Observability for WebDSL with OpenTelemetry

Architecture

Reference architecture

OT Reference Architecture suggests the following deployment scheme. Every WebDSL application gets deployed together with an OpenTelemetry Collector [2]. Separately a metrics back-end service (such as Prometheus) is deployed. The application.ini file should contain the port of the collector, and the collector should be independently configured with a back-end address.

For a production build inability to initialize OT library or reach OT Collector should lead to the application shutdown because in production the absence of telemetry from an application signals it's not working and needs to be restarted anyway. In a debug run though OT failure should not cause the application to stop.

We should decide and make guidelines regarding what metrics collection back-end use for debugging and for production and how to set them up.

Components

Traces

Traces are the central concept in OT, they represent a trace of a single request (not necessarily Web-request) execution and comprise a tree of spans. Spans represent particular operations and encapsulate attributes, events and other spans.

In the context of WebDSL it seems natural to associate a trace with each Web-request. As long as at the moment WebDSL applications handle two kinds of requests — page requests and AJAX requests — it feels logical to associate the root span of a trace with the page or AJAX template respectively. Each template ("usual" one or another AJAX template) invoked during a request handling should create its own span.

Minimal user-facing API: emitting a named event in the current span.
Additional APIs: setting attributes and a status to the current span.

Metrics

Default set of metrics is an open question. Performance metrics (request count, request latency, DB latency and such) are obvious candidates.

User-facing API to record user-relevant metrics should be designed and implemented.

Logs

Existing logging should be migrated to OT API for uniformity.
Additionally we should adopt structured logging. The (mandatory part of the) structure needs to be designed and specified.

Implementation considerations

The natural choice for the implementation “backbone” is the official OT Java library [1].

Impact analysis

- OT collector settings should be added to application.ini file and thus the parser and settings data structure need to be extended.
- OT library initialization code should be added to the WebDSL runtime initialization
- OT tracer object needs to be created with every request and saved together with the request object
- Current OT span needs to be created and saved for each component (page, template and so on)
- User-facing API methods should access current span object

References

1. Java library
2. OpenTelemetry Collector
3. Example Collector configuration with Docker Compose

Open questions

- "What telemetry data should be collected by default, what's most useful for all the Web applications?" “What telemetry data proved to be actually useful for debugging in real-world settings?” It would be nice to see some empirical research into that and base decisions on top of it.
- The next question is about a span tree inside a trace. I suggested associating a span with every page and every template inside it, but how useful that is or where to stop is another open question.
- What useful data WebDSL can already associate with these spans to make debugging easier (we generate unique IDs for the template instantiations, how useful is that?).