RUM Dashboard Update Guide - Latest Backend Integration

Overview

This guide shows you how to update your RUM dashboard to fetch and display the latest data from your backend, including all new metrics, improved charts, better data handling, and advanced features.

Part 1: Update Backend API Endpoints

First, ensure your backend has these endpoints. Add them to your `RUMStatsController.java`:

Step 1: Create Stats Controller

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

```
package com.rum.controller;

import com.rum.service.RUMStatsService;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.time.LocalDateTime;
import java.util.Map;

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

    private final RUMStatsService statsService;

    /**
     * Get web vitals statistics
     * GET /api/rum/stats/web-vitals?startMs=1000&amp;endMs=2000
     */
    @GetMapping("/web-vitals")
    public ResponseEntity&lt;Map&lt;String, Object&gt;&gt; getWebVitals(
        @RequestParam Long startMs,
        @RequestParam Long endMs
    ) {
        return ResponseEntity.ok(statsService.getWebVitalsStats(startMs, endMs));
    }
```

```
    /**
     * Get page views statistics
     * GET /api/rum/stats/page-views?startMs=1000&amp;endMs=2000
     */
    @GetMapping("/page-views")
    public ResponseEntity&lt;Map&lt;String, Object&gt;&gt; getPageViews(
        @RequestParam Long startMs,
        @RequestParam Long endMs
    ) {
        return ResponseEntity.ok(statsService.getPageViewsStats(startMs, endMs));
    }

    /**
     * Get error statistics
     * GET /api/rum/stats/errors?startMs=1000&amp;endMs=2000
     */
    @GetMapping("/errors")
    public ResponseEntity&lt;Map&lt;String, Object&gt;&gt; getErrors(
        @RequestParam Long startMs,
        @RequestParam Long endMs
    ) {
        return ResponseEntity.ok(statsService.getErrorStats(startMs, endMs));
    }

    /**
     * Get engagement statistics
     * GET /api/rum/stats/engagement?startMs=1000&amp;endMs=2000
     */
    @GetMapping("/engagement")
    public ResponseEntity&lt;Map&lt;String, Object&gt;&gt; getEngagement(
        @RequestParam Long startMs,
        @RequestParam Long endMs
    ) {
        return ResponseEntity.ok(statsService.getEngagementStats(startMs, endMs));
    }

    /**
     * Get top pages
     * GET /api/rum/stats/top-pages?startMs=1000&amp;endMs=2000&amp;limit=10
     */
    @GetMapping("/top-pages")
    public ResponseEntity&lt;Map&lt;String, Object&gt;&gt; getTopPages(
        @RequestParam Long startMs,
        @RequestParam Long endMs,
        @RequestParam(defaultValue = "10") Integer limit
    ) {
        return ResponseEntity.ok(statsService.getTopPages(startMs, endMs, limit));
    }

    /**
     * Get unique users count
     * GET /api/rum/stats/unique-users?startMs=1000&amp;endMs=2000
     */
    @GetMapping("/unique-users")
    public ResponseEntity&lt;Map&lt;String, Object&gt;&gt; getUniqueUsers(
        @RequestParam Long startMs,
```

```
        @RequestParam Long endMs
    ) {
        return ResponseEntity.ok(statsService.getUniqueUsersCount(startMs, endMs));
    }

    /**
     * Get page speed statistics
     * GET /api/rum/stats/page-speed?startMs=1000&amp;endMs=2000
     */
    @GetMapping("/page-speed")
    public ResponseEntity&lt;Map&lt;String, Object&gt;&gt; getPageSpeed(
        @RequestParam Long startMs,
        @RequestParam Long endMs
    ) {
        return ResponseEntity.ok(statsService.getPageSpeedStats(startMs, endMs));
    }

    /**
     * Get network errors
     * GET /api/rum/stats/network-errors?startMs=1000&amp;endMs=2000
     */
    @GetMapping("/network-errors")
    public ResponseEntity&lt;Map&lt;String, Object&gt;&gt; getNetworkErrors(
        @RequestParam Long startMs,
        @RequestParam Long endMs
    ) {
        return ResponseEntity.ok(statsService.getNetworkErrorStats(startMs, endMs));
    }

    /**
     * Get user sessions
     * GET /api/rum/stats/sessions?startMs=1000&amp;endMs=2000&amp;limit=20
     */
    @GetMapping("/sessions")
    public ResponseEntity&lt;Map&lt;String, Object&gt;&gt; getSessions(
        @RequestParam Long startMs,
        @RequestParam Long endMs,
        @RequestParam(defaultValue = "20") Integer limit
    ) {
        return ResponseEntity.ok(statsService.getSessionStats(startMs, endMs, limit));
    }

    /**
     * Get page performance comparison
     * GET /api/rum/stats/page-comparison?startMs=1000&amp;endMs=2000
     */
    @GetMapping("/page-comparison")
    public ResponseEntity&lt;Map&lt;String, Object&gt;&gt; getPageComparison(
        @RequestParam Long startMs,
        @RequestParam Long endMs
    ) {
        return ResponseEntity.ok(statsService.getPageComparison(startMs, endMs));
    }

    /**
     * Get real-time events stream
```

```
     * GET /api/rum/stats/live-events?limit=50
     */
    @GetMapping("/live-events")
    public ResponseEntity&lt;Map&lt;String, Object&gt;&gt; getLiveEvents(
        @RequestParam(defaultValue = "50") Integer limit
    ) {
        return ResponseEntity.ok(statsService.getLiveEvents(limit));
    }
}
```

