

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	13
2.6	Cibles et parties prenantes	14
2.7	Exigences fonctionnelles	15
2.8	Définition du MVP	18
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	23
3.2.2	Conventions de branches	23
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	31
4.3	Conception de l'interface graphique	31
4.3.1	Architecture des composants React	31
4.3.2	Charte graphique détaillée	33
4.3.3	Maquettes et prototypes	34
4.4	Conception de base de données	34
4.4.1	Modèle Conceptuel de Données (MCD)	34

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)	35
4.5.3	Couche Données	37
4.6	Stratégie de tests	37
4.6.1	Couverture de tests	38
4.6.2	Automatisation des tests	39
4.7	Plan de déploiement et infrastructure	39
4.7.1	Architecture de déploiement	39
4.7.2	CI/CD et monitoring	41
4.8	Liens utiles	41
5	Architecture 3 tiers	45
5.1	Architecture 3 tiers	45
5.1.1	Couche Présentation (Frontend)	45
5.1.2	Couche Logique Métier (Backend)	46
5.1.3	Couche Données (Database)	48
5.1.4	Communication entre les tiers	49
5.1.5	Avantages de l'architecture 3 tiers	49
5.2	Développement Frontend	50
5.3	Développement Backend	53
5.4	Gestion des données	57
5.5	Liens utiles	61
6	Sécurité applicative et RGPD	63
6.1	Protection contre les vulnérabilités OWASP	63
6.2	Authentification et autorisation	65
6.3	Conformité RGPD	68
6.4	Sécurité des données et monitoring	71
6.5	Liens utiles	75
7	Tests et qualité logicielle	77
7.1	Stratégie de tests	77
7.2	Tests de performance	79
7.3	Qualité du code avec SonarQube	81
7.4	Liens utiles	83
8	Déploiement et intégration continue (CI/CD)	85
8.1	Cadre et objectifs du déploiement	85
8.2	Containerisation de l'application avec Docker	85
8.2.1	Architecture des services	85
8.2.2	Dockerfile Backend (Node.js)	85
8.2.3	Dockerfile Frontend (HTML / CSS / JavaScript)	85
8.2.4	Orchestration avec Docker Compose	86
8.3	Gestion des environnements et des variables	86
8.4	Intégration continue (CI)	86
8.4.1	Objectifs de la CI	86
8.4.2	Pipeline GitHub Actions	86
8.5	Quality Gates	87
8.6	Déploiement continu (CD)	87

8.6.1	Principe de déploiement	87
8.6.2	Commandes de déploiement	87
8.7	Monitoring et logs	87
8.7.1	Logs applicatifs	87
8.7.2	Surveillance basique	87
8.8	Documentation de déploiement	88
8.8.1	Prérequis	88
8.8.2	Lancement de l'application	88
8.8.3	Arrêt et redémarrage	88
8.9	Conclusion	88
9	Veille technologique et sécurité	89
9.1	Veille technologique stack JavaScript	89
9.2	Sécurité applicative	90
9.3	Architecture Docker	91
9.4	Base de données PostgreSQL	91
9.5	Monitoring et métriques	92
9.6	Plan de réponse aux incidents	93
9.7	Améliorations continues	94
9.8	Conclusion	95
9.9	Liens utiles	95
10	Bilan et retour d'expérience (REX)	97
10.1	Objectifs atteints et non atteints	97
10.2	Difficultés rencontrées et solutions	97
10.3	Dettes techniques et apprentissages	98
10.4	Liens utiles	100
11	Conclusion et remerciements	101
11.1	Synthèse du projet	101
11.2	Perspectives d'évolution	102
11.3	Remerciements	103
11.4	Déploiement et documentation	103
11.4.1	Docker	104
11.4.2	GitHub (code source)	105
11.4.3	CI/CD	106
11.4.4	SonarQube	106
11.4.5	Swagger	107
11.5	Liens utiles	109

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 vétérane 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 personnelle, 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 se fera sur la période octobre 2025 – mai 2026, MVP déployé en mars 2026 en 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 $\geq 4,5/5$ 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 < 500 ms 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>

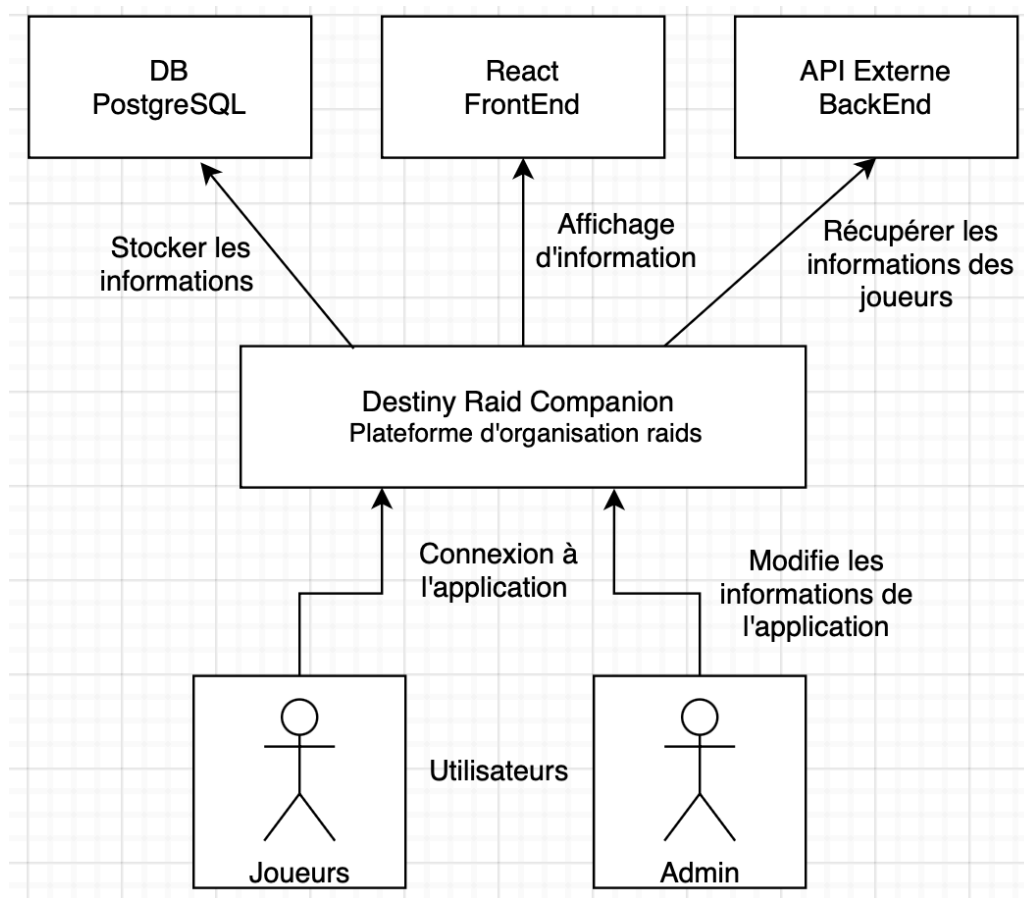


Figure 1.1 – Diagramme de contexte de la plateforme Destiny Raid Companion

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 x 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% up-time	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 $\geq 4.5/5$	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
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 qua non
Must Have	Guides interactifs raids	Cœur valeur ajoutée, différenciation	Résolution problème principal
Must Have	Gestion escouades	Fonctionnalité collaborative essentielle	Rétention utilisateurs
Must Have	Base de données PostgreSQL	Persistance données, performances	Fondation technique
Should Have	Calendrier collaboratif	Réduction temps organisation mesurable	Gain temps 55%
Should Have	Profil joueur + statistiques	Personnalisation expérience utilisateur	Engagement +25%
Could Have	Système de badges	Gamification, motivation	Augmentation rétention
Could Have	Notifications	Rappels sessions, engagement	Réduction absentéisme
Won't Have	App mobile native	Coût développement trop élevé MVP	Report version 2.0
Won't Have	Chat temps réel	Complexité technique, coût	Discord reste solution externe
Won't Have	Streaming intégré	Hors scope, complexité légale	Solutions dédiées existent

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 play-style	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	X	✓	✓	✓	✓
Créer escouade	X	✓	✓	✓	✓
Planifier session raid	X	X	✓	✓	✓
Modifier guides	X	X	X	✓	✓
Admin utilisateurs	X	X	X	X	✓
Accès analytics	X	X	X	✓	✓
Configuration système	X	X	X	X	✓

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 exclus (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 es-couades	Nombre es-couades 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 :

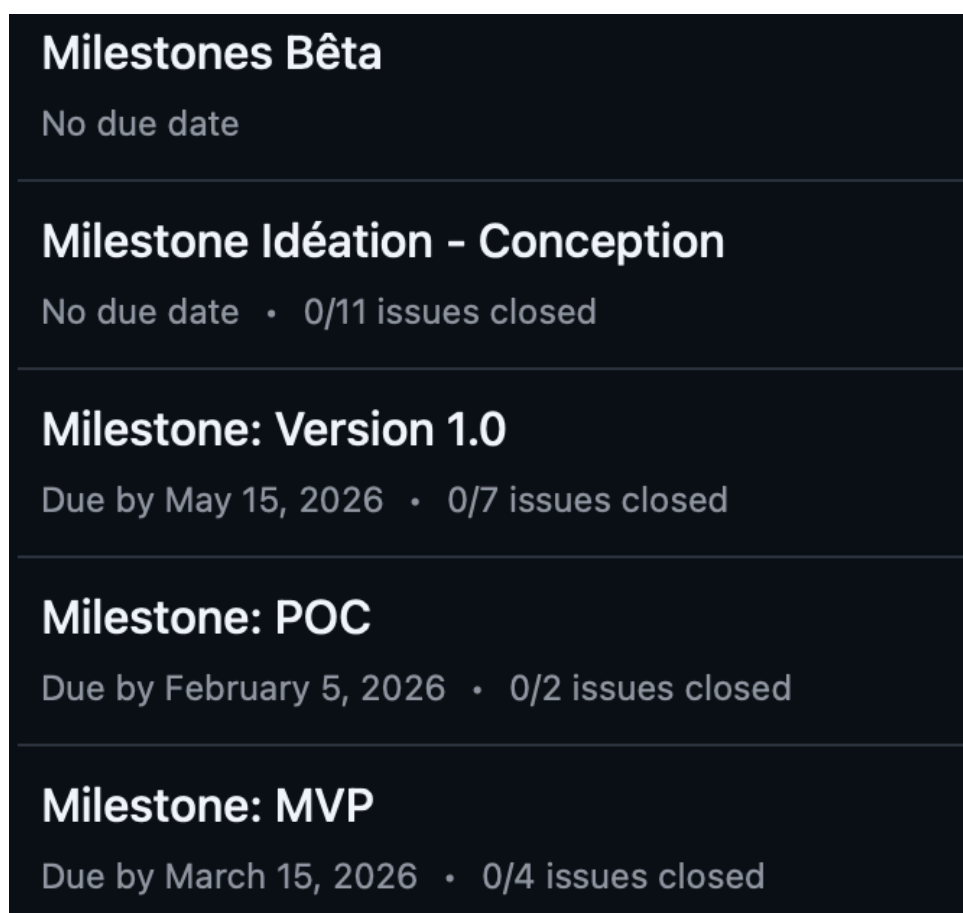


Figure 2.1 – User stories

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.mountangoatsoftware.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).

