

TP 1/2 – Mise en place de la journalisation dans une application Web Spring Boot

Objectifs du TP :

Dans ce TP, vous allez découvrir et mettre en place la **journalisation d'audit** dans une application Web de gestion de tâches développée avec **Spring Boot** et **Kotlin**.

Vous partez d'une application *starter* opérationnelle qui fournit les fonctionnalités suivantes :

- La gestion des utilisateurs (création et connexion des utilisateurs),
- La sécurité Spring Security,
- La gestion des tâches (tâches de la *to do list*) (CRUD),
- Une page **/admin** prête à afficher des logs,
- **Aucun mécanisme de journalisation d'audit.**

Remarque : La gestion des utilisateurs n'est pas et ne sera pas implémentée dans l'interface de l'administrateur.

Votre mission sera d'implémenter vous-même :

- L'entité **AuditLog**,
- Le **DAO/dépôt AuditLogRepository**,
- Le service de gestion des logs **AuditLogService**,
- La journalisation contiendra les informations suivantes :
 - Liste des connexions / déconnexions des utilisateurs avec la date et l'IP,
 - Liste des actions sur la création / modification / suppression de tâches,
- L'affichage des logs dans la page **/admin**.

1. L'application starter

1.1. Fonctionnalités déjà existantes :

L'application starter que vous recevez est **fonctionnelle** et contient déjà :

- La connexion/création d'utilisateurs;
- La gestion des rôles (**USER** / **ADMIN**);
- La gestion de tâches (*créer, modifier, supprimer*);
- Une page **/admin** avec une vue **HTML** prête à afficher des logs (mais la liste est vide);
- Une base de données **H2** en mémoire;
- Les templates au format **Thymeleaf** pour les vues.

1.2. Comptes par défaut :

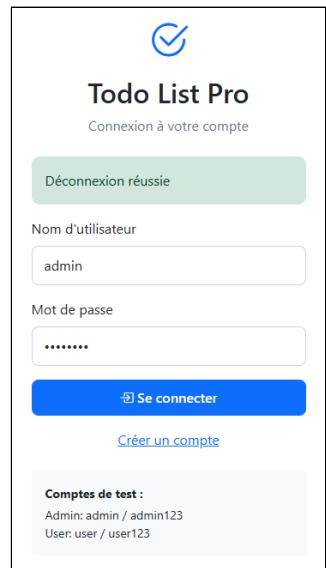
Deux comptes sont créés automatiquement au démarrage (via une classe d'initialisation) :

Identifiant	Mot de passe	Rôle
admin	admin123	ADMIN
user	user123	USER

A FAIRE :

- Cloner l'application starter : `git clone https://github.com/ljules/Starter-ToDoList-LOGS`
- Testez la connexion sur : <http://localhost:8080/login>
- Prenez en main l'application en la testant en prenant en vous connectant avec les 2 comptes par défaut (*admin* et *user*) :
 - Tester la création, modification de tâches
 - Observer le dashboard d'administration actuel du compte *admin*

Page de connexion au lancement de l'application :



Exemple de tâches créées :

TITRE	DESCRIPTION	STATUT	ÉCHÉANCE	ACTIONS
Une tâche en cours	Démonstration d'une tâche en cours...	En cours	01/12/2025 07:10	Modifier Supprimer
Une tâche à faire	Démonstration d'une tâche à faire.	À faire	02/12/2025 17:15	Modifier Supprimer
Tâche démo 1	Tâche initiale pour démonstration La tâche est modifiée.	Terminée	01/12/2025 06:15	Modifier Supprimer

Page d'administration dans l'état actuel :

Information RGPD : Ces logs sont conservés conformément aux exigences de traçabilité du RGPD. Ils enregistrent toutes les actions des utilisateurs pour assurer la sécurité et la transparence du traitement des données personnelles.

