

SkillMates

Réseau social pour les apprenants

Projet réalisé dans le cadre de la présentation au

Titre Professionnel Développeur Concepteur D'application

TABLE DES MATIERES

1. Résumé du projet en anglais
2. Liste des compétences du RC couvertes par le projet
3. Expression des besoins de l'application à réaliser (cahier des charges)
4. Environnement technique
5. Gestion de projet
6. Spécifications fonctionnelles
7. Spécification techniques du projet
8. Réalisations techniques
9. Tests et qualité
10. Conclusion

1 - Résumé du projet en anglais

For this solo project, I wanted to resolved a problem my formation mates and I encountered before enrolled into this formation and having the chance to learn with other people : ***the peer to peer Learning.***

The purpose of this projet is to connect people with similar goals to do peer to peer learning online by planning learning sessions.

This was a good way for me to check all the needs for this certification by creating something that might be useful for others.

How it works:

1- The user simply create an account, follow the creation profile steps by adding the username, skills, goals.

2 - The connected user can navigate to the « Mes partenaires » (Partnership) Tab and search threw the list of users and ses their « goals », when he find one goal that match is interests, he can send a partnership request followed by a clear message to the user and see his own request in the pending area of the partnership section.

3 - On the other side the user can see in his pending demand area the partnership request that was sent to him with the connection message and choose to accept or denied this request.

4 - When a demand has been accepted, both users can see their partnerships in their active partnership area.

5 - Now that we finally have a mate, we can schedule sessions in the « Session » Tab, by adding a name, description, type on meeting (virtual/physic) and the date and our, we sélect the partner we want for this session and we have our session scheduled with the meeting link or adress and dates revealed in the session Tab .

2 - Liste des compétences du référentiel qui sont couvertes par ce projet

Pour l'activité type 1 « Développer une application sécurisée » :

- Développer des interfaces utilisateur
- Développer des composants métier d'une application
- Contribuer à la gestion d'un projet informatique

Pour l'activité type 2 « Concevoir et développer une application sécurisée organisée en couches » :

- Analyser les besoins et maquetter une application
- Définir l'architecture logicielle d'une application
- Concevoir et mettre en place une base de données relationnelle
- Développer des composants d'accès aux données SQL et NoSQL

Pour l'activité type 3 « Préparer le déploiement d'une application sécurisée » :

- Préparer et exécuter les plans de tests d'une application

3. Expression des besoins de l'application à réaliser (cahier des charges)

3.1 Contexte et objectifs

Dans un monde où l'apprentissage continu devient essentiel, de nombreuses personnes cherchent à acquérir de nouvelles compétences de manière autodidacte. Cependant, l'apprentissage en solitaire présente des défis majeurs : manque de motivation, difficulté à maintenir la régularité, et absence de feedback. L'application SkillMates vise à résoudre ces problèmes en créant un écosystème favorisant l'apprentissage collaboratif.

Objectifs principaux :

- Faciliter la mise en relation de personnes partageant des intérêts d'apprentissage similaires
- Permettre la création de partenariats d'apprentissage structurés
- Favoriser la régularité et l'engagement via un système de sessions programmées
- Créer une communauté d'entraide et de partage de connaissances

3.2 Cible utilisateurs

L'application s'adresse à un public varié :

- Étudiants cherchant à compléter leur formation académique
- Professionnels en reconversion ou montant en compétences
- Autodidactes dans tous domaines (développement, langues, design, etc.)
- Personnes isolées géographiquement cherchant des partenaires d'apprentissage

3.3 Fonctionnalités attendues

Gestion des utilisateurs :

- Inscription et authentification des utilisateurs
- Création et édition de profil détaillé (compétences, intérêts, objectifs)
- Complétion progressive du profil en plusieurs étapes

Système de partenariat :

- Recherche de partenaires potentiels
- Envoi et gestion des demandes de partenariat
- Acceptation ou refus des demandes reçues
- Vue d'ensemble des partenariats actifs et en attente

Gestion des objectifs d'apprentissage :

- Définition d'objectifs personnels

- Partage d'objectifs avec des partenaires

Organisation de sessions :

- Planification de sessions d'apprentissage (virtuelles ou physiques)
- Gestion du calendrier des sessions

Interface utilisateur :

- Tableau de bord personnalisé

3.4 Contraintes techniques

- Architecture microservices pour la scalabilité et la maintenance
- Base de données NoSQL pour la flexibilité des schémas de données
- Base de données SQL pour la robustesse des schémas de données
- Interface utilisateur responsive compatible avec les principaux navigateurs
- Sécurisation des données personnelles des utilisateurs
- Temps de réponse optimisés pour une expérience fluide

3.5 Livrables attendus

- Code source de l'application complet et documenté
- Documentation technique détaillant l'architecture et les choix techniques
- Procédure de déploiement et d'installation

4 - Environnement technique

Technologies utilisées :

LANGAGES DE PROGRAMMATION:

Comme langage de programmation j'utilise Java pour toute la partie backend, pour la parti frontend j'utilise Thymeleaf comme moteur de template pour Java afin de générer du contenu HTML, CSS et Javascript (langages frontend utilisés). Comme framework j'utilise SpringBoot.

OUTILS DE DÉVELOPPEMENT

Comme IDE j'utilise IntelliJ pour la facilité qu'il offre au développement avec java/ Springboot et VsCode par familiarité.

Git & Github pour le versionning

Docker pour la conteneurisation et le déploiement de l'application

ARCHITECTURE

L'application est organisée en microservices.

Le code est organisé dans un pattern architectural MVC (Model View Controller)

5 - Gestion de projet

Pour assurer la bonne gestion du projet durant son développement, je me suis appuyé sur les techniques et technologies suivantes:

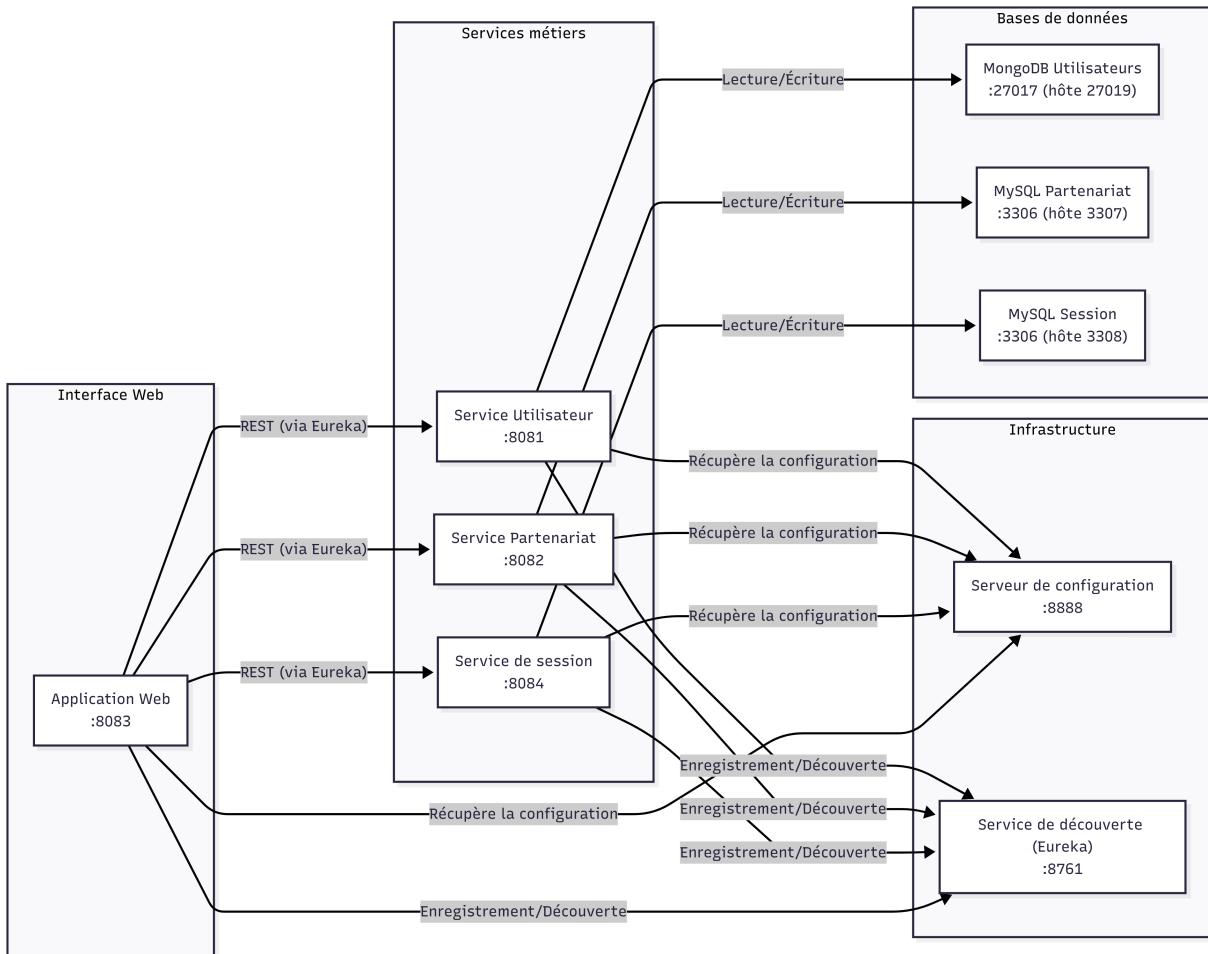
- **Méthodologie agile** : Chaque semaines (jours), je fais un sprint planning pour prioriser et réajuster si nécessaire. Cette méthode me permet d'adapter les contraintes de temps au projet afin de garantir productivité et efficacité.
- **Versionning** : Pour cela, j'utilise Git & Github. Je commit à chaque correction de bug ou développement d'une nouvelle fonctionnalité avec des noms de commit clair afin de pouvoir développer sans prendre de risque de pertes. Car je travail seul, je travail sur deux branches: la branch « dev » tout au long de la partie développement et test puis je merge sur la branch « main » lorsque la partie tester est valide et fonctionnelle.
- Tableau Kanban: Sur GitHub dans « Project » j'utilise un tableau Kanban avec les colonnes suivantes:
 - « Backlogs » les features à considérer pour plus tard
 - « Ready » pour une ou deux features à considérer au plus vite (une durée maximale de deux jours)
 - « In progress » pour la/les features en cours (max 2)
 - « In Review » pour les features finies mais à vérifier/reconsidérer
 - « Done » pour les features terminées.

