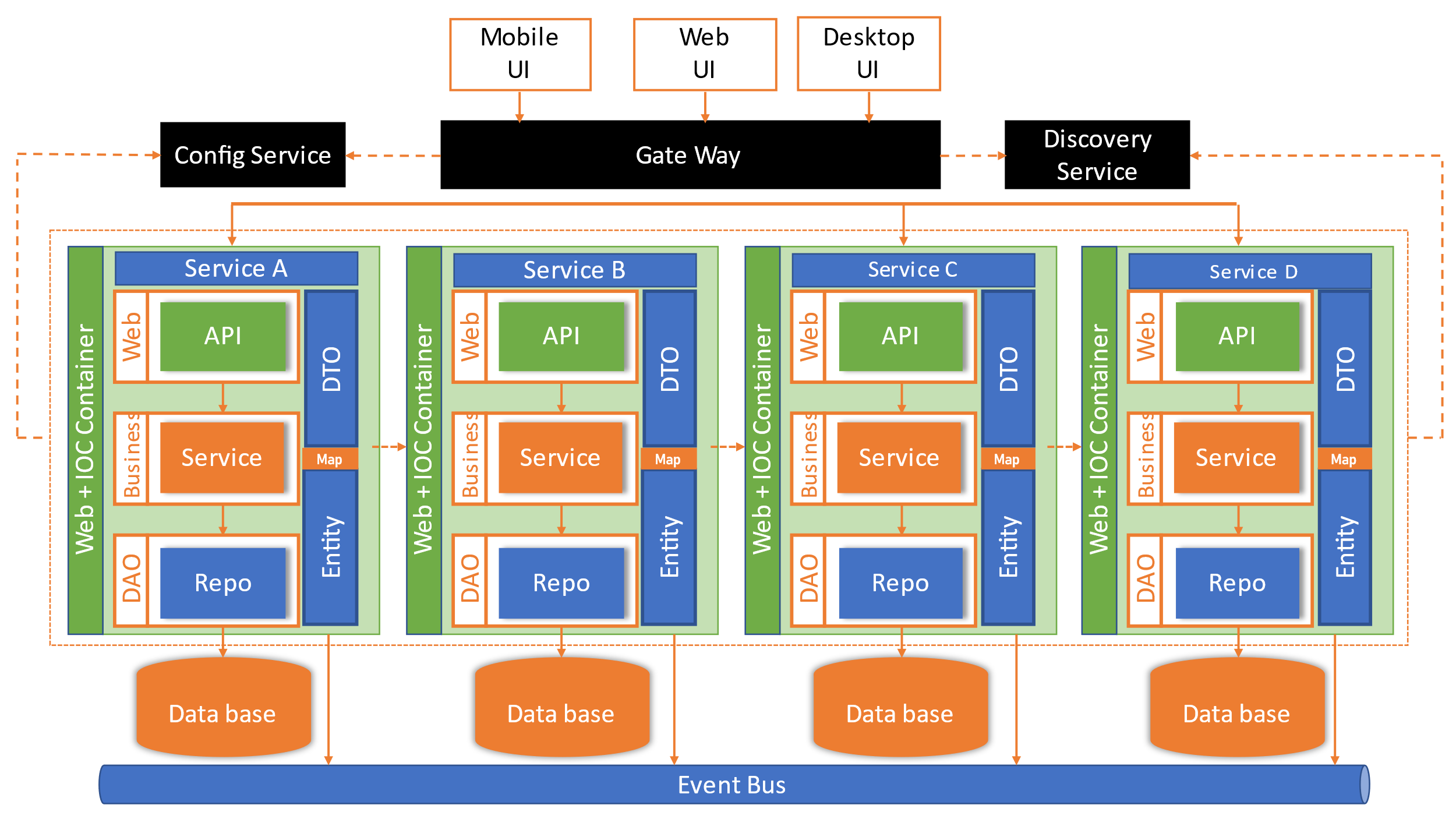
**PHASE 2: COMPREHENSION DU SUJET**

## I – Les micro services

### 1. Définition et concepts des micro services

Les micro services sont une **architecture logicielle** qui permettent de développer des applications en les découpant en services indépendants et spécialisés. Chaque service peut être développé, testé, déployé et géré de manière autonome, ce qui facilite la maintenance et l'évolution de l'application. Cette architecture favorise également la scalabilité horizontale et la résilience de l'application en cas de panne d'un service. Les micro services sont de plus en plus populaires pour la conception d'applications modernes, notamment dans le contexte du développement agile et de DevOps.

