Plan d'Implémentation Technique – Plateforme A3E (InfraQC)

Introduction

Objectif du Document

Ce rapport constitue le plan directeur technique pour la conception et la mise en œuvre de la plateforme SaaS A3E (désormais "InfraQC"). Il a pour vocation de traduire les impératifs stratégiques de l'entreprise — conformité absolue à la Loi 25, expérience utilisateur terrain inégalée, et innovation disruptive via l'IA — en une architecture technique robuste, évolutive et sécurisée. Chaque décision, chaque composant et chaque ligne de code décrits dans ce document sont subordonnés à la réalisation de ces objectifs fondamentaux.

Principes Directeurs

L'architecture d'InfraQC n'est pas une simple collection de technologies ; elle est l'incarnation de notre stratégie d'entreprise. Par conséquent, son développement sera guidé par trois principes fondamentaux et non négociables :

1. **Souveraineté par Conception (Sovereignty-by-Design) :** La conformité à la Loi 25 n'est pas une case à cocher, mais notre principal argument de vente auprès des organismes publics québécois. L'architecture doit garantir, sans ambiguïté, que 100% des données client, y compris les renseignements personnels les plus sensibles, sont stockées et traitées au Canada par une entité de droit canadien. Cette approche proactive élimine les risques juridiques et le fardeau administratif des Évaluations des Facteurs relatifs à la Vie Privée (EFVP) pour nos clients, nous positionnant comme le partenaire de confiance par défaut.[1]
2. **Fiabilité sur le Terrain (Field-First Reliability) :** Pour les utilisateurs sur un chantier, la performance et la fiabilité ne sont pas négociables. Notre plateforme doit être conçue en partant du principe que la connectivité réseau est un luxe et non une garantie. L'expérience hors-ligne n'est pas une fonctionnalité ajoutée, mais le mode de fonctionnement par défaut de l'application mobile. Chaque action de l'utilisateur doit être instantanément sauvegardée localement, assurant une productivité ininterrompue.

3. **Innovation Intégrée (Integrated Intelligence) :** L'intelligence artificielle n'est pas un gadget, mais une composante intégrale de la valeur que nous offrons. Les capacités d'IA, notamment notre moteur de recherche sémantique sur les documents de projet, ne doivent pas être un module complémentaire. Elles doivent être profondément intégrées à l'architecture de données pour créer une synergie transparente entre la gestion de projet et l'aide à la décision intelligente.[3]

Public Cible

Ce document s'adresse à l'équipe d'ingénierie, aux architectes de solutions, à l'équipe DevOps et aux chefs de produit d'InfraQC. Il servira de référence unique ("single source of truth") pour tous les efforts de développement, garantissant l'alignement, la cohérence et l'excellence technique tout au long du cycle de vie du produit.

Section 1 : Architecture Fondamentale : Souveraineté, Sécurité et Conformité

Cette section établit les fondations non négociables de la plateforme InfraQC. Les choix effectués ici sont les plus critiques car ils conditionnent la conformité légale, la confiance des clients et la résilience globale du système. Ils constituent la base sur laquelle toutes les autres couches de l'application seront construites.

1.1. Hébergement sur un Cloud Souverain : Le Pilier de la Confiance

Analyse du Contexte et Justification

Le marché des logiciels de construction pour le secteur public au Québec est profondément influencé par la **Loi 25** (Loi modernisant des dispositions législatives en matière de protection des renseignements personnels). Cette loi impose aux organismes publics, lors de l'acquisition d'un système d'information ou de tout projet impliquant un transfert de renseignements personnels (RP) hors du Québec, de réaliser une **Évaluation des Facteurs relatifs à la Vie Privée (EFVP)**.[4] Cette obligation, bien que nécessaire pour la protection des citoyens, représente un fardeau administratif et un risque juridique significatif pour les clients potentiels d'InfraQC.[2]
Les solutions concurrentes dominantes, telles que Procore et Autodesk Construction Cloud, bien qu'elles puissent offrir des options de stockage des fichiers de projet au Canada, sont

des entreprises de droit américain.[6] Leurs politiques de confidentialité indiquent souvent que les données de compte utilisateur (noms, courriels, identifiants), qui sont des renseignements personnels, peuvent être traitées et stockées aux États-Unis.[8] Cette pratique déclenche systématiquement l'obligation de réaliser une EFVP pour leurs clients québécois, car les données sont soumises à des lois extraterritoriales comme le **CLOUD Act américain**, qui permet aux autorités américaines de demander l'accès aux données, même si elles sont stockées à l'étranger.[2]

La stratégie d'InfraQC est de transformer cette contrainte réglementaire en un avantage concurrentiel décisif. Pour ce faire, il est impératif de sélectionner un fournisseur de services infonuagiques qui non seulement stocke les données au Canada, mais qui garantit également que **100% des données (fichiers de projet et renseignements personnels) sont stockées ET traitées au Canada, par une entité de droit canadien non assujettie au CLOUD Act**.[2] Cette approche élimine la nécessité pour nos clients de réaliser une EFVP liée au transfert de données transfrontalier, simplifiant leur processus d'acquisition et renforçant notre proposition de valeur.

Sélection du Fournisseur

Les critères de sélection pour notre fournisseur cloud sont stricts et directement alignés sur notre stratégie de souveraineté :

1. **Entité Juridique :** L'entreprise doit être de droit canadien.
2. **Localisation des Centres de Données :** Les centres de données primaires et de sauvegarde doivent être situés physiquement au Canada, avec une préférence pour le Québec.
3. **Garantie Contractuelle :** Le contrat de service doit garantir explicitement que les données ne sont pas soumises au CLOUD Act américain.
4. **Certifications de Sécurité :** Le fournisseur doit détenir des certifications reconnues internationalement, telles que ISO/IEC 27001, SOC 2 Type 2, et TIER III pour la résilience de l'infrastructure.

Analyse des Candidats :

- **R2i (Edwin Cloud) :** Ce fournisseur représente le choix privilégié. Il s'agit d'une entreprise 100% québécoise qui possède et opère ses propres centres de données au Québec et en Ontario. R2i garantit explicitement par contrat la souveraineté totale des données et la non-soumission au CLOUD Act. De plus, ils détiennent un portefeuille complet de certifications critiques, incluant ISO 27001, SOC 2 Type 2, et TIER III, ce qui atteste de la robustesse de leurs pratiques de sécurité et de leur infrastructure.[11]
- **Clever Cloud :** Ce fournisseur européen met un fort accent sur la souveraineté numérique. Pour ses opérations canadiennes, il utilise le centre de données d'OVHCloud à Beauharnois, QC, qui est certifié ISO 27001.[13] Bien que Clever Cloud soit une option viable et respectueuse de la souveraineté, le fait que R2i soit une entité de propriété canadienne offre une garantie de souveraineté encore plus directe et alignée

avec notre message marketing.

Décision et Implémentation :

InfraQC déploiera son infrastructure sur R2i (Edwin Cloud). Le contrat de service sera négocié pour inclure des clauses explicites sur la résidence et la souveraineté des données. L'équipe DevOps collaborera avec les architectes de R2i pour concevoir une architecture de cloud privé virtuel (VPC) dédiée, isolée et sécurisée, qui servira de fondation à tous nos services.