A noter que sur Github, chaque feature correspond à une Issue donc peut être lié à des commits, annoté de niveau de priorité, temps estimé, attribuer à un user(si travail en équipe), on peut également donner une taille correspondant au volume de la feature (de l'issue)

6 - Spécifications fonctionnelles

6.1 Diagramme d'architecture logicielle

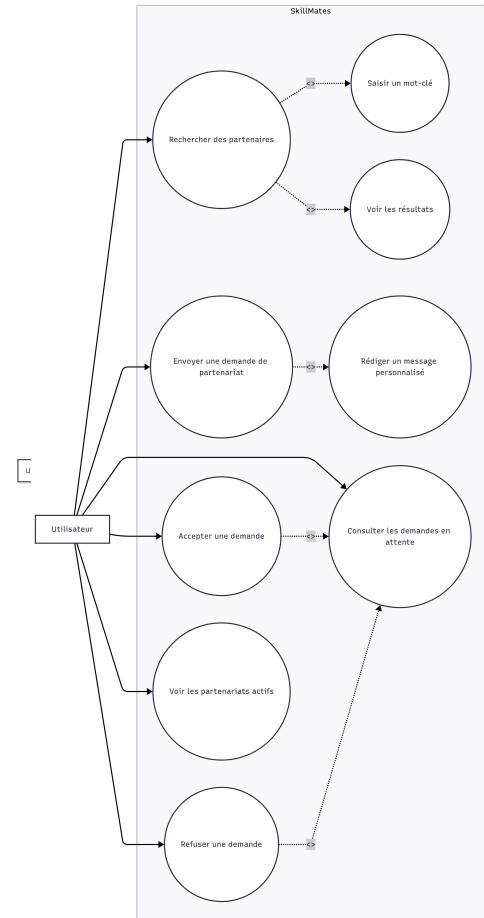
Un diagramme d'architecture logicielle, c'est comme le plan d'une maison mais pour un logiciel : il montre comment tous les éléments du système s'emboîtent et communiquent entre eux pour faire fonctionner l'application.



6.2 Parcours utilisateur (Use Case)

Inscription et création de profil :

1. L'utilisateur s'inscrit avec son email, un nom d'utilisateur et un mot de passe
2. Il est guidé à travers un processus en 4 étapes pour compléter son profil :
 - Informations personnelles (nom, bio)
 - Sélection des compétences possédées
 - Sélection des centres d'intérêt
 - Définition des objectifs d'apprentissage
3. Une fois le profil complété, l'utilisateur est amené vers la page de connexion, rentre ses identifiants et accède à son tableau de bord



Recherche et gestion des partenariats :

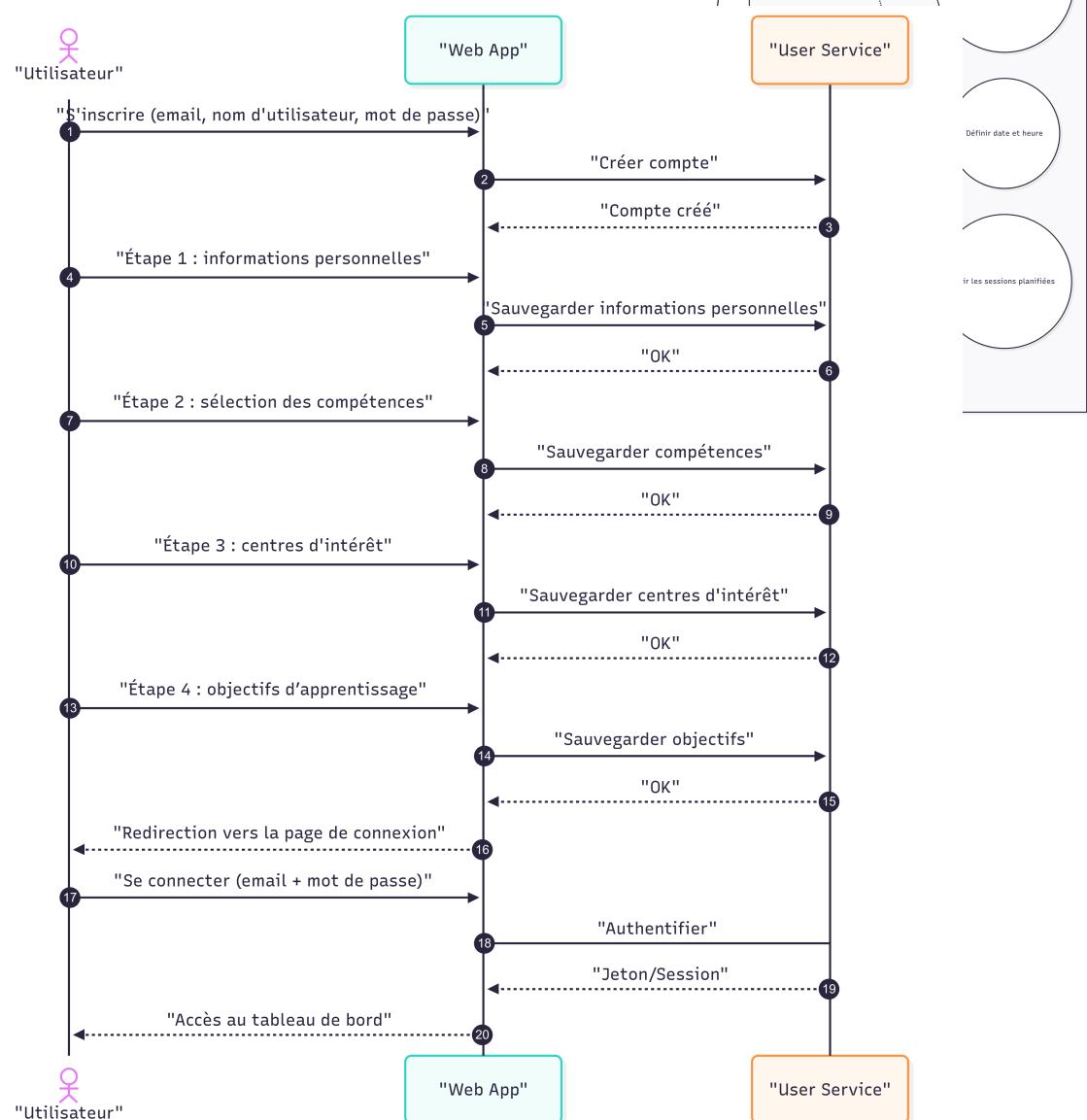
1. L'utilisateur peut rechercher des partenaires potentiels via une barre de recherche
2. Il peut envoyer des demandes de partenariat avec un message personnalisé
3. Les demandes reçues apparaissent dans la section "Demandes en attente"
4. L'utilisateur peut accepter ou refuser les demandes reçues
5. Les partenariats actifs sont visibles dans une section dédiée

Planification et gestion des sessions :

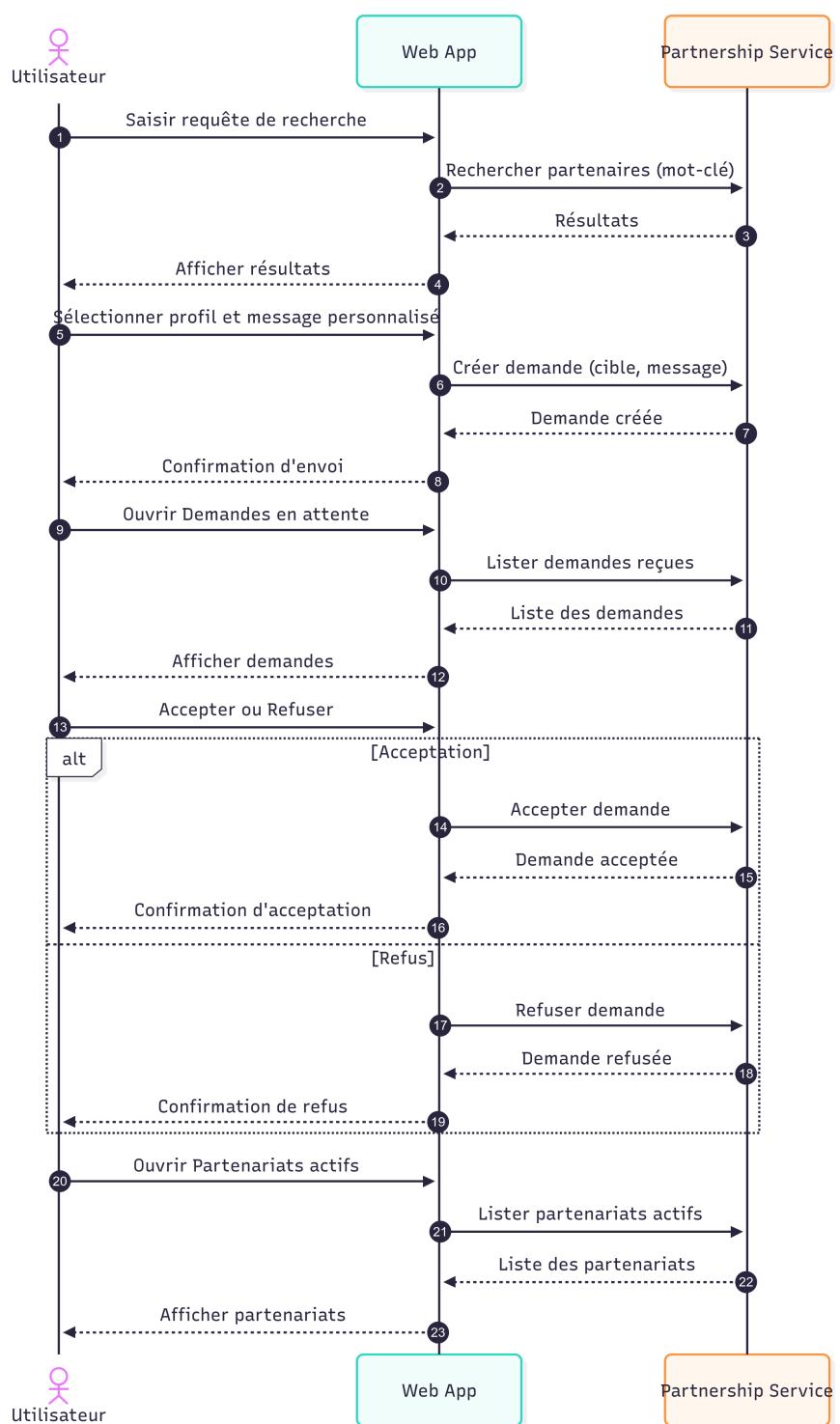
1. L'utilisateur peut créer une nouvelle session avec un partenaire actif
2. Il définit le titre, la description, le type (virtuel/physique), la date et l'heure
3. Les sessions planifiées apparaissent dans la liste des sessions
4. L'utilisateur peut consulter les détails d'une session