User stories avec estimations et alignement roadmap :

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 docu- mentation sans risque code

2	⦿ US-POC-1 - Setup environnement technique #6
3	⦿ US-POC-2 - Authentification Bungie basique #7
4	⦿ US-POC-3 - Mockup guides interactifs #8
5	⦿ US-MVP-1 - Authentification complète #9
6	⦿ US-MVP-2 - Guides 3 raids principaux #10
7	⦿ US-MVP-3 - Gestion escouades basique #11
8	⦿ US-MVP-4 - Calendrier collaboratif #12
9	⦿ US-MVP-5 - Interface responsive #13
10	⦿ US-1.0-1 - Profil joueur détaillé #14
11	⦿ US-1.0-2 - Système de badges #15
12	⦿ US-1.0-3 - Recherche avancée joueurs #16
13	⦿ US-1.0-4 - Notifications automatiques #17
14	⦿ US-1.0-5 - Analytics administrateur #18

Figure 3.1 – User stories

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) :

```
1 feat: add OAuth Bungie authentication system
2 fix: resolve login token expiration issue
3 docs: update API integration guide
4 test: add unit tests for user service
5 refactor: improve raid guide component structure
6 style: format code with prettier
7 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)
- **Politiques** : 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

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.mountangoatsoftware.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 : Spécifications des cas d'usage critiques : UC002 - Consul-

tation 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

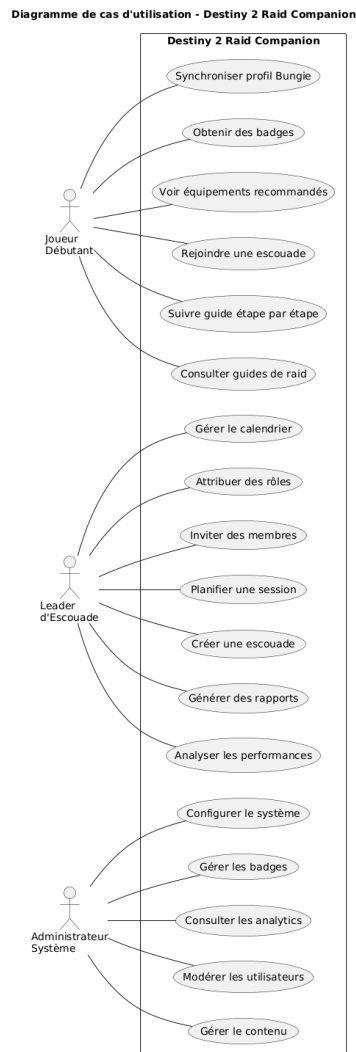


Figure 4.1 – Diagramme Use Case

- **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 :

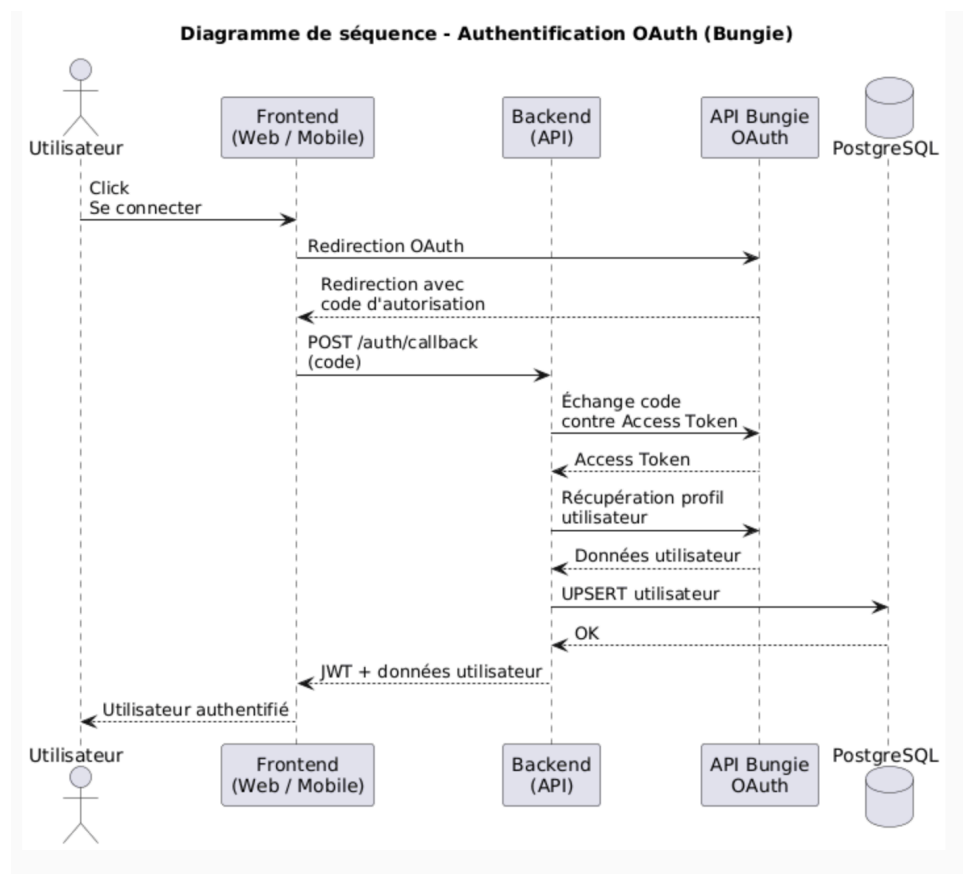


Figure 4.2 – Diagramme de séquence - Authentification OAuth

Diagramme de séquence - Consultation guide avec cache :

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/
  
```

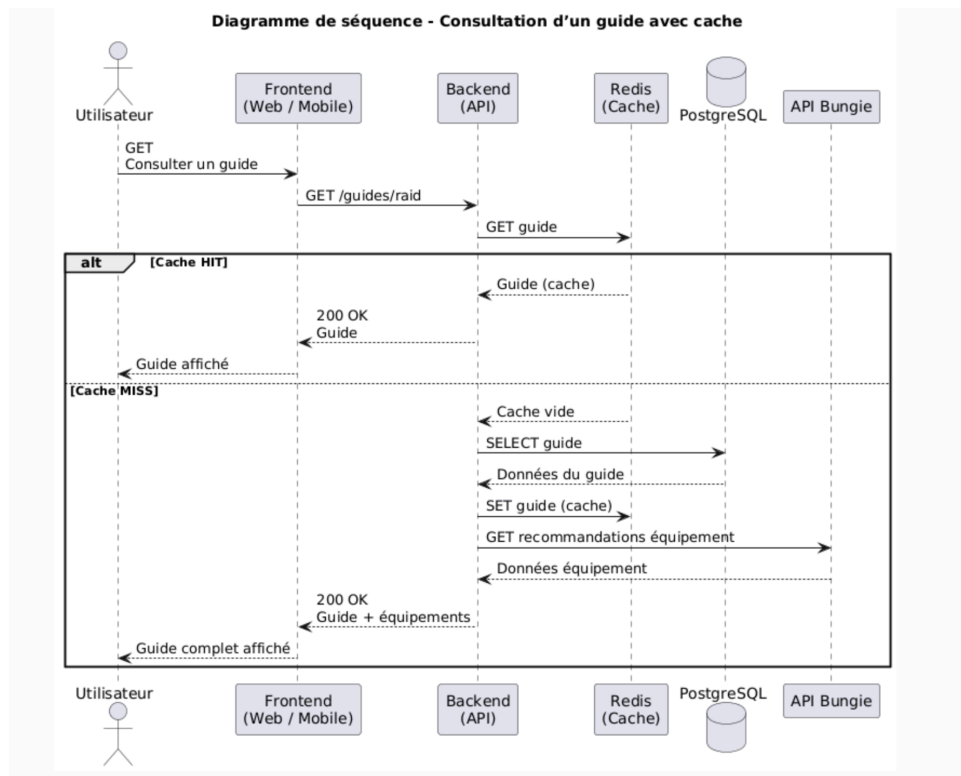


Figure 4.3 – Diagramme de séquence - Consultation guide avec cache

```

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** : guidId, 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 :

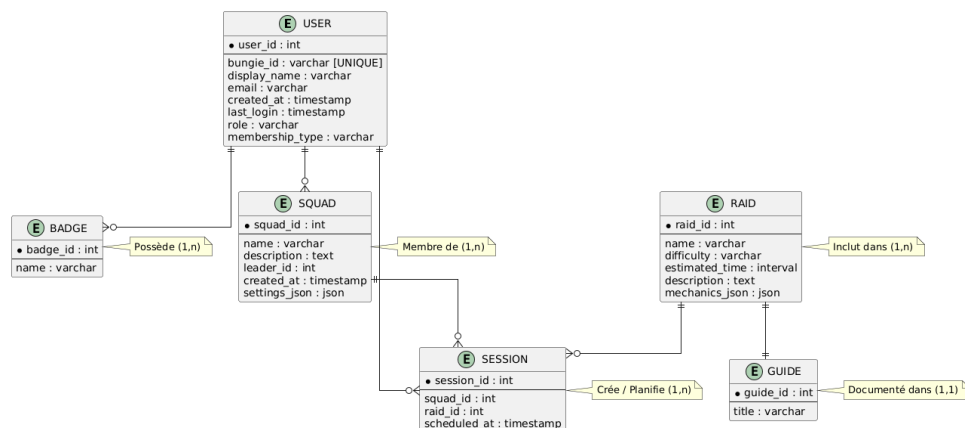


Figure 4.4 – Entités principales et relations

Diagramme MCD complet :

4.4.2 Modèle Logique de Données (MLD)

4.4.3 Modèle Physique de Données (MPD)

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) {
6       throw new ValidationError(validation.error);
7     }
8
9     // Vérification des limites
10    const userSquadCount = await this.getUserSquadCount(leaderId);
11    if (userSquadCount >= MAX_SQUADS_PER_USER) {
12      throw new BusinessError('Limite d\'escouades atteinte');
13    }
14
15    // Création transaction
16    return db.transaction(async (trx) => {
17      const squad = await Squad.create(trx, {
18        ...squadData,
19        leader_id: leaderId
20      });
21
22      // Ajout du leader comme membre
23      await SquadMember.create(trx, {
24        squad_id: squad.id,
25        user_id: leaderId,
26        role: 'leader',
27        joined_at: new Date()
28      });
29
30      // Audit log
31      await AuditLog.create(trx, {
32        action: 'squad_created',
33        user_id: leaderId,
34        squad_id: squad.id,
35        metadata: { squad_name: squad.name }
36      });
37
38      return squad;
39    });
40  }