Tableau 1 : Comparatif et Sélection du Fournisseur de Cloud Souverain

Le tableau suivant résume l'analyse comparative qui justifie la sélection de R2i.

Critère	R2i (Edwin Cloud)	Clever Cloud (sur OVHCloud)	Géants US (AWS/Azure/GCP avec région Canada)
Propriété de l'Entité	Canadienne (Québécoise) [11]	Européenne [2]	Américaine
Localisation des Données Projet	Québec / Ontario [11]	Québec (Beauharnois) [13]	Canada (Central)
Localisation Traitement des RP	Garanti au Canada [10]	Garanti au Canada [13]	Souvent aux États-Unis [9]
Assujettissement au CLOUD Act	Non (explicitement garanti) [11]	Non (entité européenne)	Oui (par la société mère)
Déclenchement EFVP pour Client	Non	Non	Oui
Certifications Clés	ISO 27001, SOC 2 Type 2, TIER III [11]	ISO 27001 (via OVHCloud) [13]	ISO 27001, SOC 2 Type 2
Conclusion	**Sélectionné**	Alternative viable	Risque de conformité inacceptable

Le choix de R2i n'est pas seulement une décision d'hébergement, mais un principe architectural unificateur. Pour maintenir une souveraineté de bout en bout, chaque composant de notre pile technologique doit pouvoir être hébergé et géré au sein de notre infrastructure R2i. Cette approche, que nous nommons "l'Empilement Souverain" (Sovereign Stack), signifie que nous privilégierons des solutions auto-hébergeables lorsque c'est possible. Par exemple, pour notre base de données vectorielle, une extension comme pgvector au sein de notre instance PostgreSQL est préférable à un service SaaS externe comme Pinecone. De même, pour notre moteur d'IA, l'utilisation d'API de modèles comme GPT-4 sera considérée comme une solution de démarrage, avec une feuille de route claire vers l'hébergement d'un modèle open-source sur notre propre infrastructure. Ce principe directeur garantit que notre promesse de souveraineté n'est pas seulement une déclaration marketing, mais une réalité technique vérifiable à chaque couche de notre application.

1.2. Architecture de Sécurité "Zero Trust" et Conformité Continue

Philosophie

La sécurité n'est pas une fonctionnalité que l'on ajoute à la fin du développement, mais une fondation intégrée à chaque étape du cycle de vie de l'application. Nous adopterons une architecture de sécurité **"Zero Trust"**, dont le principe est simple : ne jamais faire confiance, toujours vérifier. Dans ce modèle, il n'y a plus de périmètre réseau "sûr". L'accès n'est pas accordé à des réseaux entiers, mais à des applications spécifiques, et ce, uniquement après une authentification et une autorisation rigoureuses et continues pour chaque requête.

Composants d'Implémentation

- **Authentification et Autorisation :**
  - **Standardisation :** L'ensemble de la plateforme utilisera les standards de l'industrie **OAuth 2.0 et OpenID Connect (OIDC)** pour sécuriser toutes les API, qu'elles soient exposées à notre application cliente ou à des services tiers. Cela garantit une gestion des accès robuste et interopérable.
  - **Authentification Multi-Facteurs (MFA) :** La MFA sera obligatoire pour tous les comptes utilisateurs sans exception, qu'il s'agisse des clients ou des administrateurs internes. Cette mesure simple est l'une des plus efficaces pour prévenir les accès non autorisés.
  - **Contrôle d'Accès Basé sur les Rôles (RBAC) :** Un système RBAC granulaire sera implémenté pour appliquer rigoureusement le principe du moindre privilège. Des rôles prédéfinis (ex: "Gestionnaire de Projet", "Surintendant de chantier", "Administrateur Client", "Lecteur seul") seront associés à des ensembles de permissions spécifiques (ex: CREATE_REPORT, READ_DOCUMENT, UPDATE_PROJECT_BUDGET). Chaque utilisateur se verra assigner un ou plusieurs rôles au sein d'une organisation, limitant ses actions à ce qui est strictement nécessaire pour sa fonction.
- **Gestion des Secrets :**
  - Il est formellement interdit de stocker des informations sensibles (clés d'API, mots de passe de base de données, certificats TLS) directement dans le code source ou dans des fichiers de configuration non chiffrés.
  - Nous déploierons une instance de **HashiCorp Vault** sur notre infrastructure R2i. Vault agira comme un coffre-fort centralisé pour tous les secrets d'application. Les applications s'authentifieront auprès de Vault pour récupérer les secrets nécessaires au démarrage, et nous mettrons en place des politiques de rotation automatique des clés pour minimiser la fenêtre d'exposition en cas de

compromission.

- **Sécurité Réseau :**
  - **Micro-segmentation :** Au sein de notre VPC chez R2i, nous appliquerons des règles de pare-feu strictes pour isoler les services les uns des autres (micro-segmentation). Par exemple, le service de gestion des documents ne pourra communiquer qu'avec la base de données PostgreSQL et le service d'authentification, et sera incapable d'initier une connexion vers le service de facturation. Cela limite la propagation latérale d'une éventuelle attaque.
  - **Pare-feu d'Application Web (WAF) :** Tous les points d'entrée publics de l'application seront protégés par un WAF configuré pour bloquer les attaques web courantes identifiées par l'OWASP Top 10, telles que l'injection SQL, le cross-site scripting (XSS) et l'inclusion de fichiers locaux.
- **DevSecOps et Conformité Continue :**
  - La sécurité sera intégrée dans notre pipeline de **CI/CD (Continuous Integration/Continuous Deployment)**, qui sera géré via **GitHub Actions**.
  - **Analyse de Code Statique (SAST) :** Chaque pull request déclenchera automatiquement une analyse SAST pour identifier les vulnérabilités dans le code avant même la fusion.
  - **Analyse de la Composition Logicielle (SCA) :** Des outils SCA scanneront nos dépendances (bibliothèques Python et JavaScript) pour détecter les vulnérabilités connues et les problèmes de licence.
  - **Tests de Pénétration :** Des tests de pénétration seront effectués trimestriellement par une firme externe spécialisée et réputée pour évaluer de manière proactive la sécurité de notre plateforme.
  - **Surveillance de la Configuration :** Des outils automatisés surveilleront en permanence la configuration de notre infrastructure cloud pour détecter les dérives (ex: un port ouvert par erreur) et alerter l'équipe DevOps.

Section 2 : Architecture Applicative Backend : Performance et Évolutivité

Cette section définit l'architecture du cœur logique de la plateforme InfraQC. Les décisions prises ici visent à optimiser la vitesse de développement initiale tout en garantissant la maintenabilité à long terme et la capacité à évoluer avec les besoins de l'entreprise et la croissance du nombre d'utilisateurs.

