CAHIER DE CHARGE TECHNIQUE

12/05/2025

CAHIER DES CHARGES TECHNIQUE : APPLICATION DE GESTION DE PROJET

1. INTRODUCTION

Objectif du cahier des charges technique : Définir l'architecture, les technologies, les composants et les exigences techniques nécessaires au développement de l'application de gestion de projet.

Portée du document : Ce document couvre les aspects techniques de la conception, du développement, du déploiement et de la maintenance de l'application.

2. ARCHITECTURE TECHNIQUE GÉNÉRALE

Architecture logicielle : Microservices. Chaque microservice sera responsable d'une fonctionnalité spécifique (gestion des tâches, gestion des utilisateurs, gestion des projets, etc.).

Architecture matérielle : Cloud (AWS, Azure, Google Cloud). L'application sera déployée sur une infrastructure cloud pour garantir la scalabilité et la disponibilité.

Schémas d'architecture réseau et applicative :

3. ENVIRONNEMENT TECHNIQUE

Langages de programmation :

Backend : Java (Spring Boot)

Frontend : React

Microservices : Polyglot (Java, Python, Node.js selon les besoins spécifiques)

Frameworks utilisés :

Spring Boot (Backend)

React (Frontend)

Docker (Containerisation)

Kubernetes (Orchestration)

SGBD :

PostgreSQL (Données relationnelles)

MongoDB (Données non relationnelles, logs, événements)

Redis (Cache)

Systèmes d'exploitation : Linux (serveurs)

Outils de développement, IDE, CI/CD, versionning :

IntelliJ IDEA (IDE)

Git (Versionning)

GitHub/GitLab (Repository)

Jenkins/GitLab CI (CI/CD)

SonarQube (Analyse de code)

4. SPÉCIFICATIONS DES COMPOSANTS

Gestion des utilisateurs :

Objectif : Gérer les utilisateurs, leurs rôles et leurs permissions.

Technologies associées : Java, Spring Security, PostgreSQL.

Interfaces : API REST (CRUD pour les utilisateurs, authentification, autorisation).

Schémas UML :

Gestion des projets :

Objectif : Gérer les projets, leurs détails et leurs membres.

Technologies associées : Java, Spring Boot, PostgreSQL.

Interfaces : API REST (CRUD pour les projets, ajout/suppression de membres).

Schémas UML :

Gestion des tâches :

Objectif : Gérer les tâches, leur statut, leur assignation et leurs dépendances.

Technologies associées : Java, Spring Boot, PostgreSQL, Redis (pour le cache).

Interfaces : API REST (CRUD pour les tâches, gestion des dépendances, assignation).

Schémas UML :

Gestion des rapports :

Objectif : Générer des rapports sur l'avancement des projets, l'utilisation des ressources, etc.

Technologies associées : Python, Pandas, MongoDB (pour les données agrégées), visualisation (ex : Grafana).

Interfaces : API REST (pour la génération de rapports).

Schémas UML :

Gestion des notifications :

Objectif : Envoyer des notifications aux utilisateurs concernant les changements importants (tâche assignée, date limite dépassée, etc.).

Technologies associées : Node.js, RabbitMQ, Websockets.

Interfaces : API pour l'envoi de notifications.

Schémas UML :

5. ARCHITECTURE DES BASES DE DONNÉES

Modèle conceptuel (MCD) et modèle logique (MLD) :

Contraintes d’intégrité : Clés primaires, clés étrangères, contraintes d'unicité, validation des données.

Réplication, sauvegarde, performances :

Réplication : PostgreSQL (réplication asynchrone) et MongoDB (replica sets).

Sauvegarde : Sauvegardes régulières automatisées sur S3 ou un autre stockage cloud.

Performances : Indexation appropriée, optimisation des requêtes, caching (Redis).

Plan de migration de données : Utilisation de scripts SQL et d'outils de migration (ex : Flyway) pour la gestion des schémas et la migration des données.

6. CONTRAINTES TECHNIQUES

Performances :

Temps de réponse : Inférieur à 2 secondes pour la plupart des requêtes.

Volumétrie : Support de milliers d'utilisateurs et de projets simultanément.

Disponibilité : 99.9% de disponibilité.

Sécurité :

Authentification : OAuth2/JWT.

Autorisation : RBAC (Role-Based Access Control).

Chiffrement : Chiffrement des données sensibles au repos et en transit (HTTPS).

Pare-feu : Utilisation de pare-feu cloud (AWS WAF, Azure Firewall) et de règles de sécurité réseau.

Compatibilité :

Navigateurs : Chrome, Firefox, Safari, Edge (dernières versions).

Appareils : Responsive design pour une compatibilité avec les ordinateurs de bureau, les tablettes et les smart-phones.

Systèmes tiers : Intégration avec des API tierces (ex : Slack, Jira) via API REST.

Scalabilité : Architecture microservices et déploiement cloud pour permettre une scalabilité horizontale (ajout de nouvelles instances).

7. EXIGENCES DE SÉCURITÉ

Protocoles utilisés : HTTPS, OAuth2, JWT.

