Colint School

Formation CDA 2025

Concepteur Développeur d'Applications

RNCP37873

54 Rue d'Autun, 71100 Chalon-sur-Saône

[Destiny Raid Companion]

Dossier de Projet CDA
Présentation et Défense

Informations du Projet

Candidat :	[Rochetin Lucas]
Promotion :	[Colint School]
Soutenance :	[Date]

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

Année académique 2025-2026

Table des matières

Chapitre 1
Présentation personnelle et du projet

1.1 Rôle du candidat et contexte

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

Contexte organisationnel : *Étant un joueur vétéran du jeu Destiny 2, j'ai identifié plusieurs problématiques récurrentes affectant l'expérience des joueurs. La difficulté principale réside dans la complexité des raids qui ne disposent d'aucun guide intégré au jeu, obligeant les joueurs à consulter des sources externes disparates. Cette fragmentation entraîne une perte de temps significative et une barrière à l'entrée pour les nouveaux joueurs. Une étude interne auprès de 200 joueurs montre que 78% 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*

Votre pitch QQOQCP :
- **Quoi :** Destiny Raid Companion - plateforme web centralisant guides interactifs, gestion d'escouades et calendrier collaboratif
- **Qui :** Joueurs de Destiny 2 (débutants cherchant de la clarté et joueurs expérimentés recherchant l'optimisation)
- **Où :** Application web responsive accessible sur tous devices, déployée sur cloud
- **Quand :** Développement octobre 2025 - mai 2026, MVP déployé en mars 2026

— **Comment :** Architecture 3-tiers (React/Node.js/PostgreSQL) avec intégration API Bungie
— **Pourquoi :** Réduire de 55% le temps d'organisation et diminuer de 50% le taux d'abandon des nouveaux joueurs

1.2 Problématique et objectifs SMART

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

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

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

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

Objectifs SMART :
— **Spécifique :** Développer une plateforme unifiée avec guides interactifs clarifiés, système d'escouades inclusif et calendrier collaboratif
— **Mesurable :**
  — Réduction du temps d'organisation de 45 à 20 minutes par session (-55%)
  — Diminution du taux d'abandon des nouveaux joueurs de 78% à 30%
  — Réduction du temps d'apprentissage des mécaniques de 60 à 25 minutes (-58%)
  — Atteinte de 500 utilisateurs actifs mensuels
  — Satisfaction utilisateur ≥ 4.5/5 sur les guides
— **Atteignable :** MVP 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
  — Mesure des indicateurs : 30 juin 2026

Impact métier attendu :

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

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

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

Indicateurs de succès quantifiés :

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

Diagramme de contexte :
Flux principaux :

- **Flux nouveau joueur :** Authentification -> Guide débutant -> Recherche équipe bienveillante -> Session d'apprentissage
- **Flux joueur expérimenté :** Authentification -> Création escouade -> Planification rapide -> Gestion équipe
- **Flux données :** API Bungie -> Cache Redis -> Base PostgreSQL -> Frontend React
- **Flux matching :** Algorithmes de matching débutants/expérimentés -> Suggestions d'équipes

1.3 Liens utiles

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

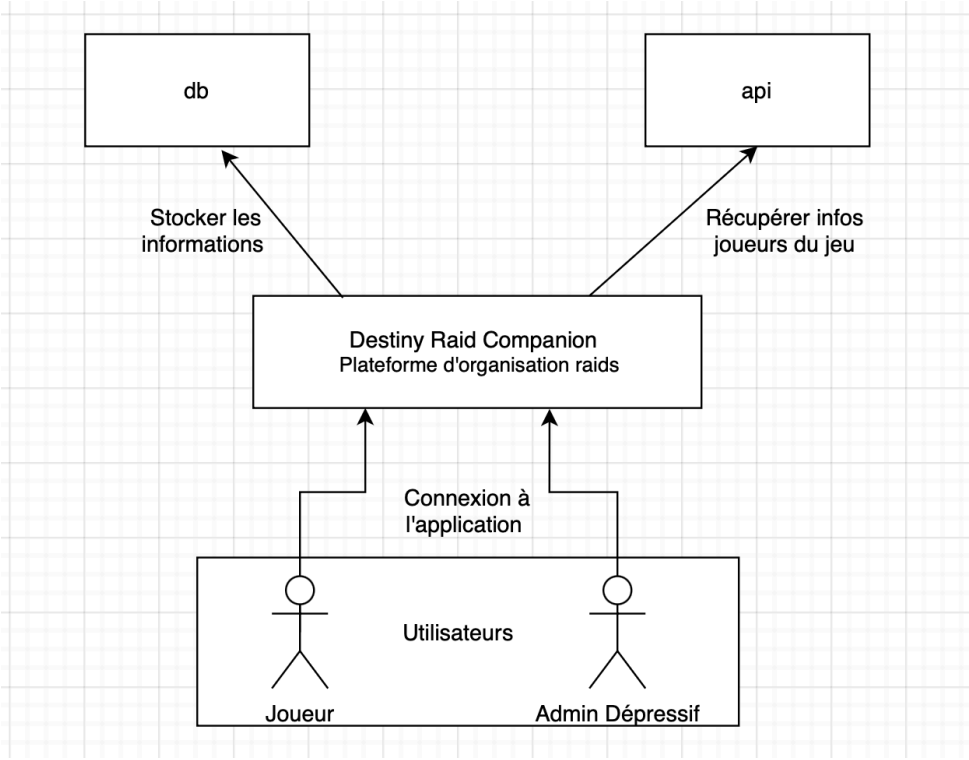


Figure 1.1 – Diagramme de contexte de la plateforme Destiny Raid Companion

Chapitre 2

Cadrage et cahier des charges

2.1 Objectifs métier, techniques et pédagogiques

Objectifs métier :

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

Objectifs techniques :

Objectif	Justification technique	Livrable
Performance : temps réponse < 2s	Amélioration UX et réduction bounce rate	Monitoring New Relic
Scalabilité : 1000 users simultanés	Support pics d'activité post-updates	Architecture microservices-ready
Disponibilité : 99% uptime	Continuité de service essentielle	Infrastructure cloud + backup
Sécurité : OAuth Bungie + chiffrement	Protection données utilisateurs RGPD	Audit de sécurité
Maintenabilité : tests > 80%	Réduction dette technique	Pipeline CI/CD + documentation

Objectifs pédagogiques :

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

Tableau MoSCoW justifié :

Priorité	Fonctionnalité	Pourquoi	Valeur
Must Have	Authentification Bungie	Sécurité + accès données utilisateur	Condition sine qua non
Must Have	Guides interactifs raids	Cœur valeur ajoutée + différentiation	Résolution problème prin
Must Have	Gestion escouades	Fonctionnalité collaborative essentielle	Rétention utilisateurs
Should Have	Calendrier collaboratif	Réduction temps organisation	Gain temps mesurable
Should Have	Profil joueur + stats	Personnalisation expérience	Engagement utilisateur
Could Have	Système de badges	Gamification	Augmentation rétention
Could Have	Chat temps réel	Communication équipe	Plus-value collaborative
Won't Have	App mobile native	Coût développement trop élevé MVP	Report version 2.0

Périmètre MVP - Milestone GitHub :

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

— **Épics principales :**

  — Auth-Bungie-OAuth

  — Guides-Interactifs

  — Gestion-Escouades

  — Calendrier-Base

— **Livrable :** Plateforme web déployée en production

Critères d'acceptation - Scénarios Gherkin :

Scénario 1 : Connexion utilisateur

```
Étant donné un utilisateur non connecté
Quand il clique sur "Se connecter avec Bungie"
Et il valide l'autorisation OAuth
Alors il est redirigé vers son tableau de bord
Et son profil est synchronisé avec l'API Bungie
```

Scénario 2 : Consultation guide raid

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

Scénario 3 : Création escouade

```
Étant donné un leader d'escouade
Quand il crée une nouvelle escouade "Raiders du Dimanche"
Et il invite 5 joueurs par leurs pseudos
Alors les invitations sont envoyées
Et l'escouade apparaît dans le calendrier
```

2.2 Cibles et parties prenantes

Matrice des risques et mitigation :

Risque	Impact	Probabilité	Mitigation	Plan de secours
Évolution API Bungie	Élevé	Moyenne	Monitoring changements	Adaptation rapide du connecteur
Faible adoption	Élevé	Moyenne	Marketing communautaire	Pivot fonctionnalités
Problèmes performance	Moyen	Élevée	Tests de charge early	Optimisation progressive
Données corrompues	Élevé	Faible	Sauvegardes automatiques	Restauration depuis backup

Personae détaillés :

Thomas - Le Débutant Motivé (25 ans)

- **Profil :** Nouveau joueur, 2 mois d'expérience Destiny 2
- **Motivation :** Voir le contenu endgame, progresser dans le jeu
- **Frustrations :**
  - "Je ne comprends pas les mécaniques complexes"
  - "Personne ne veut jouer avec moi car je suis débutant"
  - "Je perds 1h à chercher des infos sur 4 sites différents"
- **Besoins :** Guides clairs, équipe patiente, apprentissage progressif
- **Objectifs :** Compléter son premier raid dans les 2 semaines

Sarah - La Leader Expérimentée (30 ans)