2.1. Modèle Architectural : Le Monolithe Modulaire comme Fondation Stratégique

Analyse et Justification

Le débat entre une architecture en microservices et une architecture monolithique est central dans la conception de systèmes modernes. Alors que les microservices offrent une flexibilité et une scalabilité maximales en découpant l'application en petits services indépendants, ils introduisent une complexité opérationnelle considérable, notamment en matière de gestion de réseau, de déploiement coordonné, de transactions distribuées et d'observabilité. Pour une équipe de développement en phase de démarrage comme celle d'InfraQC, cette complexité peut considérablement ralentir la vélocité et détourner l'attention du développement des fonctionnalités métier.
Par conséquent, l'approche la plus pragmatique et stratégique pour InfraQC est d'adopter un **monolithe modulaire**. Concrètement, l'application sera développée et déployée comme une seule unité (un monolithe), ce qui simplifie le développement, les tests et le déploiement. Cependant, le code source sera rigoureusement organisé en modules logiques distincts et faiblement couplés, chacun avec des responsabilités claires. Ces modules pourraient inclure, par exemple, Authentication, ProjectManagement, DocumentStore, Reporting, et RAGService. Cette approche offre le meilleur des deux mondes : la vitesse de développement et la simplicité opérationnelle d'un monolithe, tout en préparant le terrain pour une future migration vers les microservices. La communication entre les modules se fera via des interfaces bien définies (par exemple, des classes de service), simulant des appels d'API internes. Cette discipline de conception garantit que si un module doit être extrait pour devenir un microservice indépendant à l'avenir, le couplage avec le reste de l'application sera déjà minimisé.

Stratégie d'Évolution

La migration vers une architecture de microservices ne sera pas un objectif en soi, mais une réponse à des besoins concrets. Elle sera envisagée lorsque des déclencheurs spécifiques apparaîtront, tels que :

- **Goulots d'étranglement de performance :** Si un module spécifique (par exemple, le traitement des documents) consomme une part disproportionnée des ressources et nécessite une mise à l'échelle indépendante du reste de l'application.
- **Croissance de l'équipe :** Lorsque l'équipe d'ingénierie s'agrandira au point où des équipes distinctes pourront prendre en charge des domaines fonctionnels complets, la séparation en microservices facilitera l'autonomie des équipes.

2.2. Pile Technologique Backend : Sélection et Justification

Analyse et Justification

Le choix du langage et du framework backend est une décision fondamentale qui influence la performance, la productivité des développeurs et la capacité à intégrer des technologies futures. L'arbitrage principal pour une application web moderne en 2025 se situe entre l'écosystème JavaScript, avec Node.js, et l'écosystème Python.

- **Node.js avec TypeScript** est un choix dominant pour les applications web générales en raison de ses excellentes performances en matière d'entrées/sorties (I/O) non bloquantes et de son immense écosystème de paquets via npm.
- **Python avec FastAPI** est spécifiquement recommandé pour les applications qui ont une forte composante d'intelligence artificielle. FastAPI est un framework moderne qui offre des performances asynchrones comparables à celles de Node.js, une syntaxe claire et concise, et surtout, une intégration native et sans friction avec l'écosystème de data science et d'IA (bibliothèques comme LangChain, PyTorch, Hugging Face, etc.).[14]

Compte tenu que la fonctionnalité de Recherche Augmentée par Récupération (RAG) est un différenciateur stratégique et non une simple fonctionnalité annexe pour InfraQC, la pile **Python 3.12+ avec le framework FastAPI** est sélectionnée. Ce choix stratégique permet de minimiser les frictions techniques et cognitives entre l'application principale et les services d'IA, et de capitaliser sur un seul écosystème de talents et d'outils.

Composants de la Pile

- **Framework Web : FastAPI** pour sa performance, sa validation de données intégrée via Pydantic, et sa génération automatique de documentation OpenAPI.
- **Serveur ASGI : Uvicorn** sera utilisé pour le développement, et en production, il sera géré par **Gunicorn** pour une gestion robuste des processus workers.
- **Validation des Données : Pydantic**, qui est au cœur de FastAPI, sera utilisé pour définir des schémas de données clairs, fortement typés et auto-validés.
- **ORM (Object-Relational Mapper) : SQLAlchemy 2.0** sera utilisé pour toutes les interactions avec la base de données PostgreSQL, en tirant parti de son support complet pour les opérations asynchrones. Les migrations de schéma de base de données seront gérées avec **Alembic**.
- **Mise en Cache : Redis** sera utilisé comme cache en mémoire pour accélérer les réponses d'API fréquentes, stocker les sessions utilisateur et servir de broker de messages pour les tâches asynchrones.
- **Gestion des Tâches Asynchrones :** Pour les opérations longues ou gourmandes en ressources (ex: traitement de PDF lors de l'ingestion pour le RAG, génération de rapports complexes), nous utiliserons **Celery** avec Redis comme broker de messages. Cela permet de décharger ces tâches du thread principal de l'API, garantissant que l'application reste réactive.
- **Conteneurisation :** L'ensemble de l'application et de ses dépendances sera

conteneurisé à l'aide de **Docker**. Les conteneurs seront déployés et orchestrés via les services gérés de R2i, qui sont basés sur des standards comme Kubernetes, assurant la portabilité et la scalabilité.

Tableau 2 : Sélection et Justification de la Pile Technologique Complète

Ce tableau offre une vue d'ensemble de la pile technologique complète d'InfraQC, justifiant chaque choix en fonction des objectifs du projet.

Composant	Technologie Choisie	Justification	Sources Clés
Hébergement Cloud	R2i (Edwin Cloud)	Souveraineté des données (Loi 25), propriété canadienne, non-soumis au CLOUD Act.	[11]
Frontend	React (avec Vite)	Écosystème mature, idéal pour les PWA, performance avec Vite.	
Backend	Python (FastAPI)	Performance asynchrone, intégration native avec l'écosystème IA (LangChain), typage fort avec Pydantic.	
Base de Données Opérationnelle	PostgreSQL 16+	Robustesse, conformité ACID, support JSONB, écosystème d'extensions (pgvector).	
Base de Données Vectorielle	Extension pgvector	Maintient la souveraineté des données, unifie la pile de données, performance et coût compétitifs.	[2]
Mise en Cache / Broker	Redis	Standard de l'industrie pour le cache en mémoire et la gestion de files d'attente (Celery).	

Orchestration IA	LangChain	Simplifie et standardise les pipelines RAG complexes.	[17]
CI/CD	GitHub Actions	Intégration native avec le dépôt de code, automatisation des tests, de la sécurité et des déploiements.	
Gestion des Secrets	HashiCorp Vault	Solution open-source robuste, auto-hébergeable pour une souveraineté totale des secrets.	

2.3. Conception des API : Approche "API-First" avec GraphQL

Analyse et Justification

Une approche **"API-First"** est impérative pour garantir la découplage et l'évolutivité de la plateforme. L'API est le contrat formel qui définit les interactions entre le frontend, le backend et tout service tiers potentiel.[2] Plutôt que de concevoir l'API comme une conséquence de l'implémentation du backend, nous la définirons en premier lieu.
Bien que REST soit un standard éprouvé, **GraphQL** est choisi comme technologie pour l'API principale exposée à l'application cliente (la PWA). Cette décision est motivée par plusieurs avantages stratégiques, particulièrement pertinents pour notre cas d'usage :

1. **Efficacité sur Réseaux Instables :** La principale force de GraphQL est de permettre au client de demander précisément les données dont il a besoin, et rien de plus. Pour une PWA utilisée sur un chantier de construction avec une connectivité 3G ou intermittente, la capacité de réduire la taille des payloads de données est un avantage considérable qui se traduit par une application plus rapide et plus fiable.
2. **Développement Frontend Simplifié :** Avec GraphQL, le frontend n'a besoin de connaître qu'un seul point d'entrée (/graphql). Le schéma GraphQL, fortement typé, sert de documentation vivante et permet aux développeurs frontend d'explorer l'API et de construire des requêtes complexes sans dépendre de la disponibilité de multiples endpoints REST et sans risque de sur- ou sous-récupération de données ("over-fetching" ou "under-fetching").
3. **Évolution sans Rupture :** L'ajout de nouveaux champs ou de nouveaux types au schéma GraphQL n'impacte pas les clients existants. Cela élimine la nécessité de gérer

des versions d'API complexes (ex: /api/v1/, /api/v2/), ce qui simplifie grandement la maintenance et l'évolution à long terme de la plateforme.