```

```

41 }

Middleware d'authentification :

1 const authenticateToken = async (req, res, next) => {
2   const authHeader = req.headers['authorization'];
3   const token = authHeader && authHeader.split(' ')[1];
4
5   if (!token) {
6     return res.status(401).json({ error: 'Token_manquant' });
7   }
8
9   try {
10    const decoded = jwt.verify(token, process.env.JWT_SECRET);
11    const user = await User.findById(decoded.userId);
12
13    if (!user) {
14      return res.status(401).json({ error: 'Utilisateur_non_trouvé' });
15    }
16
17    req.user = user;
18    next();
19  } catch (error) {
20    if (error.name === 'TokenExpiredError') {
21      return res.status(401).json({ error: 'Token_expiré' });
22    }
23    return res.status(403).json({ error: 'Token_invalide' });
24  }
25 };

```

4.5.3 Couche Données

Configuration PostgreSQL :

```

1 -- Configuration des performances
2 ALTER SYSTEM SET shared_buffers = '1GB';
3 ALTER SYSTEM SET work_mem = '64MB';
4 ALTER SYSTEM SET maintenance_work_mem = '256MB';
5 ALTER SYSTEM SET effective_cache_size = '3GB';
6
7 -- Configuration de la réplication
8 ALTER SYSTEM SET wal_level = 'replica';
9 ALTER SYSTEM SET max_wal_senders = 10;
10 ALTER SYSTEM SET hot_standby = 'on';
11
12 -- Redémarrage pour appliquer les changements
13 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 :

```

1 describe('SquadService', () => {
2   describe('createSquad', () => {
3     it('should create squad with valid data', async () => {
4       const mockLeader = { id: 1, squad_count: 2 };
5       const squadData = { name: 'TestSquad', description: 'Test' };
6
7       userRepository.getUserSquadCount.mockResolvedValue(2);
8       squadRepository.create.mockResolvedValue({ id: 1, ...squadData });
9
10      const result = await squadService.createSquad(squadData, mockLeader.
11        id);
12
13      expect(result).toHaveProperty('id', 1);
14      expect(result.name).toBe('TestSquad');
15      expect(userRepository.getUserSquadCount).toHaveBeenCalledTimes(1);
16    });
17
18    it('should throw error when user exceeds squad limit', async () => {
19      const mockLeader = { id: 1, squad_count: 5 };
20      userRepository.getUserSquadCount.mockResolvedValue(5);
21
22      await expect(
23        squadService.createSquad({ name: 'Test' }, mockLeader.id)
24      ).rejects.toThrow('Limite d\'escouades atteinte');
25    });
26  });

```

Tests d'intégration API :

```

1 describe('SquadAPI', () => {
2   describe('POST /api/squads', () => {
3     it('should create squad with authentication', async () => {
4       const authToken = await createTestUser();
5       const squadData = { name: 'API Test Squad' };
6
7       const response = await request(app)
8         .post('/api/squads')
9         .set('Authorization', `Bearer ${authToken}`)
10        .send(squadData)
11        .expect(201);
12
13      expect(response.body).toHaveProperty('id');
14      expect(response.body.name).toBe('API Test Squad');
15      expect(response.body.leader_id).toBe(1);
16    });
17
18    it('should reject unauthenticated requests', async () => {
19      await request(app)

```

```

20     .post('/api/squads')
21     .send({ name: 'Test' })
22     .expect(401);
23   });
24 });
25 });

```

4.6.2 Automatisation des tests

Pipeline de tests GitHub Actions :

```

1 name: Test Pipeline
2 on: [push, pull_request]
3 jobs:
4   unit-tests:
5     runs-on: ubuntu-latest
6     steps:
7       - uses: actions/checkout@v3
8       - uses: actions/setup-node@v3
9         with: { node-version: '18' }
10      - run: npm ci
11      - run: npm run test:unit
12      - uses: codecov/codecov-action@v3
13
14   integration-tests:
15     runs-on: ubuntu-latest
16     services:
17       postgres:
18         image: postgres:14
19         env: { POSTGRES_PASSWORD: test }
20         options: >-
21           --health-cmd pg_isready
22           --health-interval 10s
23           --health-timeout 5s
24           --health-retries 5
25     steps:
26       - uses: actions/checkout@v3
27       - run: npm ci
28       - run: npm run test:integration
29
30   e2e-tests:
31     runs-on: ubuntu-latest
32     steps:
33       - uses: actions/checkout@v3
34       - run: npm ci
35       - run: npm run build
36       - 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 :

```

1 FROM node:18-alpine
2 WORKDIR /app
3
4 # Installation des dépendances
5 COPY package*.json ./
6 RUN npm ci --only=production
7
8 # Copie du code
9 COPY . .
10
11 # Sécurité
12 RUN addgroup -g 1001 -S nodejs
13 RUN adduser -S nextjs -u 1001
14 USER nextjs
15
16 # Exposition du port
17 EXPOSE 4000
18
19 # Health check
20 HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
21   CMD curl -f http://localhost:4000/health || exit 1
22
23 CMD ["node", "src/server.js"]

```

docker-compose.yml :

```

1 version: '3.8'
2 services:
3   frontend:
4     build: ./frontend
5     ports: ["3000:3000"]
6     environment:
7       - REACT_APP_API_URL=http://localhost:4000
8     depends_on: [backend]
9
10  backend:
11    build: ./backend
12    ports: ["4000:4000"]
13    environment:
14      - NODE_ENV=development
15      - DATABASE_URL=postgresql://user:pass@db:5432/raidcompanion
16      - REDIS_URL=redis://redis:6379
17      - BUNGIE_API_KEY=${BUNGIE_API_KEY}
18    depends_on:
19      db:
20        condition: service_healthy
21      redis:
22        condition: service_healthy
23
24  db:
25    image: postgres:14
26    environment:
27      - POSTGRES_DB=raidcompanion
28      - POSTGRES_USER=user
29      - POSTGRES_PASSWORD=pass
30    volumes:
31      - postgres_data:/var/lib/postgresql/data

```



```

32     healthcheck:
33       test: ["CMD-SHELL", "pg_isready -U user -d raidcompanion"]
34       interval: 10s
35       timeout: 5s
36       retries: 5
37
38     redis:
39       image: redis:7-alpine
40       healthcheck:
41         test: ["CMD", "redis-cli", "ping"]
42         interval: 10s
43         timeout: 3s
44         retries: 3
45
46 volumes:
47     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/goldbergonyi/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/>

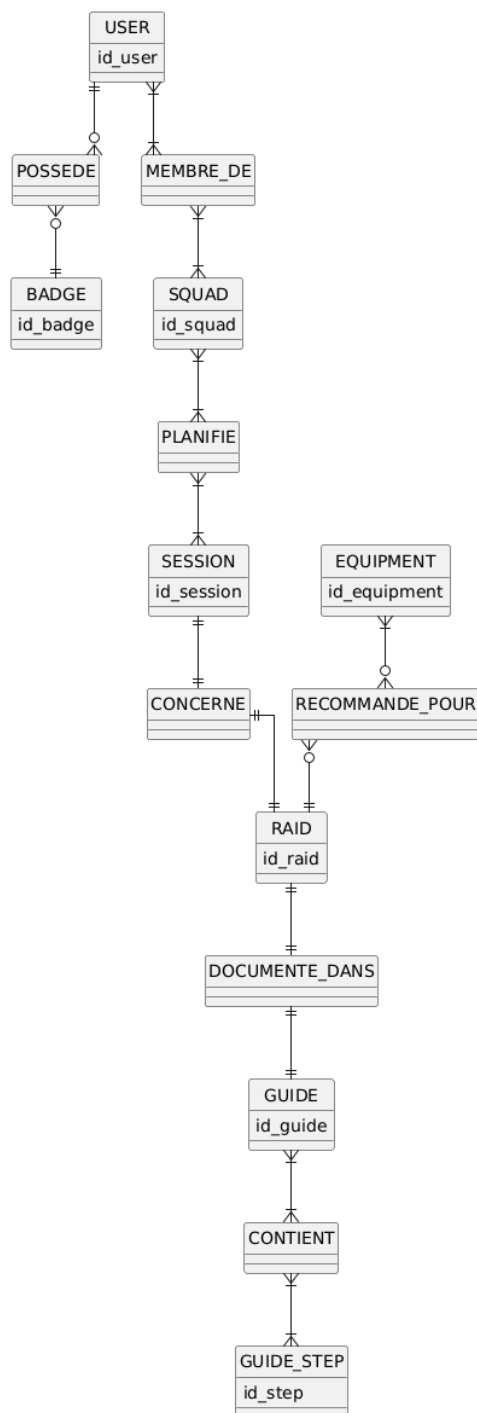


Figure 4.5 – Diagramme MCD complet

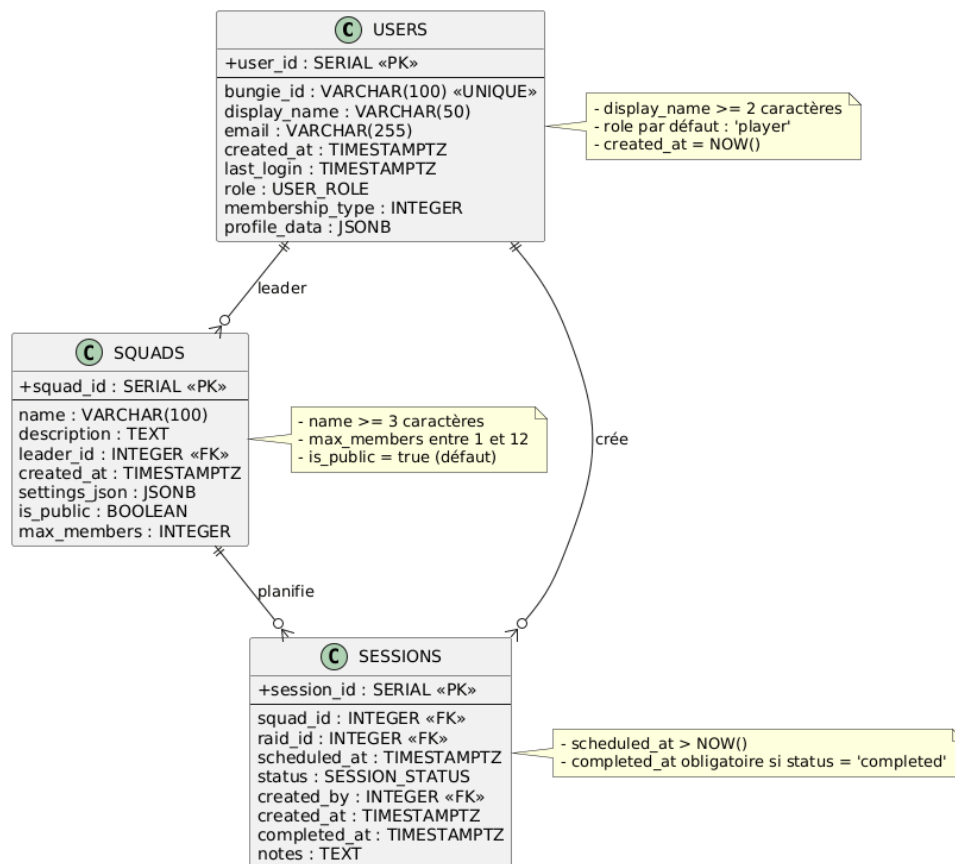


Figure 4.6 – Schéma relationnel normalisé



Figure 4.7 – Modèle Physique de Données

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.

