Unlocking Observability with React and Node.js

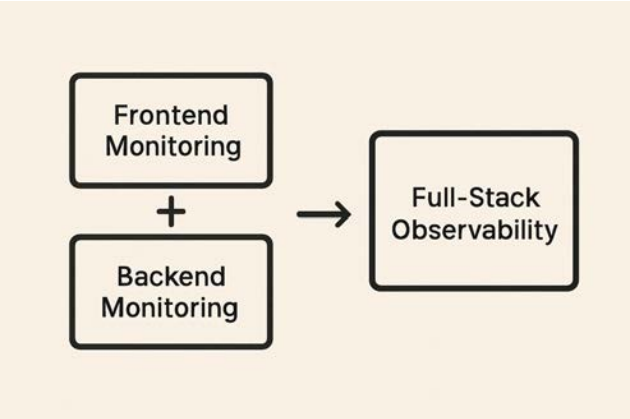
Mohit Menghnani,
Staff Software Engineer
Twilio

Agenda

- Introduction: The need for a unified view
- Defining Unified Full-Stack Observability
- Correlating Frontend and Backend Data
- Case Studies
- Tools & Best Practices
- Conclusion

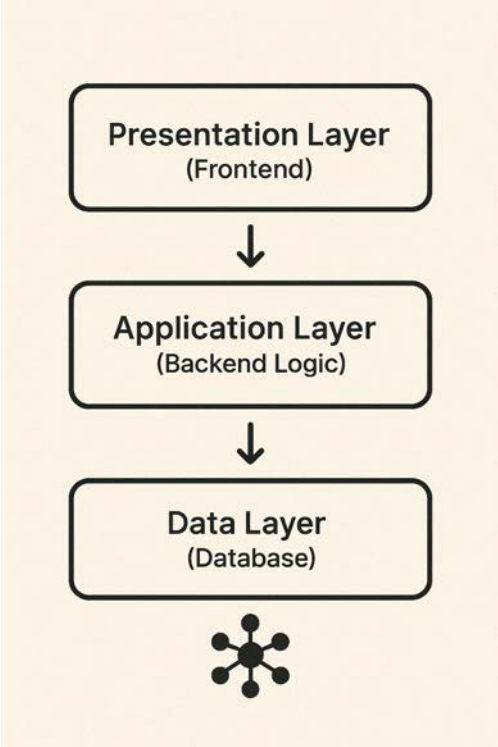
Introduction - The Need for a Unified View

- Modern web applications (React frontend, Node.js backend) are increasingly complex.
- Traditional monitoring often treats frontend and backend separately, leading to blind spots.
- Unified full-stack observability provides a holistic understanding of application behavior across all layers.



Defining Unified Full-Stack Observability for React and Node.js

- Monitoring the entire application stack from the React UI to the Node.js backend within a cohesive framework.
- Emphasizes the critical connections and dependencies between frontend and backend.
- Requires a single platform or tightly integrated tools for data correlation.
- Contrasts with siloed monitoring that limits the ability to diagnose cross-tier issues.



The Compelling Advantages of Integrated Observability

- **Faster Issue Resolution:** Holistic view enables quicker root cause analysis and reduces MTTR/MTTD.
- **Improved Team Collaboration:** Shared understanding and common platform break down silos.
- **Enhanced Performance Insights:** Identifies bottlenecks across both frontend and backend.
- **Data-Driven Decision Making:** Comprehensive data informs optimization and resource allocation.
- **Proactive Issue Detection:** Anomaly detection across the full stack helps prevent user impact.

Achieving End-to-End Visibility: Correlating Frontend and Backend Data

- **Trace Propagation:** Unique IDs generated on the frontend and passed to the backend (W3C Trace Context, `sentry-trace`, `traceparent`).
- **Correlation IDs:** Unique identifiers generated on the frontend and passed to the backend.
- **Session IDs:** Linking frontend user sessions with backend activities.
- **Log Correlation:** Embedding trace/correlation IDs in both frontend and backend logs.
- **RUM/APM Integration:** Automatic correlation of frontend user experience data with backend traces.
- **OpenTelemetry:** Provides a standardized approach to context propagation.

