La Couche Service

November 15, 2024

1 Introduction

La **couche service** dans une application multicouche fournit la logique métier qui agit comme un intermédiaire entre la couche de persistance (où les données sont stockées et récupérées) et la couche contrôleur (qui gère les requêtes des utilisateurs). Elle garantit que les règles et les opérations métiers de l'application sont implémentées de manière structurée et réutilisable.

Dans le framework Spring, des annotations telles que `@Service`, `@Autowired` et `@Transactional` sont couramment utilisées pour définir et gérer les composants de service et leurs interactions. Ces annotations simplifient la configuration et la gestion des services.

2 Rôle de la Couche Service

La couche service dans une application basée sur Spring remplit les rôles clés suivants :

- **Logique Métier** : Elle contient la logique métier principale de l'application, garantissant que les règles métier sont respectées lors du traitement des requêtes des utilisateurs.
- **Gestion des Transactions** : Elle gère les limites des transactions, garantissant que plusieurs opérations sur la base de données sont traitées comme une seule unité de travail, préservant ainsi l'intégrité des données.
- **Interaction avec les Autres Couches** : Elle agit comme un médiateur entre la couche de persistance (DAO) et la couche contrôleur, garantissant que l'accès aux données est effectué efficacement et correctement.

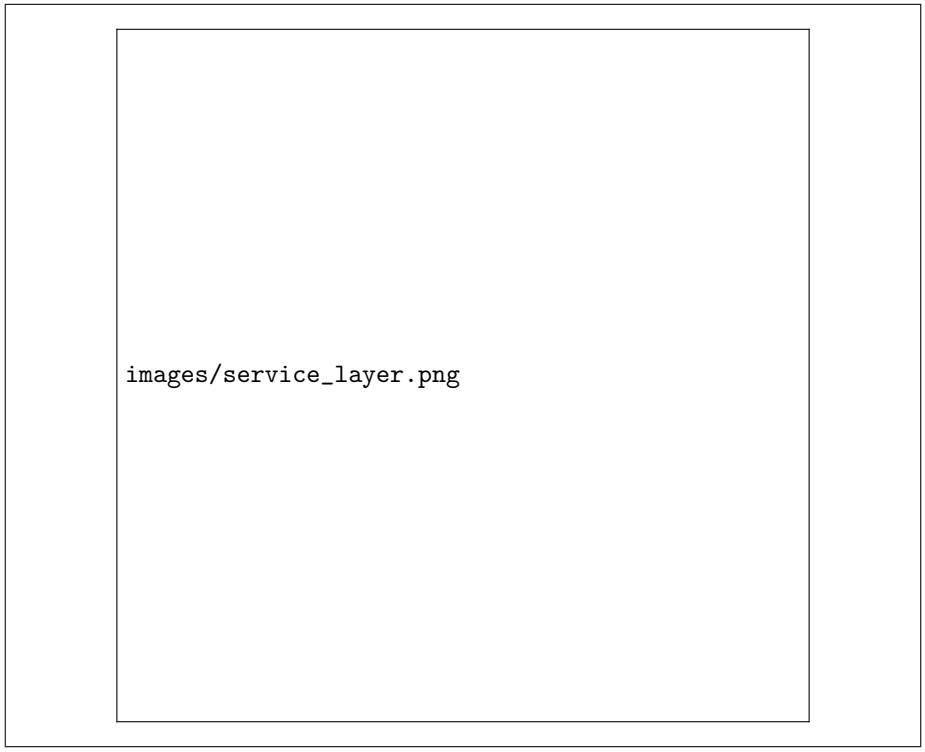

Figure 1: Architecture de la Couche Service

3 Annotations Clés dans la Couche Service

Spring fournit plusieurs annotations pour simplifier la définition et la gestion des composants de la couche service. Les annotations les plus utilisées sont :

- `@Service` : Marque la classe comme un composant de service dans le conteneur Spring, indiquant qu'elle contient la logique métier.
- `@Autowired` : Injecte automatiquement les dépendances (comme les DAO) dans la classe de service.
- `@Transactional` : Définit la portée d'une seule transaction pour une méthode ou une classe. Cela garantit que les opérations sont terminées avec succès ou annulées en cas d'échec.
- `@Component` : Une annotation générique qui peut également être utilisée pour marquer les classes de la couche service, bien que `@Service` soit préféré pour les composants de logique métier.