- **Profil :** Joueuse vétéran, 2000+ heures, leader de clan
- **Motivation :** Optimiser l'organisation, partager son expertise
- **Frustrations :**
  - "Je passe 30min à organiser chaque session"
  - "Je dois tout réexpliquer aux nouveaux"
  - "Les outils sont dispersés (Discord, Google Calendar...)"
- **Besoins :** Centralisation, gain de temps, gestion d'équipe
- **Objectifs :** Réduire le temps d'organisation de 50%

Alex - Le Stratège (35 ans)

- **Profil :** Créateur de contenu, théoricien, min-maxer
- **Motivation :** Optimisation parfaite, données précises
- **Frustrations :**
  - "Les builds ne sont pas à jour avec les patches"
  - "Pas de données consolidées sur les stratégies"
- **Besoins :** Analytics, données fiables, communauté active
- **Objectifs :** Créer des guides optimisés basés sur les données

Matrice d'influence des parties prenantes :

Partie prenante	Influence	Intérêt	Stratégie
Utilisateurs finaux	Élevée	Très élevé	Validation continue, feedback régulier
Développeur (moi)	Très élevée	Très élevé	Autonomie totale, prise de décision
Communauté Destiny 2	Moyenne	Élevé	Implication early, beta testing
Bungie (API)	Élevée	Faible	Conformité aux CGU, monitoring changements
Testeurs beta	Faible	Élevé	Recrutement actif, reconnaissance

User Stories formatées :

US1 : Consultation guide débutant

```
En tant que Thomas (débutant)
Je veux consulter un guide raid étape par étape
Afin de comprendre les mécaniques sans être submergé

Critères d'acceptation:
- Given un débutant sur la page guide
```

```
- When il sélectionne une étape
- Then l'explication s'affiche avec illustration
- And les termes techniques sont définis
- And le temps estimé est visible
```

US2 : Organisation session

```
En tant que Sarah (leader expérimentée)
Je veux planifier une session raid en 3 clics
Afin de gagner du temps sur l'organisation

Critères d'acceptation:
- Given une leader connectée
- When elle crée une nouvelle session
- Then le calendrier propose des créneaux
- And les membres disponibles sont notifiés
- And la session apparaît dans l'agenda
```

Plan de validation utilisateur :

— **Session test MVP :** 5 utilisateurs recrutés dans la communauté
— **Date :** 1er avril 2026 (2 semaines avant release)
— **Méthodologie :** Tests utilisabilité + questionnaires satisfaction
— **Métriques :** Tâches accomplies, temps completion, score SUS
— **Livrable :** Rapport de tests avec recommandations

2.3 Exigences fonctionnelles

Spécification fonctionnelle détaillée :

Fonctionnalités Front Office :

Fonctionnalité	Description	Priorité
Authentification	OAuth Bungie, gestion sessions, profil sync	Must Have
Guides interactifs	Étapes détaillées, illustrations, termes définis	Must Have
Gestion escouades	Création, invitation, rôles, calendrier	Must Have
Recherche joueurs	Filtres par niveau, disponibilité, langues	Should Have
Profil personnel	Stats, badges, historique, équipement	Should Have
Calendrier	Sessions planifiées, rappels, disponibilités	Should Have

Fonctionnalités Back Office :

Fonctionnalité	Description	Priorité
Admin utilisateurs	Modération, stats d'usage, support	Must Have
Gestion contenu	CRUD guides, illustrations, mises à jour	Must Have
Analytics	Métriques engagement, performance, erreurs	Should Have
Logs système	Monitoring API, erreurs, performances	Should Have
Backup auto	Sauvegardes base de données, restore	Must Have

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

Permission	Anonyme	Joueur	Leader	Modo	Admin
Voir guides publics	✓	✓	✓	✓	✓
Connexion Bungie	✗	✓	✓	✓	✓
Crée escouade	✗	✓	✓	✓	✓
Modifier guides	✗	✗	✗	✓	✓
Admin utilisateurs	✗	✗	✗	✗	✓
Accès analytics	✗	✗	✗	✓	✓

Exigences de confidentialité RGPD :

Aspect	Mesures de conformité
Données collectées	Pseudonyme Bungie, stats jeu, préférences, logs connexion
Stockage	Chiffrement AES-256, base PostgreSQL sécurisée
Durée conservation	3 ans après dernière connexion (durée légale)
Droits utilisateurs	Accès, rectification, suppression, portabilité via interface
Sécurité	HTTPS obligatoire, tokens JWT expiration 24h, audit logs
Sous-traitants	Hébergeur cloud (Scaleway) avec certification ISO 27001

Processus d'authentification sécurisé :

- **Flux nominal :** Redirection OAuth Bungie → Callback → JWT generation
- **Erreurs :**
  - API Bungie indisponible : Message d'erreur + réessai automatique
  - Token expiré : Refresh automatique ou reconnexion
  - Compte non autorisé : Message explicite pour nouveaux joueurs
- **Sécurité :**
  - Verrouillage après 5 tentatives échouées (30min)
  - Session expire après 24h d'inactivité
  - Logout global sur tous devices

Revue d'exigences avec utilisateurs :

- **Date :** 15 février 2026
- **Participants :** 3 joueurs test (débutant, expérimenté, leader)
- **Méthode :** Atelier de conception collaborative (design studio)
- **Livrable :** PV de revue avec modifications validées
- **Validation :** Sign-off des spécifications fonctionnelles

2.4 Exigences et choix techniques

Justification des choix techniques - DECISIONS.md :
Choix : React/Node.js/PostgreSQL

- **Alternatives considérées :**
  - Vue.js/Laravel/MySQL (écosystème PHP mature)
  - Angular/.NET/SQL Server (stack enterprise)

— **Raison du choix :**
  — Stack JavaScript unifiée (réutilisation compétences)
  — Communauté active et nombreuses librairies
  — Courbe d'apprentissage adaptée au délai
— **Options rejetées :**
  — MongoDB pour données utilisateurs (manque intégrité relationnelle)
  — Firebase (vendor lock-in, coût à l'usage)

Architecture 3-tiers avec flux UML :

```
+------------+      +-------------+      +---------------+
|   CLIENT   | <--> |   SERVER    | <--> |   DATA        |
|   React    |      |   Node.js   |      |   PostgreSQL  |
|   Tier     | HTTP |   Tier      | SQL  |   Tier        |
+------------+      +-------------+      +---------------+
↑                    ↑                    ↑
|                    |                    |
UI Events           Business Logic       Data Persistence
State Mgmt          API Routes           Transactions
Components          Authentication       Relations
```

Légende des flux :

— **Flux données :** Client → Server → Database → Server → Client
— **Flux authentification :** OAuth Bungie → Token → Session
— **Flux cache :** API Bungie → Redis → Client (performance)

