### **Cahier des Charges des Données et Schéma Entité-Relation (CDC-EDR)**

* **Version :** 1.0
* **Date :** 15 juin 2025
* **Propriétaire :** Lead Database / Architecte

### **Partie 1 : Introduction et Principes de Conception**

#### **1.1. Objectif du Document**

Ce document est la **source de vérité absolue et immuable** pour la structure de la base de données du projet SkillForge AI. Il sert de plan de construction détaillé et de référence unique pour tous les développeurs et architectes interagissant avec les données.

Son objectif est de garantir les trois piliers fondamentaux de la gestion de nos données :

1. **Intégrité :** S'assurer que les données sont exactes, valides et conformes aux règles métier, en utilisant des contraintes, des types et des relations stricts.
2. **Cohérence :** Garantir que tous les services et composants de la plateforme partagent une compréhension et une définition uniques des données, éliminant ainsi toute ambiguïté.
3. **Performance :** Concevoir une structure et une stratégie d'indexation qui permettent un accès efficace et rapide aux données, soutenant une expérience utilisateur réactive.

#### **1.2. Audience Cible**

Ce document est destiné à un public technique qui conçoit, implémente ou consomme les données de la plateforme.

* **Développeurs Back-End :** C'est leur référence principale pour créer les modèles ORM (SQLAlchemy), écrire les requêtes et comprendre les relations entre les entités.
* **Développeurs IA/ML :** Pour comprendre la structure des données qu'ils doivent analyser, enrichir (via les vecteurs) ou utiliser pour l'entraînement de modèles.
* **Architectes Logiciels :** Pour avoir une vision claire et à jour du modèle de données lors de la planification d'évolutions de l'architecture globale.
* **Administrateurs de Base de Données (DBA) :** Pour les tâches de maintenance, d'optimisation des performances, de sauvegarde et de sécurité.

#### **1.3. Base de Données Cible**

La technologie de base de données sélectionnée pour le projet est la suivante, et tous les schémas décrits ici sont conçus pour elle.

* **Système de Gestion de Base de Données (SGBDR) :** PostgreSQL, version 16 ou supérieure.
* **Extensions Requises :** L'instance de base de données doit impérativement avoir les extensions suivantes activées :
  + **pgvector :** Pour le stockage et l'indexation des vecteurs sémantiques (embeddings).
  + **uuid-ossp :** Pour la génération de clés primaires de type UUID au niveau de la base de données via la fonction uuid\_generate\_v4().

#### **1.4. Conventions de Nommage Obligatoires**

Une convention de nommage stricte et uniforme est appliquée à tous les objets de la base de données pour garantir la lisibilité et la prévisibilité du schéma. Ces règles ne sont pas optionnelles.

| Type d'Objet | Convention | Justification | Exemple(s) |
| --- | --- | --- | --- |
| Tables | snake\_case, au pluriel | Convention standard pour PostgreSQL, facilite la lecture et la distinction entre les tables et les autres objets. | users, projects, project\_assignments |
| Colonnes | snake\_case | Cohérence avec le nommage des tables. | first\_name, created\_at, project\_title |
| Clés Primaires (PK) | Toujours nommées id | Standard simple et universel. Le type est toujours UUID. | id UUID PRIMARY KEY |
| Clés Étrangères (FK) | {nom\_table\_singulier}\_id | Explicite et sans ambiguïté sur la table et la colonne référencées. | user\_id, project\_id, deliverable\_id |
| Index | ix\_{nom\_table}\_{nom\_colonnes} | Le préfixe ix\_ identifie immédiatement l'objet comme un index. Le reste du nom est descriptif. | ix\_users\_email, ix\_messages\_conversation\_id |
| Contraintes d'Unicité | uq\_{nom\_table}\_{nom\_colonnes} | Le préfixe uq\_ identifie immédiatement l'objet comme une contrainte d'unicité. | uq\_users\_email |
| Types Énumérés | {nom\_table}\_{nom\_colonne}\_enum | Permet d'identifier clairement à quelle colonne et quelle table le type est principalement associé. | project\_status\_enum |

### **Partie 2 : Schéma Entité-Relation (ERD) Conceptuel**

Cette section fournit une description de haut niveau du schéma de la base de données. L'objectif est de visualiser les entités principales et leurs relations cardinales avant de plonger dans le détail de chaque table et de chaque colonne dans la partie suivante.

#### **2.1. Description Textuelle du Diagramme**

La description suivante sert de base à la génération d'un diagramme Entité-Relation visuel.

##### **A. Les Entités Fondamentales**

Le système s'articule autour des entités principales suivantes :

* **users :** L'entité centrale qui représente toute personne ou organisation interagissant avec le système. Chaque utilisateur a un rôle (apprenant, entreprise, admin) qui définit ses capacités.
* **projects :** Représente une mission ou une tâche proposée par une entreprise. C'est l'objet principal autour duquel se déroule l'apprentissage.
* **deliverables :** Représente le travail soumis par un apprenant en réponse à un projet.
* **evaluations :** Contient le résultat structuré de l'analyse d'un livrable par un agent IA.
* **portfolios :** La vitrine publique qui expose les projets réussis et les compétences d'un apprenant.
* **skills :** Une table de référence pour les compétences techniques (ex: 'Python', 'FastAPI', 'NLP'), qui peuvent être associées aux projets et aux portfolios.
* **conversations :** Le conteneur qui regroupe les messages échangés entre les participants dans le cadre d'un projet.
* **messages :** Une unité de communication individuelle (un message texte avec une éventuelle pièce jointe) au sein d'une conversation.
* **notifications :** Un enregistrement d'une notification (in-app) destinée à un utilisateur spécifique.

