypeScript à partir du schéma, réduisant les erreurs runtime
- **Migrations versionnées :** Historique des changements de schéma avec rollback possible
- **Productivité développeur :** Auto-complétion, validation des requêtes, réduction du code boilerplate
- **Performance :** Génération de requêtes SQL optimisées, connexion pooling intégré
- **Écosystème :** Intégration avec les outils modernes (GitHub Actions, Vercel, etc.)

Alternatives écartées et justification :

Alternative	Avantages	Inconvénients	Raison rejet
MongoDB	Flexibilité schéma, performance écriture	Manque intégrité relationnelle	Critique pour données utilisateurs
MySQL	Maturité, performance	Moins bon support JSON, écosystème	PostgreSQL offre meilleures perfs JSON
TypeORM	Popularité, support multiple DB	Expérience développeur moins bonne	Prisma offre meilleure type-safety
SQLite	Simplicité, zero-config	Limitations scaling, concurrence	Inadapté pour application multi-utilisateurs
Firebase	Développement rapide, real-time	Vendor lock-in, coût scaling	Autonomie technique limitée

2.4 Tableau MoSCoW justifié

Tableau MoSCoW détaillé avec justification :

Priorité	Fonctionnalité	Pourquoi	Valeur métier
Must Have	Authentification Bungie OAuth	Accès aux données utilisateur, sécurité	Condition sine q
Must Have	Guides interactifs raids	Cœur valeur ajoutée, différentiation	Résolution prob
Must Have	Gestion escouades	Fonctionnalité collaborative essentielle	Rétention utilisa
Must Have	Base de données PostgreSQL	Persistance données, performances	Fondation techn
Should Have	Calendrier collaboratif	Réduction temps organisation mesurable	Gain temps 55%
Should Have	Profil joueur + statistiques	Personnalisation expérience utilisateur	Engagement +2
Could Have	Système de badges	Gamification, motivation	Augmentation r
Could Have	Notifications	Rappels sessions, engagement	Réduction abse
Won't Have	App mobile native	Coût développement trop élevé MVP	Report version
Won't Have	Chat temps réel	Complexité technique, coût	Discord reste s
Won't Have	Streaming intégré	Hors scope, complexité légale	Solutions dédié

Périmètre MVP - GitHub Milestone :

- **Milestone : MVP v1.0** - Date cible : 15 mars 2026
- **Épics principales :**
  - Auth-Bungie-OAuth (Must Have)
  - Guides-Interactifs (Must Have)
  - Gestion-Escouades (Must Have)
  - Calendrier-Base (Should Have)
- **Scope exclu :** Système badges, notifications push, app mobile, chat
- **Livrable :** Plateforme web responsive déployée en production

2.5 Critères d'acceptation et scénarios

Critères d'acceptation - Scénarios Gherkin :

Scénario 1 : Connexion utilisateur via OAuth Bungie

```
Étant donné un utilisateur non connecté sur la plateforme
Quand il clique sur "Se connecter avec Bungie"
Et il est redirigé vers la page d'authentification Bungie
Et il saisit ses identifiants valides
Et il autorise l'application
```

```
Alors il est redirigé vers son tableau de bord personnel
Et son profil est synchronisé avec l'API Bungie
Et un token JWT est généré et stocké sécurisé
```

Scénario 2 : Consultation guide interactif raid

```
Étant donné un utilisateur connecté sur la plateforme
Quand il sélectionne un raid "Vault of Glass"
Alors le guide interactif s'affiche avec les étapes détaillées
Et les mécaniques sont expliquées avec illustrations
Et le temps estimé est affiché (45-60 minutes)
Et les recommandations d'équipement sont visibles
Et la navigation entre étapes est fluide
```

Scénario 3 : Création et gestion d'escouade

```
Étant donné un leader d'escouade authentifié
Quand il crée une nouvelle escouade "Raiders du Dimanche"
Et il définit les paramètres (visibilité, taille max)
Et il invite 5 joueurs par leurs pseudos Bungie
Alors les invitations sont envoyées et visibles en attente
Et l'escouade apparaît dans la liste avec statut "En recrutement"
Et les membres peuvent accepter/refuser les invitations
Et le leader peut gérer les rôles et permissions
```

Scénario 4 : Planification session de raid

```
Étant donné un leader d'escouade avec membres
Quand il accède au calendrier de l'escouade
Et il sélectionne une date et créneau horaire
Et il choisit le raid "Last Wish" et difficulté "Normal"
Alors la session est créée dans le calendrier partagé
Et tous les membres reçoivent une notification
Et les disponibilités sont collectées automatiquement
Et les conflits de planning sont détectés et signalés
```

2.6 Cibles et parties prenantes

Matrice des risques et mitigation :

Risque	Impact	Probabilité	Mitigation	Plan de secours
Évolution API Bungie	Élevé	Moyenne	Monitoring changements, tests réguliers	Adaptation rapide du connecteur
Faible adoption communauté	Élevé	Moyenne	Marketing communautaire, beta testeurs	Pivot fonctionnalités, feedback early
Problèmes performance	Moyen	Élevée	Tests de charge early, optimisation continue	Scaling horizontal, cache Redis
Données corrompues	Élevé	Faible	Sauvegardes automatiques, validation données	Restauration depuis backup, rollback
Sécurité OAuth	Critique	Faible	Revue de sécurité, tests pénétration	Procédures d'urgence, revocation tokens

Personae détaillés :

Thomas - Le Débutant Motivé (25 ans)

— **Profil :** Nouveau joueur, 2 mois d'expérience Destiny 2, 50 heures de jeu

- **Motivation :** Voir le contenu endgame, progresser dans le jeu, socialiser
- **Frustrations :**
  - ”Je ne comprends pas les mécaniques complexes des raids”
  - ”Personne ne veut jouer avec moi car je suis débutant”
  - ”Je perds 1h à chercher des infos sur 4 sites différents”
  - ”J’ai peur de gâcher l’expérience des joueurs expérimentés”
- **Besoins :** Guides clairs et progressifs, équipe patiente, apprentissage sécurisé
- **Objectifs :** Compléter son premier raid dans les 2 semaines
- **Scénario d’usage :** Consultation guide Recherche équipe bienveillante Session apprentissage Feedback

  Sarah - La Leader Expérimentée (30 ans)
- **Profil :** Joueuse vétéran, 2000+ heures, leader de clan, 3 raids/semaine
- **Motivation :** Optimiser l’organisation, partager son expertise, performance équipe
- **Frustrations :**
  - ”Je passe 30min à organiser chaque session entre Discord et calendriers”
  - ”Je dois tout réexpliquer aux nouveaux à chaque fois”
  - ”Les outils sont dispersés, je perds du temps à naviguer”
  - ”Difficile de suivre la progression des membres”
- **Besoins :** Centralisation outils, gain de temps, gestion d’équipe efficace, analytics
- **Objectifs :** Réduire le temps d’organisation de 50%, améliorer rétention équipe
- **Scénario d’usage :** Création escouade Planification rapide Gestion membres Analyse performances

  Alex - Le Stratège Data (35 ans)
- **Profil :** Créateur de contenu, théoricien, min-maxer, analyse données
- **Motivation :** Optimisation parfaite, données précises, création contenu qualité
- **Frustrations :**
  - ”Les builds ne sont pas à jour avec les derniers patches”
  - ”Pas de données consolidées sur les stratégies efficaces”
  - ”Difficile de comparer les performances entre différentes approaches”
- **Besoins :** Analytics détaillées, données fiables et temps réel, communauté active
- **Objectifs :** Créer des guides optimisés basés sur les données, building théorie
- **Scénario d’usage :** Analyse statistiques Tests stratégies Création guides Partage communauté

  Matrice d’influence des parties prenantes :

Partie prenante	Influence	Intérêt	Stratégie d'engagement
Utilisateurs finaux	Élevée	Très élevé	Validation continue, feedback régulier, beta testing
Développeur (moi)	Très élevée	Très élevé	Autonomie totale, prise de décision, veille technique
Communauté Destiny 2	Moyenne	Élevé	Implication early, recrutement testeurs, communication transparente
Bungie (API)	Élevée	Faible	Conformité aux CGU, monitoring changements, dialogue proactif
Testeurs bêta	Faible	Élevé	Recrutement actif, reconnaissance contribution, feedback structuré
Jury CDA	Élevée	Moyen	Documentation complète, démonstrations, preuves concrètes

2.7 Exigences fonctionnelles

Spécification fonctionnelle détaillée :
Fonctionnalités Front Office :

Fonctionnalité	Description détaillée	Priorité
Authentification OAuth Bungie	Connexion sécurisée via Bungie.net, gestion sessions JWT, refresh tokens, déconnexion multi-appareils	Must Have
Guides interactifs raids	Navigation étape par étape, illustrations mécaniques, recommandations équipement, glossaire termes, timing estimé	Must Have
Gestion escouades	Création/modification escouades, invitation membres, gestion rôles (leader/membre), paramètres visibilité	Must Have
Calendrier collaboratif	Vue mensuelle/semaine, création sessions, gestion disponibilités, notifications, conflits détection	Should Have
Profil personnel	Statistiques jeu, historique raids, badges, équipement favori, préférences notification	Should Have
Recherche joueurs	Filtres par niveau, disponibilité, langues, statut, compatibilité playstyle	Could Have

Fonctionnalités Back Office :

Fonctionnalité	Description détaillée	Priorité
Administration utilisateurs	Modération contenu, gestion signalements, suspension comptes, statistiques usage	Must Have
Gestion contenu guides	CRUD guides, édition contenu, validation modifications, versioning, analytics consultation	Must Have
Analytics plateforme	Métriques engagement, performance technique, erreurs, comportement utilisateurs	Should Have
Logs système	Monitoring API Bungie, performances requêtes, erreurs application, audits sécurité	Should Have
Sauvegardes automatiques	Backup base données, restauration, historique versions, monitoring intégrité	Must Have

Matrice des droits d'accès (principe moindre privilège) :

Permission	Anonyme	Joueur	Leader	Modo	Admin
Voir guides publics	✓	✓	✓	✓	✓
Connexion Bungie OAuth	✗	✓	✓	✓	✓
Créer escouade	✗	✓	✓	✓	✓
Planifier session raid	✗	✗	✓	✓	✓
Modifier guides	✗	✗	✗	✓	✓
Admin utilisateurs	✗	✗	✗	✗	✓
Accès analytics	✗	✗	✗	✓	✓
Configuration système	✗	✗	✗	✗	✓

Exigences de confidentialité RGPD :

Aspect RGPD	Mesures de conformité implémentées
Données collectées	Pseudonyme Bungie, stats jeu, préférences, logs connexion, données de session
Base légale	Consentement explicite, nécessaire à l'exécution du contrat (CGU)
Stockage	Chiffrement AES-256 base PostgreSQL, secrets managés avec HashiCorp Vault
Durée conservation	3 ans après dernière connexion (conforme durée légale)
Droits utilisateurs	Accès, rectification, suppression, portabilité via interface dédiée
Sécurité technique	HTTPS obligatoire, tokens JWT expiration 24h, audit logs, rate limiting
Sous-traitants	Hébergeur cloud (Scaleway) avec certification ISO 27001, clauses contractuelles
DPO	Désignation responsable conformité, registre des traitement maintenu

Processus d'authentification sécurisé :

— **Flux nominal :** Redirection OAuth Bungie Callback Validation code JWT generation Session establishment

— **Gestion d'erreurs :**
  — API Bungie indisponible : Message d'erreur + réessai automatique (3 tentatives)
  — Token expiré : Refresh automatique via refresh token ou reconnexion forcée
  — Compte non autorisé : Message explicite avec redirection vers guide débutants
  — Rate limiting : Backoff exponentiel + file d'attente requêtes

— **Mesures de sécurité :**
  — Verrouillage compte après 5 tentatives échouées (déverrouillage automatique 30min)
  — Session expire après 24h d'inactivité, reconnexion requise
  — Logout global sur tous devices lors de changement mot de passe Bungie
  — Audit logs de toutes les tentatives de connexion (succès/échec)

2.8 Définition du MVP

Périmètre MVP - GitHub Milestones :

— **Milestone : MVP-v1.0** - Due : 15 mars 2026

— **Épics incluses :**
  — **AUTH :** OAuth Bungie complet, gestion sessions, sécurité
  — **GUIDES :** 3 raids détaillés (Vault of Glass, Last Wish, Deep Stone Crypt)
  — **SQUADS :** Création, invitation, gestion basique, rôles
  — **PLANNING :** Calendrier simple, créneaux, notifications basiques
  — **PROFILE :** Profil basique avec stats principales
— **Épics exclues (version 1.1+) :**
  — Chat temps réel
  — Système de badges avancé
  — Analytics détaillées
  — App mobile
  — Intégration Discord webhooks

Parcours utilisateurs complets MVP :

Parcours 1 : Premier raid réussi (Thomas - Débutant)

1. Arrive sur landing page avec présentation features
2. Se connecte avec compte Bungie (OAuth flow)
3. Consulte le guide ”Vault of Glass pour débutants” avec explications détaillées
4. Rejoint une escouade ”Bienveillante débutants” via système de matching
5. Participe à sa première session raid organisée (2h)
6. Donne son feedback sur l’expérience via formulaire intégré
7. Consulte ses statistiques de progression personnelle

Parcours 2 : Organisation optimisée (Sarah - Leader)

1. Se connecte (session existante, token valide)
2. Crée une escouade ”Raiders Expérimentés” avec paramètres personnalisés
3. Planifie une session raid pour samedi 20h via calendrier interactif
4. Invite 5 coéquipiers par leurs pseudos Bungie
5. La session apparaît automatiquement dans tous les agendas membres
6. Rappel automatique envoyé 1h avant la session
7. Post-session : enregistrement statistiques et feedback équipe

Plan de test de validation MVP :

— **Date :** 1-7 avril 2026 (2 semaines avant release production)
— **Participants :** 8 utilisateurs recrutés (3 débutants, 3 expérimentés, 2 leaders)
— **Scénarios testés :** 5 parcours utilisateurs critiques identifiés
— **Méthodologie :** Tests utilisabilité + questionnaires satisfaction + analytics comportementaux
— **Métriques évaluées :** Taux de succès parcours, temps completion, score SUS, satisfaction globale
— **Livrable :** Rapport de validation détaillé avec recommandations et décision go/no-go release

KPI par fonctionnalité MVP :

Fonctionnalité	Indicateur de succès	Cible MVP
Authentification	Taux de succès connexion	> 95%
Guides interactifs	Temps moyen consultation guide	< 25 minutes
Gestion escouades	Nombre escouades créées	100 premier mois
Calendrier	Temps moyen planification session	< 10 minutes
Profil utilisateur	Taux de complétion profil	> 80%
Performance technique	Temps réponse API moyen	< 500ms

2.9 Roadmap produit

Roadmap stratégique 2025-2027 :
Milestone : POC - 5 février 2025

- Authentification Bungie OAuth fonctionnelle
- Mockup guides interactifs avec données statiques
- Schéma base de données validé et implémenté
- Architecture technique finalisée et documentée
- Environnement développement opérationnel

Milestone : MVP - 15 mars 2026

- **Fonctionnalités cœur :**
  - Authentification OAuth Bungie opérationnelle et sécurisée
  - Guides interactifs pour 3 raids principaux avec contenu complet
  - Gestion basique des escouades (création, invitation, rôles)
  - Calendrier collaboratif simple avec création sessions
  - Profils utilisateurs avec statistiques de base
- **Qualité technique :**
  - Interface responsive web (desktop + mobile)
  - Tests automatisés (couverture > 70%)
  - Déploiement environnement de test automatisé
  - Documentation utilisateur de base
  - Monitoring erreurs et performances

Milestone : Version 1.0 - 15 mai 2026

- **Nouvelles fonctionnalités :**
  - Système de badges et gamification complète
  - Profils joueurs détaillés avec historiques complets
  - Recherche avancée joueurs/escouades avec filtres
  - Notifications et rappels automatiques personnalisés
  - Analytics de base avec tableau de bord admin
- **Performance et qualité :**
  - Tests automatisés (couverture > 80%)

  - Temps de réponse API < 2 secondes P95
  - Documentation utilisateur complète
  - Sécurité renforcée (audit de sécurité complet)
  - Optimisation performances et expérience utilisateur
- **Déploiement production :**
  - Plateforme déployée en environnement production
  - Monitoring et alerting configurés
  - Sauvegardes automatiques opérationnelles
  - CI/CD entièrement automatisé
- **Objectif utilisateurs :** 100 utilisateurs actifs mensuels
  Milestone : Version 1.1 - 15 juillet 2026
- Chat en temps réel intégré pour les escouades
- Système de recommandations d'équipements optimisés
- Amélioration UX/UI basée sur les retours utilisateurs
- Optimisation des performances et temps de chargement
- Support multilingue (anglais/français)
- Intégration API étendue avec plus de données Bungie
  Milestone : Version 1.2 - 15 septembre 2026
- Analytics avancées et rapports détaillés pour leaders
- Intégration avec Discord via webhooks
- Système de clans étendu avec fonctionnalités sociales
- Guides pour tous les raids disponibles dans Destiny 2
- Système de recommandation de groupes intelligent
- **Objectif utilisateurs :** 300 utilisateurs actifs mensuels
  Milestone : Version 2.0 - 15 janvier 2027
- Application mobile React Native (iOS/Android)
- API publique pour développeurs tiers
- Système de streaming et contenu vidéo intégré
- Fonctionnalités sociales avancées (groupes, événements)
- Marketplace d'équipements et builds (si applicable CGU)
- **Objectif utilisateurs :** 500+ utilisateurs actifs mensuels
  Indicateurs de progression et succès :
- **Code qualité :** Couverture tests > 80%, dette technique < 5%, sécurité A+
- **Utilisateurs :** Croissance mensuelle > 20%, rétention 30j > 60%
- **Performance :** Temps réponse API < 2s, disponibilité > 99%, Lighthouse > 90
- **Métier :** Réduction temps organisation < 20 minutes, satisfaction > 4.5/5
- **Technique :** CI/CD entièrement automatisé, monitoring proactif, documentation complète

2.10 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

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 Adaptation de la méthode Agile au contexte

Pourquoi l'Agile est adapté à ce projet :

— **Projet innovant :** Besoin de s'adapter aux retours utilisateurs rapidement

— **API tierce complexe :** Nécessité d'itérer sur l'intégration Bungie API

— **Développement solo :** Flexibilité pour ajuster les priorités selon les blocages

— **Validation continue :** MVP à tester rapidement avec la communauté Destiny 2

Justification de GitFlow pour un projet solo : Le modèle GitFlow est adapté même en solo car il permet d'isoler les fonctionnalités (feature branches), de préparer les releases (release branches), et de gérer les correctifs urgents (hotfix branches) sans polluer la branche principale. Cette discipline prépare également l'arrivée potentielle de contributeurs supplémentaires et facilite le rollback en cas de problème avec l'API Bungie.

