This document contains the complete source code and Maven build files for the two major versions of the "Nexus Logger" library.

- **Version 1.0.0** is built with Java 8 and establishes the initial API.
- **Version 2.0.0** is a major, breaking-change upgrade built for Java 21, showcasing a significant modernization of the API for improved performance, security, and developer ergonomics.

## Side-by-Side Application Code Comparison

| Aspect | Nexus Logger v1 (Java 8) | Nexus Logger v2 (Java 21) |
|---|---|---|
| **Logging a Message** | Relies on manual `Map` creation, which is verbose. The message and context are separate arguments. &lt;br>&lt;br> `java<br>logger.info(<br> "Processing payment",<br> new HashMap<>() {{<br> put("orderId", orderId);<br> put("amount", amount);<br> }}<br>);<br>` | Uses a `StringTemplate` for a clean, inline, and unified logging statement. The structure is part of the message itself. &lt;br>&lt;br> ``java<br>// `LOG` is the template processor<br>logger.info(<br> LOG."Processing payment for {orderId} of {amount}"<br>);<br>`` |
| **Developer Ergonomics** | Clunky. Developers have to remember to build a map. It's easy to forget context or make typos in keys. | Excellent. Feels like natural string interpolation, but with the power of structured logging behind the scenes. It's concise and readable. |
| **Performance** | Good. Asynchronous dispatching prevents blocking the main application thread. `Supplier` usage avoids premature `toString()` calls. | Superior. Dispatching I/O-heavy log events (e.g., to Kafka or a logging service) is done on virtual threads, allowing for massive throughput with fewer OS resources. |

| | | |
|---|---|---|
| **Security** | Good. But vulnerable to common mistakes like `logger.info("User " + user.getName())` which bypasses structured logging. | Excellent. The string template processor approach is inherently more secure. It can be designed to automatically sanitize values and prevents injection issues common with basic string concatenation. |

Each section represents a complete, packageable Maven project.

## Nexus Logger v1.0.0 (for Java 8)

This version establishes the core concepts: a facade API, structured logging via `Map` objects, deferred execution with `Supplier`, and `ThreadLocal`-based context propagation.

### File: `pom.xml`

This file defines the project as a Java 8 artifact with a version of `1.0.0`.

Unset

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.mycorp.nexus</groupId>
    <artifactId>nexus-logger</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <name>Nexus Logger v1</name>
    <description>A structured, distributed logging library for
enterprise use (Java 8).</description>

    <properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding
>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

</project>
```

### Source Code (`src/main/java`)

**Package: `com.mycorp.nexus.logger.v1`**

**File: `Logger.java`**

The primary public interface for application developers.

Unset
```
package com.mycorp.nexus.logger.v1;

import java.util.Map;
import java.util.function.Supplier;

/**
 * The main interface for logging in Nexus Logger v1.
 */
public interface Logger {
    boolean isInfoEnabled();
    boolean isDebugEnabled();

    void info(String message, Map<String, Object> context);
    void info(String message, Supplier<Map<String, Object>>
contextSupplier);
    void warn(String message, Map<String, Object> context);
    void error(String message, Throwable t, Map<String, Object>
context);
    void debug(String message, Supplier<Map<String, Object>>
contextSupplier);
}
```

**File:** **`NexusLoggerFactory.java`**

The entry point for obtaining a logger instance.

Unset
```
package com.mycorp.nexus.logger.v1;

import com.mycorp.nexus.logger.v1.internal.LoggerImpl;
import com.mycorp.nexus.logger.v1.internal.LoggingProvider;

/**
 * Factory for creating Logger instances.
 */
public final class NexusLoggerFactory {
    private static final LoggingProvider provider = new
LoggingProvider();

    public static Logger getLogger(Class<?> clazz) {
        return provider.getLogger(clazz.getName());
    }
}
```

**File:** **`NexusContext.java`**

A static helper for managing request-level context via a `ThreadLocal`.

