✓×*makepdf*

Colint School

Formation CDA 2025

Concepteur Développeur d'Applications

RNCP37873

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

My Smart Budget

Dossier de Projet CDA
Présentation et Défense

Informations du Projet

Candidat : SAIDI Abdelghani

Promotion : Bachelor 3

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 Présentation du rôle et du contexte

Mon rôle

Je suis **développeur full-stack**, chargé de concevoir, développer et maintenir des applications web performantes, en assurant la cohérence entre le front-end et le back-end. Mon objectif est de transformer des besoins fonctionnels en solutions techniques robustes, maintenables et évolutives.

Dans le cadre de mon alternance, j'occupe un rôle transversal impliquant une collaboration étroite avec le chef de projet, le designer UI/UX et les autres développeurs. J'interviens à chaque étape du cycle de développement :

- **Analyse du besoin** : compréhension des attentes utilisateurs et formalisation du cahier des charges ;
- **Conception technique** : définition de l'architecture, choix des technologies, création des modèles de données ;
- **Développement front-end et back-end** : implémentation des fonctionnalités principales et intégration des API ;
- **Tests et validation** : mise en place de tests unitaires et fonctionnels pour assurer la fiabilité de l'application ;
- **Déploiement et maintenance** : conteneurisation avec Docker, déploiement cloud (Render/AWS) et suivi des performances.

Ce rôle me permet d'acquérir une vision complète du cycle de vie d'un projet web tout en développant mes compétences techniques et organisationnelles.

Contexte organisationnel

Je travaille au sein d'une entreprise spécialisée dans le développement de solutions numériques sur mesure. L'équipe technique, composée d'un chef de projet, de deux développeurs full-stack et d'un designer UI/UX, fonctionne selon une organisation agile (Scrum) qui favorise l'autonomie et la collaboration.
L'objectif de l'entreprise est de proposer des applications modernes, intuitives et sécurisées, conçues pour répondre à des problématiques réelles d'efficacité et d'accessibilité. Dans ce cadre, mon projet principal, **My Smart Budget**, a pour mission de répondre à un besoin croissant : aider les utilisateurs à mieux gérer leurs finances personnelles grâce à une solution claire, intelligente et accessible depuis tout support.
My Smart Budget permet aux utilisateurs de :

— suivre leurs revenus et dépenses en temps réel ;
— catégoriser automatiquement leurs transactions ;
— fixer des objectifs d'épargne et recevoir des alertes personnalisées ;
— visualiser leurs données financières sous forme de graphiques interactifs.

Le projet repose sur une architecture web moderne et sécurisée : *React.js* pour le front-end, *Node.js/Express* pour le back-end, *MongoDB* pour le stockage, et *Docker/AWS* pour le déploiement.

Durée et planification du projet

Mon alternance s'étend sur une année complète. Le développement de **My Smart Budget** s'est déroulé sur six mois selon quatre grandes phases :

— **Phase 1 – Analyse et conception (mois 1)** : étude des besoins utilisateurs, benchmark des solutions existantes et création des maquettes Figma ;
— **Phase 2 – Développement (mois 2 à 4)** : mise en place de l'API REST, développement du front-end, intégration des services externes et mise en place de la base de données ;
— **Phase 3 – Tests et validation (mois 5)** : élaboration des scénarios de tests, validation fonctionnelle et corrections ;
— **Phase 4 – Déploiement et documentation (mois 6)** : conteneurisation, déploiement sur Render/AWS et rédaction de la documentation technique.

1.2 Problématique identifiée

Problématique reformulée

Cause : De nombreux utilisateurs, notamment les jeunes actifs et les familles, rencontrent des difficultés à suivre leurs dépenses quotidiennes de manière claire et centralisée. Les outils bancaires existants sont souvent complexes, peu personnalisés et centrés sur la comptabilité pure.
Conséquence : Cette complexité décourage les utilisateurs, qui perdent en visibilité sur leurs finances, entraînant un manque de maîtrise budgétaire et des dépassements réguliers.
Impact : L'utilisateur subit une perte de confiance et de contrôle sur sa situation financière.
Problématique principale : *Comment concevoir une application simple, visuelle et intelligente permettant à un utilisateur non expert de reprendre le contrôle de son budget, d'anticiper ses dépenses et d'adopter de meilleures habitudes financières ?*

1.3 Bénéfices métier attendus

Analyse des gains et bénéfices attendus

Le projet **My Smart Budget** vise à générer des gains mesurables tant pour les utilisateurs que pour l'entreprise :

— **Gain de productivité :** réduction du temps de saisie des dépenses grâce à l'automatisation et à la catégorisation intelligente (gain estimé de 30 %) ;
— **Réduction d'erreurs :** fiabilisation des calculs et des rapports budgétaires grâce à la centralisation des données ;
— **Amélioration de l'expérience utilisateur :** interface simplifiée, accessible et motivante (taux de satisfaction visé : 90 %) ;
— **Valorisation de la marque :** démonstration du savoir-faire technique de l'entreprise sur un produit complet et fonctionnel.

1.4 Objectifs SMART

Objectifs SMART du projet

- **Spécifique :** Fournir une application web permettant de suivre et d'analyser les dépenses personnelles en temps réel via des tableaux de bord et des alertes automatiques ;
- **Mesurable :** Atteindre un taux d'adoption de 70 % sur un panel de 20 utilisateurs internes et réduire de 25 % les dépassements budgétaires mensuels ;
- **Atteignable :** Développement réalisé par une équipe de trois personnes sur six mois, avec des sprints de deux semaines ;
- **Pertinent :** Répond à un besoin utilisateur réel d'autonomie et de compréhension financière ;
- **Temporel :** Version 1.0 déployée en octobre 2025, version 2.0 planifiée pour décembre 2025 avec retour d'expérience.

1.5 Indicateurs de succès

Indicateurs de performance mesurables

- **Taux d'adoption utilisateur :** 70 % trois mois après la mise en ligne ;
- **Taux de satisfaction :** supérieur à 90 % selon un questionnaire d'évaluation post-utilisation ;
- **Réduction du temps de saisie des dépenses :** 20 % en moyenne grâce à l'automatisation ;
- **Réduction des dépassements budgétaires :** 25 % en deux mois d'utilisation ;
- **Disponibilité du service :** 99,5 % garantie grâce à l'hébergement cloud.

1.6 Diagramme de contexte

Description du diagramme

Le diagramme suivant illustre les principaux flux d'informations entre les acteurs et les composants techniques de l'application. Chaque flèche représente une interaction ou un échange de données entre les différentes parties du système.

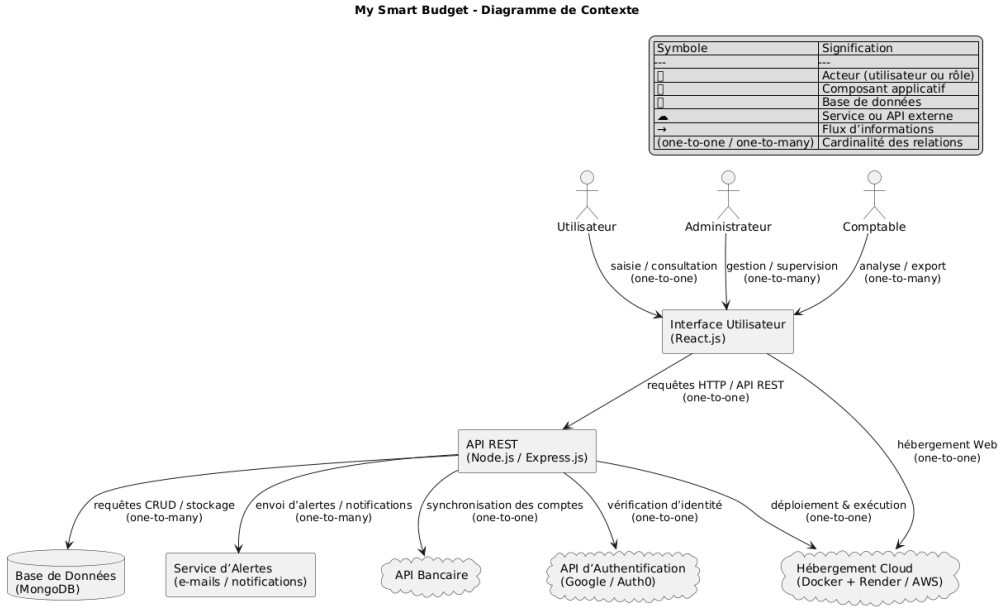


FIGURE 1.1 – Diagramme de contexte de l'application My Smart Budget — flux d'informations et interactions principales

Légende du diagramme

— Les flèches pleines représentent les flux d'informations (*requêtes, réponses, synchronisations*) ;
— Les flèches pointillées représentent les interactions utilisateur (*actions sur l'interface*) ;
— Chaque acteur (utilisateur, administrateur, comptable) interagit via l'interface web avec l'API REST, qui gère la logique métier et la base de données MongoDB.

1.7 Valeur ajoutée du projet

Apports du projet My Smart Budget

- Simplifie la gestion financière personnelle grâce à l'automatisation et à la visualisation claire des données ;
- Réduit les erreurs et le temps de saisie manuelle de 20 % ;
- Offre une meilleure compréhension des habitudes de dépenses grâce à des rapports détaillés et personnalisés ;
- Favorise la prise de décision financière et l'autonomie des utilisateurs ;
- Met en valeur mes compétences techniques (React, Node.js, MongoDB, Docker, AWS) et ma capacité à gérer un projet complet en méthodologie Agile.

1.8 Diagramme de classes UML

Description du diagramme

Le diagramme de classes UML présenté ci-dessous modélise la structure interne de l'application **My Smart Budget**. Il illustre les principales entités du système (utilisateurs, transactions, budgets, catégories, alertes, etc.) ainsi que leurs relations. Chaque classe correspond à une table logique dans la base de données MongoDB, tandis que les associations définissent les liens entre ces entités.

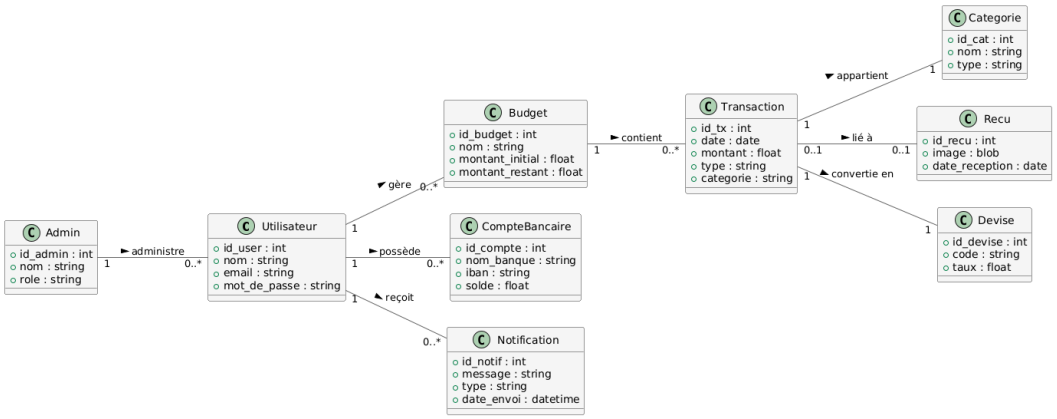


FIGURE 1.2 – Diagramme de classes UML de l'application My Smart Budget

Analyse du modèle de données

Le modèle repose sur une organisation simple, cohérente et évolutive, pensée pour faciliter les traitements financiers et les analyses statistiques :

— **Utilisateur** : entité principale du système. Chaque utilisateur possède un ou plusieurs comptes, budgets, transactions et objectifs. Les données d'authentification et de profil sont sécurisées.

— **Compte** : regroupe les informations liées aux comptes bancaires ou portefeuilles virtuels. Il contient plusieurs transactions.

— **Transaction** : enregistre les revenus et dépenses de l'utilisateur, en précisant la catégorie, la date, le montant et la description. Chaque transaction est liée à un compte et une catégorie.

— **Catégorie** : permet de classifier les transactions (ex. alimentation, transport, loisirs). Elle sert également à agréger les données dans les tableaux de bord.

— **Budget** : définit des limites de dépenses par catégorie ou par période. Il est associé à un utilisateur et à une catégorie spécifique.

— **Objectif** : représente une cible d'épargne ou de dépenses à atteindre dans un délai défini (ex. "économiser 500 €").

— **Alerte** : notifie l'utilisateur lorsqu'un seuil budgétaire est atteint ou lorsqu'une transaction inhabituelle est détectée.