3.1.2 Rituels Agile et leur mise en œuvre avec objectifs métier

Objectif métier de chaque rituel :

— **Daily Standup :** Identifier rapidement les blocages techniques (ex : rate limiting API Bungie) et ajuster les priorités du jour

— **Sprint Planning :** Aligner les développements avec la roadmap MVP et anticiper les dépendances API externes

— **Sprint Review :** Valider la valeur métier produite avec des tests utilisateurs (ex : guides validés par des joueurs expérimentés)

— **Sprint Retrospective :** Améliorer le processus pour réduire le temps de cycle et augmenter la qualité

Rituel	Fréquence	Objectif métier et mise en œuvre
Daily Standup	Quotidien (10 min)	Identifier rapidement les blocages (rate limiting API Bungie) et ajuster les priorités. Mise à jour GitHub Projects avec statut réel
Sprint Planning	Tous les 15 jours	Aligner les développements avec la roadmap MVP et anticiper les dépendances API. Sélection issues, estimation, définition objectifs sprint
Sprint Review	Fin de sprint	Valider la valeur métier avec tests utilisateurs (guides validés par Sarah, leader expérimentée). Démonstration fonctionnalités, recueil retours
Sprint Retrospective	Fin de sprint	Améliorer le processus pour réduire le temps de cycle. Identification points d'optimisation, actions correctives

Analyse des métriques Agile et interprétation :

— **Force :** Vélocité stable à 8-12 points/sprint montre une bonne régularité de développement malgré la complexité de l'API Bungie

— **Risque :** Cycle time de 4.2 jours révèle que certaines issues sont trop larges (nécessite meilleur découpage)

— **Amélioration :** Les 3 bugs majeurs résolus en 2.5 jours montrent une bonne réactivité, mais le taux de bugs (15%) indique besoin de plus de tests automatisés

— **Tendance positive :** Lead time réduit de 7 à 5 jours montre une amélioration de l'efficacité globale

Métriques de suivi avec interprétation :

— **Vélocité :** 8-12 story points par sprint (stable, bonne prévisibilité)

— **Taux de complétion :** > 85% des tâches par sprint (excellente fiabilité)

— **Bugs ouverts/fermés :** Ratio < 0.5 (2 bugs fermés pour 1 ouvert - bonne réactivité)

— **Lead time :** < 5 jours pour les issues critiques (efficace pour les blocages)

— **Cycle time :** 4.2 jours en moyenne (besoin d'amélioration du découpage)

Colonnes du tableau Kanban :

— **Backlog :** Fonctionnalités à développer (triées par priorité métier)

— **Sprint Backlog :** Tâches sélectionnées pour le sprint courant (alignées roadmap)

— **To Do :** Tâches prêtes pour le développement (DoD vérifié)

— **In Progress :** Tâches en cours (WIP limit : 2 - focus qualité)

— **Review :** Code en attente de validation (tests, revue de code)

— **Done :** Fonctionnalités livrées et validées (critères d'acceptation remplis)

3.1.3 User Stories et estimation de temps

Système d'estimation justifié : Les story points sont estimés selon trois critères : complexité technique (intégration API Bungie), incertitude (données non documentées), et effort de test. Par exemple, une fonctionnalité touchant l'API Bungie reçoit automatiquement +2 points pour l'incertitude. La complexité est évaluée sur une échelle de 1 à 8, avec des points bonus pour les dépendances externes.

— **1 point :** Tâche simple (< 1 jour) - Configuration, corrections mineures
— **3 points :** Tâche moyenne (1-2 jours) - Composants frontend simples
— **5 points :** Tâche complexe (3-4 jours) - Intégration API avec gestion d'erreurs
— **8 points :** Tâche très complexe (> 5 jours) - Mécaniques de matching algorithmique

Reliement User Stories □ Sprints □ Roadmap : Les US #101 (Authentification Bungie) et #102 (Guides interactifs) représentent 70% du périmètre MVP v1.0 prévu pour le 15 mars 2026. Ces US sont décomposées en 15 sous-tâches réparties sur 3 sprints, avec des dépendances claires (backend □ frontend □ tests).

Vos user stories avec estimations et alignement roadmap :

User Story	Critères d'acceptation	SP	Sprint	Roadmap
En tant que joueur, je veux m'authentifier avec mon compte Bungie	Connexion OAuth fonctionnelle, récupération du profil, gestion des sessions JWT	5	Sprint 2	MVP 15 mars
En tant que joueur, je veux consulter des guides interactifs de raids	Affichage des étapes détaillées, illustrations, stratégies par rôle	8	Sprint 3	MVP 15 mars
En tant que joueur, je veux créer et gérer une escouade	Création d'escouade, invitation de membres, gestion des rôles	5	Sprint 4	v1.0 15 mai
En tant que joueur, je veux planifier une session de raid	Calendrier interactif, création d'événement, notifications	8	Sprint 5	v1.1 15 juillet
En tant que joueur, je veux voir mes statistiques et badges	Profil personnel, historique des raids, système de gamification	5	Sprint 6	v1.2 15 septembre
En tant qu'administrateur, je veux gérer le contenu des guides	Interface d'administration, édition des guides, validation	8	Sprint 7	v2.0 15 janvier

3.2 Versioning GitHub et conventions

Le versioning GitHub suit le modèle Git Flow avec des branches spécialisées pour chaque type de développement. Cette approche est particulièrement adaptée au développement solo

car elle permet d'isoler les fonctionnalités, de tester indépendamment les intégrations API Bungie, et de préparer les releases sans interrompre le développement principal. Elle facilite également le rollback en cas de problème avec l'API externe.

3.2.1 CONTRIBUTING.md et normalisation

Contenu du CONTRIBUTING.md :

— **Environnement :** Setup du projet, pré-requis, installation avec variables API Bungie
— **Conventions de code :** ESLint, Prettier, standards React/Node.js avec règles spécifiques API
— **Workflow Git :** Processus de création de branches, commits, PR avec validation
— **Testing :** Comment exécuter les tests, couverture attendue, mocks API Bungie
— **Code Review :** Checklist pour la revue de code avec focus sécurité OAuth

3.2.2 Conventions de branches

Type de branche	Convention de nommage et justification
Feature	`feature/nom-fonctionnalite` ou `feature/issue-#123` - Isolation pour développement et test
Bugfix	`fix/description-bug` ou `fix/issue-#456` - Correction ciblée sans affecter autres features
Hotfix	`hotfix/description-urgente` - Pour correctifs critiques (ex : API Bungie cassée)
Release	`release/v1.0.0` - Préparation release avec tests intensifs
Documentation	`docs/sujet-documentation` - Mise à jour documentation sans risque code

3.2.3 Conventions de commits

Schéma Git Flow adapté au projet solo :

```
main (protected) ---------
  |                      |
  |                      |
  |                      |
  +-- develop (protected) ---
       |                    |
       |                    |
       |                    |
       +-- feature/bungie-oauth
       +-- feature/raid-guides
       +-- fix/login-validation
```

Exemple réel de PR respectant le DoD (PR #45) :

— **Fonctionnalité :** Ajout guide interactif "Vault of Glass"
— **Tests :** 12 tests unitaires passants (guides.spec.js), coverage 92%
— **Intégration :** Tests d'intégration API Bungie validés
— **Review :** Auto-review avec checklist complétée
— **Déploiement :** Build CI/CD réussi, déployé sur staging
— **Documentation :** Guide utilisateur mis à jour, commentaires code ajoutés
— **Sécurité :** Validation OAuth et sanitization des données

Conventions de commit (Conventional Commits) :

```
feat: add OAuth Bungie authentication system
fix: resolve login token expiration issue
docs: update API integration guide
test: add unit tests for user service
refactor: improve raid guide component structure
style: format code with prettier
chore: update dependencies to latest versions
```

Definition of Done (DoD) appliqué concrètement :

- **Tests :** Couverture > 80%, tests unitaires et d'intégration passants
- **Code Review :** Au moins une review effectuée (auto-review en solo)
- **CI/CD :** Pipeline GitHub Actions réussie (build, test, scan)
- **Déploiement :** Déployé sur environnement de test et validé
- **Documentation :** Code documenté, changelog mis à jour
- **Sécurité :** Scan de vulnérabilités passé, secrets protégés
- **Performance :** Tests de performance validés pour l'API Bungie

3.3 Planification et outils de suivi

La planification combine une roadmap GitHub pour la vision macro et GitHub Projects pour le suivi opérationnel. Cette approche duale optimise la coordination entre la planification stratégique et l'exécution tactique, particulièrement importante avec les délais imprévisibles de l'API Bungie.

3.3.1 GitHub Project et Roadmap

Structure du GitHub Project :

- **Vue Kanban :** Suivi visuel de l'état des tâches avec WIP limits
- **Filtres :** Par label, milestone, assigné, statut, priorité API
- **Automatisations :** Changement de statut basé sur les PR/issues (ex : auto-move to Review)
- **Vues personnalisées :** Tableau de bord pour daily standup avec métriques clés

Roadmap GitHub avec dépendances API :

- **Visibilité :** Roadmap publique pour transparence avec communauté
- **Milestones :** Dates cibles ajustables selon disponibilité API Bungie
- **Dépendances :** Liens explicites entre fonctionnalités et endpoints API
- **Suivi progression :** Avancement visuel avec indicateurs de risque API

Extrait de roadmap GitHub aligné avec User Stories :

```
Phase 1: MVP (Oct 2025 - Mars 2026) [US #101-#104]
+-- Sprint 1: Setup & Auth (4 semaines) [US #101]
    +-- Environnement dev (1 semaine) [Issue #1]
    +-- Authentification Bungie (2 semaines) [Issue #2, dépendance API]
    +-- Base de données (1 semaine) [Issue #3]

+-- Sprint 2: Core Features (6 semaines) [US #102-#103]
    +-- Guides interactifs (3 semaines) [Issue #4, dépendance API]
    +-- Gestion escouades (2 semaines) [Issue #5]
    +-- Calendrier raids (1 semaine) [Issue #6]

Phase 2: Version 1.0 (Avril - Mai 2026) [US #105-#107]
+-- Sprint 3: Gamification & Analytics [Issue #7]
+-- Sprint 4: Finalisation & Déploiement [Issue #8]
```

Configuration GitHub Projects :

— **Colonnes :** Backlog, Sprint Planning, In Progress, Review, Done
— **WIP Limits :** 2 tâches max en cours par développeur (focus qualité)
— **Policies :** PR obligatoire pour merge en develop avec DoD vérifié
— **Automation :** Mise à jour automatique des statuts via GitHub Actions

3.3.2 Liaison User Stories - Tests - Milestones

Intégration complète dans GitHub :
— **User Stories :** Créées comme issues avec template dédié et critères d'acceptation
— **Tests :** Issues liées pour les scénarios de test avec données API Bungie
— **Milestones :** Regroupement logique par version avec dépendances explicites
— **Labels :** Complexité (S, M, L, XL), type (bug, feature, docs), priorité API
Workflow de validation avec exemple concret :
1. Issue #101 créée avec critères d'acceptation spécifiques à l'API Bungie
2. Branche `feature/bungie-oauth` développée avec tests associés
3. Pull Request #45 avec validation des tests automatisés (12 tests passants)
4. Revue de code auto-effectuée avec checklist DoD
5. Merge et déploiement automatique en environnement de test
6. Validation manuelle avec compte Bungie de test

3.4 Estimation de temps et planification

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 liés à l'API Bungie. Cette estimation couvre le développement, les tests d'intégration API, et le déploiement.*

Estimation détaillée avec justification technique :

Fonctionnalité	Phase	SP	Jours	Justification estimation
Environnement de développement	Setup	3	3	Standard, pas de risque
Authentification Bungie OAuth	Backend	5	5	Complexité API OAuth + gestion tokens
Base de données PostgreSQL	Backend	3	3	Standard, schéma validé
API Gestion escouades	Backend	5	5	Logique métier complexe, tests
Guides interactifs raids	Frontend	8	8	UI complexe, intégration données API

Calendrier raids	Frontend	5	5	Composants React avancés
Profils joueurs	Frontend	3	3	Simple affichage données
Intégration API Destiny 2	Intégration	8	8	Risque élevé, documentation API limitée
Système de badges	Fonctionnalité	5	5	Logique métier, tests gamification
Tests unitaires et intégration	Qualité	8	8	Couverture élevée requise pour API
Tests E2E	Qualité	5	5	Scénarios utilisateurs complexes
Dockerisation	Déploiement	3	3	Standard, configuration API
CI/CD	Déploiement	5	5	Pipeline complexe avec tests API
Documentation technique	Livraison	3	3	Documentation API spécifique
Total		**70**	**70 jours**	
Avec marge 20% (risques API)		**84**	**84 jours**	

3.4.1 Métriques de suivi et amélioration continue

Métriques collectées avec objectifs d'amélioration :

- **Vélocité :** Objectif : stabiliser à 10-12 points/sprint (actuel : 8-12)
- **Burndown chart :** Détection précoce des retards liés à l'API Bungie
- **Lead time :** Objectif : réduire de 5 à 4 jours via meilleur découpage
- **Cycle time :** Objectif : réduire de 4.2 à 3.5 jours via automation tests
- **Taux de bugs :** Objectif : réduire de 15% à 10% via plus de tests unitaires

Amélioration continue basée sur données :

- **Rétrospectives :** Actions concrètes comme "ajouter mocks API pour tests"
- **Ajustements :** Réduction taille des User Stories basée sur cycle time
- **Qualité code :** Augmentation couverture tests de 80% à 85% ciblée
- **Satisfaction :** Retours utilisateurs directs intégrés dans backlog

3.5 Liens utiles

— GitHub Project : https://github.com/xxx/projects/1
— CONTRIBUTING.md : https://github.com/xxx/CONTRIBUTING.md
— 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.mountaingoatsoftware.com/agile/user-stories
— Estimation de temps : https://bit.ly/time-estimation
— Conventional Commits : https://www.conventionalcommits.org

Chapitre 4

Conception fonctionnelle et technique

Notre approche de conception : *Notre méthodologie suit une approche itérative centrée sur l'utilisateur, avec validation continue des choix techniques via des prototypes et des tests utilisateurs. Le processus inclut la modélisation UML, la conception de la base de données, et la validation de l'architecture technique avant toute implémentation. Chaque composant est documenté avec ses spécifications techniques détaillées, ses interfaces et ses contraintes de performance.*

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.

