Web Statique & Dynamique

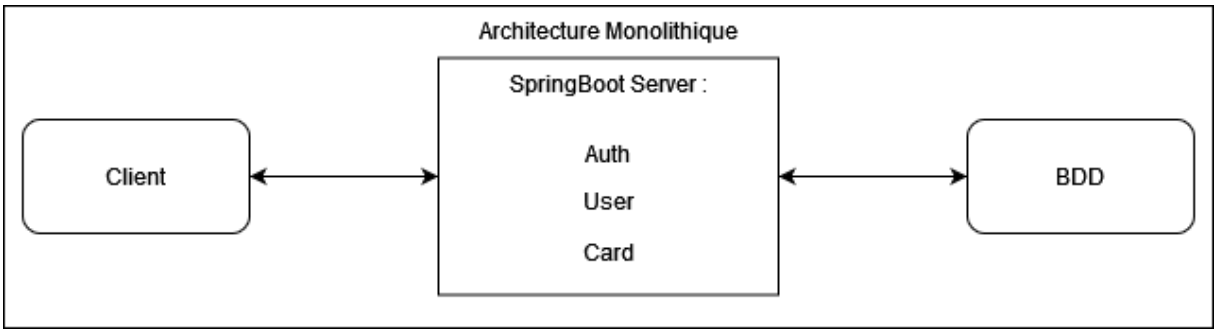
2023-2024

Théo CLERE, Maxime BROSSARD, Sandro SPINA, Ceif-Edine MAROUANI, Julien BUC
CPE LYON

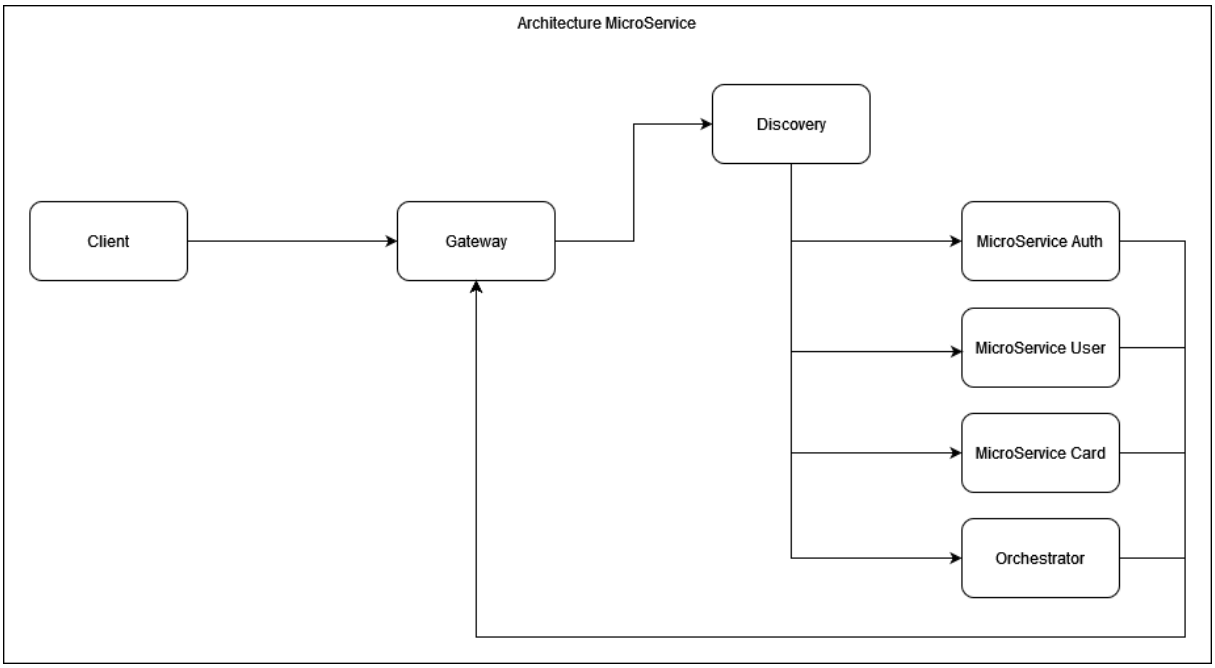
Evolution Architecturale

Au cours de cet atelier 3, nous avons pu reprendre les travaux effectués lors de l'atelier 2 : une application monolithique de gestion de cartes du jeu FIFA. Pendant cet atelier, nous avons transformé cette architecture monolithique en une architecture à micro-services.

