

TP 2/2 – De la journalisation en base à la journalisation par fichiers

Objectifs du TP :

Dans ce second TP, vous allez faire évoluer l'application de **To Do List** développée lors du **TP 1**.

Etat de l'application dans sa version actuelle :

- Enregistre les actions d'audit dans la base (**AUDIT_LOG**) ;
- Permet à un administrateur de consulter les logs via **/admin**.

Dans ce TP, vous allez :

1. **Identifier les limites** de la journalisation en base.
2. Découvrir les **bonnes pratiques** de journalisation par fichiers.
3. Mettre en œuvre une journalisation par fichiers via **SLF4J + Logback**.
4. Séparer :
 - Les **logs techniques** (**app.log**),
 - Les **logs d'audit** (**audit.log**).
5. Compléter l'application pour écrire simultanément dans les fichiers de logs.

Log d'audit vs Log technique :

Dans une application, toutes les écritures dans les journaux (logs) n'ont pas la même finalité. On distingue principalement **les logs techniques** et **les logs d'audit**.

Log technique :

- Produit par les développeurs et les composants internes (*Spring, Hibernate, etc.*).
- Sert à **comprendre le fonctionnement technique** de l'application.
- Utilisé pour : diagnostiquer un bug; analyser une erreur; suivre un flux d'exécution.
- Contenu typique :
 - message d'erreur; stacktrace; événement système; information d'exécution.

Exemple :

- `INFO TaskController - Mise à jour de la tâche 42`

Finalité : débogage, surveillance, analyse technique.

Portée : interne à l'équipe technique.

Log d'audit :

- Produit pour **tracer les actions des utilisateurs**.
- Sert à garantir la **traçabilité**, la **responsabilité** et la **sécurité**.
- Utilisé pour :
 - Prouver qu'un utilisateur a réalisé une action,
 - Détecter un usage frauduleux,
 - Répondre à des obligations légales (RGPD, conformité).
- Contenu typique :
 - *Utilisateur; action; date/heure; IP; ressource concernée.*

Exemple :

- `LOGIN user=jdupont ip=192.168.1.15`

Finalité : sécurité, conformité, analyse post-incident.

Portée : accessibilité restreinte (administrateurs habilités).

À retenir :

- **Log technique** = comprendre comment l'application fonctionne.
- **Log d'audit** = comprendre ce que les utilisateurs ont fait.

1. Contexte — Limites de la journalisation en base de données

La journalisation via table **SQL** (comme **AUDIT_LOG**) présente plusieurs limites :

Charge sur le SGBD :

Chaque action (login, création, suppression...) déclenche un **INSERT**, ce qui augmente la charge sur le **SGBD**.

Risque de perte de logs :

En cas de rollback transactionnel, certains logs ne sont jamais écrits.

Risque de compromission :

Si un attaquant accède à la base :

- Il peut supprimer des logs;
- Ou les falsifier sans laisser de trace.

Problèmes d'archivage / performances :

Une table de logs peut :

- Atteindre des millions de lignes,
- Ralentir certaines opérations **SQL**,
- Compliquer les sauvegardes.

2. Journalisation par fichiers — Principe général

Créer des logs dans des fichiers texte présente plusieurs avantages :

- Indépendant des transactions **SQL**;
- Difficile à falsifier (*append-only*);
- Rotation automatique (un fichier par jour, par taille...),
- Standard pour l'analyse (*ELK*, *Splunk*, *Loki*...),
- Plus performant pour de gros volumes.

Les solutions de centralisation de logs (*ELK*, *Splunk*, *Loki*...) :

Dans les environnements professionnels, les logs ne sont souvent pas stockés seulement dans des fichiers locaux. On utilise des **solutions de centralisation et d'analyse de logs** comme **ELK**, **Splunk** ou **Loki**. Leur rôle est de collecter, stocker, indexer et visualiser de grandes quantités de logs qui sont dispersés sur plusieurs serveurs.

Fonctionnalités principales :

1. Centralisation :

- Récupèrent les logs depuis plusieurs serveurs/services (*API*, *microservices*, *bases*, *firewalls*...)
- Unifient les formats pour une analyse plus simple.

2. Indexation et recherche avancée :

- Recherche plein-texte ou requêtes structurées.
- Filtrage par service, utilisateur, date, action, niveau (**INFO**, **ERROR**, etc.).
- Très utile pour diagnostiquer un incident complexe.

