Logging using Sentry

We use Sentry, an awesome logging cockpit, for logging of errors, crashes and problems.

The reason for that is that, we don't login to our services every day to check for problems in the log files, since it's cumbersome and with all of our applications, it's just a huge waste of time.

Sentry will summarize all the problems, logs and other important things with additional debugging information like browser, system, ... across different projects within one dashboard and provides email reports, recurrence tracking, ...

This is why we make sure to integrate it in as many projects as possible to always know what's going on, even if we don't check manually every day.

Generally rule of thumb: Only log stuff to Sentry, that is really important and of interest for us!

Don't just use it as a debugging output collecting millions of entries for nothing, because every sentry request produces more data on the sentry server and leads to an additional request, that slows the request down. But that also does not mean, you should not use it at all.

Just use it responsibly!

Where and how can I access Sentry?

We have an onpremise sentry instance available:

https://sentry.mqt.at

Just log in using the credentials you received.

If you don't have any Sentry credentials yet, ask for them. ;)

How do I add Sentry to a react application?

When you create a new project in Sentry, Sentry will provide you with all the necessary steps to get up and running with it.

Here you can find more detailed instructions:

https://docs.sentry.io/platforms/javascript/react/

But in short, we just have to init sentry in our project:

Install the `sentry/sentry-laravel` package:

```
$ yarn add @sentry/react
```

Connect the SDK to Sentry:

```
import React from 'react';
import ReactDOM from 'react-dom';
import * as Sentry from '@sentry/react';
import App from './App';

Sentry.init({dsn:
"https://${#####SENTRY_PROJECT_KEY#####}@sentry.mqt.at/${#####SENTRY_PROJECT_ID#####}"})
;

ReactDOM.render(<App />, document.getElementById('root'));
```

On its own, `@sentry/react` will report any uncaught exceptions triggered by your application.

You can trigger your first event from your development environment by raising an exception somewhere within your application. An example of this would be rendering a button:

Custom Logging/Capturing

The setup above will already log all fatal errors. But Sentry also allows us to log custom messages from within our application as described here:

https://docs.sentry.io/error-reporting/capturing/?platform=javascript

When to use custom logging?

The solution above only logs crashes and hard errors within our system.

BUT we often catch errors, before they crash the application, to make sure everything works as expected, but still would like to log a problem to fix it or at least be aware of it.

This is where we should log an exception.

```
try {
    aFunctionThatMightFail();
} catch (err) {
    Sentry.captureException(err);
}
```

Logging messages

Sometimes we don't have an exception, but still want to log something to inform our selves:

```
// Example: we inform ourselves, that in a specific case data, we really expect is not
given
if (typeof user.profile === 'undefined') {
   Sentry.configureScope(function(scope) {
     scope.setUser({"email": user.email});
   });
   Sentry.captureMessage('Unexpected: Users profile data was not provided');
}

// Example: we add a log if the impossible happens for some reason and we want to check
if this really happens
if (checkIfEverythingIsGoodWhichIsTheNormalCase()) {
  return;
}

Sentry.configureScope(function(scope) {
  scope.setTag("problemType, "userProfileCheck");
  scope.setExtra('superDuperData', theData);
});
Sentry.captureMessage('User did not pass everythingIsOkCheck. Checkout why');
```

Also see how to add additional debugging data using context:

https://docs.sentry.io/enriching-error-data/additional-data/?platform=javascript

https://docs.sentry.io/enriching-error-data/scopes/?platform=javascript

Sentry holds context in the current scope, and thus context clears out at the end of each operation (for example, requests). You can also push and pop your scopes to apply context data to a specific code block or function.

Setting the log level

There are different logging levels:

- debug (for debugging info and just development)
- info (information, that could be important, but not an error)
- warning (not fatal, but should be looked and checked)
- error (an error occured, that should be looked at soon)
- fatal (a crash that should be checked immediately)

Use the following command to set the level accordingly:

```
Sentry.configureScope(function(scope) {
  scope.setLevel('warning');
});
```

Where to find more information:

https://docs.sentry.io/