Implémentation

- **Bibliothèque Serveur :** Nous utiliserons la bibliothèque **Strawberry** pour implémenter le serveur GraphQL. Strawberry a été choisie pour son intégration native et élégante avec FastAPI et Pydantic, ce qui permet de réutiliser nos modèles de données Pydantic pour générer automatiquement le schéma GraphQL.
- **Communications Internes :** Pour les communications internes entre les modules du monolithe (ou entre de futurs microservices), des appels de fonction directs ou des API REST internes, plus légères et plus performantes pour des cas d'usage serveur-à-serveur, pourront être utilisées. GraphQL sera réservé à l'interface exposée au client.

Tableau 3 : Schéma GraphQL Initial des Entités Principales (Exemple Simplifié)

Ce schéma initial sert de point de départ pour le développement. Il définit les entités de base et les opérations fondamentales de la plateforme.

GraphQL

```
# Enum pour les rôles utilisateurs
enum UserRole {
  ADMIN
  PROJECT_MANAGER
  FIELD_WORKER
}

# Enum pour le statut de synchronisation
enum SyncStatus {
  SYNCED
  PENDING
  FAILED
}

# Enum pour le statut des documents
enum DocumentStatus {
  PROCESSING
  INDEXED
```

```
  FAILED_INDEXING
}

# Type pour les utilisateurs de la plateforme
type User {
  id: ID!
  name: String!
  email: String!
  role: UserRole!
  projects: [Project!]
}

# Type pour un projet de construction
type Project {
  id: ID!
  name: String!
  projectNumber: String!
  client: String!
  users: [User!]
  documents(first: Int, after: String): DocumentConnection!
  reports(first: Int, after: String): VisitReportConnection!
}

# Type pour un document (plan, devis, etc.)
type Document {
  id: ID!
  fileName: String!
  fileType: String!
  uploadDate: DateTime!
  status: DocumentStatus!
  url: String!
}

# Type pour une photo attachée à un rapport
type Photo {
    id: ID!
    url: String!
    caption: String
}

# Type pour un rapport de visite de chantier
type VisitReport {
  id: ID!
```

```
  title: String!
  creationDate: DateTime!
  author: User!
  content: String!
  photos: [Photo!]
  syncStatus: SyncStatus!
}

# Type pour les résultats de la recherche sémantique
type DocumentSearchResult {
    document: Document!
    score: Float!
    relevantChunk: String!
}

# Types pour la pagination (Connections & Edges)
type DocumentConnection {
    edges:
    pageInfo: PageInfo!
}
type DocumentEdge {
    cursor: String!
    node: Document!
}
#... (définitions similaires pour VisitReportConnection)

type PageInfo {
    hasNextPage: Boolean!
    endCursor: String
}

# Requêtes disponibles
type Query {
  project(id: ID!): Project
  myProjects: [Project!]
  searchDocuments(projectId: ID!, query: String!):
}

# Mutations pour modifier les données
type Mutation {
  createVisitReport(projectId: ID!, title: String!, content: String!, photos: [Upload!]): VisitReport
  uploadDocument(projectId: ID!, file: Upload!): Document
}
```

Section 3 : Architecture Mobile : La Progressive Web App (PWA) "Offline-First"

Cette section détaille l'architecture de l'application cliente, qui est sans doute l'élément le plus critique pour l'adoption par les utilisateurs finaux. Elle est conçue pour offrir une expérience utilisateur robuste, rapide et fiable sur les chantiers de construction, où la connectivité Internet est souvent intermittente ou inexistante.

3.1. Fondations de la PWA et Choix du Framework Frontend

Justification de la PWA

Une **Progressive Web App (PWA)** est choisie comme technologie pour l'application cliente. Cette approche stratégique combine les avantages du web et des applications natives, ce qui est idéal pour notre cas d'usage :

- **Accessibilité :** Accessible via une simple URL, sans nécessiter de passer par un magasin d'applications (App Store, Google Play), ce qui simplifie la distribution et les mises à jour.
- **Installable :** L'utilisateur peut "installer" l'application sur l'écran d'accueil de son appareil (mobile ou de bureau) pour un accès rapide.
- **Fonctionnalités Natives :** Elle peut fonctionner hors ligne, envoyer des notifications push et accéder à certaines fonctionnalités matérielles de l'appareil.

Framework Frontend

Le framework **React** est sélectionné pour le développement de la PWA. Ce choix est motivé par son écosystème extrêmement mature, sa vaste communauté, la disponibilité des talents et ses bibliothèques robustes pour la gestion d'état et la création de PWA complexes. Pour l'initialisation du projet, nous utiliserons **Vite** plutôt que Create React App. Vite offre un environnement de développement significativement plus rapide grâce à son serveur de développement natif ESM (ES Modules) et son système de Hot Module Replacement (HMR) performant, ainsi qu'un processus de build optimisé pour la production.[18]

Composants Clés de la PWA

- **Web App Manifest (manifest.json) :** Ce fichier JSON est essentiel pour qu'un navigateur reconnaisse l'application comme une PWA installable. Il sera méticuleusement configuré pour définir le nom de l'application (InfraQC), les icônes pour différentes résolutions d'écran, la couleur du thème de l'application, et surtout, le mode d'affichage display: "standalone". Ce dernier est crucial pour offrir une expérience immersive, sans l'interface du navigateur (barre d'URL, boutons de navigation), renforçant ainsi la perception d'une application native.
- **Service Worker :** Le Service Worker est le cœur de la fonctionnalité hors-ligne d'InfraQC. Un script service-worker.js sera créé et enregistré lors du premier chargement de l'application. Ce script s'exécute en arrière-plan, indépendamment de la page web, et agit comme un proxy réseau programmable. Il interceptera toutes les requêtes réseau émises par l'application, ce qui nous permettra de mettre en œuvre des stratégies de mise en cache sophistiquées.[19]

3.2. Gestion des Données et du Cache Hors Ligne

Principe "Offline-First"

L'architecture de notre PWA est fondée sur le principe **"Offline-First"**. Cela signifie que l'application est conçue en supposant par défaut que le réseau est indisponible. L'interface utilisateur (UI) interagit *toujours* et *uniquement* avec des données stockées localement sur l'appareil de l'utilisateur. La synchronisation avec le serveur est traitée comme une tâche d'arrière-plan, et non comme une condition préalable à l'interaction de l'utilisateur. Cette approche garantit que l'application reste rapide, réactive et pleinement fonctionnelle, quelle que soit la qualité de la connexion réseau.