```

1 // squadController.js
2 class SquadController {
3   async createSquad(req, res) {
4     try {
5       // Validation des données d'entrée
6       const validation = squadSchema.safeParse(req.body);
7       if (!validation.success) {
8         return res.status(400).json({
9           error: 'Données invalides',
10          details: validation.error.issues
11        });
12      }
13
14      // Appel du service métier
15      const squad = await this.squadService.createSquad(
16        validation.data,
17        req.user.id
18      );
19
20      // Réponse standardisée
21      res.status(201).json({
22        success: true,
23        data: squad,
24        message: 'Escouade créée avec succès'
25      });
26    } catch (error) {
27      // Gestion centralisée des erreurs
28      if (error instanceof BusinessError) {
29        return res.status(400).json({
30          success: false,
31          error: error.message
32        });
33      }
34      next(error);
35    }
36  }
37 }

```

```

36 }
37 }

```

Service

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

```

1 // squadService.js
2 class SquadService {
3   async createSquad(squadData, leaderId) {
4     // Validation métier
5     if (squadData.maxMembers > 12) {
6       throw new BusinessError('Une escouade ne peut pas dépasser 12 membres');
7     }
8
9     // Vérification des limites utilisateur
10    const userSquadCount = await this.squadRepository.countByUser(leaderId);
11    ;
12    if (userSquadCount >= 5) {
13      throw new BusinessError('Limite de 5 escouades par utilisateur atteinte');
14    }
15
16    // Transaction pour cohérence des données
17    return await this.db.transaction(async (trx) => {
18      // Création de l'escouade
19      const squad = await this.squadRepository.create(trx, {
20        ...squadData,
21        leader_id: leaderId
22      });
23
24      // Ajout du leader comme membre
25      await this.squadMemberRepository.create(trx, {
26        squad_id: squad.id,
27        user_id: leaderId,
28        role: 'leader'
29      });
30
31      // Audit de l'action
32      await this.auditService.log(trx, {
33        action: 'squad_created',
34        user_id: leaderId,
35        metadata: { squad_id: squad.id }
36      });
37
38      return squad;
39    });
40 }

```

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.).

```

1 // squadRepository.js

```

```

2 class SquadRepository {
3   async create(trx, squadData) {
4     return await trx.squad.create({
5       data: {
6         name: squadData.name,
7         description: squadData.description,
8         leader_id: squadData.leader_id,
9         max_members: squadData.maxMembers,
10        is_public: squadData.isPublic ?? true,
11        settings: squadData.settings || {}
12      },
13      include: {
14        leader: {
15          select: { id: true, display_name: true }
16        }
17      }
18    });
19  }
20
21  async countByUser(userId) {
22    return await this.db.squad.count({
23      where: {
24        OR: [
25          { leader_id: userId },
26          { members: { some: { user_id: userId } } }
27        ]
28      }
29    });
30  }
31 }

```

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

```

1 // schema.prisma
2 model User {
3   id          Int          @id @default(autoincrement())
4   bungie_id   String       @unique
5   display_name String
6   email       String?
7   created_at  DateTime @default(now())
8   last_login  DateTime?
9   role        UserRole @default(PLAYER)
10
11   squads_led   Squad[] @relation("SquadLeader")
12   squad_members SquadMember[]
13   sessions_created Session[]
14 }
15
16 model Squad {

```



```

17   id          Int          @id @default(autoincrement())
18   name        String
19   description  String?
20   leader_id   Int
21   created_at  DateTime @default(now())
22   is_public   Boolean @default(true)
23   max_members Int          @default(6)
24
25   leader      User          @relation("SquadLeader", fields: [leader_id],
26     references: [id])
27   members     SquadMember []
28   sessions    Session []
29 }
30
31 model Session {
32   id          Int          @id @default(autoincrement())
33   squad_id    Int
34   raid_id     Int
35   scheduled_at DateTime
36   status      SessionStatus @default(SCHEDULED)
37   created_by  Int
38   created_at  DateTime @default(now())
39
40   squad       Squad         @relation(fields: [squad_id], references: [id])
41   raid        Raid          @relation(fields: [raid_id], references: [id])
42   creator     User          @relation(fields: [created_by], references: [id])
43 }

```

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 :**

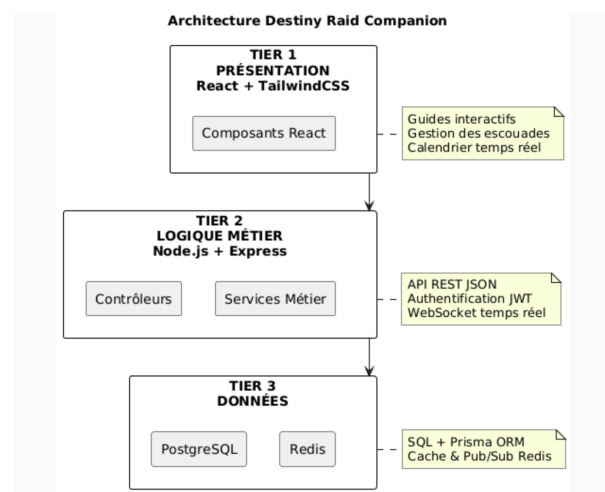


Figure 5.1 – Architecture 3 tiers de Destiny Raid Companion

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 présentionnels 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/présentionnels
|   |   +-- 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

1 const Joi = require('joi');
2
3 // Schéma de validation avec Joi
4 const createSquadSchema = Joi.object({
5   name: Joi.string().min(3).max(100).required(),
6   description: Joi.string().max(500).optional(),

```

```
7   maxMembers: Joi.number().min(1).max(12).default(6),
8   isPublic: Joi.boolean().default(true),
9   settings: Joi.object({
10     language: Joi.string().default('fr'),
11     experienceLevel: Joi.string().valid('beginner', 'intermediate', 'expert')
12   }).optional()
13 });
14
15 // Contrôleur de création d'escouade
16 const createSquad = async (req, res, next) => {
17   try {
18     // Validation des données avec Joi
19     const { error, value } = createSquadSchema.validate(req.body);
20     if (error) {
21       return res.status(400).json({
22         success: false,
23         error: 'Données invalides',
24         details: error.details
25       });
26     }
27
28     const squadData = value;
29     const userId = req.user.id;
30
31     // Appel du service métier
32     const squad = await squadService.createSquad(squadData, userId);
33
34     // Audit log
35     await auditService.logAction('squad_created', userId, {
36       squadId: squad.id,
37       squadName: squad.name
38     });
39
40     // Réponse standardisée
41     res.status(201).json({
42       success: true,
43       data: squad,
44       message: 'Escouade créée avec succès'
45     });
46
47   } catch (error) {
48     // Gestion centralisée des erreurs métier
49     if (error.name === 'BusinessError') {
50       return res.status(400).json({
51         success: false,
52         error: error.message
53       });
54     }
55
56     // Journalisation des erreurs techniques
57     logger.error('Error creating squad', {
58       userId: req.user.id,
59       error: error.message,
60       stack: error.stack
61     });
62
63     next(error);
64   }
65 }
```

```

64     }
65 };
66
67 module.exports = { createSquad };

1 // Service de gestion des escouades
2 class SquadService {
3     async createSquad(squadData, leaderId) {
4         // Validation métier
5         if (squadData.maxMembers > 12) {
6             throw new Error('Une escouade ne peut pas dépasser 12 membres');
7         }
8
9         // Vérification des limites utilisateur
10        const userSquadCount = await this.squadRepository.countByUser(leaderId)
11        ;
12        if (userSquadCount >= 5) {
13            throw new Error('Limite de 5 escouades par utilisateur atteinte');
14        }
15
16        // Transaction pour cohérence des données
17        return await this.db.transaction(async (trx) => {
18            // Création de l'escouade
19            const squad = await this.squadRepository.create(trx, {
20                ...squadData,
21                leader_id: leaderId
22            });
23
24            // Ajout automatique du leader
25            await this.squadMemberRepository.create(trx, {
26                squad_id: squad.id,
27                user_id: leaderId,
28                role: 'leader'
29            });
30
31            // Invalidation du cache
32            await this.cacheService.delete(`user_squads:${leaderId}`);
33
34            // Notification asynchrone
35            this.notificationQueue.add('squad_created', {
36                squadId: squad.id,
37                leaderId: leaderId
38            });
39
40            return squad;
41        });
42
43        async getSquadWithMembers(squadId) {
44            // Pattern Cache-Aside
45            const cacheKey = `squad:${squadId}:members`;
46            const cached = await this.cacheService.get(cacheKey);
47
48            if (cached) {
49                return JSON.parse(cached);
50            }
51
52            // Cache miss - lecture base
53            const squad = await this.squadRepository.findByIdWithMembers(squadId);

```

```

54
55     if (squad) {
56         // Mise en cache avec TTL
57         await this.cacheService.setex(
58             cacheKey,
59             300, // 5 minutes
60             JSON.stringify(squad)
61         );
62     }
63
64     return squad;
65 }
66 }
67
68 module.exports = SquadService;

```

```

1 // Middleware d'authentification JWT
2 const authenticateToken = async (req, res, next) => {
3     const authHeader = req.headers['authorization'];
4     const token = authHeader && authHeader.split(' ')[1];
5
6     if (!token) {
7         return res.status(401).json({
8             success: false,
9             error: 'Token manquante'
10        });
11    }
12
13    try {
14        const decoded = jwt.verify(token, process.env.JWT_SECRET);
15        const user = await userRepository.findById(decoded.userId);
16
17        if (!user) {
18            return res.status(401).json({
19                success: false,
20                error: 'Utilisateur non trouvé'
21            });
22        }
23
24        req.user = user;
25        next();
26    } catch (error) {
27        if (error.name === 'TokenExpiredError') {
28            return res.status(401).json({
29                success: false,
30                error: 'Token expiré'
31            });
32        }
33
34        return res.status(403).json({
35            success: false,
36            error: 'Token invalide'
37        });
38    }
39 };
40
41 // Middleware de gestion d'erreur centralisée
42 const errorHandler = (err, req, res, next) => {
43     logger.error('Unhandled error', {

```



```

44     url: req.url,
45     method: req.method,
46     userId: req.user?.id,
47     error: err.message,
48     stack: err.stack
49   });
50
51   // Erreur de validation Joi
52   if (err.isJoi) {
53     return res.status(400).json({
54       success: false,
55       error: 'Données invalides',
56       details: err.details
57     });
58   }
59
60   // Erreur métier
61   if (err.name === 'BusinessError') {
62     return res.status(400).json({
63       success: false,
64       error: err.message
65     });
66   }
67
68   // Erreur base de données
69   if (err.code === '23505') { // Violation de contrainte unique
70     return res.status(409).json({
71       success: false,
72       error: 'Une ressource avec cet identifiant existe déjà'
73     });
74   }
75
76   // Erreur générique
77   res.status(500).json({
78     success: false,
79     error: 'Erreur interne du serveur'
80   });
81 };
82
83 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.

