Kibana — wykorzystanie danych z RUM JavaScript (Angular)

Propozycja dokumentu: User Experience (RUM) + Discover/Lens, KQL, alerty i korelacje dla aplikacji Angular z agentem RUM JS.  
*Data: 2025-08-21 07:38*

# 1) Cel dokumentu

* Pokazać co i jak wyciągać z danych RUM w Kibanie, gdy Angular jest zainstrumentowany agentem RUM JS.
* Dostarczyć gotowe widoki w User Experience, zapytania KQL, przepisy Lens i alerty do szybkiego użycia.
* Uwzględnić specyfikę SPA (Angular): transakcje page-load i route-change, XHR/fetch, błędy JS.

# 2) Gdzie są dane i jak je przeglądać

* Aplikacja: Observability → User Experience (RUM).
* Data view (Discover/Lens): apm-\* (zwykle aliasy dla danych RUM), ewentualnie traces-apm\* / logs-apm\*.
* Najważniejsze pola: service.name, service.environment, transaction.type (page-load/route-change/user-interaction), transaction.name, transaction.duration.us;
* Spany (XHR/fetch/resources): processor.event:"span", span.type:"external"/"resource", span.subtype:"http"; http.\* i destination.\*;
* Pola UX i przeglądarki: user\_agent.\*, url.full/url.path, page.\* (title/referrer), experience.\* (np. lcp/cls/fid/inp jeżeli dostępne).

# 3) Gotowe widoki (User Experience + RUM)

* UX Overview — Core Web Vitals (LCP/CLS/…\*), Page load/route change latency, liczba sesji i błędów JS.
* UX Breakdown — porównania wg device type, OS, przeglądarki, regionu (jeśli dane geo dostępne).
* Errors — grupy błędów JS (message/stack), trend i próbki zdarzeń.
* AJAX (XHR/fetch) — czas odpowiedzi, statusy, domeny zależności (external).

\*INP/FID zależnie od wersji agenta i konfiguracji — jeśli dostępne, pojawią się w panelach UX/Discover.

# 4) Typowe pytania, na które odpowie Kibana

* Czy czasy „page-load” albo „route-change” pogorszyły się w ostatnich wydaniach?
* Które przeglądarki/urządzenia są najwolniejsze?
* Które endpointy XHR/fetch opóźniają interakcje użytkownika?
* Czy rośnie liczba błędów JS i na jakich stronach/trasach?
* Jak wygląda wpływ nowego releasu (service.version) na UX (LCP/CLS/latencje)?

# 5) Przepisy „Lens” (wizualizacje 30–90 s)

A. p95 czasu page-load/route-change per trasa

* Filter: processor.event:"transaction" and transaction.type:("page-load" or "route-change") and service.name:"my-spa"; Metric: 95th percentile of transaction.duration.us (→ ms); Break down by: transaction.name.

B. Error rate JS w czasie

* Filter: processor.event:"error" and service.name:"my-spa"; Metric: Count; dodaj wskaźnik (Count errors / Count transactions page-load+route-change — opcjonalnie).

C. Top powolne XHR/fetch

* Filter: processor.event:"span" and span.type:"external" and span.subtype:"http" and service.name:"my-spa"; Metric: 95th percentile of span.duration.us; Break down by: destination.service.resource/url.domain.

D. Core Web Vitals — rozkład LCP

* Jeśli pola experience.lcp dostępne: Metric: Percentile (50/75/95) of experience.lcp (ms); Break down by: url.path lub transaction.name.

E. Porównanie przeglądarek

* Filter: processor.event:"transaction" and transaction.type:("page-load" or "route-change"); Metric: Median of transaction.duration.us; Break down by: user\_agent.name.

# 6) Propozycje dashboardów własnych

„UX — Performance & Stability” — panele:

* p95 transaction.duration.us (page-load+route-change) per transaction.name (trend).
* Top wolne przeglądarki/OS (user\_agent.name, user\_agent.os.name).
* Top wolne XHR/fetch (destination.service.resource/url.domain) z p95 span.duration.us.
* Tabela „ostatnie błędy JS”: @timestamp, url.full, error.exception.message, user\_agent.name.

„Core Web Vitals” — panele:

* Percentyle LCP i CLS per url.path/transaction.name.
* Udział „good/needs improvement/poor” (jeśli dostępne metryki).
* Trend LCP po release (labels.release lub service.version).