Relations entre les entités

Le modèle repose sur des relations claires et normalisées :

— **Utilisateur 1..* Compte** : un utilisateur peut avoir plusieurs comptes (ex. compte courant, compte épargne) ;

— **Compte 1..* Transaction** : un compte contient plusieurs transactions ;

— **Transaction *..1 Catégorie** : chaque transaction appartient à une seule catégorie, mais une catégorie peut regrouper plusieurs transactions ;

— **Utilisateur 1..* Budget** : un utilisateur peut définir plusieurs budgets, selon les catégories ;

— **Budget 1..1 Catégorie** : un budget est lié à une catégorie unique ;

— **Utilisateur 1..* Objectif** : chaque utilisateur peut planifier plusieurs objectifs financiers ;

— **Budget 1..* Alerte** : lorsqu'un budget dépasse un seuil, une ou plusieurs alertes peuvent être générées.

Synthèse du modèle

L'architecture du modèle de données garantit :

— une **extensibilité** facilitant l'ajout de nouvelles fonctionnalités (ex. gestion multi-devise, import bancaire) ;

— une **intégrité des données** grâce aux liens explicites entre les entités ;

— une **optimisation des performances** avec des requêtes structurées pour l'analyse budgétaire ;

— une **maintenance simplifiée** via des relations logiques et des entités bien isolées.

Ainsi, le diagramme de classes constitue la base de la conception technique et de la logique applicative de **My Smart Budget**.

1.9 Synthèse et perspectives d'évolution du projet

Synthèse du projet

Le projet **My Smart Budget** s'inscrit dans une démarche de modernisation et de simplification de la gestion financière personnelle. Conçu comme une application web intuitive et intelligente, il vise à offrir une expérience utilisateur fluide tout en intégrant des fonctionnalités avancées telles que :

— la **centralisation des transactions** (revenus et dépenses) en temps réel ;

— la **catégorisation automatique** des opérations selon leur nature ;

— la **gestion des budgets** avec alertes intelligentes et seuils configurables ;

— la **visualisation graphique** des tendances financières et des objectifs atteints ;

— l'accès sécurisé via un système d'authentification et une architecture conforme aux bonnes pratiques du développement web moderne.

Le développement du projet a permis de mettre en œuvre l'ensemble des compétences clés du référentiel CDA : de la conception UML à l'intégration front/back, en passant par la gestion de projet agile, la sécurité applicative et la conteneurisation avec Docker.

Enjeux techniques et fonctionnels maîtrisés

Ce projet m'a permis de consolider mes compétences dans plusieurs domaines :

— **Front-end :** création d'interfaces réactives et ergonomiques avec *React.js* et *Tailwind CSS* ;

— **Back-end :** mise en place d'une API REST sécurisée sous *Node.js / Express.js* avec validation des données et gestion des erreurs ;

— **Base de données :** modélisation et gestion des collections sous *MongoDB* avec schémas cohérents ;

— **Sécurité et performances :** intégration de l'authentification JWT, limitation des requêtes et cryptage des mots de passe ;

— **Déploiement :** conteneurisation via *Docker*, tests avec Postman et hébergement sur *Render / AWS*.

Ces choix technologiques garantissent la fiabilité, la scalabilité et la maintenabilité de l'application.

Perspectives d'évolution

Afin d'assurer la continuité du projet et d'enrichir son potentiel fonctionnel, plusieurs pistes d'amélioration sont envisagées pour les versions futures :

— **Synchronisation bancaire automatique** : import direct des transactions via API sécurisée (type Linxo, Budget Insight) ;

— **Application mobile native** : développement d'une version mobile avec *React Native* pour une accessibilité complète ;

— **Gestion multi-devise et multi-profil** : permettre à un utilisateur de gérer plusieurs comptes ou familles de budgets ;

— **Module de prévision intelligente** : ajout d'un algorithme prédictif basé sur l'historique des dépenses ;

— **Partage collaboratif du budget** : option permettant à plusieurs utilisateurs (ex. couple) de suivre un budget commun en temps réel.

Ces évolutions permettront de transformer **My Smart Budget** en un véritable assistant financier personnalisé, combinant simplicité d'usage, puissance analytique et connectivité moderne.

Bilan personnel

Ce projet représente pour moi une expérience professionnelle formatrice et concrète. Il m'a permis de :

— renforcer mes compétences techniques en développement full-stack ;

— améliorer ma rigueur dans la gestion des versions et la documentation du code ;

— développer une approche orientée utilisateur et centrée sur l'expérience client ;

— adopter une méthodologie agile efficace, basée sur la planification en sprints et la communication d'équipe ;

— prendre conscience de l'importance de la sécurité, de la maintenabilité et de l'évolutivité dans un projet réel.

En conclusion, **My Smart Budget** constitue une première étape solide vers la conception d'applications web à forte valeur ajoutée, combinant technologie, ergonomie et utilité concrète.

Chapitre 2
Cadrage et Cahier des Charges

2.1 Objectifs métier, techniques et pédagogiques

2.1.1 Objectifs métier

My Smart Budget répond à un besoin concret identifié auprès de plusieurs profils d'utilisateurs : de nombreux étudiants, jeunes actifs et familles peinent à suivre leurs dépenses quotidiennes et à planifier leurs budgets mensuels de manière efficace.

L'objectif métier principal est de **simplifier la gestion budgétaire personnelle** via une interface claire, automatisée et personnalisable, tout en offrant une vision en temps réel de la situation financière.

Bénéfices attendus mesurables

Bénéfice	**Indicateur de succès**	**Objectif**
Gain de temps	Temps moyen de saisie mensuelle	$<$ 15 min/mois via import CSV
Accessibilité multi-profils	Taux d'adoption par profil	$\geq$ 70 % pour chaque persona
Éducation financière	% utilisateurs atteignant objectifs	$\geq$ 60 % après 3 mois
Visibilité financière	Temps d'accès données clés	$<$ 5 secondes (dashboard)
Simplification du suivi	Réduction erreurs budgétaires	-40 % vs saisie manuelle

2.1.2 Objectifs techniques

Architecture **3-tiers évolutive** : Front-end (Next.js 14/React 18), Back-end (Node.js 20/NestJS 10), Base de données (PostgreSQL 15 via Prisma 5).

Exigences de performance (mesurables)

Critère	Métrique	Objectif	Justification métier
2***Performance**	Time-to-Render dashboard	< 2s (P95)	Expérience utilisateur fluide
	Latence API	< 500ms (P95)	Réactivité opérations CRUD
3***Sécurité**	Authentification	JWT + bcrypt (cost ≥ 12)	Protection données RGPD
	Chiffrement	HTTPS/TLS 1.3	Confidentialité transactions
	Audit	Logs connexions	Traçabilité conformité
2***Maintenabilité**	Couverture tests	$\geq 60\,\%$ (v1.0)	Réduction bugs production
	Migrations	Schémas versionnés	Évolutivité sans rupture
2***Scalabilité**	Architecture	Stateless API + Docker	Support 100 $\rightarrow$ 10k users
	Cache (v2)	Redis sessions	Réduction charge DB
2***Fiabilité**	Disponibilité	99,5 % (hors maint.)	Accès continu données
	Sauvegardes	Quotidiennes + tests	RPO $<$ 24h, RTO $<$ 4h

2.1.3 Objectifs pédagogiques

Projet inscrit dans le cadre du **Titre Professionnel Concepteur Développeur d'Applications (CDA - Niveau 6)**.

Compétences visées et référentiel

Bloc de compétences CDA	Compétences développées	Livrables associés
Concevoir et développer des composants d'interface	Design System, composants React réutilisables, accessibilité (WCAG 2.1 AA)	Pages Auth, Dashboard, formulaires validés
Concevoir et développer la persistance des données	Modélisation Merise (MCD/MLD/MPD), Prisma ORM, migrations versionnées	Schémas DB, documentation MCD, scripts SQL
Concevoir et développer une application multicouche répartie	Architecture 3-tiers, API REST (OpenAPI 3.0), services métier (NestJS)	Documentation architecture, API Swagger, tests E2E

Méthodologie agile : Sprints de 2 semaines, backlog GitHub, retrospectives documentées, déploiement continu (CI/CD).

2.2 Priorisation des fonctionnalités (Méthode MoSCoW)

La méthode MoSCoW permet de classer les exigences selon leur criticité pour garantir un MVP (Minimum Viable Product) cohérent et livrable.

2.2.1 Tableau de priorisation détaillé

Priorité	Fonctionnalité	Justification métier	Complexité
Must Have	Authentification JWT + gestion sessions	Sécurité, protection RGPD, prérequis légal	Moyenne (5j)
Must Have	Transactions CRUD + catégorisation	Cœur fonctionnel du suivi budgétaire	Moyenne (8j)
Must Have	Enveloppes budgétaires (plafonds mensuels)	Différenciateur vs tableurs, contrôle budgétaire	Moyenne (6j)
Must Have	Dashboard synthétique (KPIs + graphiques)	Visibilité immédiate, prise de décision	Élevée (10j)
Should Have	Import CSV standardisé	Gain de temps ×10, adoption utilisateurs	Moyenne (5j)
Should Have	Export CSV/JSON (portabilité RGPD)	Conformité Article 20 RGPD, transparence	Faible (3j)
Could Have	Notifications (dépassement enveloppe)	Valeur ajoutée, prévention dépassements	Faible (4j)
Could Have	Gestion multi-comptes bancaires	Réalisme pour utilisateurs multiples banques	Moyenne (5j)
Won't Have (v1)	Agrégation bancaire automatique (DSP2)	Complexe (PSD2, APIs tierces), coût élevé	—
Won't Have (v1)	OCR de factures/tickets	Technologie ML, précision insuffisante v1	—
Won't Have (v1)	Recommandations IA budgétaires	Nécessite historique > 6 mois	—

2.2.2 Justification des exclusions (Won't Have v1)

— **Agrégation bancaire** : Nécessite certification PSD2 (6–12 mois), coûts APIs élevés (Plaid/Budget Insight), maintenance complexe. Prévu v2.0 après POC.

— **OCR** : Taux d'erreur élevé sur tickets français (formats variés), coût API externe ($1.50/1000 requêtes). Évaluation v2.1.

— **IA de recommandations** : Algorithmes ML nécessitant dataset de 1000+ utilisateurs sur 6 mois minimum. Prévu v2.5.

2.3 Périmètre du MVP (Version 1.0)

2.3.1 Fonctionnalités incluses

Module	Fonctionnalités	Livrables techniques
Authentification	Inscription, Connexion JWT (exp 24h), Déconnexion, Réinit. mot de passe	Module NestJS `auth`, endpoints REST, tests unitaires
Profil utilisateur	CRUD profil, Modification mot de passe, Suppression compte (RGPD)	Module `users`, page React `/profile`, validation Zod
Catégories	15 catégories prédéfinies, Catégories personnalisées, CRUD	Table `Category`, API CRUD, composant `CategoryManager`
Enveloppes	Création avec plafond mensuel, Calcul "reste à vivre", Indicateurs visuels, Historique	Module `budgets`, calculs temps réel, page `/envelopes`
Transactions	CRUD complet, Filtres (date, catégorie, montant), Recherche textuelle	Module `transactions`, API paginée (limit 50), tests E2E
Import CSV	Format gabarit, Validation + détection doublons, Rapport d'import	Endpoint `/import/csv`, parser Papaparse, page `/import`
Dashboard	KPIs (solde, dépenses, revenus), Graphiques (camembert, courbes), Filtres temporels	Page `/dashboard`, composants Recharts, requêtes optimisées
Export	Export CSV et JSON des transactions	Endpoints `/export/csv` et `/export/json`, boutons UI
Droits d'accès	Rôle Utilisateur et Admin	Guards NestJS `RoleGuard`, middleware autorisation

2.3.2 Fonctionnalités exclues de la v1.0

— Agrégation bancaire automatique
— OCR de factures
— Recommandations IA
— Multi-devises (euro uniquement en v1)
— Partage de budgets entre utilisateurs
— Application mobile native (PWA responsive uniquement)

2.3.3 Contraintes d'acceptation globales

Critère	Seuil d'acceptation	Méthode de mesure
Performance dashboard	TTR P95 < 2s	Lighthouse, GTmetrix
Fiabilité API	Taux d'erreur $< 1\,\%$ (P95)	Logs applicatifs, monitoring
Disponibilité	99,5 % (hors maintenance)	Uptime monitoring (UptimeRobot)
Sécurité	0 vulnérabilité critique (OWASP Top 10)	npm audit, Snyk, revue code

2.4 Cibles, personae et parties prenantes

2.4.1 Personae détaillés

Persona 1 — Léa, 23 ans, Étudiante en Master

Profil : 800€/mois (APL + job étudiant), vit en colocation à Lyon.