3.1 Exemple de Code avec Annotations

Voici un exemple de la manière dont la couche service est structurée en utilisant ces annotations dans une application basée sur Spring.

```
@Service
public class ProductService {

    @Autowired
    private ProductRepository productRepository;  // Inject  avec @Autowired

    @Transactional
    public void createProduct(Product product) {
        // Logique m tier : validation, manipulation, etc.
        if (product != null) {
            productRepository.save(product);  // Interaction avec la couche de persistance
        } else {
            throw new IllegalArgumentException("Le produit ne peut pas  tre  nul");
        }
    }

    @Transactional(readOnly = true)
    public Product getProductById(Long id) {
        return productRepository.findById(id).orElse(null); // R cup ration des donn es
    }

    public void updateProduct(Long id, Product product) {
        if (productRepository.existsById(id)) {
            productRepository.save(product);
        } else {
            throw new EntityNotFoundException("Produit non trouv ");
        }
    }

    @Transactional
    public void deleteProduct(Long id) {
        productRepository.deleteById(id);  // Supprime le produit de la base de
```

```
                donn es
33      }
34  }
```

Listing 1: Exemple de Couche Service avec Annotations

Dans ce code :

- `@Service` marque la classe `ProductService` comme un bean de service géré par Spring.
- `@Autowired` est utilisé pour injecter une instance de `ProductRepository` dans la classe de service.
- `@Transactional` garantit que les méthodes `createProduct`, `updateProduct` et `deleteProduct` sont exécutées dans un contexte transactionnel. L'attribut `readOnly` dans `@Transactional(readOnly = true)` marque la méthode `getProductById` comme une opération en lecture seule, optimisant ainsi les performances.

4 Interaction entre la Couche Service et la Couche de Persistance

La couche service interagit directement avec la couche de persistance pour accéder et modifier les données stockées dans la base de données. La couche de persistance fournit un accès aux données via des repositories ou des DAO (Data Access Objects), et la couche service les invoque pour effectuer des opérations CRUD (Créer, Lire, Mettre à jour, Supprimer).

Dans le framework Spring, l'annotation `@Autowired` est couramment utilisée pour injecter des DAO ou des repositories dans la classe de service. De plus, la gestion des transactions est généralement gérée au niveau de la couche service à l'aide de `@Transactional`, garantissant que toute la transaction est validée ou annulée en cas de besoin.

Figure 2: Interaction de la Couche Service avec la Couche de Persistance

5 Gestion des Exceptions dans la Couche Service

La gestion des exceptions est un autre aspect crucial de la couche service. Dans le contexte de la couche service, les exceptions sont souvent utilisées pour communiquer les erreurs qui surviennent lors de l'exécution de la logique métier ou des interactions avec la couche de persistance. Une pratique courante consiste à encapsuler les exceptions de bas niveau dans une classe d'exception personnalisée, puis à les propager vers la couche contrôleur, où elles peuvent être gérées et traduites en réponses HTTP appropriées.

```
1  @Service
2  public class ProductService {
3
4      @Autowired
5      private ProductRepository productRepository;
6
7      @Transactional
```

```
8       public void createProduct(Product product) {
9           try {
10              if (product != null) {
11                  productRepository.save(product);
12              } else {
13                  throw new InvalidProductException("
                        Le produit ne peut pas  tre  nul"
                        );
14              }
15          } catch (Exception e) {
16              throw new ProductServiceException("
                    Erreur lors de la cr ation du
                    produit", e);
17          }
18      }
19  }
```

Listing 2: Gestion Personnalisée des Exceptions dans la Couche Service

Dans cet exemple :

- `InvalidProductException` est une exception personnalisée levée si l'entrée est invalide.
- `ProductServiceException` est une exception personnalisée qui encapsule les exceptions de bas niveau, fournissant plus de contexte avant de les relancer.

6 Conclusion

La couche service joue un rôle essentiel dans la gestion de la logique métier et des interactions entre les couches de persistance et contrôleur. En utilisant les annotations Spring comme `@Service`, `@Autowired` et `@Transactional`, nous pouvons facilement définir et gérer les composants de service, garantissant que la logique métier est exécutée correctement et efficacement. De plus, la gestion des exceptions au sein de la couche service améliore la gestion des erreurs et l'expérience utilisateur en fournissant des messages d'erreur significatifs aux contrôleurs.