### **Guide d'Architecture Générale (GAG) - SkillForge AI**

* **Version** : 1.0
* **Date** : 7 juin 2025
* **Statut** : Adopté
* **Auteur** : Gemini, Expert en Architecture Projet

### **1. Introduction et Objectifs**

Ce document a pour mission de définir les fondations, les frontières et les règles d'interaction de tous les composants logiciels du projet SkillForge AI. Il n'a pas vocation à détailler l'implémentation de chaque composant, mais à garantir que leur assemblage forme un tout cohérent, performant, sécurisé et maintenable.

**Objectifs :**

* **Assurer la Cohérence** : Garantir que tous les développements suivent une vision technique unifiée.
* **Définir les Frontières** : Clarifier les responsabilités de chaque service et de chaque équipe.
* **Guider le Développement Parallèle** : Permettre aux équipes Front-End, Back-End et IA de travailler de manière autonome et efficace en se basant sur des contrats clairs.
* **Faciliter l'Intégration (Onboarding)** : Servir de point d'entrée unique pour tout nouveau développeur rejoignant le projet.

### **2. Principes Architecturaux Fondamentaux**

L'architecture de SkillForge AI repose sur les sept principes non négociables suivants :

1. **Approche Monorepo** : Tout le code source (Front-End, Back-End, Agents IA, Infrastructure) est hébergé dans un unique dépôt Git. Cela garantit des dépendances cohérentes, des modifications atomiques et des pipelines CI/CD unifiés.
2. **Architecture Microservices** : Le Back-End est décomposé en services indépendants, chacun centré sur un domaine métier spécifique (utilisateurs, projets...). Cela favorise la séparation des préoccupations, la résilience et la scalabilité indépendante des composants.
3. **Communication Asynchrone et Événementielle (Event-Driven)** : Pour les actions longues ou les processus transverses (ex: évaluation d'un livrable), la communication entre services se fait via un broker de messages. Les services émettent des événements sans attendre de réponse, ce qui assure le découplage et la performance.
4. **API First** : Le contrat de l'API REST entre le Front-End et le Back-End est la première chose à être définie. Il est formel, versionné et sert de base de travail pour les deux équipes. L'implémentation doit respecter ce contrat à la lettre.
5. **Services Sans État (Stateless)** : Tous les services Back-End et les Agents IA doivent être sans état. Toute information persistante doit être stockée dans une base de données (PostgreSQL) ou un cache externe (Redis). Cela est fondamental pour la scalabilité horizontale et la résilience.
6. **Infrastructure en tant que Code (Infrastructure as Code - IaC)** : Toute l'infrastructure sur Google Cloud Platform (bases de données, clusters, services...) est définie sous forme de code (ex: Terraform), versionnée dans le monorepo et déployée via les pipelines CI/CD. Aucune ressource ne doit être créée manuellement en production.
7. **Sécurité dès la Conception (Security by Design)** : La sécurité n'est pas une surcouche, mais un prérequis. Chaque composant doit intégrer les principes de sécurité (authentification, autorisation, validation des entrées) dès sa conception.

### **3. Diagramme d'Architecture de Haut Niveau**

L'architecture de SkillForge AI s'articule comme suit :

*(Note: l'image est une représentation visuelle du texte qui suit)*

1. **L'Utilisateur** (Apprenant, Entreprise, Admin) interagit exclusivement avec l'**Application Front-End (React)** via son navigateur web.
2. L'**Application Front-End** est une Single-Page Application (SPA) qui communique de manière **synchrone** avec le système via un point d'entrée unique : l'**API Gateway**.
3. L'**API Gateway** (ex: Google Cloud API Gateway) gère l'authentification, le routage et le "rate limiting". Elle redirige les requêtes HTTP vers les **Microservices Back-End (FastAPI)** appropriés.
4. Les **Microservices Back-End** (user-service, project-service, etc.) contiennent la logique métier. Ils sont sans état et s'exécutent dans des conteneurs sur **Google Kubernetes Engine (GKE)**.
5. Pour toute persistance de données, les microservices communiquent avec la **Base de Données unique (PostgreSQL + pgvector)**.
6. Pour le stockage de fichiers (livrables, images de profil), les services utilisent le **Stockage Objet (Google Cloud Storage)**.
7. Lorsqu'une action longue est nécessaire (ex: un apprenant soumet un livrable), le microservice concerné (project-service) ne la traite pas directement. Il publie un **événement** (un message) dans le **Broker de Messages (Redis Pub/Sub)**.
8. Les **Agents IA (Workers Python)** sont des services indépendants qui **s'abonnent** aux événements du Broker de Messages. Quand un événement pertinent est publié, l'agent concerné (evaluation-agent) le consomme, exécute sa tâche (analyse, évaluation), puis peut à son tour publier un nouvel événement (ex: evaluation.completed) ou mettre à jour la base de données. Ils s'exécutent sur une infrastructure serverless (**Google Cloud Run**).

### 

### **4. Définition des Interfaces et Contrats**

Ceci est le chapitre le plus important. Il définit les règles de communication entre les composants.