### 2. Modèle d’architecture des micro services



Dans cette architecture, on a les éléments suivants :

* Web UI, Mobile UI et Desktop UI qui représentent l’interface qui sera affiche à l’utilisateur qu’il soit sur une application web, mobile ou desktop et celle-ci est transparente à l’ensemble des micro services car l’utilisateur n’a pas connaissance du fait que l’application ait été découpée en plusieurs services.
* **La config service :** ce service contient toutes les configurations de l’ensemble des micro services et chaque microservice à son démarrage ira à son niveau demander sa configuration.
* **La Discovery Service :** une fois que les microservices aient récupéré leur configuration auprès de la **config service**, elles ont besoin de les stocker dans un annuaire qui est le service de découverte dont le rôle sera de les conserver et faciliter la communication entre l’interface utilisateur et les autres microservices.
* **La Gateway Service :** ce service a pour rôle de gérer la communication entre l’interface utilisateur et les microservices. Ainsi, lorsqu’un utilisateur va adresser une requête **HTTP** a un service particulier**,** la Gateway service ira au niveau de la Discovery pour demander l’adresse du microservice en question et par la suite ira dans le service récupérer l’information pour afficher à l’utilisateur.
* Les microservices sont base sur une architecture 3-tier dont les composantes sont les suivante :
* **La couche Web :** c’est sur cette couche que sera développé l’API (Application Programmation Interface) qui sera consomme par les utilisateurs et cette meme API permettra aux systèmes externes de pourvoir communiquer avec nos microservices.
* **La couche service ou métier :** elle contient les services qui s’occupent des traitements.
* **La couche DAO (Data Access Object) :** elle va s’occuper de gérer les requêtes permettant d’accéder à la base de données.
* **Data Base (Base de données) :** elle va stocker les données de chaque microservice grâce à un SGBD qu’il soit relationnel ou NoSQL.
* Afin d’assurer la communication entre les microservices, il existe principalement deux modèles de communication à savoir :
* **La communication synchrone :** ici, si un microservice a besoin d’un autre microservice pour parfaire son fonctionnement, on va associer a se microservice les informations sur le microservice en question en utilisant **REST. Open Feign** est une solution implémentée par Spring Boot pour gérer cette situation
* **La communication asynchrone :** ici, plutôt que d’utiliser une API, on va créer un **broker ou bus d’évènement (KAFKA, RabbitMQ)** qui va gérer les topics d’évènement. Ce bus d’évènement est vu comme une base de données global des microservices. Au démarrage, chaque microservice fera un subscribe (envoie ces données) vers le bus d’évènement pour permettre aux autres de pourvoir y récupérer lorsqu’il le sollicite.

## II – DevOps et CI/CD

DevOps est un ensemble de pratiques visant à combiner les équipes de développement et d'opérations pour permettre une livraison plus rapide et plus fiable des logiciels. Le CI/CD (Continuous Integration / Continuous Deployment) est une pratique qui permet une intégration continue du code produit par les développeurs dans un référentiel centralisé et une livraison continue de l'application aux utilisateurs finaux. Dans cette partie, nous allons examiner comment les principes DevOps et CI/CD peuvent être appliqués à notre projet de gestion de notes basé sur une architecture à microservices.

### Présentation du pipeline CI/CD

Le pipeline CI/CD est un processus de déploiement continu qui permet de livrer rapidement des applications de haute qualité en automatisant la construction, les tests et le déploiement de logiciels. Il se compose de plusieurs étapes qui incluent la compilation du code, les tests unitaires, les tests d'intégration, le déploiement et la livraison continue.