```

1 // schema.prisma
2 generator client {
3   provider = "prisma-client-js"
4 }
5
6 datasource db {
7   provider = "postgresql"

```

```

8   url      = env("DATABASE_URL")
9 }
10
11 model User {
12   id          Int          @id @default(autoincrement())
13   bungie_id   String       @unique
14   display_name String      @db.VarChar(50)
15   email       String?
16   created_at  DateTime     @default(now())
17   last_login  DateTime?
18   role        UserRole     @default(PLAYER)
19   membership_type Int?
20
21   // Relations
22   squads_led   Squad[]    @relation("SquadLeader")
23   squad_members SquadMember[]
24   sessions_created Session[] @relation("SessionCreator")
25   badges       UserBadge[]
26
27   @@map("users")
28 }
29
30 model Squad {
31   id          Int          @id @default(autoincrement())
32   name         String      @db.VarChar(100)
33   description  String?
34   leader_id   Int
35   created_at  DateTime     @default(now())
36   is_public   Boolean      @default(true)
37   max_members Int          @default(6)
38   settings    Json?
39
40   // Relations
41   leader      User         @relation("SquadLeader", fields: [leader_id],
42     references: [id], onDelete: Cascade)
43   members     SquadMember[]
44   sessions    Session[]
45
46   @@map("squads")
47   @@index([leader_id])
48   @@index([created_at])
49 }
50
51 model SquadMember {
52   id          Int          @id @default(autoincrement())
53   squad_id    Int
54   user_id     Int
55   role        SquadRole   @default(MEMBER)
56   joined_at   DateTime     @default(now())
57
58   // Relations
59   squad Squad @relation(fields: [squad_id], references: [id], onDelete:
60     Cascade)
61   user User   @relation(fields: [user_id], references: [id], onDelete:
62     Cascade)
63
64   @@unique([squad_id, user_id])
65   @@map("squad_members")

```

```

63 }

1 // SquadRepository.js
2 class SquadRepository {
3   async createWithMembers(trx, squadData, leaderId) {
4     // Création atomique de l'escouade avec son leader
5     return await trx.squad.create({
6       data: {
7         name: squadData.name,
8         description: squadData.description,
9         leader_id: leaderId,
10        max_members: squadData.maxMembers,
11        is_public: squadData.isPublic,
12        settings: squadData.settings,
13        // Création simultanée du membre leader
14        members: {
15          create: {
16            user_id: leaderId,
17            role: 'leader'
18          }
19        },
20      },
21      include: {
22        leader: {
23          select: {
24            id: true,
25            display_name: true,
26            bungie_id: true
27          }
28        },
29        members: {
30          include: {
31            user: {
32              select: {
33                id: true,
34                display_name: true,
35                bungie_id: true
36              }
37            }
38          }
39        }
40      }
41    });
42  }
43
44  async findSquadsWithStats(userId) {
45    // Requête optimisée avec agrégations
46    const whereClause = userId ? {
47      OR: [
48        { leader_id: userId },
49        { members: { some: { user_id: userId } } }
50      ]
51    } : { is_public: true };
52
53    return await this.db.squad.findMany({
54      where: whereClause,
55      include: {
56        _count: {
57          select: {

```

```

58         members: true,
59         sessions: {
60             where: {
61                 status: 'completed'
62             }
63         }
64     },
65 },
66 leader: {
67     select: {
68         display_name: true,
69         bungie_id: true
70     }
71 },
72 sessions: {
73     take: 1,
74     orderBy: { scheduled_at: 'desc' },
75     select: {
76         scheduled_at: true,
77         status: true
78     }
79 },
80 },
81 orderBy: { created_at: 'desc' }
82 });
83 }
84
85 async countUserSquads(userId) {
86     return await this.db.squad.count({
87         where: {
88             OR: [
89                 { leader_id: userId },
90                 { members: { some: { user_id: userId } } }
91             ]
92         }
93     });
94 }
95 }

```

```

1 // CacheService.js
2 class CacheService {
3     constructor(redisClient) {
4         this.redis = redisClient;
5         this.config = {
6             bungie: { ttl: 3600, prefix: 'bungie:' },
7             guides: { ttl: 1800, prefix: 'guides:' },
8             sessions: { ttl: 86400, prefix: 'session:' },
9             queries: { ttl: 300, prefix: 'query:' }
10        };
11    }
12
13    async cacheBungieData(key, data) {
14        const cacheKey = `${this.config.bungie.prefix}${key}`;
15        await this.redis.setex(
16            cacheKey,
17            this.config.bungie.ttl,
18            JSON.stringify(data)
19        );
20    }

```

```
21
22 async getBungieData(key) {
23   const cacheKey = `${this.config.bungie.prefix}${key}`;
24   const cached = await this.redis.get(cacheKey);
25   return cached ? JSON.parse(cached) : null;
26 }
27
28 async cacheUserSquads(userId, squads) {
29   const cacheKey = `${this.config.queries.prefix}user_squads:${userId}`;
30   await this.redis.setex(
31     cacheKey,
32     this.config.queries.ttl,
33     JSON.stringify(squads)
34   );
35 }
36
37 async getUserSquads(userId) {
38   const cacheKey = `${this.config.queries.prefix}user_squads:${userId}`;
39   const cached = await this.redis.get(cacheKey);
40
41   if (cached) {
42     return JSON.parse(cached);
43   }
44   return null;
45 }
46
47 async invalidateUserSquads(userId) {
48   const cacheKey = `${this.config.queries.prefix}user_squads:${userId}`;
49   await this.redis.del(cacheKey);
50 }
51 }
```

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.

Middleware de sécurité Express spécifique au projet :

```
1 const helmet = require('helmet');
2 const rateLimit = require('express-rate-limit');
3
4 // Configuration Helmet pour Destiny Raid Companion
5 app.use(helmet({
6   contentSecurityPolicy: {
7     directives: {
8       defaultSrc: ["'self'", "https://www.bungie.net"],
9       styleSrc: ["'self'", "'unsafe-inline'", "https://fonts.googleapis.com"],
10      scriptSrc: ["'self'", "https://www.bungie.net"],
11      imgSrc: ["'self'", "data:", "https://www.bungie.net", "https://assets.destinyraidcompanion.com"],
12      connectSrc: ["'self'", "https://www.bungie.net", "https://api.destinyraidcompanion.com"]
13    }
14  },
15  hsts: {
16    maxAge: 31536000,
17    includeSubDomains: true,
18    preload: true
19  },
20  xFrameOptions: { action: 'deny' }
21 }));
22
23 // Rate limiting adapté aux besoins de l'API Bungie
24 const apiLimiter = rateLimit({
25   windowMs: 15 * 60 * 1000,
26   max: 150,
27   message: 'Trop de requêtes vers l\'API Bungie'
28 });
29 app.use('/api/bungie/', apiLimiter);
30
31 const authLimiter = rateLimit({
32   windowMs: 15 * 60 * 1000,
33   max: 10,
34   message: 'Trop de tentatives de connexion'
35 });
36 app.use('/api/auth/', authLimiter);
```

```

37
38 // Validation spécifique aux données Destiny 2
39 const validateDestinyData = (req, res, next) => {
40   const { characterId, membershipType, destinyMembershipId } = req.body;
41
42   // Validation des formats Bungie
43   if (characterId && !/^\\d{19}$/.test(characterId)) {
44     return res.status(400).json({ error: 'Format_de_characterId_invalide'
45     });
46   }
47
48   if (membershipType && ![1, 2, 3, 4, 5, 10].includes(Number(membershipType
49   ))) {
50     return res.status(400).json({ error: 'Type_de_membership_invalide' });
51   }
52   next();
53 };

```

Protection XSS pour les guides interactifs :

```

1 // Composant sécurisé pour les guides de raid
2 const RaidGuide = ({ guideData }) => {
3   const sanitizeHtml = (html) => {
4     // Utilisation de DOMPurify pour le contenu dynamique
5     return DOMPurify.sanitize(html, {
6       ALLOWED_TAGS: ['p', 'strong', 'em', 'ul', 'li', 'h2', 'h3'],
7       ALLOWED_ATTR: ['class']
8     });
9   };
10
11   return (
12     <div className="raid-guide">
13       <h1>{guideData.title}</h1>
14       <div
15         className="guide-content"
16         dangerouslySetInnerHTML={{
17           __html: sanitizeHtml(guideData.content)
18         }}
19       />
20       { /* Données utilisateur échappées automatiquement */ }
21       <div className="author-info">
22         <span>Auteur: {guideData.author}</span>
23         <span>Mis à jour: {new Date(guideData.updatedAt).toLocaleDateString
24         ()}</span>
25       </div>
26     </div>
27   );
28 };
29
30 // Validation des données de raid avec Zod
31 const raidSchema = z.object({
32   name: z.string().min(1).max(100),
33   description: z.string().max(1000),
34   difficulty: z.enum(['novice', 'normal', 'maître']),
35   recommendedPower: z.number().min(0).max(2000),
36   estimatedTime: z.number().min(15).max(180)
37 });

```

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

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.

Configuration OAuth Bungie et JWT :

```

1 const axios = require('axios');
2 const jwt = require('jsonwebtoken');
3 const crypto = require('crypto');
4
5 // Configuration Bungie OAuth
6 const BUNGIE_CLIENT_ID = process.env.BUNGIE_CLIENT_ID;
7 const BUNGIE_CLIENT_SECRET = process.env.BUNGIE_CLIENT_SECRET;
8 const BUNGIE_OAUTH_URL = 'https://www.bungie.net/en/OAuth/Authorize';
9 const BUNGIE_TOKEN_URL = 'https://www.bungie.net/platform/app/oauth/token/';
10
11 // Configuration JWT interne
12 const JWT_SECRET = process.env.JWT_SECRET;
13 const JWT_EXPIRES_IN = '1h'; // Court pour la sécurité
14 const REFRESH_EXPIRES_IN = '30d';
15
16 // Échange du code OAuth contre un token Bungie
17 const exchangeBungieCode = async (code) => {
18   const response = await axios.post(BUNGIE_TOKEN_URL,
19     new URLSearchParams({
20       grant_type: 'authorization_code',
21       code: code,
22       client_id: BUNGIE_CLIENT_ID,
23       client_secret: BUNGIE_CLIENT_SECRET
24     }), {
25     headers: {
26       'Content-Type': 'application/x-www-form-urlencoded'
27     }
28   });
29
30   return {
31     accessToken: response.data.access_token,
32     refreshToken: response.data.refresh_token,
33     expiresIn: response.data.expires_in,
34     membershipId: response.data.membership_id
35   };
36 };
37
38 // Génération des tokens JWT internes
39 const generateInternalTokens = (bungieMembershipId, displayName, role) => {

```

