Enhanced RUM Backend Integration Guide

Overview

This guide will help you add Real User Monitoring (RUM) capabilities to your existing Spring Boot backend. We'll add:

- New JPA Entities for different event types
- DTOs for data transfer
- Repositories for database queries
- Service layer for business logic
- REST Controller for API endpoints
- Database configuration

Step 1: Update pom.xml

Add these dependencies to your `pom.xml`:

```
&lt;dependency&gt;
    &lt;groupId&gt;com.fasterxml.jackson.core&lt;/groupId&gt;
    &lt;artifactId&gt;jackson-databind&lt;/artifactId&gt;
&lt;/dependency&gt;


&lt;dependency&gt;
    &lt;groupId&gt;org.projectlombok&lt;/groupId&gt;
    &lt;artifactId&gt;lombok&lt;/artifactId&gt;
    &lt;optional&gt;true&lt;/optional&gt;
&lt;/dependency&gt;


&lt;dependency&gt;
    &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
    &lt;artifactId&gt;spring-boot-starter-data-jpa&lt;/artifactId&gt;
&lt;/dependency&gt;


&lt;dependency&gt;
    &lt;groupId&gt;com.h2database&lt;/groupId&gt;
    &lt;artifactId&gt;h2&lt;/artifactId&gt;
    &lt;scope&gt;runtime&lt;/scope&gt;
&lt;/dependency&gt;
```

```
&lt;dependency&gt;
    &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
    &lt;artifactId&gt;spring-boot-starter-logging&lt;/artifactId&gt;
&lt;/dependency&gt;
```

Step 2: Update application.properties

Add RUM configuration to your existing `application.properties`:

```
# ============ EXISTING CONFIGURATION ============
# (Keep your existing properties here)

# ============ ADD RUM CONFIGURATION ============

# H2 Database (Development)
spring.datasource.url=jdbc:h2:mem:rumdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

# H2 Console (for viewing data)
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

# JPA/Hibernate Configuration
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.jdbc.batch_size=20
spring.jpa.properties.hibernate.order_inserts=true
spring.jpa.properties.hibernate.order_updates=true

# RUM Service Logging
logging.level.com.rum=DEBUG
logging.level.org.springframework.web=INFO

# Jackson Configuration
spring.jackson.serialization.write-dates-as-timestamps=false
spring.jackson.default-property-inclusion=non_null
```

For Production (Oracle Database)

Create `application-oracle.properties`:

```
# Oracle Database
spring.datasource.url=jdbc:oracle:thin:@your-db-server:1521:ORCL
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver

# JPA/Hibernate for Oracle
spring.jpa.database-platform=org.hibernate.dialect.OracleDialect
spring.jpa.hibernate.ddl-auto=validate

# Connection Pooling
spring.datasource.hikari.maximum-pool-size=20
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.connection-timeout=30000
```

To use Oracle config: `java -jar app.jar --spring.profiles.active=oracle`

Step 3: Create Base Entity Class

Create `src/main/java/com/rum/model/BaseEntity.java`:

```java
package com.rum.model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import java.time.LocalDateTime;

@MappedSuperclass
@Getter
@Setter
public abstract class BaseEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, updatable = false)
    private LocalDateTime createdAt = LocalDateTime.now();

    @Column(nullable = false)
    private LocalDateTime updatedAt = LocalDateTime.now();

    @PreUpdate
    protected void onUpdate() {
        this.updatedAt = LocalDateTime.now();
    }
}
```

Step 4: Create Entity Classes

4.1 WebVitalEvent Entity

Create `src/main/java/com/rum/model/WebVitalEvent.java`:

```java
package com.rum.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;

@Entity
@Table(
    name = "web_vital_events",
    indexes = {
        @Index(name = "idx_wv_session", columnList = "sessionId"),
        @Index(name = "idx_wv_user", columnList = "userId"),
        @Index(name = "idx_wv_metric", columnList = "metricName"),
        @Index(name = "idx_wv_timestamp", columnList = "eventTimestamp")
    }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
public class WebVitalEvent extends BaseEntity {

    @Column(nullable = false, length = 50)
    private String sessionId;

    @Column(nullable = false, length = 50)
    private String userId;

    @Column(nullable = false, length = 500)
    private String pageUrl;

    @Column(nullable = false, length = 20)
    private String metricName;  // LCP, FCP, CLS, etc.

    @Column(nullable = false)
    private Double value;

    @Column(length = 20)
    private String rating;  // good, needs-improvement, poor

    @Column(length = 50)
    private String navigationType;

    @Column(length = 300)
    private String userAgent;

    @Column(nullable = false)
    private LocalDateTime eventTimestamp;
}
```