Gestion des accès, rôles et permissions : Mise en place d'un système de gestion des rôles et des permissions (RBAC) pour contrôler l'accès aux fonctionnalités.

Journalisation (logs, audits) : Journalisation complète des actions des utilisateurs et des événements système pour l'audit et le débogage.

Conformité réglementaire : RGPD (respect de la confidentialité des données personnelles).

8. TESTS ET VALIDATIONS TECHNIQUES

Types de tests prévus :

Tests unitaires (Jest, JUnit)

Tests d'intégration (Spring Test, Cypress)

Tests de charge et performance (JMeter, Gatling)

Tests de sécurité (OWASP ZAP, SonarQube)

Outils utilisés pour les tests : Jest, JUnit, Spring Test, Cypress, JMeter, Gatling, OWASP ZAP, SonarQube.

Critères d'acceptation technique : Code conforme aux standards de qualité, pas de vulnérabilités critiques, performances conformes aux exigences, tests unitaires et d'intégration réussis.

9. DÉPLOIEMENT

Stratégie de déploiement : Blue/Green Deployment (pour minimiser les interruptions de service).

Environnements :

Dev (Développement)

Staging (Pré-production)

Prod (Production)

Outils : Docker, Kubernetes, Ansible.

Procédure d'installation et configuration : Documentation détaillée de la procédure d'installation et de configuration de chaque composant.

10. MAINTENANCE ET SUPERVISION

Suivi des performances (APM, monitoring) : Utilisation d'outils d'APM (ex : New Relic, Dynatrace) et de monitoring (ex : Prometheus, Grafana) pour suivre les performances de l'application.

Alerting (Grafana, Prometheus, Sentry) : Mise en place d'alertes pour détecter les problèmes de performance ou les erreurs.

Plan de maintenance : Mises à jour régulières des librairies et des frameworks, correctifs de sécurité, optimisation des performances.

SLA techniques (temps de réponse, taux de disponibilité…) : Définition de SLAs pour le temps de réponse et le taux de disponibilité (ex : 99.9%).

11. GESTION DES ÉVOLUTIONS

Processus de gestion des changements techniques : Utilisation d'un système de gestion des tickets (Jira) pour suivre les demandes de changement.

Suivi des versions / changelog : Utilisation de Git et de tags pour le suivi des versions, création d'un changelog pour documenter les modifications.

Outils de ticketing : Jira.

12. PLANNING TECHNIQUE

Découpage du projet en phases techniques :

Phase 1 : Conception de l'architecture et mise en place de l'infrastructure cloud.

Phase 2 : Développement des microservices (gestion des utilisateurs, gestion des projets, gestion des tâches).

Phase 3 : Développement du frontend (React).

Phase 4 : Tests et validation.

Phase 5 : Déploiement et mise en production.

Tâches techniques majeures :

Conception de l'architecture microservices.

Mise en place de l'infrastructure cloud (AWS, Azure, Google Cloud).

Développement des API REST pour chaque microservice.

Développement du frontend (React).

Tests unitaires, d'intégration, de performance et de sécurité.

Déploiement sur Kubernetes.

Estimations de charge et de durée :

13. ANNEXES TECHNIQUES

Documentation API : Documentation Swagger/OpenAPI pour chaque API REST.

Codes d’exemple ou conventions de codage : Style guides (ex : Airbnb JavaScript Style Guide), conventions de nommage, etc.

Spécifications tierces (API externes, SDK, etc.) : Documentation des API tierces utilisées (ex : Slack API, Jira API).

FINANCEMENT DU PROJET

1. **Bootstrapping :**

Utiliser les fonds propres de l'entreprise ou des fondateurs.

Avantage : Contrôle total sur le projet.

Inconvénient : Limité aux ressources disponibles.

1. **Love Money :**

Financement par la famille et les amis.

Avantage : Conditions généralement plus souples que les investisseurs traditionnels.

Inconvénient : Risque de tensions si le projet échoue.

1. **Prêts bancaires :**

Obtenir un prêt auprès d'une banque.

Avantage : Possibilité de financer des montants importants.

Inconvénient : Nécessite un business plan solide et des garanties.

1. **Subventions et aides publiques :**

Rechercher des subventions ou des aides proposées par les gouvernements locaux, régionaux ou nationaux.

Avantage : Financement non dilutif (pas de perte de capital).

Inconvénient : Processus de demande souvent long et complexe.

1. **Crowdfunding :**

Collecter des fonds auprès d'un grand nombre de personnes via des plateformes de crowdfunding.

Avantage : Permet de valider l'intérêt du marché.

Inconvénient : Nécessite une campagne de communication efficace.

1. **Business Angels :**

Obtenir un financement auprès d'investisseurs privés qui investissent dans des startups.

Avantage : Expertise et réseau des investisseurs.

Inconvénient : Perte d'une partie du contrôle de l'entreprise.

1. **Capital-risque :**

Obtenir un financement auprès de fonds de capital-risque.

Avantage : Possibilité de financer des montants très importants.

Inconvénient : Forte pression pour une croissance rapide et une sortie à court terme.