Avant :



Après :



Ainsi, nous avons divisé les responsabilités du serveur Spring en trois nouveaux microservices :

- **MicroService Auth** :
  - **Responsabilités** : Gérer l'authentification et l'autorisation des utilisateurs.
  - **Fonctionnalités** :
    - Inscription des utilisateurs.
    - Connexion et déconnexion des utilisateurs.
    - Gestion des tokens JWT (JSON Web Tokens) pour l'authentification.

- **MicroService User** :
  - **Responsabilités** : Gérer les informations des utilisateurs et leur profil.
  - **Fonctionnalités** :
    - Création, lecture, mise à jour et suppression des profils utilisateurs.
    - Gestion du compte bancaire des utilisateurs.
    - Accès et mise à jour des informations personnelles (comme l'email, le nom, etc.).

- **MicroService Card** :
  - **Responsabilités** : Gérer les cartes du jeu FIFA.
  - **Fonctionnalités** :
    - Création, lecture, mise à jour et suppression des cartes.
    - Gestion des attributs des cartes (comme les statistiques des joueurs, les niveaux, etc.).
    - Marketplace

En plus de ces 3 micro-services, deux autres éléments nécessaires au bon fonctionnement de notre architecture. Ces éléments sont les suivants :

- **Discovery Server** :
  - **Responsabilités** : Gérer le recensement des MicroServices
  - **Fonctionnalités** :
    - Enregistrement des micro-services.
    - Gestion des informations relatives aux MicroServices
  - **Explications** : Le Discovery Server va fonctionner avec la technologie Eureka. Une simple configuration est nécessaire pour mettre en place ce serveur. Ensuite, il faudra configurer Eureka Discovery Client sur les micro-services. Il faudra lui renseigner l'adresse du Discovery Server et son port afin que les micro-services aillent s'enregistrer vers le serveur Eureka.

- **Gateway Server** :
  - **Responsabilités** : Router le trafic en fonction des requêtes utilisateurs
  - **Fonctionnalités** :
    - Définit les différentes utilisable par les requêtes utilisateurs
    - Gère le recensement des micro-services en fonction du contexte

- **Explications** : Le serveur Gateway va gérer le « routage » de nos requêtes vers les différents micro-services. Ce serveur va fonctionner avec le composant Gateway de Spring. La Gateway aura un lien direct avec le Discovery server afin de connaitre l'adresse ou redirigé les requêtes clients.

L'orchestrateur joue un rôle crucial dans la gestion des transactions distribuées à travers différents microservices. Dans le cadre de notre application qui gère l'achat et la vente de carte, l'orchestrateur assure la coordination de plusieurs services indépendants pour garantir que à l'ensemble de l'application une cohérence. Voici comment cela fonctionne généralement :

Fonctionnement de l'Orchestrateur dans une Transaction Distribuée

1. **Initiation de la Transaction :**

Le processus commence lorsqu'un utilisateur initie l'achat d'une carte via l'interface utilisateur. Cette action déclenche une requête vers le service gateway.

2. **Coordination des Services :**

Service Gateway : Le service gateway reçoit la requête et la redirige vers l'orchestrateur.

Orchestrateur : L'orchestrateur reçoit la requête de transaction et commence à coordonner les différents microservices impliqués.