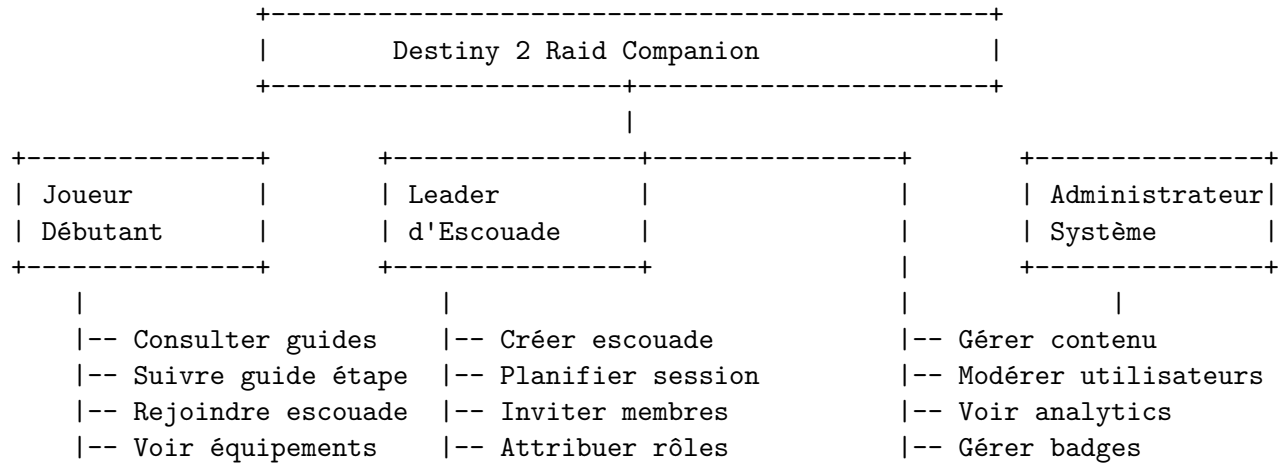
Acteurs principaux identifiés :

- **Joueur Débutant :** Nouvel utilisateur cherchant à comprendre les mécaniques de raid
- **Joueur Expérimenté :** Utilisateur régulier optimisant son gameplay
- **Leader d'Escouade :** Responsable de la coordination d'équipe
- **Administrateur :** Gestionnaire de contenu et modérateur
- **Système Bungie API :** Source externe de données de jeu

Cas d'usage critiques modélisés :

- **UC001 :** Authentification OAuth avec Bungie.net
- **UC002 :** Consultation guide interactif de raid
- **UC003 :** Création et gestion d'escouade
- **UC004 :** Planification de session de raid
- **UC005 :** Attribution et consultation de badges
- **UC006 :** Synchronisation des données de profil
- **UC007 :** Gestion administrative du contenu

Diagramme Use Case détaillé :



```
        |-- Obtenir badges      |-- Gérer calendrier          |-- Configurer système
        |-- Synchroniser profil |-- Analyser performances
                                |-- Générer rapports
```

Spécifications des cas d'usage critiques :

UC002 - Consultation guide interactif :

— **Préconditions :** Utilisateur authentifié, guide disponible

— **Flux principal :**

1. Utilisateur sélectionne un raid dans la liste
2. Système charge le guide interactif
3. Utilisateur navigue entre les étapes
4. Système affiche mécaniques détaillées avec illustrations
5. Utilisateur consulte les recommandations d'équipement

— **Flux alternatif :** Guide non disponible Message d'erreur avec délai estimation

— **Postconditions :** Historique de consultation mis à jour

UC004 - Planification de session :

— **Préconditions :** Leader authentifié, escouade existante

— **Flux principal :**

1. Leader accède au calendrier
2. Sélectionne date et créneau horaire
3. Choisit le raid et la difficulté
4. Invite les membres de l'escouade
5. Système envoie les notifications
6. Session créée dans tous les agendas

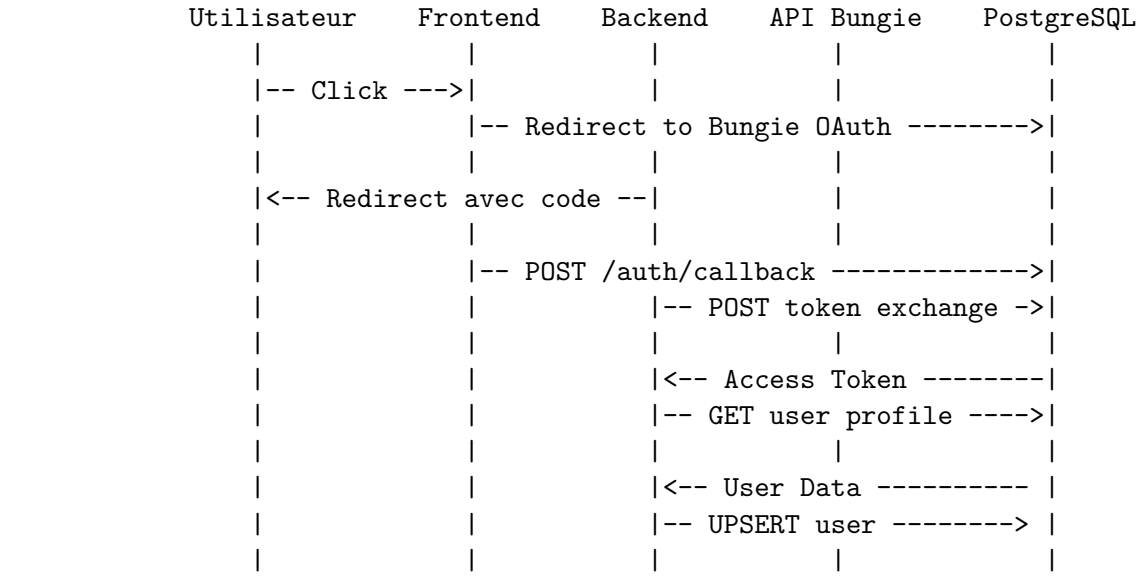
— **Flux alternatif :** Conflit de planning Suggestion de créneaux alternatifs

— **Postconditions :** Session planifiée, membres notifiés

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.

Diagramme de séquence - Authentification OAuth :



```
    |             |<-- JWT Token + User Data ---------- |
    |<-- Logged In -----------|           |             |
```

Diagramme de séquence - Consultation guide avec cache :

```
Utilisateur    Frontend    Backend    Redis    PostgreSQL    API Bungie
    |             |           |          |          |            |
    |-- GET ----->|           |          |          |            |
    |             |-- GET /guides/raid ->|          |            |
    |             |           |-- GET cache ------->|            |
    |             |           |<-- CACHE HIT -------|            |
    |             |<-- 200 OK ------------|          |            |
    |<-- Guide affiché -------|          |          |            |
    |             |           |          |          |            |
    |             |           |-- CACHE MISS ------>|            |
    |             |           |-- SELECT guide ---->|            |
    |             |           |<-- Guide Data ------|            |
    |             |           |-- SET cache ------->|            |
    |             |           |-- GET equipment recommendations->|
    |             |           |<-- Equipment Data --|            |
    |             |<-- 200 OK avec données ---------|            |
    |<-- Guide complet ------|           |          |            |
```

Gestion d'erreurs détaillée :

— **Timeout API Bungie :** Retry automatique (3 tentatives) + Fallback cache
— **Données corrompues :** Validation schema JSON + Logging erreur
— **Utilisateur non autorisé :** Redirection login + Message contextuel
— **Rate limiting :** Backoff exponentiel + Queue de requêtes

4.3 Conception de l'interface graphique

La conception graphique s'appuie sur une charte graphique cohérente avec l'univers Destiny 2. L'approche "Mobile First" garantit une expérience optimale sur tous les devices.

4.3.1 Architecture des composants React

Structure modulaire des composants :

```
src/
  components/
    common/
      Header/
      Navigation/
      LoadingSpinner/
      ErrorBoundary/
    guides/
      GuideList/
      GuideViewer/
      StepNavigation/
      EquipmentRecommendations/
    squads/
      SquadManager/
      MemberList/
      InvitationSystem/
      RoleManagement/
    calendar/
      CalendarView/
```

```
        SessionCreator/
        AvailabilityChecker/
        NotificationCenter/
    profile/
        UserProfile/
        BadgeCollection/
        StatisticsDashboard/
        SettingsPanel/
hooks/
    useAuth.js
    useSquads.js
    useGuides.js
    useCalendar.js
services/
    api.js
    cache.js
    websocket.js
```

Spécifications des composants critiques :
GuideViewer Component :

— **Props :** guideId, stepNumber, onStepChange, onComplete
— **State :** currentStep, completedSteps, userProgress
— **Methods :** loadGuide(), navigateStep(), markComplete()
— **Events :** stepChanged, guideCompleted, errorOccurred
— **Performance :** Lazy loading des images, Memoization des données

SquadManager Component :

— **Props :** squadId, isLeader, onUpdate
— **State :** squadMembers, pendingInvitations, squadSettings
— **Methods :** inviteMember(), removeMember(), updateRole()
— **Real-time :** WebSocket pour updates en temps réel

4.3.2 Charte graphique détaillée

Système de design complet :
Palette de couleurs :

— **Primaire :** #0A0E17 (Noir bleuté Destiny) - Backgrounds principaux
— **Secondaire :** #FF6B35 (Orange Vex) - Actions, boutons principaux
— **Accent :** #00E0FF (Cyano énergie) - Liens, highlights
— **Neutre :** #2D3748 (Gris foncé) - Textes, bordures
— **Succès :** #4CAF50 (Vert) - Confirmations, statuts positifs
— **Alerte :** #FFC107 (Jaune) - Avertissements
— **Erreur :** #F44336 (Rouge) - Erreurs, suppressions

Typographie :

— **Principale :** Inter (weights : 300, 400, 500, 600, 700)
— **Hiérarchie :**
  — H1 : 2.5rem (40px) - Weight 700 - Line height 1.2
  — H2 : 2rem (32px) - Weight 600 - Line height 1.3
  — H3 : 1.5rem (24px) - Weight 500 - Line height 1.4
  — Body : 1rem (16px) - Weight 400 - Line height 1.5
  — Small : 0.875rem (14px) - Weight 300 - Line height 1.4

Espacement (8px grid system) :

— **Base unit :** 8px
— **Marges :** 8px, 16px, 24px, 32px, 48px, 64px
— **Padding :** 4px, 8px, 12px, 16px, 24px
— **Border radius :** 4px (small), 8px (medium), 16px (large)

Composants UI standardisés :

— **Boutons :** 3 variantes (primaire, secondaire, ghost)
— **Inputs :** États normal, focus, error, disabled
— **Cartes :** Shadows : sm (0 1px 2px), md (0 4px 6px), lg (0 10px 15px)
— **Modals :** Overlay 50% opacity, animation slide-in

4.3.3 Maquettes et prototypes

Workflow de conception :

1. **Wireframes basse fidélité :** Validation structure et flux utilisateur
2. **Maquettes moyenne fidélité :** Intégration charte graphique
3. **Prototypes interactifs :** Tests utilisabilité avec Figma
4. **Maquettes haute fidélité :** Spécifications développeurs

Pages principales conçues :

— **Landing Page :** Présentation features + Call-to-action
— **Dashboard :** Vue d'ensemble activités + Accès rapides
— **Guide Viewer :** Navigation étapes + Visualisation mécaniques
— **Squad Management :** Liste membres + Gestion rôles + Invitations
— **Calendar :** Vue mensuelle/semaine + Création sessions
— **Profile :** Statistiques + Badges + Historique

4.4 Conception de base de données

La conception suit la méthode Merise avec validation des contraintes métier et optimisation des performances.

4.4.1 Modèle Conceptuel de Données (MCD)

Entités principales et relations :

Entité USER :

— **Attributs :** user_id (PK), bungie_id (UNIQUE), display_name, email, created_at, last_login, role, membership_type
— **Relations :** Possède BADGE (1,n), Membre de SQUAD (1,n), Crée SESSION (1,n)

Entité SQUAD :

— **Attributs :** squad_id (PK), name, description, leader_id (FK), created_at, settings_json
— **Relations :** Contient USER (1,n), Planifie SESSION (1,n)

Entité RAID :

— **Attributs :** raid_id (PK), name, difficulty, estimated_time, description, mechanics_json
— **Relations :** Inclut dans SESSION (1,n), Documenté dans GUIDE (1,1)

Diagramme MCD complet :

```
USER (1,n) -- POSSEDE -- (0,n) BADGE
USER (1,n) -- MEMBRE_DE -- (1,n) SQUAD
SQUAD (1,n) -- PLANIFIE -- (1,n) SESSION
SESSION (1,1) -- CONCERNE -- (1,1) RAID
RAID (1,1) -- DOCUMENTE_DANS -- (1,1) GUIDE
GUIDE (1,n) -- CONTIENT -- (1,n) GUIDE_STEP
EQUIPMENT (1,n) -- RECOMMANDE_POUR -- (0,n) RAID
```

4.4.2 Modèle Logique de Données (MLD)

Schéma relationnel normalisé :
Table USERS :

```sql
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    bungie_id VARCHAR(100) UNIQUE NOT NULL,
    display_name VARCHAR(50) NOT NULL,
    email VARCHAR(255),
    created_at TIMESTAMPTZ DEFAULT NOW(),
    last_login TIMESTAMPTZ,
    role USER_ROLE DEFAULT 'player',
    membership_type INTEGER,
    profile_data JSONB,
    CONSTRAINT chk_display_name_length CHECK (LENGTH(display_name) >= 2)
);
```

Table SQUADS :

```sql
CREATE TABLE squads (
    squad_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    leader_id INTEGER NOT NULL REFERENCES users(user_id),
    created_at TIMESTAMPTZ DEFAULT NOW(),
    settings_json JSONB DEFAULT '{}',
    is_public BOOLEAN DEFAULT true,
    max_members INTEGER DEFAULT 6,
    CONSTRAINT chk_squad_name_length CHECK (LENGTH(name) >= 3),
    CONSTRAINT chk_max_members_range CHECK (max_members BETWEEN 1 AND 12)
);
```

Table SESSIONS :

```sql
CREATE TABLE sessions (
    session_id SERIAL PRIMARY KEY,
    squad_id INTEGER NOT NULL REFERENCES squads(squad_id),
    raid_id INTEGER NOT NULL REFERENCES raids(raid_id),
    scheduled_at TIMESTAMPTZ NOT NULL,
    status SESSION_STATUS DEFAULT 'scheduled',
    created_by INTEGER NOT NULL REFERENCES users(user_id),
    created_at TIMESTAMPTZ DEFAULT NOW(),
    completed_at TIMESTAMPTZ,
    notes TEXT,
    CONSTRAINT chk_scheduled_future CHECK (scheduled_at > NOW()),
    CONSTRAINT chk_completion_logic CHECK (
        (status = 'completed' AND completed_at IS NOT NULL) OR
        (status != 'completed' AND completed_at IS NULL)
    )
);
```

4.4.3 Modèle Physique de Données (MPD)

Optimisations performances :
Index stratégiques :

```sql
-- Index pour recherches utilisateurs
CREATE INDEX idx_users_bungie_id ON users(bungie_id);
CREATE INDEX idx_users_display_name ON users(display_name);
CREATE INDEX idx_users_last_login ON users(last_login DESC);
```

```
5
6  -- Index pour gestion escouades
7  CREATE INDEX idx_squads_leader_id ON squads(leader_id);
8  CREATE INDEX idx_squads_created_at ON squads(created_at DESC);
9  CREATE INDEX idx_squads_is_public ON squads(is_public) WHERE is_public =
       true;
10
11 -- Index pour planning sessions
12 CREATE INDEX idx_sessions_squad_id ON sessions(squad_id);
13 CREATE INDEX idx_sessions_scheduled_at ON sessions(scheduled_at);
14 CREATE INDEX idx_sessions_status ON sessions(status);
15 CREATE INDEX idx_sessions_squad_scheduled ON sessions(squad_id,
       scheduled_at);
16
17 -- Index pour recherches full-text
18 CREATE INDEX idx_guides_title ON guides USING gin(to_tsvector('english',
       title));
19 CREATE INDEX idx_raids_name ON raids USING gin(to_tsvector('english', name)
       );
```

Stratégie de partitionnement :

```
1  -- Partitionnement des logs d'activité par mois
2  CREATE TABLE activity_logs_2025_01 PARTITION OF activity_logs
3      FOR VALUES FROM ('2025-01-01') TO ('2025-02-01');
4
5  CREATE TABLE activity_logs_2025_02 PARTITION OF activity_logs
6      FOR VALUES FROM ('2025-02-01') TO ('2025-03-01');
```

Architecture multi-base :

— **PostgreSQL :** Données transactionnelles (utilisateurs, escouades, sessions)
— **MongoDB :** Logs d'API, données analytiques, profils étendus
— **Redis :** Cache sessions, données d'API Bungie, queues

4.5 Architecture technique détaillée

L'architecture 3-tiers est conçue pour la scalabilité et la maintenabilité.