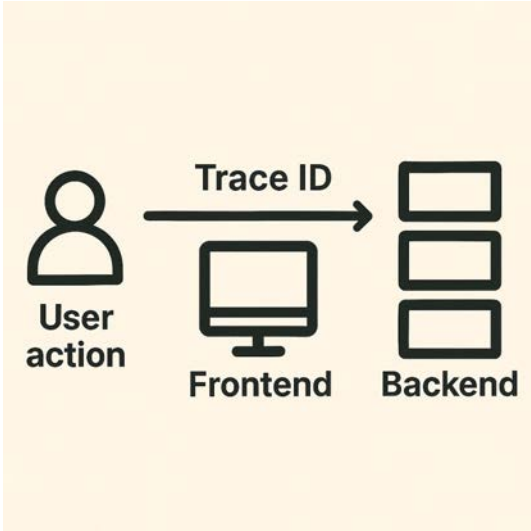
```
GET /api/user/info HTTP/1.1
Host: example.com
traceparent: 00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01
tracestate: rojo=00f067aa0ba902b7,congo=t61rcWkgMzE
```

Key Performance Indicators for Holistic Application Understanding

KPI Name	Frontend Relevance	Backend Relevance	Importance for Unified Observability
User-Perceived Latency	Rendering speed, resource loading, interactivity	API response times, data retrieval, server processing	Directly reflects user experience and the combined performance of frontend and backend.
Error Rates Across Tiers	JavaScript errors, HTTP errors, unhandled exceptions	Server errors, API errors, database errors	Indicates overall stability and reliability across all layers.
Request Latency (End-to-End)	Time in browser, network to backend	Backend processing, database queries, network back	Provides a holistic view of the request lifecycle and identifies bottlenecks across tiers.
Throughput	User interactions, API calls initiated	API requests processed	Measures the application's capacity and scalability under load.
Resource Utilization	Browser performance, memory leaks	Node.js process performance, memory leaks, CPU	Helps identify resource constraints and optimize performance on both client and server sides.
User Engagement Metrics	Page views, session duration, conversion rates	Backend impact on user behavior, API usage	Connects technical performance with business outcomes and user satisfaction.
Core Web Vitals	LCP, FID, CLS	Backend impact on initial content delivery	Essential for SEO, user experience, and perceived performance; reflects backend's role in loading.

Implementing Distributed Tracing Across React and Node.js

- **Traces:** Represent the entire lifecycle of a request.
- **Spans:** Denote individual operations within a trace.
- **Context Propagation:** Linking spans using unique trace IDs carried through all operations.
- **Benefit:** Identify performance bottlenecks and latency issues in complex systems.
- **OpenTelemetry:** Standardized approach for context propagation.
- **Tools:** OpenTelemetry, Jaeger, Zipkin.



Case Studies

- Correlating frontend error spikes with backend API latency to identify root causes.
- Analyzing user navigation patterns leading to high backend resource consumption.
- Example:
    - Payment error on frontend traced to a timeout in a backend payment API.
    - Slow frontend page load corelated with slow backend API response for image data.

MERN stack observability benefits from unified tools and integrations.

Tooling the Unified Approach: Integrated Observability Platforms

- **Datadog:** Full-stack monitoring with RUM for frontend and APM for backend.
- **New Relic:** Integrated solutions with quickstarts for MERN stack.
- **Honeycomb:** Emphasizes unified analysis of telemetry data.
- **Dynatrace:** AI-powered full-stack observability.
- **Elastic Observability:** Unifies logs, metrics, and traces from various sources.
- **Grafana Cloud:** Frontend RUM integrated with backend data sources.
- **Observe:** Connects end-user experience with backend troubleshooting.
- **OpenObserve:** Open-source platform for logs, metrics, and traces.
- **OpenTelemetry:** Vendor-neutral standard for instrumentation.

Best Practices for Proactive Issue Detection and Resolution

- Create **unified dashboards** visualizing key metrics and traces from both tiers 
- Set up **alerts** based on correlated frontend and backend data 
- Utilize **anomaly detection** to proactively identify unusual behavior. 
- Implement effective **tagging strategies** for easy data correlation. 
- Establish clear **observability goals** and **SLOs**. 
- Involve both frontend and backend teams in defining alerts and dashboards 

Conclusion - Embracing the Future of Full-Stack Observability

- Unified full-stack observability provides faster issue resolution, enhanced performance insights, and proactive problem detection.
- Breaks down silos between frontend and backend monitoring.
- Enabled by modern observability tools and open standards like OpenTelemetry.
- Crucial for maintaining system reliability, optimizing performance, and delivering exceptional user experiences.

Thank you