Objectifs
— Suivre ses dépenses contraintes (loyer 400€, transport 70€, alimentation 200€)
— Garder un reste à vivre hebdomadaire de 30€ minimum
— Éviter les découverts bancaires (frais 8€/incident)

Points de douleur actuels
— Oublie de noter 40 % de ses achats quotidiens
— Fin de mois difficile (reste < 20€ les 5 derniers jours)
— Utilise un tableur Excel compliqué et non synchronisé

Critères de succès avec My Smart Budget
— Visualiser son reste à vivre en 1 clic (< 5s)
— Recevoir une alerte 3 jours avant épuisement d'une enveloppe
— Réduire le temps de saisie mensuel de 2h à 15 min (import CSV)

Scénarios d'usage prioritaires
1. Ajout rapide d'une transaction mobile (course de 15€)
2. Consultation du reste à vivre avant une sortie
3. Import CSV mensuel de son relevé bancaire

Persona 2 — Marc, 38 ans, Père de famille

Profil : 4500€/mois (couple biactif), 2 enfants (8 et 12 ans), propriétaire en région parisienne.

Objectifs
— Répartir le budget familial par enveloppes (courses 600€, loisirs 300€, enfants 400€, épargne 500€)
— Anticiper les pics de dépenses (rentrée scolaire +800€, vacances +1500€)
— Impliquer sa conjointe dans le suivi budgétaire

Points de douleur actuels
— Manque de visibilité par catégorie (dépenses enfants dispersées)
— Dépassements fréquents sur l'enveloppe "loisirs" (+20 % en moyenne)
— Aucun outil partagé avec sa conjointe (double saisie fastidieuse)

Critères de succès avec My Smart Budget
— Respecter 90 % des plafonds mensuels d'enveloppes
— Visualiser les dépenses enfants consolidées (école, activités, santé)
— Exporter un bilan mensuel en PDF pour discussions de couple

Scénarios d'usage prioritaires
1. Création de 8 enveloppes avec plafonds mensuels
2. Analyse des dépassements du mois précédent
3. Export CSV pour comptable (déclaration impôts)

2.4.2 Parties prenantes

Parties prenantes primaires

Acteur	Rôle	Besoins	Critères de satisfaction
Abdelghani Saidi	Porteur projet / Dev CDA	Valider compétences CDA, livrables conformes	Soutenance réussie, projet déployé, doc complète
Utilisateurs finaux	Testeurs MVP (3–5 pers.)	Appli stable, intuitive, sans bugs	SUS Score $\geq$ 75, erreurs $<$ 5 %
Jury CDA	Évaluateurs certification	Conformité référentiel, qualité technique	Grille CDA : $\geq$ 12/20 par bloc

Parties prenantes secondaires

Acteur	Influence	Intérêt	Stratégie d'engagement
Hébergeur cloud (OVH/Vercel)	Moyenne	Faible	Contrat SLA, surveillance coûts ($<$ 50€/mois v1)
Fournisseurs APIs bancaires (v2)	Moyenne	Moyen	Veille technologique, POC Budget Insight (2026)
GitHub/Docker Hub	Faible	Faible	Utilisation gratuite (repos publics), CI/CD inclus
Communauté open-source (post-v1)	Faible	Moyen	Doc contributeurs, licence MIT, roadmap publique

2.4.3 Matrice influence/intérêt

Quadrant	Acteurs	Stratégie
Forte influence + Fort intérêt	Porteur de projet, Jury CDA	**Gérer étroitement** : communication hebdo, validation jalons
Forte influence + Faible intérêt	Hébergeur (si indispo)	**Maintenir satisfait** : monitoring 24/7, SLA respecté
Faible influence + Fort intérêt	Utilisateurs testeurs	**Tenir informés** : newsletter sprint, changelog, feedback loops
Faible influence + Faible intérêt	Fournisseurs secondaires	**Surveiller** : veille passive, pas d'actions v1

2.5 User Stories et Critères d'Acceptation

2.5.1 User Stories priorisées (P1 → P3)

ID	P	User Story	Valeur	Effort
US-01	P1	En tant qu'utilisateur, je peux créer un compte et me connecter	Prérequis RGPD	5
US-02	P1	En tant qu'utilisateur, je peux créer/éditer/supprimer des transactions et les catégoriser	Cœur fonctionnel	8
US-03	P1	En tant qu'utilisateur, je peux créer des enveloppes et voir mon reste à vivre	Différenciateur	8
US-04	P2	En tant qu'utilisateur, je peux importer un CSV de transactions	Gain temps x10	5
US-05	P2	En tant qu'utilisateur, je vois un dashboard avec graphiques et KPIs	Décision rapide	13
US-06	P2	En tant qu'utilisateur, je peux exporter mes transactions (CSV/JSON)	Conformité RGPD	3
US-07	P3	En tant qu'utilisateur, je reçois une notification à 80 % du plafond	Prévention	5
US-08	P3	En tant qu'admin, j'accède au backoffice avec stats globales	Pilotage produit	8

2.5.2 Critères d'acceptation détaillés (Definition of Done)

US-01 : Compte & Connexion

Critères fonctionnels

1. Formulaire d'inscription avec validations :
   — Email unique (format RFC 5322)
   — Mot de passe ≥ 12 caractères (1 maj, 1 min, 1 chiffre, 1 spécial)
   — Confirmation mot de passe identique
2. Mot de passe haché avec bcrypt (cost factor ≥ 12)
3. Token JWT signé (HS256, expiration 24h, payload : userId + rôle)
4. Redirection automatique vers `/dashboard` après connexion

5. Mécanisme anti-brute force : 3 tentatives → temporisation 5 min

Critères techniques
— Tests unitaires : 5 cas (inscription OK, email doublon, mot de passe faible, JWT valide, JWT expiré)
— Tests E2E : parcours complet inscription → connexion → page protégée
— Latence endpoint `/register` < 800ms (P95)
— Logs audit : tentatives échouées + IP (conservation 90j)

Critères d'acceptation mesurables
✓ 100 % des connexions avec credentials valides réussissent
✓ 0 token JWT accepté après expiration
✓ Taux d'activation ≥ 80 % (compte créé → 1re connexion dans 7j)

US-02 : Transactions CRUD

Critères fonctionnels
1. Champs obligatoires : date (ISO 8601), montant (décimal 2 chiffres), catégorie (FK), libellé (max 200 char), type (DEBIT/CREDIT)
2. Création d'une transaction < 500ms (P95)
3. Liste paginée (50 items/page) avec tri (date DESC) et filtres (catégorie, dates, montant)
4. Édition conservant historique (champ `updated_at`)
5. Suppression avec confirmation modale
6. Total par catégorie mis à jour instantanément

Critères techniques
— Index DB : `(userId, date DESC)`, `(userId, category_id)`
— Validation backend (Zod) + frontend (React Hook Form)
— Tests unitaires : CRUD complet (8 cas), calculs totaux (3 cas)
— Tests E2E : scénario ajout → édition → suppression

Critères d'acceptation mesurables
✓ Latence création P95 < 500ms
✓ 0 incohérence entre total affiché et somme DB
✓ Taux d'erreur validation < 2 %

US-03 : Enveloppes Budgétaires

Critères fonctionnels
1. Création enveloppe : nom (unique), plafond mensuel (€), catégories liées (1 à N)
2. Calcul "reste à vivre" : Revenus − Σ(Dépenses enveloppes) − Σ(Dépenses hors env.)

3. Indicateurs visuels :
   — Vert : consommé $< 80\,\%$
   — Orange : $80\,\% \leq$ consommé $< 100\,\%$
   — Rouge : consommé $\geq 100\,\%$
4. Historique mensuel : graphique évolution sur 12 mois
5. Export CSV des enveloppes

Critères techniques
— Requête SQL optimisée : agrégation par `envelope_id` avec index
— Recalcul temps réel : polling 30s (v1)
— Tests unitaires : calculs ”reste à vivre” (5 cas limites)

Critères d'acceptation mesurables
✓ Temps chargement page `/envelopes` $<$ 1,5s (P95)
✓ Précision calculs : 0 écart vs calcul SQL manuel
✓ Adoption : $\geq 70\,\%$ utilisateurs créent ≥ 3 enveloppes (30j)

US-04 : Import CSV

Critères fonctionnels
1. Format gabarit : `date,amount,category,label,type`
2. Validation fichier :
   — Format date : ISO 8601 ou français (DD/MM/YYYY)
   — Montant : décimal positif
   — Catégorie : nom exact ou ID existant
   — Détection doublons : même date + montant + libellé
3. Performance : import 10 000 lignes en $<$ 20s
4. Rapport d'import : créées/ignorées/erreurs avec numéros de ligne

Critères techniques
— Parser : Papaparse (UTF-8/Latin1)
— Traitement par batchs de 500 lignes
— Transaction DB : rollback si $> 10\,\%$ lignes invalides
— Tests unitaires : 8 cas (fichier valide, date invalide, doublons, etc.)

Critères d'acceptation mesurables
✓ Taux d'import réussi $> 95\,\%$ (fichiers réels)
✓ Temps import 10k lignes $<$ 20s
✓ 0 perte de données si interruption

US-05 : Dashboard Interactif

Critères fonctionnels

1. KPIs affichés :
   — Solde total actuel (€)
   — Dépenses du mois en cours (€)
   — Revenus du mois en cours (€)
   — Reste à vivre estimé (€)
2. Graphiques :
   — Camembert : répartition dépenses par catégorie (top 10)
   — Courbe : évolution dépenses/revenus sur 12 mois
3. Filtres temporels : mois, trimestre, année, période personnalisée
4. Responsive : breakpoints 400px, 768px, 1024px

Critères techniques

— Librairie graphiques : Recharts (composants React natifs)
— Optimisation requêtes : agrégations SQL avec `GROUP BY` + index sur `date`
— Cache côté serveur : memoization (v1) pour données mensuelles (TTL 1h)
— Tests E2E : vérification affichage graphiques, filtres fonctionnels

Critères d'acceptation mesurables

✓ Time-to-Render (TTR) < 2s (P95, Lighthouse)
✓ Taux d'usage hebdomadaire $> 60\,\%$ (Google Analytics)
✓ 0 erreur JS console sur Chrome 120+, Firefox 120+, Safari 17+

US-06 : Export CSV/JSON

Critères fonctionnels

1. Export CSV : format compatible Excel/LibreOffice (séparateur `;`, encodage UTF-8 BOM)
2. Export JSON : structure conforme Article 20 RGPD (machine-readable)
3. Période sélectionnable : mois, trimestre, année, tout l'historique
4. Nom fichier : `transactions_2025-11.csv` (inclut période)

Critères techniques

— Génération côté serveur (éviter surcharge client)
— Limite : 50 000 transactions max par export
— Tests unitaires : 3 formats (CSV, JSON, période vide)

Critères d'acceptation mesurables

✓ Temps génération < 5s pour 10k transactions

✓ 100 % des exports téléchargés sont lisibles

✓ Conformité RGPD : export inclut toutes les données personnelles

2.6 Spécifications Fonctionnelles et Techniques

2.6.1 Répartition Front-end / Back-end

Couche	Technologies	Responsabilités	Livrables v1.0
Front-end	Next.js 14, React 18, TypeScript, Tailwind CSS, React Hook Form + Zod, Recharts	Interface responsive, Validation formulaires, State management, Graphiques, Accessibilité WCAG 2.1 AA	Pages : `/auth`, `/transactions`, `/envelopes`, `/dashboard`, `/profile`, `/import`. Composants réutilisables. Tests E2E (Playwright)
Back-end	Node.js 20, NestJS 10, TypeScript, Passport (JWT), class-validator, Swagger/OpenAPI	API REST (CRUD), Logique métier, Authentification, Rate limiting (10 req/sec), Logs (Winston)	Modules : `auth`, `users`, `transactions`, `budgets`, `categories`. Documentation OpenAPI. Tests unitaires (Jest, couv. $\geq 60\,\%$)
Base de données	PostgreSQL 15, Prisma 5, Prisma Migrate	Schémas versionnés, Contraintes d'intégrité, Index performance, Sauvegardes (pg_dump)	MCD/MLD/MPD (diagrammes UML + Merise). Scripts migrations. Seeds. Politique rétention (soft delete < 7 ans)