##### **B. Les Relations et Cardinalités**

Les entités ci-dessus sont connectées par les relations suivantes :

1. **Relation users ↔ projects (Propriété et Assignation) :**
   * Un user (de rôle 'entreprise') peut posséder **plusieurs** projects. Un project appartient à **un seul** user propriétaire. C'est une relation **One-to-Many**, implémentée par une clé étrangère owner\_id dans la table projects.
   * Un user (de rôle 'apprenant') peut être assigné à **plusieurs** projects. Un project peut avoir **plusieurs** apprenants assignés. C'est une relation **Many-to-Many**, implémentée via une table de liaison project\_assignments (contenant user\_id et project\_id).
2. **Relation projects → deliverables :**
   * Un project peut avoir **plusieurs** deliverables soumis. Un deliverable est lié à **un seul** project.
   * C'est une relation **One-to-Many**, implémentée par une clé étrangère project\_id dans la table deliverables. La table deliverables contient aussi une clé user\_id pour identifier l'auteur.
3. **Relation deliverables ↔ evaluations :**
   * Un deliverable ne peut avoir qu'**une seule et unique** evaluation. Une evaluation correspond à **un seul et unique** deliverable.
   * C'est une relation **One-to-One**, implémentée par une clé étrangère deliverable\_id dans la table evaluations, sur laquelle une contrainte UNIQUE est appliquée.
4. **Relation users ↔ portfolios :**
   * Un user (de rôle 'apprenant') ne possède qu'**un seul et unique** portfolio.
   * C'est une relation **One-to-One**, implémentée par une clé étrangère user\_id dans la table portfolios, sur laquelle une contrainte UNIQUE est appliquée.
5. **Relation portfolios ↔ projects (Contenu du Portfolio) :**
   * Un portfolio peut exposer **plusieurs** projects complétés avec succès. Un project peut, en théorie, être exposé dans **plusieurs** portfolios (si plusieurs apprenants ont collaboré).
   * C'est une relation **Many-to-Many**, implémentée via une table de liaison portfolio\_items (contenant portfolio\_id, project\_id, et potentiellement le résumé et les compétences spécifiques à cette entrée de portfolio).
6. **Relation avec les skills :**
   * Un project peut requérir **plusieurs** skills. Une skill peut être requise par **plusieurs** projets. C'est une relation **Many-to-Many**, implémentée via une table de liaison project\_skills.
   * Un portfolio peut mettre en avant **plusieurs** skills. C'est une relation **Many-to-Many**, implémentée via une table de liaison portfolio\_skills.
7. **Relations du Système de Messagerie :**
   * Une conversation est composée de **plusieurs** messages. Un message appartient à **une seule** conversation. C'est une relation **One-to-Many**, implémentée par une clé étrangère conversation\_id dans la table messages.
   * Une conversation implique **plusieurs** users (participants). Un user peut participer à **plusieurs** conversations. C'est une relation **Many-to-Many**, implémentée via une table de liaison conversation\_participants.
8. **Relation users → notifications :**
   * Un user peut recevoir **plusieurs** notifications. Une notification est destinée à **un seul** user.
   * C'est une relation **One-to-Many**, implémentée par une clé étrangère user\_id dans la table notifications.

### **Partie 3 : Dictionnaire de Données Détaillé**

Cette section fournit la description physique et exhaustive de chaque table du schéma de la base de données PostgreSQL. Le respect des types et des contraintes définis ici est impératif pour garantir l'intégrité des données.

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| id | UUID | PRIMARY KEY, DEFAULT uuid\_generate\_v4() | Identifiant unique de l'utilisateur. |
| email | VARCHAR(255) | NOT NULL, UNIQUE | Adresse e-mail de l'utilisateur, utilisée pour la connexion. |
| hashed\_password | VARCHAR(255) | NOT NULL | Mot de passe de l'utilisateur après avoir été haché par bcrypt. |
| full\_name | VARCHAR(255) | NOT NULL | Nom complet de l'utilisateur ou nom de la personne de contact. |
| role | user\_role\_enum | NOT NULL | Rôle de l'utilisateur dans la plateforme ('apprenant', 'entreprise', 'admin'). |
| avatar\_url | TEXT | NULL | URL vers l'image de profil stockée sur GCS. |
| bio | TEXT | NULL | Courte biographie de l'utilisateur. |
| company\_name | VARCHAR(255) | NULL | Nom de l'entreprise (uniquement pour le rôle 'entreprise'). |
| is\_active | BOOLEAN | NOT NULL, DEFAULT true | Permet de désactiver un compte sans le supprimer (soft delete). |
| created\_at | TIMESTAMPTZ | NOT NULL, DEFAULT now() | Timestamp de la création du compte. |
| updated\_at | TIMESTAMPTZ | NOT NULL, DEFAULT now() | Timestamp de la dernière mise à jour du compte. |