```
Unset
package com.mycorp.nexus.logger.v1;

import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

/**
 * Manages thread-local context for logs (e.g., traceId, userId).
 * NOTE: It is critical to call clear() in a finally block.
 */
public final class NexusContext {
    private static final ThreadLocal<Map<String, Object>> CONTEXT
=
        ThreadLocal.withInitial(HashMap::new);

    private NexusContext() {}

    public static void set(String key, Object value) {
        CONTEXT.get().put(key, value);
    }

    public static Map<String, Object> get() {
        return Collections.unmodifiableMap(new
HashMap<>(CONTEXT.get()));
    }

    public static void clear() {
        CONTEXT.remove();
    }
}
```

**Package:** `com.mycorp.nexus.logger.v1.internal`

Implementation details hidden from the public API.

**File:** `LoggerImpl.java`

The internal implementation of the `Logger` interface.

```
Unset
package com.mycorp.nexus.logger.v1.internal;

import com.mycorp.nexus.logger.v1.Logger;
import com.mycorp.nexus.logger.v1.NexusContext;

import java.util.Collections;
import java.util.Map;
import java.util.function.Supplier;

public class LoggerImpl implements Logger {
    private final String name;
    private final LogDispatcher dispatcher;
    private final LogLevel configuredLevel;

    public LoggerImpl(String name, LogDispatcher dispatcher,
LogLevel configuredLevel) {
        this.name = name;
        this.dispatcher = dispatcher;
        this.configuredLevel = configuredLevel;
    }

    @Override
    public boolean isInfoEnabled() {
        return
configuredLevel.isAsOrMoreSpecificThan(LogLevel.INFO);
    }

    @Override
    public boolean isDebugEnabled() {
        return
configuredLevel.isAsOrMoreSpecificThan(LogLevel.DEBUG);
    }

    @Override
    public void info(String message, Map<String, Object> context)
{
        if (isInfoEnabled()) {
            dispatch(LogLevel.INFO, message, null, context);
        }
    }

    @Override
    public void info(String message, Supplier<Map<String,
Object>> contextSupplier) {
        if (isInfoEnabled()) {
            dispatch(LogLevel.INFO, message, null,
contextSupplier.get());
        }
    }

    @Override
    public void warn(String message, Map<String, Object> context)
{
        if
(configuredLevel.isAsOrMoreSpecificThan(LogLevel.WARN)) {
            dispatch(LogLevel.WARN, message, null, context);
        }
    }

    @Override
    public void error(String message, Throwable t, Map<String,
Object> context) {
         if
(configuredLevel.isAsOrMoreSpecificThan(LogLevel.ERROR)) {
            dispatch(LogLevel.ERROR, message, t, context);
        }
    }

    @Override
    public void debug(String message, Supplier<Map<String,
Object>> contextSupplier) {
        if (isDebugEnabled()) {
            dispatch(LogLevel.DEBUG, message, null,
contextSupplier.get());
        }
    }

    private void dispatch(LogLevel level, String message,
Throwable t, Map<String, Object> context) {
        MutableLogEvent event = new MutableLogEvent(
            System.currentTimeMillis(),
            level.name(),
            this.name,
            Thread.currentThread().getName(),
            message,
            t,
            context == null ? Collections.emptyMap() : context,
            NexusContext.get()
        );
        dispatcher.dispatch(event);
    }
}
```

**File: `MutableLogEvent.java`**

A mutable POJO representing a single log event.

Unset
```
package com.mycorp.nexus.logger.v1.internal;

import java.io.PrintWriter;
import java.io.StringWriter;
import java.util.Map;
import java.util.stream.Collectors;
```

```
public class MutableLogEvent {
    private final long timestamp;
    private final String level;
    private final String loggerName;
    private final String threadName;
    private final String message;
    private final Throwable throwable;
    private final Map<String, Object> context;
    private final Map<String, Object> globalContext;

    public MutableLogEvent(long timestamp, String level, String
loggerName, String threadName, String message, Throwable
throwable, Map<String, Object> context, Map<String, Object>
globalContext) {
        this.timestamp = timestamp;
        this.level = level;
        this.loggerName = loggerName;
        this.threadName = threadName;
        this.message = message;
        this.throwable = throwable;
        this.context = context;
        this.globalContext = globalContext;
    }

    public String toJson() {
        StringBuilder sb = new StringBuilder();
        sb.append("{\n");
        sb.append("  \"timestamp\": \"").append(new
java.util.Date(timestamp).toInstant().toString()).append("\",\n")
;
        sb.append("  \"level\":
\"").append(level).append("\",\n");
        sb.append("  \"logger\":
\"").append(loggerName).append("\",\n");
```