#### **4.1. Communication Synchrone : API REST**

* **Contrat** : La spécification **OpenAPI 3.0** est la seule et unique source de vérité pour le contrat d'API.
* **Source de Vérité** : Le fichier openapi.json généré automatiquement par le service Back-End principal et accessible via l'endpoint /openapi.json. Les équipes Front-End DOIVENT l'utiliser pour générer leurs clients API.
* **Authentification** : Toutes les requêtes vers des endpoints sécurisés DOIVENT contenir un header Authorization: Bearer <JWT>. Le JWT (JSON Web Token) doit contenir au minimum les claims : sub (ID de l'utilisateur), role (apprenant, entreprise, admin), exp (timestamp d'expiration).
* **Versioning** : Le versioning se fait par l'URL. Toutes les routes de l'API doivent être préfixées par /api/v1/.
* **Structure des Réponses** :
  + **Succès (2xx)** : Le corps de la réponse contient directement les données demandées au format JSON.
  + **Erreur (4xx, 5xx)** : Le corps de la réponse DOIT être un objet JSON respectant la structure suivante :

{

"error": {

"code": "CODE\_ERREUR\_STANDARDISÉ",

"message": "Un message d'erreur clair destiné au développeur."

}

}

#### **4.2. Communication Asynchrone : Événementielle**

* **Technologie** : Redis Pub/Sub.
* **Format des Messages** : Les payloads des messages DOIVENT être au format JSON.
* **Contrat** : Le contrat d'un événement (la structure de son payload) est défini et documenté dans le CDC du service qui l'émet.
* **Convention de Nommage des Topics** : Les noms des topics DOIVENT suivre la convention domaine.ressource.verbe\_au\_passé\_participe.
  + Exemple 1 : L'entreprise soumet un nouveau projet. Le project-service publie dans le topic project.project.submitted.
  + Exemple 2 : L'agent d'évaluation a terminé son travail. Il publie dans evaluation.result.generated.

### **5. Description et Liens vers les Cahiers des Charges Spécifiques (Les Spokes)**

Ce GAG est le "moyeu". Voici les "rayons" (spokes), qui sont les documents détaillés pour chaque domaine.

* **Titre du Document** : CDC Front-End  
  + **Propriétaire** : Lead Dev Front-End
  + **Mission** : Détailler l'architecture de l'application React, les conventions de code, la gestion de l'état, la bibliothèque de composants et la stratégie de test Front-End.
  + **Lien** : [Lien vers le document CDC Front-End]
* **Titre du Document** : CDC Back-End  
  + **Propriétaire** : Lead Dev Back-End
  + **Mission** : Détailler l'architecture de chaque microservice, les patrons de conception utilisés (patterns), les conventions FastAPI, et la définition des événements publiés.
  + **Lien** : [Lien vers le document CDC Back-End]
* **Titre du Document** : CDC Agents IA  
  + **Propriétaire** : Lead IA/ML
  + **Mission** : Détailler le fonctionnement interne de chaque agent IA, les modèles de langage utilisés, les prompts, les logiques d'analyse et les flux d'orchestration.
  + **Lien** : [Lien vers le document CDC Agents IA]
* **Titre du Document** : CDC Données (EDR)  
  + **Propriétaire** : Lead Dev Back-End / Architecte de Données
  + **Mission** : Fournir le Schéma Entité-Relation (ERD) complet et détaillé de la base de données PostgreSQL. C'est la source de vérité pour toutes les tables, colonnes et relations.
  + **Lien** : [Lien vers le document CDC Données (EDR)]
* **Titre du Document** : CDC DevOps  
  + **Propriétaire** : Ingénieur DevOps / SRE
  + **Mission** : Détailler la configuration de l'infrastructure (IaC), les pipelines CI/CD dans GitHub Actions, les stratégies de monitoring, de logging, d'alerting et de sauvegarde.
  + **Lien** : [Lien vers le document CDC DevOps]

### **6. Annexe : Stack Technologique Officielle**

La liste suivante est exhaustive et prescriptive. Aucune technologie ne peut être ajoutée au projet sans une modification formelle de ce GAG.

| Domaine | Technologie | Version / Spécification | Notes |
| --- | --- | --- | --- |
| Langages | Python | 3.12+ | Pour tout le Back-End et les Agents IA. |
|  | TypeScript | 5.x+ | Pour tout le Front-End. |
| Front-End | React | 18+ | Framework principal. |
|  | Vite | 5.x+ | Outil de build. |
|  | Zustand | 4.x+ | Pour la gestion d'état globale. |
|  | Tailwind CSS | 3.x+ | Pour le style. |
|  | Vitest | 1.x+ | Pour les tests unitaires/intégration. |
| Back-End | FastAPI | 0.110+ | Framework d'API. |
|  | Uvicorn | 0.29+ | Serveur ASGI. |
|  | SQLAlchemy | 2.x+ | ORM pour l'accès à la base de données. |
|  | Alembic | 1.13+ | Pour les migrations de base de données. |
| Base de Données | PostgreSQL | 16+ | SGBDR principal. |
|  | pgvector | 0.7+ | Extension pour la recherche vectorielle. |
| Messagerie | Redis | 7.x+ | Pour le cache et le Pub/Sub. |
| Orchestration | Prefect | 2.x+ | Pour orchestrer les flux des Agents IA. |
| Infrastructure | Google Cloud | N/A | Fournisseur Cloud unique. |
|  | Docker | N/A | Pour la conteneurisation. |
|  | Kubernetes (GKE) | 1.28+ | Pour l'orchestration des conteneurs. |
| CI/CD | GitHub Actions | N/A | Pour l'intégration et le déploiement continus. |