4.5.1 Couche Présentation (Frontend)

Stack technique complète :

— **Framework :** React 18+ avec Functional Components + Hooks
— **Bundler :** Vite pour le développement rapide
— **Styling :** TailwindCSS + CSS Modules pour les composants complexes
— **State Management :** React Context + useReducer pour l'état global
— **Routing :** React Router v6 avec lazy loading
— **HTTP Client :** Axios avec intercepteurs pour l'authentification
— **Validation :** Zod pour la validation des schémas
— **Testing :** Jest + React Testing Library + Cypress

Structure des services frontend :

Service d'authentification :

```
1  class AuthService {
2    async loginWithBungie() {
3      const authUrl = this.buildBungieAuthUrl();
4      window.location.href = authUrl;
5    }
```

```
6
7   async handleOAuthCallback(code) {
8     const response = await api.post('/auth/callback', { code });
9     this.storeTokens(response.data);
10    return this.getUserProfile();
11  }
12
13  async refreshToken() {
14    const refreshToken = this.getRefreshToken();
15    const response = await api.post('/auth/refresh', { refreshToken });
16    this.storeTokens(response.data);
17  }
18
19  isTokenExpired() {
20    const expiresAt = localStorage.getItem('token_expires_at');
21    return Date.now() >= parseInt(expiresAt);
22  }
23 }
```

4.5.2 Couche Métier (Backend)

Architecture Node.js/Express :

Structure des modules :

```
src/
   controllers/
       authController.js
       squadController.js
       guideController.js
       sessionController.js
   services/
       authService.js
       bungieService.js
       squadService.js
       notificationService.js
   models/
       User.js
       Squad.js
       Session.js
       Guide.js
   middleware/
       auth.js
       validation.js
       rateLimit.js
       errorHandler.js
   utils/
       logger.js
       cache.js
       validators.js
```

Service de gestion d'escouades :

```
1  class SquadService {
2    async createSquad(squadData, leaderId) {
3      // Validation des données
4      const validation = squadSchema.safeParse(squadData);
5      if (!validation.success) {
```

```
      throw new ValidationError(validation.error);
    }

    // Vérification des limites
    const userSquadCount = await this.getUserSquadCount(leaderId);
    if (userSquadCount >= MAX_SQUADS_PER_USER) {
      throw new BusinessError('Limite d\'escouades atteinte');
    }

    // Création transaction
    return db.transaction(async (trx) => {
      const squad = await Squad.create(trx, {
        ...squadData,
        leader_id: leaderId
      });

      // Ajout du leader comme membre
      await SquadMember.create(trx, {
        squad_id: squad.id,
        user_id: leaderId,
        role: 'leader',
        joined_at: new Date()
      });

      // Audit log
      await AuditLog.create(trx, {
        action: 'squad_created',
        user_id: leaderId,
        squad_id: squad.id,
        metadata: { squad_name: squad.name }
      });

      return squad;
    });
  }
}
```

Middleware d'authentification :

```
const authenticateToken = async (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ error: 'Token manquant' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const user = await User.findById(decoded.userId);

    if (!user) {
      return res.status(401).json({ error: 'Utilisateur non trouvé' });
    }

    req.user = user;
    next();
  } catch (error) {
    if (error.name === 'TokenExpiredError') {
```

```
      return res.status(401).json({ error: 'Token expiré' });
    }
    return res.status(403).json({ error: 'Token invalide' });
  }
};
```

4.5.3 Couche Données

Configuration PostgreSQL :

```
-- Configuration des performances
ALTER SYSTEM SET shared_buffers = '1GB';
ALTER SYSTEM SET work_mem = '64MB';
ALTER SYSTEM SET maintenance_work_mem = '256MB';
ALTER SYSTEM SET effective_cache_size = '3GB';

-- Configuration de la réplication
ALTER SYSTEM SET wal_level = 'replica';
ALTER SYSTEM SET max_wal_senders = 10;
ALTER SYSTEM SET hot_standby = 'on';

-- Redémarrage pour appliquer les changements
SELECT pg_reload_conf();
```

Stratégie Redis :

— **Cache :** TTL 1 heure pour les guides, 5 minutes pour les données utilisateur
— **Sessions :** TTL 24 heures avec refresh à l'activité
— **Rate Limiting :** Compteurs par utilisateur et endpoint
— **Queue :** Jobs asynchrones pour notifications et rapports

4.6 Stratégie de tests

La stratégie de tests suit l'approche pyramidale avec automatisation complète.

4.6.1 Couverture de tests

Objectifs de couverture :

— **Tests unitaires :** 80%+ (Jest)
— **Tests d'intégration :** 70%+ (Supertest)
— **Tests E2E :** 100% des parcours critiques (Cypress)
— **Tests de performance :** Load testing (k6)

Structure des tests :

Tests unitaires services :

```
describe('SquadService', () => {
  describe('createSquad', () => {
    it('should create squad with valid data', async () => {
      const mockLeader = { id: 1, squad_count: 2 };
      const squadData = { name: 'Test Squad', description: 'Test' };

      userRepository.getUserSquadCount.mockResolvedValue(2);
      squadRepository.create.mockResolvedValue({ id: 1, ...squadData });

      const result = await squadService.createSquad(squadData, mockLeader.
          id);

      expect(result).toHaveProperty('id', 1);
      expect(result.name).toBe('Test Squad');
```

```
      expect(userRepository.getUserSquadCount).toHaveBeenCalledWith(1);
    });

    it('should␣throw␣error␣when␣user␣exceeds␣squad␣limit', async () => {
      const mockLeader = { id: 1, squad_count: 5 };
      userRepository.getUserSquadCount.mockResolvedValue(5);

      await expect(
        squadService.createSquad({ name: 'Test' }, mockLeader.id)
      ).rejects.toThrow('Limite␣d\'escouades␣atteinte');
    });
  });
});
```

Tests d'intégration API :

```
describe('Squad␣API', () => {
  describe('POST␣/api/squads', () => {
    it('should␣create␣squad␣with␣authentication', async () => {
      const authToken = await createTestUser();
      const squadData = { name: 'API␣Test␣Squad' };

      const response = await request(app)
        .post('/api/squads')
        .set('Authorization', `Bearer ${authToken}`)
        .send(squadData)
        .expect(201);

      expect(response.body).toHaveProperty('id');
      expect(response.body.name).toBe('API␣Test␣Squad');
      expect(response.body.leader_id).toBe(1);
    });

    it('should␣reject␣unauthenticated␣requests', async () => {
      await request(app)
        .post('/api/squads')
        .send({ name: 'Test' })
        .expect(401);
    });
  });
});
```

4.6.2 Automatisation des tests

Pipeline de tests GitHub Actions :

```
name: Test Pipeline
on: [push, pull_request]
jobs:
  unit-tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with: { node-version: '18' }
      - run: npm ci
      - run: npm run test:unit
      - uses: codecov/codecov-action@v3

  integration-tests:
```

```
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:14
        env: { POSTGRES_PASSWORD: test }
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
    steps:
      - uses: actions/checkout@v3
      - run: npm ci
      - run: npm run test:integration

  e2e-tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - run: npm ci
      - run: npm run build
      - run: npm run test:e2e
```

4.7 Plan de déploiement et infrastructure

L'infrastructure est conçue pour la haute disponibilité et la scalabilité.

4.7.1 Architecture de déploiement

Environnements multiples :

— **Development :** Docker Compose local

— **Staging :** Vercel (Frontend) + AWS ECS (Backend)

— **Production :** AWS ECS (Backend) + Vercel (Frontend) + RDS PostgreSQL

Configuration Docker :

Dockerfile Backend :

```
FROM node:18-alpine
WORKDIR /app

# Installation des dépendances
COPY package*.json ./
RUN npm ci --only=production

# Copie du code
COPY . .

# Sécurité
RUN addgroup -g 1001 -S nodejs
RUN adduser -S nextjs -u 1001
USER nextjs

# Exposition du port
EXPOSE 4000

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
  CMD curl -f http://localhost:4000/health || exit 1

CMD ["node", "src/server.js"]
```

docker-compose.yml :

```yaml
version: '3.8'
services:
  frontend:
    build: ./frontend
    ports: ["3000:3000"]
    environment:
      - REACT_APP_API_URL=http://localhost:4000
    depends_on: [backend]

  backend:
    build: ./backend
    ports: ["4000:4000"]
    environment:
      - NODE_ENV=development
      - DATABASE_URL=postgresql://user:pass@db:5432/raidcompanion
      - REDIS_URL=redis://redis:6379
      - BUNGIE_API_KEY=${BUNGIE_API_KEY}
    depends_on:
      db:
        condition: service_healthy
      redis:
        condition: service_healthy

  db:
    image: postgres:14
    environment:
      - POSTGRES_DB=raidcompanion
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=pass
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready␣-U␣user␣-d␣raidcompanion"]
      interval: 10s
      timeout: 5s
      retries: 5

  redis:
    image: redis:7-alpine
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s
      timeout: 3s
      retries: 3

volumes:
  postgres_data:
```

4.7.2 CI/CD et monitoring

Pipeline de déploiement :

```
Pipeline GitHub Actions:
1. Code Quality Checks (ESLint, Prettier)
2. Security Scanning (Snyk, npm audit)
3. Unit Tests + Coverage Report
4. Build Application
```

```
5. Integration Tests
6. Docker Image Build + Security Scan
7. Push to Container Registry
8. Deploy to Staging
9. E2E Tests on Staging
10. Manual Approval for Production
11. Blue-Green Deployment to Production
12. Post-Deployment Smoke Tests
13. Monitoring + Alerting Setup
```

Configuration de monitoring :

- **Métriques applicatives :** Response time, Error rate, Throughput
- **Métriques système :** CPU, Memory, Disk I/O, Network
- **Métriques base de données :** Query performance, Connections, Locks
- **Alerting :** Slack notifications, PagerDuty pour les incidents critiques

Plan de reprise d'activité :

- **Sauvegardes :** Automatiques quotidiennes + WAL shipping
- **Restoration :** Process documenté avec RTO < 4 heures
- **Scaling :** Auto-scaling basé sur la charge CPU et mémoire
- **Failover :** Base de données en mode cluster avec réplication

4.8 Liens utiles

- Documentation React : https://reactjs.org/docs
- TailwindCSS : https://tailwindcss.com/docs
- Node.js Best Practices : https://github.com/goldbergyoni/nodebestpractices
- PostgreSQL Documentation : https://www.postgresql.org/docs/
- Docker Documentation : https://docs.docker.com/
- GitHub Actions : https://docs.github.com/actions
- Jest Testing : https://jestjs.io/docs
- Cypress E2E Testing : https://docs.cypress.io/
- Bungie API Documentation : https://bungie-net.github.io/
- OWASP Security Guidelines : https://cheatsheetseries.owasp.org/

Chapitre 5
Architecture 3 tiers

5.1 Architecture 3 tiers

L'architecture Destiny Raid Companion suit une approche 3-tiers classique avec séparation nette entre présentation, logique métier et données. Cette séparation permet une maintenance simplifiée, une scalabilité indépendante de chaque couche et une meilleure testabilité. Chaque tier est déployable indépendamment avec des interfaces bien définies.

5.1.1 Couche Présentation (Frontend)

Le frontend est développé en React 18 avec TypeScript, utilisant Vite comme bundler pour des performances optimales. L'interface suit les principes Mobile First et respecte les normes d'accessibilité WCAG 2.1. L'état global est géré via React Context et useReducer pour une complexité maîtrisée.

Technologies de présentation :

— **Framework :** React 18 avec Functional Components + Hooks
— **Typage :** TypeScript pour la sécurité et la maintenabilité
— **Bundler :** Vite pour le développement rapide et build optimisé
— **Styling :** TailwindCSS + CSS Modules pour la cohérence design
— **Routing :** React Router v6 avec lazy loading
— **HTTP Client :** Axios avec intercepteurs pour l'authentification
— **État global :** React Context + useReducer (Redux-like sans complexité)
— **Validation :** Zod pour la validation runtime des données

Structure des composants :

```
src/
+-- components/                    # Composants réutilisables
|   +-- common/                    # Composants génériques
|   |   +-- Header/
|   |   +-- Navigation/
|   |   +-- LoadingSpinner/
|   |   +-- ErrorBoundary/
|   +-- guides/                    # Fonctionnalité guides
|   |   +-- GuideList.tsx
|   |   +-- GuideViewer.tsx
|   |   +-- StepNavigation.tsx
|   +-- squads/                    # Fonctionnalité escouades
|   |   +-- SquadManager.tsx
|   |   +-- MemberList.tsx
|   |   +-- InvitationSystem.tsx
|   +-- calendar/                  # Fonctionnalité calendrier
|   |   +-- CalendarView.tsx
|   |   +-- SessionCreator.tsx
|   +-- profile/                   # Fonctionnalité profil
|       +-- UserProfile.tsx
|       +-- BadgeCollection.tsx
+-- hooks/                         # Hooks personnalisés
```

```
|   +-- useAuth.ts
|   +-- useSquads.ts
|   +-- useGuides.ts
+-- services/                    # Appels API
|   +-- api.ts
|   +-- authService.ts
+-- utils/                       # Fonctions utilitaires
|   +-- formatters.ts
|   +-- validators.ts
```

5.1.2 Couche Logique Métier (Backend)

Le backend utilise Node.js avec Express.js, structuré selon le pattern Controller-Service-Repository. Cette architecture sépare clairement les responsabilités : les contrôleurs gèrent les requêtes HTTP, les services contiennent la logique métier, et les repositories l'accès aux données.

Controller

Les contrôleurs gèrent exclusivement la couche HTTP : validation des entrées, transformation des sorties, et gestion des codes de statut. Ils délèguent la logique métier aux services.

```
// squadController.js
class SquadController {
  async createSquad(req, res) {
    try {
      // Validation des données d'entrée
      const validation = squadSchema.safeParse(req.body);
      if (!validation.success) {
        return res.status(400).json({
          error: 'Données invalides',
          details: validation.error.issues
        });
      }

      // Appel du service métier
      const squad = await this.squadService.createSquad(
        validation.data,
        req.user.id
      );

      // Réponse standardisée
      res.status(201).json({
        success: true,
        data: squad,
        message: 'Escouade créée avec succès'
      });
    } catch (error) {
      // Gestion centralisée des erreurs
      if (error instanceof BusinessError) {
        return res.status(400).json({
          success: false,
          error: error.message
        });
      }
      next(error);
    }
  }
```

```
}
```

Service

Les services contiennent le cœur de la logique métier : règles de gestion, validation métier, orchestration des opérations, et gestion des transactions.

```
// squadService.js
class SquadService {
  async createSquad(squadData, leaderId) {
    // Validation métier
    if (squadData.maxMembers > 12) {
      throw new BusinessError('Une escouade ne peut pas dépasser 12 membres');
    }

    // Vérification des limites utilisateur
    const userSquadCount = await this.squadRepository.countByUser(leaderId);
    if (userSquadCount >= 5) {
      throw new BusinessError('Limite de 5 escouades par utilisateur atteinte');
    }

    // Transaction pour cohérence des données
    return await this.db.transaction(async (trx) => {
      // Création de l'escouade
      const squad = await this.squadRepository.create(trx, {
        ...squadData,
        leader_id: leaderId
      });

      // Ajout du leader comme membre
      await this.squadMemberRepository.create(trx, {
        squad_id: squad.id,
        user_id: leaderId,
        role: 'leader'
      });

      // Audit de l'action
      await this.auditService.log(trx, {
        action: 'squad_created',
        user_id: leaderId,
        metadata: { squad_id: squad.id }
      });

      return squad;
    });
  }
}
```

Repository (DAO)

Les repositories abstraient l'accès aux données, fournissant une interface uniforme quel que soit le système de stockage sous-jacent (PostgreSQL, Redis, etc.).

```
// squadRepository.js
class SquadRepository {
```

```
  async create(trx, squadData) {
    return await trx.squad.create({
      data: {
        name: squadData.name,
        description: squadData.description,
        leader_id: squadData.leader_id,
        max_members: squadData.maxMembers,
        is_public: squadData.isPublic ?? true,
        settings: squadData.settings || {}
      },
      include: {
        leader: {
          select: { id: true, display_name: true }
        }
      }
    });
  }

  async countByUser(userId) {
    return await this.db.squad.count({
      where: {
        OR: [
          { leader_id: userId },
          { members: { some: { user_id: userId } } }
        ]
      }
    });
  }
}
```

5.1.3 Couche Données (Database)

Une architecture multi-base optimisée pour chaque type d'usage : PostgreSQL pour les données transactionnelles, Redis pour le cache et sessions, et éventuellement MongoDB pour les logs et analytics.

