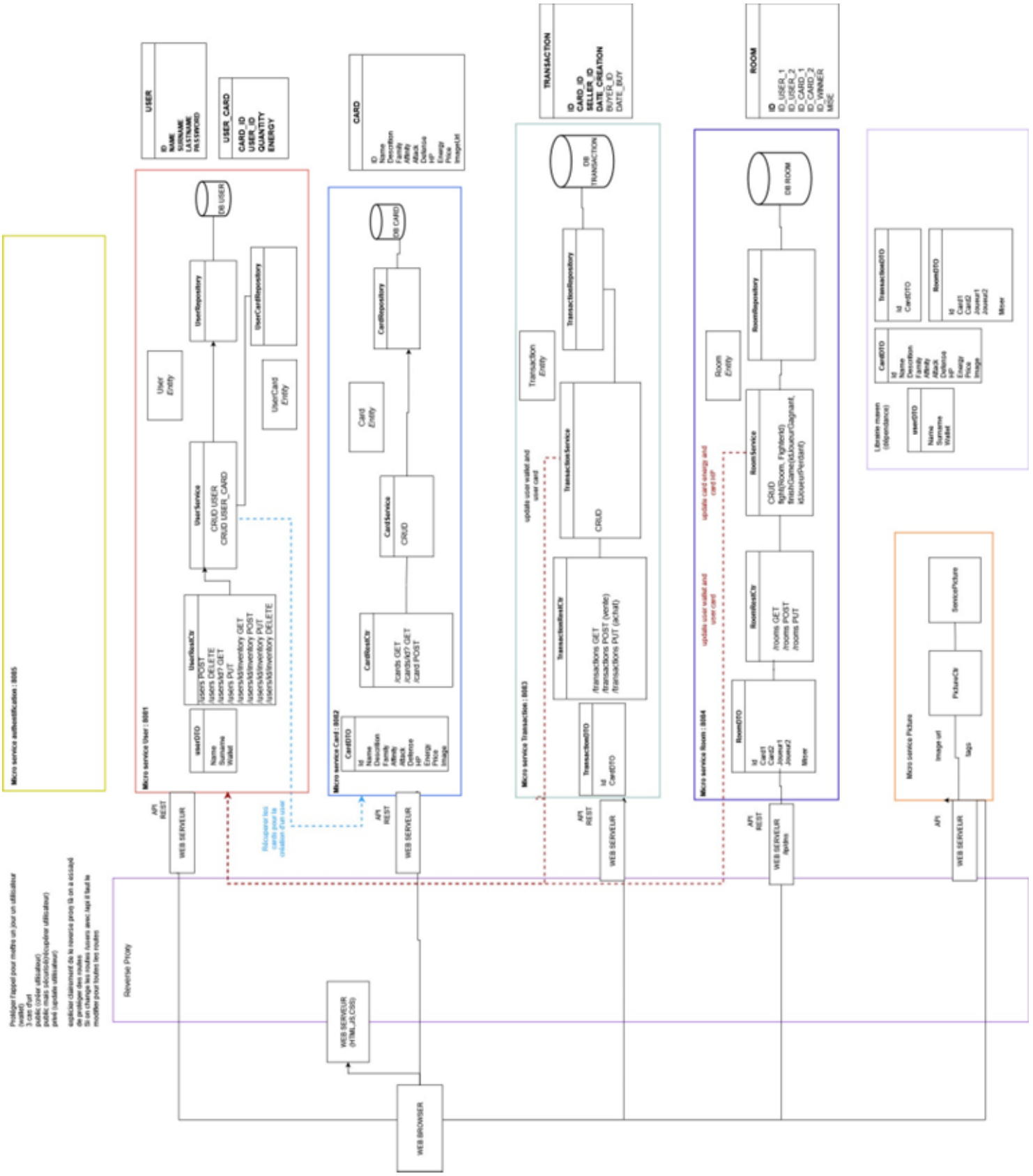
Schéma d’architecture



Avantages et différences SOA / Microservice

	SOA	**Microservice**
Avantages	Recyclage des composants et réutilisation des services. Facilite la maintenance de l'application. Partage des ressources et données entre les services. Code cohérent car on peut utiliser qu'un seul langage. L'application est plus fiable, les services peuvent être testés indépendamment.	Faible couplage entre les différents services qui sont indépendants. La logique métier est mieux découpée. Déploiement du code facilité (Conteneurisation Docker). Aisément scalable (dédoublement des services + load balancing). Les équipes de dev peuvent travailler plus facilement sur chaque service. Les services peuvent être développés dans différents langages. Couple très faible entre les microservices

Différences	SOA	Microservice
Langages	Langage de développement unique.	Plusieurs langages de développement sont possibles.
Communication avec les services	Utilisation des services en mémoire avec des appels de fonctions.	Utilisation des services à l'aide d'appels réseaux.
Persistence des données	Centralisation sur un unique SGBD + Transactionnel + optimisation des requêtes.	Perte du mode transactionnel + communication avec les services lors de requête complexe (surcharge du réseau)
Ouverture aux évolutions	Obligation de mettre à niveau tous les services même les moins utilisés.	Les services sont scalables indépendamment on peut choisir d'améliorer seulement celui très utilisé.

Couverture de tests

La couverture de test avec Sonar n'a pas pu être effectuée faute de temps. Cependant, nous avons mis en place des tests unitaires dans le service UserMicroservice.

A l'aide de JUnit et Mockito, nous avons testé les fonctions de la classe UserServiceTest.

```java
package com.sp.unittest;

import static org.junit.Assert.assertEquals;

@RunWith(MockitoJUnitRunner.class)
public class UserServiceTest {

    @Mock
    UsersService userServiceMock;

    @Mock
    UserRepository userRepository;


    @Test
    public void testCreateUserWithWallet5000() {
        String password = "tata";
        Users user = new Users("toto", "titi", password);
        userServiceMock.addUser(user);

        //par défaut wallet de 5000
        assertEquals(5000, user.getWallet());
    }
```

Par soucis de temps, nous n’avons implémenté et testé les tests unitaires que dans le service UserMicroservice. Cependant, si nous devions élargir nos tests unitaires, nous repartirions sur la même logique de tests pour les autres services.