#### **Table : company\_profiles**

#### **Description : Stocke les informations spécifiques aux entités de type 'entreprise'. Relation One-to-One avec users.**

#### 

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| id | UUID | PRIMARY KEY, DEFAULT uuid\_generate\_v4() | Identifiant unique du profil entreprise. |
| user\_id | UUID | NOT NULL, UNIQUE, FOREIGN KEY REFERENCES users(id) | Utilisateur (personne de contact) associé à cette entreprise. |
| website\_url | TEXT | NULL | URL du site web de l'entreprise. |
| siret\_number | VARCHAR(14) | NULL | Numéro SIRET pour la vérification (marché français). |
| address | TEXT | NULL | Adresse postale de l'entreprise. |
| validation\_status | company\_validation\_status\_enum | NOT NULL, DEFAULT 'PENDING' | Statut de la validation du compte par les admins. |

#### 

#### 

#### 

#### **Table : projects**

**Description :** Stocke les informations relatives aux projets soumis par les entreprises.

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| id | UUID | PRIMARY KEY, DEFAULT uuid\_generate\_v4() | Identifiant unique du projet. |
| owner\_id | UUID | NOT NULL, FOREIGN KEY REFERENCES users(id) | ID de l'utilisateur (entreprise) propriétaire du projet. |
| title | VARCHAR(255) | NOT NULL | Titre du projet. |
| description | TEXT | NOT NULL | Description complète et brief du projet. |
| status | project\_status\_enum | NOT NULL, DEFAULT 'DRAFT' | Statut actuel du projet. |
| embedding | vector(384) | NULL | Vecteur sémantique de la description pour la recherche de similarité. |
| created\_at | TIMESTAMPTZ | NOT NULL, DEFAULT now() | Timestamp de la création du projet. |
| updated\_at | TIMESTAMPTZ | NOT NULL, DEFAULT now() | Timestamp de la dernière mise à jour. |
| completed\_at | TIMESTAMPTZ | NULL | Timestamp de la complétion du projet. |

#### **Table : skills**

**Description :** Table de référence (lookup table) pour les compétences techniques.

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| id | UUID | PRIMARY KEY, DEFAULT uuid\_generate\_v4() | Identifiant unique de la compétence. |
| name | VARCHAR(100) | NOT NULL, UNIQUE | Nom de la compétence (ex: 'Python', 'FastAPI'). |

#### **Table : project\_skills**

**Description :** Table de liaison pour la relation Many-to-Many entre projects et skills.

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| project\_id | UUID | PRIMARY KEY, FOREIGN KEY REFERENCES projects(id) | ID du projet. |
| skill\_id | UUID | PRIMARY KEY, FOREIGN KEY REFERENCES skills(id) | ID de la compétence. |

#### **Table : project\_assignments**

**Description :** Table de liaison pour la relation Many-to-Many entre users (apprenants) et projects.

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| user\_id | UUID | PRIMARY KEY, FOREIGN KEY REFERENCES users(id) | ID de l'apprenant assigné. |
| project\_id | UUID | PRIMARY KEY, FOREIGN KEY REFERENCES projects(id) | ID du projet concerné. |
| assigned\_at | TIMESTAMPTZ | NOT NULL, DEFAULT now() | Timestamp de l'assignation. |

#### **Table : milestones**

#### **Description : Définit les étapes ou jalons clés pour la réalisation d'un projet.**

#### 

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| id | UUID | PRIMARY KEY, DEFAULT uuid\_generate\_v4() | Identifiant unique du jalon. |
| project\_id | UUID | NOT NULL, FOREIGN KEY REFERENCES projects(id) | Projet auquel ce jalon appartient. |
| title | VARCHAR(255) | NOT NULL | Titre du jalon (ex: "Étape 1: Analyse des données"). |
| description | TEXT | NULL | Description détaillée de ce qui est attendu pour ce jalon. |
| due\_date | TIMESTAMPTZ | NULL | Date d'échéance indicative pour ce jalon. |
| status | milestone\_status\_enum | NOT NULL, DEFAULT 'PENDING' | Statut du jalon (ex: 'PENDING', 'COMPLETED'). |
| display\_order | INTEGER | NOT NULL, DEFAULT 0 | Ordre d'affichage des jalons dans un projet. |

#### 

#### **Table : deliverables**

**Description :** Stocke les soumissions de travaux (livrables) par les apprenants pour un projet.

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| id | UUID | PRIMARY KEY, DEFAULT uuid\_generate\_v4() | Identifiant unique du livrable. |
| project\_id | UUID | NOT NULL, FOREIGN KEY REFERENCES projects(id) | Projet pour lequel ce livrable est soumis. |
| user\_id | UUID | NOT NULL, FOREIGN KEY REFERENCES users(id) | Apprenant qui a soumis le livrable. |
| type | deliverable\_type\_enum | NOT NULL | (Ajout) Type de livrable ('FILE\_UPLOAD', 'GIT\_REPO\_URL', 'EXTERNAL\_URL'). |
| file\_path\_or\_url | TEXT | NOT NULL | Chemin vers le fichier sur GCS ou URL du dépôt/lien. |
| notes | TEXT | NULL | Notes optionnelles de l'apprenant sur sa soumission. |
| submitted\_at | TIMESTAMPTZ | NOT NULL, DEFAULT now() | Timestamp de la soumission. |