Step 2: Create Stats Service

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

```
package com.rum.service;

import com.rum.repository.*;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;
import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.util.*;
import java.util.stream.Collectors;

@Service
@AllArgsConstructor
@Slf4j
public class RUMStatsService {

    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;

    // Helper method to convert timestamps
    private LocalDateTime convertTimestamp(Long ms) {
        return LocalDateTime.ofInstant(
            Instant.ofEpochMilli(ms),
            ZoneId.systemDefault()
        );
    }

    /**
     * Get web vitals statistics with averages and distribution
     */
    public Map&lt;String, Object&gt; getWebVitalsStats(Long startMs, Long endMs) {
        try {
            LocalDateTime start = convertTimestamp(startMs);
            LocalDateTime end = convertTimestamp(endMs);
```

```
            List&lt;com.rum.model.WebVitalEvent&gt; events = webVitalRepository.findByTim

            Map&lt;String, Object&gt; metrics = new HashMap&lt;&gt;();

            // Group by metric name
            Map&lt;String, List&lt;com.rum.model.WebVitalEvent&gt;&gt; grouped = events.s
                .collect(Collectors.groupingBy(com.rum.model.WebVitalEvent::getMetricName

            // Calculate stats for each metric
            Map&lt;String, Map&lt;String, Object&gt;&gt; metricStats = new HashMap&lt;&gt

            grouped.forEach((name, eventList) -&gt; {
                double average = eventList.stream()
                    .mapToDouble(com.rum.model.WebVitalEvent::getValue)
                    .average()
                    .orElse(0);

                double min = eventList.stream()
                    .mapToDouble(com.rum.model.WebVitalEvent::getValue)
                    .min()
                    .orElse(0);

                double max = eventList.stream()
                    .mapToDouble(com.rum.model.WebVitalEvent::getValue)
                    .max()
                    .orElse(0);

                // Calculate distribution by rating
                Map&lt;String, Long&gt; ratingDist = eventList.stream()
                    .collect(Collectors.groupingBy(
                        com.rum.model.WebVitalEvent::getRating,
                        Collectors.counting()
                    ));

                Map&lt;String, Object&gt; stats = new HashMap&lt;&gt;();
                stats.put("average", average);
                stats.put("min", min);
                stats.put("max", max);
                stats.put("count", eventList.size());
                stats.put("rating", getOverallRating(average, name));
                stats.put("distribution", ratingDist);

                metricStats.put(name, stats);
            });

            metrics.put("metrics", metricStats);
            metrics.put("timestamp", System.currentTimeMillis());

            return metrics;
        } catch (Exception e) {
            log.error("Error getting web vitals stats", e);
            return Map.of("error", e.getMessage());
        }
    }
```

```
    /**
     * Get page views statistics with timeline
     */
    public Map&lt;String, Object&gt; getPageViewsStats(Long startMs, Long endMs) {
        try {
            LocalDateTime start = convertTimestamp(startMs);
            LocalDateTime end = convertTimestamp(endMs);

            List&lt;com.rum.model.PageViewEvent&gt; events = pageViewRepository.findByTim

            Map&lt;String, Object&gt; stats = new HashMap&lt;&gt;();
            stats.put("count", events.size());
            stats.put("uniqueUsers", events.stream()
                .map(com.rum.model.PageViewEvent::getUserId)
                .distinct()
                .count());

            // Timeline data (hourly aggregation)
            Map&lt;String, Long&gt; timeline = events.stream()
                .collect(Collectors.groupingBy(
                    e -&gt; new java.text.SimpleDateFormat("yyyy-MM-dd HH:00").format(
                        java.sql.Timestamp.valueOf(e.getEventTimestamp())
                    ),
                    Collectors.counting()
                ));

            stats.put("timeline", timeline);
            stats.put("timestamp", System.currentTimeMillis());

            return stats;
        } catch (Exception e) {
            log.error("Error getting page views stats", e);
            return Map.of("error", e.getMessage());
        }
    }

    /**
     * Get error statistics with severity breakdown
     */
    public Map&lt;String, Object&gt; getErrorStats(Long startMs, Long endMs) {
        try {
            LocalDateTime start = convertTimestamp(startMs);
            LocalDateTime end = convertTimestamp(endMs);

            List&lt;com.rum.model.ErrorEvent&gt; errors = errorEventRepository.findByTime

            Map&lt;String, Object&gt; stats = new HashMap&lt;&gt;();
            stats.put("totalErrors", errors.size());

            // Count by severity
            Map&lt;String, Long&gt; severityCount = errors.stream()
                .collect(Collectors.groupingBy(
                    e -&gt; e.getSeverity() != null ? e.getSeverity() : "unknown",
                    Collectors.counting()
                ));
```

```
            stats.put("bySeverity", severityCount);

            // Error rate percentage
            double errorRate = errors.isEmpty() ? 0 :
                (severityCount.getOrDefault("critical", 0L) + severityCount.getOrDefault(

            stats.put("errorRate", errorRate);

            // Top errors
            Map&lt;String, Long&gt; topErrors = errors.stream()