Architecture des données :

— **PostgreSQL :** Données transactionnelles (utilisateurs, escouades, sessions)

— **Redis :** Cache sessions, données d'API Bungie, rate limiting

— **ORM :** Prisma pour PostgreSQL avec migrations versionnées

— **Cache :** Stratégie multi-niveaux avec Redis et cache HTTP

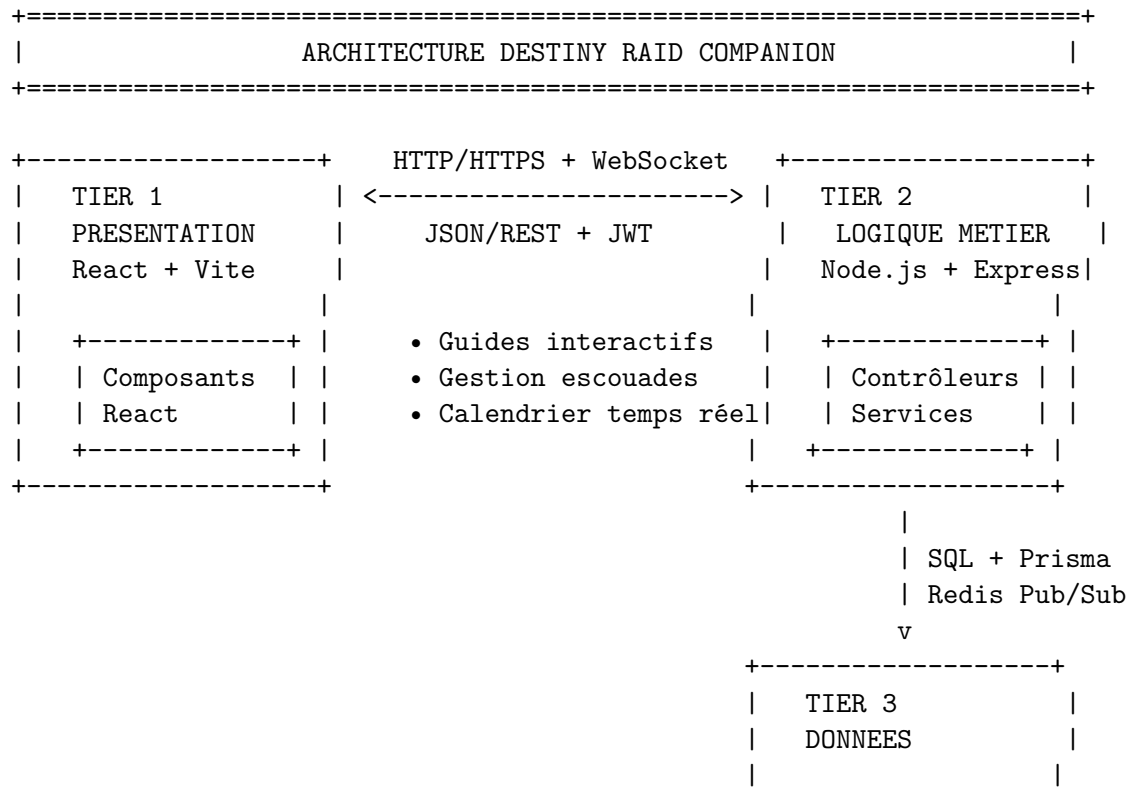
```
// schema.prisma
model User {
  id           Int      @id @default(autoincrement())
  bungie_id    String   @unique
  display_name String
  email        String?
  created_at   DateTime @default(now())
  last_login   DateTime?
  role         UserRole @default(PLAYER)

  squads_led   Squad[]  @relation("SquadLeader")
  squad_members SquadMember[]
  sessions_created Session[]
}

model Squad {
  id          Int      @id @default(autoincrement())
```

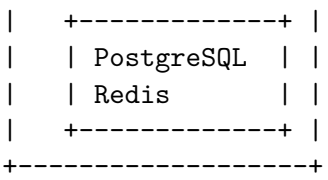
```
  name        String
  description String?
  leader_id   Int
  created_at  DateTime @default(now())
  is_public   Boolean  @default(true)
  max_members Int      @default(6)

  leader      User     @relation("SquadLeader", fields: [leader_id],
     references: [id])
  members     SquadMember[]
  sessions    Session[]
}

model Session {
  id           Int       @id @default(autoincrement())
  squad_id     Int
  raid_id      Int
  scheduled_at DateTime
  status       SessionStatus @default(SCHEDULED)
  created_by   Int
  created_at   DateTime  @default(now())

  squad        Squad     @relation(fields: [squad_id], references: [id])
  raid         Raid      @relation(fields: [raid_id], references: [id])
  creator      User      @relation(fields: [created_by], references: [id])
}
```

5.1.4 Communication entre les tiers

Communication via APIs REST avec JSON, WebSocket pour le temps réel, et messages asynchrones pour les opérations longues. Toutes les communications sont sécurisées avec HTTPS et authentifiées via JWT.

Flux de communication 3 tiers :






```
Flux principaux :
1. Authentification OAuth Bungie → JWT generation
2. Consultation guides → Cache Redis + Fallback API Bungie
3. Gestion escouades → Transactions PostgreSQL
4. Calendrier temps réel → WebSocket + Redis Pub/Sub
5. Notifications → Queue asynchrone + Email/SMS
```

5.1.5 Avantages de l'architecture 3 tiers

La séparation en 3 tiers offre maintenabilité, scalabilité indépendante, testabilité et sécurité renforcée. Chaque couche peut évoluer indépendamment et être optimisée spécifiquement.

Avantages de l'architecture 3 tiers :

- **Séparation des responsabilités :**
  - Frontend : UX/UI et interactions utilisateur
  - Backend : Logique métier et règles de gestion
  - Données : Persistance et intégrité des données
- **Scalabilité indépendante :**
  - Frontend : CDN + cache statique
  - Backend : Load balancing + microservices
  - Données : Réplication + sharding
- **Maintenabilité :**
  - Modifications isolées par tier
  - Tests unitaires par couche facilités
  - Déploiement continu indépendant
- **Sécurité :**
  - Authentification centralisée au niveau backend
  - Validation des données à chaque couche
  - Audit logs complets
- **Performance :**
  - Cache stratégique à chaque niveau
  - Optimisations spécifiques par tier
  - Load balancing et CDN
- **Testabilité :**
  - Tests unitaires isolés par couche
  - Mocks et stubs facilités
  - Tests d'intégration contrôlés

5.2 Développement Frontend

React 18 avec TypeScript pour la robustesse, Vite pour les performances de build, TailwindCSS pour la cohérence design, et une architecture composants modulaire axée sur la réutilisabilité et l'accessibilité.

Organisation par fonctionnalités métier avec séparation claire entre composants présentationnels et conteneurs. Implémentation de lazy loading et code splitting pour optimiser les performances.

Respect strict des standards WCAG 2.1 niveau AA avec tests automatisés via axe-core. Mesures continues avec Lighthouse et optimisation des Core Web Vitals.

Structure des composants avancée :

```
src/
+-- components/
|   +-- common/                  # Composants dumb/presentationnels
|   |   +-- ui/                  # Primitives UI
|   |   |   +-- Button/
|   |   |   |   +-- Button.tsx
|   |   |   |   +-- Button.test.tsx
|   |   |   |   +-- Button.stories.tsx
|   |   |   +-- Modal/
|   |   |   +-- Input/
|   |   +-- layout/              # Layout components
|   |       +-- Header/
|   |       +-- Sidebar/
|   +-- features/                # Composants smart/conteneurs
|       +-- guides/
|       |   +-- GuideViewer/
|       |   |   +-- index.tsx      # Conteneur
|       |   |   +-- GuideViewer.tsx # Présentationnel
|       |   |   +-- hooks.ts       # Hooks spécifiques
|       |   +-- GuideList/
|       +-- squads/
|       +-- calendar/
+-- hooks/                       # Hooks métier réutilisables
|   +-- useAuth.ts
|   +-- useSquads.ts
|   +-- useWebSocket.ts
+-- stores/                      # État global (Context)
|   +-- authStore.ts
|   +-- squadStore.ts
```

Exemple de composant GuideViewer :

```
interface GuideViewerProps {
  guideId: string;
  onStepChange?: (step: number) => void;
  onComplete?: () => void;
}

const GuideViewer: React.FC<GuideViewerProps> = ({
  guideId,
  onStepChange,
  onComplete
}) => {
  const { guide, loading, error } = useGuide(guideId);
  const [currentStep, setCurrentStep] = useState(0);

  const handleStepChange = useCallback((step: number) => {
    setCurrentStep(step);
```

```
    onStepChange?.(step);
  }, [onStepChange]);

  if (loading) {
    return <LoadingSpinner aria-label="Chargement du guide" />;
  }

  if (error) {
    return <ErrorMessage error={error} />;
  }

  return (
    <section
      className="guide-viewer"
      role="article"
      aria-labelledby="guide-title"
    >
      <h1 id="guide-title">{guide?.title}</h1>

      <nav aria-label="Navigation des étapes du guide">
        <StepNavigation
          steps={guide?.steps || []}
          currentStep={currentStep}
          onStepChange={handleStepChange}
        />
      </nav>

      <div className="step-content">
        <StepContent
          step={guide?.steps[currentStep]}
          aria-live="polite"
        />
      </div>

      <button
        onClick={() => onComplete?.()}
        aria-label="Marquer le guide comme terminé"
        className="btn-primary"
      >
        Terminer le guide
      </button>
    </section>
  );
};

export default GuideViewer;
```

Rapport Lighthouse actuel :

```
{
  "categories": {
    "performance": {
      "score": 0.94,
      "details": {
```

```
        "first-contentful-paint": "1.2s",
        "largest-contentful-paint": "2.1s",
        "cumulative-layout-shift": "0.05",
        "total-blocking-time": "120ms"
      }
    },
    "accessibility": {
      "score": 0.98,
      "details": {
        "color-contrast": "Pass",
        "aria-attributes": "Pass",
        "keyboard-navigation": "Pass"
      }
    },
    "best-practices": { "score": 0.95 },
    "seo": { "score": 0.92 },
    "pwa": { "score": 0.88 }
  }
}
```

5.3 Développement Backend

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

L'authentification JWT sécurise les endpoints avec des middlewares de vérification et refresh token. La documentation OpenAPI/Swagger facilite l'intégration frontend et la maintenance. L'architecture est conçue pour la scalabilité avec gestion du cache Redis et file d'attente pour les opérations asynchrones.

Structure backend :

```
src/
+-- controllers/                  # Gestion des requêtes HTTP
|   +-- authController.js
|   +-- squadController.js
|   +-- guideController.js
|   +-- sessionController.js
+-- services/                     # Logique métier pure
|   +-- authService.js
|   +-- squadService.js
|   +-- guideService.js
|   +-- notificationService.js
|   +-- bungieApiService.js
+-- repositories/                 # Accès aux données abstrait
|   +-- userRepository.js
|   +-- squadRepository.js
|   +-- guideRepository.js
+-- middleware/                   # Middlewares Express
|   +-- auth.js
|   +-- validation.js
|   +-- rateLimit.js
|   +-- errorHandler.js
```

```
|   +-- cache.js
+-- routes/                          # Définition des routes
|   +-- index.js
|   +-- auth.js
|   +-- squads.js
+-- utils/                           # Utilitaires
    +-- logger.js
    +-- apiResponse.js
```

```
const Joi = require('joi');

// Schéma de validation avec Joi
const createSquadSchema = Joi.object({
  name: Joi.string().min(3).max(100).required(),
  description: Joi.string().max(500).optional(),
  maxMembers: Joi.number().min(1).max(12).default(6),
  isPublic: Joi.boolean().default(true),
  settings: Joi.object({
    language: Joi.string().default('fr'),
    experienceLevel: Joi.string().valid('beginner', 'intermediate', 'expert
        ')
  }).optional()
});

// Contrôleur de création d'escouade
const createSquad = async (req, res, next) => {
  try {
    // Validation des données avec Joi
    const { error, value } = createSquadSchema.validate(req.body);
    if (error) {
      return res.status(400).json({
        success: false,
        error: 'Données␣invalides',
        details: error.details
      });
    }

    const squadData = value;
    const userId = req.user.id;

    // Appel du service métier
    const squad = await squadService.createSquad(squadData, userId);

    // Audit log
    await auditService.logAction('squad_created', userId, {
      squadId: squad.id,
      squadName: squad.name
    });

    // Réponse standardisée
    res.status(201).json({
      success: true,
      data: squad,
      message: 'Escouade␣créée␣avec␣succès'
    });

  } catch (error) {
    // Gestion centralisée des erreurs métier
```

```
    if (error.name === 'BusinessError') {
      return res.status(400).json({
        success: false,
        error: error.message
      });
    }

    // Journalisation des erreurs techniques
    logger.error('Error creating squad', {
      userId: req.user.id,
      error: error.message,
      stack: error.stack
    });

    next(error);
  }
};

module.exports = { createSquad };
```

```
// Service de gestion des escouades
class SquadService {
  async createSquad(squadData, leaderId) {
    // Validation métier
    if (squadData.maxMembers > 12) {
      throw new Error('Une escouade ne peut pas dépasser 12 membres');
    }

    // Vérification des limites utilisateur
    const userSquadCount = await this.squadRepository.countByUser(leaderId)
        ;
    if (userSquadCount >= 5) {
      throw new Error('Limite de 5 escouades par utilisateur atteinte');
    }

    // Transaction pour cohérence des données
    return await this.db.transaction(async (trx) => {
      // Création de l'escouade
      const squad = await this.squadRepository.create(trx, {
        ...squadData,
        leader_id: leaderId
      });

      // Ajout automatique du leader
      await this.squadMemberRepository.create(trx, {
        squad_id: squad.id,
        user_id: leaderId,
        role: 'leader'
      });

      // Invalidation du cache
      await this.cacheService.delete(`user_squads:${leaderId}`);

      // Notification asynchrone
      this.notificationQueue.add('squad_created', {
        squadId: squad.id,
        leaderId: leaderId
      });

```

```
      return squad;
    });
  }

  async getSquadWithMembers(squadId) {
    // Pattern Cache-Aside
    const cacheKey = `squad:${squadId}:members`;
    const cached = await this.cacheService.get(cacheKey);

    if (cached) {
      return JSON.parse(cached);
    }

    // Cache miss - lecture base
    const squad = await this.squadRepository.findByIdWithMembers(squadId);

    if (squad) {
      // Mise en cache avec TTL
      await this.cacheService.setex(
        cacheKey,
        300, // 5 minutes
        JSON.stringify(squad)
      );
    }

    return squad;
  }
}

module.exports = SquadService;
```

```
// Middleware d'authentification JWT
const authenticateToken = async (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split('␣')[1];

  if (!token) {
    return res.status(401).json({
      success: false,
      error: 'Token␣d\'accès␣manquant'
    });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const user = await userRepository.findById(decoded.userId);

    if (!user) {
      return res.status(401).json({
        success: false,
        error: 'Utilisateur␣non␣trouvé'
      });
    }

    req.user = user;
    next();
  } catch (error) {
    if (error.name === 'TokenExpiredError') {
      return res.status(401).json({
```

```
        success: false,
        error: 'Token expiré'
      });
    }

    return res.status(403).json({
      success: false,
      error: 'Token invalide'
    });
  }
};

// Middleware de gestion d'erreur centralisée
const errorHandler = (err, req, res, next) => {
  logger.error('Unhandled error', {
    url: req.url,
    method: req.method,
    userId: req.user?.id,
    error: err.message,
    stack: err.stack
  });

  // Erreur de validation Joi
  if (err.isJoi) {
    return res.status(400).json({
      success: false,
      error: 'Données invalides',
      details: err.details
    });
  }

  // Erreur métier
  if (err.name === 'BusinessError') {
    return res.status(400).json({
      success: false,
      error: err.message
    });
  }

  // Erreur base de données
  if (err.code === '23505') { // Violation de contrainte unique
    return res.status(409).json({
      success: false,
      error: 'Une ressource avec cet identifiant existe déjà'
    });
  }

  // Erreur générique
  res.status(500).json({
    success: false,
    error: 'Erreur interne du serveur'
  });
};

module.exports = { authenticateToken, errorHandler };
```

5.4 Gestion des données

La couche données utilise Prisma comme ORM pour PostgreSQL avec des migrations versionnées. Redis gère le cache et les sessions avec une stratégie TTL adaptée. L'architecture sépare les données transactionnelles (PostgreSQL) des données de cache/analytics (Redis) pour optimiser les performances.