Stratégie de Mise en Cache

Une stratégie de mise en cache à deux niveaux sera implémentée :

- **Cache Statique (App Shell) :** Lors de l'événement install du Service Worker, nous utiliserons l'**API Cache** pour pré-mettre en cache toutes les ressources statiques qui constituent l'interface de base de l'application (le "shell applicatif"). Cela inclut le fichier HTML principal, les bundles CSS et JavaScript, les polices de caractères et les icônes. Une fois mis en cache, le chargement de l'application sera quasi instantané lors des visites ultérieures, même en l'absence totale de réseau.[20]
- **Cache Dynamique (Données) :** Toutes les données dynamiques de l'application (informations sur les projets, rapports de visite, listes de tâches, photos, documents)

seront stockées dans **IndexedDB**. IndexedDB est une base de données NoSQL transactionnelle de bas niveau, intégrée au navigateur, conçue pour stocker de grandes quantités de données structurées. Elle est la solution idéale pour notre cas d'usage, car elle permet des requêtes complexes et des transactions atomiques directement sur le client.[21]

Structure d'IndexedDB

Pour simplifier les interactions avec l'API complexe d'IndexedDB, nous utiliserons une bibliothèque d'abstraction comme **Dexie.js**. Dexie.js fournit une syntaxe plus simple et plus expressive, similaire à celle des ORM modernes, tout en gérant la complexité des transactions et des migrations de schémas. Nous définirons des "object stores" (similaires à des tables dans une base de données relationnelle) pour chaque type d'entité principale : projects, reports, photos, users, etc. Des index seront créés sur les champs fréquemment interrogés (par exemple, projectId dans la table reports) pour permettre des recherches rapides et efficaces localement, sans avoir besoin de solliciter le serveur.[21]

3.3. Stratégie de Synchronisation Robuste et Gestion des Conflits

Flux de Données (Écriture)

Le flux de données pour la création ou la modification d'informations est conçu pour être résilient et transparent pour l'utilisateur :

1. **Écriture Locale d'Abord :** Toute action de l'utilisateur qui modifie des données (par exemple, la rédaction d'un rapport de visite, l'ajout d'une photo, la mise à jour d'une tâche) est **immédiatement écrite dans la base de données IndexedDB locale**. L'interface utilisateur est mise à jour instantanément à partir de ces données locales, offrant un retour visuel immédiat et une expérience fluide.[21]
2. **Mise en File d'Attente :** Simultanément, une entrée décrivant l'opération à synchroniser est ajoutée à une file d'attente dédiée, également stockée dans IndexedDB (par exemple, une table sync_queue). Cette entrée contient toutes les informations nécessaires pour rejouer l'opération sur le serveur : la charge utile (payload), le type d'action (CREATE, UPDATE, DELETE), et l'endpoint de l'API à appeler.

Déclenchement de la Synchronisation

La synchronisation est déclenchée de manière opportuniste et robuste :

- **Mécanisme Principal (Background Sync API) :** Après avoir ajouté une tâche à la sync_queue, l'application enregistre une tâche de synchronisation avec le Service Worker en utilisant l'API SyncManager et en lui donnant une balise unique (par exemple, 'sync-reports'). Le navigateur prend alors le relais et déclenchera l'événement sync dans le Service Worker dès qu'il détectera une connectivité réseau stable. Ce mécanisme est extrêmement fiable car il fonctionne même si l'onglet du navigateur ou l'application est fermé.[19]
- **Mécanismes de Repli (Fallback) :** Étant donné que la Background Sync API n'est pas encore supportée par tous les navigateurs, des mécanismes de repli sont essentiels. La synchronisation sera également déclenchée manuellement par le Service Worker lors d'événements clés, assurant que les données sont synchronisées dès que possible :
    - Au chargement initial de l'application (événement load).
    - Lorsque l'appareil revient en ligne (écoute de l'événement online).
    - Lorsque l'application redevient visible (écoute de l'événement visibilitychange).[21]

Processus de Synchronisation dans le Service Worker

Lorsque l'événement de synchronisation est déclenché, le Service Worker exécute la logique suivante :

1. Il lit les tâches en attente dans la table sync_queue d'IndexedDB.
2. Pour chaque tâche, il envoie les données au serveur en exécutant la mutation GraphQL correspondante.
3. Si la requête réussit (réponse 2xx), il supprime la tâche de la file d'attente.
4. Si la requête échoue en raison d'une erreur réseau ou d'une erreur serveur temporaire (5xx), la tâche est conservée dans la file d'attente pour être réessayée ultérieurement. Un compteur de tentatives sera implémenté pour éviter les boucles infinies et marquer les tâches qui échouent de manière persistante.

Gestion des Conflits

La gestion des conflits est un défi majeur et souvent sous-estimé dans les architectures "offline-first". Un conflit survient lorsque la même donnée est modifiée localement sur un appareil et sur le serveur (potentiellement par un autre utilisateur) avant que la synchronisation n'ait lieu. Une stratégie simpliste comme "le dernier qui écrit gagne" (Last Write Wins), basée sur un timestamp, est facile à implémenter mais peut entraîner une perte de données silencieuse et inacceptable, ce qui est un risque que nous ne pouvons pas prendre dans le contexte de la construction où l'intégrité des données est primordiale. InfraQC adoptera une approche plus sophistiquée pour garantir l'intégrité des données :

1. **Détection de Conflit par Versionnement :** Chaque enregistrement dans notre base de données (côté serveur et client dans IndexedDB) aura un champ de version (par exemple, _version ou _last_updated_at). Lorsqu'une mise à jour est envoyée au serveur,

la version de l'enregistrement envoyé est comparée à la version actuellement stockée sur le serveur. Si les versions diffèrent, cela signifie que les données sur le serveur ont été modifiées depuis la dernière synchronisation du client. Le serveur rejettera alors la mise à jour et retournera une erreur de conflit (par exemple, un code de statut HTTP 409 Conflict) avec la version la plus récente des données.

2. Résolution de Conflit Côté Client : Lorsque l'application cliente reçoit une erreur de conflit, elle ne doit pas écraser les données locales. Au lieu de cela, elle doit :
   a. Stocker la version du serveur reçue dans IndexedDB.
   b. Informer l'utilisateur qu'un conflit a été détecté pour un enregistrement spécifique (par exemple, le rapport de visite "Inspection du 3ème étage").
   c. Présenter à l'utilisateur une interface de résolution de conflit dédiée. Cette interface affichera la version locale et la version du serveur côte à côte, en surlignant les différences. L'utilisateur pourra alors choisir de conserver sa version, d'accepter la version du serveur, ou de fusionner manuellement les changements des deux versions.

Bien que cette approche soit plus complexe à implémenter, elle place l'utilisateur au centre du processus de décision et garantit qu'aucune donnée n'est perdue à son insu, renforçant ainsi la confiance et la fiabilité de la plateforme.

Section 4 : Architecture des Données et de l'Intelligence Artificielle

Cette section décrit la colonne vertébrale de la plateforme InfraQC : la manière dont les données sont stockées, gérées et, surtout, exploitées. L'architecture de données est conçue pour supporter de manière optimale à la fois les opérations transactionnelles quotidiennes et le moteur d'intelligence artificielle innovant qui constitue notre principal différenciateur.