4.2 ErrorEvent Entity

Create `src/main/java/com/rum/model/ErrorEvent.java`:

```java
package com.rum.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;

@Entity
@Table(
    name = "error_events",
    indexes = {
        @Index(name = "idx_err_session", columnList = "sessionId"),
        @Index(name = "idx_err_user", columnList = "userId"),
        @Index(name = "idx_err_severity", columnList = "severity"),
        @Index(name = "idx_err_timestamp", columnList = "eventTimestamp")
    }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ErrorEvent extends BaseEntity {

    @Column(nullable = false, length = 50)
    private String sessionId;

    @Column(nullable = false, length = 50)
    private String userId;

    @Column(nullable = false, length = 500)
    private String pageUrl;

    @Column(nullable = false)
    private String message;

    @Column(length = 300)
    private String source;  // File name

    @Column
    private Integer lineno;

    @Column
    private Integer colno;

    @Column(columnDefinition = "TEXT")
    private String stack;  // Stack trace

    @Column(nullable = false, length = 30)
    private String errorType;  // javascript, unhandledRejection, network

    @Column(length = 20)
    private String severity;  // low, medium, high, critical
    @Column(length = 300)
    private String userAgent;

    @Column(nullable = false)
    private LocalDateTime eventTimestamp;
}
```

4.3 PageViewEvent Entity

Create `src/main/java/com/rum/model/PageViewEvent.java`:

```
package com.rum.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;

@Entity
@Table(
    name = "page_view_events",
    indexes = {
        @Index(name = "idx_pv_session", columnList = "sessionId"),
        @Index(name = "idx_pv_user", columnList = "userId"),
        @Index(name = "idx_pv_path", columnList = "pagePath"),
        @Index(name = "idx_pv_timestamp", columnList = "eventTimestamp")
    }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
public class PageViewEvent extends BaseEntity {

    @Column(nullable = false, length = 50)
    private String sessionId;

    @Column(nullable = false, length = 50)
    private String userId;

    @Column(nullable = false, length = 500)
    private String pageUrl;

    @Column(nullable = false, length = 500)
    private String pagePath;

    @Column(length = 200)
    private String pageTitle;

    @Column(length = 500)
    private String referrer;

    @Column(length = 500)
    private String previousPage;
    @Column(length = 300)
    private String userAgent;

    @Column(nullable = false)
    private LocalDateTime eventTimestamp;
}
```

4.4 PageSpeedEvent Entity

Create `src/main/java/com/rum/model/PageSpeedEvent.java`:

```
package com.rum.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;

@Entity
@Table(
    name = "page_speed_events",
    indexes = {
        @Index(name = "idx_ps_session", columnList = "sessionId"),
        @Index(name = "idx_ps_user", columnList = "userId"),
        @Index(name = "idx_ps_timestamp", columnList = "eventTimestamp")
    }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
public class PageSpeedEvent extends BaseEntity {

    @Column(nullable = false, length = 50)
    private String sessionId;

    @Column(nullable = false, length = 50)
    private String userId;

    @Column(nullable = false, length = 500)
    private String pageUrl;

    @Column(nullable = false)
    private Double loadTime;

    @Column(nullable = false)
    private Double domContentLoaded;

    @Column(nullable = false)
    private Double domInteractive;

    @Column(nullable = false)
    private Double resourceLoadTime;
    @Column
    private Double firstPaint;

    @Column(nullable = false)
    private LocalDateTime eventTimestamp;
}
```

4.5 EngagementEvent Entity

Create `src/main/java/com/rum/model/EngagementEvent.java`:

```
package com.rum.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;

@Entity
@Table(
    name = "engagement_events",
    indexes = {
        @Index(name = "idx_eng_session", columnList = "sessionId"),
        @Index(name = "idx_eng_user", columnList = "userId"),
        @Index(name = "idx_eng_timestamp", columnList = "eventTimestamp")
    }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
public class EngagementEvent extends BaseEntity {

    @Column(nullable = false, length = 50)
    private String sessionId;

    @Column(nullable = false, length = 50)
    private String userId;

    @Column(nullable = false, length = 500)
    private String pageUrl;

    @Column(nullable = false)
    private Long timeOnPage;  // milliseconds

    @Column(nullable = false)
    private Integer scrollDepth;  // 0-100%

    @Column(nullable = false)
    private Integer interactionCount;

    @Column(length = 20)
    private String exitType;  // navigation, close, refresh, timeout

    @Column(nullable = false)
    private LocalDateTime eventTimestamp;
}
```