Page d'administration attendue pour la fin du TP :

DATE/HEURE	UTILISATEUR	ACTION	DÉTAILS	ADRESSE IP
30/11/2025 17:02:16	admin	LOGIN	Connexion réussie	0:0:0:0:0:0:1
30/11/2025 17:02:08	user	LOGOUT	Déconnexion	0:0:0:0:0:0:1
30/11/2025 17:01:55	user	UPDATE_TASK	Modification de la tâche: Tâche démo 1	0:0:0:0:0:0:1
30/11/2025 17:01:33	user	LOGIN	Connexion réussie	0:0:0:0:0:0:1
30/11/2025 17:00:57	user	LOGOUT	Déconnexion	0:0:0:0:0:0:1
30/11/2025 17:00:46	user		Modification de la tâche: Tâche démo 1	0:0:0:0:0:0:1
30/11/2025 17:00:13	user	CREATE_TASK	Création de la tâche: Tâche démo 1	0:0:0:0:0:0:1
30/11/2025 16:59:00	user	LOGIN	Connexion réussie	0:0:0:0:0:0:1
30/11/2025 16:58:46	admin	LOGOUT	Déconnexion	0:0:0:0:0:0:1
30/11/2025 16:57:38	admin	LOGIN	Connexion réussie	0:0:0:0:0:0:1

Information RGPD : Ces logs sont conservés conformément aux exigences de traçabilité du RGPD. Ils enregistrent toutes les actions des utilisateurs pour assurer la sécurité et la transparence du traitement des données personnelles.

2. Analyse rapide de la structure du projet

Le projet est organisé de la façon suivante (simplifiée) :

```
src/main/kotlin/org/ldv/AppStarter_ToDoList/
└── config
    ├── SecurityConfig.kt
    └── DataInitializer.kt
└── controller
    ├── AuthController.kt
    ├── TaskController.kt
    └── AdminController.kt    (liste des logs → vide pour l'instant)
└── entity
    ├── User.kt
    └── Task.kt
└── repository
    ├── UserRepository.kt
    └── TaskRepository.kt
└── service
    ├── UserService.kt
    └── TaskService.kt
└── AppStarterToDoListApplication.kt
```

💡 Important

Les éléments liés à la journalisation ([AuditLog](#), [AuditLogRepository](#), [AuditLogService](#), appels de logs, etc.) ont été volontairement retirés de cette version starter.

Ca sera à vous de les recréer.

3. Étape 1 – Créer l'entité AuditLog

Nous allons commencer par modéliser ce que nous voulons journaliser.

3.1. Spécifications de l'entité :

Créez un fichier :

- `src/main/kotlin/org/ldv/AppStarter_ToDoList/entity/AuditLog.kt`

L'entité `AuditLog` doit contenir au minimum :

- `id: Long` : Identifiant auto-généré,
- `username: String` : – Nom de l'utilisateur à l'origine de l'action,
- `action: String` : – Type d'action (LOGIN, LOGOUT, CREATE_TASK, etc.),
- `details: String?` : – Texte décrivant l'action (facultatif),
- `ipAddress: String` : – Adresse IP du client,
- `timestamp: LocalDateTime` : Date et heure de l'action (par défaut `LocalDateTime.now()`).

RGPD : Droits & obligations

La journalisation des accès constitue un **traitement de données personnelles** (identifiant, IP, horodatage...). Elle est autorisée mais strictement encadrée par le **RGPD**.

Finalité : Les logs doivent avoir une finalité claire (sécurité, diagnostic, traçabilité).

- **Interdit** : collecter "au cas où".

Minimisation : Ne journaliser que ce qui est nécessaire : *utilisateur, action, date/heure, IP*.

- **Jamais** de mot de passe ou de données sensibles.

Conservation limitée : Durée proportionnée (souvent 6 à 12 mois).

- Une purge régulière doit être prévue.

Sécurisation : Les journaux doivent être protégés (accès réservés, intégrité garantie).