6.3 Diagrammes de Séquences

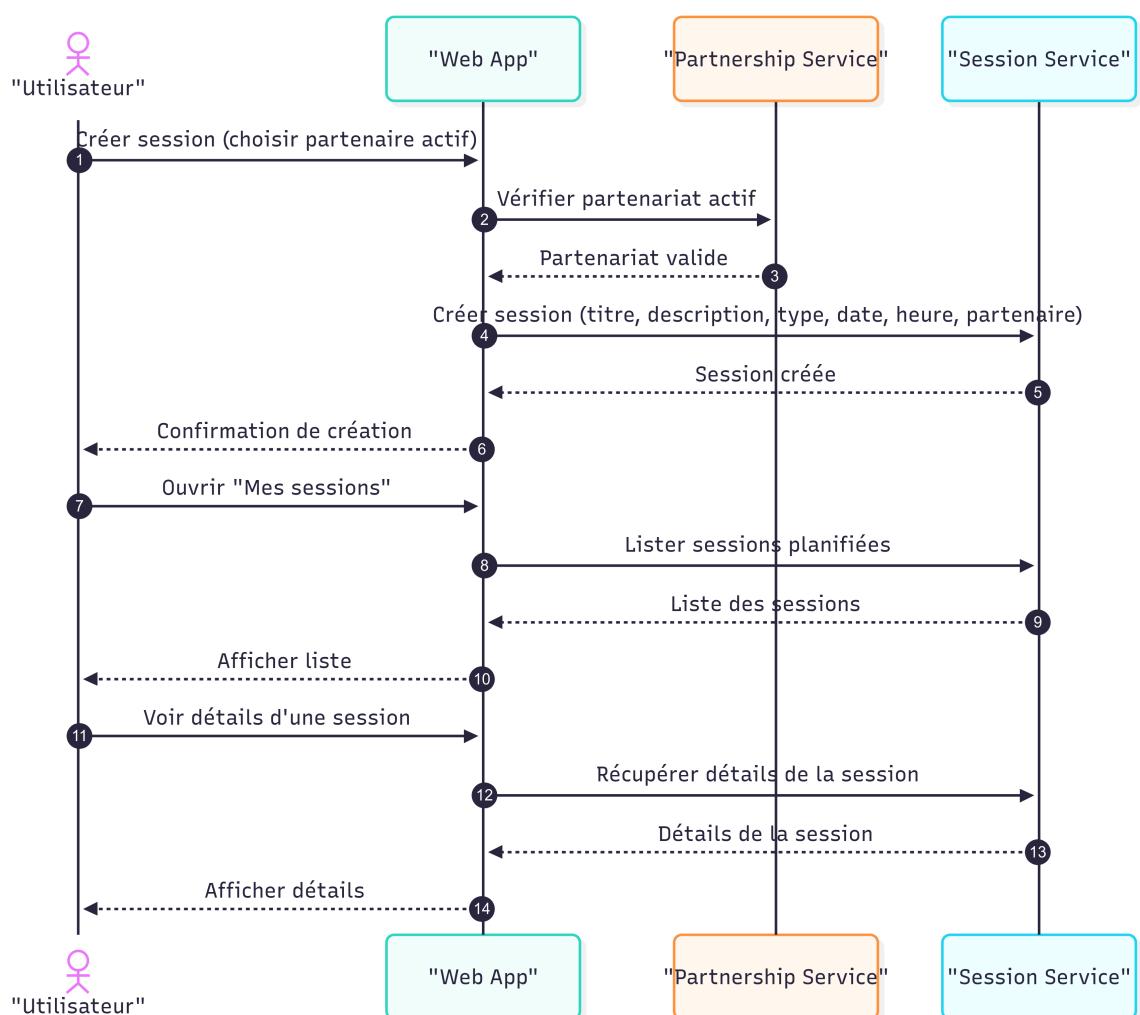
Inscription et création de profil:



Recherche et gestion des partenaires:



Planification et gestion des sessions :



6.4 Description détaillée des fonctionnalités

Module utilisateur :

- Gestion complète du profil utilisateur
- Système de tags pour les compétences et intérêts
- Complétion progressive du profil
- Stockage sécurisé des informations personnelles
- Recherche d'utilisateurs par nom (à l'avenir par tags de compétences)

Module partenariat :

- Système de demande/acceptation de partenariat
- Gestion du cycle de vie des partenariats (demande, actif, terminé)
- Interface dédiée pour la gestion des partenariats

Module session :

- Création de sessions avec date, heure, type et localisation
- Association des sessions à un partenariat spécifique
- Vue calendrier des sessions à venir
- Gestion des statuts de session (planifiée, terminée, annulée)

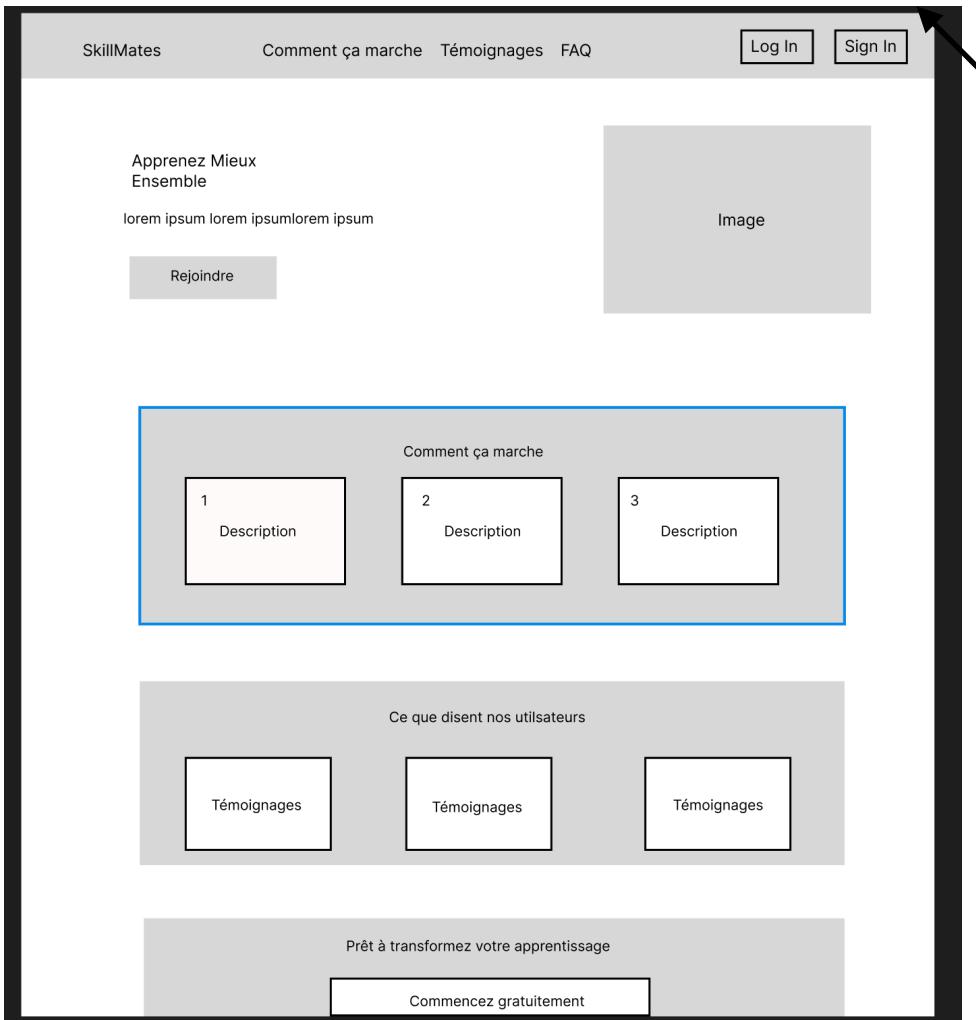
6.5 Wireframes et UI

Je présenterai dans cette section la phase de croquis (Wireframes) et l'UI finale de l'interface web. Note: Pour les Wireframes j'ai utilisé Figma.

1 - L'utilisateur se rend sur la landing page

Wireframe:

Sign In



Wireframe:

The wireframe shows the "Créer un compte" (Create account) form. It features a header "Créer un compte", followed by three input fields: "Nom d'utilisateur" (Username), "Adresse email" (Email address), and "Mot de passe" (Password). Below these fields is a "Créer un compte" button. At the bottom of the form is a link "Déjà un compte ? Connectez vous" (Already have an account? Log in).

SkillMates

Créer un compte

Nom d'utilisateur

Adresse email

Mot de passe

Créer un compte

Déjà un compte ? Connectez vous

UI:

The screenshot shows a blue-themed sign-up form titled "Créer un compte". The form includes fields for "Nom d'utilisateur" (Your name), "Adresse email" (Your email address), and "Mot de passe" (Create a secure password). A large orange button at the bottom right says "Créer mon compte →". Below the button, a link reads "Déjà un compte ? [Se connecter](#)". The top left corner features the SkillMates logo.

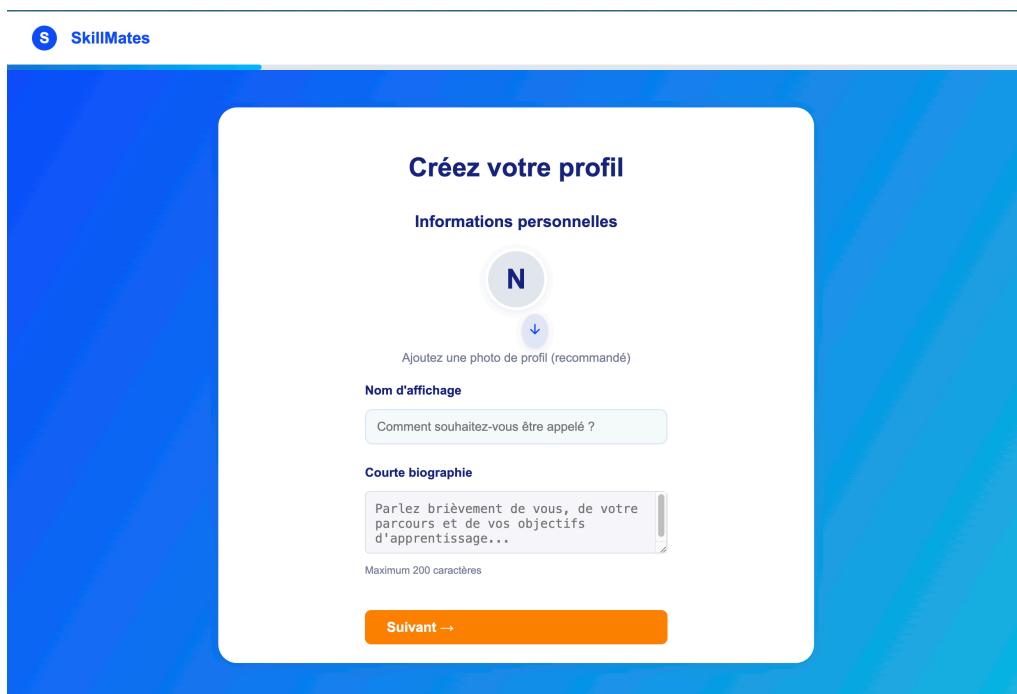
3 - L'utilisateur est redirigé vers la première étape de création de profil utilisateur

Wireframe:

Inscription step 1

The wireframe for "Inscription step 1" shows a "Créer votre profil" screen. It features a large circular placeholder for a profile picture. Below it are two input fields: one for "Nom d'affichage" (Display name) and another for "Courte biographie" (Short biography). At the bottom is a "Next" button.