#### **Table : evaluations**

**Description :** Stocke le résultat de l'évaluation par l'IA d'un livrable. Relation One-to-One avec deliverables.

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| id | UUID | PRIMARY KEY, DEFAULT uuid\_generate\_v4() | Identifiant unique de l'évaluation. |
| deliverable\_id | UUID | NOT NULL, UNIQUE, FOREIGN KEY REFERENCES deliverables(id) | Livrable évalué. La contrainte UNIQUE garantit la relation 1-to-1. |
| overall\_score | NUMERIC(4, 2) | NOT NULL | Note globale sur 10.00, avec 2 décimales. |
| feedback\_data | JSONB | NOT NULL | Objet JSON contenant l'évaluation détaillée (forces, faiblesses, etc.). |
| evaluated\_at | TIMESTAMPTZ | NOT NULL, DEFAULT now() | Timestamp de la fin de l'évaluation. |

#### **Table : portfolios**

**Description :** Entité principale du portfolio d'un apprenant. Relation One-to-One avec users.

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| id | UUID | PRIMARY KEY, DEFAULT uuid\_generate\_v4() | Identifiant unique du portfolio. |
| user\_id | UUID | NOT NULL, UNIQUE, FOREIGN KEY REFERENCES users(id) | Apprenant propriétaire du portfolio. Contrainte UNIQUE pour la relation 1-to-1. |
| public\_url\_slug | VARCHAR(100) | NOT NULL, UNIQUE | Partie de l'URL lisible par l'homme (ex: "jean-dupont-ai"). |
| is\_public | BOOLEAN | NOT NULL, DEFAULT false | Contrôle la visibilité publique du portfolio. |

#### **Table : portfolio\_items**

**Description :** Représente une entrée (un projet) dans un portfolio.

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| id | UUID | PRIMARY KEY, DEFAULT uuid\_generate\_v4() | Identifiant unique de l'entrée du portfolio. |
| portfolio\_id | UUID | NOT NULL, FOREIGN KEY REFERENCES portfolios(id) | Portfolio auquel cette entrée appartient. |
| project\_id | UUID | NOT NULL, FOREIGN KEY REFERENCES projects(id) | Projet mis en avant. |
| summary | TEXT | NULL | Résumé du projet, généré par l'IA ou personnalisé. |
| display\_order | INTEGER | NOT NULL, DEFAULT 0 | Ordre d'affichage des projets dans le portfolio. |

#### **Table : portfolio\_skills**

**Description :** Table de liaison pour la relation Many-to-Many entre portfolio\_items et skills.

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| portfolio\_item\_id | UUID | PRIMARY KEY, FOREIGN KEY REFERENCES portfolio\_items(id) | ID de l'entrée de portfolio. |
| skill\_id | UUID | PRIMARY KEY, FOREIGN KEY REFERENCES skills(id) | ID de la compétence associée. |

#### **Table : conversations**

**Description :** Conteneur pour un fil de discussion entre plusieurs utilisateurs.

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| id | UUID | PRIMARY KEY, DEFAULT uuid\_generate\_v4() | Identifiant unique de la conversation. |
| project\_id | UUID | NOT NULL, FOREIGN KEY REFERENCES projects(id) | Projet auquel la conversation est rattachée. |
| created\_at | TIMESTAMPTZ | NOT NULL, DEFAULT now() | Timestamp de la création de la conversation. |

#### **Table : conversation\_participants**

**Description :** Table de liaison pour la relation Many-to-Many entre conversations et users.

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| conversation\_id | UUID | PRIMARY KEY, FOREIGN KEY REFERENCES conversations(id) | ID de la conversation. |
| user\_id | UUID | PRIMARY KEY, FOREIGN KEY REFERENCES users(id) | ID de l'utilisateur participant. |

#### **Table : messages**

**Description :** Stocke un message individuel au sein d'une conversation.

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| id | UUID | PRIMARY KEY, DEFAULT uuid\_generate\_v4() | Identifiant unique du message. |
| conversation\_id | UUID | NOT NULL, FOREIGN KEY REFERENCES conversations(id) | Conversation à laquelle le message appartient. |
| sender\_id | UUID | NOT NULL, FOREIGN KEY REFERENCES users(id) | Utilisateur qui a envoyé le message. |
| content | TEXT | NOT NULL | Contenu textuel du message. |
| attachment\_url | TEXT | NULL | URL vers une éventuelle pièce jointe sur GCS. |
| created\_at | TIMESTAMPTZ | NOT NULL, DEFAULT now() | Timestamp de l'envoi du message. |

#### **Table : notifications**

**Description :** Stocke les notifications in-app pour les utilisateurs.

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| id | UUID | PRIMARY KEY, DEFAULT uuid\_generate\_v4() | Identifiant unique de la notification. |
| user\_id | UUID | NOT NULL, FOREIGN KEY REFERENCES users(id) | Utilisateur destinataire de la notification. |
| content | TEXT | NOT NULL | Texte de la notification. |
| status | notification\_status\_enum | NOT NULL, DEFAULT 'UNREAD' | Statut de la notification (lue ou non lue). |
| link\_url | TEXT | NULL | URL de redirection lorsque l'utilisateur clique sur la notification. |
| created\_at | TIMESTAMPTZ | NOT NULL, DEFAULT now() | Timestamp de la création de la notification. |