3.2. Code à implémenter :

☞ À écrire dans **AuditLog.kt** :

Implémenter le code de classe **AuditLog**.

```
package org.ldv.AppStarter_ToDoList.entity

import jakarta.persistence.*
import java.time.LocalDateTime

@Entity
data class AuditLog(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    val id: Long = 0,

    @Column(nullable = false)
    val username: String,

    @Column(nullable = false)
    val action: String,

    @Column(nullable = true)
    val details: String? = null,

    @Column(nullable = false)
    val ipAddress: String,

    @Column(nullable = false)
    val timestamp: LocalDateTime = LocalDateTime.now()
)
```

4. Étape 2 – Créer AuditLogRepository

Nous voulons pouvoir accéder aux logs en base de données.

Créez le fichier :

- `src/main/kotlin/org/ldv/AppStarter_ToDoList/repository/AuditLogRepository.kt`

4.1. Spécifications :

Le repository (DAO) doit permettre :

- D'accéder aux logs via `JpaRepository`;
- De récupérer tous les logs triés par date décroissante;
- De récupérer les logs d'un utilisateur donné, triés par date décroissante.

4.2. Code à implémenter :

✍ À écrire dans `AuditLogRepository.kt` :

```
package org.ldv.AppStarter_ToDoList.repository

import org.ldv.AppStarter_ToDoList.entity.AuditLog
import org.springframework.data.jpa.repository.JpaRepository

interface AuditLogRepository : JpaRepository<AuditLog, Long> {

    fun findAllByOrderByTimestampDesc(): List<AuditLog>

    fun findByUsernameOrderByTimestampDesc(username: String): List<AuditLog>
}
```

Comment Spring Boot implémente les méthodes du **AuditLogRepository** ?

Spring Data JPA permet de créer des requêtes **sans écrire** la moindre ligne de **SQL** ou **JPQL**. Il analyse **le nom des méthodes** déclarées dans l'interface du **repository** et génère **automatiquement** la requête correspondante.

Exemple dans AuditLogRepository :

```
fun findAllByOrderByTimestampDesc(): List<AuditLog>
fun findByUsernameOrderByTimestampDesc(username: String): List<AuditLog>
```

Ces noms respectent une grammaire interne définie par Spring Data :

- **findBy...** → clause WHERE
- **OrderBy...Desc** → tri DESC sur un champ
- **Username** → correspond au champ de l'entité
- **Timestamp** → champ utilisé pour l'ordre chronologique

Ainsi, **Spring** génère automatiquement :

- **SELECT * FROM audit_log ORDER BY timestamp DESC**
- **SELECT * FROM audit_log WHERE username = ? ORDER BY timestamp DESC**

Aucun code SQL n'est nécessaire : c'est le nom de la méthode qui décrit la requête.

5. Étape 3 – Créer AuditLogService

Ce service centralise la logique de journalisation :

- Création des logs
- Lecture des logs.

Créez le fichier :

- `src/main/kotlin/org/ldv/AppStarter_ToDoList/service/AuditLogService.kt`

5.1. Spécifications :

Le service doit proposer au minimum :

```
fun log(username: String, action: String, details: String?, request:  
HttpServletResponse)
```

```
fun getAllLogs(): List<AuditLog>
```

```
fun getUserLogs(username: String): List<AuditLog>
```

L'adresse **IP** sera récupérée via l'objet **HttpServletRequest** :

```
val ip = request.remoteAddr
```

🌐 Qu'est-ce que **HttpServletRequest** ?

HttpServletRequest est un objet fourni par **Spring MVC** qui représente la **requête HTTP** envoyée par le client au serveur. Il permet d'accéder facilement à des informations sur la requête, par exemple :

- l'adresse **IP** du client (`request.remoteAddr`),
- les en-têtes **HTTP** (User-Agent, Accept...),
- les paramètres transmis,
- l'**URL** appelée,
- la méthode **HTTP** (`GET, POST...`).

Dans ce TP, nous l'utilisons principalement pour **récupérer l'adresse IP** lors de la journalisation des actions (`auditLogService.log(...)`).