```
        sb.append("  \"thread\":
\"").append(threadName).append("\",\n");
        sb.append("  \"message\":
\"").append(escapeJson(message)).append("\",\n");

        sb.append("  \"context\": {\n");
        appendMap(sb, context);
        sb.append("  },\n");

        sb.append("  \"globalContext\": {\n");
        appendMap(sb, globalContext);
        sb.append("  },\n");

        if (throwable != null) {
            StringWriter sw = new StringWriter();
            throwable.printStackTrace(new PrintWriter(sw));
            sb.append("  \"stacktrace\":
\"").append(escapeJson(sw.toString())).append("\"\n");
        } else {
            // remove trailing comma if no stacktrace
            if (sb.charAt(sb.length() - 2) == ',') {
               sb.setLength(sb.length() - 2);
               sb.append("\n");
            }
        }
        sb.append("}");
        return sb.toString();
    }

    private void appendMap(StringBuilder sb, Map<String, Object>
map) {
        if (map != null && !map.isEmpty()) {
            String mapJson = map.entrySet().stream()
                .map(entry -> "    \"" +
escapeJson(entry.getKey()) + "\": " +
formatValue(entry.getValue()))
```

```
                    .collect(Collectors.joining(",\n"));
                sb.append(mapJson).append("\n");
            }
        }

        private String formatValue(Object value) {
            if (value instanceof String) {
                return "\"" + escapeJson((String) value) + "\"";
            }
            return value.toString();
        }

        private String escapeJson(String value) {
            if (value == null) return "";
            return value.replace("\\", "\\\\")
                        .replace("\"", "\\\"")
                        .replace("\b", "\\b")
                        .replace("\f", "\\f")
                        .replace("\n", "\\n")
                        .replace("\r", "\\r")
                        .replace("\t", "\\t");
        }
    }
```

**File:** **`LogLevel.java`** **&** **`LogDispatcher.java`** **&** **`LoggingProvider.java`**

Internal support classes.

```
Unset
package com.mycorp.nexus.logger.v1.internal;

public enum LogLevel {
    DEBUG(0), INFO(1), WARN(2), ERROR(3);
    private final int level;
    LogLevel(int level) { this.level = level; }
```

~~~
    public boolean isAsOrMoreSpecificThan(LogLevel other) {
        return this.level >= other.level;
    }
}
```java
package com.mycorp.nexus.logger.v1.internal;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.ThreadFactory;

public class LogDispatcher {
    private final ExecutorService executor;

    public LogDispatcher() {
        ThreadFactory threadFactory = r -> {
            Thread t = new Thread(r, "nexus-logger-dispatcher");
            t.setDaemon(true);
            return t;
        };
        this.executor =
Executors.newSingleThreadExecutor(threadFactory);
    }

    public void dispatch(MutableLogEvent event) {
        executor.submit(() ->
System.out.println(event.toJson()));
    }

    public void shutdown() {
        executor.shutdown();
    }
}
```java
package com.mycorp.nexus.logger.v1.internal;
~~~

```
import com.mycorp.nexus.logger.v1.Logger;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;

public class LoggingProvider {
    private final LogDispatcher dispatcher = new LogDispatcher();
    private final ConcurrentMap<String, Logger> loggers = new
ConcurrentHashMap<>();
    // In a real library, this would be loaded from a config
file.
    private final LogLevel rootLevel = LogLevel.INFO;

    public Logger getLogger(String name) {
        return loggers.computeIfAbsent(name, n -> new
LoggerImpl(n, dispatcher, rootLevel));
    }
}
```

## Nexus Logger v2.0.0 (for Java 21)

This version is a complete modernization, using Virtual Threads, Records, Sealed Interfaces, Scoped Values, and String Templates. All v1 APIs that are being replaced are formally deprecated.

### File: `pom.xml`

This file defines the project as a Java 21 artifact with a version of `2.0.0` and enables preview features.

