Documentation Technique du Backend ProjectFlow

ProjectFlow est une application de gestion de projets conçue pour simplifier la collaboration d'équipes à travers différents niveaux d'accès et responsabilités. Le backend développé en Spring Boot fournit une API REST robuste qui interagit avec une base de données MySQL et communique avec un frontend React.js + Vite. L'application est conçue pour offrir une expérience de gestion de projets complète, sécurisée et scalable.

Flux de Données Principal

1. Requête entrante → Filtre d'authentification JWT
2. Validation des rôles → Application des règles RBAC
3. Traitement métier → Services métiers
4. Accès aux données → Repositories JPA
5. Réponse formatée → DTO (Data Transfer Objects)

Modèle de Données et Base de Données

Structure de la Base de Données

La base de données utilise le schéma projectflow avec les tables principales :

Tables de Sécurité

- users : Stocke les informations des utilisateurs (identifiants, hachage de mot de passe, informations personnelles)
- roles : Définit les 5 rôles prédéfinis avec niveaux hiérarchiques
- permissions : Permissions granulaires par catégorie (user, project, task, organization, system)
- role_permissions : Relation many-to-many entre rôles et permissions
- user_roles : Association des utilisateurs aux rôles, avec contexte organisationnel et de projet

Tables Métier

- organizations : Entités organisationnelles avec métadonnées (logo, site web)
- projects : Projets avec détails (statut, dates, budget) liés aux organisations
- tasks : Tâches hiérarchiques avec priorités, statuts et assignations

Caractéristiques des Données

- JSON settings : Champs flexibles pour stocker des configurations personnalisées
- Hiérarchie de tâches : Possibilité d'avoir des tâches parent/enfant
- Audit automatique : Date de création et date de mise à jour pour toutes les entités
- Performance : Indexation des colonnes fréquemment utilisées pour les requêtes complexes

Gestion des Rôles et Permissions (RBAC)

Hiérarchie des Rôles

Rôle	Niveau	Description
SUPER_ADMIN	1	Accès complet à toutes les fonctionnalités et organisations
ORG_ADMIN	2	Gestion complète d'une organisation spécifique
PROJECT_MANAGER	3	Gestion des projets et des équipes au sein d'une organisation
TEAM_MEMBER	4	Accès limité aux tâches qui leur sont assignées et aux projets visibles
CLIENT	5	Accès en lecture seule aux projets dont ils sont clients

Permissions Clés par Rôle

- SUPER_ADMIN : [*] (Toutes les permissions)
- ORG_ADMIN : Gestion des organisations, création de projets, gestion des utilisateurs
- PROJECT_MANAGER : Gestion complète des projets et tâches, génération de rapports
- TEAM_MEMBER : Mise à jour des tâches assignées, visualisation des projets
- CLIENT : Accès en lecture aux projets clients et aux livrables

Logique d'Autorisation

- Annotations Spring Security : @PreAuthorize sur les contrôleurs
- Héritage de permissions : Les rôles supérieurs héritent des permissions des rôles inférieurs
- Contexte organisationnel : Les permissions sont évaluées dans le contexte de l'organisation
- Validation au runtime : Vérification des permissions pour chaque opération sensible

Services Métiers et Logique d'Implémentation

Service d'Authentification (AuthService)

Fonctionnalités :

- Génération et validation de JWT
- Authentification utilisateur sécurisée
- Création de nouveaux comptes avec validation
- Gestion de la session sécurisée

Logique d'implémentation :

1. Valide les identifiants avec Spring Security
2. Génère un JWT contenant l'ID utilisateur et les rôles
3. Gère les erreurs d'authentification (tentatives multiples, compte bloqué)
4. Configure le contexte de sécurité pour les requêtes suivantes

Service des Utilisateurs (UserService)

Fonctionnalités :

- Création, lecture, mise à jour des utilisateurs
- Gestion des associations rôles/utilisateurs
- Validation des identifiants uniques (email, username)
- Recherche par rôle

Logique d'implémentation :

1. Vérifie l'unicité du nom d'utilisateur et de l'email
2. Hache les mots de passe avec BCrypt avant stockage
3. Gère les associations rôles/utilisateurs via la table de jonction
4. Fournit des méthodes de recherche optimisées par rôle

Service des Organisations (OrganizationService)

Fonctionnalités :

- Création et gestion des organisations
- Attribution d'un créateur/propriétaire à chaque organisation
- Configuration flexible via le champ JSON settings
- Suppression sécurisée avec vérification des dépendances

Logique d'implémentation :

1. Associe automatiquement le créateur à l'organisation
2. Gère les paramètres personnalisés via JSON
3. Implémente des vérifications de sécurité pour les opérations sensibles
4. Gère les relations avec les projets associés

Service des Projets (ProjectService)

Fonctionnalités :

- Gestion du cycle de vie des projets (création, mise à jour, suppression)
- Association des projets aux organisations
- Suivi des indicateurs clés (budget, statut, progression)
- Gestion des dates de début et de fin