#### **Table : audit\_logs**

**Description :** Journal d'audit pour tracer les actions sensibles effectuées sur la plateforme.

| Nom de la Colonne | Type de Données | Contraintes | Description |
| --- | --- | --- | --- |
| id | UUID | PRIMARY KEY, DEFAULT uuid\_generate\_v4() | Identifiant unique de l'entrée de log. |
| actor\_id | UUID | NULL, FOREIGN KEY REFERENCES users(id) | ID de l'utilisateur qui a effectué l'action (NULL si action système). |
| action\_type | VARCHAR(100) | NOT NULL | Type d'action effectuée (ex: 'USER\_BANNED', 'PROJECT\_VALIDATED'). |
| target\_entity | VARCHAR(100) | NULL | Entité cible de l'action (ex: 'USER', 'PROJECT'). |
| target\_id | UUID | NULL | ID de l'entité cible. |
| details | JSONB | NULL | Objet JSON contenant des détails additionnels sur l'action. |
| timestamp | TIMESTAMPTZ | NOT NULL, DEFAULT now() | Timestamp exact de l'action. |

### **Partie 4 : Vues Matérialisées pour l'Analyse**

Cette section définit les vues matérialisées qui seront créées pour répondre aux besoins d'analyse et de reporting du MVP. Elles seront rafraîchies périodiquement (ex: toutes les nuits) pour garantir des performances de lecture rapides.

#### **Vue 1 : analytics\_user\_activity**

* **Objectif :** Fournir une vue d'ensemble de l'activité de chaque utilisateur.
* **Colonnes :**
  + user\_id (UUID)
  + full\_name (VARCHAR)
  + role (user\_role\_enum)
  + project\_count\_assigned (INTEGER) : Nombre de projets auxquels un apprenant est assigné.
  + project\_count\_owned (INTEGER) : Nombre de projets qu'une entreprise a soumis.
  + deliverable\_count\_submitted (INTEGER) : Nombre de livrables soumis.
  + average\_score (NUMERIC) : Note moyenne obtenue sur tous les livrables.
* **Logique de Refresh :** Nocturne.

#### **Vue 2 : analytics\_project\_performance**

* **Objectif :** Agréger les statistiques de performance pour chaque projet.
* **Colonnes :**
  + project\_id (UUID)
  + project\_title (VARCHAR)
  + owner\_company\_name (VARCHAR)
  + status (project\_status\_enum)
  + assignment\_count (INTEGER) : Nombre d'apprenants assignés.
  + submission\_count (INTEGER) : Nombre de livrables reçus.
  + average\_score\_for\_project (NUMERIC) : Note moyenne de tous les livrables pour ce projet.
* **Logique de Refresh :** Nocturne.

#### **Vue 3 : analytics\_skill\_demand\_and\_performance**

* **Objectif :** Analyser la popularité des compétences et la performance des apprenants sur celles-ci.
* **Colonnes :**
  + skill\_id (UUID)
  + skill\_name (VARCHAR)
  + project\_count\_with\_skill (INTEGER) : Nombre de projets requérant cette compétence.
  + average\_score\_on\_skill\_projects (NUMERIC) : Note moyenne obtenue sur les projets liés à cette compétence.
* **Logique de Refresh :** Nocturne.

### **Partie 5 : Intégrité des Données et Types Personnalisés**

Cette section décrit les mécanismes de bas niveau utilisés pour renforcer les règles métier et garantir la validité des données directement dans la base de données. C'est une couche de protection supplémentaire qui s'ajoute aux validations effectuées par l'application.

#### **5.1. Contraintes de Vérification (CHECK Constraints)**

Les contraintes CHECK sont des règles qui doivent être vraies pour chaque ligne d'une table. Elles empêchent l'insertion de données invalides selon notre logique métier.

Les contraintes suivantes doivent être implémentées :

1. **Table evaluations :**
   * **Contrainte :** Le score global d'une évaluation doit impérativement être compris entre 0.00 et 10.00.
   * **Implémentation SQL :**

ALTER TABLE evaluations

ADD CONSTRAINT score\_range\_check CHECK (overall\_score >= 0.00 AND overall\_score <= 10.00);

**Table users :**

* **Contrainte :** L'adresse e-mail doit suivre un format standard.
* **Implémentation SQL :**

ALTER TABLE users

ADD CONSTRAINT email\_format\_check CHECK (email ~\* '^[A-Za-z0-9.\_+%-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$');

**Note :** Cette contrainte de format de base n'exclut pas une validation plus poussée côté application (ex: via un service de vérification d'e-mail).

#### **5.2. Types Énumérés (ENUM)**

Pour les colonnes qui ne peuvent accepter qu'un ensemble fini de valeurs prédéfinies, nous utilisons systématiquement les types ENUM natifs de PostgreSQL.

* **Justification :** L'utilisation des ENUM est obligatoire car elle offre trois avantages majeurs sur l'utilisation de VARCHAR :
  1. **Intégrité des Données :** La base de données elle-même rejette toute tentative d'insérer une valeur qui ne fait pas partie de la liste autorisée.
  2. **Efficacité du Stockage :** Un type ENUM est stocké beaucoup plus efficacement qu'une chaîne de caractères équivalente.
  3. **Auto-documentation :** Le schéma de la base de données documente de manière explicite les états possibles pour une colonne.