UI :



4 - Après avoir saisi les données dans les champs, l'utilisateur passe à la saisie des compétences

Wireframe:

UI :

The screenshot shows the 'Créer votre profil' (Create your profile) step of the SkillMates sign-up process. At the top, there's a header bar with the text 'Inscription step 2'. Below it, a large central box contains the title 'Créer votre profil' and a section titled 'Compétences' with the sub-section 'Ajout des compétences'. A blue rectangular highlight surrounds the 'Ajout des compétences' input field. To the right of this field, a black arrow points towards the text 'Tags personnalisés' (Personalized tags). The bottom of the screen features a large blue sidebar with the SkillMates logo and navigation buttons for 'Précédent' (Previous) and 'Suivant' (Next).

5- Il clique sur Suivant est rempli ses centres d'intérêts

Wireframe:

UI:

The image shows a comparison between a wireframe and the final UI for a profile creation step. The wireframe, labeled 'Inscription step 3', is a simple gray rectangle with the text 'Créer votre profil' in the center. The UI, labeled 'SkillMates', is a more complex version with a blue header and footer. It features a title 'Créez votre profil', a section titled 'Centres d'intérêt' with a sub-instruction 'Sélectionnez les domaines qui vous intéressent pour trouver des partenaires partageant vos passions'. Below this is a list of interest categories each with a checkbox, such as 'Web Development', 'Mobile App Development', etc. A 'Suivant →' button is at the bottom right.

6 - Il clique sur Suivant est rempli de 1 à 3 objectifs d'apprentissage

Wireframe :

Inscription step 4

Créer votre profil

Objectifs d'apprentissage

UI :

S SkillMates

Créez votre profil

Personnalisez votre profil pour trouver les meilleurs partenaires d'apprentissage

Objectifs d'apprentissage

Définissez vos objectifs pour trouver des partenaires avec des ambitions similaires

Objectif d'apprentissage + Ajouter un objectif

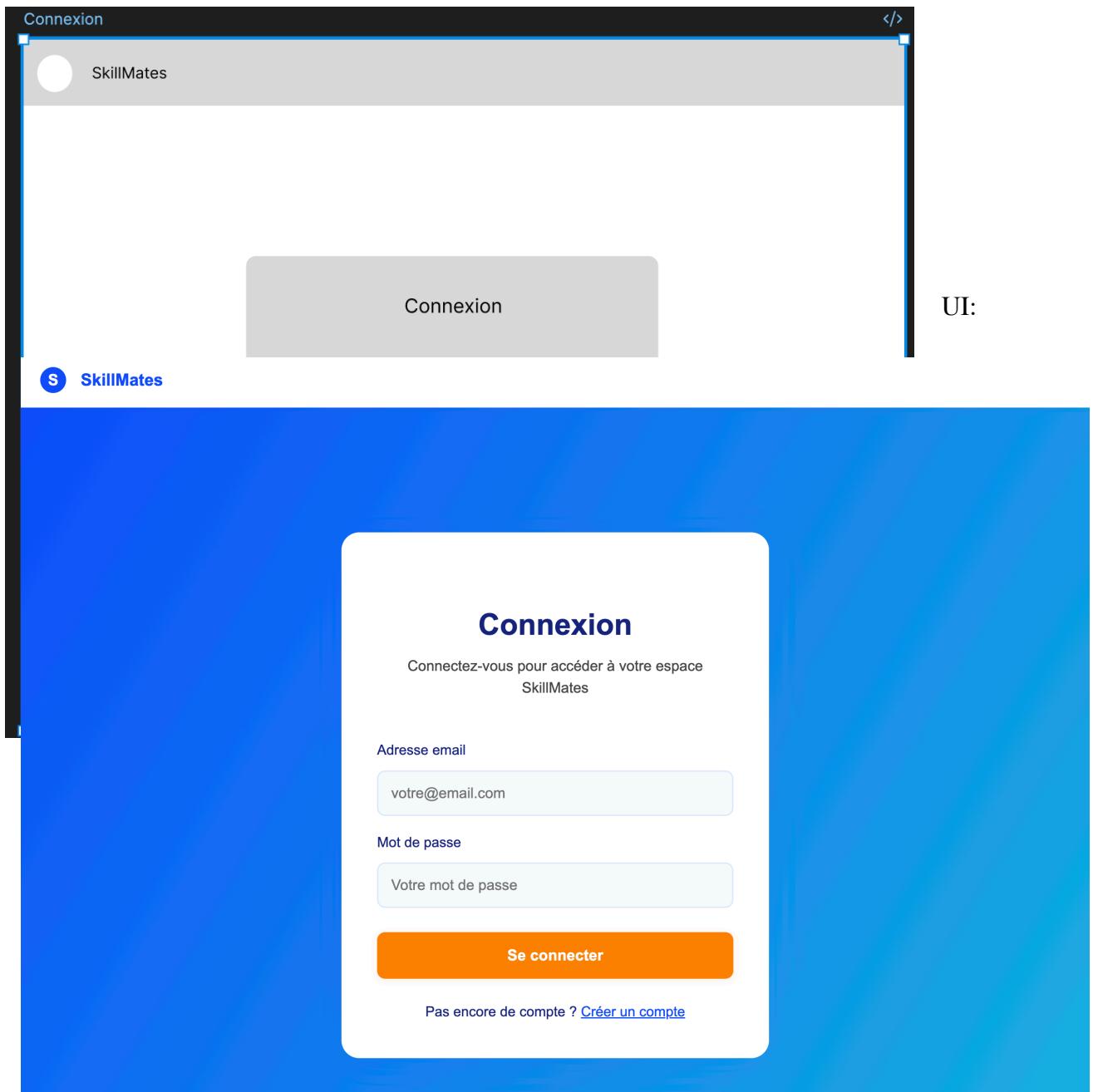
Conseils pour définir de bons objectifs

- Soyez spécifique (ex: "Apprendre React" plutôt que "Apprendre le développement web")
- Définissez des objectifs mesurables
- Fixez-vous des délais réalistes

← Précédent Finaliser mon profil →

7 - l'utilisateur est ensuite redirigé vers la page de connexion où il entrera ses données pour s'authentifier

Wireframe



8 - Lorsque l'utilisateur se connecte, il est redirigé vers le Dashboard (Tableau de bord)

Wireframe :

The wireframe illustrates the user interface of the SkillMates application. It features a sidebar on the left with a 'Dashboard' section containing links to 'Dashboard', 'Mon Profil', 'Session', and 'Partenaires'. Below this is the 'SkillMates' navigation bar with 'Tableau de bord' selected. The main content area includes a 'Tags' section with a bio placeholder, a 'Mes objectifs' section with a placeholder, and a 'Tableau de bord' summary for 'meymey' showing availability and Python apprenticeship status. A 'Mes objectifs d'apprentissage' section displays a learning goal for React with a progress bar at 0%.

9 - En cliquant sur « Partenaires » il accepte aux fonctionnalités de partenariat

Il peut consulter ses demandes en attentes ainsi que ses partenaires actifs

Wireframe :

Partenaires - Mes partenaires

Partenaires

Mes partenaires Trouver un Partenaire

Demandes en attentes

Nom
Message : Lor ipsum lorem ipsumem i

Partenaires

Gérez vos partenaires d'apprentissage et trouvez de nouveaux mates

Mes partenaires Trouver un mate

Demandes en attente

Aucune demande de partenariat en attente

Partenariats actifs

- kola** ACTIF
 - Message
 - Planifier
- antonio** ACTIF
 - Message
 - Planifier

UI :

10 - Sur « Trouver un mate » il peut parcourir la liste des utilisateurs de l'application et leurs objectifs et faire une demande de partenariat

Wireframe:

Partenaires - Trouver un partenaire

The wireframe on the left shows the initial state of the 'Partenaires' section, featuring a sidebar with 'Dashboard', 'Mon Profil', 'Session', and 'Partenaires' options, and a main area with 'Mes partenaires' and 'Trouver un Mate' tabs, a search bar, and a placeholder card for 'Bob'.

The UI on the right shows the final state after a user has selected the 'Partenaires' tab. It includes a sidebar with 'SkillMates' navigation (Tableau de bord, Mon profil, Mes apprentissages, Objectifs, Session, Messages, Partenaires), a main header 'Partenaires', a sub-header 'Gérez vos partenaires d'apprentissage et trouvez de nouveaux mates', and a 'Trouver un mate' tab which is active. Below this is a search bar and a list of users:

- yoyo** — Objectifs: being better Demander un partenariat
- joe** — Objectifs: apprendre rust Demande un partenariat
- ben** — Objectifs: Next, REact, Angular Demande un partenariat
- ken** — Objectifs: devenir devops Demande un partenariat
- kam** — Objectifs: devenir dev confirmé rust Demande un partenariat

UI :

11 - L'utilisateur ayant un ou des partenaire(s) peut voir ses sessions programmer et l'historique des sessions

Wireframe:

Session

The wireframe shows a sidebar with navigation options: Dashboard, Mon Profil, Session, and Partenaires. The main area is titled "Session" and contains a table for creating a session. The table has columns: Titre, Début, Fin, Statut, and Partenaire. A "Créer une Session" button is at the top right. Below the table are three buttons: Détails, Annuler, and Terminer.

Titre	Début	Fin	Statut	Partenaire
				Détails Annuler Terminer

UI:

SkillMates

Mes sessions

Créer une session

Titre	Début	Fin	Statut	Partenaire	
apprendre Python	2025-09-01T17:30	2025-09-01T19:30	CANCELED	kola	<button>Détails</button>
apprendre python	2025-09-01T10:00	2025-09-01T12:00	COMPLETED	kola	<button>Détails</button>
apprendre python	2025-09-01T10:15	2025-09-01T12:15	CANCELED	kola	<button>Détails</button>
apprendre React	2025-09-01T11:20	2025-09-01T12:30	CANCELED	kola	<button>Détails</button>
apprendre les bases de l'IA	2025-08-31T08:45	2025-08-31T10:45	CANCELED	kola	<button>Détails</button>
apprendre les bases de Go	2025-08-31T08:50	2025-08-31T11:50	PLANNED	antonio	<button>Détails</button> <button>Annuler</button> <button>Terminer</button>
test	2025-08-31T10:00	2025-08-31T12:00	PLANNED	antonio	<button>Détails</button> <button>Annuler</button> <button>Terminer</button>

Tableau de bord

Mon profil

Mes apprentissages

Objectifs

Session

Messages

Partenaires

Paramètres

Déconnexion

12 - L'utilisateur peut créer une session et l'associer à un partenaire

Wireframe :

SESSION - Créer une session

- Dashboard
- Mon Profil
- Session
- Partenaires

SkillMates

- Tableau de bord
- Mon profil
- Mes apprentissages
- Objectifs
- Session
- Messages
- Partenaires

Paramètres

Déconnexion

Créer une session

Titre

Description

Type Virutelle > lieu/lien

Créer une session

Retour

Titre

Description

Type Lieu/Lien

Début Fin Partenaire **Créer**

6.6 Règles de gestion

Gestion des utilisateurs :

- Un email ne peut être associé qu'à un seul compte
- Le nom d'utilisateur doit être unique
- Le mot de passe est stocké de manière cryptée (BCrypt)
- Le profil est considéré comme complet après les 4 étapes d'inscription

Gestion des partenariats :

- Un utilisateur ne peut pas s'envoyer une demande de partenariat à lui-même
- Une demande de partenariat doit contenir un message explicatif
- Un partenariat ne devient actif qu'après acceptation par le destinataire
- Un utilisateur peut avoir plusieurs partenariats actifs simultanément

Gestion des sessions :

- Une session doit être associée à un partenariat actif
- Une session doit avoir une date de début et de fin
- Le type de session (virtuel/physique) détermine le format de la localisation
- Seuls les participants à une session peuvent la modifier ou l'annuler

7. Spécification techniques du projet

7.1 Architecture globale

L'application SkillMates est basée sur une architecture microservices, composée de plusieurs services indépendants qui communiquent entre eux via des API REST. Cette architecture offre une meilleure scalabilité, une maintenance facilitée et une évolution plus souple des différentes composantes.