Le pipeline CI/CD permet de garantir la qualité du code tout au long du processus de développement et de réduire les risques d'erreurs lors du déploiement. Il facilite également la collaboration entre les membres de l'équipe de développement et permet de détecter les erreurs dès le début du processus de développement. Pour mettre en place un pipeline CI/CD, il est nécessaire d'utiliser des outils tels que Git/GitLab pour la gestion de versions, Jenkins pour l'intégration continue, Ansible pour l'automatisation du déploiement, Docker pour la conteneurisation, et Kubernetes pour l'orchestration de conteneurs.

### Les outils DevOps

La mise en place d'un pipeline CI/CD nécessite l'utilisation de différents outils. Voici une présentation des principaux outils qui seront utilisés :

* Git : système de contrôle de version de code source qui permet de stocker et de gérer les différentes versions du code. Il est souvent utilisé comme outil de base pour la gestion du code source dans les pipelines CI/CD.
* Jenkins : outil open source de déploiement continu. Il permet de créer et de configurer des pipelines CI/CD, de les exécuter automatiquement et de surveiller leur progression. Jenkins offre également la possibilité de déployer des applications sur différents environnements, tels que des serveurs de test ou de production.
* Docker : plateforme de conteneurs qui permet d'isoler les applications et les services dans des conteneurs. Cette isolation permet de faciliter la gestion des applications et leur déploiement dans les différents environnements. Docker est souvent utilisé dans les pipelines CI/CD pour créer et gérer les conteneurs nécessaires à l'exécution des applications.
* Kubernetes : outil de gestion de conteneurs open source qui permet de déployer et de gérer des conteneurs à grande échelle. Il offre des fonctionnalités de haute disponibilité, de répartition de charge et de mise à l'échelle automatique. Kubernetes est souvent utilisé pour déployer et gérer les conteneurs créés avec Docker.
* Ansible : outil open source de gestion de configuration et de déploiement d'applications. Il permet de déployer des applications sur des environnements locaux ou distants, d'automatiser la configuration de l'infrastructure et de gérer les mises à jour du système.
* GitLab : plateforme de gestion de code source qui intègre des fonctionnalités de CI/CD. Il permet de créer et de gérer des pipelines de déploiement continu, de surveiller leur progression et d'automatiser les tests et le déploiement des applications.

## III - Cas d’étude : mise en place d’une application de gestion des notes

* 1. **Description**

L'application de gestion de notes universitaires vise à faciliter la gestion des notes et des résultats des étudiants au sein d'une université. Elle permettra aux étudiants de consulter leurs résultats de manière centralisée, d'obtenir un aperçu de leur performance académique et d'accéder à des informations pertinentes concernant leurs cours et leurs évaluations.

L'application fournira également aux professeurs et aux administrateurs un moyen efficace de gérer les résultats des étudiants, de saisir et de mettre à jour les notes, ainsi que de générer des rapports et des statistiques.