2.6.2 Sécurité, Confidentialité et RGPD

Authentification et Chiffrement

Mesure	Implémentation	Justification
Authentification	JWT (algorithme HS256, clé 256 bits, exp 24h)	Stateless, scalable, standard industrie
Mots de passe	Bcrypt (cost factor 12, salts automatiques)	Résistance brute-force (2^{12} itérations $\approx$ 250ms/hash)
HTTPS	TLS 1.3 obligatoire (Let's Encrypt)	Chiffrement transit, protection MITM
Secrets	Variables d'environnement (`.env` exclu Git)	Pas de secrets en clair dans le code

Matrice Rôles / Permissions

Action	Utilisateur	Admin	Invité
Voir ses transactions	✓	✓	×
Modifier ses transactions	✓	✓	×
Voir transactions autres users	×	✓(anon.)	×
Créer enveloppes	✓	✓	×
Accéder backoffice stats	×	✓	×
Supprimer compte utilisateur	✓(son compte)	✓(tous)	×
Gérer catégories globales	×	✓	×
Exporter données (RGPD)	✓	✓	×

Conformité RGPD

Droits des utilisateurs (implémentation)

Droit RGPD	Implémentation technique	Délai de réponse
Droit à l'oubli (Art. 17)	Endpoint `DELETE /users/:id` avec suppression en cascade (transactions, enveloppes, catégories perso)	Immédiat (soft delete 30j)
Droit à la portabilité (Art. 20)	Endpoint `GET /export/all` (JSON structuré : profil + transactions + enveloppes)	< 5 secondes
Droit d'accès (Art. 15)	Page `/profile/data` listant toutes les données stockées	Temps réel
Consentement explicite (Art. 7)	Case à cocher lors de l'inscription ("J'accepte la politique de confidentialité")	Inscription bloquée si refus
Journalisation accès (Art. 30)	Table `audit_logs` : `user_id`, `action`, `ip`, `timestamp` (conservation 90j)	Temps réel

Procédure d'exécution (SOP)

1. **Suppression compte** :
   — Utilisateur clique sur "Supprimer mon compte" (`/profile/delete`)
   — Confirmation par email (lien sécurisé valide 24h)
   — Soft delete : `deleted_at = NOW()`, données conservées 30j (récupération possible)
   — Hard delete après 30j : purge définitive par job CRON quotidien
2. **Export données** :

— Utilisateur clique sur "Télécharger mes données" (`/profile/export`)
— Génération archive ZIP (JSON + CSV) côté serveur
— Téléchargement direct (pas de stockage temporaire)
— Log de l'action dans `audit_logs`

2.7 Architecture 3-tiers et Responsabilités

2.7.1 Vue d'ensemble (description textuelle)

L'application suit une architecture **3-tiers classique** séparant les responsabilités :

— **Client Web** (navigateur) : Interface utilisateur interactive, validation côté client, gestion du state local (React Context/Zustand).
— **API REST** (NestJS sur Node.js) : Logique métier, authentification JWT, validation des entrées, orchestration des requêtes DB, rate limiting, logs structurés.
— **Base de données** (PostgreSQL) : Persistance des données, contraintes d'intégrité référentielle, index de performance, sauvegardes automatisées.

Déploiement : Conteneurs Docker orchestrés via Docker Compose (v1) ou Kubernetes (v2). CI/CD via GitHub Actions (tests automatisés, build, déploiement sur Vercel/OVH).

Observabilité minimale (v1) : Journaux applicatifs centralisés (Winston → fichiers), métriques de performance (Lighthouse CI), monitoring uptime (UptimeRobot).

2.7.2 Architecture logique (responsabilités détaillées)

Couche Présentation (Front-end)

— **Pages** : Routage App Router Next.js (`app/auth/page.tsx`, `app/dashboard/page.tsx`, etc.)
— **Composants** : Découpage atomique (Atomic Design) : atomes (boutons, inputs), molécules (formulaires), organismes (cartes dashboard)
— **Validation UI** : React Hook Form + Zod (schémas partagés avec back-end via package commun)
— **State management** : Zustand pour état global (utilisateur connecté, filtres actifs), Context API pour thèmes/i18n
— **Accessibilité** : Attributs ARIA, navigation clavier, contrastes WCAG 2.1 AA, tests avec axe-core
— **Internationalisation (v2)** : next-i18next (français/anglais), détection automatique locale navigateur

Couche Métier (Back-end)

— **Modules NestJS** : Architecture modulaire avec injection de dépendances (DI) :
  — `AuthModule` : Inscription, connexion, JWT strategy (Passport), refresh tokens (v2)

  - `UsersModule` : CRUD profils, gestion rôles, suppression RGPD
  - `TransactionsModule` : CRUD transactions, filtres, agrégations (totaux par catégorie)
  - `BudgetsModule` : CRUD enveloppes, calcul "reste à vivre", historiques
  - `CategoriesModule` : CRUD catégories (globales + personnalisées)
  - `ImportModule` : Parsing CSV, validation, détection doublons, rapports
  - `ExportModule` : Génération CSV/JSON, compression ZIP
- **Services métier** : Logique de calcul isolée (testable unitairement) :
  - `BudgetCalculatorService` : Calcul reste à vivre, pourcentages consommation enveloppes
  - `TransactionAggregatorService` : Agrégations (sommes par catégorie, évolutions mensuelles)
- **Guards & Interceptors** : Contrôle d'accès (JWT guard, Role guard), transformation réponses, gestion erreurs globales
- **Rate limiting** : Throttler NestJS (10 req/sec par IP, 100 req/min par utilisateur)
- **Logs** : Winston avec niveaux (error, warn, info, debug), rotation quotidienne, format JSON pour parsing

Couche Données (Base de données)

- **Schémas Prisma** : Modèles typés (User, Transaction, Budget, Category, AuditLog)
- **Relations** : Foreign keys avec `ON DELETE CASCADE` (suppression utilisateur → purge transactions)
- **Index de performance** :
  - `CREATE INDEX idx_transactions_user_date ON transactions(user_id, date DESC)`
  - `CREATE INDEX idx_transactions_category ON transactions(category_id)`
  - `CREATE INDEX idx_budgets_user ON budgets(user_id)`
- **Migrations versionnées** : Prisma Migrate (`prisma migrate dev`, `prisma migrate deploy`)
- **Seeds** : Données de test (15 catégories prédéfinies, 2 utilisateurs démo, 50 transactions)
- **Sauvegardes** : pg_dump quotidien (rétention 30j), restauration testée mensuellement

2.8 Choix Technologiques et Alternatives

2.8.1 Tableau comparatif des choix

Composant	Choix	Alternative	Raison du choix
Back-end	NestJS	Express.js	Structure modulaire (DI), décorateurs, tests facilités, support TypeScript natif, CLI puissant
Front-end	Next.js	React pur (Vite)	Routage/SSR intégrés, optimisation images, App Router (RSC), SEO, performances, DX supérieure
Base de données	PostgreSQL	MongoDB	Données transactionnelles relationnelles (ACID), agrégations SQL complexes, contraintes d'intégrité, maturité
ORM	Prisma	TypeORM	Schémas typés auto-générés, migrations déclaratives, Prisma Studio (GUI), performances (requêtes optimisées)
Cache/Queue (v2)	Redis	RabbitMQ	Simplicité d'usage, TTL natifs, incréments atomiques (compteurs), pub/sub pour notifications
Graphiques	Recharts	Chart.js	Intégration React fluide (composants déclaratifs), personnalisation facile, bundle size raisonnable
Validation	Zod	Yup, Joi	Inférence TypeScript automatique, schémas composables, performances (parsing rapide), partage front/back
State management	Zustand	Redux Toolkit	Simplicité (sans boilerplate), devtools intégrés, API minimale, performances (re-renders optimisés)
Tests E2E	Playwright	Cypress	Support multi-navigateurs natif (Chrome, Firefox, Safari), parallélisation, debugging puissant
CI/CD	GitHub Actions	GitLab CI	Gratuit pour repos publics, workflows YAML simples, marketplace d'actions, intégration GitHub native
Hébergement front	Vercel	Netlify	Optimisations Next.js natives, edge functions, analytics intégrés, DX (déploiement auto sur push)
Hébergement back/DB	OVH (VPS)	AWS, Google Cloud	Coûts maîtrisés (20€/mois), souveraineté données (RGPD EU), support français

2.8.2 Justifications détaillées (choix critiques)

PostgreSQL vs MongoDB

Pourquoi PostgreSQL ?

— **Données transactionnelles** : Les transactions financières nécessitent des garanties ACID (Atomicity, Consistency, Isolation, Durability). PostgreSQL excelle sur ces propriétés.

— **Relations complexes** : Modèle relationnel adapté aux liens User ↔ Transaction ↔ Category ↔ Budget.

— **Agrégations SQL** : Calculs complexes (sommes par catégorie, évolutions temporelles) plus performants en SQL qu'en aggregation pipeline MongoDB.

— **Intégrité référentielle** : Foreign keys avec `ON DELETE CASCADE` garantissent la cohérence (suppression utilisateur → purge automatique transactions).

MongoDB écarté car :

— Denormalisation nécessaire (duplication données) pour performances → risque incohérences

— Transactions multi-documents complexes (support depuis v4.0 mais moins mature)

— Pas de garantie ACID stricte par défaut

NestJS vs Express.js

Pourquoi NestJS ?

— **Architecture modulaire** : Découpage clair par domaines métier (AuthModule, TransactionsModule, etc.), réutilisabilité, maintenabilité
— **Injection de dépendances (DI)** : Testabilité maximale (mocks faciles), couplage faible entre composants
— **Décorateurs TypeScript** : Code déclaratif lisible (`@Get()`, `@UseGuards()`, `@Body()`)
— **Écosystème riche** : Passport (auth), Swagger (doc auto), class-validator (validation), throttler (rate limiting)
— **Standards** : Conformité aux design patterns (Repository, Service, Controller), onboarding facilité pour nouveaux développeurs

Express.js écarté car :

— Minimaliste → nécessite configuration manuelle (routing, validation, error handling)
— Pas de structure imposée → divergences architecture entre développeurs
— Tests plus complexes (moins d'abstractions pour DI/mocks)

Next.js vs React pur (Vite)

Pourquoi Next.js ?

— **Routage intégré** : App Router avec layouts, loading states, error boundaries (pas de react-router à configurer)
— **SSR/SSG** : Server-Side Rendering pour SEO (page d'accueil publique), Static Site Generation pour performances
— **Optimisations automatiques** : Images (next/image), fonts (next/font), code splitting, tree shaking
— **API Routes** : Endpoints API co-localisés (utile pour webhooks, génération PDFs en v2)
— **DX supérieure** : Fast Refresh, TypeScript out-of-the-box, devtools intégrés

React pur (Vite) écarté car :

— Configuration manuelle : routage, bundling, optimisations images
— Pas de SSR natif (nécessite framework custom ou Remix/Astro)
— SEO limité (CSR uniquement sauf configuration complexe)

2.9 Contraintes, Risques et Mitigations

2.9.1 Matrice des risques

Risque/Contrainte	Impact	Prob.	Mitigation
Retard développement backend	Moyen	Élevée	Sprints courts (2 sem.), feature flags (déploiement progressif), priorité P1 stricte, buffer 20 % planning
Failles sécurité (JWT/SQL injection)	Élevé	Moyen	Revue de code obligatoire (PR), tests automatisés (Snyk, npm audit), linters (ESLint security rules), checklist OWASP Top 10
Charge import CSV (timeout)	Moyen	Moyen	Traitement par batchs (500 lignes), transactions DB atomiques, worker asynchrone (Bull/Redis en v2), index DB optimisés
Complexité responsive (breakpoints)	Moyen	Moyen	Design mobile-first (Tailwind), tests sur appareils réels (BrowserStack), composants adaptatifs (useMediaQuery)
Dépendance hébergeur cloud	Faible	Faible	Abstraction infra (Docker), variables env (12-factor app), sauvegardes externes (S3/Backblaze), documentation déploiement
Dépassement budget hébergement	Faible	Moyen	Monitoring coûts (alerts OVH), optimisation requêtes (cache Redis v2), rate limiting, scaling horizontal différé (v2)
Perte de données (crash DB)	Élevé	Faible	Sauvegardes quotidiennes (pg_dump), réplication (read replica en v2), tests de restauration mensuels, transactions ACID
Conformité RGPD non respectée	Élevé	Faible	Checklist RGPD exhaustive, endpoints dédiés (export/suppression), logs audit, revue juridique (DPO consulté), doc politique confidentialité

2.9.2 Plan de contingence (risques critiques)

Failles de sécurité découvertes en production

Scénario : Vulnérabilité critique détectée (ex : JWT mal signé, SQL injection possible).