3. Visualisation :

- Tableaux de bord (dashboards) personnalisables.
- Graphiques, timelines, heatmaps, cartes d'erreurs.
- Vue temps réel des événements critiques.

4. Alertes et supervision :

- Déclenchent une alerte (mail, SMS, webhook...) en cas :
 - De pic d'erreurs;
 - De tentatives de connexions anormales;
 - De volume anormal de logs.

5. Scalabilité :

- Gèrent des **millions de logs par jour** grâce à des architectures distribuées (cluster).

Exemples de solutions :

■ ELK (Elasticsearch + Logstash + Kibana)

- Open source, très utilisé.
- Collecte → transformation → stockage → visualisation.

■ Loki (Grafana Labs)

- Optimisé pour les environnements cloud / Kubernetes.
- Très léger, conçu pour fonctionner avec Grafana.

■ Splunk

- Leader du marché, très complet.
- Offre des capacités avancées (machine learning, détection d'anomalies).

À retenir :

- Ces outils **remplacent ou complètent** les journaux locaux.
- Ils permettent d'**analyser plus vite**, de **corrélérer les logs**, et de **mieux sécuriser** les systèmes.
- Indispensables dans les architectures modernes (microservices, SI complexes).

2.1 Format standard d'une ligne de log :

Exemple :

- 2025-02-15T10:23:45.123 INFO TaskController [user=jdupont, ip=192.168.1.10]
Création de la tâche "Acheter du café"

Un bon log contient :

- date / heure (ISO 8601)
- niveau (INFO, ERROR, ...)
- nom du logger (classe)
- message (souvent paramétré)

3. Étape 1 — Créer la configuration Logback

3.1 Création de `logback-spring.xml` :

Logback est le système de journalisation utilisé par défaut dans **Spring Boot**.

Il fonctionne avec **SLF4J**, l'interface standard de logging en **Java/Kotlin**.

Quand vous écrivez dans le code :

```
private val logger = LoggerFactory.getLogger(MyClass::class.java) // Création d'un
logger
logger.info("Message important") // Ajout d'un message INFO dans le logger
```

C'est **Logback** qui décide :

- Comment ce message sera formaté,
- Dans quelle sortie il sera écrit (console, fichier, plusieurs fichiers...),
- S'il doit être affiché ou ignoré selon son niveau (*INFO*, *DEBUG*, *ERROR*...).

Le fichier `logback-spring.xml` : la "carte d'instructions" de Logback

Spring Boot charge automatiquement un fichier nommé `logback-spring.xml`.

Ce fichier sert à **configurer le comportement complet du logging**, notamment :

1. Les **appenders**

Un *appender* définit **où** les logs doivent être écrits :

- La console (**CONSOLE**),
- Un fichier (**FILE**),
- Plusieurs fichiers (ex. `app.log`, `audit.log`),
- Un service distant (*ELK*, *Splunk*...).

Exemple :

```
<appender name="FILE" class="ch.qos.logback.core.FileAppender">
  <file>logs/app.log</file>
</appender>
```

2. Les **patterns**

Ils définissent **le format d'une ligne de log** :

```
<pattern>%d{ISO8601} %-5level %logger{36} - %msg%n</pattern>
```

Ce qui affiche quelque chose comme :

```
2025-01-15T14:25:15.443 INFO TaskController - Création de la tâche Acheter café
```

3. Les **loggers**

Ils définissent :

- Quelles classes produisent quel type de logs,
- Dans quels fichiers elles écrivent,
- Et avec quel niveau minimum (**INFO**, **DEBUG**, ...).

Exemple d'un logger spécialisé pour l'audit :

```
<logger name="AUDIT" level="INFO" additivity="false">  
  <appender-ref ref="AUDIT_FILE"/>  
</logger>
```

4. Le **root logger**

C'est le logger par défaut :

S'il n'y a pas de règle spécifique, c'est lui qui s'applique.

```
<root level="INFO">  
  <appender-ref ref="CONSOLE"/>  
  <appender-ref ref="FILE"/>  
</root>
```

Pourquoi utiliser **logback-spring.xml** dans Spring Boot ?

