TP3: Software Logging and Observability

Miguel Delhelle, Oscar Jimenez Flores

December 9, 2025

Contents

1 Introduction **2**

2 System Architecture & Design **2**
- 2.1 UML Class Diagram 2
- 2.2 Profiling Workflow (Exercise 1) 2

3 Exercise 1: Implementation Details **3**
- 3.1 Structured Logging 3
- 3.2 Profiling Results 3

4 Exercise 2: Distributed Tracing Implementation **3**
- 4.1 Sequence Diagram: Context Propagation 3
- 4.2 Frontend Implementation 4

5 Results and Screenshots **4**
- 5.1 Application Interface 4
- 5.2 Observability Results (Jaeger) 5

6 Conclusion **6**

A Log Artifacts **6**

1 Introduction

This report documents the implementation of a software system designed to explore Structured Logging and Distributed Observability. The project evolves from a monolithic logging strategy (Exercise 1) to a full distributed tracing architecture (Exercise 2).

2 System Architecture & Design

2.1 UML Class Diagram

The application follows a standard Controller-Service-Repository pattern. The `StoreService` is the central point for business logic and logging instrumentation.

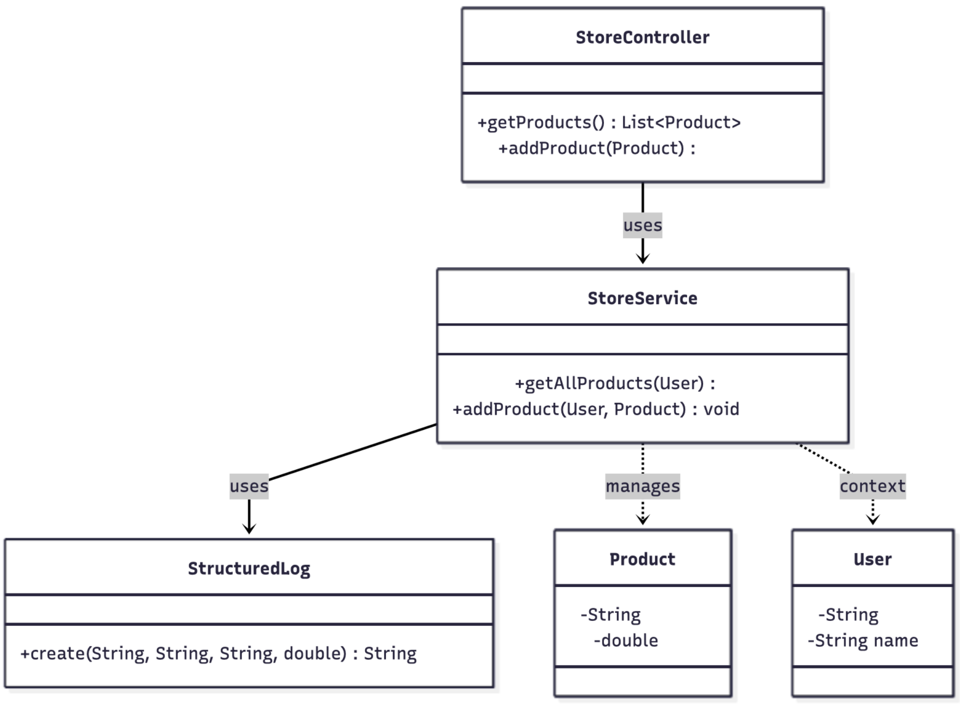


Figure 1: UML Class Diagram showing the dependency between Controller, Service, and the Logging Utility.

2.2 Profiling Workflow (Exercise 1)

User profiles are constructed via a post-execution analysis pipeline. Logs are generated in JSON format during runtime and parsed offline to categorize user behavior.

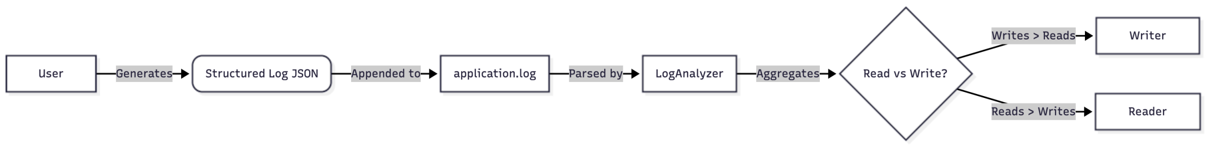


Figure 2: Workflow for generating User Profiles from Structured Logs.