La liste suivante définit tous les types personnalisés à créer dans notre base de données.

1. **Type : user\_role\_enum**
   * **Utilisation :** Colonne users(role).
   * **Définition SQL :**

CREATE TYPE user\_role\_enum AS ENUM ('apprenant', 'entreprise', 'admin');

**Type : project\_status\_enum**

* **Utilisation :** Colonne projects(status).
* **Définition SQL :**

CREATE TYPE project\_status\_enum AS ENUM ('DRAFT', 'PENDING\_VALIDATION', 'ACTIVE', 'COMPLETED', 'ARCHIVED');

**Type : deliverable\_type\_enum**

* **Utilisation :** Colonne deliverables(type).
* **Définition SQL :**

CREATE TYPE deliverable\_type\_enum AS ENUM ('FILE\_UPLOAD', 'GIT\_REPO\_URL', 'EXTERNAL\_URL');

(pense aussi a specifier les type de fichier uploader, c’est à dire au niveau de l’interface si on choisi file\_upload on devrait aussi avoir une slection pour choisir le format du fichier a uploader et en fonction de chaque format on doit avoir une bloc précis)  
  
**Type : company\_validation\_status\_enum**

* **Utilisation :** Colonne company\_profiles(validation\_status).
* **Définition SQL :**

CREATE TYPE company\_validation\_status\_enum AS ENUM ('PENDING', 'VALIDATED', 'REJECTED');

**Type : milestone\_status\_enum**

* **Utilisation :** Colonne milestones(status).
* **Définition SQL :**

CREATE TYPE milestone\_status\_enum AS ENUM ('PENDING', 'IN\_PROGRESS', 'COMPLETED');

**Type : notification\_status\_enum**

* **Utilisation :** Colonne notifications(status).
* **Définition SQL :**

CREATE TYPE notification\_status\_enum AS ENUM ('UNREAD', 'READ');

### **Partie 6 : Stratégie d'Indexation**

Cette section définit la philosophie et les règles pour la création d'index dans notre base de données PostgreSQL. L'objectif est d'optimiser les performances en lecture pour les requêtes les plus fréquentes, sans pour autant pénaliser excessivement les opérations d'écriture (INSERT, UPDATE).

#### **6.1. Philosophie d'Indexation**

Les index sont des structures de données spécialisées qui permettent au moteur de la base de données de localiser les lignes beaucoup plus rapidement qu'en parcourant la table entière. Notre approche est la suivante :

* **Proactivité :** Nous créons de manière proactive les index pour tous les cas d'usage évidents (clés étrangères, colonnes de filtrage fréquentes).
* **Mesure et Itération :** La performance des requêtes sera surveillée en continu. Des index supplémentaires pourront être ajoutés à l'avenir en se basant sur l'analyse de requêtes lentes via la commande EXPLAIN ANALYZE.

#### **6.2. Index Standards et Obligatoires**

1. **Index sur Clés Primaires et Contraintes d'Unicité :**
   * **Règle :** Il n'est pas nécessaire de créer manuellement des index pour les colonnes définies comme PRIMARY KEY ou ayant une contrainte UNIQUE. PostgreSQL crée automatiquement un UNIQUE INDEX pour garantir ces contraintes.
2. **Index sur Clés Étrangères (Foreign Keys) :**
   * **Règle Absolue :** Toutes les colonnes servant de clé étrangère (se terminant par \_id) **doivent être explicitement indexées**.
   * **Justification Critique :** Cette règle est fondamentale pour deux raisons :
     1. **Performance des Jointures (JOIN) :** L'absence d'un index sur une clé étrangère force PostgreSQL à effectuer un balayage séquentiel de la table enfant lors d'une jointure, ce qui est extrêmement coûteux en performance.
     2. **Prévention des Verrous de Table :** Lors d'une opération UPDATE ou DELETE sur une clé primaire de la table parente, PostgreSQL doit vérifier qu'aucune ligne de la table enfant ne la référence. Sans index sur la clé étrangère, cette vérification peut nécessiter un verrou sur la table enfant entière, bloquant les opérations concurrentes.
   * **Exemples d'Index à Créer :** CREATE INDEX ix\_projects\_owner\_id ON projects(owner\_id);, CREATE INDEX ix\_deliverables\_project\_id ON deliverables(project\_id);, etc., pour toutes les clés étrangères.

#### **6.3. Index Personnalisés (Custom Indexes)**

En plus des index sur les clés étrangères, les index suivants doivent être créés pour accélérer les requêtes métier spécifiques.

1. **Table users :**
   * **Objectif :** Accélérer la recherche d'un utilisateur par son e-mail lors de la connexion.
   * **Implémentation SQL :**

CREATE INDEX ix\_users\_email ON users (LOWER(email));

* **Note :** L'index est créé sur LOWER(email) pour supporter des recherches insensibles à la casse de manière performante.