```
Unset
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.mycorp.nexus</groupId>
    <artifactId>nexus-logger</artifactId>
    <version>2.0.0</version>
    <packaging>jar</packaging>

    <name>Nexus Logger v2</name>
    <description>A modern, structured, distributed logging
library (Java 21).</description>

    <properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding
>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
    </properties>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.11.0</version>
                <configuration>
                    <compilerArgs>
                        <arg>--enable-preview</arg>
                    </compilerArgs>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

## Source Code (`src/main/java`)

**Package:** `com.mycorp.nexus.logger.v2`

**File:** `Logger.java`

The modernized public interface.

```
Unset
package com.mycorp.nexus.logger.v2;

import com.mycorp.nexus.logger.v2.internal.LogEvent;
import com.mycorp.nexus.logger.v2.internal.TemplateProcessor;

import java.lang.StringTemplate;
import java.util.Map;
import java.util.function.Supplier;

/**
 * The main interface for logging in Nexus Logger v2.
 * Uses String Templates for modern, secure, and structured
logging.
 */
public interface Logger {
    /**
     * The primary StringTemplate processor for creating log
events.
     */
    StringTemplate.Processor<LogEvent, RuntimeException> LOG =
TemplateProcessor.INSTANCE;

    // --- Modern v2 API ---
    void info(StringTemplate template);
    void warn(StringTemplate template);
    void error(Throwable t, StringTemplate template);
    void debug(StringTemplate template);

    boolean isInfoEnabled();
    boolean isDebugEnabled();

    // --- Deprecated v1 API for backward compatibility ---

    /**
     * @deprecated since 2.0.0, for removal in 3.0.0. Use the
StringTemplate-based info() method instead.
     * Example: logger.info(LOG."Message with {key}");
     */
    @Deprecated(forRemoval = true, since = "2.0.0")
    default void info(String message, Map<String, Object>
context) {
        // Implementation provided for backward compatibility.
        StringTemplate st = StringTemplate.of(message);
        LogEvent event = new LogEvent(System.currentTimeMillis(),
"INFO", getLoggerName(), Thread.currentThread().getName(),
st.interpolate(), null, context, Map.of());
        dispatchLogEvent(event);
    }

    /**
     * @deprecated since 2.0.0, for removal in 3.0.0. The
StringTemplate API is implicitly deferred.
     */
    @Deprecated(forRemoval = true, since = "2.0.0")
    default void debug(String message, Supplier<Map<String,
Object>> contextSupplier) {
        if (isDebugEnabled()) {
             info(message, contextSupplier.get());
        }
    }

    // Internal methods for default implementations to call
    String getLoggerName();
    void dispatchLogEvent(LogEvent event);
}
```

**File: `NexusLoggerFactory.java`**

Factory for v2 loggers and the new `LoggerControl`.

```
Unset
package com.mycorp.nexus.logger.v2;

import com.mycorp.nexus.logger.v2.internal.LoggingProvider;
import com.mycorp.nexus.logger.v2.internal.LoggerControl;

public final class NexusLoggerFactory {
    private static final LoggingProvider provider = new
LoggingProvider();

    public static Logger getLogger(Class<?> clazz) {
        return provider.getLogger(clazz.getName());
    }

    public static LoggerControl getControl(String loggerName) {
        return provider.getControl(loggerName);
    }
}
```

**File: `NexusContext.java`**

Reimagined with `ScopedValue` for safe, immutable, lexically-scoped context.

```
Unset
package com.mycorp.nexus.logger.v2;

/**
 * Manages request-scoped context using Java 21's ScopedValue.
 * This is inherently safe and requires no cleanup.
 */
public final class NexusContext {
    private NexusContext() {}
    public static final ScopedValue<String> TRACE_ID =
ScopedValue.newInstance();
    public static final ScopedValue<String> USER_ID =
ScopedValue.newInstance();
}
```

**Package: `com.mycorp.nexus.logger.v2.config`**

Type-safe programmatic configuration.