L'IMPLEMENTATION EST SUR LA PAGE SUIVANTE -->

5.2. Code à implémenter :

☞ À écrire dans `AuditLogService.kt` :

```
package org.ldv.AppStarter_ToDoList.service

import jakarta.servlet.http.HttpServletRequest
import org.ldv.AppStarter_ToDoList.entity.AuditLog
import org.ldv.AppStarter_ToDoList.repository.AuditLogRepository
import org.springframework.stereotype.Service

@Service
class AuditLogService(
    private val auditLogRepository: AuditLogRepository
) {

    fun log(username: String, action: String, details: String?, request: HttpServletRequest) {
        val ip = request.remoteAddr

        val logEntry = AuditLog(
            username = username,
            action = action,
            details = details,
            ipAddress = ip
        )

        auditLogRepository.save(logEntry)
    }

    fun getAllLogs(): List<AuditLog> =
        auditLogRepository.findAllByOrderByTimestampDesc()

    fun getUserLogs(username: String): List<AuditLog> =
        auditLogRepository.findByUsernameOrderByTimestampDesc(username)
}
```

6. Étape 4 – Journaliser login et logout (SecurityConfig)

Nous allons maintenant journaliser :

- chaque **connexion réussie** (**LOGIN**),
- chaque **déconnexion** (**LOGOUT**).

Ouvrir :

- `src/main/kotlin/org/ldv/AppStarter_ToDoList/config/SecurityConfig.kt`

La version starter configure déjà :

- Les URL publiques,
- La page de login,
- La redirection par défaut,
- La déconnexion.

Nous allons modifier cette configuration pour appeler **AuditLogService**.

6.1. Injection de **AuditLogService** :

Modifier la signature de la classe :

```
package org.ldv.AppStarter_ToDoList.config

import org.ldv.AppStarter_ToDoList.service.AuditLogService
// ... autres imports ...

@Configuration
@EnableWebSecurity
class SecurityConfig(
    private val auditLogService: AuditLogService
) {
    // ...
}
```

(si la classe avait déjà des paramètres de constructeur, ajoutez simplement `private val auditLogService: AuditLogService` à la liste).

6.2. Handler (gestionnaire) de succès d'authentification :

Ajouter une fonction privée dans la classe :

```
import org.springframework.security.web.authentication.AuthenticationSuccessHandler

// ...

private fun customAuthenticationSuccessHandler(): AuthenticationSuccessHandler =
    AuthenticationSuccessHandler { request, response, authentication ->
        val username = authentication.name

        auditLogService.log(
            username = username,
            action = "LOGIN",
            details = "Connexion réussie",
            request = request
        )

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

6.3. Handler (gestionnaire) de logout :

Ajouter également :

```
import org.springframework.security.web.authentication.logout.LogoutSuccessHandler

// ...

private fun customLogoutSuccessHandler(): LogoutSuccessHandler =
    LogoutSuccessHandler { request, response, authentication ->
        val username = authentication?.name ?: "anonymous"

        auditLogService.log(
            username = username,
            action = "LOGOUT",
            details = "Déconnexion",
            request = request
        )

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

6.4. Brancher les handlers (gestionnaires) dans filterChain :

Dans la méthode `filterChain(http: HttpSecurity)` remplacer les blocs de configuration `formLogin` et `logout` existants par :

```
.formLogin { form ->
    form
        .loginPage("/login")
        .successHandler(customAuthenticationSuccessHandler())
        .permitAll()
}
.logout { logout ->
    logout
    .logoutSuccessHandler(customLogoutSuccessHandler())
    .permitAll()
}
```

Le reste de la configuration (`authorizeHttpRequests`, `csrf`, `headers`, etc.) reste inchangée.

7. Étape 5 – Journaliser les actions sur les tâches (TaskController)

Nous voulons maintenant journaliser :

- La création d'une tâche (**CREATE_TASK**);
- La mise à jour d'une tâche (**UPDATE_TASK**);
- La suppression d'une tâche (**DELETE_TASK**).

