Endress+Hauser Digital Solutions Coding Guidelines

Coding Conventions for JavaScript and TypeScript in the context of React, Next and Nest.

1 Preamble

1.1 Exceptions from the E+H Coding Conventions

It is permitted to add project specific rules to the E+H Coding Conventions as long as compliance with the mandatory definitions listed in this document is ensured.

1.2 Eslint

Eslint is our linter of choice because it is practically the standard in the world of JavaScript. It can also be used as a formatter, but we'll primarily use it as a linter.

1.3 Prettier

Prettier is the formatter that should be used. It has also become a standard in the JavaScript world.

1.4 Husky

Husky allows commands to be executed automatically with hooks. We use Husky to make sure that the code is lined and formatted before committing with the pre-commit hook.

1.5 Editor settings

```json
{
  "useTabs": false,
  "tabWidth": 2,
  "printWidth": 100,
  "endOfLine": "auto",
  "arrowParens": "avoid",
  "semi": true,
  "singleQuote": true,
  "bracketSpacing": true,
  "trailingComma": "none"
}
```

2 Generic Coding Conventions

2.1 Blank line after statement block

A statement such as if-cases, switch cases, loops or anything else that defines a new scope within should end with a blank line to aid readability.

```js
foo.map(bar => {
  // does something
});

console.log(foo);
if (foo) {
  // does something
}

console.log(foo);
```

2.2 Naming Booleans

A Booleans name should always ask something to which you can get a yes or no answer.

```js
const isTokenStillValid = token.validUntil > new Date.now();

// easily readable
if (isTokenStillValid) {
  // easy to understand that in this case the token is still valid
} else {
  // and isn't in this case
}
```

2.3 Function length

The length of a function should not exceed 15 lines. Exceptions are functional React components and functions in repositories of ORMs, as in these cases it makes no sense to outsource the logic. If a case arises in which it also makes no sense, this must be documented.

2.4 React components

Class-based components are no longer part of the standard since React 16.8. Since then, functional components have been used. However, these differ from the normal functions in two aspects. The name should be capitalized, and a regular function should be used instead of the ES6 arrow functions.

```
// do
export default function Header(properties) {
  return <p>This is a header!</p>;
}

// don't
export default header = (properties) => {
  return <p>This is a header!</p>;
}
```

2.5 Avoid single letter names. Be descriptive with your naming

```
// bad
function q() {
  // ...
}

// good
function query() {
  // ...
}
```

2.6 Use camelCase when naming objects, functions, and instances

```
// bad
const OBJEcttsssss = {};
const this_is_my_object = {};
function c() {}

// good
const thisIsMyObject = {};
function thisIsMyFunction() {}
```

2.7 Do not use trailing or leading underscores

Why? JavaScript does not have the concept of privacy in terms of properties or methods. Although a leading underscore is a common convention to mean “private”, in fact, these properties are fully public, and as such, are part of your public API contract. This convention might lead developers to wrongly think that a change won’t count as breaking, or that tests aren’t needed. tl;dr: if you want something to be “private”, it must not be observably present.

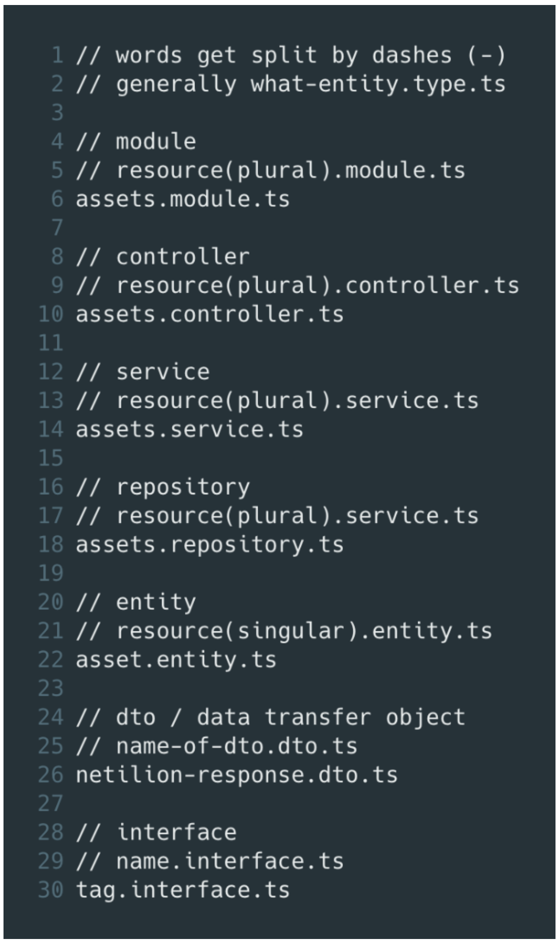
```
// bad
this.__firstName__ = 'Panda';
this.firstName_ = 'Panda';
this._firstName = 'Panda';

// good
this.firstName = 'Panda';

// good, in environments where WeakMaps are available
// see https://kangax.github.io/compat-table/es6/#test-WeakMap
const firstNames = new WeakMap();
firstNames.set(this, 'Panda');
```

2.8 Don’t save references to this. Use arrow functions or binds.

```
// bad
function foo() {
  const self = this;
  return function () {
    console.log(self);
  };
}

// bad
function foo() {
  const that = this;
  return function () {
    console.log(that);
  };
}

// good
function foo() {
  return () => {
    console.log(this);
  };
}
```

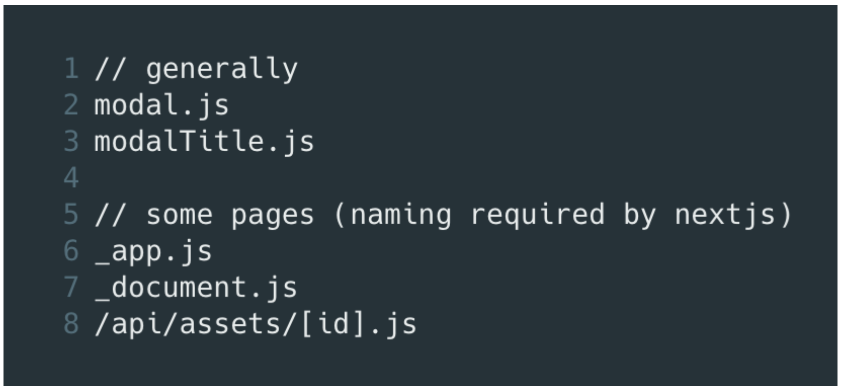
Or https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind

2.9 Nestjs specific file naming



```
// words get split by dashes (-)
// generally what-entity.type.ts

// module
// resource(plural).module.ts
assets.module.ts

// controller
// resource(plural).controller.ts
assets.controller.ts

// service
// resource(plural).service.ts
assets.service.ts

// repository
// resource(plural).service.ts
assets.repository.ts

// entity
// resource(singular).entity.ts
asset.entity.ts

// dto / data transfer object
// name-of-dto.dto.ts
netilion-response.dto.ts

// interface
// name.interface.ts
tag.interface.ts
```

2.10 Nextjs specific file naming



```
// generally
modal.js
modalTitle.js

// some pages (naming required by nextjs)
_app.js
_document.js
/api/assets/[id].js
```