                .collect(Collectors.groupingBy(
                    com.rum.model.ErrorEvent::getMessage,
                    Collectors.counting()
                ))
                .entrySet().stream()
                .sorted((a, b) -&gt; b.getValue().compareTo(a.getValue()))
                .limit(10)
                .collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue));

            stats.put("topErrors", topErrors);
            stats.put("timestamp", System.currentTimeMillis());

            return stats;
        } catch (Exception e) {
            log.error("Error getting error stats", e);
            return Map.of("error", e.getMessage());
        }
    }

    /**
     * Get engagement statistics
     */
    public Map&lt;String, Object&gt; getEngagementStats(Long startMs, Long endMs) {
        try {
            LocalDateTime start = convertTimestamp(startMs);
            LocalDateTime end = convertTimestamp(endMs);

            List&lt;com.rum.model.EngagementEvent&gt; events = engagementRepository.findB

            Map&lt;String, Object&gt; stats = new HashMap&lt;&gt;();

            double avgTimeOnPage = events.stream()
                .mapToLong(com.rum.model.EngagementEvent::getTimeOnPage)
                .average()
                .orElse(0);

            double avgScrollDepth = events.stream()
                .mapToInt(com.rum.model.EngagementEvent::getScrollDepth)
                .average()
                .orElse(0);

            stats.put("averageTimeOnPage", avgTimeOnPage);
            stats.put("averageScrollDepth", avgScrollDepth);
            stats.put("totalSessions", events.size());

            // Exit type distribution
```

```
            Map&lt;String, Long&gt; exitTypes = events.stream()
                .collect(Collectors.groupingBy(
                    com.rum.model.EngagementEvent::getExitType,
                    Collectors.counting()
                ));

            stats.put("exitTypeDistribution", exitTypes);
            stats.put("timestamp", System.currentTimeMillis());

            return stats;
        } catch (Exception e) {
            log.error("Error getting engagement stats", e);
            return Map.of("error", e.getMessage());
        }
    }

    /**
     * Get top pages with metrics
     */
    public Map&lt;String, Object&gt; getTopPages(Long startMs, Long endMs, Integer limit)
        try {
            LocalDateTime start = convertTimestamp(startMs);
            LocalDateTime end = convertTimestamp(endMs);

            List&lt;com.rum.model.PageViewEvent&gt; pageViews = pageViewRepository.findBy

            // Group by page path and collect stats
            Map&lt;String, List&lt;com.rum.model.PageViewEvent&gt;&gt; grouped = pageView
                .collect(Collectors.groupingBy(com.rum.model.PageViewEvent::getPagePath))

            List&lt;Map&lt;String, Object&gt;&gt; topPages = grouped.entrySet().stream()
                .map(entry -&gt; {
                    Map&lt;String, Object&gt; page = new HashMap&lt;&gt;();
                    page.put("path", entry.getKey());
                    page.put("views", entry.getValue().size());
                    page.put("uniqueUsers", entry.getValue().stream()
                        .map(com.rum.model.PageViewEvent::getUserId)
                        .distinct()
                        .count());
                    return page;
                })
                .sorted((a, b) -&gt; ((Integer) b.get("views")).compareTo((Integer) a.get
                .limit(limit)
                .collect(Collectors.toList());

            Map&lt;String, Object&gt; result = new HashMap&lt;&gt;();
            result.put("pages", topPages);
            result.put("timestamp", System.currentTimeMillis());

            return result;
        } catch (Exception e) {
            log.error("Error getting top pages", e);
            return Map.of("error", e.getMessage());
        }
    }
```

```
    /**
     * Get unique users count
     */
    public Map&lt;String, Object&gt; getUniqueUsersCount(Long startMs, Long endMs) {
        try {
            LocalDateTime start = convertTimestamp(startMs);
            LocalDateTime end = convertTimestamp(endMs);

            List&lt;com.rum.model.PageViewEvent&gt; pageViews = pageViewRepository.findBy

            long uniqueUsers = pageViews.stream()
                .map(com.rum.model.PageViewEvent::getUserId)
                .distinct()
                .count();

            Map&lt;String, Object&gt; result = new HashMap&lt;&gt;();
            result.put("count", uniqueUsers);
            result.put("timestamp", System.currentTimeMillis());

            return result;
        } catch (Exception e) {
            log.error("Error getting unique users", e);
            return Map.of("error", e.getMessage());
        }
    }

    /**
     * Get page speed statistics
     */
    public Map&lt;String, Object&gt; getPageSpeedStats(Long startMs, Long endMs) {
        try {
            LocalDateTime start = convertTimestamp(startMs);
            LocalDateTime end = convertTimestamp(endMs);

            List&lt;com.rum.model.PageSpeedEvent&gt; speeds = pageSpeedRepository.findByT

            Map&lt;String, Object&gt; stats = new HashMap&lt;&gt;();

            double avgLoadTime = speeds.stream()
                .mapToDouble(com.rum.model.PageSpeedEvent::getLoadTime)
                .average()
                .orElse(0);

            stats.put("averageLoadTime", avgLoadTime);
            stats.put("count", speeds.size());
            stats.put("timestamp", System.currentTimeMillis());