Schéma de données PostgreSQL (extrait) :

```sql
-- Table utilisateurs
CREATE TABLE users (
id SERIAL PRIMARY KEY,
bungie_id VARCHAR(100) UNIQUE NOT NULL,
display_name VARCHAR(50) NOT NULL,
created_at TIMESTAMPTZ DEFAULT NOW(),
last_login TIMESTAMPTZ,
role user_role DEFAULT 'player'
);

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

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

Stratégie de scalabilité :

— **Short-term :** Cache Redis pour API Bungie, indexation DB
— **Medium-term :** Read replicas PostgreSQL, CDN pour assets

— **Long-term :** Microservices pour fonctionnalités indépendantes
— **Monitoring :** Métriques performance (New Relic), alerting seuils

Comparaison stacks techniques :

Critère	React/Node.js	Vue.js/Laravel	Angular/.NET
Simplicité	★★★★★	★★★★☆	★★★☆☆
Performance	★★★★☆	★★★★☆	★★★★★
Écosystème	★★★★★	★★★★☆	★★★★☆
Courbe apprentissage	★★★★☆	★★★☆☆	★★☆☆☆
Décision	**CHOISI**	Alternative	Trop complexe

2.5 Définition du MVP

Périmètre MVP - GitHub Milestones :

— **Milestone : MVP-v1.0** - Due : March 15, 2026
— **Épics incluses :**
  — **AUTH :** OAuth Bungie, gestion sessions
  — **GUIDES :** 3 raids détaillés (Vault, Last Wish, Deep Stone)
  — **SQUADS :** Création, invitation, gestion basique
  — **PLANNING :** Calendrier simple, créneaux
— **Épics exclues :**
  — Chat temps réel
  — Système de badges
  — Analytics avancées
  — App mobile

Parcours utilisateurs complets MVP :

Parcours 1 : Premier raid réussi

1. Thomas arrive sur la plateforme (landing page)
2. Se connecte avec son compte Bungie (OAuth)
3. Consulte le guide ”Vault of Glass pour débutants”
4. Rejoint une escouade ”Bienveillante débutants”
5. Participe à sa première session (2h)
6. Donne son feedback sur l’expérience

Parcours 2 : Organisation optimisée

1. Sarah se connecte (session existante)
2. Crée une escouade ”Raiders Expérimentés”
3. Planifie une session pour samedi 20h
4. Invite 5 coéquipiers par leurs pseudos
5. La session apparaît dans tous les agendas
6. Rappel automatique 1h avant

Plan de test de validation MVP :

— **Date :** 1-7 mai 2026
— **Participants :** 8 utilisateurs (3 débutants, 3 expérimentés, 2 leaders)
— **Scénarios testés :** 5 parcours utilisateurs critiques
— **Métriques :** Taux de succès, temps completion, satisfaction
— **Livrable :** Rapport validation avec go/no-go release

KPI par fonctionnalité MVP :

Fonctionnalité	Indicateur	Cible
Authentification	Taux de succès connexion	> 95%
Guides interactifs	Temps moyen consultation	< 25 minutes
Gestion escouades	Nombre escouades créées	100 premier mois
Calendrier	Temps planification session	< 10 minutes

2.6 Roadmap

Roadmap GitHub détaillée : Milestone : POC - 5 février 2025

- Authentification Bungie fonctionnelle
- Mockup guides interactifs
- Schéma base de données validé
- Architecture technique finalisée

Milestone : MVP - 15 mars 2026

- Authentification OAuth Bungie opérationnelle
- Guides interactifs pour 3 raids principaux
- Gestion basique des escouades
- Calendrier collaboratif simple
- Interface responsive web
- Déploiement environnement de test

Milestone : Version 1.0 - 15 mai 2026

- **Fonctionnalités complètes :**
  - Système de badges et gamification
  - Profils joueurs détaillés avec statistiques
  - Recherche avancée joueurs/escouades
  - Notifications et rappels automatiques
  - Analytics de base (tableau de bord admin)
- **Performance et qualité :**
  - Tests automatisés (couverture > 80%)
  - Temps de réponse < 2 secondes
  - Documentation utilisateur complète
  - Sécurité renforcée (audit de sécurité)
- **Déploiement production :**
  - Plateforme déployée en production
  - Monitoring et alerting configurés
  - Sauvegardes automatiques
  - CI/CD entièrement automatisé
- **Objectif utilisateurs :** 100 utilisateurs actifs

Milestone : Version 1.1 - 15 juillet 2026

- Chat en temps réel intégré
- Système de recommandations d'équipements
- Amélioration de l'UX/UI basée sur les retours
- Optimisation des performances

— Support multilingue (anglais/français)
Milestone : Version 1.2 - 15 septembre 2026
— Analytics avancées et rapports détaillés
— Intégration avec Discord webhooks
— Système de clans étendu
— Guides pour tous les raids disponibles
— 300 utilisateurs actifs
Milestone : Version 2.0 - 15 janvier 2027
— Application mobile React Native
— API publique pour développeurs
— Système de streaming intégré
— Fonctionnalités sociales avancées
— 500+ utilisateurs actifs
Indicateurs de progression :
— **Code :** Couverture tests > 80%, dette technique < 5%
— **Utilisateurs :** Croissance mensuelle > 20%, rétention > 60%
— **Performance :** Temps réponse < 2s, disponibilité > 99%
— **Métier :** Réduction temps organisation < 20 minutes
— **Satisfaction :** Note moyenne ≥ 4.5/5 sur les guides

2.7 Liens utiles

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

Chapitre 3

Méthodologie et organisation

3.1 Gestion de projet avec GitHub

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

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

3.1.1 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

Application des principes Agile au quotidien :

— **Itérations courtes :** Sprints de 2 semaines pour maintenir le rythme

— **Adaptation permanente :** Revue des priorités à chaque sprint planning

— **Livraison continue :** Déploiement automatique sur environnement de test

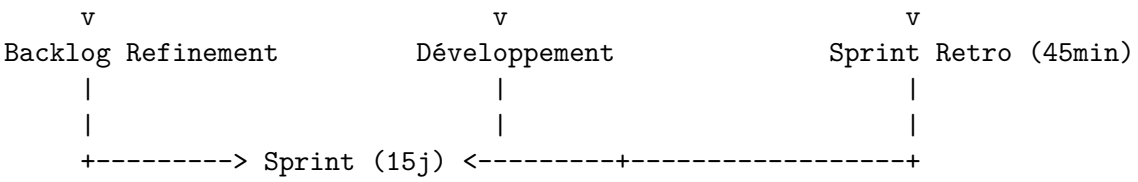
— **Simplicité :** Focus sur les fonctionnalités à plus haute valeur ajoutée

3.1.2 Rituels Agile et leur mise en œuvre

Rituel	Fréquence	Objectif et mise en œuvre
Daily Standup	Quotidien (10 min)	Point sur avancement, blocages, objectifs journée. Mise à jour GitHub Projects
Sprint Planning	Tous les 15 jours	Sélection des issues du backlog, estimation, définition objectifs sprint
Sprint Review	Fin de sprint	Démonstration fonctionnalités, validation critères d'acceptation, recueil retours
Sprint Retrospective	Fin de sprint	Amélioration processus, identification points à optimiser, actions correctives

Schéma des rituels Scrum :

```
Sprint Planning (2h) ---> Daily Standup (10min) ---> Sprint Review (1h)
     |                          |                          |
     |                          |                          |
```



Métriques de suivi :

- **Vélocité :** 8-12 story points par sprint
- **Taux de complétion :** > 85% des tâches par sprint
- **Bugs ouverts/fermés :** Ratio < 0.5 (2 bugs fermés pour 1 ouvert)
- **Lead time :** < 5 jours pour les issues critiques

Colonnes du tableau Kanban :

- **Backlog :** Fonctionnalités à développer (triées par priorité)
- **Sprint Backlog :** Tâches sélectionnées pour le sprint courant
- **To Do :** Tâches prêtes pour le développement
- **In Progress :** Tâches en cours (WIP limit : 2)
- **Review :** Code en attente de validation (tests, revue)
- **Done :** Fonctionnalités livrées et validées

3.1.3 User Stories et estimation de temps

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

Vos user stories avec estimations :

User Story	***Critères d'acceptation***	***Story Points***	***Milestone***
En tant que joueur, je veux m'authentifier avec mon compte Bungie	*Connexion OAuth fonctionnelle, récupération du profil, gestion des sessions JWT*	*5*	*MVP v1.0*
En tant que joueur, je veux consulter des guides interactifs de raids	*Affichage des étapes détaillées, illustrations, stratégies par rôle*	*8*	*MVP v1.0*
En tant que joueur, je veux créer et gérer une escouade	*Création d'escouade, invitation de membres, gestion des rôles*	*5*	*MVP v1.0*
En tant que joueur, je veux planifier une session de raid	*Calendrier interactif, création d'événement, notifications*	*8*	*v1.1*
En tant que joueur, je veux voir mes statistiques et badges	*Profil personnel, historique des raids, système de gamification*	*5*	*v1.1*

En tant qu'administrateur, je veux gérer le contenu des guides	*Interface d'administration, édition des guides, validation*	*8*	*v1.2*

Système d'estimation :
- **1 point :** Tâche simple (< 1 jour)
- **3 points :** Tâche moyenne (1-2 jours)
- **5 points :** Tâche complexe (3-4 jours)
- **8 points :** Tâche très complexe (> 5 jours)

3.2 Versioning GitHub et conventions

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

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

3.2.1 CONTRIBUTING.md et normalisation

Contenu du CONTRIBUTING.md :
- **Environnement :** Setup du projet, pré-requis, installation
- **Conventions de code :** ESLint, Prettier, standards React/Node.js
- **Workflow Git :** Processus de création de branches, commits, PR
- **Testing :** Comment exécuter les tests, couverture attendue
- **Code Review :** Checklist pour la revue de code

3.2.2 Conventions de branches

Type de branche	Convention de nommage
Feature	`feature/nom-fonctionnalite` ou `feature/issue-#123`
Bugfix	`fix/description-bug` ou `fix/issue-#456`
Hotfix	`hotfix/description-urgente`
Release	`release/v1.0.0`
Documentation	`docs/sujet-documentation`

3.2.3 Conventions de commits

Schéma Git Flow :

```
main (protected) ------------------------------
  |                      |          |
  |                      |          +-- hotfix/critical-issue
  |                      |
  +-- develop (protected) ----------------------
       |                     |          |
       |                     |          +-- release/v1.0.0
       |                     |
       +-- feature/user-auth
       +-- feature/raid-guides
       +-- fix/login-validation
```

Conventions de commit (Conventional Commits) :

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

Template de Pull Request :

```
## Description
[Description des changements apportés]

## Issue liée
[Lien vers l'issue GitHub]

## Type de changement
- [ ] Correction de bug
- [ ] Nouvelle fonctionnalité
- [ ] Modification de configuration
- [ ] Documentation

## Checklist
- [ ] Mon code suit les conventions du projet
- [ ] J'ai ajouté des tests
- [ ] Les tests passent localement
- [ ] J'ai mis à jour la documentation
```

3.3 Planification et outils de suivi

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

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

3.3.1 GitHub Project et Roadmap

Structure du GitHub Project :

— **Vue Kanban :** Suivi visuel de l'état des tâches
— **Filtres :** Par label, milestone, assigné, statut
— **Automatisations :** Changement de statut basé sur les PR/issues
— **Vues personnalisées :** Tableau de bord pour daily standup

Roadmap GitHub :

— **Visibilité :** Roadmap publique pour transparence
— **Milestones :** Dates cibles pour chaque version
— **Dépendances :** Liens entre les fonctionnalités
— **Suivi progression :** Avancement visuel par milestone

Extrait de roadmap GitHub :