4.1. Base de Données Opérationnelle : PostgreSQL pour la Fiabilité Transactionnelle

Justification

Pour les données structurées qui constituent le cœur de la gestion de projet (utilisateurs, projets, budgets, rôles, etc.), la fiabilité, la cohérence et l'intégrité sont primordiales. **PostgreSQL** est sélectionné comme notre système de gestion de base de données relationnelle (SGBDR). Ce choix est fondé sur sa réputation de standard de l'industrie pour les applications SaaS robustes, offrant une conformité ACID (Atomicité, Cohérence, Isolation, Durabilité) totale, un support avancé pour des types de données complexes comme JSONB,

et un écosystème d'extensions inégalé qui sera crucial pour notre stratégie d'IA.

Conception du Schéma Initial

- **Modèle Relationnel :** Un schéma relationnel normalisé sera conçu pour les entités principales de l'application, telles que organizations, users, projects, roles, permissions, documents, reports, etc.
- **Intégrité Référentielle :** Les relations entre les entités seront rigoureusement définies à l'aide de clés étrangères pour garantir l'intégrité des données à travers la base de données. Par exemple, un report devra toujours être associé à un project et un user valides.
- **Optimisation des Performances :** Des index B-tree seront systématiquement créés sur les clés primaires, les clés étrangères et les colonnes fréquemment utilisées dans les clauses WHERE et JOIN pour optimiser les performances des requêtes.
- **Architecture Multi-Tenant :** Le modèle de multi-location, essentiel pour une application SaaS, sera implémenté au niveau de la base de données par une isolation logique. Chaque table principale (comme projects, documents, reports) contiendra une colonne organization_id. Une couche d'abstraction dans notre code (au niveau de l'ORM SQLAlchemy) s'assurera que toutes les requêtes SQL sont systématiquement et automatiquement filtrées par l'organization_id du locataire actuellement authentifié. Cette approche garantit une séparation stricte des données entre les clients, empêchant toute fuite de données d'une organisation à une autre.[2]

4.2. Base de Données Vectorielle : pgvector pour une IA Intégrée et Souveraine

Analyse et Justification

La fonctionnalité de Recherche Augmentée par Récupération (RAG) est au cœur de la proposition de valeur d'InfraQC. Elle nécessite une base de données capable de stocker des représentations numériques de documents (des vecteurs ou "embeddings") et d'effectuer des recherches de similarité sémantique à très grande vitesse.[3] Le choix de cette base de données vectorielle est une décision stratégique qui a des implications profondes sur la souveraineté de nos données, nos coûts opérationnels et la complexité de notre pile technologique.
L'utilisation d'un service SaaS spécialisé et externe comme Pinecone, bien que performant, présenterait plusieurs inconvénients majeurs pour InfraQC.[3] Cela introduirait un silo de données supplémentaire, potentiellement hébergé en dehors de notre infrastructure

souveraine R2i, ce qui compromettrait notre argument de vente principal. De plus, cela ajouterait un coût mensuel récurrent significatif et créerait un risque de dépendance vis-à-vis d'un fournisseur ("vendor lock-in").
C'est pourquoi nous avons pris la décision stratégique d'utiliser l'extension **pgvector** pour PostgreSQL.[2] Cette approche est techniquement et stratégiquement supérieure pour les raisons suivantes :

1. **Souveraineté Totale des Données :** Avec pgvector, les vecteurs sont stockés directement dans notre instance PostgreSQL, aux côtés de nos données opérationnelles. Cela signifie qu'aucune donnée, même les représentations vectorielles de documents confidentiels, ne quitte notre environnement cloud souverain R2i. Notre promesse de conformité à la Loi 25 est ainsi renforcée à un niveau technique profond.[23]
2. **Simplicité Opérationnelle Unifiée :** Il n'y a pas de nouvelle base de données à provisionner, gérer, sauvegarder, surveiller et sécuriser. Notre équipe DevOps peut se concentrer sur la maîtrise d'un seul système de base de données (PostgreSQL), ce qui réduit la charge cognitive et les risques opérationnels.[24]
3. **Performance et Coût Compétitifs :** Des benchmarks récents (2025) démontrent que pgvector, surtout lorsqu'il est combiné avec des optimisations comme pgvectorscale, est non seulement compétitif mais peut surpasser des solutions spécialisées comme Pinecone en termes de débit de requêtes (throughput) et de latence, tout en étant jusqu'à 75% moins cher.[16] Il offre une performance de production robuste pour des millions de vecteurs.[27]
4. **Puissance des Requêtes Hybrides :** L'avantage le plus puissant de pgvector est la capacité de combiner des filtres SQL traditionnels (par exemple, filtrer les documents par date, par type ou par permissions d'accès utilisateur) avec une recherche de similarité vectorielle dans une seule et même requête atomique. Cela permet de construire des systèmes de recherche contextuels et sécurisés de manière beaucoup plus simple et efficace qu'en essayant de synchroniser les métadonnées entre une base de données relationnelle et une base de données vectorielle séparée.[22]

Implémentation avec pgvector

- **Activation :** L'extension pgvector sera activée sur notre instance PostgreSQL managée par R2i.
- **Schéma :** Une table document_chunks sera créée pour stocker les morceaux de documents. Son schéma inclura des colonnes comme id (PK), document_id (FK vers la table documents), organization_id (pour la multi-location), chunk_text (le texte du morceau), et embedding VECTOR(1024). La dimension du vecteur (1024) est choisie pour correspondre à la sortie du modèle d'embedding que nous avons sélectionné (multilingual-e5-large).
- **Indexation :** Pour garantir des recherches de similarité rapides, un index **HNSW (Hierarchical Navigable Small World)** sera créé sur la colonne embedding. HNSW est

préféré à IVFFlat car il offre généralement un meilleur équilibre entre la vitesse de recherche, la précision des résultats (recall) et la performance sur des données qui sont mises à jour dynamiquement, ce qui sera notre cas.[29] Les paramètres de l'index, tels que m (nombre de connexions par nœud) et ef_construction (taille de la liste de candidats lors de la construction), seront ajustés en fonction de benchmarks réalisés sur nos propres données pour trouver le compromis optimal entre vitesse et précision.[31]

4.3. Implémentation du Moteur de Recherche Augmentée par Récupération (RAG)

Framework d'Orchestration

Pour construire et gérer le pipeline RAG complexe, nous utiliserons le framework **LangChain**. LangChain fournit un ensemble d'outils et d'abstractions de haut niveau qui simplifient considérablement l'orchestration des différentes étapes : chargement de documents, découpage, interaction avec les modèles d'embedding et les LLM, et construction de chaînes de traitement complexes ("chains").[17] Son utilisation nous permettra d'accélérer le développement tout en suivant les meilleures pratiques de l'industrie.

Pipeline d'Ingestion (Indexation)

Ce processus s'exécutera de manière asynchrone (via Celery) chaque fois qu'un nouveau document est téléversé sur la plateforme :

1. **Chargement du Document :** Un DocumentLoader approprié (par exemple, PyPDFLoader pour les fichiers PDF, ou des loaders pour les fichiers DOCX, etc.) sera utilisé pour charger le contenu brut du document téléversé.
2. **Découpage (Chunking) :** Le document sera ensuite passé à un RecursiveCharacterTextSplitter. Ce "splitter" intelligent divise le texte en morceaux (chunks) de taille sémantiquement cohérente (par exemple, 1000 caractères) tout en maintenant un chevauchement entre les morceaux (par exemple, 200 caractères). Ce chevauchement est crucial pour préserver le contexte sémantique aux frontières des chunks.[17]
3. **Création des Embeddings :** Chaque chunk de texte est ensuite passé à notre modèle d'embedding pour être converti en un vecteur numérique (embedding).
4. **Stockage :** Le texte du chunk, son embedding vectoriel, et l'ID du document parent sont alors stockés comme une nouvelle ligne dans notre table document_chunks dans PostgreSQL.