**File: `LogLevel.java`**

```
Unset
package com.mycorp.nexus.logger.v2.config;

public enum LogLevel {
    DEBUG(0), INFO(1), WARN(2), ERROR(3);
    private final int level;
    LogLevel(int level) { this.level = level; }
    public boolean isAsOrMoreSpecificThan(LogLevel other) {
        return this.level >= other.level;
    }
}
```

**File: `NexusConfig.java`**

A builder for creating and applying configuration.

```
Unset
package com.mycorp.nexus.logger.v2.config;
```

```java
import com.mycorp.nexus.logger.v2.internal.LoggingProvider;
import com.mycorp.nexus.logger.v2.spi.LogSink;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class NexusConfig {
    private final LogLevel rootLevel;
    private final List<LogSink> sinks;

    private NexusConfig(LogLevel rootLevel, List<LogSink> sinks)
{
        this.rootLevel = rootLevel;
        this.sinks = sinks;
    }

    public LogLevel getRootLevel() { return rootLevel; }
    public List<LogSink> getSinks() { return sinks; }

    public static Builder builder() {
        return new Builder();
    }

    public static class Builder {
        private LogLevel level = LogLevel.INFO;
        private final List<LogSink> sinks = new ArrayList<>();

        public Builder level(LogLevel level) {
            this.level = Objects.requireNonNull(level);
            return this;
        }

        public Builder addSink(LogSink sink) {
            this.sinks.add(Objects.requireNonNull(sink));
            return this;
        }

        public NexusConfig buildAndApply() {
            NexusConfig config = new NexusConfig(level, sinks);
            LoggingProvider.applyConfig(config);
            return config;
        }
    }
}
```

**Package: `com.mycorp.nexus.logger.v2.spi`**

The modernized Service Provider Interface.

**File: `LogSink.java`**

A `sealed interface` for defining logging destinations.

Unset
```
package com.mycorp.nexus.logger.v2.spi;

import com.mycorp.nexus.logger.v2.internal.LogEvent;

/**
 * A sealed interface representing a destination for log events.
 */
public sealed interface LogSink permits ConsoleSink, FileSink {
    void dispatch(LogEvent event);
}
```

**File: `ConsoleSink.java` & `FileSink.java`**

Concrete, immutable record implementations of `LogSink`.

Unset

~~~
package com.mycorp.nexus.logger.v2.spi;

import com.mycorp.nexus.logger.v2.internal.LogEvent;
import com.mycorp.nexus.logger.v2.util.JsonUtil;

public record ConsoleSink() implements LogSink {
    @Override
    public void dispatch(LogEvent event) {
        System.out.println(JsonUtil.toJson(event));
    }
}
```java
package com.mycorp.nexus.logger.v2.spi;

import com.mycorp.nexus.logger.v2.internal.LogEvent;
import com.mycorp.nexus.logger.v2.util.JsonUtil;

import java.io.IOException;
import java.io.UncheckedIOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.Collections;

public record FileSink(String path) implements LogSink {
    @Override
    public void dispatch(LogEvent event) {
        try {
            String json = JsonUtil.toJson(event) + "\n";
            Files.write(Paths.get(path), json.getBytes(),
StandardOpenOption.CREATE, StandardOpenOption.APPEND);
        } catch (IOException e) {
            // In a real logger, this would write to a fallback
sink (like stderr).
            throw new UncheckedIOException(e);
        }
~~~

```
    }
}
```

**Package: `com.mycorp.nexus.logger.v2.internal`**

**File: `LogEvent.java`**

An immutable `record` representing a log event.

```
Unset
package com.mycorp.nexus.logger.v2.internal;

import java.util.Map;

/**
 * An immutable representation of a single log event.
 */
public record LogEvent(
    long timestamp,
    String level,
    String loggerName,
    String threadName,
    String message,
    Throwable throwable,
    Map<String, Object> context,
    Map<String, Object> scopedContext
) {}
```

**File: `LoggerImpl.java`**

The v2 implementation, using a Virtual Thread dispatcher.

Unset

```
package com.mycorp.nexus.logger.v2.internal;

import com.mycorp.nexus.logger.v2.Logger;
import com.mycorp.nexus.logger.v2.config.LogLevel;

import java.lang.StringTemplate;

public class LoggerImpl implements Logger {
    private final String name;
    private final LogDispatcher dispatcher;
    private volatile LogLevel configuredLevel;

    public LoggerImpl(String name, LogDispatcher dispatcher,
LogLevel configuredLevel) {
        this.name = name;
        this.dispatcher = dispatcher;
        this.configuredLevel = configuredLevel;
    }

    public void setLevel(LogLevel level) { this.configuredLevel =
level; }
    @Override public String getLoggerName() { return this.name; }
    @Override public void dispatchLogEvent(LogEvent event) {
dispatcher.dispatch(event); }