Principaux composants :

- **user-service** : Gestion des utilisateurs, compétences, intérêts et objectifs
- **partnership-service** : Gestion des partenariats entre utilisateurs
- **session-service** : Gestion des sessions d'apprentissage
- **web-app** : Application web servant l'interface utilisateur
- **config-server** : Serveur de configuration centralisée
- **eureka-server** : Service de découverte pour l'enregistrement des microservices

7.2 Technologies utilisées

Backend :

- **Java 17** : Langage de programmation principal
- **Spring Boot 3.5.0** : Framework pour le développement des microservices
- **Spring Cloud 2025.0.0** : Outils pour l'architecture cloud-native
- **Spring Data MongoDB** : Persistance des données dans MongoDB
- **Spring Security** : Sécurisation de l'application
- Spring Cloud Netflix Eureka : Découverte de services
- **Spring Cloud Config** : Configuration centralisée
- **Spring Cloud OpenFeign** : Client HTTP déclaratif pour les appels entre services

Frontend :

- **Thymeleaf** : Moteur de templates pour le rendu côté serveur
- **HTML5/CSS3** : Structure et style des pages web
- **JavaScript** : Interactivité côté client
- **Bootstrap** (ou framework CSS personnalisé) : Framework CSS pour l'interface responsive
-

Base de données :

- **MongoDB** : Base de données NoSQL pour le stockage des données utilisateurs
- MySQL : Bases de données SQL pour le stockage des données métier

Outils de build et déploiement :

- **Maven** : Gestion des dépendances et du build
- **Git** : Gestion de versions
- **Docker** : Conteneurisation des services et déploiement

7.3 Modèle entités-associations et modèle physique des bases de données

Le modèle de données est organisé par domaine, avec des collections MongoDB ou MySQL spécifiques pour chaque service :

user-service :

- Collection `users` : Profils utilisateurs
- Collection `skills` : Tags de compétences
- Collection `interests` : Tags d'intérêts
- Collection `learning_objectives` : Objectifs d'apprentissage

Partnership-service:

Entité Partnership

- **id**: Long (clé primaire auto-générée)
- **requesterId**: String (ID de l'utilisateur qui demande le partenariat)
- **requestedId**: String (ID de l'utilisateur à qui est demandé le partenariat)
- **status**: PartnershipStatus (énumération: PENDING, ACCEPTED, DENIED, CANCELLED, ENDED)
- **message**: String (message d'accompagnement de la demande)
- **createdAt**: LocalDateTime (horodatage de création)
- **updatedAt**: LocalDateTime (horodatage de mise à jour)
- **acceptedAt**: LocalDateTime (date d'acceptation du partenariat)
- **endedAt**: LocalDateTime (date de fin du partenariat)
- **goals**: Liste de PartnershipGoal (objectifs du partenariat)

Entité PartnershipGoal

- **id**: Long (clé primaire auto-générée)
- **partnership**: Partnership (relation avec le partenariat parent)
- **title**: String (titre de l'objectif)
- **description**: String (description détaillée)
- **progressPercentage**: Integer (pourcentage de progression, défaut à 0)
- **targetDate**: LocalDate (date cible pour atteindre l'objectif)
- **createdAt**: LocalDateTime (horodatage de création)

Énumération PartnershipStatus

- PENDING (en attente)
- ACCEPTED (accepté)
- DENIED (refusé)
- CANCELLED (annulé)
- ENDED (terminé)

Service Session

Entité Session

- **id**: Long (clé primaire auto-générée)
- **title**: String (titre de la session, max 200 caractères)
- **description**: String (description au format TEXT)
- **type**: SessionType (énumération: VIRTUAL, PHYSICAL)
- **locationOrLink**: String (lieu physique ou lien virtuel, max 512 caractères)
- **startAt**: LocalDateTime (date et heure de début)
- **endAt**: LocalDateTime (date et heure de fin)
- **status**: SessionStatus (énumération: PLANNED, COMPLETED, CANCELED, défaut à PLANNED)
- **partnershipId**: Long (ID du partenariat associé)
- **organizerId**: String (ID de l'utilisateur organisateur, max 64 caractères)
- **partnerId**: String (ID de l'utilisateur partenaire, max 64 caractères)
- **createdAt**: LocalDateTime (horodatage de création, non modifiable)
- **updatedAt**: LocalDateTime (horodatage de mise à jour)

Énumération SessionStatus

- PLANNED (planifiée)
- COMPLETED (terminée)
- CANCELED (annulée)

Énumération SessionType

- VIRTUAL (virtuelle)
- PHYSICAL (physique)

Script et Modèles physiques des bases de données:

Partnership et Partnership-goals:

```
sql
CREATE DATABASE IF NOT EXISTS partnership
    CHARACTER SET utf8mb4
    COLLATE utf8mb4_0900_ai_ci;
USE partnership;

CREATE TABLE partnerships (
    id BIGINT NOT NULL AUTO_INCREMENT,
    requester_id VARCHAR(64) NOT NULL,
    requested_id VARCHAR(64) NOT NULL,
    status VARCHAR(32) NULL,
    message TEXT NULL,
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    accepted_at DATETIME NULL,
    ended_at DATETIME NULL,
    PRIMARY KEY (id),
    KEY idx_partnerships_requester_requested (requester_id, requested_id)
) ENGINE=InnoDB;

CREATE TABLE partnership_goals (
    id BIGINT NOT NULL AUTO_INCREMENT,
    partnership_id BIGINT NOT NULL,
    title VARCHAR(255) NOT NULL,
    description TEXT NULL,
    progress_percentage INT NOT NULL DEFAULT 0,
    target_date DATE NULL,
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (id),
    KEY idx_goals_partnership_id (partnership_id),
    CONSTRAINT fk_goals_partnership
        FOREIGN KEY (partnership_id) REFERENCES partnerships(id)
        ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB;
```

PARTNERSHIPS		
BIGINT	id	PK
VARCHAR	requester_id	
VARCHAR	requested_id	
STRING	status	
TEXT	message	
DATETIME	created_at	
DATETIME	updated_at	
DATETIME	accepted_at	
DATETIME	ended_at	

+—————
| contains
| ○—————|

PARTNERSHIP_GOALS		
BIGINT	id	PK
BIGINT	partnership_id	FK
VARCHAR	title	
TEXT	description	
INT	progress_percentage	
DATE	target_date	
DATETIME	created_at	

Sessions:

```
sql
CREATE DATABASE IF NOT EXISTS skillmates_session
    CHARACTER SET utf8mb4
    COLLATE utf8mb4_0900_ai_ci;
USE skillmates_session;

CREATE TABLE sessions (
    id BIGINT NOT NULL AUTO_INCREMENT,
    title VARCHAR(200) NOT NULL,
    description TEXT NULL,
    type VARCHAR(16) NOT NULL,
    location_or_link VARCHAR(512) NULL,
    start_at DATETIME NOT NULL,
    end_at DATETIME NOT NULL,
    status VARCHAR(16) NOT NULL DEFAULT 'PLANNED',
    partnership_id BIGINT NOT NULL,
    organizer_id VARCHAR(64) NOT NULL,
    partner_id VARCHAR(64) NOT NULL,
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    PRIMARY KEY (id),
    KEY idx_sessions_partnership_id (partnership_id)
) ENGINE=InnoDB;
```

SESSIONS		
BIGINT	id	PK
VARCHAR	title	
TEXT	description	
STRING	type	
VARCHAR	location_or_link	
DATETIME	start_at	
DATETIME	end_at	
STRING	status	
BIGINT	partnership_id	
VARCHAR	organizer_id	
VARCHAR	partner_id	
DATETIME	created_at	
DATETIME	updated_at	

User-service script mongosh & mondial physique de la base de donnée NoSQL:

Script « skills » :

```

use('skillMates');

// skills
db.createCollection('skills', {
  validator: {
    $jsonSchema: {
      bsonType: 'object',
      required: ['name'],
      properties: {
        _id: { bsonType: 'string' },
        name: { bsonType: 'string', description: 'nom du tag (unique)' },
        category: { bsonType: ['string', 'null'] }
      }
    }
  }
});

// interests
db.createCollection('interests', {
  validator: {
    $jsonSchema: {
      bsonType: 'object',
      required: ['name'],
      properties: {
        _id: { bsonType: 'string' },
        name: { bsonType: 'string', description: 'nom du tag (unique)' },
        category: { bsonType: ['string', 'null'] },
        isPredefined: { bsonType: 'bool' }
      }
    },
    additionalProperties: true
  }
},
  validationLevel: 'moderate'
});
db.interests.createIndex({ name: 1 }, { unique: true, name: 'uniq_interest_name' });

```

Script
« interests »
:

Script « learning_objectives » :

```

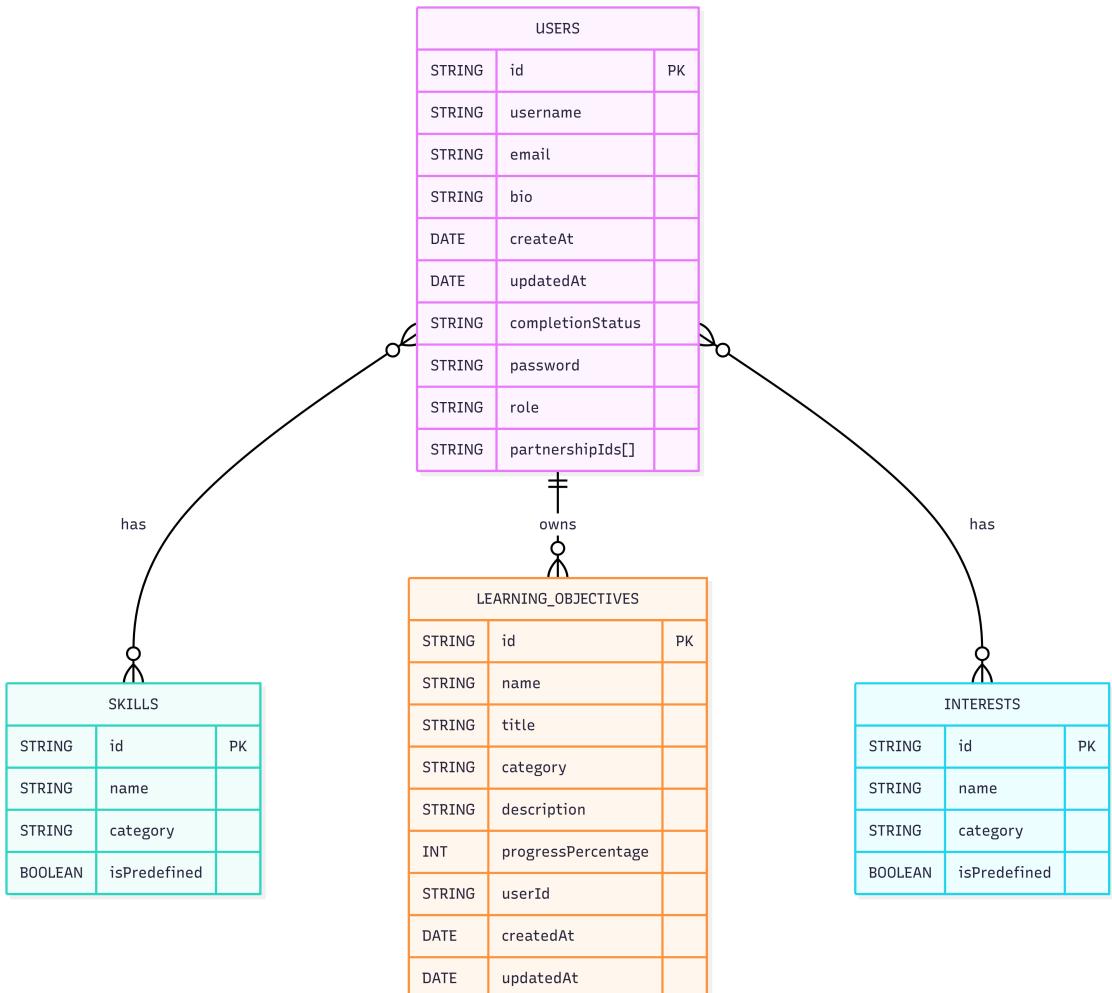
// learning_objectives
db.createCollection('learning_objectives', {
  validator: {
    $jsonSchema: {
      bsonType: 'object',
      properties: {
        _id: { bsonType: 'string' },
        name: { bsonType: ['string', 'null'] },
        title: { bsonType: ['string', 'null'] },
        category: { bsonType: ['string', 'null'] },
        description: { bsonType: ['string', 'null'] },
        progressPercentage: { bsonType: 'int', minimum: 0 },
        userId: { bsonType: ['string', 'null'] },
        createdAt: { bsonType: ['date', 'null'] },
        updatedAt: { bsonType: ['date', 'null'] }
      }
    },
    additionalProperties: true
  }
},
  validationLevel: 'moderate'
});

```

Script « users » :

```
// users
db.createCollection('users', {
  validator: {
    $jsonSchema: {
      bsonType: 'object',
      properties: {
        _id: { bsonType: 'string' },
        username: { bsonType: ['string', 'null'] },
        email: { bsonType: ['string', 'null'] },
        bio: { bsonType: ['string', 'null'] },
        createdAt: { bsonType: ['date', 'null'] },
        updatedAt: { bsonType: ['date', 'null'] },
        completionStatus: {
          bsonType: ['string', 'null'],
          enum: ['INITIAL','PERSONAL_INFO_COMPLETED','SKILLS_COMPLETED','INTERESTS_COMPLETED','COMPLETED']
        },
        password: { bsonType: ['string', 'null'] },
        role: { bsonType: ['string', 'null'] },
        skills: {
          bsonType: ['array', 'null'],
          items: {
            anyOf: [
              { bsonType: 'object', properties: { $ref: { bsonType: 'string' }, $id: { bsonType: ['string','objectId'] } }, required: ['$ref','$id'], additionalProperties: { bsonType: 'string' } }
            ]
          }
        },
        learningObjectives: {
          bsonType: ['array', 'null'],
          items: {
            anyOf: [
              { bsonType: 'object', properties: { $ref: { bsonType: 'string' }, $id: { bsonType: ['string','objectId'] } }, required: ['$ref','$id'], additionalProperties: { bsonType: 'string' } }
            ]
          }
        },
        learningObjectives: {
          bsonType: ['array', 'null'],
          items: {
            anyOf: [
              { bsonType: 'object', properties: { $ref: { bsonType: 'string' }, $id: { bsonType: ['string','objectId'] } }, required: ['$ref','$id'], additionalProperties: { bsonType: 'string' } }
            ]
          }
        },
        interests: {
          bsonType: ['array', 'null'],
          items: {
            anyOf: [
              { bsonType: 'object', properties: { $ref: { bsonType: 'string' }, $id: { bsonType: ['string','objectId'] } }, required: ['$ref','$id'], additionalProperties: { bsonType: 'string' } }
            ]
          }
        },
        partnershipIds: {
          bsonType: ['array', 'null'],
          items: { bsonType: 'string' }
        }
      },
      additionalProperties: true
    },
    validationLevel: 'moderate'
  }
});
```

Model physique user-service



6.5 Sécurité

La sécurité de l'application est assurée par plusieurs mécanismes :

- **Authentification** : Gestion des sessions utilisateurs avec Spring Security
- **Autorisation** : Contrôle d'accès basé sur les rôles (USER, ADMIN)
- **Protection des mots de passe** : Encodage avec BCryptPasswordEncoder
- **Sécurisation des communications** : HTTPS pour les échanges client-serveur
- **Validation des entrées** : Validation côté serveur des données entrantes

EXEMPLE DE VULNÉRABILITÉ IDENTIFIÉ AVEC LA MESURE CORRECTIVE:

Authentification non sécurisée dans le fichier « RegisterController »:

Dans ce fichier, lors de l'inscription d'un nouvel utilisateur, il y a une vulnérabilité:

Code avant correction:

```
UsernamePasswordAuthenticationToken auth = new UsernamePasswordAuthenticationToken(  
    userDetails, userDto.getPassword(), userDetails.getAuthorities());  
SecurityContextHolder.getContext().setAuthentication(auth);
```

Problème : le mot de passe en clair « userDto.getPassword() » est passé directement comme credential dans le token d'authentification. Cela expose le mot de passe dans la mémoire de l'application et dans les journaux de débogage potentiels.

mesure corrective :

```
UsernamePasswordAuthenticationToken auth = new UsernamePasswordAuthenticationToken(  
    userDetails, null, userDetails.getAuthorities());  
SecurityContextHolder.getContext().setAuthentication(auth);
```

Explication: Le deuxième paramètre du constructeur

« UsernamePasswordAuthenticationToken » représente les credentials. Après l'authentification réussie, ces credentials ne sont plus nécessaires et devraient être null pour éviter de conserver le mot de passe en mémoire.

6.6 Performance et scalabilité

Plusieurs mécanismes sont mis en place pour assurer la performance et la scalabilité :

- **Architecture microservices** : Permet le scaling horizontal des services les plus sollicités
- **Pagination** : Limitation du volume de données transmises lors des requêtes
- **DTOs** : Transfert uniquement des données nécessaires entre les couches
- **Discovery Service** : Facilite l'ajout dynamique d'instances de services

7. Réalisations significatives

7.1 Mise en place d'une architecture microservices complète

La conception et l'implémentation d'une architecture microservices fonctionnelle représentent une réalisation technique majeure du projet. Cette architecture permet :

- Une séparation claire des responsabilités entre services
- Une évolution indépendante de chaque composant
- Une meilleure résilience face aux pannes
- Un déploiement flexible et évolutif

Le choix de Spring Cloud comme socle technologique a permis de résoudre efficacement les défis inhérents aux architectures distribuées, comme la découverte de services(Eureka) , la configuration centralisée (ConfigServer) et la communication inter-services (OpenFeign)

7.2 Système de profil utilisateur évolutif

Le système de gestion des profils utilisateurs présente plusieurs aspects innovants :

- **Processus d'inscription progressif** : Découpage en étapes logiques pour améliorer l'expérience utilisateur
- **Système de tags flexible** : Architecture permettant l'ajout facile de nouvelles catégories de compétences et d'intérêts
- **Modèle de données extensible** : Utilisation de MongoDB pour une évolution facile du schéma de données

7.5 Système de gestion des sessions d'apprentissage

Le module de gestion des sessions représente une fonctionnalité clé qui différencie SkillMates des simples réseaux sociaux :

- Planification précise des rencontres d'apprentissage
- Support des sessions virtuelles et physiques
- Intégration avec le système de partenariat
- Suivi de l'historique des sessions

Cette fonctionnalité concrétise la mission de la plateforme en facilitant les interactions réelles entre partenaires d'apprentissage.

8 - Réalisations techniques

8.1 Extraits de code de composants les plus significatifs

L'application est structurée selon une architecture de microservices avec trois entités métier principales : User, Partnership et Session. Voici les extraits de code les plus significatifs de ces composants :

1 - MODÈLES DE DOMAINE

Modèle User :

Le modèle User est au cœur de l'application, représentant un utilisateur avec ses compétences, intérêts et objectifs d'apprentissage.

```
@Document(collection = "users")
@NoArgsConstructor
@AllArgsConstructor
public class User {
    @Id
    private String id;
    private String username;
    private String email;
    private String bio;
    private Date createdAt;
    private Date updatedAt;
    private ProfileCompletionStatus completionStatus = ProfileCompletionStatus.INITIAL;
    private String password;
    private String role;

    @DBRef
    private List<SkillTag> skills = new ArrayList<>();

    @DBRef
    private List<LearningObjective> learningObjectives = new ArrayList<>();

    @DBRef
    private List<InterestTag> interests = new ArrayList<>();

    private List<String> partnershipIds = new ArrayList<>();

    // Getters and setters...
}
```

Modèle Partnership :

Le modèle Partnership représente une relation de mentorat ou d'apprentissage entre deux utilisateurs.

```
@Entity
@Table(name = "partnerships")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Partnership {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "requester_id", nullable = false)
    private String requesterId;

    @Column(name = "requested_id", nullable = false)
    private String requestedId;

    @Enumerated(EnumType.STRING)
    private PartnershipStatus status;

    private String message;

    @CreationTimestamp
    private LocalDateTime createdAt;

    @UpdateTimestamp
    private LocalDateTime updatedAt;

    private LocalDateTime acceptedAt;

    private LocalDateTime endedAt;

    @OneToMany(mappedBy = "partnership", cascade = CascadeType.ALL)
    private List<PartnershipGoal> goals = new ArrayList<>();
}
```

Le
entre

```
@Entity
@Table(name = "sessions")
@EntityListeners(AuditingEntityListener.class)
public class Session {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, length = 200)
    private String title;

    @Column(columnDefinition = "TEXT")
    private String description;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 16)
    private SessionType type;

    @Column(name = "location_or_link", length = 512)
    private String locationOrLink;

    @Column(name = "start_at", nullable = false)
    private LocalDateTime startAt;

    @Column(name = "end_at", nullable = false)
    private LocalDateTime endAt;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 16)
    private SessionStatus status = SessionStatus.PLANNED;

    @Column(name = "partnership_id", nullable = false)
    private Long partnershipId;

    @Column(name = "organizer_id", nullable = false, length = 64)
    private String organizerId;

    @Column(name = "partner_id", nullable = false, length = 64)
    private String partnerId;

    // Getters and setters...
}
```

Modèle Session :

modèle Session
représente une
rencontre planifiée
deux partenaires.