Sélection du Modèle d'Embedding

La performance de notre moteur RAG dépend de manière critique de la qualité des embeddings. Un bon modèle d'embedding doit être capable de capturer la nuance sémantique du jargon technique de la construction en français. Notre sélection est donc basée sur des données objectives issues du **Massive Text Embedding Benchmark (MTEB) for French**, une initiative communautaire visant à évaluer les modèles sur des tâches en français.[32]

- **Analyse :** Les benchmarks montrent que les modèles multilingues de grande taille, comme multilingual-e5-large et Cohere-multilingual-v3, obtiennent d'excellents résultats en français, souvent supérieurs à ceux de modèles spécifiquement francophones plus anciens.[33] L'utilisation de modèles propriétaires via API, comme text-embedding-3-large d'OpenAI ou celui de Cohere, poserait un problème de souveraineté des données, car les textes des documents seraient envoyés à des serveurs tiers.
- **Décision :** Nous sélectionnerons le modèle **intfloat/multilingual-e5-large**. C'est un modèle open-source de premier plan qui a démontré d'excellentes performances sur le benchmark MTEB-Fr. Il sera hébergé sur notre propre infrastructure R2i, garantissant que les données de nos clients ne quittent jamais notre environnement contrôlé et souverain.

Tableau 4 : Analyse Comparative des Modèles d'Embedding pour le Français

Ce tableau justifie le choix du modèle d'embedding en se basant sur la performance, la dimension et la capacité d'auto-hébergement.

Modèle	Type	Performance MTEB-Fr (Rang Relatif)	Dimension	Souveraineté (Auto-hébergeable)	Conclusion
intfloat/multilingual-e5-large	Open-Source	Très Élevée (Top 3 multilingue) [33]	1024	**Oui**	**Choix Principal**
voyage-lite-02-instruct	Propriétaire (API)	Élevée (spécialisé)	1024	Non	Alternative performante, mais non souveraine.
Cohere-multilingual-v3	Propriétaire (API)	Élevée (Top 4 multilingue) [33]	1024	Non	Alternative performante, mais non souveraine.

OpenAI text-embedding-3-large	Propriétaire (API)	Non classé sur MTEB-Fr, mais performant en général	3072	Non	Problème de souveraineté, dimension élevée.

Pipeline de Requête (Récupération et Génération)

Ce processus est déclenché lorsqu'un utilisateur pose une question dans l'interface de l'IA conversationnelle :

1. **Requête Utilisateur :** La question de l'utilisateur (par exemple, "Quelle est la norme de résistance au feu pour les portes des cages d'escalier?") est reçue par notre API GraphQL.
2. **Embedding de la Requête :** La question est immédiatement transformée en un vecteur d'embedding en utilisant le même modèle multilingual-e5-large.
3. **Récupération des Chunks :** Une requête est exécutée sur pgvector. Cette requête recherche dans la table document_chunks les N chunks dont les vecteurs sont les plus proches (en utilisant la similarité cosinus) du vecteur de la question, tout en filtrant pour ne rechercher que dans les documents du projet en cours.
4. **Construction du Prompt :** Un prompt structuré et précis est assemblé dynamiquement. Ce prompt inclura les chunks de texte récupérés comme contexte et une instruction claire pour le Grand Modèle de Langage (LLM) : *"Tu es un assistant spécialisé dans le domaine de la construction au Québec. Réponds à la question suivante en te basant exclusivement sur les extraits de documents fournis ci-dessous. Cite tes sources en indiquant le nom du fichier et la page si disponible. Si l'information n'est pas présente dans les extraits, réponds 'Je ne sais pas'."* Cette instruction est cruciale pour éviter les "hallucinations" du modèle.[17]
5. **Génération de la Réponse :** Le prompt complet est envoyé à un LLM. Pour le lancement du produit (MVP), nous utiliserons l'API de **GPT-4** d'OpenAI, en raison de ses capacités de raisonnement et de synthèse supérieures, qui garantiront une expérience utilisateur de haute qualité dès le départ.
6. **Feuille de Route vers la Souveraineté du LLM :** Conscient que l'utilisation de l'API d'OpenAI représente une dépendance et un transfert de données (le prompt) hors de notre infrastructure, une initiative parallèle sera lancée dès le début du projet. Cette initiative visera à évaluer, fine-tuner et déployer un modèle de langage open-source performant (comme Mixtral ou Llama 3) sur nos propres serveurs chez R2i. L'objectif à moyen terme est d'atteindre une souveraineté complète de l'ensemble du pipeline d'IA, de l'embedding à la génération.

Conclusion : Une Architecture au Service de la Stratégie

Le plan d'implémentation technique détaillé pour la plateforme InfraQC n'est pas une simple sélection de technologies à la mode. Il s'agit d'une architecture délibérée, où chaque composant, chaque framework et chaque protocole a été choisi pour matérialiser directement et de manière tangible nos trois avantages concurrentiels sur le marché québécois de la construction publique.

1. **La Confiance par la Souveraineté :** En faisant le choix stratégique d'une pile technologique 100% canadienne, ancrée sur le cloud souverain de R2i et unifiée autour de PostgreSQL avec l'extension pgvector, nous éliminons un risque juridique et un fardeau administratif majeurs pour nos clients du secteur public. Nous ne nous contentons pas de respecter la Loi 25 ; nous la transformons en notre principal argument de vente, offrant une tranquillité d'esprit que nos concurrents internationaux ne peuvent garantir.[11]
2. **L'Adoption par la Fiabilité :** En adoptant une architecture "Offline-First" pour notre Progressive Web App, avec une gestion rigoureuse des données locales dans IndexedDB et une stratégie de synchronisation robuste utilisant la Background Sync API, nous garantissons une expérience utilisateur qui fonctionne de manière transparente et fiable dans les conditions réelles et souvent difficiles des chantiers de construction. Cette fiabilité inébranlable est la clé pour gagner la confiance et l'adhésion des utilisateurs sur le terrain, assurant ainsi une adoption rapide et durable de notre plateforme.
3. **La Différenciation par l'Intelligence :** En intégrant un moteur de Recherche Augmentée par Récupération (RAG) souverain au cœur même de notre architecture de données, nous élevons InfraQC bien au-delà d'un simple outil de gestion documentaire. Nous le transformons en un assistant de projet intelligent, capable de fournir des réponses précises, contextuelles et sourcées à partir de la documentation complexe des projets. Cette capacité à extraire de la valeur et de la connaissance à partir des données existantes crée une valeur ajoutée unique et un avantage concurrentiel difficilement imitable.[3]

En somme, cette architecture n'est pas une fin en soi, mais le moyen par lequel InfraQC réalisera sa mission. Elle constitue la fondation technique solide sur laquelle nous bâtirons notre succès, en offrant une solution non seulement performante, sécurisée et évolutive, mais surtout, une solution parfaitement alignée avec les besoins uniques, les contraintes réglementaires et les aspirations du secteur de la construction au Québec.