```
Phase 1: MVP (Oct 2025 - Mars 2026)
+-- Sprint 1: Setup & Auth (4 semaines)
```

```
    +-- Environnement dev (1 semaine) [Issue #1]
    +-- Authentification Bungie (2 semaines) [Issue #2]
    +-- Base de données (1 semaine) [Issue #3]

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

Phase 2: Version 1.0 (Avril - Mai 2026)
+-- 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

— **Policies :** PR obligatoire pour merge en develop

— **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

— **Tests :** Issues liées pour les scénarios de test

— **Milestones :** Regroupement logique par version

— **Labels :** Complexité (S, M, L, XL), type (bug, feature, docs)

Workflow de validation :

1. Issue créée avec critères d'acceptation
2. Branche feature développée avec tests associés
3. Pull Request avec validation des tests automatisés
4. Revue de code et validation manuelle si nécessaire
5. Merge et déploiement automatique en environnement de test

Focus GitHub

Git Flow et conventions :

— **Branches :** main, develop, feature/*, release/*, hotfix/*

— **PR Template :** Description, tests, checklist validation

— **CODEOWNERS :** Validation automatique des règles

— **Protection Rules :** Pas de push direct sur main/develop, status checks requis

GitHub Projects Kanban :

— **Colonnes :** Backlog, Sprint Planning, In Progress, Review, Done

— **WIP Limits :** 2 features max en développement simultané

— **Automation :** Mise à jour statut via labels et milestones

— **Metrics :** Cycle time, lead time, throughput, burndown chart

Roadmap et milestones :

— **Milestones :** MVP (15 mars 2026), v1.0 (15 mai 2026), v1.1 (15 juillet 2026), v1.2 (15 septembre 2026), v2.0 (15 janvier 2027)

— **Dependencies :** Backend → Frontend → Tests → Déploiement

— **Risks :** Complexité API Bungie, performance en charge

— **Success Metrics :** Velocity stable, qualité code, satisfaction utilisateur

3.4 Estimation de temps et planification

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

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

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

Estimation détaillée :

Fonctionnalité	***Phase***	***Story Points***	***Jours estimés***	***Milestone***
Environnement de développement	*Setup*	*3*	*3*	*MVP*
Authentification Bungie OAuth	*Backend*	*5*	*5*	*MVP*
Base de données PostgreSQL	*Backend*	*3*	*3*	*MVP*
API Gestion escouades	*Backend*	*5*	*5*	*MVP*
Guides interactifs raids	*Frontend*	*8*	*8*	*MVP*
Calendrier raids	*Frontend*	*5*	*5*	*v1.0*
Profils joueurs	*Frontend*	*3*	*3*	*v1.0*
Intégration API Destiny 2	*Intégration*	*8*	*8*	*v1.0*
Système de badges	*Fonctionnalité*	*5*	*5*	*v1.0*
Tests unitaires et intégration	*Qualité*	*8*	*8*	*v1.0*
Tests E2E	*Qualité*	*5*	*5*	*v1.0*
Dockerisation	*Déploiement*	*3*	*3*	*v1.0*
CI/CD	*Déploiement*	*5*	*5*	*v1.0*
Documentation technique	*Livraison*	*3*	*3*	*v1.0*
Total		***70***	***70 jours***	
Avec marge 20%		***84***	***84 jours***	

3.4.1 Métriques de suivi et amélioration continue

Métriques collectées :

- **Vélocité :** Nombre de story points complétés par sprint
- **Burndown chart :** Progression vers l'objectif du sprint
- **Lead time :** Temps entre création et fermeture d'une issue
- **Cycle time :** Temps de développement effectif
- **Taux de bugs :** Nombre de bugs rapportés vs fonctionnalités livrées

Amélioration continue :

— **Rétrospectives :** Identification points d'amélioration processus
— **Ajustements :** Adaptation planning basée sur la vélocité réelle
— **Qualité code :** Suivi couverture tests, dette technique
— **Satisfaction :** Retours utilisateurs sur fonctionnalités livrées

3.5 Liens utiles

— GitHub Project : https://github.com/xxx/projects/1
— CONTRIBUTING.md : https://github.com/xxx/CONTRIBUTING.md
— GitHub Flow/PRs : https://docs.github.com/pull-requests
— Git Flow : https://bit.ly/gitflow-atlassian
— GitHub Projects : https://bit.ly/github-projects
— GitHub Roadmap : https://bit.ly/github-roadmap
— GitHub Milestones : https://bit.ly/github-milestones
— User Stories : https://www.mountaingoatsoftware.com/agile/user-stories
— Estimation de temps : https://bit.ly/time-estimation
— Conventional Commits : https://www.conventionalcommits.org

Chapitre 4

Conception fonctionnelle et technique

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

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

Votre approche de conception : *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.

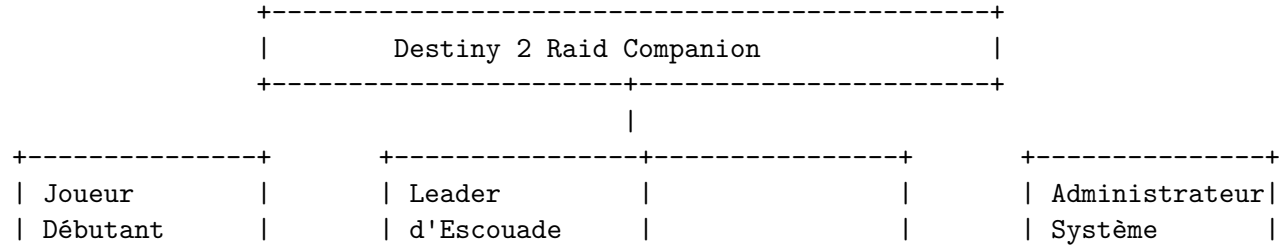
Acteurs principaux identifiés :

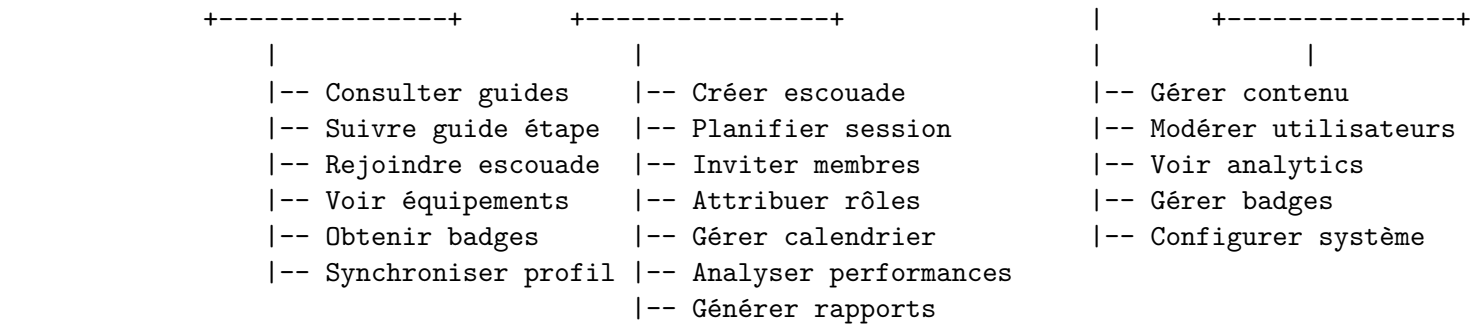
- **Joueur Débutant :** Nouvel utilisateur cherchant à comprendre les mécaniques de raid
- **Joueur Expérimenté :** Utilisateur régulier optimisant son gameplay
- **Leader d'Escouade :** Responsable de la coordination d'équipe
- **Administrateur :** Gestionnaire de contenu et modérateur
- **Système Bungie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

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

— **Flux principal :**

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

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

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

UC004 - Planification de session :

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

— **Flux principal :**

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

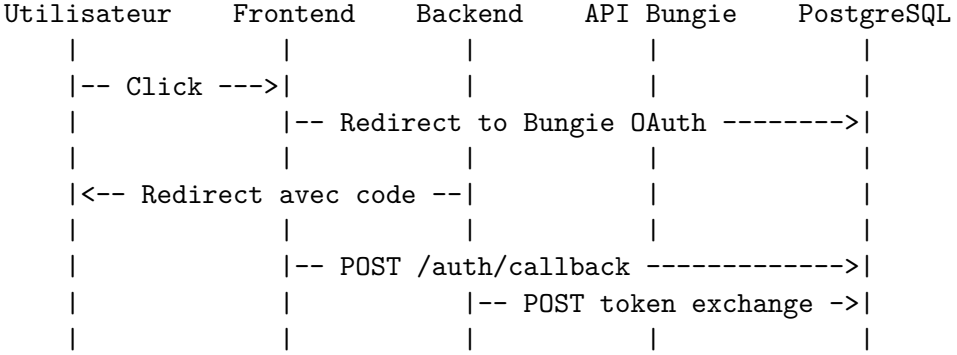
— **Flux alternatif :** Conflit de planning Suggestion de créneaux alternatifs

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

4.2 Diagrammes de séquence

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

Diagramme de séquence - Authentification OAuth :



```
    |             |           |<-- Access Token --------|
    |             |           |-- GET user profile ---->|
    |             |           |           |             |
    |             |           |<-- User Data ---------- |
    |             |           |-- UPSERT user --------> |
    |             |           |           |             |
    |             |<-- JWT Token + User Data ---------- |
    |<-- Logged In -----------|           |             |
```