Plan d'action :

1. **Immédiat (H+0)** : Mise hors ligne temporaire de l'API (page maintenance), notification utilisateurs par email
2. **H+2** : Patch de sécurité développé et testé sur environnement staging
3. **H+4** : Déploiement du patch en production, tests de non-régression
4. **H+6** : Remise en ligne, audit de sécurité complet (Snyk, SonarQube), communication transparente (changelog)
5. **J+7** : Post-mortem rédigé, mesures préventives ajoutées (nouveaux tests, revue processus)

Perte totale de la base de données

Scénario : Crash serveur OVH, corruption base PostgreSQL irréparable.

Plan d'action :

1. **Immédiat** : Basculement sur dernière sauvegarde pg_dump (< 24h de perte)
2. **H+1** : Restauration sur nouveau serveur (VPS de secours pré-configuré)
3. **H+2** : Vérification intégrité données (checksums, requêtes de validation)
4. **H+4** : Remise en production, notification utilisateurs (perte potentielle dernières 24h)

5. **J+1** : Migration vers réplication maître-esclave (éviter scénario futur)

2.10 Scénarios d'Usage Prioritaires (P1)

2.10.1 Scénario 1 : Création du compte et première connexion

Préconditions

— Utilisateur non inscrit, navigateur moderne (Chrome 120+, Firefox 120+, Safari 17+)
— Accès internet stable
— Email valide et accessible

Étapes du scénario

1. Utilisateur accède à `https://mysmartbudget.com/auth/register`
2. Remplit formulaire : nom, prénom, email, mot de passe (12+ caractères, 1 maj, 1 min, 1 chiffre, 1 spécial), confirmation mot de passe
3. Coche case "J'accepte la politique de confidentialité" (consentement RGPD)
4. Clique sur "Créer mon compte"
5. Système valide les données, hache le mot de passe (bcrypt cost 12), crée l'utilisateur en base
6. Redirection automatique vers `/dashboard` avec token JWT (expiration 24h)
7. Dashboard affiche message de bienvenue : "Bienvenue [Prénom] ! Ajoutez votre première transaction"

Postconditions

— Utilisateur authentifié avec session active (token JWT valide)
— Profil créé en base avec rôle "Utilisateur"
— 15 catégories prédéfinies associées automatiquement au compte
— Log audit : inscription enregistrée (IP, timestamp)

Variantes et exceptions

— **Email déjà utilisé** : Message d'erreur "Cet email est déjà associé à un compte. Connectez-vous ou réinitialisez votre mot de passe."
— **Mot de passe faible** : Validation temps réel avec indicateur de force (rouge/orange/vert), blocage soumission si critères non respectés
— **Erreur réseau** : Message "Impossible de créer le compte. Vérifiez votre connexion et réessayez." + retry automatique après 3s

Critères de succès

— Temps total du parcours : < 2 min (P95)
— Taux de réussite : $\geq 95\,\%$ (hors erreurs utilisateur : email doublon, mot de passe faible)
— Taux d'activation : $\geq 80\,\%$ (compte créé $\rightarrow 1^{re}$ connexion dans 7j)

2.10.2 Scénario 2 : Ajout d'une transaction et affectation à une enveloppe

Préconditions

— Utilisateur connecté
— Au moins 1 enveloppe créée (ex : "Alimentation" avec plafond 400€/mois)
— Catégorie "Courses" existante

Étapes du scénario

1. Utilisateur navigue vers `/transactions`
2. Clique sur bouton "+ Nouvelle transaction"
3. Remplit formulaire modal :
   — Date : aujourd'hui (pré-rempli, modifiable)
   — Montant : 35.50€
   — Type : Dépense (sélecteur radio)
   — Catégorie : "Courses" (dropdown)
   — Enveloppe : "Alimentation" (dropdown filtré par catégorie, optionnel)
   — Libellé : "Supermarché Carrefour" (optionnel)
4. Clique sur "Enregistrer"
5. Système valide les données, enregistre la transaction en base (< 500ms)
6. Mise à jour temps réel :
   — Liste des transactions : nouvelle ligne ajoutée en haut (tri date DESC)
   — Enveloppe "Alimentation" : consommation passe de 150€ à 185.50€ (46 % du plafond, indicateur vert)
   — Dashboard (si ouvert) : KPI "Dépenses du mois" incrémenté de 35.50€
7. Message de succès (toast vert) : "Transaction ajoutée avec succès"

Postconditions

— Transaction persistée en base avec `created_at = NOW()`
— Enveloppe "Alimentation" mise à jour (consommation, pourcentage)
— Historique modifiable (bouton "Éditer" visible sur la ligne)

Variantes et exceptions

— **Montant négatif ou invalide** : Validation temps réel, champ bordure rouge, message "Le montant doit être positif"
— **Date future** : Avertissement "Êtes-vous sûr ? La date est dans le futur" + confirmation requise
— **Enveloppe dépassée** : Si ajout dépasse 100 % du plafond, modal d'alerte : "Attention, cette transaction fait dépasser votre enveloppe Alimentation de 15€. Continuer ?" (boutons Oui/Non)
— **Erreur serveur (500)** : Message "Impossible d'enregistrer la transaction. Réessayez." + retry automatique

Critères de succès

— Temps ajout transaction : < 30s (P95, incluant saisie utilisateur)
— Latence API `POST /transactions` : < 500ms (P95)
— Taux d'erreur validation : $< 5\,\%$ (erreurs utilisateur normales : montant invalide, etc.)
— Cohérence données : 0 écart entre montant saisi et montant enregistré

2.10.3 Scénario 3 : Consultation du dashboard et interprétation des indicateurs

Préconditions

— Utilisateur connecté avec historique de transactions (minimum 10 transactions sur 2 mois)
— Au moins 2 enveloppes créées

Étapes du scénario

1. Utilisateur navigue vers `/dashboard`
2. Chargement de la page (< 2s P95) :
   — Affichage immédiat du skeleton (cartes grises pulsantes)
   — Requêtes API parallèles : `GET /transactions/summary`, `GET /budgets/summary`
   — Rendu progressif : KPIs → graphiques
3. KPIs affichés (cartes en haut de page) :
   — Solde total : 1 245€ (icône porte-monnaie, couleur verte si positif)
   — Dépenses du mois : 890€ (icône flèche bas, couleur rouge)
   — Revenus du mois : 2 100€ (icône flèche haut, couleur verte)
   — Reste à vivre : 355€ (icône tirelire, couleur orange si $< 20\,\%$ revenus)
4. Graphiques interactifs (section centrale) :
   — Camembert : Répartition dépenses par catégorie (top 10, "Autres" agrégé), survol affiche montant exact + pourcentage

   - Courbe : Évolution dépenses/revenus sur 12 derniers mois (2 lignes, légende interactive), zoom/pan désactivé par défaut
5. Filtres temporels (haut de page) :
   - Boutons radio : "Mois en cours" (sélectionné par défaut), "Trimestre", "Année", "Personnalisé"
   - Si "Personnalisé" : datepickers (début/fin), bouton "Appliquer"
6. Utilisateur clique sur "Trimestre" :
   - Rechargement des données (< 1s), graphiques mis à jour
   - KPIs recalculés sur Q4 2025 (octobre–décembre)
7. Section enveloppes (bas de page) :
   - Liste des enveloppes avec barres de progression :
     - "Alimentation" : 385€/400€ (96 %, barre orange, attention)
     - "Loisirs" : 120€/300€ (40 %, barre verte, valider)
     - "Transport" : 210€/200€ (105 %, barre rouge, refusé, texte "Dépassement de 10€")

Postconditions

- Utilisateur a une vision claire de sa situation financière
- Identification visuelle immédiate des enveloppes problématiques (orange/rouge)
- Données consultées sans modification (lecture seule)

Variantes et exceptions

- **Aucune transaction** : Message centré "Vous n'avez pas encore de transactions. Ajoutez-en une pour voir vos statistiques." + bouton CTA "+ Ajouter une transaction"
- **Erreur chargement graphiques** : Fallback texte "Impossible de charger les graphiques. Réessayez." + bouton reload
- **Période personnalisée invalide** : Si date début > date fin, message d'erreur "La date de début doit être antérieure à la date de fin"

Critères de succès

- Time-to-Render (TTR) : < 2s (P95, mesure Lighthouse)
- Taux d'usage hebdomadaire : > 60 % des utilisateurs actifs consultent le dashboard au moins 1 fois/semaine
- Compréhension des indicateurs : Score SUS $\geq$ 75 (question "Je comprends facilement ma situation financière")

2.11 Plan de Validation Utilisateur

2.11.1 Méthodologie de test

Échantillon de testeurs

Profil	Critères de sélection	Objectifs de test
Étudiante (Léa)	18–25 ans, budget serré (< 1000€/mois), usage smartphone prioritaire	Validation scénario 1 et 2, ergonomie mobile, import CSV
Père de famille (Marc)	35–45 ans, revenus 3000–5000€/mois, gestion enveloppes multiples	Validation scénario 2 et 3, dashboard complexe, export CSV
Senior	60+ ans, faible appétence numérique, besoin simplicité	Accessibilité (taille texte, contrastes), navigation intuitive

Nombre de testeurs : 3–5 personnes (1–2 par profil), recrutement via réseau personnel, associations locales, offre de compensation symbolique (20€ chèque-cadeau Amazon).

Protocole de test

1. **Session modérée individuelle (45 min)** :
   — Introduction (5 min) : Contexte projet, consentement enregistrement (audio/écran), rappel liberté d'abandon
   — Think-aloud (30 min) : Testeur verbalise ses actions/pensées pendant 3 scénarios P1
   — Débriefing (10 min) : Questions ouvertes ("Qu'avez-vous aimé/détesté ?", "Recommanderiez-vous l'appli ?")
2. **Questionnaire SUS (5 min)** : 10 questions échelle Likert (1–5), calcul score global (0–100)
3. **Mesures objectives** :
   — Temps d'accomplissement par tâche (chronomètre)
   — Nombre d'erreurs (clics incorrects, chemins abandonnés)
   — Taux de réussite (tâche complétée sans aide / avec aide / échouée)

2.11.2 Critères de validation

Métrique	**Seuil acceptable**	**Seuil cible**
Score SUS global	$\geq$ 68 (moyenne industrie)	$\geq$ 75 (bon)
Temps scénario 1 (création compte)	$<$ 3 min (P95)	$<$ 2 min
Temps scénario 2 (ajout transaction)	$<$ 1 min (P95)	$<$ 30s
Temps scénario 3 (consultation dashboard)	$<$ 2 min (P95)	$<$ 1 min
Taux de réussite sans aide	$\geq$ 80 %	$\geq$ 90 %
Taux d'erreur (clics incorrects)	$<$ 10 %	$<$ 5 %
Recommandation (NPS)	Score $\geq$ 0 (neutre)	Score $\geq$ 50 (excellent)

2.11.3 Traçabilité et itération

— **Documentation** : Compte-rendu par session (verbatims, captures écran, enregistrements), synthèse globale (problèmes récurrents, recommandations)

— **Priorisation problèmes** : Matrice sévérité (bloquant/majeur/mineur) $\times$ fréquence (tous/majorité/rare)

— **Issues GitHub** : 1 issue par problème identifié, étiquette `usability`, assignation sprint suivant si priorité élevée

— **Retests** : Après corrections majeures, nouvelle session abrégée (15 min, focus problèmes corrigés) avec 1–2 testeurs

2.12 Mesure de l'Impact par Fonctionnalité (KPIs)

2.12.1 Tableau des KPIs produit

Feature	Indicateur	Objectif v1.0 (3 mois post-lancement)
Auth	Taux d'activation (compte créé → 1re connexion 7j)	≥ 80 %
	Taux de rétention J+30	≥ 40 %
	Temps moyen inscription	< 2 min (P95)
Transactions	Latence création P95	< 500ms
	# transactions/semaine par utilisateur actif	≥ 5 (moyenne)
	Taux d'édition (transactions modifiées / créées)	< 10 % (qualité saisie initiale)
Enveloppes	% utilisateurs avec ≥ 3 enveloppes actives	≥ 60 %
	% enveloppes dépassées/mois (moyenne utilisateurs)	< 20 % (efficacité budgétaire)
	Temps moyen création enveloppe	< 1 min
Import CSV	Taux d'import réussi (fichiers valides)	> 95 %
	Temps import moyen (10k lignes)	< 20s
	% utilisateurs ayant importé ≥ 1 fichier	≥ 30 % (adoption fonctionnalité)
Dashboard	Time-to-Render (TTR) P95	< 2s
	Taux d'usage hebdomadaire (visites dashboard/semaine)	> 60 %
	Taux de rebond (quitte immédiatement)	< 30 %
	Durée session moyenne	> 3 min (engagement)
Export	% utilisateurs ayant exporté données (trimestre 1)	≥ 20 %
	Temps génération export (10k transactions)	< 5s