            return stats;
        } catch (Exception e) {
            log.error("Error getting page speed stats", e);
            return Map.of("error", e.getMessage());
        }
    }

    /**
     * Get network errors statistics
```

```
     */
    public Map&lt;String, Object&gt; getNetworkErrorStats(Long startMs, Long endMs) {
        try {
            LocalDateTime start = convertTimestamp(startMs);
            LocalDateTime end = convertTimestamp(endMs);

            List&lt;com.rum.model.NetworkErrorEvent&gt; errors = networkErrorRepository.f

            Map&lt;String, Object&gt; stats = new HashMap&lt;&gt;();
            stats.put("totalErrors", errors.size());

            // Count by error type
            Map&lt;String, Long&gt; byType = errors.stream()
                .collect(Collectors.groupingBy(
                    com.rum.model.NetworkErrorEvent::getErrorType,
                    Collectors.counting()
                ));

            stats.put("byErrorType", byType);
            stats.put("timestamp", System.currentTimeMillis());

            return stats;
        } catch (Exception e) {
            log.error("Error getting network error stats", e);
            return Map.of("error", e.getMessage());
        }
    }

    /**
     * Get session statistics
     */
    public Map&lt;String, Object&gt; getSessionStats(Long startMs, Long endMs, Integer li
        try {
            LocalDateTime start = convertTimestamp(startMs);
            LocalDateTime end = convertTimestamp(endMs);

            // Get page views grouped by session
            List&lt;com.rum.model.PageViewEvent&gt; pageViews = pageViewRepository.findBy

            Map&lt;String, List&lt;com.rum.model.PageViewEvent&gt;&gt; sessions = pageVie
                .collect(Collectors.groupingBy(com.rum.model.PageViewEvent::getSessionId)

            List&lt;Map&lt;String, Object&gt;&gt; sessionList = sessions.entrySet().strea
                .map(entry -&gt; {
                    Map&lt;String, Object&gt; session = new HashMap&lt;&gt;();
                    session.put("sessionId", entry.getKey());
                    session.put("pageCount", entry.getValue().size());
                    session.put("userId", entry.getValue().get(0).getUserId());
                    return session;
                })
                .sorted((a, b) -&gt; ((Integer) b.get("pageCount")).compareTo((Integer) a
                .limit(limit)
                .collect(Collectors.toList());

            Map&lt;String, Object&gt; result = new HashMap&lt;&gt;();
            result.put("sessions", sessionList);
```

```
            result.put("timestamp", System.currentTimeMillis());

            return result;
        } catch (Exception e) {
            log.error("Error getting session stats", e);
            return Map.of("error", e.getMessage());
        }
    }

    /**
     * Get page comparison metrics
     */
    public Map&lt;String, Object&gt; getPageComparison(Long startMs, Long endMs) {
        try {
            LocalDateTime start = convertTimestamp(startMs);
            LocalDateTime end = convertTimestamp(endMs);

            // This would compare performance across pages
            // Implementation depends on your specific needs

            Map&lt;String, Object&gt; result = new HashMap&lt;&gt;();
            result.put("comparison", new HashMap&lt;&gt;());
            result.put("timestamp", System.currentTimeMillis());

            return result;
        } catch (Exception e) {
            log.error("Error getting page comparison", e);
            return Map.of("error", e.getMessage());
        }
    }

    /**
     * Get live events (latest events)
     */
    public Map&lt;String, Object&gt; getLiveEvents(Integer limit) {
        try {
            List&lt;com.rum.model.PageViewEvent&gt; recent = pageViewRepository.findAll()

            List&lt;Map&lt;String, Object&gt;&gt; events = recent.stream()
                .sorted((a, b) -&gt; b.getEventTimestamp().compareTo(a.getEventTimestamp(
                .limit(limit)
                .map(e -&gt; {
                    Map&lt;String, Object&gt; event = new HashMap&lt;&gt;();
                    event.put("type", "pageView");
                    event.put("timestamp", e.getEventTimestamp());
                    event.put("userId", e.getUserId());
                    event.put("page", e.getPagePath());
                    return event;
                })
                .collect(Collectors.toList());

            Map&lt;String, Object&gt; result = new HashMap&lt;&gt;();
            result.put("events", events);
            result.put("timestamp", System.currentTimeMillis());

            return result;
```

```
        } catch (Exception e) {
            log.error("Error getting live events", e);
            return Map.of("error", e.getMessage());
        }
    }

    // Helper method to determine overall rating
    private String getOverallRating(double value, String metricName) {
        Map&lt;String, Map&lt;String, Double&gt;&gt; thresholds = Map.of(
            "LCP", Map.of("good", 2500.0, "poor", 4000.0),
            "FCP", Map.of("good", 1800.0, "poor", 3000.0),
            "CLS", Map.of("good", 0.1, "poor", 0.25),
            "INP", Map.of("good", 200.0, "poor", 500.0),
            "TTFB", Map.of("good", 800.0, "poor", 1800.0)
        );

        Map&lt;String, Double&gt; threshold = thresholds.get(metricName);
        if (threshold == null) return "good";

        if (value &lt;= threshold.get("good")) return "good";
        if (value &lt;= threshold.get("poor")) return "needs-improvement";
        return "poor";
    }
}
```