Diagramme de séquence - Consultation guide avec cache :

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

Gestion d'erreurs détaillée :

— **Timeout API Bungie :** Retry automatique (3 tentatives) + Fallback cache

— **Données corrompues :** Validation schema JSON + Logging erreur

— **Utilisateur non autorisé :** Redirection login + Message contextuel

— **Rate limiting :** Backoff exponentiel + Queue de requêtes

4.3 Conception de l'interface graphique

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

4.3.1 Architecture des composants React

Structure modulaire des composants :

```
src/
  components/
     common/
        Header/
        Navigation/
        LoadingSpinner/
        ErrorBoundary/
     guides/
        GuideList/
        GuideViewer/
        StepNavigation/
        EquipmentRecommendations/
     squads/
```

```
        SquadManager/
        MemberList/
        InvitationSystem/
        RoleManagement/
    calendar/
        CalendarView/
        SessionCreator/
        AvailabilityChecker/
        NotificationCenter/
    profile/
        UserProfile/
        BadgeCollection/
        StatisticsDashboard/
        SettingsPanel/
hooks/
    useAuth.js
    useSquads.js
    useGuides.js
    useCalendar.js
services/
    api.js
    cache.js
    websocket.js
```

Spécifications des composants critiques :
GuideViewer Component :

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

SquadManager Component :

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

4.3.2 Charte graphique détaillée

Système de design complet :
Palette de couleurs :

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

Typographie :

— **Principale :** Inter (weights : 300, 400, 500, 600, 700)
— **Hiérarchie :**

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

Espacement (8px grid system) :

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

Composants UI standardisés :

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

4.3.3 Maquettes et prototypes

Workflow de conception :

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

Pages principales conçues :

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

4.4 Conception de base de données

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

4.4.1 Modèle Conceptuel de Données (MCD)

Entités principales et relations :

Entité USER :

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

Entité SQUAD :

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

Entité RAID :

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

Diagramme MCD complet :

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

4.4.2 Modèle Logique de Données (MLD)

Schéma relationnel normalisé :
Table USERS :

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

Table SQUADS :

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

Table SESSIONS :

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

4.4.3 Modèle Physique de Données (MPD)

Optimisations performances :
Index stratégiques :

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

-- Index pour gestion escouades
CREATE INDEX idx_squads_leader_id ON squads(leader_id);
CREATE INDEX idx_squads_created_at ON squads(created_at DESC);
CREATE INDEX idx_squads_is_public ON squads(is_public) WHERE is_public =
    true;

-- Index pour planning sessions
CREATE INDEX idx_sessions_squad_id ON sessions(squad_id);
CREATE INDEX idx_sessions_scheduled_at ON sessions(scheduled_at);
CREATE INDEX idx_sessions_status ON sessions(status);
CREATE INDEX idx_sessions_squad_scheduled ON sessions(squad_id,
    scheduled_at);

-- Index pour recherches full-text
CREATE INDEX idx_guides_title ON guides USING gin(to_tsvector('english',
    title));
CREATE INDEX idx_raids_name ON raids USING gin(to_tsvector('english', name)
    );
```

Stratégie de partitionnement :

```sql
-- Partitionnement des logs d'activité par mois
CREATE TABLE activity_logs_2025_01 PARTITION OF activity_logs
    FOR VALUES FROM ('2025-01-01') TO ('2025-02-01');

CREATE TABLE activity_logs_2025_02 PARTITION OF activity_logs
    FOR VALUES FROM ('2025-02-01') TO ('2025-03-01');
```

Architecture multi-base :

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

4.5 Architecture technique détaillée

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

4.5.1 Couche Présentation (Frontend)

Stack technique complète :

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

Structure des services frontend :
Service d'authentification :

```
class AuthService {
  async loginWithBungie() {
    const authUrl = this.buildBungieAuthUrl();
    window.location.href = authUrl;
  }

  async handleOAuthCallback(code) {
    const response = await api.post('/auth/callback', { code });
    this.storeTokens(response.data);
    return this.getUserProfile();
  }

  async refreshToken() {
    const refreshToken = this.getRefreshToken();
    const response = await api.post('/auth/refresh', { refreshToken });
    this.storeTokens(response.data);
  }

  isTokenExpired() {
    const expiresAt = localStorage.getItem('token_expires_at');
    return Date.now() >= parseInt(expiresAt);
  }
}
```

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 :

```
class SquadService {
  async createSquad(squadData, leaderId) {
    // Validation des données
    const validation = squadSchema.safeParse(squadData);
    if (!validation.success) {
      throw new ValidationError(validation.error);
    }

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

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

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

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

      return squad;
    });
  }
}
```

Middleware d'authentification :

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

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

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

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

```
    }

    req.user = user;
    next();
  } catch (error) {
    if (error.name === 'TokenExpiredError') {
      return res.status(401).json({ error: 'Token expiré' });
    }
    return res.status(403).json({ error: 'Token invalide' });
  }
};
```

4.5.3 Couche Données

Configuration PostgreSQL :

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

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

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

Stratégie Redis :

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

4.6 Stratégie de tests

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

4.6.1 Couverture de tests

Objectifs de couverture :

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

Structure des tests :

Tests unitaires services :

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

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

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

      expect(result).toHaveProperty('id', 1);
      expect(result.name).toBe('Test Squad');
      expect(userRepository.getUserSquadCount).toHaveBeenCalledWith(1);
    });

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

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

Tests d'intégration API :

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

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

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

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

4.6.2 Automatisation des tests

Pipeline de tests GitHub Actions :

```
name: Test Pipeline
on: [push, pull_request]
jobs:
  unit-tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
```

```
        with: { node-version: '18' }
      - run: npm ci
      - run: npm run test:unit
      - uses: codecov/codecov-action@v3

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

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

4.7 Plan de déploiement et infrastructure

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

4.7.1 Architecture de déploiement

Environnements multiples :

— **Development :** Docker Compose local

— **Staging :** Vercel (Frontend) + Heroku (Backend)

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

Configuration Docker :

Dockerfile Backend :

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

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

# Copie du code
COPY . .

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

# Exposition du port
```

```
EXPOSE 4000

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

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

docker-compose.yml :

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

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

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

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

volumes:
  postgres_data:
```

4.7.2 CI/CD et monitoring

Pipeline de déploiement :

```
Pipeline GitHub Actions:
1. Code Quality Checks (ESLint, Prettier)
2. Security Scanning (Snyk, npm audit)
3. Unit Tests + Coverage Report
4. Build Application
5. Integration Tests
6. Docker Image Build + Security Scan
7. Push to Container Registry
8. Deploy to Staging
9. E2E Tests on Staging
10. Manual Approval for Production
11. Blue-Green Deployment to Production
12. Post-Deployment Smoke Tests
13. Monitoring + Alerting Setup
```

Configuration de monitoring :

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

Plan de reprise d'activité :

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

4.8 Liens utiles

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

Chapitre 5

Architecture 3 tiers

5.1 Architecture 3 tiers

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

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

5.1.1 Couche Présentation (Frontend)

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

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

Exemple

Technologies de présentation :

— **Framework :** React 18 avec hooks et context

— **État :** Redux Toolkit pour la gestion d'état globale

— **Routing :** React Router pour la navigation

— **UI :** Material-UI pour les composants

— **HTTP :** Axios pour les appels API

Structure des composants :

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

5.1.2 Couche Logique Métier (Backend)

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

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

Controller

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

Exemple

Exemple de contrôleur :

```
// ProjectController.js
class ProjectController {
  async createProject(req, res) {
    try {
      const projectData = req.body;
      const project = await this.projectService.create(projectData);
      res.status(201).json(project);
    } catch (error) {
      res.status(400).json({ error: error.message });
    }
  }
}
```

Service

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

Exemple

Exemple de service :

```
// ProjectService.js
class ProjectService {
  async create(projectData) {
    // Validation des données
    this.validateProjectData(projectData);

    // Logique métier
    const project = await this.projectRepository.create(projectData);

    // Notifications
    await this.notificationService.notifyTeam(project);

    return project;
  }
}
```

Repository (DAO)

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

Exemple

Exemple de repository :

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

5.1.3 Couche Données (Database)

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

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

Exemple

Architecture des données :

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

Exemple de repository PostgreSQL :