```

41  const accessToken = jwt.sign(
42    {
43      bungieId: bungieMembershipId,
44      displayName: displayName,
45      role: role,
46      type: 'access',
47      iss: 'destiny-raid-companion'
48    },
49    JWT_SECRET,
50    { expiresIn: JWT_EXPIRES_IN }
51  );
52
53  const refreshToken = crypto.randomBytes(64).toString('hex');
54
55  // Stockage sécurisé du refresh token
56  await redis.setex(
57    `refresh:${bungieMembershipId}`,
58    30 * 24 * 60 * 60, // 30 jours
59    refreshToken
60  );
61
62  return { accessToken, refreshToken };
63 };
64
65 // Middleware d'authentification hybride (Bungie + JWT)
66 const authenticateUser = async (req, res, next) => {
67   try {
68     // Vérification du token JWT
69     const authHeader = req.headers['authorization'];
70     const token = authHeader && authHeader.split(' ')[1];
71
72     if (!token) {
73       return res.status(401).json({ error: 'Token_manquant' });
74     }
75
76     const decoded = jwt.verify(token, JWT_SECRET);
77
78     // Vérification de la validité du token Bungie
79     const bungieToken = await redis.get(`bungie_token:${decoded.bungieId}`)
80     ;
81     if (!bungieToken) {
82       // Tentative de refresh
83       const newToken = await refreshBungieToken(decoded.bungieId);
84       if (!newToken) {
85         return res.status(401).json({ error: 'Session_expirée' });
86       }
87
88       req.user = decoded;
89       req.user.bungieToken = bungieToken;
90       next();
91     } catch (error) {
92       return res.status(403).json({ error: 'Token_invalide' });
93     }
94   };

```

Système de rôles et permissions spécifique :

```

1 // Permissions spécifiques à Destiny Raid Companion

```

```
2 const PERMISSIONS = {
3   // Guides
4   GUIDE_CREATE: 'guide:create',
5   GUIDE_READ: 'guide:read',
6   GUIDE_UPDATE: 'guide:update',
7   GUIDE_DELETE: 'guide:delete',
8
9   // Escouades
10  SQUAD_CREATE: 'squad:create',
11  SQUAD_INVITE: 'squad:invite',
12  SQUAD_KICK: 'squad:kick',
13  SQUAD_DELETE: 'squad:delete',
14
15  // Calendrier
16  EVENT_CREATE: 'event:create',
17  EVENT_EDIT: 'event:edit',
18  EVENT_DELETE: 'event:delete',
19
20  // Administration
21  USER_BAN: 'user:ban',
22  CONTENT_MODERATE: 'content:moderate',
23  SYSTEM_CONFIG: 'system:config'
24 };
25
26 // Rôles avec permissions adaptées
27 const ROLES = {
28   PLAYER: [
29     PERMISSIONS.GUIDE_READ,
30     PERMISSIONS.SQUAD_CREATE,
31     PERMISSIONS.EVENT_CREATE
32   ],
33
34   SQUAD_LEADER: [
35     ...ROLES.PLAYER,
36     PERMISSIONS.SQUAD_INVITE,
37     PERMISSIONS.SQUAD_KICK,
38     PERMISSIONS.EVENT_EDIT,
39     PERMISSIONS.EVENT_DELETE
40   ],
41
42   GUIDE_WRITER: [
43     ...ROLES.PLAYER,
44     PERMISSIONS.GUIDE_CREATE,
45     PERMISSIONS.GUIDE_UPDATE
46   ],
47
48   MODERATOR: [
49     ...ROLES.SQUAD_LEADER,
50     ...ROLES.GUIDE_WRITER,
51     PERMISSIONS.USER_BAN,
52     PERMISSIONS.CONTENT_MODERATE
53   ],
54
55   ADMIN: Object.values(PERMISSIONS)
56 };
57
58 // Middleware de vérification de permission
59 const requirePermission = (permission) => {
```

```

60 return (req, res, next) => {
61   const userRole = req.user.role;
62   const userPermissions = ROLES[userRole] || [];
63
64   if (!userPermissions.includes(permission)) {
65     return res.status(403).json({
66       error: 'Permission_refusée',
67       required: permission,
68       userPermissions: userPermissions
69     });
70   }
71
72   // Audit log
73   await auditService.logPermissionCheck(
74     req.user.bungieId,
75     permission,
76     req.path,
77     'success'
78   );
79
80   next();
81 };
82 };
83
84 // Utilisation dans les routes
85 router.post('/squads',
86   authenticateUser,
87   requirePermission(PERMISSIONS.SQUAD_CREATE),
88   squadController.createSquad
89 );
90
91 router.put('/guides/:id',
92   authenticateUser,
93   requirePermission(PERMISSIONS.GUIDE_UPDATE),
94   guideController.updateGuide
95 );

```

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.

Registre des traitements pour Destiny Raid Companion :

```

1 const gdprRegistry = {
2   'user-authentication': {
3     purpose: 'Authentification_via_Bungie.net_et_gestion_du_compte',
4     legalBasis: 'Consentement_explicite_lors_de_la_connexion_OAuth',
5     dataCategories: [
6       'Bungie_Membership_ID',
7       'Nom_d\'affichage',
8       'Tokens_d\'accès_OAuth',
9       'Date_de_dernière_connexion'
10    ],
11     retentionPeriod: '3_ans_après_dernière_activité',
12     recipients: ['Équipe_technique', 'Bungie(via_API)'],

```

```

13     transfers: ['France_(hébergement)', 'États-Unis_(API_Bungie)'],
14     safeguards: 'Clauses_contractuelles_types_avec_Bungie'
15 },
16
17 'game-statistics': {
18     purpose: 'Affichage_des_statistiques_de_jeu_et_progression',
19     legalBasis: 'Exécution_du_contrat_(service_demandé)',
20     dataCategories: [
21         'Niveau_de_lumière',
22         'Temps_de_jeu',
23         'Raids_complétés',
24         'Succès_débloqués',
25         'Équipement_possédé'
26     ],
27     retentionPeriod: '5_ans_après_fermeture_du_compte',
28     recipients: ['Utilisateur_uniquement'],
29     transfers: ['France_uniquement'],
30     safeguards: 'Chiffrement_AES-256_au_repos'
31 },
32
33 'squad-management': {
34     purpose: 'Gestion_des_escouades_et_communication_entre_joueurs',
35     legalBasis: 'Intérêt_légitime_(fonctionnalité_collaborative)',
36     dataCategories: [
37         'Liste_des_membres_\''escouade'',
38         'Messages_dans_le_chat_\''escouade'',
39         'Dates_des_sessions_planifiées',
40         'Notes_de_progression'
41     ],
42     retentionPeriod: '2_ans_après_dissolution_de_\''escouade'',
43     recipients: ['Membres_de_\''escouade'],
44     transfers: ['France_uniquement'],
45     safeguards: 'Accès_contrôlé_par_système_de_permissions'
46 },
47
48 'raid-guides': {
49     purpose: 'Publication_et_consultation_de_guides_de_raids',
50     legalBasis: 'Intérêt_légitime_(partage_communautaire)',
51     dataCategories: [
52         'Contenu_des_guides',
53         'Auteur',
54         'Date_de_publication',
55         'Notes_et_commentaires'
56     ],
57     retentionPeriod: 'Indéfiniment_(contenu_public)',
58     recipients: ['Tous_les_utilisateurs'],
59     transfers: ['France_uniquement'],
60     safeguards: 'Modération_manuelle_et_automatique'
61 }
62 };
63
64 // API RGPD complète
65 const gdprController = {
66     // Droit d'accès - Export complet des données
67     async exportUserData(req, res) {
68         const bungieId = req.user.bungieId;
69
70         // Récupération de toutes les données

```

```
71     const userData = {
72       profile: await userService.getProfile(bungieId),
73       statistics: await statsService.getUserStats(bungieId),
74       squads: await squadService.getUserSquads(bungieId),
75       events: await calendarService.getUserEvents(bungieId),
76       guides: await guideService.getUserGuides(bungieId),
77       processingPurposes: gdprRegistry
78     };
79
80     // Format standardisé RGPD
81     res.json({
82       requestDate: new Date().toISOString(),
83       userId: bungieId,
84       data: userData,
85       metadata: {
86         format: 'JSON',
87         version: '1.0',
88         schema: 'https://gdpr.destinyraidcompanion.com/schema.json'
89       }
90     });
91   },
92
93   // Droit à l'effacement - Anonymisation progressive
94   async deleteUserData(req, res) {
95     const bungieId = req.user.bungieId;
96
97     // 1. Marquer le compte comme supprimé
98     await userService.softDelete(bungieId);
99
100    // 2. Anonymiser les données personnelles
101    await userService.anonymizePersonalData(bungieId);
102
103    // 3. Conserver les données nécessaires (facturation, légal)
104    await auditService.logGdprDeletion(bungieId, req.ip);
105
106    // 4. Planifier la suppression complète après délai
107    scheduleCompleteDeletion(bungieId, 30); // 30 jours
108
109    res.json({
110      message: 'Demande de suppression prise en compte',
111      deletionDate: new Date(Date.now() + 30 * 24 * 60 * 60 * 1000),
112      reference: generateDeletionReference(bungieId)
113    });
114  },
115
116  // Droit à la portabilité - Format standard
117  async exportDataPortability(req, res) {
118    const bungieId = req.user.bungieId;
119    const format = req.query.format || 'json';
120
121    const data = await gdprService.exportForPortability(bungieId);
122
123    if (format === 'csv') {
124      const csv = convertToCSV(data);
125      res.attachment(`destiny-data-${bungieId}.csv`);
126      res.type('text/csv').send(csv);
127    } else {
128      res.attachment(`destiny-data-${bungieId}.json`);
```

```

129     res.json(data);
130   }
131 },
132
133 // Droit d'opposition - Désinscription ciblée
134 async updateConsent(req, res) {
135   const { bungieId } = req.user;
136   const { processing, consent } = req.body;
137
138   if (!gdprRegistry[processing]) {
139     return res.status(400).json({ error: 'Traitement_inconnu' });
140   }
141
142   await consentService.updateConsent(bungieId, processing, consent);
143
144   // Appliquer immédiatement si refus
145   if (!consent) {
146     await gdprService.stopProcessing(bungieId, processing);
147   }
148
149   res.json({
150     message: 'Consentement_mis_à_jour',
151     processing: processing,
152     consent: consent,
153     effectiveDate: new Date().toISOString()
154   });
155 }
156 };

```

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

6.4 Sécurité des données et monitoring

Système de chiffrement pour les données Destiny 2 :

```

1 const crypto = require('crypto');
2
3 class DestinyDataEncryptor {

```

```
4 constructor() {
5   this.algorithm = 'aes-256-gcm';
6   this.key = Buffer.from(process.env.ENCRIPTION_KEY, 'hex');
7
8   if (this.key.length !== 32) {
9     throw new Error('Clé de chiffrement invalide (doit être 32 bytes)');
10  }
11 }
12
13 // Chiffrement des tokens Bungie (très sensibles)
14 encryptBungieToken(tokenData) {
15   const iv = crypto.randomBytes(16);
16   const cipher = crypto.createCipheriv(this.algorithm, this.key, iv);
17
18   // Données additionnelles pour authentification
19   const aad = Buffer.from('bungie-token');
20   cipher.setAAD(aad);
21
22   let encrypted = cipher.update(JSON.stringify(tokenData), 'utf8', 'hex')
23   ;
24   encrypted += cipher.final('hex');
25   const authTag = cipher.getAuthTag();
26
27   return {
28     encrypted,
29     iv: iv.toString('hex'),
30     authTag: authTag.toString('hex'),
31     version: '1.0',
32     timestamp: new Date().toISOString()
33   };
34 }
35
36 // Chiffrement des données utilisateur sensibles
37 encryptUserData(userData, userId) {
38   const iv = crypto.randomBytes(16);
39   const cipher = crypto.createCipheriv(this.algorithm, this.key, iv);
40
41   // Associer le chiffrement à l'utilisateur
42   const aad = Buffer.from(`user:${userId}`);
43   cipher.setAAD(aad);
44
45   let encrypted = cipher.update(JSON.stringify(userData), 'utf8', 'hex');
46   encrypted += cipher.final('hex');
47   const authTag = cipher.getAuthTag();
48
49   // Hacher pour vérification d'intégrité
50   const hash = crypto.createHash('sha256')
51     .update(encrypted)
52     .digest('hex');
53
54   return {
55     encrypted,
56     iv: iv.toString('hex'),
57     authTag: authTag.toString('hex'),
58     hash,
59     userId,
60     encryptedAt: new Date().toISOString()
61   };
62 }
```