2 - SERVICE MÉTIERS

UserServiceImpl:

Le service UserServiceImpl implémente la logique métier liée aux utilisateurs, notamment l'inscription, la mise à jour du profil et la gestion des compétences et intérêts.

```
@Service
@Slf4j
public class UserServiceImpl implements UserService {
    // Méthode d'inscription d'un utilisateur
    @Override
    public User registerUser(User userProfile) {
        // Vérifier si l'email existe déjà
        if (userRepository.existsByEmail(userProfile.getEmail())) {
            throw new IllegalArgumentException("Un utilisateur avec cet email existe déjà");
        }

        // Vérifier si le nom d'utilisateur existe déjà
        if (userRepository.existsByUsername(userProfile.getUsername())) {
            throw new IllegalArgumentException("Ce nom d'utilisateur est déjà pris");
        }

        // Encoder le mot de passe
        userProfile.setPassword(passwordEncoder.encode(userProfile.getPassword()));

        // Initialiser les dates
        Date now = new Date();
        userProfile.setCreateAt(now);
        userProfile.setUpdatedAt(now);

        // Initialiser le statut de complétion
        userProfile.setCompletionStatus(ProfileCompletionStatus.INITIAL);

        return userRepository.save(userProfile);
    }

    // Méthode de mise à jour des compétences
    @Override
    public User updateSkills(String userId, List<SkillTag> skills) {
        User user = getUserId(userId)
            .orElseThrow(() -> new IllegalArgumentException("Utilisateur non trouvé avec l'ID: " + userId));
    }
}
```

```

// Traiter les compétences
if (skills != null && !skills.isEmpty()) {
    List<SkillTag> processedSkills = new ArrayList<>();

    for (SkillTag skill : skills) {
        if (skill.getId() == null) {
            // Vérifier si une compétence avec ce nom existe déjà
            Optional<SkillTag> existingSkill = skillService.getSkillByName(skill.getName());

            if (existingSkill.isPresent()) {
                // Utiliser la compétence existante
                processedSkills.add(existingSkill.get());
            } else {
                // Créer une nouvelle compétence
                SkillTag newSkill = new SkillTag();
                newSkill.setName(skill.getName());
                newSkill.setCategory(skill.getCategory());
                newSkill.setPredefined(false);

                SkillTag savedSkill = skillService.createSkill(newSkill);
                processedSkills.add(savedSkill);
            }
        } else {
            // Vérifier que la compétence existe
            Optional<SkillTag> existingSkill = skillService.getSkillById(skill.getId());
            if (existingSkill.isPresent()) {
                processedSkills.add(existingSkill.get());
            }
        }
    }

    user.setSkills(processedSkills);
}

// Mettre à jour le statut de compléction si nécessaire
if (user.getCompletionStatus() == ProfileCompletionStatus.PERSONAL_INFO_COMPLETED) {
    user.setCompletionStatus(ProfileCompletionStatus.SKILLS_COMPLETED);
}

user.setUpdatedAt(new Date());
return userRepository.save(user);
}

```

PartnershipServiceImpl

Le service PartnershipServiceImpl gère la création et la gestion des partenariats entre utilisateurs

```

@Service
public class PartnershipServiceImpl implements PartnershipService {
    // Méthode de création d'une demande de partenariat
    @Override
    public ApiResponse<PartnershipDTO> createPartnershipRequest(CreatePartnershipRequestDTO request) {
        try {
            // Validations de base
            if (request.getRequesterId() == null || request.getRequesterId().isBlank()) {
                return new ApiResponse<>(false, "L'identifiant du demandeur est requis", null);
            }
            if (request.getRequestedId() == null || request.getRequestedId().isBlank()) {
                return new ApiResponse<>(false, "L'identifiant du destinataire est requis", null);
            }
            if (request.getRequesterId().equals(request.getRequestedId())) {
                return new ApiResponse<>(false, "Vous ne pouvez pas créer un partenariat avec vous-même", null);
            }

            // Empêcher les doublons
            if (existsPartnershipBetweenUsers(request.getRequesterId(), request.getRequestedId())) {
                return new ApiResponse<>(false, "Un partenariat existe déjà entre ces utilisateurs", null);
            }

            // Créer un nouveau partenariat
            Partnership partnership = new Partnership();
            partnership.setRequesterId(request.getRequesterId());
            partnership.setRequestedId(request.getRequestedId());
            partnership.setMessage(request.getMessage());
            partnership.setStatus(PartnershipStatus.PENDING);
            partnership.setCreatedAt(LocalDateTime.now());
            partnership.setUpdatedAt(LocalDateTime.now());
        }
    }
}

```

```

// Sauvegarder le partenariat
partnership = partnershipRepository.save(partnership);

PartnershipDTO dto = convertToDTO(partnership);
return new ApiResponse<>(true, "Demande de partenariat créée avec succès", dto);
} catch (Exception e) {
    return new ApiResponse<>(false, "Erreur lors de la création de la demande de partenariat", null);
}

// Méthode pour accepter un partenariat
@Override
public ApiResponse<PartnershipDTO> acceptPartnership(Long partnershipId) {
    try {
        Partnership partnership = updatePartnershipStatus(partnershipId, PartnershipStatus.ACCEPTED);
        PartnershipDTO dto = convertToDTO(partnership);
        return new ApiResponse<>(true, "Partenariat accepté avec succès", dto);
    } catch (IllegalArgumentException e) {
        return new ApiResponse<>(false, e.getMessage(), null);
    }
}

```

SessionServiceImpl

Le service SessionServiceImpl gère la création et la gestion des sessions entre partenaires.

```

@Service
@Transactional
public class SessionServiceImpl implements SessionService {
    // Méthode de création d'une session
    @Override
    public SessionResponse create(CreateSessionRequest request) {
        if (request.getOrganizerId() != null && request.getOrganizerId().equals(request.getPartnerId())) {
            throw new IllegalStateException("organizerId and partnerId must be different");
        }
        // Validate partnership exists and is ACCEPTED, and members match organizer/partner
        ApiResponse<PartnershipDTO> resp = partnershipClient.getPartnership(request.getPartnershipId());
        PartnershipDTO p = resp != null ? resp.getData() : null;
        if (p == null || p.getStatus() == null || !"ACCEPTED".equals(p.getStatus())) {
            throw new IllegalStateException("Partnership must be ACCEPTED");
        }
        String organizerStr = request.getOrganizerId();
        String partnerStr = request.getPartnerId();
        boolean matches = (organizerStr.equals(p.getRequesterId()) && partnerStr.equals(p.getRequestedId()))
            || (organizerStr.equals(p.getRequestedId()) && partnerStr.equals(p.getRequesterId()));
        if (!matches) {
            throw new IllegalStateException("Organizer/Partner do not match partnership members");
        }
        Session session = new Session();
        session.setTitle(request.getTitle());
        session.setDescription(request.getDescription());
        session.setType(request.getType());
        session.setLocationOrLink(request.getLocationOrLink());
        session.setStartAt(request.getStartAt());
        session.setEndAt(request.getEndAt());
        session.setStatus(SessionStatus.PLANNED);
        session.setPartnershipId(request.getPartnershipId());
        session.setOrganizerId(request.getOrganizerId());
        session.setPartnerId(request.getPartnerId());
        Session saved = sessionRepository.save(session);
        return toResponse(saved);
    }
}

```

```

// Méthode pour compléter une session
@Override
public SessionResponse complete(Long id, String requesterId) {
    Session session = getSessionOrThrow(id);
    assertParticipant(session, requesterId);
    assertPlanned(session);
    session.setStatus(SessionStatus.COMPLETED);
    return toResponse(session);
}

```

8.2 Extrait de code de composants d'accès aux données

Quelques exemple d'extraits de code.

1 - *Repositories pour MongoDB (User Service)*

Le microservice User Service utilise MongoDB comme base de données. Voici les repositories principaux :

UserRepository

```
@Repository
public interface UserRepository extends MongoRepository<User, String> {
    Optional<User> findByEmail(String email);
    Optional<User> findByUsername(String username);
    List<User> findBySkillsName(String skillName);
    List<User> findByInterestsName(String interestName);
    boolean existsByEmail(String email);
    boolean existsByUsername(String username);
    List<User> findByUsernameContainingIgnoreCaseOrEmailContainingIgnoreCase(String username, String email);
}
```

Ce repository gère les opérations CRUD pour l'entité User et définit des méthodes de recherche spécifiques comme la recherche par email, nom d'utilisateur, compétences ou intérêts.

SkillTagRepository

```
public interface SkillTagRepository extends MongoRepository<SkillTag, String> {
    boolean existsByName(String name);
    boolean existsByNameAndCategory(String name, String category);
    SkillTag findByName(String name);
    SkillTag findByNameAndCategory(String name, String category);
    List<SkillTag> findByNameContainingIgnoreCase(String name);
}
```

Ce repository gère les compétences des utilisateurs avec des méthodes pour vérifier l'existence et rechercher par nom ou catégorie.

2 - *Repositories pour MySQL (Partnership et Session Services)*

Les microservices Partnership et Session utilisent MySQL comme base de données.

PartnershipRepository

```
@Repository
public interface PartnershipRepository extends JpaRepository<Partnership, Long> {
    List<Partnership> findByRequesterIdOrRequestedId(String requesterId, String requestedId);
    boolean existsByRequesterIdAndRequestedId(String requesterId, String requestedId);
}
```

Ce repository gère les partenariats entre utilisateurs avec des méthodes pour rechercher par demandeur ou destinataire.

SessionRepository

```
public interface SessionRepository extends JpaRepository<Session, Long> {
    List<Session> findByOrganizerIdOrPartnerId(String organizerId, String partnerId);
    List<Session> findByOrganizerIdOrPartnerIdAndStatus(String organizerId, String partnerId, SessionStatus status);

    @Query("select s from Session s where (s.organizerId = :userId or s.partnerId = :userId) and s.startAt >= :from and s.endAt <= :to")
    List<Session> findUserSessionsBetween(@Param("userId") String userId,
                                          @Param("from") LocalDateTime from,
                                          @Param("to") LocalDateTime to);
}
```

Ce repository gère les sessions d'apprentissage avec des méthodes pour rechercher par organisateur, partenaire, statut ou période.

3 - *Configuration des bases de données :*

Configuration MongoDB (User Service) :

Cette configuration définit la connexion à MongoDB pour le User Service.

Configuration MySQL (Partnership Service)

Cette configuration définit la connexion à MySQL pour le Partnership Service.