    @Override
    public boolean isInfoEnabled() {
        return
configuredLevel.isAsOrMoreSpecificThan(LogLevel.INFO);
    }

    @Override
    public boolean isDebugEnabled() {
        return
configuredLevel.isAsOrMoreSpecificThan(LogLevel.DEBUG);
    }
    @Override
    public void info(StringTemplate template) {
        if (isInfoEnabled()) {
            processAndDispatch(LogLevel.INFO, template, null);
        }
    }

    @Override
    public void warn(StringTemplate template) {
        if
(configuredLevel.isAsOrMoreSpecificThan(LogLevel.WARN)) {
            processAndDispatch(LogLevel.WARN, template, null);
        }
    }

    @Override
    public void error(Throwable t, StringTemplate template) {
        if
(configuredLevel.isAsOrMoreSpecificThan(LogLevel.ERROR)) {
            processAndDispatch(LogLevel.ERROR, template, t);
        }
    }

    @Override
    public void debug(StringTemplate template) {
        if (isDebugEnabled()) {
            processAndDispatch(LogLevel.DEBUG, template, null);
        }
    }

    private void processAndDispatch(LogLevel level,
StringTemplate template, Throwable t) {
        LogEvent event =
TemplateProcessor.INSTANCE.process(template);
        // Enrich the event with runtime info
        LogEvent enrichedEvent = new LogEvent(
            System.currentTimeMillis(),
            level.name(),
            this.name,
            Thread.currentThread().toString(), // More
descriptive with virtual threads
            event.message(),
            t,
            event.context(),
            TemplateProcessor.getScopedContext()
        );
        dispatcher.dispatch(enrichedEvent);
    }
}
```

**Other Internal Classes**

(`LoggerControl.java`, `LogDispatcher.java`, `LoggingProvider.java`, `TemplateProcessor.java`, `util/JsonUtil.java`)

These classes provide the core mechanics for configuration, dispatching, and template processing. For brevity, their stubs are fleshed out below.

Unset
~~~
// LoggerControl.java
package com.mycorp.nexus.logger.v2.internal;
import com.mycorp.nexus.logger.v2.config.LogLevel;
public class LoggerControl {
    private final LoggerImpl logger;
    public LoggerControl(LoggerImpl logger) { this.logger =
logger; }
    public void setLevel(LogLevel level) {
logger.setLevel(level); }
}
```java
~~~

~~~
// LogDispatcher.java
package com.mycorp.nexus.logger.v2.internal;
import com.mycorp.nexus.logger.v2.spi.LogSink;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class LogDispatcher {
    private final ExecutorService executor =
Executors.newVirtualThreadPerTaskExecutor();
    private volatile List<LogSink> sinks;

    public LogDispatcher(List<LogSink> sinks) { this.sinks =
sinks; }
    public void setSinks(List<LogSink> sinks) { this.sinks =
sinks; }

    public void dispatch(LogEvent event) {
        if (sinks != null && !sinks.isEmpty()) {
            executor.submit(() -> {
                for (LogSink sink : sinks) {
                    try {
                        sink.dispatch(event);
                    } catch (Exception e) {
                        // In a real library, handle sink
exceptions
                        e.printStackTrace();
                    }
                }
            });
        }
    }
}
```java
// LoggingProvider.java
package com.mycorp.nexus.logger.v2.internal;
~~~

~~~~
import com.mycorp.nexus.logger.v2.Logger;
import com.mycorp.nexus.logger.v2.config.NexusConfig;
import java.util.List;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;

public class LoggingProvider {
    private static volatile NexusConfig currentConfig = null;
    private final ConcurrentMap<String, LoggerImpl> loggers = new
ConcurrentHashMap<>();
    private final LogDispatcher dispatcher = new
LogDispatcher(List.of());

    public Logger getLogger(String name) {
        return loggers.computeIfAbsent(name, n -> new
LoggerImpl(n, dispatcher, currentConfig != null ?
currentConfig.getRootLevel() :
com.mycorp.nexus.logger.v2.config.LogLevel.INFO));
    }