2.12.2 Instrumentation technique

— **Analytics** : Google Analytics 4 (GA4) + événements personnalisés :
  — `user_signup`, `user_login`, `transaction_created`, `envelope_created`, `csv_imported`, `data_exported`
— **Performance** : Web Vitals (LCP, FID, CLS) via Next.js Analytics, alertes si dégradation > 10 %
— **Erreurs** : Sentry (monitoring erreurs front/back), agrégation par type, notification Slack si erreur critique
— **Logs métier** : Winston (back-end), logs structurés JSON :
  — `{action: "transaction_created", userId, amount, category, timestamp}`
— **Dashboard interne** : Metabase (BI tool open-source) connecté à PostgreSQL, requêtes SQL pour KPIs temps réel

2.13 Définition du MVP (Rappel Synthétique)

Le **MVP v1.0** comprend les 5 modules essentiels suivants :

1. **Authentification & Sessions** : Inscription, connexion JWT, réinitialisation mot de passe, suppression compte (RGPD)
2. **Transactions Manuelles** : CRUD complet, catégorisation (15 catégories + perso), filtres/recherche, pagination
3. **Enveloppes Budgétaires** : CRUD enveloppes, plafonds mensuels, calcul "reste à vivre", indicateurs visuels (vert/orange/rouge), historique 12 mois

4. **Dashboard Synthétique** : 4 KPIs (solde, dépenses, revenus, reste à vivre), 2 graphiques (camembert catégories, courbe évolution), filtres temporels (mois/trim./année)
5. **Import/Export CSV** : Import transactions (format gabarit, validation, doublons, rapport), export CSV/JSON (portabilité RGPD)

Chaque module est couvert par :

— Critères d'acceptation mesurables (DoD)

— Tests automatisés (unitaires + E2E, couverture $\geq 60\,\%$)

— Documentation technique (API Swagger, README)

2.14 Roadmap Jalonnée (Milestones et Dates Estimées)

2.14.1 Planning détaillé par version

Version	Période cible	Jalons / Livrables principaux
v1.0 (MVP)	Nov. 2025 – Jan. 2026	**M0 (Nov. 2025)** : Schémas Prisma finalisés (MCD/MLD/MPD), API Auth opérationnelle (JWT, bcrypt), CI/CD GitHub Actions configuré **M1 (Déc. 2025)** : Modules Transactions & Enveloppes complets (CRUD, tests), Import CSV fonctionnel (validation, rapport) **M2 (Jan. 2026)** : Dashboard avec graphiques Recharts, tests E2E Playwright (scénarios P1), déploiement staging (Vercel + OVH), tests utilisateurs (3–5 personnes), corrections bugs critiques
v1.1	Février 2026	Import CSV robuste (encodages multiples, formats dates flexibles), amélioration UX (feedback utilisateurs), premiers KPIs en prod (GA4, Metabase), optimisation performances (lazy loading, pagination)
v2.0	Avril 2026	Agrégation bancaire POC (Budget Insight API, 3 banques pilotes), notifications push (dépassement enveloppes, objectifs atteints), cache Redis (sessions, compteurs), queue Bull (imports asynchrones), optimisation perfs (P95 dashboard $<$ 1s), tests charge (Gatling, 1000 users concurrents)

2.14.2 Dépendances critiques et risques planning

— **M0** $\rightarrow$ **M1** : Blocage si schémas DB non finalisés (refactoring coûteux ensuite). *Mitigation* : Validation schémas avec pair programming, revue architecture J+7.

— **M1** $\rightarrow$ **M2** : Import CSV complexe (formats variés, doublons). *Mitigation* : Spécifications détaillées dès M0, tests avec fichiers réels (10 banques françaises).

— **M2** $\rightarrow$ **Livraison** : Tests utilisateurs révèlent bugs majeurs. *Mitigation* : Buffer 2 semaines post-M2, feature flags (rollback rapide si nécessaire).

2.15 Traçabilité et Responsabilités

2.15.1 Gestion du backlog (GitHub Projects)

— **Organisation** : Backlog centralisé dans GitHub Projects (vue Kanban), colonnes : Backlog, Todo, In Progress, In Review, Done

— **Issues** : 1 issue par user story ou bug, template standardisé :

  — Titre : `[US-XX] Description courte` ou `[BUG] Description`

  - Corps : User story (As a..., I want..., So that...), critères d'acceptation (liste à cocher), effort estimé (points), priorité (P1/P2/P3)
  - Étiquettes : `P1`/`P2`/`P3`, `bug`, `usability`, `security`, `performance`, module (`auth`, `transactions`, etc.)
- **Milestones** : Alignés à la roadmap (M0, M1, M2, v1.1, v2.0), échéances fermes,
- **Assignation** : Toutes les issues assignées (porteur de projet en v1, équipe en v2+)

2.15.2 Definition of Done (DoD) globale

Une fonctionnalité est considérée **terminée** (Done) si et seulement si :

1. **Critères d'acceptation respectés** : Tous les éléments de la checklist user story validés
2. **Tests passants** :
   - Tests unitaires : couverture $\geq 60\,\%$ du code ajouté/modifié (Jest)
   - Tests E2E : scénario principal fonctionnel (Playwright)
   - Tests manuels : vérification visuelle sur 3 navigateurs (Chrome, Firefox, Safari) et 2 tailles (mobile 375px, desktop 1920px)
3. **Documentation mise à jour** :
   - API : Swagger/OpenAPI (décorateurs NestJS auto-générés)
   - Code : Commentaires JSDoc pour fonctions complexes, README module si nécessaire
   - Utilisateur : Section FAQ/Help Center mise à jour (si nouvelle fonctionnalité majeure)
4. **Revue de code effectuée** :
   - Pull Request (PR) sur GitHub avec description détaillée (contexte, changements, tests)
   - Revue par pair (ou auto-revue avec checklist si solo) : lisibilité, sécurité, performances
   - CI/CD validé : linters (ESLint, Prettier), tests automatisés, build réussi
5. **Déployé sur environnement staging** : Validation en conditions réelles (base de données de test, données anonymisées)
6. **Pas de régression** : Tests de non-régression (automatisés + manuels) sur fonctionnalités existantes

2.15.3 Responsabilités (RACI)

Activité	R	A	C	I
Conception architecture	Porteur	Porteur	Jury (optionnel)	—
Développement front-end	Porteur	Porteur	—	—
Développement back-end	Porteur	Porteur	—	—
Modélisation base de données	Porteur	Porteur	Jury (optionnel)	—
Tests utilisateurs	Porteur	Porteur	Testeurs	—
Rédaction documentation	Porteur	Porteur	—	Jury
Déploiement production	Porteur	Porteur	—	Hébergeur
Validation jalons	Porteur	Jury CDA	Porteur	—

Légende RACI :
— **R (Responsible)** : Réalise l'activité
— **A (Accountable)** : Approuve/valide l'activité (responsable final)
— **C (Consulted)** : Consulté pour avis/expertise
— **I (Informed)** : Informé des résultats

Chapitre 3

Conception et modélisation du projet App Budget V3

3.1 Introduction

Après la phase de planification et d'organisation présentée dans le chapitre précédent, la conception du projet **App Budget V3** a constitué une étape essentielle avant le développement. Cette phase a permis de structurer le projet, d'identifier les entités principales, de clarifier les interactions entre les différents composants, et de garantir la cohérence technique globale du système.

La conception n'est pas seulement un travail préparatoire : elle représente le fondement sur lequel repose l'ensemble du développement. C'est à ce stade que sont définies les bases logiques, fonctionnelles et techniques de l'application, afin d'assurer une implémentation fluide, évolutive et maintenable.

Pour ce faire, j'ai utilisé la méthode de modélisation **UML (Unified Modeling Language)**, permettant de représenter graphiquement les structures et comportements du système. Les différents diagrammes UML m'ont aidé à mieux visualiser les interactions entre les utilisateurs, les modules fonctionnels et la base de données, tout en favorisant une communication claire et cohérente.

3.2 Objectifs de la conception UML

La modélisation UML avait plusieurs objectifs concrets dans le cadre du développement d'App Budget V3 :

— **Structurer les données et les fonctionnalités** avant la mise en œuvre technique.
— **Anticiper les interactions et dépendances** entre les composants du système.
— **Garantir la cohérence entre le besoin utilisateur et la logique du code.**
— **Facaciliter la maintenance et les évolutions futures** de l'application.

3.3 Présentation du diagramme UML global

Le diagramme UML global ci-dessous illustre les principales entités de l'application, leurs relations et leurs rôles dans la gestion budgétaire. Il met en évidence les modules clés : gestion des utilisateurs, budgets, transactions et alertes intelligentes.

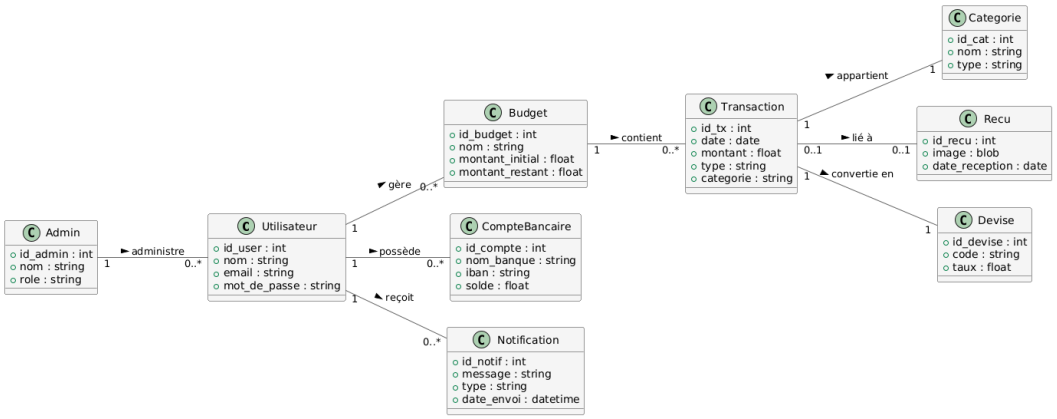


FIGURE 3.1 – Diagramme UML global du projet App Budget V3

3.4 Description détaillée du modèle conceptuel

1. Module utilisateur

— **User :** email, mot de passe chiffré (bcrypt), rôle, date de création.
— **Profile :** devise, langue, catégories favorites, seuils personnalisés.
— **Session :** connexion sécurisée via tokens JWT, avec rafraîchissement automatique.

2. Module gestion financière

— **Transaction :** chaque opération financière (revenu/dépense) avec montant, date, catégorie, commentaire.
— **Category :** regroupe les transactions similaires (alimentation, logement, etc.).
— **Budget :** fixe un montant maximum sur une période (mensuelle, hebdomadaire, personnalisée).

3. Module suivi et alertes

— **Alert :** déclenchement automatique à 70%, 90% et 100% d'un budget.
— **Dashboard :** visualisation graphique des dépenses, historiques et soldes.
— **Report :** génération et export de rapports PDF/CSV.

3.5 Architecture technique de l'application

Le projet repose sur une architecture **MVC (Model-View-Controller)** garantissant la séparation entre données, logique métier et présentation.

— **Front-end :** React.js + Tailwind CSS.
— **Back-end :** Node.js + Express.js.
— **Base de données :** MongoDB pour sa flexibilité.
— **Sécurité :** Authentification JWT, chiffrement bcrypt, gestion stricte des rôles.

— **Déploiement :** Docker + hébergement cloud (Render/AWS).