3. **Vérification de l'Authentification :**

MicroService Auth : L'orchestrateur envoie une requête au microservice Auth pour vérifier que l'utilisateur est authentifié et autorisé à effectuer la transaction. Cela implique la vérification des tokens JWT et des permissions de l'utilisateur.

4. **Validation des Informations Utilisateur :**

MicroService User : L'orchestrateur contacte le microservice User pour récupérer et modifier les informations de l'utilisateur, notamment le solde du compte bancaire.

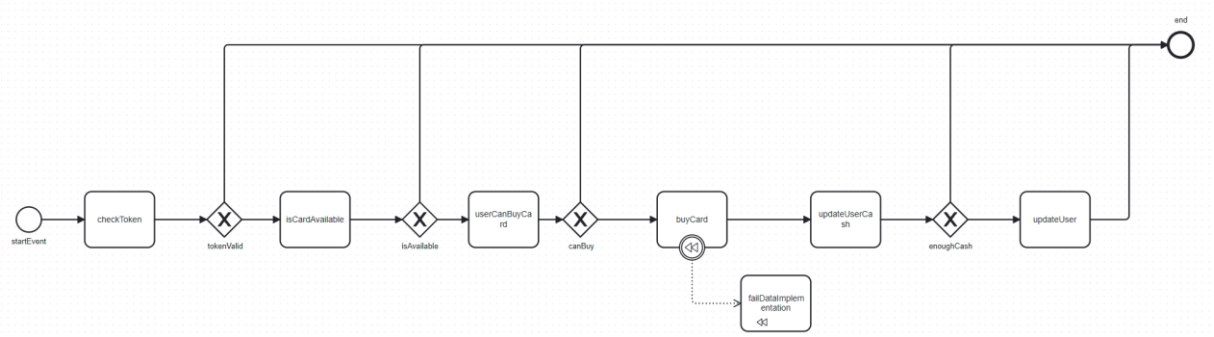
5. **Réservation de la Carte :**

MicroService Card : L'orchestrateur envoie une requête au microservice Card pour vérifier la disponibilité de la carte et la réserver.

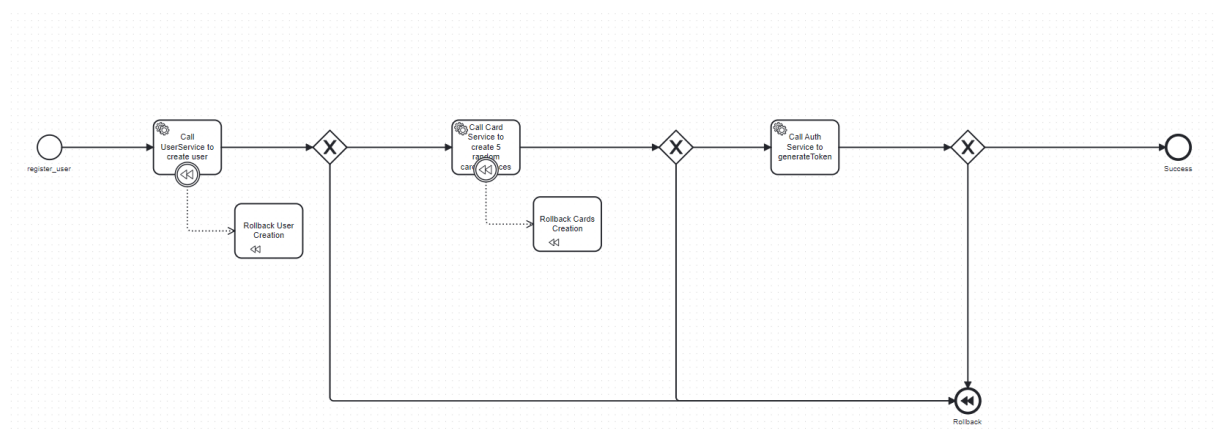
Gestion des Échecs :