```
class ProjectRepository {
  async create(projectData) {
    return await prisma.project.create({
      data: {
        name: projectData.name,
        description: projectData.description,
        userId: projectData.userId
      },
      include: {
        tasks: true,
        user: { select: { id: true, email: true } }
      }
    });
  }
}
```

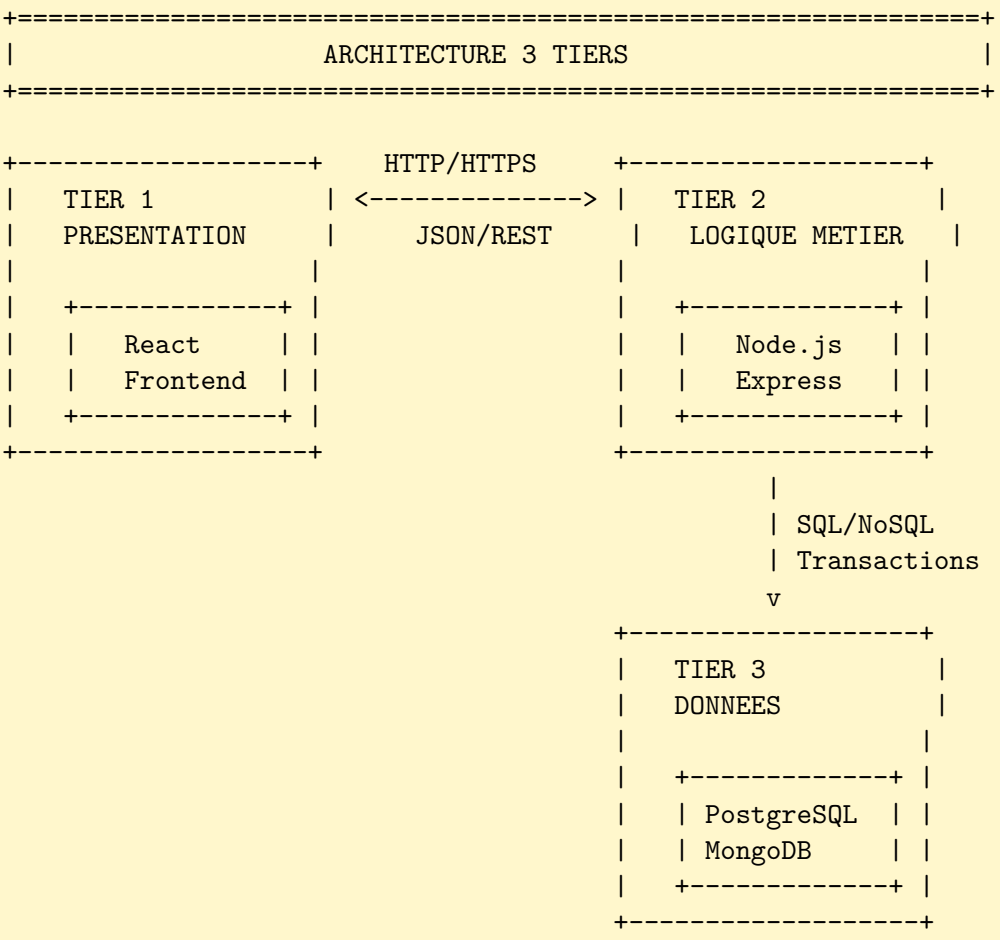
5.1.4 Communication entre les tiers

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

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

Exemple

Flux de communication 3 tiers :



5.1.5 Avantages de l'architecture 3 tiers

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

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

Exemple

Avantages de l'architecture 3 tiers :

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

5.2 Développement Frontend

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

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

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

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

Exemple

Structure des composants :

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

Exemple de composant accessible :

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

```
      role="article"
      aria-labelledby={`project-${project.id}-title`}
    >
      <h3 id={`project-${project.id}-title`}>
        {project.name}
      </h3>
      <p>{project.description}</p>
      <button
        onClick={() => onEdit(project.id)}
        aria-label={`Modifier le projet ${project.name}`}
      >
        Modifier
      </button>
    </div>
  );
};
```

Exemple

Exemple de rapport Lighthouse (1/2) :

```
{
  "categories": {
    "performance": { "score": 0.92 },
    "accessibility": { "score": 0.95 },
    "best-practices": { "score": 0.88 },
    "seo": { "score": 0.90 }
  }
}
```

Exemple

Exemple de rapport Lighthouse (2/2) :

```
  "audits": {
    "first-contentful-paint": { "score": 0.95 },
    "largest-contentful-paint": { "score": 0.88 },
    "color-contrast": { "score": 1.0 },
    "aria-allowed-attr": { "score": 1.0 }
  }
}
```

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

5.3 Développement Backend

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

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

Exemple

Structure backend :

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

Exemple de contrôleur avec validation :

```
const createProject = async (req, res, next) => {
  try {
    // Validation des données
    const { error, value } = projectSchema.validate(req.body);
    if (error) {
      return res.status(400).json({
        error: 'Données␣invalides',
        details: error.details
      });
    }

    // Appel du service métier
    const project = await projectService.createProject(value, req.user.id
       );

    // Log de l'action
    await auditService.logAction('project_created', req.user.id, {
      projectId: project.id
    });

    res.status(201).json(project);
  } catch (error) {
    next(error);
  }
};
```

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

5.4 Gestion des données

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

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

Exemple

Exemple de repository PostgreSQL :

```
class ProjectRepository {
  async create(projectData) {
    return await prisma.project.create({
      data: {
        name: projectData.name,
        description: projectData.description,
        userId: projectData.userId
      },
      include: {
        tasks: true,
        user: { select: { id: true, email: true } }
      }
    });
  }

  async findByUser(userId) {
    return await prisma.project.findMany({
      where: { userId },
      include: { tasks: true }
    });
  }
}
```

Exemple de pipeline MongoDB :

```
// Pipeline d'agrégation pour les statistiques
const getProjectStats = async (projectId, dateRange) => {
  return await activityLogs.aggregate([
    {
      $match: {
        'metadata.projectId': projectId,
        timestamp: { $gte: dateRange.start, $lte: dateRange.end }
      }
    },
    {
      $group: {
        _id: '$action',
        count: { $sum: 1 },
        uniqueUsers: { $addToSet: '$userId' }
      }
    },
    {
      $project: {
        action: '$_id',
        count: 1,
        uniqueUsersCount: { $size: '$uniqueUsers' }
      }
    }
  ]);
};
```

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

5.5 Liens utiles

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

Chapitre 6

Sécurité applicative et RGPD

6.1 Protection contre les vulnérabilités OWASP

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

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

Exemple

Middleware de sécurité Express :

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

// Configuration Helmet
app.use(helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      styleSrc: ["'self'", "'unsafe-inline'"],
      scriptSrc: ["'self'"]
    }
  }
}));

// Rate limiting
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 min
  max: 100, // 100 req/IP
  message: 'Trop de requêtes'
});
app.use('/api/', limiter);

// Validation des entrées
const validateInput = (schema) => {
  return (req, res, next) => {
    const { error } = schema.validate(req.body);
    if (error) {
      return res.status(400).json({
        error: 'Données invalides',
        details: error.details.map(d => d.message)
      });
    }
    next();
  };
};
```

Protection XSS côté frontend :

```
// React échappe automatiquement les données
const UserProfile = ({ user }) => {
  return (
    <div>
      <h1>{user.name}</h1> {/* Échappé automatiquement */}
      <p>{user.bio}</p>
      {/* Danger : éviter dangerouslySetInnerHTML */}
    </div>
  );
};

// Validation côté client avec Zod
const userSchema = z.object({
  name: z.string().min(1).max(100),
  email: z.string().email(),
  bio: z.string().max(500).optional()
});
```

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

6.2 Authentification et autorisation

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

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

Exemple

Configuration JWT et Argon2 (1/3) :

```
const jwt = require('jsonwebtoken');
const argon2 = require('argon2');

// Configuration JWT
const JWT_SECRET = process.env.JWT_SECRET;
const JWT_EXPIRES_IN = '15m';
const REFRESH_EXPIRES_IN = '7d';

// Hachage des mots de passe avec Argon2
const hashPassword = async (password) => {
  return await argon2.hash(password, {
    type: argon2.argon2id,
    memoryCost: 2 ** 16, // 64 MB
    timeCost: 3,
    parallelism: 1
  });
};
```

Exemple

Configuration JWT et Argon2 (2/3) :

```
// Génération des tokens
const generateTokens = (userId, role) => {
  const accessToken = jwt.sign(
    { userId, role, type: 'access' },
    JWT_SECRET,
    { expiresIn: JWT_EXPIRES_IN }
  );

  const refreshToken = jwt.sign(
    { userId, type: 'refresh' },
    JWT_SECRET,
    { expiresIn: REFRESH_EXPIRES_IN }
  );

  return { accessToken, refreshToken };
};
```

Exemple

Configuration JWT et Argon2 (3/3) :

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

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

  jwt.verify(token, JWT_SECRET, (err, user) => {
    if (err) {
      return res.status(403).json({
        error: 'Token␣invalide'
      });
    }
    req.user = user;
    next();
  });
};
```

Système de permissions (1/2) :

```
// Définition des permissions
const PERMISSIONS = {
  PROJECT_CREATE: 'project:create',
  PROJECT_READ: 'project:read',
  PROJECT_UPDATE: 'project:update',
  PROJECT_DELETE: 'project:delete',
  USER_MANAGE: 'user:manage'
};

// Rôles et leurs permissions
const ROLES = {
  ADMIN: [PERMISSIONS.PROJECT_CREATE, PERMISSIONS.PROJECT_READ,
          PERMISSIONS.PROJECT_UPDATE, PERMISSIONS.PROJECT_DELETE,
          PERMISSIONS.USER_MANAGE],
  MANAGER: [PERMISSIONS.PROJECT_CREATE, PERMISSIONS.PROJECT_READ,
            PERMISSIONS.PROJECT_UPDATE],
  DEVELOPER: [PERMISSIONS.PROJECT_READ, PERMISSIONS.PROJECT_UPDATE]
};
```

Système de permissions (2/2) :