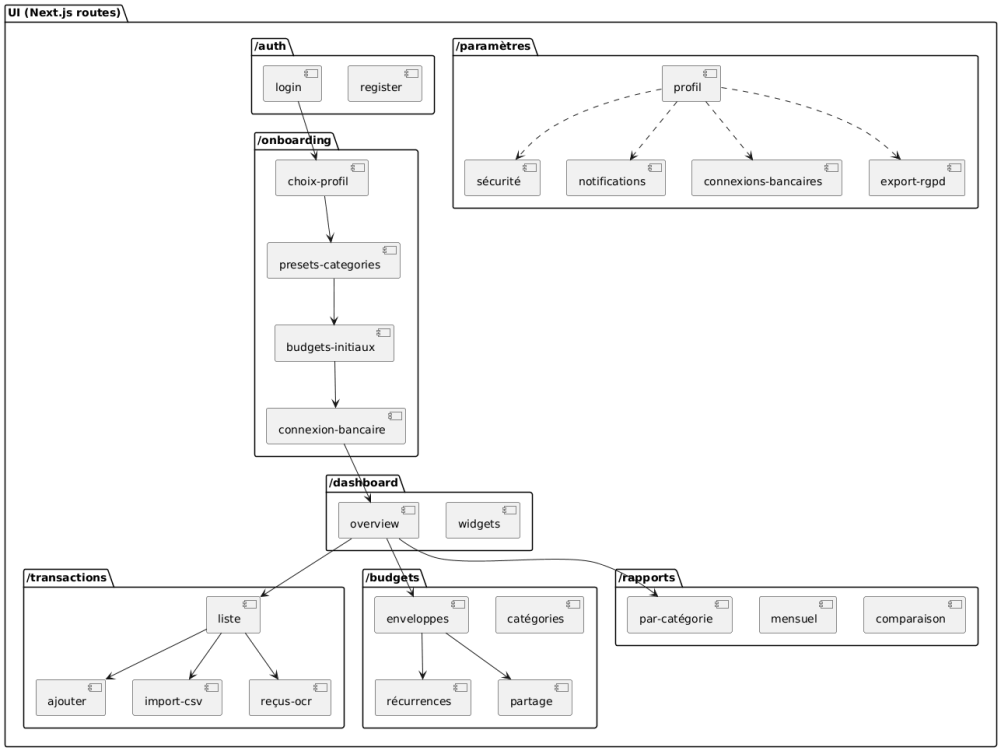


FIGURE 3.2 – Architecture technique du projet App Budget V3

3.6 Lien entre conception et développement

La modélisation UML a directement orienté la structure technique :

— Les entités UML ont été traduites en modèles **Mongoose**.

— Les relations ont guidé les schémas MongoDB.

— Les cas d'utilisation ont défini les **endpoints REST API**.

— Le dashboard du front-end repose sur ces endpoints.

3.7 Méthodologie agile et conduite de projet

3.7.1 Choix de la méthode agile

J'ai retenu une méthode **Scrum adaptée** au projet individuel : itérative, incrémentale et centrée sur la valeur livrée. Les piliers de la méthode (planification, transparence, amélioration continue) sont conservés, avec des sprints de 2 semaines.

3.7.2 Rituels agiles

Chaque rituel a une **fréquence, une durée et un livrable associé** :

— **Daily Meeting (10 min chaque matin)** : suivi d'avancement, identification des blocages et ajustement du plan de la journée.
— **Sprint Planning (1h tous les 14 jours)** : définition du périmètre de sprint, priorisation des issues selon la méthode MoSCoW et création des tâches associées dans GitHub Projects.
— **Sprint Review (30 min)** : démonstration des fonctionnalités terminées, validation des critères d'acceptation et mise à jour du backlog produit.
— **Sprint Retrospective (30 min)** : identification des axes d'amélioration, mise à jour du backlog d'amélioration continue et synthèse documentée.

Chaque rituel est documenté dans le dossier `/docs/sprints` sur GitHub afin d'assurer une traçabilité complète du processus de développement.

Ces rituels garantissent un **suivi continu de l'avancement**, une **communication claire** et une **amélioration constante** du processus. Leur régularité (daily meetings quotidiens et sprints bi-hebdomadaires) permet de maintenir un rythme soutenu tout en conservant la flexibilité nécessaire pour ajuster les priorités au fil du projet.

3.7.3 Métriques de suivi et amélioration continue

Les métriques sont automatisées dans GitHub Actions et suivies dans un fichier `metrics.md`. Elles permettent de mesurer la progression, la stabilité et la qualité du code sur chaque sprint.

Indicateur	**Objectif**	**Fréquence**
Vélocité	≥ 8 issues clôturées / sprint	Bi-hebdo
Taux de réussite CI/CD	100% sur `main`	Hebdo
Taux d'erreurs post-merge	$< 5\%$	Review sprint
Temps de livraison moyen	< 3 jours / feature	Hebdo
Couverture de tests	78% sur Jest	Review sprint

Ces métriques sont automatiquement suivies via GitHub Actions et les rapports de projet. Elles couvrent les indicateurs clés exigés dans une démarche agile : la **vélocité** (issues fermées par sprint), le **temps moyen de livraison ou de pull request**, et le **taux d'erreurs corrigées ou bugs fermés**. Elles garantissent une vision quantitative, transparente et continue de la qualité du développement, permettant d'ajuster le pilotage et les priorités de sprint en fonction des résultats mesurés.

3.7.4 Roadmap GitHub et milestones

La roadmap du projet App Budget V3 est organisée en quatre jalons principaux, correspondant à des livrables mesurables et datés. Chaque jalon (milestone) est suivi et validé sur GitHub, avec des règles d'entrée (*Definition of Ready*) et de sortie (*Definition of Done*) clairement établies.

Jalon / Version	Date cible	Objectifs principaux
POC (Proof of Concept)	05/10/2025	Mise en place du backend Node.js, connexion MongoDB, première route API testée.
MVP (v1.0)	15/10/2025	Authentification JWT, gestion des utilisateurs, CRUD Transactions, Dashboard basique.
Bêta (v1.1)	30/10/2025	Import CSV, alertes budgétaires, tests unitaires automatisés, CI/CD GitHub Actions.
Version finale (v2.0)	15/11/2025	Agrégation bancaire, export PDF/CSV, optimisation performance, documentation finale.

Règles d'entrée (Definition of Ready) :

— Les issues sont clairement décrites avec critères d'acceptation.

— Les dépendances techniques sont identifiées et planifiées.

— Le design fonctionnel ou l'UML correspondant est validé.

Règles de sortie (Definition of Done) :

— Les fonctionnalités développées sont testées (unitaires + intégration).

— Le code est revu et validé via Pull Request.

— Les livrables sont documentés (README, changelog, métriques).

— Le déploiement Docker est fonctionnel et vérifié.

Chaque milestone correspond ainsi à un **état livrable validé**, garantissant une progression maîtrisée entre les étapes du projet (POC -> MVP -> Bêta -> Finale). Cette structuration permet une **traçabilité complète des versions** et facilite la démonstration d'avancement pour l'évaluation CDA.

3.7.5 Estimation du temps et burndown chart

Les estimations globales du projet ont d'abord été définies indépendamment des issues afin de dégager une vue d'ensemble du temps nécessaire par module. Elles reposent sur une vélocité moyenne de 7 issues clôturées par sprint, avec 1 Story Point $\approx$ 1 jour de travail.

Feature	Durée estimée	Story Points
Auth JWT	3 j	3
Transactions CRUD	4 j	4
Budgets	4 j	4
Dashboard	5 j	5
Import CSV	3 j	3
Tests / Docs	2 j	2
Total	**21 jours**	**21 SP**

Chaque issue du backlog GitHub est associée à un label de complexité : `SP:1`, `SP:2`, `SP:3`, etc., permettant de suivre la charge globale sur le board. Ces estimations sont affi-

chées directement dans le projet GitHub via le plugin **Projects v2** et la vue “Burndown Chart”.

FIGURE 3.3 – Burndown chart simplifié — suivi des story points consommés par sprint

Le burndown permet de visualiser la progression : la courbe descendante illustre les story points restants, garantissant une planification réaliste et une priorisation continue des tâches.

3.7.6 Lien entre stories, tests et intégration continue

Chaque **user story** est systématiquement reliée :
— à une **issue GitHub** (description fonctionnelle et technique),
— à un ou plusieurs **tests unitaires et fonctionnels**,
— et à un **pipeline CI GitHub Actions** exécuté automatiquement.

Les tests unitaires sont gérés avec **Jest** pour le back-end et **React Testing Library** pour le front-end. Ils valident la conformité du code aux critères d’acceptation définis dans chaque story.

User Story	**Test associé**	**CI/CD Validation**
#1 Authentification utilisateur	`auth.spec.js` (login 200, 401 invalid)	GitHub Action : `run-tests.yml`
#5 CRUD Transactions	`transaction.spec.js`	GitHub Action : `lint_and_test.yml`
#9 Dashboard utilisateur	`dashboard.test.jsx` (TTR < 2s)	GitHub Action : `build-and-test.yml`
#12 Import CSV	`import.spec.js` (doublons / format)	GitHub Action : `integration.yml`

Chaque pipeline CI vérifie le bon fonctionnement des tests, le linting, la couverture de code et le déploiement conditionnel sur l’environnement Docker. Ce lien direct entre stories, tests et intégration continue assure une **traçabilité complète** et une **qualité logicielle mesurable**.

3.7.7 Synthèse

Cette démarche agile garantit :
— Une **traçabilité complète** du flux de travail.
— Des **métriques mesurables et suivies automatiquement.**
— Une **qualité de code constante** via CI/CD.
— Une **gestion de projet professionnelle** conforme aux attentes CDA.

3.8 Tableau Kanban GitHub (exemple)

To Do	In Progress	Done
Auth JWT (#1)	CRUD Transactions (#5)	Init repo
Budgets (#6)	Dashboard (#9)	Docs README
Import CSV (#12)	Tests unitaires (#14)	Docker config
UI Dashboard	CI/CD GitHub Actions	Maquette Figma

3.9 Bénéfices de la conception UML et de la gestion agile

— **Cohérence :** conception UML alignée sur les objectifs métier et techniques.
— **Agilité :** pilotage structuré, rituels réguliers, métriques suivies.
— **Traçabilité :** lien complet entre stories, issues, PR et tests.
— **Professionnalisme :** roadmap claire, CI/CD, qualité mesurée.

3.10 Liens utiles

— UML Basics : https://www.uml-diagrams.org/
— MVC Pattern : https://developer.mozilla.org/en-US/docs/Glossary/MVC
— Node.js + Express : https://expressjs.com/
— MongoDB + Mongoose : https://mongoosejs.com/
— Docker : https://docs.docker.com/
— GitHub Projects : https://docs.github.com/en/issues/planning-and-tracking-with-projects

Chapitre 4

Conception fonctionnelle et technique

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

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

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

À FAIRE / À VÉRIFIER

Pourquoi la conception est-elle si importante ?

— **Évite les refactorisations coûteuses** : Une bonne conception évite de reprendre le code

— **Guide le développement** : Chaque développeur sait exactement quoi faire

— **Facilite les tests** : Les cas d'usage définis permettent de créer des tests pertinents

— **Réduit les risques** : Les problèmes sont identifiés avant le développement

— **Améliore la communication** : Tous les acteurs comprennent le système

Checklist de conception complète :

— ✓Diagrammes de cas d'usage (Use Cases) validés

— ✓Diagrammes de séquence pour les flux principaux

— ✓Modèle de données (MCD/MLD/MPD) défini

— ✓Architecture technique choisie et justifiée

— ✓User stories détaillées avec critères d'acceptation

— ✓Maquettes et wireframes validés

— ✓Charte graphique définie

— ✓Plan de tests établi

4.1 Use Cases et diagrammes UML

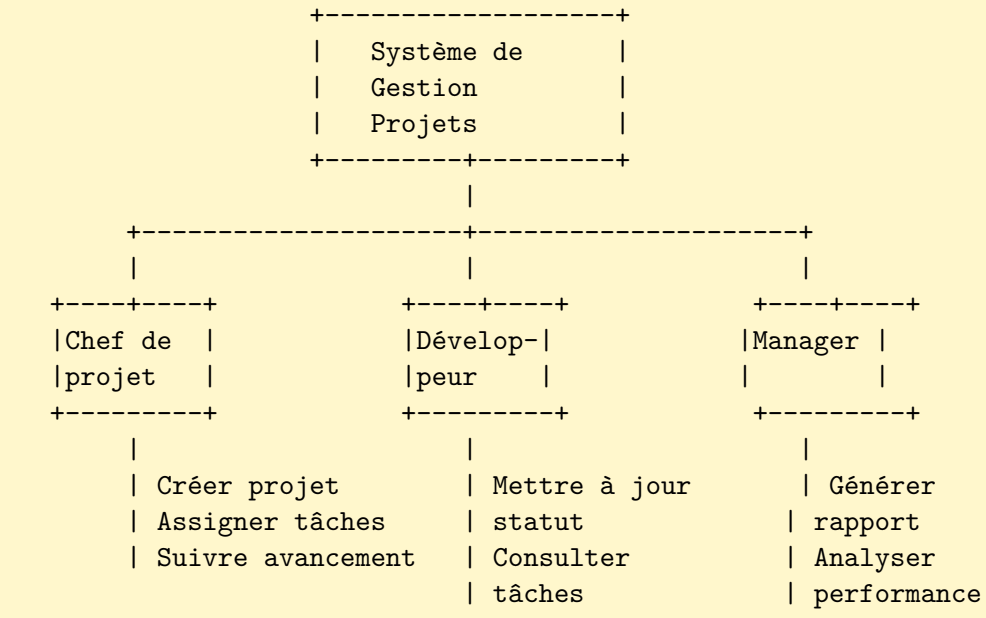
Les Use Cases modélisent les interactions entre les acteurs et le système pour identifier les fonctionnalités essentielles. Cette approche centrée utilisateur garantit que le système répond aux besoins métier réels. Les diagrammes UML facilitent la communication entre les équipes techniques et métier,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

La modélisation des cas d'usage permet d'identifier les flux principaux et alternatifs,

ainsi que les cas d'erreur à gérer. Cette analyse préalable guide la conception technique et les tests d'acceptation.