# 7) KQL — gotowe „cookbook” (kopiuj-wklej)

service.name : "my-spa" and processor.event : "transaction" and transaction.type : ("page-load" or "route-change") and transaction.duration.us >= 3000000 # ≥ 3 s

service.name : "my-spa" and processor.event : "error" # błędy JS

processor.event : "span" and span.type : "external" and span.subtype : "http" and service.name : "my-spa" # XHR/fetch

url.path : "/checkout" and processor.event : "transaction" and transaction.type : "route-change" # konkretna trasa

user\_agent.name : "Chrome" and processor.event : "transaction" # tylko Chromium/Chrome

experience.lcp >= 2500 # LCP > 2.5 s (jeśli pole dostępne)

labels.release : "2025.08.15" and processor.event : "transaction" # porównanie releasów

# 8) Alerty — gotowy pakiet reguł

A. Latency threshold (RUM)

* KQL: processor.event:"transaction" and transaction.type:("page-load" or "route-change") and service.name:"my-spa" and transaction.duration.us >= 3000000; Group by: transaction.name; Okno: 5 min.

B. Wzrost błędów JS

* KQL: processor.event:"error" and service.name:"my-spa"; Warunek: More than N matches w 5–10 min; Group by: url.path/transaction.name.

C. Degradacja XHR/fetch

* KQL: processor.event:"span" and span.type:"external" and span.subtype:"http" and span.duration.us >= 2000000; Group by: destination.service.resource/url.domain.

D. Core Web Vitals (LCP) — próg

* KQL: experience.lcp >= 4000 (jeśli dostępne); Harmonogram: 5–15 min; Group by: url.path.

E. „No data” (brak RUM)

* Typ: Less than 1 match; KQL: service.name:"my-spa" and processor.event:"transaction"; Okno: 10–15 min.

# 9) Korelacje i łączenie z innymi danymi

* APM (backend): distributed tracing (W3C) pozwala zobaczyć trasę RUM → API → DB; filtruj po trace.id.
* Filebeat (logi backendu): sprawdź błędy 5xx na endpointach, które są wolne w RUM (po url.path/domain).
* Metricbeat: CPU/RAM hostów backendowych w czasie skoków RUM latency.
* Heartbeat: monitoruj endpointy API używane przez SPA i koreluj z degradacją UX.

# 10) Raportowanie i współdzielenie

* Saved Searches: „RUM — slow routes ≥ 3s (24h)”, „RUM — JS errors (24h)”.
* Dashboard PDF (Reporting): tygodniowy raport UX (p95 page-load/route-change, XHR p95, błędy JS).
* Linki z filtrami (service.name, environment, transaction.name/route).

# 11) Dobre praktyki pracy z RUM (Angular)

* Ustaw `service.name`, `service.environment`, `service.version` i (opcjonalnie) `labels.release` — porównania wydań.
* Włącz śledzenie SPA (router) — transakcje route-change powinny mieć czytelne nazwy tras.
* Nie loguj danych wrażliwych: rozważ `sanitizeFieldNames`, kontrolę `captureBody`/`captureHeaders`.
* Próbkowanie: zaczynaj od umiarkowanego (`transactionSampleRate`), zwiększaj tylko gdy potrzebne.
* Dodaj własne „custom context” (labels.\*) — np. tenant/region/feature-flag — ułatwia filtry i SLO.
* Kardynalność: unikaj analiz terms po pełnym url.full; preferuj url.path lub transaction.name.

# 12) „Starter pack” (do importu / odtworzenia)

Saved Searches:

RUM — Slow routes ≥ 3s: processor.event:"transaction" and transaction.type:("page-load" or "route-change") and transaction.duration.us >= 3000000

RUM — JS errors (24h): processor.event:"error" and @timestamp >= now-24h

Dashboardy:

* UX — Performance & Stability (sekcja 6).
* Core Web Vitals (sekcja 6).

Alerty:

* RUM\_LATENCY\_THRESHOLD, RUM\_JS\_ERRORS\_SPIKE, RUM\_XHR\_SLOW, RUM\_NO\_DATA, (opcjonalnie) RUM\_LCP\_THRESHOLD.

# 13) Aneks — pola przydatne w analizie

* transaction.type (page-load/route-change/user-interaction), transaction.name, transaction.duration.us.
* span.type/subtype (external/http/resource), span.duration.us, destination.\* (service.name/resource), http.\*.
* url.full/url.path, page.\* (title/referrer), user\_agent.\* (name/version/os/device).
* experience.\* (np. lcp/cls/fid/inp jeśli dostępne).
* trace.id / transaction.id — korelacja z backendem i logami.

# Co dalej?

Mogę przygotować pakiet Saved Objects (dashboardy, saved searches, alerty) oraz przykładowe fragmenty kodu Angular z router instrumentation i niestandardowymi etykietami (labels.\*).