3 Exercise 1: Implementation Details

3.1 Structured Logging

We utilized **SLF4J/Logback** to generate JSON logs.

```java
public static String create(String userId, String op, String ctx, double
    metric) {
    Map<String, Object> log = new HashMap<>();
    log.put("who", userId);
    log.put("what", op); // READ or WRITE
    return mapper.writeValueAsString(log);
}
```

Listing 1: StructuredLog.java Helper

3.2 Profiling Results

The `LogAnalyzer` successfully processed the logs.

```
User U1: [READER PROFILE]  (Reads: 5, Writes: 0)
User U2: [WRITER PROFILE]  (Reads: 0, Writes: 3)
```

Listing 2: Console Output: Constructed Artifacts

4 Exercise 2: Distributed Tracing Implementation

4.1 Sequence Diagram: Context Propagation

The core challenge was propagating the W3C Trace Context from the React Frontend to the Spring Boot Backend.

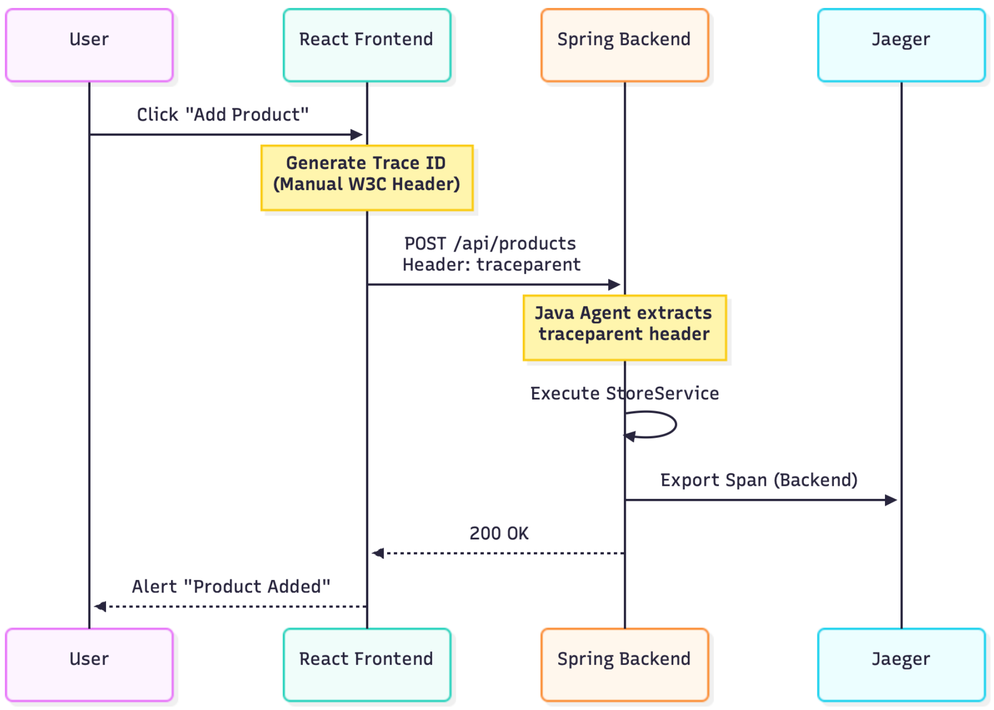


Figure 3: Sequence Diagram illustrating Manual Header Injection and Agent Extraction.

4.2 Frontend Implementation

We implemented manual header injection in React to bypass library version conflicts.

```
const generateTraceHeader = () => {
    // Generates 00-{traceId}-{spanId}-01
    return `00-${randomHex(32)}-${randomHex(16)}-01`;
};
// Used in fetch headers: 'traceparent': generateTraceHeader()
```