Ouvrir :

- `src/main/kotlin/org/ldv/AppStarter_ToDoList/controller/TaskController.kt`

Dans la version starter, ce contrôleur :

- Gère les URL `/tasks`;
- Utilise `TaskService` et `UserService`;
- **contient des commentaires du type :**
`// Journalisation à implémenter par les étudiants ici plus tard.`

7.1. Injection de `AuditLogService` :

Ajouter `AuditLogService` en paramètre du constructeur :

```
package org.ldv.AppStarter_ToDoList.controller

import jakarta.servlet.http.HttpServletRequest
import org.ldv.AppStarter_ToDoList.service.AuditLogService
// ... autres imports ...

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

7.2. Journaliser la création d'une tâche :

Dans la méthode qui gère **POST /tasks/create**, ajouter un paramètre **request: HttpServletRequest** et appeler le service d'audit en fin de méthode.

☞ **Exemple de code à ajouter :**

```
@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)

    auditLogService.log(
        username = user.username,
        action = "CREATE_TASK",
        details = "Création de la tâche : $title",
        request = request
    )

    return "redirect:/tasks"
}
```

7.3. Journaliser la mise à jour d'une tâche :

Dans la méthode qui gère `POST /tasks/update/{id}`, ajouter aussi `request: HttpServletRequest` et appeler `auditLogService.log` après la mise à jour.

👉 Exemple de code :

```
@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) {
        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
    )

    auditLogService.log(
        username = authentication.name,
        action = "UPDATE_TASK",
        details = "Modification tâche #\$id (titre=\$title, statut=\$status)",
        request = request
    )

    return "redirect:/tasks"
}
```

7.4. Journaliser la suppression d'une tâche :

Dans la méthode qui gère **POST /tasks/delete/{id}**, ajouter **request: HttpServletRequest** et appeler **auditLogService.log** si la suppression est autorisée.

👉 **Exemple de code :**

```
@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)

        auditLogService.log(
            username = authentication.name,
            action = "DELETE_TASK",
            details = "Suppression tâche #$id",
            request = request
        )
    }

    return "redirect:/tasks"
}
```

8. Étape 6 – Afficher les logs dans la page Admin

Ouvrir :

- `src/main/kotlin/org/ldv/AppStarter_ToDoList/controller/AdminController.kt`

Dans la version starter, le contrôleur envoie une **liste vide** à la vue :

```
model.addAttribute("logs", emptyList<Any>())
```

Nous voulons maintenant utiliser `AuditLogService`.

8.1. Injection du service :

Modifier la classe pour injecter `AuditLogService` :

```
package org.ldv.AppStarter_ToDoList.controller

import org.ldv.AppStarter_ToDoList.service.AuditLogService
import org.springframework.stereotype.Controller
import org.springframework.ui.Model
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.RequestMapping

@Controller
@RequestMapping("/admin")
class AdminController(
    private val auditLogService: AuditLogService
) {

    @GetMapping
    fun adminPanel(model: Model): String {
        val logs = auditLogService.getAllLogs()
        model.addAttribute("logs", logs)
        return "admin"
    }
}
```

La vue `admin.html` affichera désormais les vrais logs.

9. Étape 7 – Tests manuels

Relancer l'application et tester, dans cet ordre :

1. Connexion avec **user** / **user123**.
2. Création d'une tâche.
3. Modification de la tâche.
4. Suppression de la tâche.
5. Déconnexion.

Après chaque action, consulter la page :

<http://localhost:8080/admin>

Vérifier que chaque action correspond bien à une ligne de log (*username; action; détails; IP; timestamp*).

Vous pouvez aussi vérifier le contenu directement dans la console **H2** :

- URL : <http://localhost:8080/h2-console>
- JDBC : **jdbc:h2:mem:todolistdb**
- Utilisateur : **sa**
- Mot de passe : **(vide)**

Puis exéutez :

```
SELECT * FROM AUDIT_LOG ORDER BY TIMESTAMP DESC;
```

Bravo pour le travail ! 