4.6 NetworkErrorEvent Entity

Create `src/main/java/com/rum/model/NetworkErrorEvent.java`:

```
package com.rum.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;

@Entity
@Table(
    name = "network_error_events",
    indexes = {
        @Index(name = "idx_ne_session", columnList = "sessionId"),
        @Index(name = "idx_ne_user", columnList = "userId"),
        @Index(name = "idx_ne_error_type", columnList = "errorType")
    }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
public class NetworkErrorEvent extends BaseEntity {

    @Column(nullable = false, length = 50)
    private String sessionId;

    @Column(nullable = false, length = 50)
    private String userId;

    @Column(nullable = false, length = 500)
    private String pageUrl;

    @Column(nullable = false, length = 1000)
    private String url;

    @Column(nullable = false, length = 10)
    private String method;  // GET, POST, etc.

    @Column
    private Integer statusCode;

    @Column(nullable = false)
    private String message;

    @Column(nullable = false)
    private Double duration;  // milliseconds

    @Column(nullable = false, length = 20)
    private String errorType;  // timeout, failed, aborted
```

```java
    @Column(length = 300)
    private String userAgent;

    @Column(nullable = false)
    private LocalDateTime eventTimestamp;
}
```

4.7 ResourcePerformanceEvent Entity

Create `src/main/java/com/rum/model/ResourcePerformanceEvent.java`:

```java
package com.rum.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;

@Entity
@Table(
    name = "resource_performance_events",
    indexes = {
        @Index(name = "idx_rp_session", columnList = "sessionId"),
        @Index(name = "idx_rp_user", columnList = "userId"),
        @Index(name = "idx_rp_type", columnList = "resourceType")
    }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ResourcePerformanceEvent extends BaseEntity {

    @Column(nullable = false, length = 50)
    private String sessionId;

    @Column(nullable = false, length = 50)
    private String userId;

    @Column(nullable = false, length = 500)
    private String pageUrl;

    @Column(nullable = false, length = 1000)
    private String url;

    @Column(nullable = false, length = 30)
    private String resourceType;  // script, stylesheet, image, font, etc.

    @Column(nullable = false)
    private Double duration;

    @Column(nullable = false)
    private Long transferSize;
    @Column(nullable = false)
    private Long encodedBodySize;

    @Column(nullable = false)
    private Long decodedBodySize;

    @Column(nullable = false)
    private Boolean cacheHit;

    @Column(nullable = false)
    private LocalDateTime eventTimestamp;
}
```

4.8 UserActionEvent Entity

Create `src/main/java/com/rum/model/UserActionEvent.java`:

```
package com.rum.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;

@Entity
@Table(
    name = "user_action_events",
    indexes = {
        @Index(name = "idx_ua_session", columnList = "sessionId"),
        @Index(name = "idx_ua_user", columnList = "userId"),
        @Index(name = "idx_ua_type", columnList = "actionType")
    }
)
@Data
@NoArgsConstructor
@AllArgsConstructor
public class UserActionEvent extends BaseEntity {

    @Column(nullable = false, length = 50)
    private String sessionId;

    @Column(nullable = false, length = 50)
    private String userId;

    @Column(nullable = false, length = 500)
    private String pageUrl;

    @Column(nullable = false, length = 20)
    private String actionType;  // click, input, submit, rageClick, etc.

    @Column(nullable = false, length = 100)
    private String targetElement;

    @Column(length = 200)
    private String targetText;

    @Column(length = 100)
    private String targetId;

    @Column(length = 200)
    private String targetClass;

    @Column(columnDefinition = "TEXT")
    private String xPath;

    @Column(length = 500)
    private String value;

    @Column(nullable = false)
    private LocalDateTime eventTimestamp;
}
```

Step 5: Create Repository Interfaces

5.1 WebVitalEventRepository

Create `src/main/java/com/rum/repository/WebVitalEventRepository.java`:

```
package com.rum.repository;

import com.rum.model.WebVitalEvent;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import java.time.LocalDateTime;
import java.util.List;

@Repository
public interface WebVitalEventRepository extends JpaRepository&lt;WebVitalEvent, Long&gt;

    List&lt;WebVitalEvent&gt; findBySessionId(String sessionId);

    List&lt;WebVitalEvent&gt; findByMetricName(String metricName);

    @Query("SELECT w FROM WebVitalEvent w WHERE w.eventTimestamp BETWEEN :start AND :end'
    List&lt;WebVitalEvent&gt; findByTimeRange(
        @Param("start") LocalDateTime start,
        @Param("end") LocalDateTime end
    );

    @Query("SELECT w.metricName, AVG(w.value) as avgValue FROM WebVitalEvent w " +
           "WHERE w.eventTimestamp BETWEEN :start AND :end " +
           "GROUP BY w.metricName")
    List&lt;Object[]&gt; findAverageMetricsByTimeRange(
        @Param("start") LocalDateTime start,
        @Param("end") LocalDateTime end
    );
}
```

