# **Lab: Events Scheduling Tasks and Caching**

## **Exercise 1: Spy the Agent**

*An underground network has infiltrated the agency. Reports are still being submitted to headquarters, but there are suspicions that one or more agents are leaking sensitive data.*

You are tasked with **secretly attaching a mechanism** to the existing system - one that monitors and logs every report submitted, without alerting the agents or disrupting the current workflow. The log must record the agent's name, the report contents, and the exact time of submission, in the following format:

**[*<timestamp>*] Report received from Agent *<agent>*: *<message>***

You are **not allowed** to place logging logic in the controller or service directly. Logging utilities such as Lombok's @Slf4j **must not be used** for this purpose. The solution must be implemented using Spring's event system. The controller, service, and in-memory report storage are already implemented. You are only responsible for extending the system to monitor agent activity discreetly.  
You may test your implementation by sending a **POST** request to **http://localhost:8080/reports** with the following JSON content:

**{**

**"agent"**: **"Shadow"**,

**"description"**: **"Coordinates confirmed. Proceeding to extraction."**

**}**

## **Exercise 2: Isolate the Threat**

*The agency has activated* ***Protocol X****: all reports must now be monitored in real time on a per-agent level.*

Two new security units must be introduced - one responsible for tracking report volume, and another for identifying traitors within the network.

**Activity Tracker** - this is the first unit you must implement, that is responsible for **logging agent activity**. After every report is submitted, log the total number of reports the agency has received from the submitting agent (including the current one). The log must follow this format:  
"**Total reports submitted: *<count>* by Agent *<agent>****"*

**Threat Detection** - this is the second unit responsible for handling **internal threats**. If an incoming report contains any of the following keywords: **drop**, **target**, **extraction**, **payload**, or **signal**, perform the following:

1. Log the message – "**Agent *<agent>* triggered a security alert. Flagged keywords detected**".
2. Add the agent to a **blacklist**.
3. **Block** the agent from accessing the system. Any future request submitted by this agent **must be rejected** with the following message: "**Access denied. You are under investigation.**"

**You are not allowed to modify the controller or the service for implementing the blocking mechanism!**

## **Exercise 3: Pattern Scan**

An internal directive has initiated a global **pattern scan**. Each time a report is submitted, all previously intercepted reports must be analyzed for hidden patterns, and potential agent-to-agent correlations. The operation is **slow** and must be handled discreetly. The scanning unit is not permitted to delay the report submission process in any way - agents must not suspect that their reports are being analyzed.

You are provided with an **AnalysisService** containing a method named **performPatternScan()** that performs the full scan. The scan must be triggered after each report submission. However, it must not delay or block the submission process in any way. The operation must run silently in the background, agents should not experience delays when submitting reports.

## **Exercise 4: Trace Agent Activity**

The agency has activated live **surveillance mode**. Every **60 seconds**, a **tactical summary** must be generated, listing the number of reports received in the last minute and the agents who submitted them. You are tasked with implementing an **automated background process** that silently monitors the system and prints a status report once per minute. You are not allowed to trigger this process manually, it must execute automatically every full minute. The summary must include **only reports submitted in the last 60 seconds**, and the list of agents must not contain duplicates.

The summary must follow this format:

***[SUMMARY - 17:00:00]  
Reports in the last minute: 3  
Agents: Shadow, Nova, Viper***

## **Exercise 5: Purge the Archive**

To reduce the risk of data exposure, the agency has issued a directive to remove all outdated reports from the system. Any report **older than 1 day** must be **deleted automatically** - no manual intervention is allowed. You are tasked with implementing an automated background process that runs **every 90 seconds** and **purges** all reports **older than 1 day** from the system's archive. Each purge must be followed by a log that reports how many reports were deleted.

The output must follow this format:

***[PURGE COMPLETE - 03:00:00]*  
*Expired reports removed: 7  
Remaining reports: 2***

## **Exercise 6: Optimize the Feed**

*The agency’s analyst dashboard currently retrieves all intercepted reports with a noticeable delay, caused by a heavy internal process.*

Your task is to **improve** the existing implementation so that the **first request** for the full report feed incurs the **delay**, but **every subsequent request** returns **instantly**, unless new data has been submitted in the meantime.

You already have the controller and service methods implemented with a simulated delay in the service. Your job is to **minimize unnecessary delays** and make the **dashboard experience more responsive**.

You may test your implementation by fetching the reports via:   
**GET http://localhost:8080/reports**