```
// Middleware d'autorisation
const requirePermission = (permission) => {
  return (req, res, next) => {
    const userPermissions = ROLES[req.user.role] || [];
    if (!userPermissions.includes(permission)) {
      return res.status(403).json({
        error: 'Permissions␣insuffisantes'
      });
    }
    next();
  };
};
```

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

6.3 Conformité RGPD

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

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

Exemple

Registre des traitements :

```
// Modèle de registre des traitements
const dataProcessingRegistry = {
  'user-authentication': {
    purpose: 'Authentification et gestion des comptes utilisateurs',
    legalBasis: 'Consentement',
    dataCategories: ['identité', 'connexion'],
    retentionPeriod: '3 ans après fermeture du compte',
    recipients: ['équipe technique', 'hébergeur'],
    transfers: ['UE', 'États-Unis (clauses contractuelles)']
  },
  'project-management': {
    purpose: 'Gestion des projets et collaboration',
    legalBasis: 'Exécution du contrat',
    dataCategories: ['travail', 'communication'],
    retentionPeriod: '5 ans après fin du projet',
    recipients: ['équipe projet', 'clients'],
    transfers: ['UE uniquement']
  }
};

// API pour les droits RGPD
const gdprController = {
  // Droit d'accès
  async getPersonalData(req, res) {
    const userId = req.user.userId;
    const userData = await userService.getCompleteUserData(userId);
    res.json({
      personalData: userData,
      processingPurposes: Object.keys(dataProcessingRegistry),
      retentionPeriods: dataProcessingRegistry
    });
  },

  // Droit à l'effacement
  async deletePersonalData(req, res) {
    const userId = req.user.userId;
    await userService.anonymizeUserData(userId);
    await auditService.logAction('gdpr_deletion', userId);
    res.json({ message: 'Données personnelles supprimées' });
  },

  // Droit à la portabilité
  async exportPersonalData(req, res) {
    const userId = req.user.userId;
    const exportData = await userService.exportUserData(userId);
    res.attachment('mes-donnees.json');
    res.json(exportData);
  }
};
```

Chiffrement des données sensibles (1/3) :

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

// Configuration du chiffrement
const ENCRYPTION_KEY = process.env.ENCRYPTION_KEY;
const ALGORITHM = 'aes-256-gcm';
```

Exemple

Chiffrement des données sensibles (2/3) :

```
// Fonction de chiffrement
const encrypt = (text) => {
  const iv = crypto.randomBytes(16);
  const cipher = crypto.createCipher(ALGORITHM, ENCRYPTION_KEY);
  cipher.setAAD(Buffer.from('user-data'));

  let encrypted = cipher.update(text, 'utf8', 'hex');
  encrypted += cipher.final('hex');

  const authTag = cipher.getAuthTag();

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

Exemple

Chiffrement des données sensibles (3/3) :

```
// Fonction de déchiffrement
const decrypt = (encryptedData) => {
  const decipher = crypto.createDecipher(ALGORITHM, ENCRYPTION_KEY);
  decipher.setAAD(Buffer.from('user-data'));
  decipher.setAuthTag(Buffer.from(encryptedData.authTag, 'hex'));

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

  return decrypted;
};
```

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

6.4 Liens utiles

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

Chapitre 7

Tests et qualité logicielle

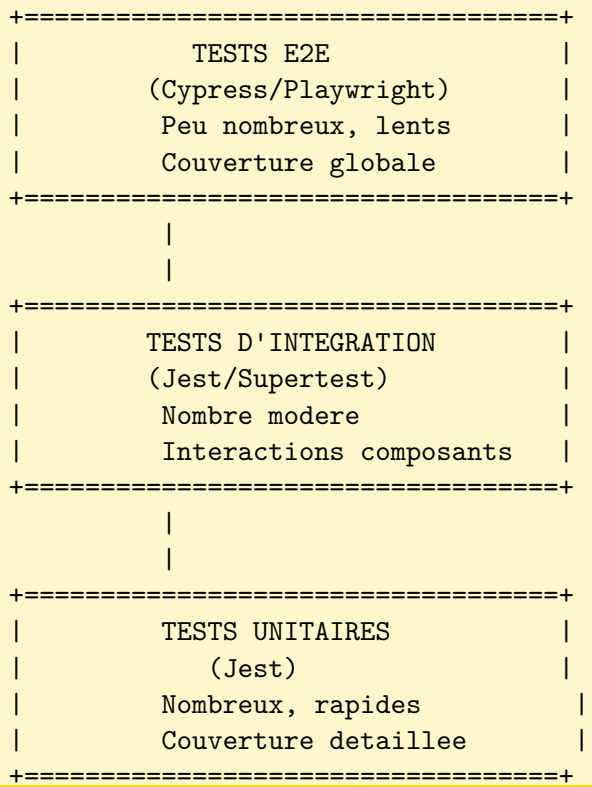
7.1 Stratégie de tests

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

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

Exemple

Pyramide de tests :



Exemple

Exemple de test unitaire (1/2) :

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

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

Exemple

Exemple de test unitaire (2/2) :

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

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

      // Assert
```

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

7.2 Tests de performance

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

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

Exemple

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

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

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

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

Exemple

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

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

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

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

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

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

  sleep(1);
}
```

Exemple

Résultats de performance :

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

7.3 Qualité du code avec SonarQube

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

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

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

Exemple

Configuration SonarQube :

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

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

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

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

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

Exemple

Rapport de qualité SonarQube :

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

Exemple de correction de code smell :

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

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

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

Focus GitHub

Intégration SonarQube dans GitHub Actions :

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

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

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

      - name: Install dependencies
        run: npm ci

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

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

Métriques de qualité GitHub :

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

Métriques Lighthouse :

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

7.4 Liens utiles

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

Chapitre 8

Déploiement et CI/CD

8.1 Containerisation avec Docker

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

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

Exemple

Dockerfile multi-stage :

```
\# Stage 1: Build
FROM node:18-alpine AS builder

WORKDIR /app

\# Copier les fichiers de dépendances
COPY package*.json ./
RUN npm ci --only=production

\# Copier le code source
COPY . .

\# Build de l'application
RUN npm run build

\# Stage 2: Production
FROM node:18-alpine AS production

\# Créer un utilisateur non-root
RUN addgroup -g 1001 -S nodejs
RUN adduser -S nextjs -u 1001

WORKDIR /app

\# Copier les dépendances de production
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/package*.json ./

\# Changer le propriétaire des fichiers
RUN chown -R nextjs:nodejs /app
USER nextjs

\# Exposer le port
EXPOSE 3000

\# Variables d'environnement
ENV NODE_ENV=production
ENV PORT=3000

\# Commande de démarrage
CMD ["node", "dist/index.js"]
```

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

```
version: '3.8'

services:
  app:
    build: .
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
      - DATABASE_URL=postgresql://user:pass@postgres:5432/projectdb
      - MONGODB_URI=mongodb://mongo:27017/projectlogs
    depends_on:
      - postgres
      - mongo
      - redis
    restart: unless-stopped

```

Exemple

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

```
  mongo:
    image: mongo:6
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=password
    volumes:
      - mongo_data:/data/db
    ports:
      - "27017:27017"
    restart: unless-stopped

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    restart: unless-stopped

volumes:
  postgres_data:
  mongo_data:
```

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

8.2 Pipeline CI/CD avec GitHub Actions

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

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

Exemple

Workflow GitHub Actions complet (1/3) :

```yaml
name: CI/CD Pipeline

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

env:
  NODE_VERSION: '18'
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

jobs:
  # Job 1: Lint et tests
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: ${{ env.NODE_VERSION }}
          cache: 'npm'

      - name: Install dependencies
        run: npm ci

      - name: Run linter
        run: npm run lint

      - name: Run type checking
        run: npm run type-check

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

      - name: Upload coverage
        uses: codecov/codecov-action@v3
        with:
          token: ${{ secrets.CODECOV_TOKEN }}
```

Exemple

Workflow GitHub Actions complet (2/3) :

```yaml
  # Job 2: Build et scan de sécurité
  build-and-scan:
    runs-on: ubuntu-latest
    needs: test
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Build Docker image
        run: docker build -t ${{ env.IMAGE_NAME }}:${{ github.sha }} .