5.2 ErrorEventRepository

Create `src/main/java/com/rum/repository/ErrorEventRepository.java`:

```
package com.rum.repository;

import com.rum.model.ErrorEvent;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import java.time.LocalDateTime;
import java.util.List;

@Repository
public interface ErrorEventRepository extends JpaRepository&lt;ErrorEvent, Long&gt; {

    List&lt;ErrorEvent&gt; findBySessionId(String sessionId);

    List&lt;ErrorEvent&gt; findBySeverity(String severity);

    @Query("SELECT e FROM ErrorEvent e WHERE e.eventTimestamp BETWEEN :start AND :end")
    List&lt;ErrorEvent&gt; findByTimeRange(
        @Param("start") LocalDateTime start,
        @Param("end") LocalDateTime end
    );

    @Query("SELECT COUNT(e) FROM ErrorEvent e WHERE e.severity = :severity AND e.eventTim
    Long countBySeverityAndTimeRange(
        @Param("severity") String severity,
        @Param("start") LocalDateTime start,
        @Param("end") LocalDateTime end
    );
}
```

5.3 PageViewEventRepository

Create `src/main/java/com/rum/repository/PageViewEventRepository.java`:

```
package com.rum.repository;

import com.rum.model.PageViewEvent;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import java.time.LocalDateTime;
import java.util.List;

@Repository
```

```
public interface PageViewEventRepository extends JpaRepository&lt;PageViewEvent, Long&gt;

    List&lt;PageViewEvent&gt; findBySessionId(String sessionId);

    @Query("SELECT pv.pagePath, COUNT(pv) as count FROM PageViewEvent pv " +
           "WHERE pv.eventTimestamp BETWEEN :start AND :end " +
           "GROUP BY pv.pagePath ORDER BY count DESC")
    List&lt;Object[]&gt; findTopPagesByTimeRange(
        @Param("start") LocalDateTime start,
        @Param("end") LocalDateTime end
    );

    @Query("SELECT COUNT(DISTINCT pv.userId) FROM PageViewEvent pv " +
           "WHERE pv.eventTimestamp BETWEEN :start AND :end")
    Long countUniqueUsersByTimeRange(
        @Param("start") LocalDateTime start,
        @Param("end") LocalDateTime end
    );
}
```

5.4 PageSpeedEventRepository

Create `src/main/java/com/rum/repository/PageSpeedEventRepository.java`:

```
package com.rum.repository;

import com.rum.model.PageSpeedEvent;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import java.time.LocalDateTime;

@Repository
public interface PageSpeedEventRepository extends JpaRepository&lt;PageSpeedEvent, Long&g

    @Query("SELECT AVG(ps.loadTime) FROM PageSpeedEvent ps " +
           "WHERE ps.eventTimestamp BETWEEN :start AND :end")
    Double findAverageLoadTime(
        @Param("start") LocalDateTime start,
        @Param("end") LocalDateTime end
    );
}
```

5.5 EngagementEventRepository

Create `src/main/java/com/rum/repository/EngagementEventRepository.java`:

```
package com.rum.repository;

import com.rum.model.EngagementEvent;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import java.time.LocalDateTime;

@Repository
public interface EngagementEventRepository extends JpaRepository&lt;EngagementEvent, Long

    @Query("SELECT AVG(e.timeOnPage) FROM EngagementEvent e " +
           "WHERE e.eventTimestamp BETWEEN :start AND :end")
    Double findAverageTimeOnPage(
        @Param("start") LocalDateTime start,
        @Param("end") LocalDateTime end
    );

    @Query("SELECT AVG(e.scrollDepth) FROM EngagementEvent e " +
           "WHERE e.eventTimestamp BETWEEN :start AND :end")
    Double findAverageScrollDepth(
        @Param("start") LocalDateTime start,
        @Param("end") LocalDateTime end
    );
}
```

5.6 NetworkErrorEventRepository

Create `src/main/java/com/rum/repository/NetworkErrorEventRepository.java`:

```
package com.rum.repository;

import com.rum.model.NetworkErrorEvent;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import java.time.LocalDateTime;

@Repository
public interface NetworkErrorEventRepository extends JpaRepository&lt;NetworkErrorEvent,

    @Query("SELECT COUNT(ne) FROM NetworkErrorEvent ne WHERE ne.errorType = :errorType AN
    Long countByErrorTypeAndTimeRange(
        @Param("errorType") String errorType,
        @Param("start") LocalDateTime start,
        @Param("end") LocalDateTime end
    );
}
```

5.7 ResourcePerformanceEventRepository

Create `src/main/java/com/rum/repository/ResourcePerformanceEventRepository.java`:

```
package com.rum.repository;

import com.rum.model.ResourcePerformanceEvent;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ResourcePerformanceEventRepository extends JpaRepository&lt;ResourcePerf
}
```