```

61 }
62
63 // Vérification et déchiffrement
64 decryptAndVerify(encryptedData) {
65   try {
66     const decipher = crypto.createDecipheriv(
67       this.algorithm,
68       this.key,
69       Buffer.from(encryptedData.iv, 'hex')
70     );
71
72     // Vérification des données additionnelles
73     if (encryptedData.userId) {
74       decipher.setAAD(Buffer.from(`user:${encryptedData.userId}`));
75     } else {
76       decipher.setAAD(Buffer.from('bungie-token'));
77     }
78
79     decipher.setAuthTag(Buffer.from(encryptedData.authTag, 'hex'));
80
81     // Vérification de l'intégrité
82     if (encryptedData.hash) {
83       const currentHash = crypto.createHash('sha256')
84         .update(encryptedData.encrypted)
85         .digest('hex');
86
87       if (currentHash !== encryptedData.hash) {
88         throw new Error('Hash de vérification invalide');
89       }
90     }
91
92     let decrypted = decipher.update(encryptedData.encrypted, 'hex', 'utf8');
93     decrypted += decipher.final('utf8');
94
95     return JSON.parse(decrypted);
96   } catch (error) {
97     // Log de sécurité en cas d'échec
98     securityLogger.logDecryptionFailure({
99       error: error.message,
100       dataId: encryptedData.userId || 'unknown',
101       timestamp: new Date().toISOString()
102     });
103     throw new Error('Échec du déchiffrement');
104   }
105 }
106 }
107
108 // Rotation automatique des clés
109 class KeyManager {
110   constructor() {
111     this.currentKey = process.env.ENCRIPTION_KEY;
112     this.previousKey = process.env.PREVIOUS_ENCRYPTION_KEY;
113     this.rotationInterval = 90 * 24 * 60 * 60 * 1000; // 90 jours
114   }
115
116   async rotateKeys() {
117     const newKey = crypto.randomBytes(32).toString('hex');

```

```

118
119 // 1. Chiffrer toutes les données avec la nouvelle clé
120 await this.reencryptAllData(newKey);
121
122 // 2. Mettre à jour les variables d'environnement
123 process.env.PREVIOUS_ENCRYPTION_KEY = this.currentKey;
124 process.env.ENCRYPTION_KEY = newKey;
125
126 // 3. Loguer la rotation
127 await auditService.logKeyRotation({
128   oldKeyHash: crypto.createHash('sha256').update(this.currentKey).
129     digest('hex'),
130   newKeyHash: crypto.createHash('sha256').update(newKey).digest('hex'),
131   rotatedBy: 'system',
132   timestamp: new Date().toISOString()
133 });
134
135 this.currentKey = newKey;
136 }
137
138 async reencryptAllData(newKey) {
139   // Implémentation de re-chiffrement progressif
140   const batchSize = 100;
141   let lastId = 0;
142
143   do {
144     const users = await db.users.findMany({
145       where: { id: { gt: lastId } },
146       take: batchSize,
147       orderBy: { id: 'asc' }
148     });
149
150     for (const user of users) {
151       // Décrypter avec l'ancienne clé
152       const decryptor = new DestinyDataEncryptor(this.currentKey);
153       const decrypted = decryptor.decryptAndVerify(user.encryptedData);
154
155       // Recrypter avec la nouvelle clé
156       const newEncryptor = new DestinyDataEncryptor(newKey);
157       const reencrypted = newEncryptor.encryptUserData(decrypted, user.id
158         );
159
160       // Mettre à jour en base
161       await db.users.update({
162         where: { id: user.id },
163         data: { encryptedData: reencrypted }
164       });
165     }
166
167     lastId = users[users.length - 1]?.id || 0;
168   } while (lastId > 0);
169 }

```

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) :

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

Exemple de test unitaire (2/2) :

```
1 describe('createProject', () => {
2   it('should create a project with valid data', async () => {
3     // Arrange
4     const projectData = {
5       name: 'Test Project',
6       description: 'Test Description',
7       userId: 'user123'
8     };
9     const expectedProject = { id: 'proj123', ...projectData };
10    mockRepository.create.mockResolvedValue(expectedProject);
11
12    // Act
13    const result = await projectService.createProject(projectData);
14
15    // Assert
16    expect(mockRepository.create).toHaveBeenCalledWith(projectData);
17    expect(result).toEqual(expectedProject);
18  });
19
20  it('should throw error for invalid project data', async () => {
21    // Arrange
```

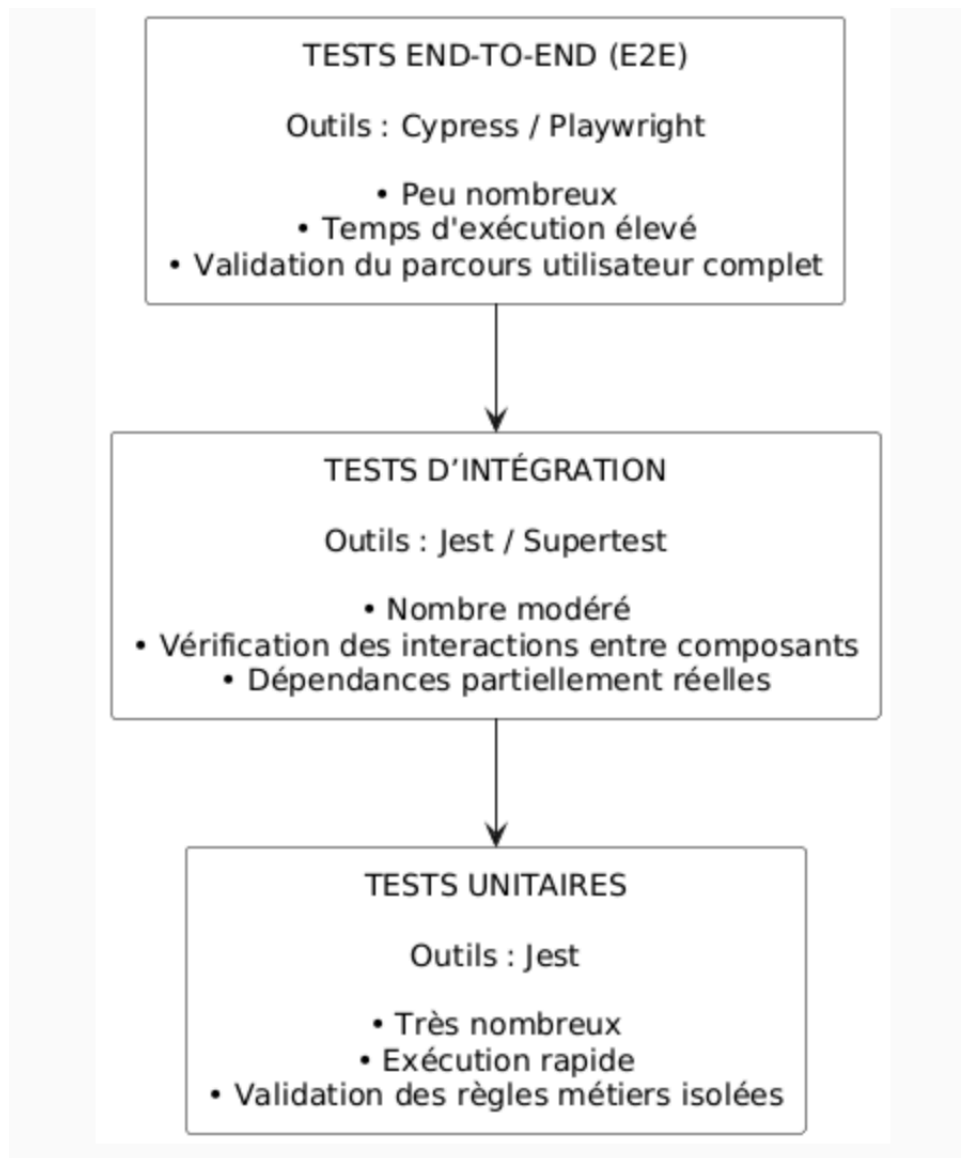


Figure 7.1 – Pyramide de tests

```

22     const invalidData = { name: '' }; // Nom vide
23
24     // Act & Assert
25     await expect(projectService.createProject(invalidData))
26       .rejects.toThrow('Le nom du projet est requis');
27   });
28 });
29 });

```

Exemple de test d'intégration :

```

1 // Test d'intégration pour l'API
2 describe('Project API Integration', () => {
3   let app;
4   let authToken;
5
6   beforeAll(async () => {
7     app = await createTestApp();
8     authToken = await getTestAuthToken();

```

```

9   });
10
11  describe('POST /api/projects', () => {
12    it('should create a project with authentication', async () => {
13      const projectData = {
14        name: 'Integration Test Project',
15        description: 'Test Description'
16      };
17
18      const response = await request(app)
19        .post('/api/projects')
20        .set('Authorization', `Bearer ${authToken}`)
21        .send(projectData)
22        .expect(201);
23
24      expect(response.body).toMatchObject({
25        id: expect.any(String),
26        name: projectData.name,
27        description: projectData.description
28      });
29    });
30
31    it('should reject request without authentication', async () => {
32      const projectData = { name: 'Test Project' };
33
34      await request(app)
35        .post('/api/projects')
36        .send(projectData)
37        .expect(401);
38    });
39  });
40 });

```

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 goulets 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) :

```

1  import http from 'k6/http';
2  import { check, sleep } from 'k6';
3  import { Rate } from 'k6/metrics';
4
5  // Métriques personnalisées
6  const errorRate = new Rate('errors');
7
8  export let options = {
9    stages: [
10     { duration: '2m', target: 10 }, // Montée en charge
11     { duration: '5m', target: 50 }, // Charge normale
12     { duration: '2m', target: 100 }, // Pic de charge

```

```

13   { duration: '5m', target: 50 }, // Retour à la normale
14   { duration: '2m', target: 0 }, // Descente
15 ],
16 thresholds: {
17   http_req_duration: ['p(95)<500'], // 95% des requêtes < 500ms
18   http_req_failed: ['rate<0.1'], // Moins de 10% d'erreurs
19   errors: ['rate<0.1']
20 }
21 };

```

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

```

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

```

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 :

```

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

```

Rapport de qualité SonarQube :

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

Exemple de correction de code smell :

```

1 // AVANT : Méthode trop longue
2 const processData = (userData) => {
3   const validatedData = validateUserData(userData);
4   const processedData = transformUserData(validatedData);
5   const enrichedData = enrichWithExternalData(processedData);
6   const formattedData = formatForDatabase(enrichedData);
7   const savedData = saveToDatabase(formattedData);
8   const auditLog = createAuditLog(savedData);
9   const notification = sendNotification(auditLog);
10  return notification;

```