Exemple

Diagramme Use Case simplifié :



À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

4.2 Diagrammes de séquence

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

d'intégration.

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

Exemple

Diagramme de séquence - Création de projet :

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

Exemple de séquence avec gestion d'erreur :

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

4.3 Conception de l'interface graphique

La conception graphique s'appuie sur une charte graphique cohérente avec l'identité visuelle de l'entreprise. Le zoning et les wireframes définissent la structure des interfaces

avant le développement des maquettes haute fidélité. Cette approche progressive valide les choix UX et facilite l'implémentation front-end.

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

4.3.1 Zoning

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

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

Exemple

Zoning d'une page projet :

Header - Navigation principale		
Sidebar	**Contenu principal**	**Panel latéral**
• Menu navigation • Filtres • Recherche • Paramètres	• Titre du projet • Liste des tâches • Tableau de données • Pagination	• Actions rapides • Statistiques • Notifications • Aide contextuelle
Footer - Informations légales		

4.3.2 Wireframe

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

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

Exemple

Exemple de wireframe - Page de connexion :

LOGO DE L'APPLICATION
Connexion Email : [_____] Mot de passe : [_____] [Se connecter] Mot de passe oublié ?

4.3.3 Maquettage

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

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

Exemple

Évolution des maquettes :

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

4.3.4 Outils de conception et diagrammes

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

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

À FAIRE / À VÉRIFIER

Outils recommandés pour des diagrammes de qualité :

— **Draw.io (diagrams.net) :** Gratuit, intégré à GitHub, parfait pour les diagrammes UML

— **Lucidchart :** Professionnel, templates UML, collaboration en équipe

— **PlantUML :** Code-based, versioning Git, intégration LaTeX

— **Mermaid :** Intégré GitHub, syntaxe simple, diagrammes de flux

Conseils pour des diagrammes professionnels :

— Utilisez des couleurs cohérentes et une légende

— Respectez les conventions UML (acteurs, cas d'usage, relations)

— Gardez vos diagrammes simples et lisibles

— Versionnez vos diagrammes avec votre code

— Intégrez-les dans votre documentation GitHub

4.3.5 Charte graphique

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

Couleurs

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

Typographie

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

Logo

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

Exemple

Charte graphique :

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

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

4.4 Conception de base de données

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

PostgreSQL gère les données transactionnelles avec des contraintes strictes, tandis

que MongoDB stocke les logs et rapports avec une structure flexible. Cette architecture hybride optimise les performances selon le type de données.

4.4.1 MCD (Modèle Conceptuel de Données)

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

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

4.4.2 MLD (Modèle Logique de Données)

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

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

4.4.3 MPD (Modèle Physique de Données)

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

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

Exemple

MCD simplifié :

```
PROJET (id, nom, description, date_debut, date_fin)
    |
    | 1,n
    |
TACHE (id, titre, description, statut, priorite)
    |
    | n,1
    |
UTILISATEUR (id, email, nom, prenom, role)
```

Exemple de contraintes PostgreSQL :

```sql
-- Contraintes d'intégrité
ALTER TABLE taches ADD CONSTRAINT fk_tache_projet
    FOREIGN KEY (projet_id) REFERENCES projets(id)
    ON DELETE CASCADE;

-- Index pour optimiser les performances
CREATE INDEX idx_taches_statut ON taches(statut);
CREATE INDEX idx_taches_projet_statut ON taches(projet_id, statut);

-- Contrainte de validation
ALTER TABLE projets ADD CONSTRAINT chk_dates
    CHECK (date_fin > date_debut);
```

Exemple de document MongoDB :

```
{
  "_id": ObjectId("..."),
  "userId": "user123",
  "action": "task_created",
  "timestamp": ISODate("2025-01-15T10:30:00Z"),
  "metadata": {
    "projectId": "proj456",
    "taskId": "task789",
    "ipAddress": "192.168.1.100"
  }
}
```

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

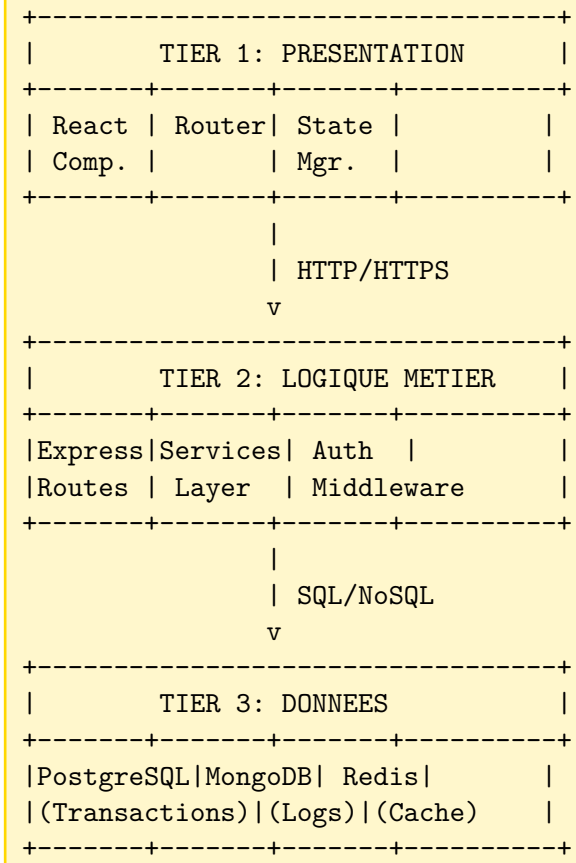
4.5 Architecture 3 tiers

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

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

Exemple

Schéma architecture 3 tiers :



Exemple de flux de données :

```
// Tier 1: Frontend (React)
const createProject = async (projectData) => {
  const response = await fetch('/api/projects', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(projectData)
  });
  return response.json();
};

// Tier 2: Backend (Node.js/Express)
app.post('/api/projects', authenticateUser, async (req, res) => {
  try {
    const project = await projectService.createProject(req.body);
    await auditService.logAction('project_created', req.user.id);
    res.status(201).json(project);
  } catch (error) {
    res.status(400).json({ error: error.message });
  }
});
```

À FAIRE / À VÉRIFIER

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

Contrôles Jury CDA

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

4.6 Liens utiles

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

Chapitre 5
Architecture 3 tiers

5.1 Architecture 3 tiers

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

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

5.1.1 Couche Présentation (Frontend)

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

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

Exemple

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

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

5.1.2 Couche Logique Métier (Backend)

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

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

Controller

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

Exemple

Exemple de contrôleur :

```
// 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

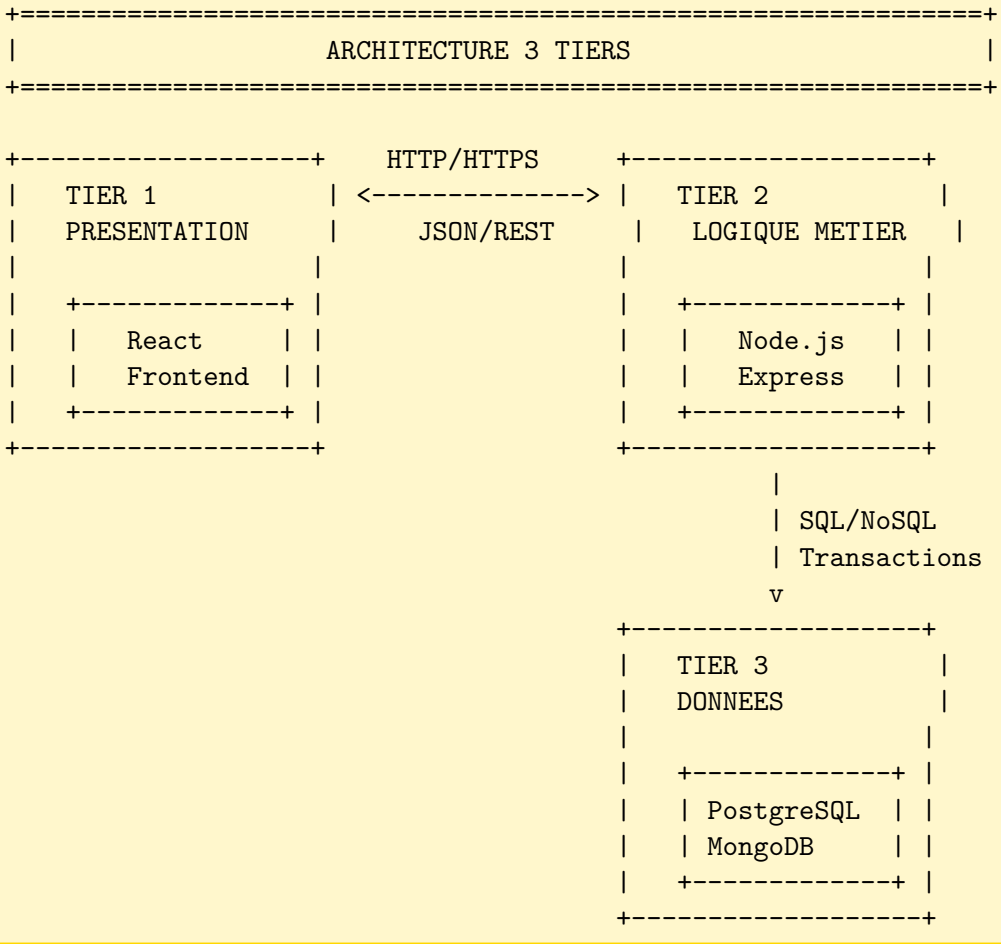
Dans cette sous-section, vous devez expliquer comment les différents tiers communiquent entre eux. Le jury attend une compréhension claire des flux de données et des

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

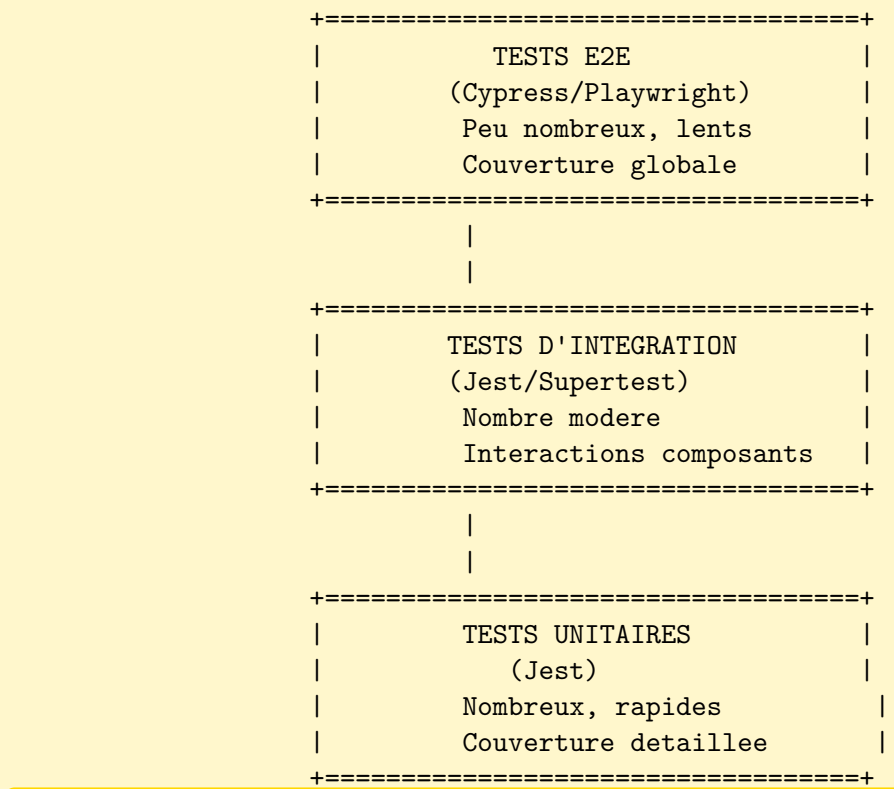
7.1 Stratégie de tests

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(
```

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
```

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/