5.8 UserActionEventRepository

Create `src/main/java/com/rum/repository/UserActionEventRepository.java`:

```
package com.rum.repository;

import com.rum.model.UserActionEvent;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserActionEventRepository extends JpaRepository&lt;UserActionEvent, Long
}
```

Step 6: Create DTOs (Data Transfer Objects)

6.1 WebVitalEventDTO

Create `src/main/java/com/rum/dto/WebVitalEventDTO.java`:

```
package com.rum.dto;

import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class WebVitalEventDTO {
    private String type;
    private Long timestamp;
    private String sessionId;
    private String userId;
    private String pageUrl;
    private String userAgent;

    @JsonProperty("data")
    private WebVitalData data;

    @Data
    @NoArgsConstructor
    @AllArgsConstructor
    public static class WebVitalData {
        private String name;
        private Double value;
        private String rating;
        private String navigationType;
    }
}
```

6.2 ErrorEventDTO

Create `src/main/java/com/rum/dto/ErrorEventDTO.java`:

```java
package com.rum.dto;

import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class ErrorEventDTO {
    private String type;
    private Long timestamp;
    private String sessionId;
    private String userId;
    private String pageUrl;
    private String userAgent;

    @JsonProperty("data")
    private ErrorData data;

    @Data
    @NoArgsConstructor
    @AllArgsConstructor
    public static class ErrorData {
        private String message;
        private String source;
        private Integer lineno;
        private Integer colno;
        private String stack;
        private String errorType;
        private String severity;
    }
}
```

6.3 PageViewEventDTO

Create `src/main/java/com/rum/dto/PageViewEventDTO.java`:

```
package com.rum.dto;

import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class PageViewEventDTO {
    private String type;
    private Long timestamp;
    private String sessionId;
    private String userId;
    private String pageUrl;
    private String userAgent;

    @JsonProperty("data")
    private PageViewData data;

    @Data
    @NoArgsConstructor
    @AllArgsConstructor
    public static class PageViewData {
        private String pagePath;
        private String pageTitle;
        private String referrer;
        private String previousPage;
    }
}
```

6.4 PageSpeedEventDTO

Create `src/main/java/com/rum/dto/PageSpeedEventDTO.java`:

```
package com.rum.dto;

import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class PageSpeedEventDTO {
    private String type;
    private Long timestamp;
    private String sessionId;
    private String userId;
    private String pageUrl;

    @JsonProperty("data")
    private PageSpeedData data;

    @Data
    @NoArgsConstructor
    @AllArgsConstructor
    public static class PageSpeedData {
        private Double loadTime;
        private Double domContentLoaded;
        private Double domInteractive;
        private Double resourceLoadTime;
        private Double firstPaint;
    }
}
```

6.5 EngagementEventDTO

Create `src/main/java/com/rum/dto/EngagementEventDTO.java`:

```
package com.rum.dto;

import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class EngagementEventDTO {
    private String type;
    private Long timestamp;
    private String sessionId;
    private String userId;
    private String pageUrl;

    @JsonProperty("data")
    private EngagementData data;

    @Data
    @NoArgsConstructor
    @AllArgsConstructor
    public static class EngagementData {
        private Long timeOnPage;
        private Integer scrollDepth;
        private Integer interactionCount;
        private String exitType;
    }
}
```

6.6 NetworkErrorEventDTO

Create `src/main/java/com/rum/dto/NetworkErrorEventDTO.java`:

```
package com.rum.dto;

import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class NetworkErrorEventDTO {
    private String type;
    private Long timestamp;
    private String sessionId;
    private String userId;
    private String pageUrl;
    private String userAgent;

    @JsonProperty("data")
    private NetworkErrorData data;

    @Data
    @NoArgsConstructor
    @AllArgsConstructor
    public static class NetworkErrorData {
        private String url;
        private String method;
        private Integer statusCode;
        private String message;
        private Double duration;
        private String errorType;
    }
}
```

6.7 ResourcePerformanceEventDTO

Create `src/main/java/com/rum/dto/ResourcePerformanceEventDTO.java`:

```
package com.rum.dto;

import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class ResourcePerformanceEventDTO {
    private String type;
    private Long timestamp;
    private String sessionId;
    private String userId;
    private String pageUrl;

    @JsonProperty("data")
    private ResourcePerformanceData data;

    @Data
    @NoArgsConstructor
    @AllArgsConstructor
    public static class ResourcePerformanceData {
        private String url;
        private String resourceType;
        private Double duration;
        private Long transferSize;
        private Long encodedBodySize;
        private Long decodedBodySize;
        private Boolean cacheHit;
    }
}
```

