Architecture Logiciel

Méthode appropriée :

Découpage par zone de fonctionnalités =>

- Payment
- Catalogue
- Livraison
- Facturation

———————————————————

Exemple schématisé :

Catalogue : - Image,Titre,Auteur, rang/qualité,format (ebook ou papier),category

Recommandation : - Listing des objets souvent achetés ensemble en fonction de l'achat en question

Livraison : - Dimensions, poids, type transport (international/ local),restrictions règlementaire du au type de contenant à livrer.

Shopping cart : - Prix ,remise etc..

Avis Clients : - Liste (rang de l'article ,avis , réputation)

Barre de recherche livre : - Par Titre, isbn (code barre), auteur

———————————————————

Architecture 1 :

p1.myShop
    dao
        BookDAO
        DvdDAO
    entities
        Book
        Dvd

```
        HasWeight (b)
        Item
        Itemid
        Quantity
        Recipient
        User
    services
        catalog
            CatalogService (c)
        shipping (a)
            DeliveryService  *(réf:  1)
        shoppingcart
            Cart (d)
            Cartitem (e)
            ShoppingCarService
```

Exemple type : Couplage élevé et cohésion faible

Réf:1 Calcul de poids

```
package p1.myshop.services.shipping; => (a)

import p1.myshop;entities.HasWeight; => (b)
import p1.myshop.services.catalog.CatalogServices; => (c)
import p1.myshop.services.shoppingcart.Cart; =>  (d)
import p1.myshop.services.shoppingcart.CartItem; => (e)

*(réf:  1)
 => public class DeliveryService {
    private final CatalogService catalogService;

    public DeliveryService(CatalogService catalogService){
        this.catalogService = catalogService;
    }

    public double calculateOrderWeight(Cart cart) {
        return cart.items().stream()
            .map(CartItem::itemId)
            .map(catalogService::loadItem)
```

```
                .mapToDouble(HasWeight::weight)
                .sum();

    }
}
```

Architecture 2 :

Séparée en 3 MicroServices

```
p2.myshop
    core
 – - - - - - - - - - - - - - -
        catalog
            Book
            CatalogService
            Dvd
            HasWeight
            Item
            Itermid
 – - - - - - - - - - - - - - -
        shipping (a)
            DeliveryRequest *(réf:  3)
            DeliveryService *(réf:  2)
            Recipient
            RecipientService
            Shippable
 – - - - - - - - - - - - - - -
        shoppingcart
            Cart
            Cartitem
            Cartitemid
            Label
            Price
            Quantity
            ShoppingCartService
```

Exemple type : Couplage(dépendance) faible grande cohésion sur toute la couche concernée

---

```
package p2.myshop.core.shipping; => (a)

*(réf:  2)
=> public class DeliveryService {
	public double calculateOrderWeight (
				DeliveryRequest deliveryRequest) {
		return deliveryRequest.weight();
	}

}
```

———————————————————

```
*(réf:  3)

=> public class DeliveryRequest {
		private final List<Shippable> shippables;
		private final Recipient recipient;

		public DeliveryRequest(List<Shippable> shippables,
					Recipient recipient) {
			this.shippables = shippables;
			this.recipient = recipient;

	}

public double weight(){
	return shippables
		.stream()
		.mapToDouble(Shippable:: weight)
		.sum();

	}

}
```

———————————————————

=> Mise en place d’un Controller (Mapping)

```
@RestController
@RequestMapping( '' /delivery '')
public class DeliveryController {

	private final p2.myshop.core.shoppingcart.ShoppingCartService
```

```
shoppingCartService;
	private final p2.myshop.core.shipping.DeliveryService
deliveryService;
	private final p2.myshop.core.catalog.CatalogService catalogService;
	private final p2.myshop.core.shipping.RecipientService
recipientService;

	@Autowired
	public DeliveryController(...){...}

	@RequestMapping(method = RequetsMethod.POST)
	public void deliver(@RequestBody DeliverCommand command){
		Cart cart = shoppingCartService.loadCart(command.cartId);
		Recipient recipient =
recipientService.loadRecipient(command.recipientId);
		List<Shippable> shippables = toShippables(cart);

		deliveryService.process(new DeliveryRequest(shippables ,
recipient));
	}
|———
|	private List<Shippable> toShippables ( Cart cart){
|		return cart.items().stream().map(this::
toShippable).collect(toList());
|
|	}
|
|	private Shippable toShippable(CartItem cartItem){
|		Item item =
catalogService.loadItem(ItemId.from(cartItem.id().asString()));
|		return new Shippable(cartItem.label().asString().item.weight());
|
|
|	}
|———-
}
```

En Résumé

Concurrence entre les Techniques
Différentes corrélations entre les contextes

Découpage pour chaque zone métier et non pas par techno => En l'occurrence Microservices

Hautes capacités de données vs Basses capacités de données : NoSQL / DB
Lecture vs Ecriture : View and Request => double/BigDecimal/ approximate/exact
Insider vs Private : Vault / DB
Volatile vs Static (Fast Consistecy vs Slow Consistency): Streaming /CDN)

!! A Connaitre !! :

- Bounded Contexts => La capacité à délimiter ET interpréter les différents services
- Contracts
- Data Governance

Choisir les designs patterns(architecture) en fonction de l'environnement , en réponse a un usage particulier ou à un enjeu particulier.
Garder en tête le potentiel et la scalabilité du logiciel.