* 1. **Besoins fonctionnels**

1. Authentification et rôles utilisateur :
   * Les étudiants, les professeurs et les administrateurs doivent pouvoir s'authentifier pour accéder à l'application.
   * Chaque utilisateur doit avoir un rôle attribué (étudiant, professeur ou administrateur) avec des autorisations spécifiques.
2. Gestion des étudiants :
   * Les administrateurs doivent pouvoir ajouter, modifier et supprimer des étudiants dans la base de données.
   * Les informations des étudiants, telles que leur nom, leur numéro d'identification, leur programme d'études, doivent être enregistrées.
3. Gestion des cours et des évaluations :
   * Les professeurs doivent pouvoir créer, modifier et supprimer des cours.
   * Les cours doivent contenir des informations telles que le nom du cours, le nom du professeur, le programme d'études associé et les évaluations prévues.
   * Les évaluations peuvent inclure des examens, des devoirs, des projets, etc.
4. Saisie des notes :
   * Les professeurs doivent pouvoir saisir et mettre à jour les notes des étudiants pour chaque évaluation.
   * Les notes peuvent être attribuées sous forme de notes numériques ou de lettres, en fonction des critères de notation du cours.
5. Consultation des résultats :
   * Les étudiants doivent pouvoir consulter leurs résultats par cours et par évaluation.
   * Les résultats peuvent être affichés sous forme de notes numériques, de lettres ou de pourcentages.
6. Génération de rapports :
   * Les professeurs et les administrateurs doivent pouvoir générer des rapports sur les performances des étudiants, les moyennes de classe, les statistiques, etc.
   * Les rapports peuvent être exportés au format PDF.
7. Notifications et alertes :
   * Les étudiants doivent recevoir des notifications et des alertes concernant les nouvelles évaluations, les résultats mis à jour, les dates limites, etc.
8. Convivialité et interface utilisateur :
   * L'application doit avoir une interface conviviale et intuitive pour faciliter la navigation et l'utilisation par les utilisateurs.
   * Les informations doivent être organisées de manière claire et compréhensible.

Il est important de noter que ces besoins fonctionnels sont une base pour concevoir une application de gestion de notes universitaires, et peuvent être adaptés en fonction des spécifications et des exigences spécifiques de votre université.

* 1. **Conception architecturale**

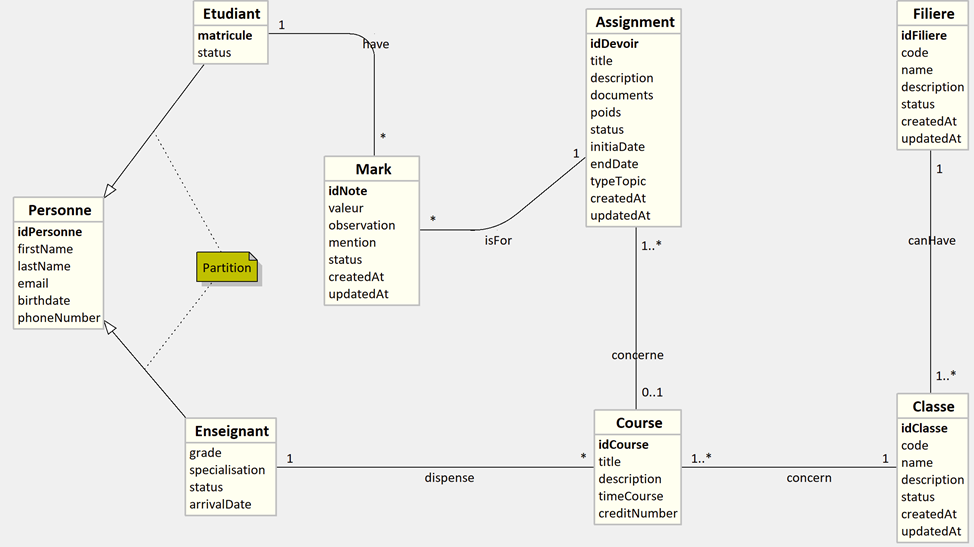
La conception architecturale est le processus de planification et de conception d'une architecture logicielle pour répondre aux exigences d'un système informatique. Elle permet de définir la structure du système, les interactions entre ses composants, ainsi que les contraintes et les exigences fonctionnelles et non-fonctionnelles. La qualité de la conception architecturale est essentielle pour garantir la robustesse, la fiabilité et la maintenabilité du système sur le long terme.

1. **Architecture logique**

Il a pour but de décrire le fonctionnement général de l’application. Dans notre cas, nous allons utiliser l’approche UML (**Unified Modeling Language**) qui permet de faire la modélisation d’un système en se basant sur un ensemble de diagramme.

### Le diagramme des classes

Il permet de spécifier la structure et les liens entre les objets dont le système est composé. Le schéma ci-dessous présente un aperçu du diagramme de classe qui a été utilisé lors de la phase d’analyse et ayant permis de générer les éléments de base suivant l’architecture définit plus haut.



1. **Découpage en microservices**