Step 3: Add Missing Repository Methods

Update `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 FROM PageViewEvent pv WHERE pv.eventTimestamp BETWEEN :start AND :e
    List&lt;PageViewEvent&gt; findByTimeRange(
        @Param("start") LocalDateTime start,
        @Param("end") LocalDateTime end
    );

    @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
    );
}
```

Update `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;
import java.util.List;

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

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

    @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
    );
}
```

Update `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;
import java.util.List;

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

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

    @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
    );
}
```

Update `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;
import java.util.List;

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

    @Query("SELECT ne FROM NetworkErrorEvent ne WHERE ne.eventTimestamp BETWEEN :start AN
    List&lt;NetworkErrorEvent&gt; findByTimeRange(
        @Param("start") LocalDateTime start,
        @Param("end") LocalDateTime end
    );

    @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
    );
}
```

Part 2: Update Frontend Dashboard

Step 1: Update API Service

Update `src/services/api.js`:

```
import axios from 'axios';

const API_BASE_URL = process.env.REACT_APP_RUM_BACKEND_URL || 'http://localhost:8080/api/

const api = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});

// Helper to convert timestamps
const getTimeRange = (range = '24h') =&gt; {
  const now = Date.now();
  const ranges = {
    '1h': 60 * 60 * 1000,
    '24h': 24 * 60 * 60 * 1000,
    '7d': 7 * 24 * 60 * 60 * 1000,
    '30d': 30 * 24 * 60 * 60 * 1000,
  };

  return {
    startMs: now - ranges[range],
    endMs: now,
  };
};

// API endpoints
export const rumAPI = {
  // Health check
  health: () =&gt; api.get('/health'),

  // Get web vitals
  getWebVitals: async (timeRange = '24h') =&gt; {
    const { startMs, endMs } = getTimeRange(timeRange);
    try {
      const response = await api.get('/stats/web-vitals', {
        params: { startMs, endMs },
      });
      return response.data;
    } catch (error) {
      console.error('Error fetching web vitals:', error);
      throw error;
    }
  },

  // Get page views
  getPageViews: async (timeRange = '24h') =&gt; {
    const { startMs, endMs } = getTimeRange(timeRange);
```

```
    try {
      const response = await api.get('/stats/page-views', {
        params: { startMs, endMs },
      });
      return response.data;
    } catch (error) {
      console.error('Error fetching page views:', error);
      throw error;
    }
  },

  // Get errors
  getErrors: async (timeRange = '24h') =&gt; {
    const { startMs, endMs } = getTimeRange(timeRange);
    try {
      const response = await api.get('/stats/errors', {
        params: { startMs, endMs },
      });
      return response.data;
    } catch (error) {
      console.error('Error fetching errors:', error);
      throw error;
    }
  },

  // Get engagement metrics
  getEngagement: async (timeRange = '24h') =&gt; {
    const { startMs, endMs } = getTimeRange(timeRange);
    try {
      const response = await api.get('/stats/engagement', {
        params: { startMs, endMs },
      });
      return response.data;
    } catch (error) {
      console.error('Error fetching engagement:', error);
      throw error;
    }
  },

  // Get top pages
  getTopPages: async (timeRange = '24h', limit = 10) =&gt; {
    const { startMs, endMs } = getTimeRange(timeRange);
    try {
      const response = await api.get('/stats/top-pages', {
        params: { startMs, endMs, limit },
      });
      return response.data;
    } catch (error) {
      console.error('Error fetching top pages:', error);
      throw error;
    }
  },

  // Get unique users
  getUniqueUsers: async (timeRange = '24h') =&gt; {
    const { startMs, endMs } = getTimeRange(timeRange);
```

```
  try {
    const response = await api.get('/stats/unique-users', {
      params: { startMs, endMs },
    });
    return response.data;
  } catch (error) {
    console.error('Error fetching unique users:', error);
    throw error;
  }
},

// Get page speed
getPageSpeed: async (timeRange = '24h') =&gt; {
  const { startMs, endMs } = getTimeRange(timeRange);
  try {
    const response = await api.get('/stats/page-speed', {
      params: { startMs, endMs },
    });
    return response.data;
  } catch (error) {
    console.error('Error fetching page speed:', error);
    throw error;
  }
},

// Get network errors
getNetworkErrors: async (timeRange = '24h') =&gt; {
  const { startMs, endMs } = getTimeRange(timeRange);
  try {
    const response = await api.get('/stats/network-errors', {
      params: { startMs, endMs },
    });
    return response.data;
  } catch (error) {
    console.error('Error fetching network errors:', error);
    throw error;
  }
},

// Get sessions (NEW)
getSessions: async (timeRange = '24h', limit = 20) =&gt; {
  const { startMs, endMs } = getTimeRange(timeRange);
  try {
    const response = await api.get('/stats/sessions', {
      params: { startMs, endMs, limit },
    });
    return response.data;
  } catch (error) {
    console.error('Error fetching sessions:', error);
    throw error;
  }
},

// Get page comparison (NEW)
getPageComparison: async (timeRange = '24h') =&gt; {
  const { startMs, endMs } = getTimeRange(timeRange);
```

```
    try {
      const response = await api.get('/stats/page-comparison', {
        params: { startMs, endMs },
      });
      return response.data;
    } catch (error) {
      console.error('Error fetching page comparison:', error);
      throw error;
    }
  },

  // Get live events (NEW)
  getLiveEvents: async (limit = 50) =&gt; {
    try {
      const response = await api.get('/stats/live-events', {
        params: { limit },
      });
      return response.data;
    } catch (error) {
      console.error('Error fetching live events:', error);
      throw error;
    }
  },
};

export default api;
```