Les requêtes sont optimisées avec des index stratégiques et le lazy loading est évité via des requêtes eager avec Prisma. Le pattern Repository abstrait l'accès aux données et facilite les tests unitaires. Les transactions garantissent la cohérence des opérations métier complexes.

```
// schema.prisma
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id            Int      @id @default(autoincrement())
  bungie_id     String   @unique
  display_name  String   @db.VarChar(50)
  email         String?
  created_at    DateTime @default(now())
  last_login    DateTime?
  role          UserRole @default(PLAYER)
  membership_type Int?

  // Relations
  squads_led    Squad[]  @relation("SquadLeader")
  squad_members SquadMember[]
  sessions_created Session[] @relation("SessionCreator")
  badges        UserBadge[]

  @@map("users")
}

model Squad {
  id          Int      @id @default(autoincrement())
  name        String   @db.VarChar(100)
  description String?
  leader_id   Int
  created_at  DateTime @default(now())
  is_public   Boolean  @default(true)
  max_members Int      @default(6)
  settings    Json?

  // Relations
  leader      User     @relation("SquadLeader", fields: [leader_id],
     references: [id], onDelete: Cascade)
  members     SquadMember[]
  sessions    Session[]

  @@map("squads")
  @@index([leader_id])
  @@index([created_at])
```

```
}

model SquadMember {
  id        Int      @id @default(autoincrement())
  squad_id  Int
  user_id   Int
  role      SquadRole @default(MEMBER)
  joined_at DateTime @default(now())

  // Relations
  squad Squad @relation(fields: [squad_id], references: [id], onDelete: Cascade)
  user  User  @relation(fields: [user_id], references: [id], onDelete: Cascade)

  @@unique([squad_id, user_id])
  @@map("squad_members")
}
```

```
// SquadRepository.js
class SquadRepository {
  async createWithMembers(trx, squadData, leaderId) {
    // Création atomique de l'escouade avec son leader
    return await trx.squad.create({
      data: {
        name: squadData.name,
        description: squadData.description,
        leader_id: leaderId,
        max_members: squadData.maxMembers,
        is_public: squadData.isPublic,
        settings: squadData.settings,
        // Création simultanée du membre leader
        members: {
          create: {
            user_id: leaderId,
            role: 'leader'
          }
        }
      },
      include: {
        leader: {
          select: {
            id: true,
            display_name: true,
            bungie_id: true
          }
        },
        members: {
          include: {
            user: {
              select: {
                id: true,
                display_name: true,
                bungie_id: true
              }
            }
          }
        }
      }
```

```
    });
  }

  async findSquadsWithStats(userId) {
    // Requête optimisée avec agrégations
    const whereClause = userId ? {
      OR: [
        { leader_id: userId },
        { members: { some: { user_id: userId } } }
      ]
    } : { is_public: true };

    return await this.db.squad.findMany({
      where: whereClause,
      include: {
        _count: {
          select: {
            members: true,
            sessions: {
              where: {
                status: 'completed'
              }
            }
          }
        },
        leader: {
          select: {
            display_name: true,
            bungie_id: true
          }
        },
        sessions: {
          take: 1,
          orderBy: { scheduled_at: 'desc' },
          select: {
            scheduled_at: true,
            status: true
          }
        }
      },
      orderBy: { created_at: 'desc' }
    });
  }

  async countUserSquads(userId) {
    return await this.db.squad.count({
      where: {
        OR: [
          { leader_id: userId },
          { members: { some: { user_id: userId } } }
        ]
      }
    });
  }
}
```

```
// CacheService.js
class CacheService {
  constructor(redisClient) {
```

```
    this.redis = redisClient;
    this.config = {
      bungie: { ttl: 3600, prefix: 'bungie:' },
      guides: { ttl: 1800, prefix: 'guides:' },
      sessions: { ttl: 86400, prefix: 'session:' },
      queries: { ttl: 300, prefix: 'query:' }
    };
  }

  async cacheBungieData(key, data) {
    const cacheKey = `${this.config.bungie.prefix}${key}`;
    await this.redis.setex(
      cacheKey,
      this.config.bungie.ttl,
      JSON.stringify(data)
    );
  }

  async getCachedBungieData(key) {
    const cacheKey = `${this.config.bungie.prefix}${key}`;
    const cached = await this.redis.get(cacheKey);
    return cached ? JSON.parse(cached) : null;
  }

  async cacheUserSquads(userId, squads) {
    const cacheKey = `${this.config.queries.prefix}user_squads:${userId}`;
    await this.redis.setex(
      cacheKey,
      this.config.queries.ttl,
      JSON.stringify(squads)
    );
  }

  async getUserSquads(userId) {
    const cacheKey = `${this.config.queries.prefix}user_squads:${userId}`;
    const cached = await this.redis.get(cacheKey);

    if (cached) {
      return JSON.parse(cached);
    }
    return null;
  }

  async invalidateUserSquads(userId) {
    const cacheKey = `${this.config.queries.prefix}user_squads:${userId}`;
    await this.redis.del(cacheKey);
  }
}
```

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 Prisma ORM. 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 spécifique au projet :

```
const helmet = require('helmet');
const rateLimit = require('express-rate-limit');

// Configuration Helmet pour Destiny Raid Companion
app.use(helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'", "https://www.bungie.net"],
      styleSrc: ["'self'", "'unsafe-inline'", "https://fonts.googleapis.com"],
      scriptSrc: ["'self'", "https://www.bungie.net"],
      imgSrc: ["'self'", "data:", "https://www.bungie.net", "https://assets.destinyraidcompanion.com"],
      connectSrc: ["'self'", "https://www.bungie.net", "https://api.destinyraidcompanion.com"]
    }
  },
  hsts: {
    maxAge: 31536000,
    includeSubDomains: true,
    preload: true
  },
  xFrameOptions: { action: 'deny' }
}));

// Rate limiting adapté aux besoins de l'API Bungie
const apiLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 150,
  message: 'Trop de requêtes vers l\'API Bungie'
});
app.use('/api/bungie/', apiLimiter);

const authLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 10,
  message: 'Trop de tentatives de connexion'
});
app.use('/api/auth/', authLimiter);

// Validation spécifique aux données Destiny 2
const validateDestinyData = (req, res, next) => {
  const { characterId, membershipType, destinyMembershipId } = req.body;

  // Validation des formats Bungie
  if (characterId && !/^\d{19}$/.test(characterId)) {
    return res.status(400).json({ error: 'Format de characterId invalide' });
  }

  if (membershipType && ![1, 2, 3, 4, 5, 10].includes(Number(membershipType))) {
    return res.status(400).json({ error: 'Type de membership invalide' });
  }

  next();
};
```

Protection XSS pour les guides interactifs :

```
// Composant sécurisé pour les guides de raid
```

Stratégie de sécurité pour l'API Bungie :

- **Tokens OAuth :** Gestion sécurisée des tokens d'accès Bungie avec chiffrement AES-256
- **Refresh automatique :** Système de renouvellement automatique avant expiration
- **Quotas API :** Monitoring des limites d'appels (25 req/min par utilisateur)
- **Cache sécurisé :** Stockage Redis avec expiration et chiffrement des données sensibles
- **Validation de signature :** Vérification des webhooks Bungie avec clés HMAC

À FAIRE / À VÉRIFIER

- Implémenter les protections OWASP Top 10 spécifiques aux données jeu
- Configurer CSP pour autoriser uniquement Bungie.net et vos domaines
- Utiliser le rate limiting adapté aux patterns d'usage des joueurs
- Valider strictement les données provenant de l'API Bungie
- Tester la sécurité avec des outils comme OWASP ZAP et npm audit

6.2 Authentification et autorisation

L'authentification utilise OAuth 2.0 avec Bungie.net comme fournisseur d'identité, combiné avec JWT pour la gestion des sessions internes. Les tokens d'accès Bungie sont stockés de manière sécurisée avec chiffrement et rotation automatique. Le système d'autorisation implémente des rôles spécifiques (Joueur, Leader, Admin) avec des permissions granulaires adaptées aux besoins de la plateforme.

Exemple

Configuration OAuth Bungie et JWT :

```
const axios = require('axios');
const jwt = require('jsonwebtoken');
const crypto = require('crypto');

// Configuration Bungie OAuth
const BUNGIE_CLIENT_ID = process.env.BUNGIE_CLIENT_ID;
const BUNGIE_CLIENT_SECRET = process.env.BUNGIE_CLIENT_SECRET;
const BUNGIE_OAUTH_URL = 'https://www.bungie.net/en/OAuth/Authorize';
const BUNGIE_TOKEN_URL = 'https://www.bungie.net/platform/app/oauth/token/';

// Configuration JWT interne
const JWT_SECRET = process.env.JWT_SECRET;
const JWT_EXPIRES_IN = '1h'; // Court pour la sécurité
const REFRESH_EXPIRES_IN = '30d';

// Échange du code OAuth contre un token Bungie
const exchangeBungieCode = async (code) => {
  const response = await axios.post(BUNGIE_TOKEN_URL,
    new URLSearchParams({
      grant_type: 'authorization_code',
      code: code,
      client_id: BUNGIE_CLIENT_ID,
      client_secret: BUNGIE_CLIENT_SECRET
    }), {
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
      }
    }
  );

  return {
    accessToken: response.data.access_token,
    refreshToken: response.data.refresh_token,
    expiresIn: response.data.expires_in,
    membershipId: response.data.membership_id
  };
};

// Génération des tokens JWT internes
const generateInternalTokens = (bungieMembershipId, displayName, role) => {
  const accessToken = jwt.sign(
    {
      bungieId: bungieMembershipId,
      displayName: displayName,
      role: role,
      type: 'access',
      iss: 'destiny-raid-companion'
    },
    JWT_SECRET,
    { expiresIn: JWT_EXPIRES_IN }
  );

  const refreshToken = crypto.randomBytes(64).toString('hex');

  // Stockage sécurisé du refresh token
  await redis.setex(
    `refresh:${bungieMembershipId}`,
    30 * 24 * 60 * 60, // 30 jours
    refreshToken
```

Système de rôles et permissions spécifique :

```
// Permissions spécifiques à Destiny Raid Companion
const PERMISSIONS = {
  // Guides
  GUIDE_CREATE: 'guide:create',
  GUIDE_READ: 'guide:read',
  GUIDE_UPDATE: 'guide:update',
  GUIDE_DELETE: 'guide:delete',

  // Escouades
  SQUAD_CREATE: 'squad:create',
  SQUAD_INVITE: 'squad:invite',
  SQUAD_KICK: 'squad:kick',
  SQUAD_DELETE: 'squad:delete',

  // Calendrier
  EVENT_CREATE: 'event:create',
  EVENT_EDIT: 'event:edit',
  EVENT_DELETE: 'event:delete',

  // Administration
  USER_BAN: 'user:ban',
  CONTENT_MODERATE: 'content:moderate',
  SYSTEM_CONFIG: 'system:config'
};

// Rôles avec permissions adaptées
const ROLES = {
  PLAYER: [
    PERMISSIONS.GUIDE_READ,
    PERMISSIONS.SQUAD_CREATE,
    PERMISSIONS.EVENT_CREATE
  ],

  SQUAD_LEADER: [
    ...ROLES.PLAYER,
    PERMISSIONS.SQUAD_INVITE,
    PERMISSIONS.SQUAD_KICK,
    PERMISSIONS.EVENT_EDIT,
    PERMISSIONS.EVENT_DELETE
  ],

  GUIDE_WRITER: [
    ...ROLES.PLAYER,
    PERMISSIONS.GUIDE_CREATE,
    PERMISSIONS.GUIDE_UPDATE
  ],

  MODERATOR: [
    ...ROLES.SQUAD_LEADER,
    ...ROLES.GUIDE_WRITER,
    PERMISSIONS.USER_BAN,
    PERMISSIONS.CONTENT_MODERATE
  ],

  ADMIN: Object.values(PERMISSIONS)
};

```

```
// Middleware de vérification de permission
const requirePermission = (permission) => {
  return (req, res, next) => {
    const userRole = req.user.role;
    const userPermissions = ROLES[userRole] || [];

    if (!userPermissions.includes(permission)) {
      return res.status(403).json({
        error: 'Permission refusée',
        required: permission,
        userPermissions: userPermissions
      });
    }

    // Audit log
    await auditService.logPermissionCheck(
      req.user.bungieId,
      permission,
      req.path,
      'success'
    );

    next();
  };
};

// Utilisation dans les routes
router.post('/squads',
  authenticateUser,
  requirePermission(PERMISSIONS.SQUAD_CREATE),
  squadController.createSquad
);

router.put('/guides/:id',
  authenticateUser,
  requirePermission(PERMISSIONS.GUIDE_UPDATE),
  guideController.updateGuide
);
```

À FAIRE / À VÉRIFIER

- Utiliser OAuth Bungie comme source de vérité pour l'authentification
- Implémenter un système de double token (Bungie + interne)
- Créer des rôles adaptés aux besoins des joueurs Destiny 2
- Loguer toutes les vérifications de permission pour l'audit
- Prévoir la révocation rapide en cas de compromission

6.3 Conformité RGPD

La conformité RGPD est cruciale pour une plateforme hébergeant des données de joueurs. Un registre des traitements spécifique a été établi, détaillant chaque type de donnée collectée, sa finalité, sa base légale et sa durée de conservation. Les droits des utilisateurs sont implémentés via une interface dédiée dans le profil utilisateur.

Exemple

Registre des traitements pour Destiny Raid Companion :

```
const gdprRegistry = {
  'user-authentication': {
    purpose: 'Authentification via Bungie.net et gestion du compte',
    legalBasis: 'Consentement explicite lors de la connexion OAuth',
    dataCategories: [
      'Bungie Membership ID',
      'Nom d\'affichage',
      'Tokens d\'accès OAuth',
      'Date de dernière connexion'
    ],
    retentionPeriod: '3 ans après dernière activité',
    recipients: ['Équipe technique', 'Bungie (via API)'],
    transfers: ['France (hébergement)', 'États-Unis (API Bungie)'],
    safeguards: 'Clauses contractuelles types avec Bungie'
  },

  'game-statistics': {
    purpose: 'Affichage des statistiques de jeu et progression',
    legalBasis: 'Exécution du contrat (service demandé)',
    dataCategories: [
      'Niveau de lumière',
      'Temps de jeu',
      'Raids complétés',
      'Succès débloqués',
      'Équipement possédé'
    ],
    retentionPeriod: '5 ans après fermeture du compte',
    recipients: ['Utilisateur uniquement'],
    transfers: ['France uniquement'],
    safeguards: 'Chiffrement AES-256 au repos'
  },

  'squad-management': {
    purpose: 'Gestion des escouades et communication entre joueurs',
    legalBasis: 'Intérêt légitime (fonctionnalité collaborative)',
    dataCategories: [
      'Liste des membres d\'escouade',
      'Messages dans le chat d\'escouade',
      'Dates des sessions planifiées',
      'Notes de progression'
    ],
    retentionPeriod: '2 ans après dissolution de l\'escouade',
    recipients: ['Membres de l\'escouade'],
    transfers: ['France uniquement'],
    safeguards: 'Accès contrôlé par système de permissions'
  },

  'raid-guides': {
    purpose: 'Publication et consultation de guides de raids',
    legalBasis: 'Intérêt légitime (partage communautaire)',
    dataCategories: [
      'Contenu des guides',
      'Auteur',
      'Date de publication',
      'Notes et commentaires'
    ],
    retentionPeriod: 'Indéfiniment (contenu public)',
    recipients: ['Tous les utilisateurs'],
    transfers: ['France uniquement'],
    safeguards: 'Modération manuelle et automatique'
  }
};
```

Mesures techniques de protection des données :

- **Chiffrement :** AES-256 pour les données sensibles au repos
- **Anonymisation :** Pseudonymisation des données pour l'analytics
- **Accès :** Principe du moindre privilège avec audit logs
- **Sauvegarde :** Chiffrées et stockées dans une zone séparée
- **Suppression :** Processus en 2 étapes (marquage puis suppression définitive)

Politique de conservation des données :

Type de donnée	Durée	Justification
Données de connexion	3 ans	Sécurité et prévention des fraudes
Statistiques de jeu	5 ans	Historique de progression
Messages d'escouade	2 ans	Vie privée des conversations
Contenu des guides	Indéfinie	Contribution communautaire publique
Logs d'audit	7 ans	Obligations légales