```

11 };
12
13 // APRÈS : Méthodes courtes et focalisées
14 const processUserData = (userData) => {
15   const validatedData = validateUserData(userData);
16   const processedData = transformUserData(validatedData);
17   return saveUserData(processedData);
18 };
19
20 const saveUserData = (data) => {
21   const enrichedData = enrichWithExternalData(data);
22   const formattedData = formatForDatabase(enrichedData);
23   const savedData = saveToDatabase(formattedData);
24   auditUserAction(savedData);
25   return savedData;
26 };

```

Intégration SonarQube dans GitHub Actions :

```

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

```

Métriques de qualité GitHub :

- **Couverture** : 85% (objectif : >80%)
- **Bugs** : 0 (objectif : 0)
- **Vulnérabilités** : 0 (objectif : 0)
- **Code smells** : 12 (objectif : <20)
- **Duplication** : 1.2% (objectif : <3%)
- Métriques Lighthouse :**
- **Performance** : 92/100 (objectif : >90)
- **Accessibilité** : 95/100 (objectif : >90)
- **Best Practices** : 88/100 (objectif : >85)
- **SEO** : 90/100 (objectif : >85)

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)

```
1 FROM node:18-alpine
2
3 WORKDIR /app
4 COPY package*.json ./
5 RUN npm ci --only=production
6 COPY . .
7
8 EXPOSE 3000
9 ENV NODE_ENV=production
10
11 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)

```
1 FROM nginx:alpine
2
3 COPY ./public /usr/share/nginx/html
4 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

```
1 version: "3.8"
2
3 services:
4   frontend:
5     build: ./frontend
6     ports:
7       - "8080:80"
8     depends_on:
9       - backend
10
11   backend:
12     build: ./backend
13     ports:
14       - "3000:3000"
15     environment:
16       - NODE_ENV=production
17       - 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

```
1 name: CI Destiny Companion
2
3 on:
4   push:
5     branches: [main]
6   pull_request:
7     branches: [main]
8
9 jobs:
10   ci:
11     runs-on: ubuntu-latest
12
```

```
13  steps:
14    - uses: actions/checkout@v3
15
16    - name: Installation Node.js
17      uses: actions/setup-node@v3
18      with:
19        node-version: 18
20
21    - name: Installation des dépendances
22      run: |
23        cd backend
24        npm ci
25
26    - name: Lint du code
27      run: |
28        cd backend
29        npm run lint
30
31    - name: Tests unitaires
32      run: |
33        cd backend
34        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 :

```
1 // scripts/dependency-monitor.js
2 const fs = require('fs');
3 const axios = require('axios');
4
5 class DependencyMonitor {
6   constructor() {
7     this.packages = ['express', 'pg', 'ioredis', 'jsonwebtoken'];
8   }
9
10  async checkForUpdates() {
11    const updates = [];
12    for (const pkg of this.packages) {
13      const response = await axios.get(
```

```

14     `https://registry.npmjs.org/${pkg}`
15   );
16   const latest = response.data['dist-tags'].latest;
17   updates.push({ package: pkg, latest });
18 }
19 return updates;
20 }
21 }

```

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 :

```

1 const crypto = require('crypto');
2
3 class TokenEncryptor {
4   encryptBungieToken(token) {
5     const algorithm = 'aes-256-gcm';
6     const key = Buffer.from(process.env.ENCRIPTION_KEY, 'hex');
7     const iv = crypto.randomBytes(16);
8
9     const cipher = crypto.createCipheriv(algorithm, key, iv);
10    let encrypted = cipher.update(token, 'utf8', 'hex');
11    encrypted += cipher.final('hex');
12
13    return {
14      encrypted,
15      iv: iv.toString('hex'),
16      authTag: cipher.getAuthTag().toString('hex')
17    };
18  }
19 }

```

Validation des guides de raid :

```

1 const Joi = require('joi');
2
3 const raidGuideSchema = Joi.object({
4   title: Joi.string().max(200).required(),
5   description: Joi.string().max(5000),
6   difficulty: Joi.string().valid('novice', 'normal', 'master'),
7   steps: Joi.array().items(
8     Joi.object({
9       title: Joi.string().max(100).required(),
10      description: Joi.string().max(1000).required()
11    })
12  ).min(1).max(20)
13 });
14
15 function validateRaidGuide(guideData) {
16   const { error, value } = raidGuideSchema.validate(guideData);

```

```

17   if (error) {
18       throw new ValidationError(error.details);
19   }
20   return value;
21 }

```

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 :

```

1 FROM node:20-alpine AS builder
2 WORKDIR /app
3 COPY package*.json ./
4 RUN npm ci --only=production --audit
5 COPY . .
6 RUN npm run build
7
8 FROM node:20-alpine
9 RUN addgroup -g 1001 -S nodejs && \
10    adduser -S nodeuser -u 1001
11 WORKDIR /app
12 COPY --from=builder /app/node_modules ./node_modules
13 COPY --from=builder /app/dist ./dist
14 COPY --from=builder /app/package*.json ./
15 USER nodeuser
16 EXPOSE 3000
17 CMD ["node", "dist/index.js"]

```

Health checks Docker Compose :

```

1 services:
2   backend:
3     healthcheck:
4       test: ["CMD", "node", "-e",
5             "require('http').get('http://localhost:3000/health',
6             (r) => process.exit(r.statusCode === 200 ? 0 : 1))"]
7       interval: 30s
8       timeout: 10s
9       retries: 3
10
11   postgres:
12     healthcheck:
13       test: ["CMD-SHELL", "pg_isready -U destiny"]
14       interval: 10s
15       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 :

```

1 -- Table des utilisateurs
2 CREATE TABLE users (
3     id SERIAL PRIMARY KEY,
4     bungie_id VARCHAR(100) UNIQUE NOT NULL,
5     display_name VARCHAR(100) NOT NULL,
6     created_at TIMESTAMPTZ DEFAULT NOW()
7 );
8
9 -- Table des escouades
10 CREATE TABLE squads (
11     id SERIAL PRIMARY KEY,
12     name VARCHAR(100) NOT NULL,
13     leader_id INTEGER REFERENCES users(id),
14     created_at TIMESTAMPTZ DEFAULT NOW()
15 );
16
17 -- Table des sessions de raid
18 CREATE TABLE raid_sessions (
19     id SERIAL PRIMARY KEY,
20     squad_id INTEGER REFERENCES squads(id),
21     raid_name VARCHAR(100) NOT NULL,
22     scheduled_at TIMESTAMPTZ NOT NULL
23 );

```

Index optimisés :

```

1 CREATE INDEX idx_users_bungie_id ON users(bungie_id);
2 CREATE INDEX idx_squads_leader_id ON squads(leader_id);
3 CREATE INDEX idx_raid_sessions_scheduled ON raid_sessions(scheduled_at);
4 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 :

```

1 groups:
2   - name: destiny-security
3     rules:
4       - alert: HighFailedLogins
5         expr: rate(auth_failed_total[5m]) > 5
6         for: 2m
7         labels:

```

```

8       severity: warning
9       annotations:
10        summary: "Trop de tentatives de connexion échouées"
11
12    - alert: BungieAPIHighErrorRate
13      expr: rate(bungie_api_errors_total[5m]) /
14            rate(bungie_api_calls_total[5m]) > 0.05
15      for: 5m
16      labels:
17        severity: critical

```

Script de monitoring :

```

1 class SecurityMonitor {
2   async checkDependencies() {
3     const packageJson = JSON.parse(
4       fs.readFileSync('./package.json', 'utf8')
5     );
6
7     // Vérification des vulnérabilités
8     const audit = await this.runNpmAudit();
9
10    return {
11      dependencies: Object.keys(packageJson.dependencies).length,
12      vulnerabilities: audit.vulnerabilities,
13      timestamp: new Date().toISOString()
14    };
15  }
16 }

```

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 :

```

1 const incidentChecklist = {
2   detection: [
3     "Vérifier les alertes Prometheus",
4     "Consulter les logs d'application",
5     "Tester les endpoints critiques"
6   ],
7   containment: [

```

```

8     "Activer_les_mode_maintenance",
9     "Isoler_le_service_affecté",
10    "Sauvegarder_les_logs_pertinents"
11  ],
12  eradication: [
13    "Identifier_la_cause_racine",
14    "Appliquer_le_correctif",
15    "Valider_la_correction"
16  ],
17  recovery: [
18    "Redémarrer_le_service",
19    "Vérifier_la_fonctionnalité",
20    "Surveiller_la_stabilité"
21  ]
22  };

```

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 :

```

1 name: Security Scan
2 on: [push, pull_request]
3
4 jobs:
5   security:
6     runs-on: ubuntu-latest
7     steps:
8       - uses: actions/checkout@v3
9       - name: NPM Audit
10        run: npm audit --audit-level=high
11       - name: Snyk Security
12        uses: snyk/actions/node@master
13       - name: Trivy Container Scan
14        uses: aquasecurity/trivy-action@master

```

Script d'audit automatisé :

```

1 const { execSync } = require('child_process');
2
3 class SecurityAudit {
4   async runDailyAudit() {

```

```
5   const audit = {
6     date: new Date().toISOString(),
7     npm: await this.auditNpm(),
8     docker: await this.auditDocker(),
9     code: await this.auditCode()
10  };
11
12  // Génération rapport
13  fs.writeFileSync(
14    `security-audit-${audit.date.split('T')[0]}.json`,
15    JSON.stringify(audit, null, 2)
16  );
17
18  return audit;
19 }
20 }
```

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 principe

Exemple de difficulté résolue :

Problème: Latence élevée des requêtes PostgreSQL

```

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

```

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

10.3 Dettes techniques et apprentissages

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

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

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

Exemple

Registre des dettes techniques :

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

Apprentissages transférables :

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

Exemple d'apprentissage concret :

```

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

```

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

10.4 Liens utiles

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

Chapitre 11

Conclusion et remerciements

11.1 Synthèse du projet

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

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

Exemple

Chiffres clés du projet :

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

Technologies maîtrisées :

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

11.2 Perspectives d'évolution

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

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

Exemple

Roadmap technique v2.0 :

Q1 2025: Fonctionnalités avancées

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

Q2 2025: Intelligence artificielle

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

Q3 2025: Évolutions technologiques

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

Compétences à développer :

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

11.3 Remerciements

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

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

Exemple**Remerciements personnalisés :**

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

Apprentissages clés :

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

11.4 Déploiement et documentation

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

opérationnelle et de la maintenabilité de votre solution.

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

11.4.1 Docker

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

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

Conteneurisation

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

Exemple

Dockerfile multi-stage :

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

Compose

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

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

```

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

```

11.4.2 GitHub (code source)

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

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

Exemple**Structure du repository :**

```

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

```

11.4.3 CI/CD

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

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

Exemple

Pipeline CI/CD GitHub Actions :

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

11.4.4 SonarQube

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

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

Exemple**Métriques de qualité SonarQube :**

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

11.4.5 Swagger

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

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

Exemple**Documentation API Swagger :**

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

11.5 Liens utiles

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