En cas d'échec à n'importe quelle étape (comme une erreur de paiement ou une indisponibilité de la carte), l'orchestrateur doit gérer la compensation. Notre cas nous devons exécuter les étapes de compensation illustrées sur le schéma

Schéma camunda achat de carte :



Nous avons aussi un autre exemple sur la création d'un compte utilisateur :



Mise en place des tests unitaire :

Nous avons décidé de mettre en place des tests unitaires sur nos micro-services. Ces tests unitaires seront exécutés lors de la mise en marche d'un pipeline. Ces pipelines se déclencheront lors de changements sur la branche principale. Une fois que les différents jobs des pipelines auront terminé de s'exécuter, un état des lieux du code sera disponible sur SonarCloud.

1. **Tests unitaires du micro-service Card :**

   Dans notre cas, nous avons commencé à mettre en place nos tests unitaires sur le micro-service Card. Ces tests vont nous permettre de nous assurer du bon fonctionnement des différents composants et services de celui-ci.

```java
@Test
void testGetAllCardInstances() {
    when(cardInstancesRepository.findAll()).thenReturn(cardInstances);

    List<CardInstance> result = cardsInstanceService.getAllCardInstances();

    assertEquals(cardInstances, result);
    verify(cardInstancesRepository, times( wantedNumberOfInvocations: 1)).findAll();
}

@Test
void testGetCardInstanceById_Success() throws CardInstanceNotFoundException {
    when(cardInstancesRepository.findById(1L)).thenReturn(Optional.of(cardInstance));

    CardInstance result = cardsInstanceService.getCardInstanceById(1L);

    assertEquals(cardInstance, result);
    verify(cardInstancesRepository, times( wantedNumberOfInvocations: 1)).findById(1L);
}
```

Figure 1 – Exemple de tests sur nos services Cards

2. **Mise en place de la pipeline :**

   Une fois les tests mis en place dans le code, il nous fallait un moyen d'exécuter ces tests automatiquement à chaque changement sur la branche principale. C'est dans ce cadre-là que nous avons mis en place le job suivant dans une pipeline GitHub Actions pour exécuter les tests unitaires et les différents tests proposés par SonarCloud.

```yaml
- name: Build and analyze micro-service-card
  env:
    GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}  # Needed to get PR information, if any
    SONAR_TOKEN: ${{ secrets.SONAR_TOKEN_CARDS }}
  run: |
    cd micro-service-card
    mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=pulien_micro-service-card sonar:sonar -Pcoverage
```

Figure 2 - Job de test du Micro-Service Card

3. Résultat des tests SonarCloud et unitaire :

Une fois ces différentes batteries de tests effectuées, un rapport SonarCloud est disponible. Voici le rapport des tests du micro-service Card.

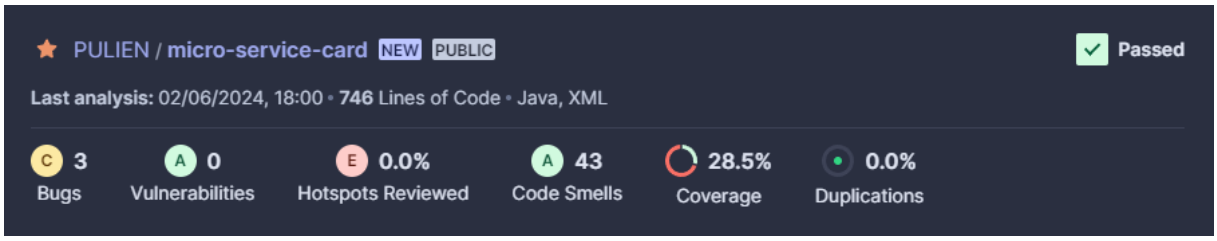


Figure 3 - Rapport SonarCloud du micro-service

Sur la capture d'écran ci-dessus, nous avons une couverture de code de 28%. Ce pourcentage assez bas s'explique par un manque de temps pour réaliser et mettre en place ces tests.