    public LoggerControl getControl(String loggerName) {
        return new
LoggerControl(loggers.computeIfAbsent(loggerName,
this::getLogger));
    }

    public static void applyConfig(NexusConfig config) {
        currentConfig = config;
        // In real impl, would also update dispatcher and
existing loggers
    }
}
```java
// TemplateProcessor.java
package com.mycorp.nexus.logger.v2.internal;
~~~~

```
import com.mycorp.nexus.logger.v2.NexusContext;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.lang.StringTemplate;

public class TemplateProcessor implements
StringTemplate.Processor<LogEvent, RuntimeException> {
    public static final TemplateProcessor INSTANCE = new
TemplateProcessor();
    private static final Pattern KEY_PATTERN =
Pattern.compile("\\{(\\w+)\\}");

    @Override
    public LogEvent process(StringTemplate st) {
        String message = st.interpolate();
        Map<String, Object> context = new HashMap<>();
        List<String> fragments = st.fragments();
        List<Object> values = st.values();

        for (int i = 0; i < fragments.size() - 1; i++) {
            Matcher matcher = KEY_PATTERN.matcher(fragments.get(i
+ 1));
            if (matcher.find()) {
                String key = matcher.group(1);
                context.put(key, values.get(i));
            }
        }
        return new LogEvent(0, "", "", "", message, null,
context, getScopedContext());
    }

    public static Map<String, Object> getScopedContext() {
        Map<String, Object> scoped = new HashMap<>();
```

~~~
        if (NexusContext.TRACE_ID.isBound())
scoped.put("traceId", NexusContext.TRACE_ID.get());
        if (NexusContext.USER_ID.isBound()) scoped.put("userId",
NexusContext.USER_ID.get());
        return scoped;
    }
}
```java
// util/JsonUtil.java
package com.mycorp.nexus.logger.v2.util;

import com.mycorp.nexus.logger.v2.internal.LogEvent;

import java.io.PrintWriter;
import java.io.StringWriter;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Collectors;

public final class JsonUtil {
    private JsonUtil() {}

    public static String toJson(LogEvent event) {
        StringBuilder sb = new StringBuilder();
        sb.append("{\n");
        sb.append("  \"timestamp\": \"").append(new
java.util.Date(event.timestamp()).toInstant().toString()).append(
"\",\n");
        sb.append("  \"level\":
\"").append(event.level()).append("\",\n");
        sb.append("  \"logger\":
\"").append(event.loggerName()).append("\",\n");
        sb.append("  \"thread\":
\"").append(event.threadName()).append("\",\n");
        sb.append("  \"message\":
\"").append(escape(event.message())).append("\"");
~~~

```
        Map<String, Object> fullContext = new HashMap<>();
        if (event.scopedContext() != null)
fullContext.putAll(event.scopedContext());
        if (event.context() != null)
fullContext.putAll(event.context());

        if (!fullContext.isEmpty()) {
            sb.append(",\n");
            appendMap(sb, fullContext);
        }

        if (event.throwable() != null) {
            StringWriter sw = new StringWriter();
            event.throwable().printStackTrace(new
PrintWriter(sw));
            sb.append(",\n  \"stacktrace\":
\"").append(escape(sw.toString())).append("\"");
        }
        sb.append("\n}");
        return sb.toString();
    }

    private static void appendMap(StringBuilder sb, Map<String,
Object> map) {
        String mapJson = map.entrySet().stream()
            .map(entry -> "    \"" + escape(entry.getKey()) +
"\": " + formatValue(entry.getValue()))
            .collect(Collectors.joining(",\n"));
        sb.append(mapJson);
    }

    private static String formatValue(Object value) {
        if (value instanceof String) return "\"" +
escape((String) value) + "\"";
```

```
        if (value instanceof Number || value instanceof Boolean)
return value.toString();
        return "\"" + escape(value.toString()) + "\"";
    }

    private static String escape(String value) {
        if (value == null) return "";
        return value.replace("\\", "\\\\").replace("\"",
"\\\"").replace("\b", "\\b")
                    .replace("\f", "\\f").replace("\n",
"\\n").replace("\r", "\\r").replace("\t", "\\t");
    }
}
```