- Pour **séparer** différents types de logs (techniques, audit, erreurs critiques...).
- Pour **configurer une rotation automatique** des fichiers (un fichier par jour, par taille...).
- Pour **éviter que les fichiers ne grossissent à l'infini**.
- Pour **formater les logs proprement** (date, user, action...).
- Pour **rediriger certains logs vers des destinations spécifiques**.

En résumé :

- Le code décide *quoi* journaliser (messages).
- **logback-spring.xml** décide *où* et *comment* les logs sont écrits.

A FAIRE :

Dans **src/main/resources**, créer :

- **logback-spring.xml**

👉 **Contenu à implémenter à la PAGE SUIVANTE :**

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <!-- 1. Appender console (dev) -->
  <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{ISO8601} %-5level [%thread] %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>

  <!-- 2. Appender fichier pour logs techniques -->
  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>logs/app.log</file>

    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>logs/app.%d{yyyy-MM-dd}.log</fileNamePattern>
      <maxHistory>14</maxHistory>
    </rollingPolicy>

    <encoder>
      <pattern>%d{ISO8601} %-5level %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>

  <!-- 3. Appender fichier dédié à l'audit -->
  <appender name="AUDIT_FILE"
class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>logs/audit.log</file>

    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>logs/audit.%d{yyyy-MM-dd}.log</fileNamePattern>
      <maxHistory>30</maxHistory>
    </rollingPolicy>

    <encoder>
      <pattern>%d{ISO8601} %-5level %msg%n</pattern>
    </encoder>
  </appender>

  <!-- Logger dédié à l'audit -->
  <logger name="AUDIT" level="INFO" additivity="false">
    <appender-ref ref="AUDIT_FILE"/>
  </logger>

  <!-- Logger root -->
  <root level="INFO">
    <appender-ref ref="CONSOLE"/>
    <appender-ref ref="FILE"/>
  </root>
</configuration>

```

4. Étape 2 — Ajouter des logs techniques dans les contrôleurs

Dans cette partie nous allons utiliser le **LoggerFactory** pour créer nos logger.

4.1 Actions à réaliser dans **TaskController** :

Ajouter les imports nécessaires :

```
// AJOUT TP2 - import du logger SLF4J pour les logs techniques
import org.slf4j.LoggerFactory
```

Ajouter l'attribut **Logger** dans toutes la classe du contrôleur

```
@Controller
@RequestMapping("/tasks")
class TaskController(
    private val taskService: TaskService,
    private val userService: UserService,
    private val auditLogService: AuditLogService
) {

    // AJOUT TP2 - logger technique pour cette classe
    private val logger = LoggerFactory.getLogger(TaskController::class.java)

    @GetMapping
    // ...

}
```

Puis, on ajoute les enregistrements des événements dans le logger en ajoutant les lignes suivantes dans les méthodes existantes :

Log pour l'affichage de la liste des tâches :

```
// ...
@GetMapping
fun listTasks(authentication: Authentication, model: Model): String {
    val user = userService.findByUsername(authentication.name)!!
    val tasks = taskService.getUserTasks(user)

    // AJOUT TP2 - log technique lors de l'affichage de la liste des tâches
    logger.info("Affichage de la liste des tâches pour l'utilisateur {}",
user.username)

    model.addAttribute("tasks", tasks)
    model.addAttribute("username", user.username)
    return "tasks"
}
// ...
```

Log pour la création des tâches :

```
// ...
@PostMapping("/create")
fun createTask(
    @RequestParam title: String,
    @RequestParam(required = false) description: String?,
    @RequestParam(required = false) dueDate: String?,
    authentication: Authentication,
    request: HttpServletRequest
): String {
    val user = userService.findByUsername(authentication.name)!!

    val parsedDueDate = dueDate?.takeIf { it.isNotBlank() }?.let {
        LocalDateTime.parse(it, DateTimeFormatter.ISO_LOCAL_DATE_TIME)
    }

    taskService.createTask(title, description, parsedDueDate, user)

    // (journalisation d'audit déjà mise en place au TP1)
    auditLogService.log(
        username = user.username,
        action = "CREATE_TASK",
        details = "Création de la tâche : $title",
        request = request
    )

    // AJOUT TP2 - log technique lors de la création d'une tâche
    logger.info(
        "Création d'une tâche pour l'utilisateur {} : titre=\"{}\", échéance={}",
        user.username,
        title,
        parsedDueDate
    )
    // ...
```

Log pour la mise à jour des tâches :