Configuration MySQL (Session Service)

```

spi:
  spring:
    application:
      name: session-service
    datasource:
      url: jdbc:mysql://localhost:3308/skillmates_session?createDatabaseIfNotExist=true&useSSL=false&allowPublicKeyRetrieval=true
      username: session_user
      password: session_pass
      driver-class-name: com.mysql.cj.jdbc.Driver
    jpa:
      hibernate:
        ddl-auto: update
        show-sql: true
    eureka:
      properties:
        hibernate:
          dialect: org.hibernate.dialect.MySQLDialect
          format_sql: true
  defaultZone: http://localhost:8761/eureka/
  register-with-eureka: true
  fetch-registry: true

```

Cette configuration définit la connexion à MySQL pour le Session Service.

4 - Initialisation des données

Le projet comprend également un composant d'initialisation des données pour précharger des valeurs prédéfinies dans la base de données :

```

@Configuration
public class DataInitializer {
    @Bean
    public CommandLineRunner initDatabase(SkillTagRepository skillTagRepository,
                                         InterestTagRepository interestTagRepository) {
        return args -> {
            // Vérifier si des données existent déjà
            if (skillTagRepository.count() == 0) {
                logger.info("Initialisation des compétences prédefinies");
                initializeSkills(skillTagRepository);
            }

            if (interestTagRepository.count() == 0) {
                logger.info("Initialisation des intérêts prédefinis");
                initializeInterests(interestTagRepository);
            }
        };
    }

    private void initializeSkills(SkillTagRepository repository) {
        // Catégorie Programmation
        List<SkillTag> programmingSkills = Arrays.asList(
            new SkillTag(null, "Java", "Programmation", true),
            new SkillTag(null, "Python", "Programmation", true),
            // autres compétences...
        );

        // Sauvegarder toutes les compétences
        repository.saveAll(programmingSkills);
    }
}

```

5 - Configuration Docker des bases de données

Dans le fichier docker-compose.yaml:

```

services:
  mongodb-user-service:
    image: mongo:latest
    container_name: mongodb-users
    ports:
      - "27019:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: skillMates
      MONGO_INITDB_ROOT_PASSWORD: skillMates_password
      MONGO_INITDB_DATABASE: skillMates
    volumes:
      - mongodb-users-data:/data/db

  mysql-partnership:
    image: mysql:8.0
    container_name: mysql-partnership
    ports:
      - "3307:3306"
    environment:
      MYSQL_DATABASE: partnership
      MYSQL_ROOT_PASSWORD: password
      MYSQL_PASSWORD: password
    volumes:
      - mysql-partnership-data:/var/lib/mysql

  mysql-session:
    image: mysql:8.0
    container_name: mysql-session
    ports:
      - "3308:3306"
    environment:
      MYSQL_DATABASE: skillmates_session
      MYSQL_USER: session_user
      MYSQL_PASSWORD: session_pass
      MYSQL_ROOT_PASSWORD: root
    volumes:
      - mysql-session-data:/var/lib/mysql

```

Cette configuration Docker définit les conteneurs pour les bases de données MongoDB et MySQL utilisées

par les différents microservices.

L'architecture d'accès aux données de SkillMates illustre une approche polyglotte, utilisant différentes technologies de base de données selon les besoins spécifiques de chaque microservice. MongoDB est utilisé pour le User Service en raison de sa flexibilité pour stocker des données utilisateur avec des structures variables (compétences, intérêts, objectifs), tandis que MySQL est utilisé pour les Partnership et Session Services qui nécessitent des relations plus structurées et des transactions ACID.

Choix des bases de données

L'application utilise une approche polyglotte pour la persistance des données, avec deux types de bases de données :

MongoDB pour le User Service :

Justifications :

- **Schéma flexible** : Idéal pour les données utilisateur qui peuvent évoluer fréquemment (ajout de nouveaux champs, compétences, intérêts)
- **Requêtes riches** : Facilite les recherches complexes sur les compétences et intérêts
- **Modèle document** : Correspond naturellement à la structure des profils utilisateurs avec leurs listes de compétences et d'intérêts

MySQL pour Partnership et Session Services :

Justifications :

- **Intégrité référentielle** : Garantit la cohérence des relations entre partenariats et sessions
- **Transactions ACID** : Essentielles pour les opérations critiques comme la création de partenariats
- **Modèle relationnel** : Adapté à la structure plus rigide et relationnelle des partenariats et sessions

8.3 Deploiement/OPS

1. Vue d'ensemble de la stratégie de conteneurisation

L'application SkillMates adopte une approche de conteneurisation complète basée sur Docker pour tous ses composants. Cette stratégie permet de créer un environnement cohérent, portable et facilement déployable, tout en isolant chaque service et ses dépendances.

Conteneurisation et déploiement

Justifications :

- **Cohérence environnementale** : Garantit que l'application fonctionne de manière identique dans tous les environnements
- **Isolation** : Chaque service s'exécute dans son propre conteneur avec ses dépendances
- **Orchestration simplifiée** : Facilite le déploiement et la gestion des services

Jib pour la création d'images Docker :

Justifications :

- **Optimisation** : Crée des images Docker optimisées sans nécessiter de Dockerfile
- **Intégration Maven** : S'intègre directement dans le cycle de build Maven
- **Multi-architecture** : Support pour différentes architectures (ARM64/AMD64) Car je travail sur Mac M1

2. Extrait de la structure du fichier docker-compose.yaml

Le fichier docker-compose est le point central de la configuration de déploiement, orchestrant l'ensemble des conteneurs et leurs interactions.

Bases de données

```
mongodb-user-service:
  image: mongo:latest
  container_name: mongodb-users
  ports:
    - "27019:27017"
  environment:
    MONGO_INITDB_ROOT_USERNAME: skillMates
    MONGO_INITDB_ROOT_PASSWORD: skillMates_password
    MONGO_INITDB_DATABASE: skillMates
  volumes:
    - mongodb-users-data:/data/db
  networks:
    - skillmates-network
  healthcheck:
    test: ["CMD", "mongosh", "--eval", "db.adminCommand('ping')"]
    interval: 10s
    timeout: 5s
    retries: 5
    start_period: 30s
```

Caractéristiques importantes :

- **Persistance des données** : Utilisation de volumes Docker pour garantir la persistance entre les redémarrages
- **Contrôle de santé** : Healthchecks configurés pour s'assurer que les bases de données sont opérationnelles avant le démarrage des services dépendants
- **Isolation réseau** : Tous les conteneurs sont connectés au même réseau Docker

Services d'infrastructure

```
config-server:
  image: noann/skillmates-config-server:latest
  container_name: config-server
  ports:
    - "8888:8888"
  networks:
    - skillmates-network
  healthcheck:
    test: ["CMD", "wget", "--spider", "-q", "http://localhost:8888/actuator/health"]
    interval: 10s
    timeout: 5s
    retries: 5
    start_period: 20s

discovery-service:
  image: noann/skillmates-discovery-service:latest
  container_name: discovery-service
  ports:
    - "8761:8761"
  environment:
    SPRING_CLOUD_CONFIG_URI: http://config-server:8888
  networks:
    - skillmates-network
  depends_on:
    config-server:
      condition: service_healthy
```

Caractéristiques importantes :

- **Ordre de démarrage** : Utilisation de « depends_on » avec condition `service_healthy` pour garantir que les services démarrent dans le bon ordre
- **Configuration externalisée** : Variables d'environnement pour paramétrier les services

9 - Tests et qualité

9.1 Plan de tests

1 - Tests d'intégration

But : Vérifier que différents composants ou systèmes fonctionnent correctement ensemble.

Exemple dans le projet :

```
53 @Test & NoannNass
54 @WithMockUser
55 void list_sessions_view_renderer_with_user() throws Exception {
56     org.mockito.Mockito.when(userInfoSession.getUserId()).thenReturn("U1");
57     SessionResponse s = new SessionResponse();
58     s.setId(1L);
59     s.setOrganizerId("U1");
60     s.setPartnerId("U2");
61     org.mockito.Mockito.when(sessionClient.list(any(), any(SessionStatus.class), any(java.time.LocalDateTime.class), any(java.time.LocalDateTime.class)))
62         .thenReturn(List.of(s));
63
64     mvc.perform(get(uriTemplate: "/sessions").param(name: "status", SessionStatus.PLANNED.name()))
65         .andExpect(status().isOk())
66         .andExpect(model().attributeExists(names: "sessions"));
67 }
```

Ce test vérifie que :

- La vue des sessions s'affiche correctement pour un utilisateur authentifié
- Le contrôleur renvoie un code HTTP 200 (OK)
- Le modèle contient un attribut "sessions" qui sera utilisé par la vue Thymeleaf

MockMvc : composant essentiel de Spring Test qui permet de tester les contrôleurs web sans déployer l'application sur un serveur. Il simule les requêtes HTTP et permet de vérifier les réponses.

SpringBootTest & WebMvcTest

```
@SpringBootTest(properties = { & NoannNass
    "spring.cloud.config.enabled=false",
    "spring.cloud.bootstrap.enabled=false",
    "eureka.client.enabled=false",
    "spring.cloud.discovery.enabled=false",
    "spring.cloud.openfeign.enabled=false",
    "feign.client.enabled=false",
    "spring.main.allow-bean-definition-overriding=true"
})
```

SpringBootTest :

- Charge le contexte complet de l'application Spring
- Permet de tester l'intégration entre différentes couches de l'application

WebMvcTest:

- Charge uniquement la couche web (contrôleurs, filtres, etc.)
- Idéal pour tester les contrôleurs en isolation

2- Test unitaire

But : Tester individuellement les plus petites unités de code (méthodes, classes) en isolation complète.

```

56     @Test  ⚒ NoannNass
57     void post_login_without_csrf_forbidden() throws Exception {
58         mvc.perform(post( uriTemplate: "/Login")
59             .contentType(MediaType.APPLICATION_FORM_URLENCODED)
60             .param( name: "username", ...values: "john@example.com")
61             .param( name: "password", ...values: "x"))
62             .andExpect(status().isForbidden());
63     }

```

9.2 Technologie utilisés pour les test

JUnit

JUnit est le framework de test unitaire standard pour Java

On l'observe avec les Annotations permettant de simplifier l'écriture des test par exemple « @Test » pour marquer une méthode comme étant un test, et bien d'autres...

Mockito

Mockito est une bibliothèque de mocking pour Java, permettant de créer des objets simulés (mocks) pour isoler le code testé

Avantages :

- Isolation complète du code testé
- API fluide et intuitive

Jacoco

JaCoCo (Java Code Coverage) est un outil d'analyse de couverture de code qui mesure quelle partie du code est exécutée pendant les tests.

Avantages :

- Génération de rapports détaillés (HTML, XML, CSV)
- Intégration avec Maven
- Identification des zones de code non testées

10 - Conclusion

La mise en oeuvre de ce projet m'a permis de mettre à profit les compétences que l'on a parcouru tout du long de la formation dans un projet concret fait de bout en bout. Je me suis rendu compte des enjeux de production et de sécurité.

Par la suite j'envisage de créer un service de messagerie avec Kafka pour sa facilité de compatibilité avec Java et SpringBoot, je pourrais alors rendre ces boutons de la Sidebar présent sur l'interface web fonctionnels.