Listing 3: App.js: Manual Injection

5 Results and Screenshots

5.1 Application Interface

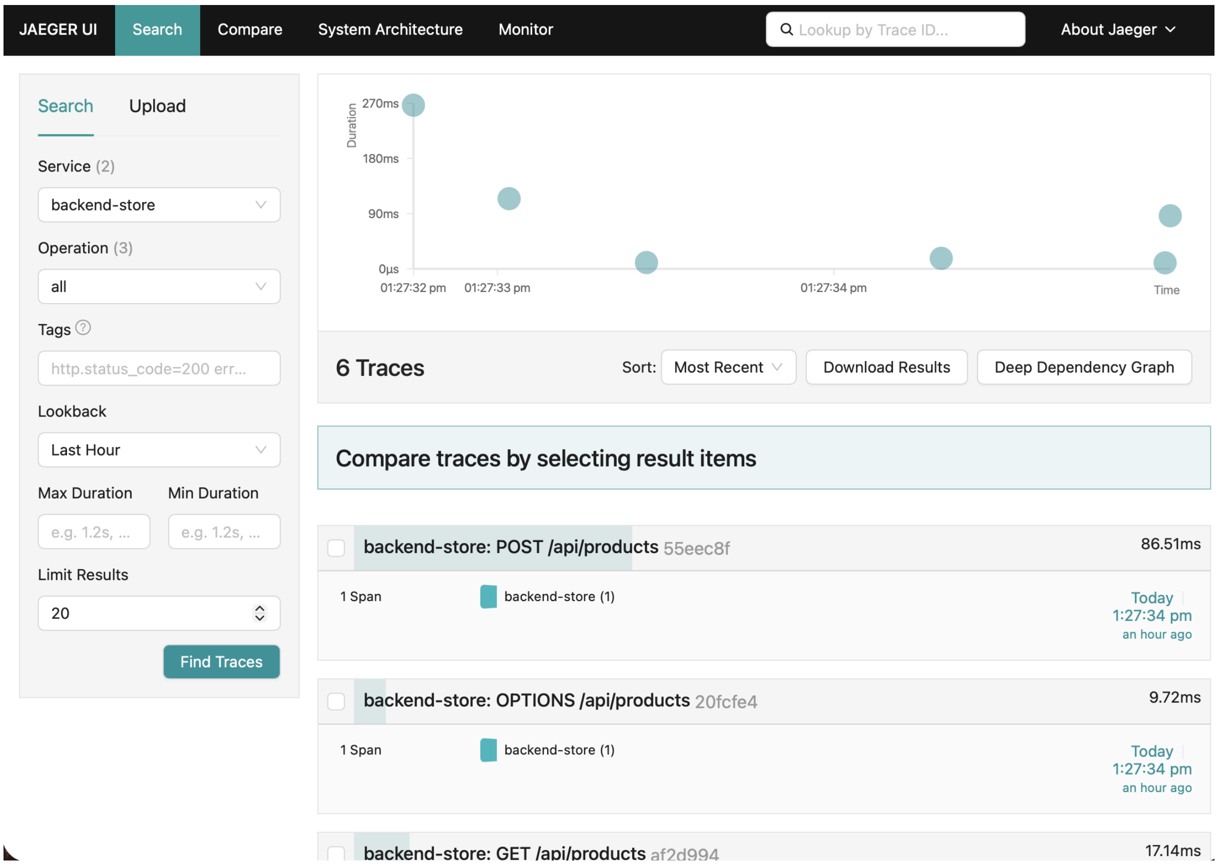
The React frontend allows simulating "Read" and "Write" scenarios.

Store Frontend (Simulation Mode)

Figure 4: Frontend UI (React) running on localhost:3000.

5.2 Observability Results (Jaeger)

After executing the scenarios, traces were captured in Jaeger.



Service (2): backend-store
Operation (3): all
Tags: http.status_code=200 err...
Lookback: Last Hour
Max Duration: e.g. 1.2s, ...
Min Duration: e.g. 1.2s, ...
Limit Results: 20

6 Traces

Sort: Most Recent

Compare traces by selecting result items

backend-store: POST /api/products 55eec8f — 86.51ms
1 Span — backend-store (1) — Today 1:27:34 pm, an hour ago

backend-store: OPTIONS /api/products 20fcfe4 — 9.72ms
1 Span — backend-store (1) — Today 1:27:34 pm, an hour ago

backend-store: GET /api/products af2d994 — 17.14ms

Figure 5: Jaeger UI showing traces captured for the 'backend-store' service.

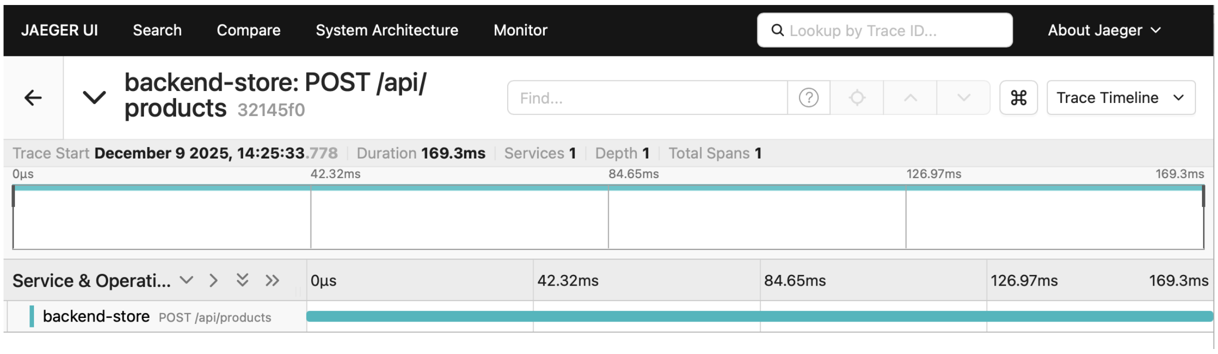


Figure 6: Detailed Trace View. The blue bar represents the Backend processing the request initiated by the Frontend.

6 Conclusion

The project successfully demonstrated that manual context propagation (Frontend) combined with auto-instrumentation (Backend) allows for complete visibility into distributed transactions, fulfilling the requirements of Exercise 2.

A Log Artifacts

```
{"timestamp":"2025-12-09...","who":"U1","what":"READ","where":"
    getProduct","metric":1500.0}
{"timestamp":"2025-12-09...","who":"U2","what":"WRITE","where":"
    addProduct","metric":20.0}
```

Listing 4: Snippet of generated application.log