**Table projects :**

* **Objectif :** Filtrer rapidement les projets par leur statut.
* **Implémentation SQL :**

CREATE INDEX ix\_projects\_status ON projects (status);

**Table messages :**

* **Objectif :** Récupérer l'historique d'une conversation de manière très efficace, trié par date décroissante.
* **Implémentation SQL :**

CREATE INDEX ix\_messages\_conversation\_id\_created\_at ON messages (conversation\_id, created\_at DESC);

* **Note :** Il s'agit d'un index composite qui optimise à la fois le filtrage par conversation\_id et le tri par date.

**Table notifications :**

* **Objectif :** Récupérer rapidement toutes les notifications non lues pour un utilisateur donné.
* **Implémentation SQL :**

CREATE INDEX ix\_notifications\_user\_id\_status ON notifications (user\_id, status) WHERE status = 'UNREAD';

* + **Note :** Il s'agit d'un index partiel, qui n'indexe que les lignes correspondant à la condition WHERE. Il est extrêmement efficace et compact pour ce cas d'usage.

#### **6.4. Index Vectoriel Spécifique (pgvector)**

L'indexation des vecteurs sémantiques est un cas particulier qui nécessite un type d'index spécialisé pour la recherche de similarité.

* **Colonne Cible :** projects(embedding)
* **Type d'Index Requis :** **HNSW (Hierarchical Navigable Small World)**.
* **Justification :** HNSW est l'algorithme de pointe pour la recherche de voisins les plus proches approximative (ANN) sur des vecteurs de haute dimension. Il offre le meilleur compromis entre vitesse de recherche et précision, et est bien supérieur à un balayage séquentiel (recherche exacte) qui serait beaucoup trop lent.
* **Implémentation SQL :**

CREATE INDEX ix\_projects\_embedding\_hnsw ON projects

USING HNSW (embedding vector\_cosine\_ops);

**Note :** L'opérateur vector\_cosine\_ops indique à l'index d'être optimisé pour les calculs de distance cosinus, ce qui correspond à la métrique de similarité utilisée par notre modèle d'embedding all-MiniLM-L6-v2.

### **Partie 7 : Gestion des Versions et des Migrations**

Un schéma de base de données n'est pas statique ; il évolue avec les fonctionnalités de l'application. Cette section définit le processus formel, sécurisé et collaboratif pour gérer ces changements. L'objectif est d'éviter toute perte de données, incohérence ou interruption de service lors des mises à jour du schéma.

#### **7.1. Outil de Migration Obligatoire**

1. **Technologie :** **Alembic** est l'unique outil autorisé pour la gestion de toutes les migrations de schéma de base de données.
2. **Justification :** Alembic est l'outil de migration standard de l'écosystème SQLAlchemy. Il s'intègre parfaitement avec nos modèles ORM, permet de générer automatiquement des scripts de migration à partir des modifications du code, et gère les dépendances entre les migrations pour assurer qu'elles soient appliquées dans le bon ordre.

#### **7.2. Workflow de Migration (Processus pour les Développeurs)**

Chaque développeur qui modifie le schéma de la base de données doit impérativement suivre le processus suivant.

* **Étape 1 : Modification du Modèle Applicatif** Le développeur modifie une classe de modèle dans le code Back-End (ex: ajout d'une colonne dans un modèle SQLModel/SQLAlchemy).
* **Étape 2 : Génération du Script de Migration** Une fois le code modifié, le développeur doit générer le script de migration correspondant via la commande Alembic. Il est essentiel de fournir un message de commit clair et descriptif.

alembic revision --autogenerate -m "Ajout de la colonne 'last\_login\_ip' a la table users"

* **Étape 3 : Revue Manuelle OBLIGATOIRE du Script** C'est l'étape de sécurité la plus importante. Le développeur **doit impérativement ouvrir et relire** le fichier de migration généré (situé dans alembic/versions/). Il doit vérifier :  
  + Que les opérations dans la fonction upgrade() correspondent exactement au changement désiré.
  + Que la fonction downgrade() inverse correctement ces opérations.
  + S'il y a des opérations potentiellement destructrices (ex: DROP COLUMN, ALTER COLUMN qui tronque des données). De telles opérations nécessitent une discussion avec l'équipe et une stratégie de déploiement prudente (parfois en plusieurs étapes).
* **Étape 4 : Tests Locaux** Le développeur doit exécuter la migration sur sa base de données de développement (alembic upgrade head) et lancer la suite de tests de l'application pour s'assurer que le changement de schéma n'a introduit aucune régression.
* **Étape 5 : Commit Atomique** Le fichier de script de migration généré doit être commité dans Git **dans le même commit** que les modifications du code de modèle qui lui correspondent. Cela garantit que le code et l'état du schéma restent synchronisés.

#### **7.3. Déploiement des Migrations (Processus CI/CD)**

1. **Automatisation Exclusive :** Les migrations de base de données ne doivent **jamais** être exécutées manuellement sur les environnements de staging ou de production.
2. **Rôle du Pipeline CI/CD :** Le pipeline de déploiement (GitHub Actions) est le seul responsable de l'application des migrations.
3. **Déroulement :**
   * **Étape A : Application de la Migration :** Juste avant de déployer une nouvelle version du code de l'application, une étape du pipeline exécutera la commande alembic upgrade head. Cette commande compare les versions des migrations déjà appliquées à la base de données avec celles présentes dans le code et applique toutes les nouvelles migrations.
   * **Étape B : Déploiement de l'Application :** Uniquement si l'étape de migration réussit sans erreur, le pipeline continuera et déploiera la nouvelle version des conteneurs de l'application.
4. **Stratégie de "Rollback" :** La stratégie privilégiée est le "roll forward". En cas de problème avec une migration en production, nous ne lancerons pas de downgrade. Nous créerons et déploierons une nouvelle migration qui corrige le problème. Les scripts downgrade sont principalement destinés à un usage en environnement de développement.