```
// ...
@PostMapping("/update/{id}")
fun updateTask(
    @PathVariable id: Long,
    @RequestParam title: String,
    @RequestParam(required = false) description: String?,
    @RequestParam status: String,
    @RequestParam(required = false) dueDate: String?,
    authentication: Authentication,
    request: HttpServletRequest
): String {
    val task = taskService.getTaskById(id) ?: return "redirect:/tasks"

    if (task.user.username != authentication.name) {
        // AJOUT TP2 - log technique si un utilisateur tente de modifier une tâche
        // qui ne lui appartient pas
        logger.warn(
            "Tentative de mise à jour non autorisée de la tâche {} par
l'utilisateur {}",
            id,
            authentication.name
        )
        return "redirect:/tasks"
    }

    val parsedDueDate = dueDate?.takeIf { it.isNotBlank() }?.let {
        LocalDateTime.parse(it, DateTimeFormatter.ISO_LOCAL_DATE_TIME)
    }

    taskService.updateTask(
        task,
        title,
        description,
        TaskStatus.valueOf(status),
        parsedDueDate
    )

    // (journalisation d'audit déjà mise en place au TP1)
    auditLogService.log(
        username = authentication.name,
        action = "UPDATE_TASK",
        details = "Modification tâche #$id (titre=$title, statut=$status)",
        request = request
    )

    // AJOUT TP2 - log technique lors de la mise à jour d'une tâche
    logger.info(
        "Mise à jour de la tâche {} par l'utilisateur {} : titre=\"{}\", statut=
{}, échéance={}",
        id,
```

```
        authentication.name,  
        title,  
        status,  
        parsedDueDate  
    )  
  
    return "redirect:/tasks"  
}  
// ...
```

Log pour la suppression des tâches :

```
// ...
@PostMapping("/delete/{id}")
fun deleteTask(
    @PathVariable id: Long,
    authentication: Authentication,
    request: HttpServletRequest
): String {
    val task = taskService.getTaskById(id)

    if (task != null && task.user.username == authentication.name) {
        taskService.deleteTask(id)

        // (journalisation d'audit déjà mise en place au TP1)
        auditLogService.log(
            username = authentication.name,
            action = "DELETE_TASK",
            details = "Suppression tâche #${id}",
            request = request
        )

        // AJOUT TP2 - log technique lors de la suppression d'une tâche
        logger.info(
            "Suppression de la tâche {} par l'utilisateur {}",
            id,
            authentication.name
        )
    } else {
        // AJOUT TP2 - log technique en cas de tentative de suppression non
        // autorisée
        logger.warn(
            "Tentative de suppression non autorisée de la tâche {} par
l'utilisateur {}",
            id,
            authentication.name
        )
    }

    return "redirect:/tasks"
}
// ...
```

5. Étape 3 — Ajouter un logger dédié à l'audit

Dans cette partie nous allons intervenir dans les classes `SecurityConfig` et `TaskController`.

Ajouts sur la classe `SecurityConfig` :

Ajout de l'import :

```
// AJOUT TP2 - Import du logger SLF4J
import org.slf4j.LoggerFactory
```

Ajout de l'attribut pour le logger d'audit :

```
// ...
class SecurityConfig(
    private val auditLogService: AuditLogService
) {

    // AJOUT TP2 - Logger dédié à l'audit (redirigé vers audit.log via logback-
    spring.xml)
    private val auditLogger = LoggerFactory.getLogger("AUDIT")
    // ...
```

Ajout de l'audit de connexion :

```
// ...
private fun customAuthenticationSuccessHandler() =
    AuthenticationSuccessHandler { request, response, authentication ->
        val username = authentication.name
        val ip = request.remoteAddr // Ajout du TP2

        // (audit en base - TP1)
        auditLogService.log(
            username = username,
            action = "LOGIN",
            details = "Connexion réussie",
            request = request
        )
        // AJOUT TP2 - écriture dans audit.log
        auditLogger.info("LOGIN user={} ip={}", username, ip)

        response.sendRedirect("/tasks")
    } // ...
```

Ajout de l'audit de déconnexion :

```
// ...
private fun customLogoutSuccessHandler() =
    LogoutSuccessHandler { request, response, authentication ->
        val username = authentication?.name ?: "anonymous"
        val ip = request.remoteAddr // Ajout du TP2

        // (audit en base - TP1)
        auditLogService.log(
            username = username,
            action = "LOGOUT",
            details = "Déconnexion",
            request = request
        )

        // AJOUT TP2 - écriture dans audit.log
        auditLogger.info("LOGOUT user={} ip={}", username, ip)

        response.sendRedirect("/login?logout")
    }
// ...
```

Ajouts sur la classe **TaskController** :

Ajout de l'attribut du logger d'audit :

```
// ....