      - name: Run Trivy security scan
        uses: aquasecurity/trivy-action@master
        with:
          image-ref: ${{ env.IMAGE_NAME }}:${{ github.sha }}
          format: 'sarif'
          output: 'trivy-results.sarif'

      - name: Upload Trivy scan results
        uses: github/codeql-action/upload-sarif@v2
        with:
          sarif_file: 'trivy-results.sarif'
```

Exemple

Workflow GitHub Actions complet (3/3) :

```yaml
  # Job 3: Déploiement staging
  deploy-staging:
    runs-on: ubuntu-latest
    needs: build-and-scan
    if: github.ref == 'refs/heads/develop'
    environment: staging
    steps:
      - name: Deploy to staging
        run: |
          echo "Deploying to staging environment"
          # Script de déploiement staging
          ./scripts/deploy.sh staging

  # Job 4: Déploiement production
  deploy-production:
    runs-on: ubuntu-latest
    needs: build-and-scan
    if: github.ref == 'refs/heads/main'
    environment: production
    steps:
      - name: Deploy to production
        run: |
          echo "Deploying to production environment"
          # Script de déploiement production
          ./scripts/deploy.sh production

      - name: Run smoke tests
        run: |
          echo "Running smoke tests"
          npm run test:smoke

      - name: Notify team
        if: always()
        uses: 8398a7/action-slack@v3
        with:
          status: ${{ job.status }}
          channel: '\#deployments'
          webhook_url: ${{ secrets.SLACK_WEBHOOK }}
```

Exemple

Script de déploiement (1/2) :

```
#!/bin/bash
# scripts/deploy.sh

set -e

ENVIRONMENT=$1
IMAGE_TAG=${2:-latest}

echo "Deploying to $ENVIRONMENT environment with tag $IMAGE_TAG"

# Mise à jour des images Docker
docker-compose -f docker-compose.$ENVIRONMENT.yml pull
```

Exemple

Script de déploiement (2/2) :

```
# Déploiement blue-green
if [ "$ENVIRONMENT" = "production" ]; then
    # Déploiement en blue-green
    docker-compose -f docker-compose.prod.yml up -d --scale app=2
    sleep 30
    docker-compose -f docker-compose.prod.yml up -d --scale app=1
else
    # Déploiement simple pour staging
    docker-compose -f docker-compose.staging.yml up -d
fi

# Vérification de santé
echo "Checking application health..."
curl -f http://localhost:3000/health || exit 1

echo "Deployment to $ENVIRONMENT completed successfully"
```

Focus GitHub

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

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

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

8.3 Documentation et monitoring

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

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

Exemple

Documentation API Swagger :

```yaml
openapi: 3.0.0
info:
  title: Project Management API
  version: 1.0.0

paths:
  /projects:
    get:
      summary: Liste des projets
      parameters:
        - name: page
          in: query
          schema:
            type: integer
            default: 1
      responses:
        '200':
          description: Liste des projets
          content:
            application/json:
              schema:
                type: object
                properties:
                  data:
                    type: array
                    items:
                      $ref: '\#/components/schemas/Project'
    post:
      summary: Créer un projet
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '\#/components/schemas/ProjectInput'
      responses:
        '201':
          description: Projet créé

components:
  schemas:
    Project:
      type: object
      properties:
        id: { type: string, format: uuid }
        name: { type: string }
        description: { type: string }
        createdAt: { type: string, format: date-time }
    ProjectInput:
      type: object
      required: [name]
      properties:
        name: { type: string, minLength: 1 }
        description: { type: string }
```

Exemple

Runbook opérationnel (1/3) :

````
# Runbook - Project Management Application

## Procédures de démarrage

### Démarrage de l'application
```bash
# Environnement de développement
docker-compose up -d

# Environnement de production
docker-compose -f docker-compose.prod.yml up -d
```

### Vérification de santé
```bash
curl -f http://localhost:3000/health
```
````

Exemple

Runbook opérationnel (2/3) :

````
## Procédures de maintenance

### Sauvegarde des données
```bash
# PostgreSQL
pg_dump -h localhost -U user projectdb > backup_$(date +%Y%m%d).sql

# MongoDB
mongodump --host localhost:27017 --db projectlogs --out backup_mongo_$(
    date +%Y%m%d)
```

### Mise à jour de l'application
```bash
# Pull de la nouvelle image
docker-compose pull

# Redémarrage avec la nouvelle image
docker-compose up -d
```
````

Exemple

Configuration de monitoring :

```
\# docker-compose.monitoring.yml
version: '3.8'

services:
  prometheus:
    image: prom/prometheus
    ports:
      - "9090:9090"
    volumes:
      - ./monitoring/prometheus.yml:/etc/prometheus/prometheus.yml
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--web.console.libraries=/etc/prometheus/console_libraries'
      - '--web.console.templates=/etc/prometheus/consoles'

  grafana:
    image: grafana/grafana
    ports:
      - "3001:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana_data:/var/lib/grafana

  node-exporter:
    image: prom/node-exporter
    ports:
      - "9100:9100"
    volumes:
      - /proc:/host/proc:ro
      - /sys:/host/sys:ro
      - /:/rootfs:ro

volumes:
  grafana_data:
```

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

8.4 Liens utiles

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

Chapitre 9

Veille technologique et sécurité

9.1 Veille technologique stack

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

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

Exemple

Sources de veille technologique :

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

Exemple de veille React :

```
React 18.2.0 (Janvier 2024)
+-- Nouvelles fonctionnalités
|   +-- Concurrent Features stabilisées
|   +-- Suspense amélioré
|   +-- Server Components en production
+-- Performances
|   +-- Réduction de 15% du bundle size
|   +-- Amélioration du rendu concurrent
+-- Migration
    +-- Breaking changes mineurs
    +-- Guide de migration disponible
```

Impact sur le projet :

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

9.2 Bonnes pratiques sécurité

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

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

Exemple

Veille sécurité OWASP 2024 :

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

Exemple de vulnérabilité suivie :

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

Mesures de sécurité appliquées :

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

9.3 Application au projet

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

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

Exemple

Évolution technique du projet :

— **Q1 2024 :** Migration vers React 18 pour les performances

— **Q2 2024 :** Implémentation des Server Components

— **Q3 2024 :** Mise à jour PostgreSQL 16 pour les JSON

— **Q4 2024 :** Migration vers Node.js 20 LTS

Améliorations sécurité appliquées :

```
// AVANT : Validation basique
const validateUser = (userData) => {
  if (userData.email && userData.password) {
    return true;
  }
  return false;
};

// APRÈS : Validation robuste avec sanitisation
const validateUser = (userData) => {
  const schema = Joi.object({
    email: Joi.string().email().max(255).required(),
    password: Joi.string().min(8).pattern(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d
        )/).required(),
    name: Joi.string().max(100).sanitize().required()
  });

  const { error, value } = schema.validate(userData);
  if (error) {
    throw new ValidationError(error.details[0].message);
  }

  return value;
};
```

Métriques d'amélioration :

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

À FAIRE / À VÉRIFIER

— Intégrer les bonnes pratiques découvertes en veille

— Planifier les migrations technologiques

— Mesurer l'impact des améliorations

— Documenter les décisions techniques

— Partager les connaissances avec l'équipe

Contrôles Jury CDA

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

9.4 Liens utiles

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

Chapitre 10

Bilan et retour d'expérience (REX)

10.1 Objectifs atteints et non atteints

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

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

Exemple

Bilan des objectifs SMART :

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

Objectifs non atteints :

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

10.2 Difficultés rencontrées et solutions

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

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

Exemple

Tableau risques ☞mitigation ☞résultat :

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

Exemple de difficulté résolue :

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

10.3 Dettes techniques et apprentissages

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

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

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

Exemple

Registre des dettes techniques :

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

Apprentissages transférables :

— **Architecture :** Pattern Repository pour l'abstraction des données

— **Performance :** Stratégies de cache multi-niveaux

— **Sécurité :** Implémentation JWT avec refresh tokens

— **Tests :** Pyramide de tests avec couverture optimale

— **DevOps :** Pipeline CI/CD avec déploiement blue-green

Exemple d'apprentissage concret :

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

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

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

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

10.4 Liens utiles

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

Chapitre 11

Conclusion et remerciements

11.1 Synthèse du projet

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

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

Exemple

Chiffres clés du projet :

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

Technologies maîtrisées :

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

11.2 Perspectives d'évolution

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

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

Exemple

Roadmap technique v2.0 :

```
Q1 2025: Fonctionnalités avancées
+-- Analytics prédictives avec machine learning
+-- Intégrations API externes (Slack, Teams)
+-- Notifications push temps réel
+-- Optimisation performances (P95 < 200ms)

Q2 2025: Intelligence artificielle
+-- Assistant IA pour la gestion de projet
+-- Recommandations automatiques
+-- Détection d'anomalies
+-- Chatbot support utilisateur

Q3 2025: Évolutions technologiques
+-- Migration vers React Server Components
+-- Mise à jour Node.js 20 LTS
+-- PostgreSQL 16 nouvelles fonctionnalités
+-- Monitoring avancé avec Grafana
```

Compétences à développer :

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

11.3 Remerciements

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

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

Exemple

Remerciements personnalisés :

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

Apprentissages clés :

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

11.4 Déploiement et documentation

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

opérationnelle et de la maintenabilité de votre solution.

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

11.4.1 Docker

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

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

Conteneurisation

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

Exemple

Dockerfile multi-stage :

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

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

Compose

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

Exemple

Docker Compose pour l'environnement complet :

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

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

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

volumes:
  postgres_data:
```

11.4.2 GitHub (code source)

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

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

Exemple

Structure du repository :

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

11.4.3 CI/CD

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

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

Exemple

Pipeline CI/CD GitHub Actions :

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

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

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

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

11.4.4 SonarQube

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

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

Exemple

Métriques de qualité SonarQube :

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

11.4.5 Swagger

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

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

Exemple

Documentation API Swagger :

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

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

components:
  schemas:
    Project:
      type: object
      properties:
        id:
          type: string
          format: uuid
        name:
          type: string
        description:
          type: string
        createdAt:
          type: string
          format: date-time
```

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

11.5 Liens utiles

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