### **Partie 8 : Considérations de Sécurité**

Cette section définit les mesures de sécurité qui doivent être implémentées directement au niveau de la base de données. Elle constitue une couche de défense fondamentale, complémentaire à la sécurité applicative, pour protéger l'intégrité, la confidentialité et la disponibilité de nos données.

#### **8.1. Principe du Moindre Privilège**

* **Règle Fondamentale :** Aucun utilisateur ou service applicatif ne doit disposer de plus de permissions que le minimum absolu requis pour accomplir ses fonctions légitimes.
* **Application :** Nous n'utiliserons **jamais** un super-utilisateur de base de données pour les opérations courantes de nos applications. Chaque microservice doit se connecter à la base de données en utilisant un rôle PostgreSQL dédié et distinct, avec des permissions limitées à son périmètre fonctionnel.

#### **8.2. Définition des Rôles et Permissions**

Pour chaque microservice interagissant avec la base de données, un rôle PostgreSQL dédié doit être créé.

* **Exemples de Rôles :** user\_service\_role, project\_service\_role, evaluation\_service\_role.
* **Attribution des Permissions :** Les permissions doivent être accordées de manière granulaire.  
  + **Exemple pour project\_service\_role :**
    - Ce rôle doit avoir les permissions SELECT, INSERT, UPDATE sur les tables projects, milestones, project\_skills, project\_assignments, et deliverables.
    - Il doit avoir la permission SELECT sur la table users pour vérifier les informations du propriétaire, mais **pas** UPDATE ou DELETE.
    - Il ne doit avoir **aucune permission** sur les tables evaluations, portfolios, ou messages.
  + **Exemple d'Implémentation SQL :**

-- Création du rôle

CREATE ROLE project\_service\_role LOGIN PASSWORD 'un\_mot\_de\_passe\_géré\_par\_secret\_manager';

-- Attribution des permissions

GRANT SELECT, INSERT, UPDATE, DELETE ON projects, milestones, deliverables TO project\_service\_role;

GRANT SELECT ON users, skills TO project\_service\_role;

GRANT USAGE ON SEQUENCE projects\_id\_seq, milestones\_id\_seq TO project\_service\_role;

#### **8.3. Politique de Chiffrement (Encryption)**

1. **Chiffrement en Transit (In-Transit) :**
   * **Règle :** Toutes les connexions entre les services applicatifs (Cloud Run, GKE) et l'instance Cloud SQL PostgreSQL **doivent** utiliser le protocole SSL/TLS.
   * **Implémentation :** L'instance Cloud SQL doit être configurée pour refuser les connexions non chiffrées. Les clients de base de données dans nos applications doivent être configurés pour se connecter en utilisant SSL.
2. **Chiffrement au Repos (At-Rest) :**
   * **Règle :** Toutes les données stockées sur disque, y compris les fichiers de la base de données, les sauvegardes (backups) et les réplicas, doivent être chiffrées.
   * **Implémentation :** Nous nous appuyons sur le **chiffrement au repos natif et activé par défaut** de Google Cloud SQL. Google gère les clés de chiffrement, ce qui est une pratique standard et sécurisée.

#### **8.4. Stratégie d'Audit**

Pour assurer la traçabilité des actions et faciliter les investigations en cas d'incident de sécurité, un audit détaillé des opérations de la base de données est nécessaire.

* **Outil :** L'extension PostgreSQL **pgaudit** doit être activée sur notre instance Cloud SQL.
* **Configuration de l'Audit :** La journalisation doit être configurée pour enregistrer au minimum les événements suivants :
  + **DDL (Data Definition Language) :** Toutes les commandes CREATE, ALTER, DROP pour tracer toute modification apportée à la structure de la base de données.
  + **ROLE :** Toutes les commandes de gestion des permissions (GRANT, REVOKE, CREATE ROLE).
  + **READ et WRITE sur les tables sensibles :** Les opérations SELECT, INSERT, UPDATE, DELETE doivent être journalisées pour les tables critiques comme users et audit\_logs.
* **Destination des Logs :** Les journaux d'audit produits par pgaudit doivent être exportés vers **Google Cloud Logging** pour une conservation à long terme, une analyse centralisée et la mise en place d'alertes.

### **Conclusion du Document**

Ce Cahier des Charges des Données constitue le plan directeur et la référence unique pour la base de données de SkillForge AI. De la structure conceptuelle aux détails physiques, en passant par les stratégies d'intégrité, de performance, de migration et de sécurité, il fournit un cadre complet et rigoureux. L'adhésion stricte à ce document par toutes les équipes techniques est la condition essentielle pour garantir que nos données, l'actif le plus précieux du projet, soient gérées avec le plus haut niveau de qualité et de professionnalisme.