6.8 UserActionEventDTO

Create `src/main/java/com/rum/dto/UserActionEventDTO.java`:

```
package com.rum.dto;

import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class UserActionEventDTO {
    private String type;
    private Long timestamp;
    private String sessionId;
    private String userId;
    private String pageUrl;

    @JsonProperty("data")
    private UserActionData data;

    @Data
    @NoArgsConstructor
    @AllArgsConstructor
    public static class UserActionData {
        private String actionType;
        private String targetElement;
        private String targetText;
        private String targetId;
        private String targetClass;
        private String xPath;
        private String value;
    }
}
```

Step 7: Create Service Layer

Create `src/main/java/com/rum/service/RUMEventService.java`:

```
package com.rum.service;

import com.rum.dto.*;
import com.rum.model.*;
import com.rum.repository.*;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneId;

@Service
@AllArgsConstructor
@Slf4j
public class RUMEventService {

    private final WebVitalEventRepository webVitalRepository;
    private final ErrorEventRepository errorEventRepository;
    private final PageViewEventRepository pageViewRepository;
    private final PageSpeedEventRepository pageSpeedRepository;
    private final EngagementEventRepository engagementRepository;
    private final NetworkErrorEventRepository networkErrorRepository;
    private final ResourcePerformanceEventRepository resourceRepository;
    private final UserActionEventRepository userActionRepository;

    @Transactional
    public void processWebVital(WebVitalEventDTO dto) {
        try {
            WebVitalEvent entity = new WebVitalEvent();
            entity.setSessionId(dto.getSessionId());
            entity.setUserId(dto.getUserId());
            entity.setPageUrl(dto.getPageUrl());
            entity.setUserAgent(dto.getUserAgent());
            entity.setMetricName(dto.getData().getName());
            entity.setValue(dto.getData().getValue());
            entity.setRating(dto.getData().getRating());
            entity.setNavigationType(dto.getData().getNavigationType());
            entity.setEventTimestamp(convertTimestamp(dto.getTimestamp()));

            webVitalRepository.save(entity);
            log.debug("Saved web vital: {} = {}", dto.getData().getName(), dto.getData().
        } catch (Exception e) {
            log.error("Error processing web vital event", e);
```

```java
        }
    }

    @Transactional
    public void processError(ErrorEventDTO dto) {
        try {
            ErrorEvent entity = new ErrorEvent();
            entity.setSessionId(dto.getSessionId());
            entity.setUserId(dto.getUserId());
            entity.setPageUrl(dto.getPageUrl());
            entity.setUserAgent(dto.getUserAgent());
            entity.setMessage(dto.getData().getMessage());
            entity.setSource(dto.getData().getSource());
            entity.setLineno(dto.getData().getLineno());
            entity.setColno(dto.getData().getColno());
            entity.setStack(dto.getData().getStack());
            entity.setErrorType(dto.getData().getErrorType());
            entity.setSeverity(dto.getData().getSeverity());
            entity.setEventTimestamp(convertTimestamp(dto.getTimestamp()));

            errorEventRepository.save(entity);
            log.debug("Saved error event: {}", dto.getData().getMessage());
        } catch (Exception e) {
            log.error("Error processing error event", e);
        }
    }

    @Transactional
    public void processPageView(PageViewEventDTO dto) {
        try {
            PageViewEvent entity = new PageViewEvent();
            entity.setSessionId(dto.getSessionId());
            entity.setUserId(dto.getUserId());
            entity.setPageUrl(dto.getPageUrl());
            entity.setUserAgent(dto.getUserAgent());
            entity.setPagePath(dto.getData().getPagePath());
            entity.setPageTitle(dto.getData().getPageTitle());
            entity.setReferrer(dto.getData().getReferrer());
            entity.setPreviousPage(dto.getData().getPreviousPage());
            entity.setEventTimestamp(convertTimestamp(dto.getTimestamp()));

            pageViewRepository.save(entity);
            log.debug("Saved page view: {}", dto.getData().getPagePath());
        } catch (Exception e) {
            log.error("Error processing page view event", e);
        }
    }

    @Transactional
    public void processPageSpeed(PageSpeedEventDTO dto) {
        try {
            PageSpeedEvent entity = new PageSpeedEvent();
            entity.setSessionId(dto.getSessionId());
            entity.setUserId(dto.getUserId());
            entity.setPageUrl(dto.getPageUrl());
            entity.setLoadTime(dto.getData().getLoadTime());
            entity.setDomContentLoaded(dto.getData().getDomContentLoaded());