Logique d'implémentation :

1. Valide la cohérence des dates (début < fin)
2. Gère automatiquement les métadonnées de création

3. Implémente des règles métier pour les changements d'état
4. Fournit des méthodes de filtrage par utilisateur et organisation

Service des Tâches (TaskService)

Fonctionnalités :

- Création et gestion hiérarchique des tâches
- Attribution des tâches aux utilisateurs
- Suivi des heures estimées et réelles
- Gestion des priorités et statuts

Logique d'implémentation :

1. Gère la relation parent/enfant pour les tâches
2. Implémente une logique de priorité (low, medium, high, urgent)
3. Valide les transitions de statut selon les règles métier
4. Fournit des méthodes de recherche par statut et par utilisateur assigné

Logique Générale de l'Application

Processus d'Initialisation

1. Démarrage Spring Boot : Chargement des configurations
2. Initialisation JPA : Création/mise à jour du schéma de base de données
3. Chargement des données initiales : Exécution des scripts SQL (schema.sql, data.sql)
4. Configuration de sécurité : Mise en place des filtres JWT et des règles RBAC

Gestion des Requêtes

1. Filtrage initial : Vérification CORS et traitement des options preflight
2. Authentification : Extraction et validation du JWT
3. Autorisation : Vérification des permissions par annotation
4. Traitement bizness : Exécution de la logique métier dans les services
5. Gestion des erreurs : Traduction des exceptions en réponses HTTP appropriées

Sécurité

- Communications sécurisées : HTTPS recommandé en production
- Stockage sécurisé des mots de passe : Hachage BCrypt avec salage
- Protection CSRF : Désactivée car les tokens JWT sont utilisés
- Sécurité des endpoints : Tous les endpoints sont sécurisés par défaut
- Gestion des sessions : Stateless avec tokens JWT

Gestion des Erreurs

- Exceptions personnalisées : Pour les cas métier spécifiques
- GlobalExceptionHandler : Pour uniformiser les réponses d'erreur
- Journalisation sécurisée : Pas de fuite d'informations sensibles dans les logs
- Codes de statut HTTP appropriés : 4xx pour les erreurs client, 5xx pour les erreurs serveur

Intégration avec le Frontend Next.js

Communication API

- Base URL : http://localhost:8080/api (en développement)
- En-têtes requis :
    - Authorization: Bearer <token> pour les requêtes authentifiées
    - Content-Type: application/json pour les requêtes POST/PUT
- Format des réponses : JSON avec structure cohérente

Mapping des Routes

Endpoint Frontend (React.js + Vite)	Endpoint Backend (Spring Boot)	Fonctionnalité
/api/auth/signin	/api/auth/signin	Authentification
/api/auth/signup	/api/auth/signup	Création de compte
/api/organizations	/api/organizations	Gestion des organisations
/api/projects	/api/projects	Gestion des projets
/api/tasks	/api/tasks	Gestion des tâches
/api/users	/api/users	Gestion des utilisateurs

Gestion du Contexte Utilisateur

- Le frontend stocke le JWT dans le localStorage ou les cookies sécurisés
- Les rôles de l'utilisateur sont utilisés pour conditionner l'UI
- Les requêtes incluent le token JWT dans l'en-tête Authorization
- Le rafraîchissement des tokens est géré automatiquement

Performance et Évolutivité

Optimisations

- Lazy Loading : Relations JPA configurées avec FetchType.LAZY
- Pagination : Implémentation future pour les grandes collections
- Caching : Utilisation de @Cacheable pour les données fréquemment accédées

- Indexes : Création d'indexes pour les colonnes fréquemment requêtées

Évolutivité

- Architecture modulaire : Facilite l'ajout de nouvelles fonctionnalités
- Services indépendants : Faible couplage entre les différents domaines
- Configuration externalisée : Facile à adapter aux différents environnements
- Tests automatisés : Base pour la maintenance continue

Déploiement et Environnements

Environnements Supportés

- Développement : Configuration avec logs détaillés et auto-redémarrage
- Tests : Base de données dédiée pour les tests automatisés
- Production : Configuration sécurisée avec logs minimisés

Configuration Clé

- JWT_SECRET : Doit être personnalisé en production
- Database credentials : Doivent être stockés dans des variables d'environnement sécurisées
- CORS configuration : Doit être restreinte aux domaines frontend autorisés en production

Perspectives d'Amélioration

Fonctionnalités Futures

- Système de notifications en temps réel (WebSocket)
- Reporting analytique avec agrégation de données
- Intégration avec des outils externes (calendriers, emails)
- API publique avec documentation Swagger/OpenAPI

Optimisations Techniques

- Implémentation de pagination et filtrage côté serveur
- Ajout de mise en cache Redis pour améliorer les performances
- Intégration de tracing distribué (Sleuth/Zipkin)
- Implémentation de rate limiting pour sécuriser l'API