Step 2: Update Hooks

Update `src/hooks/useRUMData.js`:

```
import { useQuery } from '@tanstack/react-query';
import { rumAPI } from '../services/api';

export const useRUMData = (timeRange = '24h') =&gt; {
  const webVitals = useQuery({
    queryKey: ['webVitals', timeRange],
    queryFn: () =&gt; rumAPI.getWebVitals(timeRange),
    refetchInterval: 30000,
    staleTime: 10000,
    retry: 2,
  });

  const pageViews = useQuery({
    queryKey: ['pageViews', timeRange],
    queryFn: () =&gt; rumAPI.getPageViews(timeRange),
    refetchInterval: 30000,
    staleTime: 10000,
    retry: 2,
  });

  const errors = useQuery({
    queryKey: ['errors', timeRange],
    queryFn: () =&gt; rumAPI.getErrors(timeRange),
    refetchInterval: 30000,
```

```
    staleTime: 10000,
    retry: 2,
  });

  const engagement = useQuery({
    queryKey: ['engagement', timeRange],
    queryFn: () =&gt; rumAPI.getEngagement(timeRange),
    refetchInterval: 30000,
    staleTime: 10000,
    retry: 2,
  });

  const topPages = useQuery({
    queryKey: ['topPages', timeRange],
    queryFn: () =&gt; rumAPI.getTopPages(timeRange),
    refetchInterval: 30000,
    staleTime: 10000,
    retry: 2,
  });

  const uniqueUsers = useQuery({
    queryKey: ['uniqueUsers', timeRange],
    queryFn: () =&gt; rumAPI.getUniqueUsers(timeRange),
    refetchInterval: 30000,
    staleTime: 10000,
    retry: 2,
  });

  const pageSpeed = useQuery({
    queryKey: ['pageSpeed', timeRange],
    queryFn: () =&gt; rumAPI.getPageSpeed(timeRange),
    refetchInterval: 30000,
    staleTime: 10000,
    retry: 2,
  });

  const networkErrors = useQuery({
    queryKey: ['networkErrors', timeRange],
    queryFn: () =&gt; rumAPI.getNetworkErrors(timeRange),
    refetchInterval: 30000,
    staleTime: 10000,
    retry: 2,
  });

  // NEW: Sessions
  const sessions = useQuery({
    queryKey: ['sessions', timeRange],
    queryFn: () =&gt; rumAPI.getSessions(timeRange),
    refetchInterval: 30000,
    staleTime: 10000,
    retry: 2,
  });

  // NEW: Live events
  const liveEvents = useQuery({
    queryKey: ['liveEvents'],
```

```
    queryFn: () =&gt; rumAPI.getLiveEvents(),
    refetchInterval: 10000, // More frequent updates
    staleTime: 5000,
    retry: 2,
  });

  const isLoading =
    webVitals.isLoading ||
    pageViews.isLoading ||
    errors.isLoading ||
    liveEvents.isLoading;

  const isError =
    webVitals.isError ||
    pageViews.isError ||
    errors.isError;

  return {
    webVitals,
    pageViews,
    errors,
    engagement,
    topPages,
    uniqueUsers,
    pageSpeed,
    networkErrors,
    sessions,
    liveEvents,
    isLoading,
    isError,
    refetch: async () =&gt; {
      await Promise.all([
        webVitals.refetch(),
        pageViews.refetch(),