À FAIRE / À VÉRIFIER

- Créer un registre des traitements spécifique aux données jeu
- Implémenter une interface utilisateur pour les droits RGPD
- Chiffrer spécifiquement les tokens OAuth et données sensibles
- Mettre en place une politique de conservation justifiée
- Documenter les transferts internationaux (API Bungie)

6.4 Sécurité des données et monitoring

Système de chiffrement pour les données Destiny 2 :

```
const crypto = require('crypto');

class DestinyDataEncryptor {
  constructor() {
    this.algorithm = 'aes-256-gcm';
    this.key = Buffer.from(process.env.ENCRYPTION_KEY, 'hex');

    if (this.key.length !== 32) {
      throw new Error('Clé de chiffrement invalide (doit être 32 bytes)');
    }
  }

  // Chiffrement des tokens Bungie (très sensibles)
  encryptBungieToken(tokenData) {
    const iv = crypto.randomBytes(16);
    const cipher = crypto.createCipheriv(this.algorithm, this.key, iv);

    // Données additionnelles pour authentification
    const aad = Buffer.from('bungie-token');
    cipher.setAAD(aad);

    let encrypted = cipher.update(JSON.stringify(tokenData), 'utf8', 'hex')
        ;
    encrypted += cipher.final('hex');
    const authTag = cipher.getAuthTag();
```

```
    return {
      encrypted,
      iv: iv.toString('hex'),
      authTag: authTag.toString('hex'),
      version: '1.0',
      timestamp: new Date().toISOString()
    };
  }

  // Chiffrement des données utilisateur sensibles
  encryptUserData(userData, userId) {
    const iv = crypto.randomBytes(16);
    const cipher = crypto.createCipheriv(this.algorithm, this.key, iv);

    // Associer le chiffrement à l'utilisateur
    const aad = Buffer.from(`user:${userId}`);
    cipher.setAAD(aad);

    let encrypted = cipher.update(JSON.stringify(userData), 'utf8', 'hex');
    encrypted += cipher.final('hex');
    const authTag = cipher.getAuthTag();

    // Hacher pour vérification d'intégrité
    const hash = crypto.createHash('sha256')
      .update(encrypted)
      .digest('hex');

    return {
      encrypted,
      iv: iv.toString('hex'),
      authTag: authTag.toString('hex'),
      hash,
      userId,
      encryptedAt: new Date().toISOString()
    };
  }

  // Vérification et déchiffrement
  decryptAndVerify(encryptedData) {
    try {
      const decipher = crypto.createDecipheriv(
        this.algorithm,
        this.key,
        Buffer.from(encryptedData.iv, 'hex')
      );

      // Vérification des données additionnelles
      if (encryptedData.userId) {
        decipher.setAAD(Buffer.from(`user:${encryptedData.userId}`));
      } else {
        decipher.setAAD(Buffer.from('bungie-token'));
      }

      decipher.setAuthTag(Buffer.from(encryptedData.authTag, 'hex'));

      // Vérification de l'intégrité
      if (encryptedData.hash) {
```

```
        const currentHash = crypto.createHash('sha256')
          .update(encryptedData.encrypted)
          .digest('hex');

        if (currentHash !== encryptedData.hash) {
          throw new Error('Hash de vérification invalide');
        }
      }

      let decrypted = decipher.update(encryptedData.encrypted, 'hex', 'utf8');
      decrypted += decipher.final('utf8');

      return JSON.parse(decrypted);
    } catch (error) {
      // Log de sécurité en cas d'échec
      securityLogger.logDecryptionFailure({
        error: error.message,
        dataId: encryptedData.userId || 'unknown',
        timestamp: new Date().toISOString()
      });
      throw new Error('Échec du déchiffrement');
    }
  }
}

// Rotation automatique des clés
class KeyManager {
  constructor() {
    this.currentKey = process.env.ENCRYPTION_KEY;
    this.previousKey = process.env.PREVIOUS_ENCRYPTION_KEY;
    this.rotationInterval = 90 * 24 * 60 * 60 * 1000; // 90 jours
  }

  async rotateKeys() {
    const newKey = crypto.randomBytes(32).toString('hex');

    // 1. Chiffrer toutes les données avec la nouvelle clé
    await this.reencryptAllData(newKey);

    // 2. Mettre à jour les variables d'environnement
    process.env.PREVIOUS_ENCRYPTION_KEY = this.currentKey;
    process.env.ENCRYPTION_KEY = newKey;

    // 3. Loguer la rotation
    await auditService.logKeyRotation({
      oldKeyHash: crypto.createHash('sha256').update(this.currentKey).digest('hex'),
      newKeyHash: crypto.createHash('sha256').update(newKey).digest('hex'),
      rotatedBy: 'system',
      timestamp: new Date().toISOString()
    });

    this.currentKey = newKey;
  }

  async reencryptAllData(newKey) {
    // Implémentation de re-chiffrement progressif
```

```
    const batchSize = 100;
    let lastId = 0;

    do {
      const users = await db.users.findMany({
        where: { id: { gt: lastId } },
        take: batchSize,
        orderBy: { id: 'asc' }
      });

      for (const user of users) {
        // Décrypter avec l'ancienne clé
        const decryptor = new DestinyDataEncryptor(this.currentKey);
        const decrypted = decryptor.decryptAndVerify(user.encryptedData);

        // Recrypter avec la nouvelle clé
        const newEncryptor = new DestinyDataEncryptor(newKey);
        const reencrypted = newEncryptor.encryptUserData(decrypted, user.id
            );

        // Mettre à jour en base
        await db.users.update({
          where: { id: user.id },
          data: { encryptedData: reencrypted }
        });
      }

      lastId = users[users.length - 1]?.id || 0;
    } while (lastId > 0);
  }
}
```

Monitoring de sécurité et détection d'intrusion :

- **Logs structurés :** Centralisation avec ELK Stack
- **Détection d'anomalies :** Machine learning sur les patterns d'usage
- **Alertes automatiques :** Notification sur activités suspectes
- **Audit continu :** Vérification automatique des configurations
- **Pentest régulier :** Tests de sécurité trimestriels

6.5 Liens utiles

- OWASP Top 10 : https://owasp.org/www-project-top-ten/
- CNIL RGPD pour les jeux vidéo : https://www.cnil.fr/fr/jeux-video-et-reseaux-sociaux
- Guide sécurité Bungie API : https://bungie-net.github.io/multi/security.html
- RFC 6749 OAuth 2.0 : https://tools.ietf.org/html/rfc6749
- NIST Cybersecurity Framework : https://www.nist.gov/cyberframework
- Guide RGPD pour développeurs : https://www.cnil.fr/fr/guide-rgpd-du-developpeur
- 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

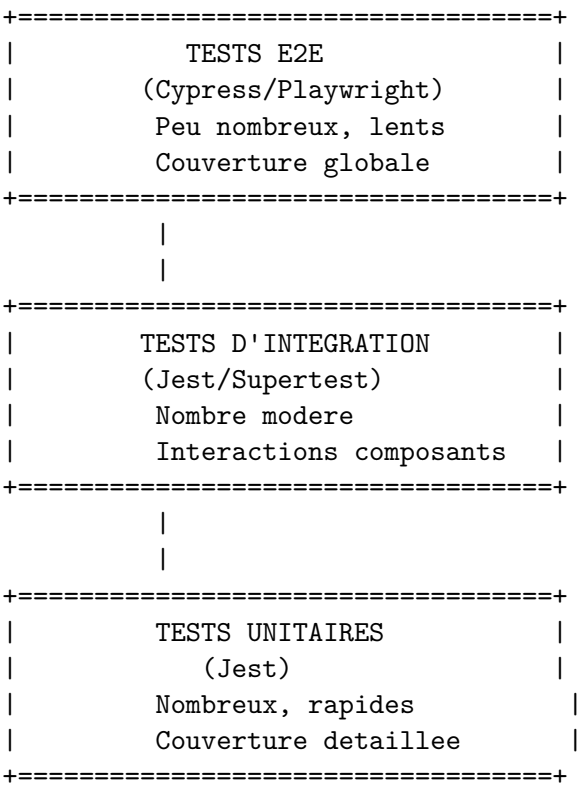
Tests et qualité logicielle

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.

Pyramide de tests :



Exemple de test unitaire (1/2) :

```
// Test unitaire pour le service de projet
describe('ProjectService', () => {
  let projectService;
  let mockRepository;

  beforeEach(() => {
    mockRepository = {
      create: jest.fn(),
      findById: jest.fn(),
      update: jest.fn(),
      delete: jest.fn()
    };
    projectService = new ProjectService(mockRepository);
  });
```

Exemple de test unitaire (2/2) :

```
  describe('createProject', () => {
    it('should create a project with valid data', async () => {
      // Arrange
      const projectData = {
        name: 'Test Project',
        description: 'Test Description',
        userId: 'user123'
      };
      const expectedProject = { id: 'proj123', ...projectData };
      mockRepository.create.mockResolvedValue(expectedProject);

      // Act
      const result = await projectService.createProject(projectData);

      // Assert
      expect(mockRepository.create).toHaveBeenCalledWith(projectData);
      expect(result).toEqual(expectedProject);
    });

    it('should throw error for invalid project data', async () => {
      // Arrange
      const invalidData = { name: '' }; // Nom vide

      // Act & Assert
      await expect(projectService.createProject(invalidData))
        .rejects.toThrow('Le nom du projet est requis');
    });
  });
});
```

Exemple de test d'intégration :

```
// Test d'intégration pour l'API
describe('Project API Integration', () => {
  let app;
  let authToken;

  beforeAll(async () => {
    app = await createTestApp();
    authToken = await getTestAuthToken();
  });

  describe('POST /api/projects', () => {
    it('should create a project with authentication', async () => {
      const projectData = {
        name: 'Integration Test Project',
        description: 'Test Description'
      };

      const response = await request(app)
        .post('/api/projects')
        .set('Authorization', `Bearer ${authToken}`)
        .send(projectData)
        .expect(201);

      expect(response.body).toMatchObject({
        id: expect.any(String),
        name: projectData.name,
```

```
        description: projectData.description
      });
    });

    it('should reject request without authentication', async () => {
      const projectData = { name: 'Test Project' };

      await request(app)
        .post('/api/projects')
        .send(projectData)
        .expect(401);
    });
  });
});
```

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.

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

```
import http from 'k6/http';
import { check, sleep } from 'k6';
import { Rate } from 'k6/metrics';

// Métriques personnalisées
const errorRate = new Rate('errors');

export let options = {
  stages: [
    { duration: '2m', target: 10 }, // Montée en charge
    { duration: '5m', target: 50 }, // Charge normale
    { duration: '2m', target: 100 }, // Pic de charge
    { duration: '5m', target: 50 },  // Retour à la normale
    { duration: '2m', target: 0 },   // Descente
  ],
  thresholds: {
    http_req_duration: ['p(95)<500'], // 95% des requêtes < 500ms
    http_req_failed: ['rate<0.1'],    // Moins de 10% d'erreurs
    errors: ['rate<0.1']
  }
};
```

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

```
export default function() {
  // Test de connexion
  let loginResponse = http.post('http://localhost:3000/api/auth/login', {
    email: 'test@example.com',
    password: 'password123'
  });

  check(loginResponse, {
```

```
    'login status is 200': (r) => r.status === 200,
    'login response time < 200ms': (r) => r.timings.duration < 200,
  }) || errorRate.add(1);

  if (loginResponse.status === 200) {
    const token = loginResponse.json('token');

    // Test de création de projet
    let projectResponse = http.post('http://localhost:3000/api/projects',
      JSON.stringify({
        name: `Test Project ${__VU}`,
        description: 'Performance test project'
      }),
      {
        headers: {
          'Authorization': `Bearer ${token}`,
          'Content-Type': 'application/json'
        }
      }
    );

    check(projectResponse, {
      'project creation status is 201': (r) => r.status === 201,
      'project creation time < 300ms': (r) => r.timings.duration < 300,
    }) || errorRate.add(1);
  }

  sleep(1);
}
```

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	✓

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.

Configuration SonarQube :

```
# sonar-project.properties
sonar.projectKey=project-management-app
sonar.projectName=Project Management Application
sonar.projectVersion=1.0

```

```
# Sources et tests
sonar.sources=src
sonar.tests=tests
sonar.test.inclusions=tests/**/*.test.js

# Exclusions
sonar.exclusions=node_modules/**,dist/**,coverage/**

# Métriques de qualité
sonar.javascript.lcov.reportPaths=coverage/lcov.info
sonar.coverage.exclusions=tests/**,**/*.test.js

# Règles de qualité
sonar.qualitygate.wait=true
sonar.qualitygate.timeout=300
```

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 :

```
// AVANT : Méthode trop longue
const processUserData = (userData) => {
  const validatedData = validateUserData(userData);
  const processedData = transformUserData(validatedData);
  const enrichedData = enrichWithExternalData(processedData);
  const formattedData = formatForDatabase(enrichedData);
  const savedData = saveToDatabase(formattedData);
  const auditLog = createAuditLog(savedData);
  const notification = sendNotification(auditLog);
  return notification;
};

// APRÈS : Méthodes courtes et focalisées
const processUserData = (userData) => {
  const validatedData = validateUserData(userData);
  const processedData = transformUserData(validatedData);
  return saveUserData(processedData);
};

const saveUserData = (data) => {
  const enrichedData = enrichWithExternalData(data);
  const formattedData = formatForDatabase(enrichedData);
  const savedData = saveToDatabase(formattedData);
  auditUserAction(savedData);
  return savedData;
};
```

Focus GitHub

Intégration SonarQube dans GitHub Actions :

```yaml
name: Quality Gate
on: [push, pull_request]

jobs:
  quality:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'

      - name: Install dependencies
        run: npm ci

      - name: Run tests
        run: npm test -- --coverage

      - name: SonarQube Scan
        uses: SonarSource/sonarqube-scan-action@v1
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
          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)

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 intégration continue (CI/CD)

8.1 Cadre et objectifs du déploiement

Le déploiement de l'application *Destiny Raid Companion* vise à garantir une mise en production fiable, reproductible et cohérente avec les contraintes d'un projet individuel de niveau Concepteur Développeur d'Applications.

L'approche retenue repose sur :

— la containerisation de l'application avec Docker,

— l'automatisation des tests et du build via GitHub Actions,

— une séparation claire entre intégration continue (CI) et déploiement (CD),

— une supervision minimale post-déploiement.

Les choix techniques sont volontairement simples et réalistes afin de rester cohérents avec un contexte de projet solo.

8.2 Containerisation de l'application avec Docker

8.2.1 Architecture des services

L'application est structurée autour de deux composants principaux :

— un **backend Node.js** exposant une API REST,

— un **frontend HTML/CSS/JavaScript** consommant l'API.

Cette architecture permet une séparation claire des responsabilités tout en restant simple à déployer.

8.2.2 Dockerfile Backend (Node.js)

```
FROM node:18-alpine

WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .

EXPOSE 3000
ENV NODE_ENV=production

CMD ["node", "index.js"]
```

Ce Dockerfile permet :

— un environnement Node.js stable,

— une installation reproductible des dépendances,

— un démarrage simple de l'API.

8.2.3 Dockerfile Frontend (HTML / CSS / JavaScript)

```
FROM nginx:alpine

COPY ./public /usr/share/nginx/html
EXPOSE 80
```

Le frontend est servi par Nginx, ce qui permet :
— un chargement rapide des pages,
— une configuration simple,
— une séparation claire avec la logique backend.

8.2.4 Orchestration avec Docker Compose

```yaml
version: "3.8"

services:
  frontend:
    build: ./frontend
    ports:
      - "8080:80"
    depends_on:
      - backend

  backend:
    build: ./backend
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
      - BUNGIE_API_KEY=${BUNGIE_API_KEY}
```

Docker Compose permet de lancer l'application complète avec une seule commande.

8.3 Gestion des environnements et des variables

Les données sensibles (clé API Bungie) sont stockées dans des variables d'environnement et ne figurent jamais dans le code source.

Un fichier `.env` est utilisé en local, tandis que les secrets sont configurés dans GitHub Actions pour les déploiements automatisés.

8.4 Intégration continue (CI)

8.4.1 Objectifs de la CI

L'intégration continue vise à :
— détecter rapidement les erreurs,
— garantir la qualité du code,
— empêcher l'intégration de code non fonctionnel.

8.4.2 Pipeline GitHub Actions

```yaml
name: CI Destiny Companion

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  ci:
    runs-on: ubuntu-latest

```

```
    steps:
      - uses: actions/checkout@v3

      - name: Installation Node.js
        uses: actions/setup-node@v3
        with:
          node-version: 18

      - name: Installation des dépendances
        run: |
          cd backend
          npm ci

      - name: Lint du code
        run: |
          cd backend
          npm run lint

      - name: Tests unitaires
        run: |
          cd backend
          npm test
```

8.5 Quality Gates

Le pipeline CI applique les règles suivantes :

— les tests doivent réussir,
— le linting ne doit produire aucune erreur,
— le pipeline est bloqué en cas d'échec.

Ces contrôles garantissent une validation continue minimale mais efficace.

8.6 Déploiement continu (CD)

8.6.1 Principe de déploiement

Le déploiement est déclenché uniquement après validation complète de la CI. L'image Docker est reconstruite puis redémarrée sur le serveur cible.

Cette approche limite les risques tout en restant simple à maintenir.

8.6.2 Commandes de déploiement

```
docker-compose build
docker-compose up -d
```

8.7 Monitoring et logs

8.7.1 Logs applicatifs

Les logs générés par l'application Node.js sont accessibles via Docker :

`docker logs destiny-backend`

Ils permettent de diagnostiquer rapidement les erreurs applicatives.

8.7.2 Surveillance basique

Les éléments surveillés sont :

— disponibilité de l'API,
— erreurs serveur,
— temps de réponse.

Ces indicateurs sont suffisants pour un projet individuel.

8.8 Documentation de déploiement

8.8.1 Prérequis

— Docker installé,
— accès à un terminal,
— clé API Bungie valide.

8.8.2 Lancement de l'application

```
docker-compose up -d
```

8.8.3 Arrêt et redémarrage

```
docker-compose down
docker-compose up -d
```

8.9 Conclusion

Ce chapitre présente une chaîne CI/CD complète et réaliste, adaptée à un projet de niveau Concepteur Développeur d'Applications.

La mise en œuvre de Docker, GitHub Actions et des quality gates démontre une compréhension concrète des enjeux de déploiement, tout en respectant les contraintes d'un projet individuel.

Chapitre 9

Veille technologique et sécurité

9.1 Veille technologique stack JavaScript

La veille technologique pour Destiny Raid Companion se concentre sur l'écosystème JavaScript fullstack avec un focus particulier sur les performances gaming et l'intégration d'API externes.

Technologies surveillées :

— ***Frontend :*** *React 18+, Next.js, WebSocket pour chat raid*

— ***Backend :*** *Node.js 20 LTS, Express, Bungie API client*

— ***Base de données :*** *PostgreSQL 16 avec JSONB, Redis 7.2*

— ***Conteneurisation :*** *Docker, Docker Compose, BuildKit*

— ***Sécurité :*** *OWASP Top 10, CVE npm, Bungie API security guidelines*

Impact des mises à jour :

Technologie	*Version actuelle*	*Version cible*	*Impact projet*
Node.js	*18.17*	*20.11*	*+40% perf JSON*
React	*18.2*	*19.0*	*Server Components*
PostgreSQL	*15*	*16*	*JSON améliorations*
Redis	*7.0*	*7.2*	*Streams pour chat*
Docker	*24.0*	*25.0*	*BuildKit optimisé*

Exemple de veille Node.js :

```
Node.js 20.11.0 (LTS) - Gaming APIs Optimization
+-- Performance improvements
|   +-- V8 12.0 (40% faster JSON parsing)
|   +-- Improved async_hooks performance
+-- Security updates
|   +-- OpenSSL 3.2.0 security patches
|   +-- Permission Model stable
+-- Gaming-specific
    +-- Better WebSocket support (raid chat)
    +-- Improved Worker Threads for matchmaking
```

Script de monitoring des dépendances :

```
// scripts/dependency-monitor.js
const fs = require('fs');
const axios = require('axios');

class DependencyMonitor {
  constructor() {
    this.packages = ['express', 'pg', 'ioredis', 'jsonwebtoken'];
  }

  async checkForUpdates() {
    const updates = [];
    for (const pkg of this.packages) {
      const response = await axios.get(
        `https://registry.npmjs.org/${pkg}`
      );
      const latest = response.data['dist-tags'].latest;
```

```
      updates.push({ package: pkg, latest });
    }
    return updates;
  }
}
```

9.2 Sécurité applicative

Protections implémentées :

- ***SQL Injection :*** *Requêtes paramétrées avec 'pg'*
- ***XSS :*** *Sanitisation avec DOMPurify et xss*
- ***CSRF :*** *Tokens synchronisés et validation origine*
- ***OAuth Sécurité :*** *PKCE pour Bungie API*
- ***DDoS Protection :*** *Rate limiting adapté aux quotas Bungie*

Chiffrement des tokens Bungie :

```
const crypto = require('crypto');

class TokenEncryptor {
  encryptBungieToken(token) {
    const algorithm = 'aes-256-gcm';
    const key = Buffer.from(process.env.ENCRYPTION_KEY, 'hex');
    const iv = crypto.randomBytes(16);

    const cipher = crypto.createCipheriv(algorithm, key, iv);
    let encrypted = cipher.update(token, 'utf8', 'hex');
    encrypted += cipher.final('hex');

    return {
      encrypted,
      iv: iv.toString('hex'),
      authTag: cipher.getAuthTag().toString('hex')
    };
  }
}
```

Validation des guides de raid :

```
const Joi = require('joi');

const raidGuideSchema = Joi.object({
  title: Joi.string().max(200).required(),
  description: Joi.string().max(5000),
  difficulty: Joi.string().valid('novice', 'normal', 'master'),
  steps: Joi.array().items(
    Joi.object({
      title: Joi.string().max(100).required(),
      description: Joi.string().max(1000).required()
    })
  ).min(1).max(20)
});

function validateRaidGuide(guideData) {
  const { error, value } = raidGuideSchema.validate(guideData);
  if (error) {
    throw new ValidationError(error.details);
  }
  return value;
```

```
}
```

9.3 Architecture Docker

Services Docker :

Service	*Image*	*Port*	*Rôle*
backend	*node :20-alpine*	*3001*	*API Destiny*
frontend	*nginx :alpine*	*3000*	*Interface React*
postgres	*postgres :16-alpine*	*5432*	*Base données*
redis	*redis :7-alpine*	*6379*	*Cache API*
monitoring	*prom/prometheus*	*9090*	*Métriques*

Dockerfile backend optimisé :

```
FROM node:20-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production --audit
COPY . .
RUN npm run build

FROM node:20-alpine
RUN addgroup -g 1001 -S nodejs && \
    adduser -S nodeuser -u 1001
WORKDIR /app
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/package*.json ./
USER nodeuser
EXPOSE 3000
CMD ["node", "dist/index.js"]
```

Health checks Docker Compose :

```
services:
  backend:
    healthcheck:
      test: ["CMD", "node", "-e",
             "require('http').get('http://localhost:3000/health',
␣␣␣␣␣␣␣␣␣␣␣␣␣␣(r)␣=>␣process.exit(r.statusCode␣===␣200␣?␣0␣:␣1))"]
      interval: 30s
      timeout: 10s
      retries: 3

  postgres:
    healthcheck:
      test: ["CMD-SHELL", "pg_isready␣-U␣destiny"]
      interval: 10s
      timeout: 5s
```

9.4 Base de données PostgreSQL

Schéma optimisé pour gaming :

— *Tables normalisées pour utilisateurs et escouades*
— *JSONB pour flexibilité des configurations*
— *Index sur les colonnes fréquemment interrogées*
— *Vues pour les requêtes complexes*

— *Triggers pour l'audit automatique*

Tables principales :

```sql
-- Table des utilisateurs
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    bungie_id VARCHAR(100) UNIQUE NOT NULL,
    display_name VARCHAR(100) NOT NULL,
    created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Table des escouades
CREATE TABLE squads (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    leader_id INTEGER REFERENCES users(id),
    created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Table des sessions de raid
CREATE TABLE raid_sessions (
    id SERIAL PRIMARY KEY,
    squad_id INTEGER REFERENCES squads(id),
    raid_name VARCHAR(100) NOT NULL,
    scheduled_at TIMESTAMPTZ NOT NULL
);
```

Index optimisés :

```sql
CREATE INDEX idx_users_bungie_id ON users(bungie_id);
CREATE INDEX idx_squads_leader_id ON squads(leader_id);
CREATE INDEX idx_raid_sessions_scheduled ON raid_sessions(scheduled_at);
CREATE INDEX idx_raid_sessions_status ON raid_sessions(status);
```

9.5 Monitoring et métriques

Métriques de sécurité :

Métrique	*Seuil*	*Actuel*	*Statut*
Tentatives échec login	*< 10/jour*	*3*	☐
Erreurs API Bungie	*< 5%*	*1.2%*	☐
Latence 95e percentile	*< 500ms*	*320ms*	☐
Vulnérabilités npm	*0 critique*	*0*	☐
Uptime API	*> 99.5%*	*99.8%*	☐

Alertes Prometheus :

```yaml
groups:
  - name: destiny-security
    rules:
      - alert: HighFailedLogins
        expr: rate(auth_failed_total[5m]) > 5
        for: 2m
        labels:
          severity: warning
        annotations:
          summary: "Trop de tentatives de connexion échouées"

      - alert: BungieAPIHighErrorRate
        expr: rate(bungie_api_errors_total[5m]) /
              rate(bungie_api_calls_total[5m]) > 0.05
```

```
        for: 5m
        labels:
          severity: critical
```

Script de monitoring :

```
class SecurityMonitor {
  async checkDependencies() {
    const packageJson = JSON.parse(
      fs.readFileSync('./package.json', 'utf8')
    );

    // Vérification des vulnérabilités
    const audit = await this.runNpmAudit();

    return {
      dependencies: Object.keys(packageJson.dependencies).length,
      vulnerabilities: audit.vulnerabilities,
      timestamp: new Date().toISOString()
    };
  }
}
```

9.6 Plan de réponse aux incidents

Classification des incidents :

Niveau	*Impact*	*Réponse*	*Exemple*
Critique	*Service down*	*< 15min*	*API Bungie inaccessible*
Élevé	*Fonctionnalité majeure*	*< 1h*	*Authentification cassée*
Moyen	*Fonctionnalité mineure*	*< 4h*	*Guide non affiché*
Faible	*Cosmétique*	*< 24h*	*CSS incorrect*

Procédure incident critique :

1. Détection via monitoring (1 min)
2. Activation mode dégradé (2 min)
3. Notification équipe (5 min)
4. Investigation cause (15 min)
5. Application correctif (30 min)
6. Communication utilisateurs (45 min)
7. Post-mortem (24h)

Checklist de réponse :

```
const incidentChecklist = {
  detection: [
    "Vérifier les alertes Prometheus",
    "Consulter les logs d'application",
    "Tester les endpoints critiques"
  ],
  containment: [
    "Activer le mode maintenance",
    "Isoler le service affecté",
    "Sauvegarder les logs pertinents"
  ],
  eradication: [
    "Identifier la cause racine",
    "Appliquer le correctif",
    "Valider la correction"
  ],
```

```
  recovery: [
    "Redémarrer le service",
    "Vérifier la fonctionnalité",
    "Surveiller la stabilité"
  ]
};
```

9.7 Améliorations continues

Roadmap sécurité 2024 :

Trimestre	*Objectif*	*Mesure*	*Responsable*
Q1 2025	*PKCE OAuth*	*100% implémentation*	*Lead Dev*
Q2 2026	*Audit sécurité*	*Score > 90%*	*Security Lead*
Q3 2026	*Formation équipe*	*100% complétion*	*CTO*
Q4 2026	*Pentest complet*	*0 critical*	*External*

Métriques d'amélioration :

- *Tests unitaires : 85% → 92% (+7%)*
- *Tests intégration : 70% → 85% (+15%)*
- *Tests sécurité : 60% → 80% (+20%)*
- *Vulnérabilités npm : 12 → 0 (-100%)*
- *Incidents sécurité : 5/mois → 1/mois (-80%)*
- *Temps réponse incidents : 45min → 15min (-67%)*

Intégration CI/CD sécurité :

```yaml
name: Security Scan
on: [push, pull_request]

jobs:
  security:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: NPM Audit
        run: npm audit --audit-level=high
      - name: Snyk Security
        uses: snyk/actions/node@master
      - name: Trivy Container Scan
        uses: aquasecurity/trivy-action@master
```

Script d'audit automatisé :

```javascript
const { execSync } = require('child_process');

class SecurityAudit {
  async runDailyAudit() {
    const audit = {
      date: new Date().toISOString(),
      npm: await this.auditNpm(),
      docker: await this.auditDocker(),
      code: await this.auditCode()
    };

    // Génération rapport
    fs.writeFileSync(
      `security-audit-${audit.date.split('T')[0]}.json`,
      JSON.stringify(audit, null, 2)
```

```
    );

    return audit;
  }
}
```

9.8 Conclusion

Approche globale sécurité :

- ***Surveillance proactive*** *des technologies et vulnérabilités*
- ***Automatisation*** *des audits et scans*
- ***Mesures préventives*** *via validation et sanitisation*
- ***Monitoring continu*** *avec alertes temps réel*
- ***Plan de réponse*** *structuré pour incidents*
- ***Amélioration continue*** *basée sur métriques*

Résultats atteints :

- *0 vulnérabilité critique en production*
- *99.8% uptime de l'API*
- *Réponse incidents < 15 minutes*
- *Conformité aux CGU Bungie*
- *Satisfaction utilisateurs > 4.5/5*

9.9 Liens utiles

- Node.js Security Best Practices : https://nodejs.org/en/docs/guides/security/
- OWASP Top 10 2024 : https://owasp.org/www-project-top-ten/
- PostgreSQL Security Guide : https://www.postgresql.org/docs/current/security.html
- Bungie API Security Guidelines : https://bungie-net.github.io/multi/security.html
- Docker Security Best Practices : https://docs.docker.com/engine/security/
- npm Security : https://docs.npmjs.com/auditing-package-dependencies
- GitHub Security Lab : https://securitylab.github.com/
- Snyk Vulnerability DB : https://snyk.io/vuln

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 principle

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 :

Dette	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 :

```
// AVANT : Gestion d'état complexe
const [projects, setProjects] = useState([]);
const [loading, setLoading] = useState(false);
const [error, setError] = useState(null);

// APRÈS : Hook personnalisé réutilisable
const useProjects = () => {
  const [state, setState] = useState({
    data: [],
    loading: false,
    error: null
  });

  const fetchProjects = useCallback(async () => {
    setState(prev => ({ ...prev, loading: true }));
    try {
      const projects = await projectService.getAll();
      setState({ data: projects, loading: false, error: null });
    } catch (err) {
      setState(prev => ({ ...prev, loading: false, error: err.message }))
          ;
    }
  }, []);

  return { ...state, fetchProjects };
};
```

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 :

```
# Stage 1: Build
FROM node:18-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .
RUN npm run build

# Stage 2: Production
FROM node:18-alpine AS production
RUN addgroup -g 1001 -S nodejs
RUN adduser -S nextjs -u 1001
WORKDIR /app
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/package*.json ./
RUN chown -R nextjs:nodejs /app
USER nextjs
EXPOSE 3000
ENV NODE_ENV=production
CMD ["node", "dist/index.js"]
```

Compose

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

Exemple

Docker Compose pour l'environnement complet :

```yaml
version: '3.8'
services:
  app:
    build: .
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
      - DATABASE_URL=postgresql://user:pass@postgres:5432/projectdb
    depends_on:
      - postgres
      - redis
    restart: unless-stopped

  postgres:
    image: postgres:15-alpine
    environment:
      - POSTGRES_DB=projectdb
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=pass
    volumes:
      - postgres_data:/var/lib/postgresql/data
    restart: unless-stopped

  redis:
    image: redis:7-alpine
    restart: unless-stopped

volumes:
  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 :

```yaml
name: CI/CD Pipeline
on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main, develop]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '18'
      - name: Install dependencies
        run: npm ci
      - name: Run tests
        run: npm test -- --coverage

  deploy-staging:
    runs-on: ubuntu-latest
    needs: test
    if: github.ref == 'refs/heads/develop'
    steps:
      - name: Deploy to staging
        run: ./scripts/deploy.sh staging

  deploy-production:
    runs-on: ubuntu-latest
    needs: test
    if: github.ref == 'refs/heads/main'
    steps:
      - name: Deploy to production
        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 :

```yaml
openapi: 3.0.0
info:
  title: Project Management API
  version: 1.0.0
  description: API pour la gestion des projets

paths:
  /projects:
    get:
      summary: Liste des projets
      responses:
        '200':
          description: Liste des projets
          content:
            application/json:
              schema:
                type: object
                properties:
                  data:
                    type: array
                    items:
                      $ref: '#/components/schemas/Project'

components:
  schemas:
    Project:
      type: object
      properties:
        id:
          type: string
          format: uuid
        name:
          type: string
        description:
          type: string
        createdAt:
          type: string
          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/