Couche Service et architecture DTO (Data Transfer Object)

22. L'Architecture en 3 couches et la Couche de Service (Service Layer)

Jusqu'à présent, nous avons vu les **Contrôleurs** (qui reçoivent les requêtes HTTP) et les **Repositories** (qui parlent à la base de données). Cependant, dans une application professionnelle respectant les principes du *Clean Code*, il manque une pièce maîtresse : la **Couche de Service**.

L'erreur du débutant : Placer la logique métier (Business Logic) directement dans le Contrôleur. Si vous faites cela, votre contrôleur devient massif, difficile à tester, et vous ne pouvez pas réutiliser cette logique ailleurs.

La solution (L'architecture 3-Tiers) :

1. **Controller Layer (Présentation) :** Ne fait QUE recevoir la requête HTTP, valider les paramètres, appeler le Service, et renvoyer la réponse HTTP.
2. **Service Layer (Logique Métier) :** Contient le "cerveau" de l'application. C'est ici que l'on effectue les calculs, les déductions, et l'orchestration des données. Le Service ne sait pas qu'il est appelé par le Web (HTTP).
3. **Data Access Layer (Repository) :** Ne fait QUE lire et écrire dans la base de données.

23. Le Pattern DTO (Data Transfer Object)

L'autre règle d'or du développement backend est de **ne jamais exposer vos Entités (Base de données) directement au client (API)**.

Pourquoi utiliser des DTOs ?
• **Sécurité :** Votre Entité `Suspect` peut contenir des champs sensibles (ex: mot de passe, champs d'audit internes) que l'application cliente (Flutter ou Angular) ne doit jamais voir.
• **Découplage :** Si vous renommez une colonne dans votre base de données, l'Entité change, mais le DTO (le contrat de l'API) reste intact, ce qui évite de casser l'application cliente.
• **Formatage spécifique :** Le client a parfois besoin de données combinées provenant de plusieurs tables différentes en une seule réponse.

apcpedagogie.com

24. Cas pratique : Logique de déduction pour l'investigation

Mettons cela en pratique. Nous allons créer un système pour analyser un indice récolté sur une scène de crime et débloquer progressivement les étapes de l'affaire "*The Vanishing Diplomat*".

Étape 1 : Créer les DTOs (Ce qui entre et sort de l'API)

```
package com.studiogames.mysterybackend.dto;

// DTO pour la requête entrante (Ce que le client Flutter envoie)
public class ClueAnalysisRequestDTO {
    private String clueId;
    private String currentStageId;

    // Getters et Setters...
    public String getClueId() { return clueId; }
    public void setClueId(String clueId) { this.clueId = clueId; }

    public String getCurrentStageId() { return currentStageId; }
    public void setCurrentStageId(String currentStageId) { this.currentStageId =
currentStageId; }
}
```

```
package com.studiogames.mysterybackend.dto;

// DTO pour la réponse sortante (Ce que le serveur renvoie au client)
public class DeductionResultDTO {
    private boolean isDeductionCorrect;
    private String narrativeOutcome; // Texte narratif affiché sur la machine à écrire
    private String nextStageUnlocked; // ID du prochain niveau

    public DeductionResultDTO(boolean isDeductionCorrect, String narrativeOutcome, String
nextStageUnlocked) {
        this.isDeductionCorrect = isDeductionCorrect;
        this.narrativeOutcome = narrativeOutcome;
        this.nextStageUnlocked = nextStageUnlocked;
    }

    // Getters...
}
```

Étape 2 : Créer le Service (Le cerveau de l'application)

Le service est annoté avec `@Service`. Il contient toute la logique de jeu (Business Logic). Il est indépendant du protocole HTTP.

```
package com.studiogames.mysterybackend.service;
```

```
import com.studiogames.mysterybackend.dto.ClueAnalysisRequestDTO;
import com.studiogames.mysterybackend.dto.DeductionResultDTO;
import org.springframework.stereotype.Service;

@Service
public class InvestigationService {

    // Ici, vous pourriez injecter un Repository pour lire/sauvegarder l'état de la partie
    // private final CaseRepository caseRepository;

    /**
     * Méthode contenant la logique métier (Business Logic) pure.
     */
    public DeductionResultDTO processDeduction(ClueAnalysisRequestDTO request) {

        // Logique complexe de déduction : vérifier si l'indice correspond à l'étape
actuelle
        if ("CLUE_001_DIPLOMAT_WATCH".equals(request.getClueId()) &&
            "STAGE_1_OFFICE".equals(request.getCurrentStageId())) {

            // Le joueur a trouvé la bonne combinaison logique !
            // On sauvegarde l'avancement en base de données (simulé ici)...

            return new DeductionResultDTO(
                true,
                "L'heure arrêtée sur la montre correspond au moment exact où la
communication a été coupée. La piste nous mène au port.",
                "STAGE_2_HARBOR"
            );
        }

        // Mauvaise déduction
        return new DeductionResultDTO(
            false,
            "Cette pièce à conviction ne semble pas liée à cette scène. Reliez les fils sur
votre tableau.",
            null
        );
    }
}
```

Étape 3 : Garder le Contrôleur propre (Clean Controller)

Le contrôleur devient incroyablement minimaliste et facile à lire. Son seul but est de déléguer le travail au Service et de renvoyer le résultat DTO.

```
package com.studiogames.mysterybackend.controller;

import com.studiogames.mysterybackend.dto.ClueAnalysisRequestDTO;
import com.studiogames.mysterybackend.dto.DeductionResultDTO;
import com.studiogames.mysterybackend.service.InvestigationService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/investigation")
public class InvestigationController {
```

```
    private final InvestigationService investigationService;

    // Injection du service via le constructeur
    public InvestigationController(InvestigationService investigationService) {
        this.investigationService = investigationService;
    }

    @PostMapping("/analyze")
    public ResponseEntity<DeductionResultDTO> analyzeClue(@RequestBody
ClueAnalysisRequestDTO request) {

        // Le contrôleur ne prend AUCUNE décision logique.
        // Il passe les DTOs au service et renvoie la réponse.
        DeductionResultDTO result = investigationService.processDeduction(request);

        return ResponseEntity.ok(result);
    }
}
```

Le résultat d'un code bien structuré :

En séparant votre code de cette manière, si vous décidez demain de créer une commande dans le terminal (CLI) ou un événement programmé (Cron Job) pour analyser des indices, vous pourrez réutiliser `InvestigationService` tel quel, sans aucune modification, car il n'est pas lié à l'API HTTP !

*Rédigé et structuré par : **Med Khalil Kribi***

Source : https://apcpedagogie.com/couche-service-et-architecture-dto-spring · téléchargé le 07/11/2026 · © APC Pédagogie · apcpedagogie.com