        errors.refetch(),
        engagement.refetch(),
        topPages.refetch(),
        uniqueUsers.refetch(),
        pageSpeed.refetch(),
        networkErrors.refetch(),
        sessions.refetch(),
        liveEvents.refetch(),
      ]);
    },
  };
};
```

Step 3: Create New Components

Create `src/components/tables/SessionsTable.jsx`:

```
import React from 'react';
import './Table.css';

const SessionsTable = ({ data }) =&gt; {
```

```
  if (!data || !data.sessions || data.sessions.length === 0) {
    return <div>No session data available</div>;
  }

  return (
    <div>
      <h2>Active Sessions</h2>
      <div>
        &lt;table className="data-table"&gt;
          &lt;thead&gt;
            &lt;tr&gt;
              &lt;th&gt;Session ID&lt;/th&gt;
              &lt;th&gt;User ID&lt;/th&gt;
              &lt;th&gt;Pages Viewed&lt;/th&gt;
              &lt;th&gt;Duration&lt;/th&gt;
            &lt;/tr&gt;
          &lt;/thead&gt;
          &lt;tbody&gt;
            {data.sessions.map((session, index) =&gt; (
              &lt;tr key={index}&gt;
                &lt;td className="session-id"&gt;{session.sessionId.substring(0, 12)}...&
                &lt;td className="user-id"&gt;{session.userId.substring(0, 12)}...&lt;/td
                &lt;td&gt;{session.pageCount}&lt;/td&gt;
                &lt;td&gt;-&lt;/td&gt;
              &lt;/tr&gt;
            ))}
          &lt;/tbody&gt;
        &lt;/table&gt;
      </div>
    </div>
  );
};

export default SessionsTable;
```

Create `src/components/tables/LiveEventsTable.jsx`:

```
import React from 'react';
import './Table.css';

const LiveEventsTable = ({ data }) =&gt; {
  if (!data || !data.events || data.events.length === 0) {
    return <div>No live events available</div>;
  }

  return (
    <div>
      <h2>Live Events (Last 50)</h2>
      <div>
        &lt;table className="data-table"&gt;
          &lt;thead&gt;
            &lt;tr&gt;
              &lt;th&gt;Time&lt;/th&gt;
              &lt;th&gt;Event Type&lt;/th&gt;
              &lt;th&gt;User ID&lt;/th&gt;
```

```
              &lt;th&gt;Page&lt;/th&gt;
            &lt;/tr&gt;
          &lt;/thead&gt;
          &lt;tbody&gt;
            {data.events.map((event, index) =&gt; (
              &lt;tr key={index}&gt;
                &lt;td&gt;{new Date(event.timestamp).toLocaleTimeString()}&lt;/td&gt;
                &lt;td&gt;<span>{event.type}</span>&lt;/td&gt;
                &lt;td className="user-id"&gt;{event.userId.substring(0, 12)}...&lt;/td&g
                &lt;td className="path-cell"&gt;{event.page}&lt;/td&gt;
              &lt;/tr&gt;
            ))}
          &lt;/tbody&gt;
        &lt;/table&gt;
      </div>
    </div>
  );
};

export default LiveEventsTable;
```