// AJOUT TP2 - Logger technique
private val logger = LoggerFactory.getLogger(TaskController::class.java)

// AJOUT TP2 - Logger d'audit dédié (redirigé vers audit.log)
private val auditLogger = LoggerFactory.getLogger("AUDIT")

// ....
```

Ajout de l'audit pour la création de tâche :

```
// TP1 - journalisation en base
auditLogService.log(
    username = user.username,
    action = "CREATE_TASK",
    details = "Création de la tâche : $title",
    request = request
)

// AJOUT TP2 - journalisation fichier : audit.log
auditLogger.info(
    "CREATE_TASK user={} title=\"{}\" dueDate={}",
    user.username,
    title,
    parsedDueDate
)

// AJOUT TP2 - log technique lors de la création d'une tâche
logger.info(
    "Création d'une tâche pour l'utilisateur {} : titre=\"{}\", échéance=
    {}",
    user.username,
    title,
    parsedDueDate
)
```

Ajout de l'audit pour la modification de tâche :

```
// TP1 - audit en base
auditLogService.log(
    username = authentication.name,
    action = "UPDATE_TASK",
    details = "Modification tâche #${id} (titre=${title}, statut=${status})",
    request = request
)

// AJOUT TP2 - audit fichier
auditLogger.info(
    "UPDATE_TASK user={} taskId={} title=\"{}\" status={} dueDate={}",
    authentication.name,
    id,
    title,
    status,
    parsedDueDate
)

// AJOUT TP2 - log technique lors de la mise à jour d'une tâche
logger.info(
    "Mise à jour de la tâche {} par l'utilisateur {} : titre=\"{}\",
statut={}, échéance={}",
    id,
    authentication.name,
    title,
    status,
    parsedDueDate
)

return "redirect:/tasks"
```

Ajout de l'audit pour la suppression de tâche :

```
// AJOUT TP2 - audit fichier
auditLogger.info(
    "DELETE_TASK user={} taskId={}",
    authentication.name,
    id
)

// AJOUT TP2 - log technique lors de la suppression d'une tâche
logger.info(
    "Suppression de la tâche {} par l'utilisateur {}",
    id,
    authentication.name
)
} else {
    // AJOUT TP2 - log technique en cas de tentative de suppression non autorisée
    logger.warn(
        "Tentative de suppression non autorisée de la tâche {} par l'utilisateur {}",
        id,
        authentication.name
    )
    // AJOUT TP2 - audit d'une tentative de suppression non autorisée
    auditLogger.warn(
        "FORBID_DELETE_TASK user={} taskId={}",
        id,
        authentication.name
    )
}
```

6. Étape 4 — Tests manuels

Les fichiers de log sont enregistrés dans le dossier `/src/logs`

Vous obtiendrez la structure suivante :

```
mon-projet/  
├── src/  
├── build.gradle.kts  
├── logback-spring.xml  
├── logs/  
│   ├── app.log  
│   ├── audit.log  
│   ├── app.2025-02-12.log  
│   └── audit.2025-02-12.log  
└── ...
```

A FAIRE :

1. **Supprimez** le dossier `logs/` (s'il existe).
2. **Relancez l'application.**
3. Connectez-vous → consultez `logs/audit.log`.
4. Créez une tâche → consultez `logs/app.log` et `logs/audit.log`.
5. Modifiez, supprimez → vérifiez les logs.
6. Déconnectez-vous → vérifiez l'entrée `LOGOUT`.

Fin du TP 2

Vous disposez maintenant :

- D'une **journalisation par fichiers** complète;
- D'une séparation claire entre logs techniques et logs d'audit;
- D'une configuration Logback adaptée à un environnement professionnel.

Bravo pour le travail ! 🎉