            entity.setDomInteractive(dto.getData().getDomInteractive());
            entity.setResourceLoadTime(dto.getData().getResourceLoadTime());
            entity.setFirstPaint(dto.getData().getFirstPaint());
            entity.setEventTimestamp(convertTimestamp(dto.getTimestamp()));

            pageSpeedRepository.save(entity);
            log.debug("Saved page speed: load={}ms", dto.getData().getLoadTime());
        } catch (Exception e) {
            log.error("Error processing page speed event", e);
        }
    }

    @Transactional
    public void processEngagement(EngagementEventDTO dto) {
        try {
            EngagementEvent entity = new EngagementEvent();
            entity.setSessionId(dto.getSessionId());
            entity.setUserId(dto.getUserId());
            entity.setPageUrl(dto.getPageUrl());
            entity.setTimeOnPage(dto.getData().getTimeOnPage());
            entity.setScrollDepth(dto.getData().getScrollDepth());
            entity.setInteractionCount(dto.getData().getInteractionCount());
            entity.setExitType(dto.getData().getExitType());
            entity.setEventTimestamp(convertTimestamp(dto.getTimestamp()));

            engagementRepository.save(entity);
            log.debug("Saved engagement: time={}ms", dto.getData().getTimeOnPage());
        } catch (Exception e) {
            log.error("Error processing engagement event", e);
        }
    }

    @Transactional
    public void processNetworkError(NetworkErrorEventDTO dto) {
        try {
            NetworkErrorEvent entity = new NetworkErrorEvent();
            entity.setSessionId(dto.getSessionId());
            entity.setUserId(dto.getUserId());
            entity.setPageUrl(dto.getPageUrl());
            entity.setUserAgent(dto.getUserAgent());
            entity.setUrl(dto.getData().getUrl());
            entity.setMethod(dto.getData().getMethod());
            entity.setStatusCode(dto.getData().getStatusCode());
            entity.setMessage(dto.getData().getMessage());
            entity.setDuration(dto.getData().getDuration());
            entity.setErrorType(dto.getData().getErrorType());
            entity.setEventTimestamp(convertTimestamp(dto.getTimestamp()));

            networkErrorRepository.save(entity);
            log.debug("Saved network error: {} {}", dto.getData().getMethod(), dto.getDat
        } catch (Exception e) {
            log.error("Error processing network error event", e);
        }
    }
    @Transactional
    public void processResourcePerformance(ResourcePerformanceEventDTO dto) {
        try {
            ResourcePerformanceEvent entity = new ResourcePerformanceEvent();
            entity.setSessionId(dto.getSessionId());
            entity.setUserId(dto.getUserId());
            entity.setPageUrl(dto.getPageUrl());
            entity.setUrl(dto.getData().getUrl());
            entity.setResourceType(dto.getData().getResourceType());
            entity.setDuration(dto.getData().getDuration());
            entity.setTransferSize(dto.getData().getTransferSize());
            entity.setEncodedBodySize(dto.getData().getEncodedBodySize());
            entity.setDecodedBodySize(dto.getData().getDecodedBodySize());
            entity.setCacheHit(dto.getData().getCacheHit());
            entity.setEventTimestamp(convertTimestamp(dto.getTimestamp()));

            resourceRepository.save(entity);
            log.debug("Saved resource performance: {}", dto.getData().getUrl());
        } catch (Exception e) {
            log.error("Error processing resource performance event", e);
        }
    }

    @Transactional
    public void processUserAction(UserActionEventDTO dto) {
        try {
            UserActionEvent entity = new UserActionEvent();
            entity.setSessionId(dto.getSessionId());
            entity.setUserId(dto.getUserId());
            entity.setPageUrl(dto.getPageUrl());
            entity.setActionType(dto.getData().getActionType());
            entity.setTargetElement(dto.getData().getTargetElement());
            entity.setTargetText(dto.getData().getTargetText());
            entity.setTargetId(dto.getData().getTargetId());
            entity.setTargetClass(dto.getData().getTargetClass());
            entity.setXPath(dto.getData().getXPath());
            entity.setValue(dto.getData().getValue());
            entity.setEventTimestamp(convertTimestamp(dto.getTimestamp()));

            userActionRepository.save(entity);
            log.debug("Saved user action: {}", dto.getData().getActionType());
        } catch (Exception e) {
            log.error("Error processing user action event", e);
        }
    }

    private LocalDateTime convertTimestamp(Long timestamp) {
        return LocalDateTime.ofInstant(
            Instant.ofEpochMilli(timestamp),
            ZoneId.systemDefault()
        );
    }
}
```