Sources des citations

1. Guide d'accompagnement – Réaliser une évaluation des facteurs relatifs à la vie privée - Commission d'accès à l'information du Québec, consulté le juillet 2, 2025, https://www.cai.gouv.qc.ca/uploads/pdfs/CAI_GU_EFVP.pdf
2. Architecture Technologique pour une Plateforme SaaS de Construction au Québec _ Meilleures Pratiques 2025.pdf
3. A Complete Guide to Retrieval-Augmented Generation - Domo, consulté le juillet

2, 2025, https://www.domo.com/blog/a-complete-guide-to-retrieval-augmented-generation
4. Évaluation des facteurs relatifs à la vie privée - Gouvernement du Québec, consulté le juillet 2, 2025, https://www.quebec.ca/gouvernement/travailler-gouvernement/travailler-fonction-publique/services-employes-etat/conformite/protection-des-renseignements-personnels/evaluation-facteurs-relatifs-vie-privee
5. Loi 25 - Démystifier l'évaluation des facteurs de risques à la vie privée : De nouveaux outils de la CAI | Canada - Norton Rose Fulbright, consulté le juillet 2, 2025, https://www.nortonrosefulbright.com/fr-ca/centre-du-savoir/publications/2793bafe/loi-25-demystifier-levaluation-des-facteurs-de-risques-a-la-vie-privee-de-nouveaux-outils-de-la-cai
6. Seeking other's experience related to getting up and running on Procore, consulté le juillet 2, 2025, https://procoretechnologies.my.site.com/s/question/0D58V00008RWgRbSAL/seeking-others-experience-related-to-getting-up-and-running-on-procore
7. Installation & Licensing Forum - Autodesk Community, consulté le juillet 2, 2025, https://forums.autodesk.com/t5/installation-licensing-forum/bd-p/installation-licensing-forum-en
8. Privacy Notice - Procore Legal Terms and Policies | Procore, consulté le juillet 2, 2025, https://www.procore.com/legal/privacy
9. Availability | Autodesk Trust Center, consulté le juillet 2, 2025, https://www.autodesk.com/trust/availability
10. Data Sovereignty in Canada: Keeping data sovereign & secure in 2025 - R2I, consulté le juillet 2, 2025, https://www.r2i.ca/en/article/data-sovereignty-in-canada-keeping-data-sovereign-secure-in-2025/
11. Cloud sécurisé et performant - 100% Canadien - Edwin – le cloud de R2i, consulté le juillet 2, 2025, https://www.r2i.ca/edwin-cloud-r2i/
12. Edwin, R2i's cloud, consulté le juillet 2, 2025, https://www.r2i.ca/en/edwin-r2i-sovereign-cloud/
13. Sécurité - Clever Cloud, consulté le juillet 2, 2025, https://www.clever-cloud.com/fr/security/
14. Understanding FastAPI: Building Production-Grade Asynchronous Applications with MCP, consulté le juillet 2, 2025, https://rileylearning.medium.com/understanding-fastapi-building-production-grade-asynchronous-applications-with-mcp-96d392535467
15. Python in the Backend in 2025: Leveraging Asyncio and FastAPI for High-Performance Systems - Nucamp Coding Bootcamp, consulté le juillet 2, 2025, https://www.nucamp.co/blog/coding-bootcamp-backend-with-python-2025-python-in-the-backend-in-2025-leveraging-asyncio-and-fastapi-for-highperformance-systems

16. Pinecone? Milvus? PgVector Is 70% Faster and Cheaper and Open Source - Medium, consulté le juillet 2, 2025, https://medium.com/@Erik_Milosevic/pinecone-milvus-pgvector-is-70-faster-and-cheaper-and-open-source-3d051a9848ac
17. Build a Retrieval Augmented Generation (RAG) App: Part 1 - Python LangChain, consulté le juillet 2, 2025, https://python.langchain.com/docs/tutorials/rag/
18. FastAPI and React in 2025 | www.joshfinnie.com, consulté le juillet 2, 2025, https://www.joshfinnie.com/blog/fastapi-and-react-in-2025/
19. Offline and background operation - Progressive web apps | MDN, consulté le juillet 2, 2025, https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Guides/Offline_and_background_operation
20. js13kGames: Making the PWA work offline with service workers - Progressive web apps, consulté le juillet 2, 2025, https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Tutorials/js13kGames/Offline_Service_workers
21. Yours insanely, Offline First. One of the important aspects of PWA is... | by Jebin | Medium, consulté le juillet 2, 2025, https://medium.com/@bvjebin/yours-insanely-offline-first-3b946e526cc1
22. Pgvector vs Pinecone: How to Choose Vector Database - Artoon Solutions, consulté le juillet 2, 2025, https://artoonsolutions.com/pgvector-vs-pinecone/
23. PostgreSQL vector search guide: Everything you need to know about pgvector - Northflank, consulté le juillet 2, 2025, https://northflank.com/blog/postgresql-vector-search-guide-with-pgvector
24. pgvector: The Critical PostgreSQL Component for Your Enterprise AI Strategy - Percona, consulté le juillet 2, 2025, https://www.percona.com/blog/pgvector-the-critical-postgresql-component-for-your-enterprise-ai-strategy/
25. Most Popular Vector Databases You Must Know in 2025 - Dataaspirant, consulté le juillet 2, 2025, https://dataaspirant.com/popular-vector-databases/
26. Pgvector vs. Pinecone: Vector Database Performance and Cost Comparison - TimescaleDB, consulté le juillet 2, 2025, https://www.tigerdata.com/blog/pgvector-vs-pinecone
27. Pgvector vs. Qdrant: Open-Source Vector Database Comparison | TigerData - TimescaleDB, consulté le juillet 2, 2025, https://www.tigerdata.com/blog/pgvector-vs-qdrant
28. Qdrant vs pgvector: Vector Database Decision Guide | by M K Pavan Kumar - Medium, consulté le juillet 2, 2025, https://medium.com/@manthapavankumar11/qdrant-vs-pgvector-vector-database-decision-guide-1db7d90850cb
29. Optimize generative AI applications with pgvector indexing: A deep dive into IVFFlat and HNSW techniques | AWS Database Blog, consulté le juillet 2, 2025, https://aws.amazon.com/blogs/database/optimize-generative-ai-applications-with-pgvector-indexing-a-deep-dive-into-ivfflat-and-hnsw-techniques/
30. Faster similarity search performance with pgvector indexes | Google Cloud Blog,

consulté le juillet 2, 2025, https://cloud.google.com/blog/products/databases/faster-similarity-search-performance-with-pgvector-indexes
31. pgvector, a guide for DBA – Part2 indexes - dbi services, consulté le juillet 2, 2025, https://www.dbi-services.com/blog/pgvector-a-guide-for-dba-part2-indexes/
32. Extending the Massive Text Embedding Benchmark to French: the datasets - Hugging Face, consulté le juillet 2, 2025, https://huggingface.co/blog/lyon-nlp-group/french-mteb-datasets
33. Extending the Massive Text Embedding Benchmark to French - arXiv, consulté le juillet 2, 2025, https://arxiv.org/html/2405.20468v1
34. Extending the Massive Text Embedding Benchmark to French | OpenReview, consulté le juillet 2, 2025, https://openreview.net/forum?id=BsZFQ-henXV