Add to `src/components/tables/Table.css`:

```
.session-id,
.user-id {
  font-family: 'Courier New', monospace;
  color: #6366f1;
  font-size: 12px;
}

.badge {
  display: inline-block;
  padding: 4px 12px;
  border-radius: 12px;
  font-size: 11px;
  font-weight: 600;
  background: #dbeafe;
  color: #0369a1;
  text-transform: uppercase;
}
```

Step 4: Update Main App

Update `src/App.jsx`:

```
import React, { useState } from 'react';
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
import { useRUMData } from './hooks/useRUMData';
import Header from './components/layout/Header';
import TimeRangeFilter from './components/filters/TimeRangeFilter';
import MetricGrid from './components/metrics/MetricGrid';
import WebVitalsChart from './components/charts/WebVitalsChart';
import PageViewsChart from './components/charts/PageViewsChart';
```

```
import TopPagesTable from './components/tables/TopPagesTable';
import ErrorsTable from './components/tables/ErrorsTable';
import SessionsTable from './components/tables/SessionsTable';
import LiveEventsTable from './components/tables/LiveEventsTable';
import './App.css';

const queryClient = new QueryClient();

const DashboardContent = () =&gt; {
  const [timeRange, setTimeRange] = useState('24h');
  const data = useRUMData(timeRange);

  if (data.isLoading) {
    return (
      <div>
        <div></div>
        <p>Loading dashboard data...</p>
      </div>
    );
  }

  if (data.isError) {
    return (
      <div>
        <p>✖ Error loading dashboard data</p>
        <p>Please check if the backend is running at {process.env.REACT_APP_RUM_BACKEND_U
      </div>
    );
  }

  return (
    <div>
      &lt;Header timeRange={timeRange} onTimeRangeChange={setTimeRange} /&gt;

      &lt;main className="dashboard-main"&gt;
        <div>
          {/* Time Range Filter */}
          <div>
            &lt;TimeRangeFilter value={timeRange} onChange={setTimeRange} /&gt;
          </div>

          {/* Metric Grid */}
          &lt;MetricGrid data={data} /&gt;

          {/* Charts Row */}
          <div>
            <div>
              &lt;WebVitalsChart data={data.webVitals?.data} /&gt;
            </div>
            <div>
              &lt;PageViewsChart data={data.pageViews?.data} /&gt;
            </div>
          </div>

          {/* Tables Row 1 */}
          <div>
```

```
            &lt;TopPagesTable data={data.topPages?.data} /&gt;
            &lt;ErrorsTable data={data.errors?.data} /&gt;
          </div>

          {/* Tables Row 2 - NEW */}
          <div>
            &lt;SessionsTable data={data.sessions?.data} /&gt;
            &lt;LiveEventsTable data={data.liveEvents?.data} /&gt;
          </div>
        </div>
      &lt;/main&gt;
    </div>
  );
};

function App() {
  return (
    &lt;QueryClientProvider client={queryClient}&gt;
      &lt;DashboardContent /&gt;
    &lt;/QueryClientProvider&gt;
  );
}

export default App;
```

Part 3: Deployment & Verification

Step 1: Build Backend

```
cd rum-backend
mvn clean package
java -jar target/rum-backend-2.0.0.jar
```

Step 2: Build Dashboard

```
cd rum-dashboard
npm install
npm start
```

Step 3: Verify Endpoints

Test with curl:

```
# Test web vitals endpoint
curl "http://localhost:8080/api/rum/stats/web-vitals?startMs=$(date +%s000)&amp;endMs=$((

# Test page views
curl "http://localhost:8080/api/rum/stats/page-views?startMs=$(date +%s000)&amp;endMs=$((
```

```
# Test errors
curl "http://localhost:8080/api/rum/stats/errors?startMs=$(date +%s000)&amp;endMs=$(($(da

# Test live events
curl "http://localhost:8080/api/rum/stats/live-events?limit=50"
```

Step 4: Verify Dashboard

1. Open http://localhost:3000
2. Check if all cards load with data
3. Verify charts render correctly
4. Check tables populate
5. Test time range filter
6. Monitor console for errors

Summary of Updates

✅ **Backend Enhancements:**

- 9 new stats endpoints
- Aggregated metrics
- Time-based queries
- Session tracking
- Live events stream

✅ **Dashboard Improvements:**

- New sessions table
- Live events stream
- Better error handling
- Auto-refresh with stale time
- Improved data caching
- More reliable retry logic

✅ **Performance Optimizations:**

- Reduced API calls
- Better data caching
- Stale time implementation
- Retry strategies

Your updated RUM platform now provides comprehensive real-time monitoring!