Step 8: Create REST Controller

Create `src/main/java/com/rum/controller/RUMEventController.java`:

```
package com.rum.controller;

import com.rum.dto.*;
import com.rum.service.RUMEventService;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/api/rum")
@AllArgsConstructor
@Slf4j
@CrossOrigin(origins = "*")
public class RUMEventController {

    private final RUMEventService rumEventService;
    private final ObjectMapper objectMapper;

    /**
     * Ingest batch of RUM events
     * POST /api/rum
     * Body: Array of event objects
     */
    @PostMapping
    public ResponseEntity&lt;Map&lt;String, Object&gt;&gt; ingestEvents(@RequestBody List
        try {
            log.info("Received batch of {} RUM events", events.size());

            if (events.isEmpty()) {
                return ResponseEntity.badRequest()
                    .body(Map.of("error", "Empty event batch"));
            }

            int processed = 0;

            for (JsonNode event : events) {
                String type = event.get("type").asText();

                switch (type) {
                    case "webVital":
                        WebVitalEventDTO vitalDto = objectMapper.treeToValue(event, WebVi
                        rumEventService.processWebVital(vitalDto);
                        processed++;
                        break;
                    case "error":
                        ErrorEventDTO errorDto = objectMapper.treeToValue(event, ErrorEve
                        rumEventService.processError(errorDto);
                        processed++;
                        break;
                    case "pageView":
                        PageViewEventDTO pageViewDto = objectMapper.treeToValue(event, Pa
                        rumEventService.processPageView(pageViewDto);
                        processed++;
                        break;
                    case "pageSpeed":
                        PageSpeedEventDTO pageSpeedDto = objectMapper.treeToValue(event,
                        rumEventService.processPageSpeed(pageSpeedDto);
                        processed++;
                        break;
                    case "engagement":
                        EngagementEventDTO engagementDto = objectMapper.treeToValue(event
                        rumEventService.processEngagement(engagementDto);
                        processed++;
                        break;
                    case "networkError":
                        NetworkErrorEventDTO networkErrorDto = objectMapper.treeToValue(e
                        rumEventService.processNetworkError(networkErrorDto);
                        processed++;
                        break;
                    case "resourcePerformance":
                        ResourcePerformanceEventDTO rpDto = objectMapper.treeToValue(ever
                        rumEventService.processResourcePerformance(rpDto);
                        processed++;
                        break;
                    case "userAction":
                        UserActionEventDTO uaDto = objectMapper.treeToValue(event, UserAc
                        rumEventService.processUserAction(uaDto);
                        processed++;
                        break;
                    default:
                        log.warn("Unknown event type: {}", type);
                }
            }

            return ResponseEntity.ok(Map.of(
                "status", "success",
                "message", "Processed " + processed + " events",
                "processed", processed,
                "failed", events.size() - processed
            ));
        } catch (Exception e) {
            log.error("Error ingesting events", e);
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body(Map.of("error", e.getMessage()));
        }
    }

    /**
     * Health check endpoint
     * GET /api/rum/health
     */
    @GetMapping("/health")
    public ResponseEntity&lt;Map&lt;String, String&gt;&gt; health() {
        return ResponseEntity.ok(Map.of(
            "status", "UP",
            "service", "RUM Backend",
            "timestamp", System.currentTimeMillis() + ""
        ));
    }
}
```

Step 9: Build and Test

Build the application:

```
mvn clean package
```

Run the application:

```
mvn spring-boot:run
```

Test the health endpoint:

```
curl http://localhost:8080/api/rum/health
```

Access H2 Console (if enabled):

```
http://localhost:8080/h2-console
```

JDBC URL: `jdbc:h2:mem:rumdb`
Username: `sa`
Password: (leave blank)

Step 10: Testing Event Ingestion

Test with a sample event batch:

```
curl -X POST http://localhost:8080/api/rum \
  -H "Content-Type: application/json" \
  -d '[
    {
      "type": "webVital",
      "timestamp": 1700000000000,
      "sessionId": "session-123",
      "userId": "user-456",
      "pageUrl": "http://example.com",
      "userAgent": "Mozilla/5.0...",
      "data": {
        "name": "LCP",
        "value": 2500,
        "rating": "good",
        "navigationType": "navigate"
      }
    },
    {
      "type": "pageView",
      "timestamp": 1700000000000,
      "sessionId": "session-123",
      "userId": "user-456",
      "pageUrl": "http://example.com",
      "userAgent": "Mozilla/5.0...",
      "data": {
        "pagePath": "/home",
        "pageTitle": "Home Page",
        "referrer": "http://google.com"
      }
    }
  ]'
```

Expected response:

```
{
  "status": "success",
  "message": "Processed 2 events",
  "processed": 2,
  "failed": 0
}
```

Summary

You now have a complete backend for RUM with:

✅ 8 Entity models for different event types
✅ 8 Repository interfaces for data access
✅ 8 DTOs for API communication
✅ Service layer for business logic
✅ REST controller for API endpoints
✅ Full support for H2 and Oracle databases
✅ Proper indexing for performance
✅ Error handling and logging

Your backend is ready to receive and store RUM events!