Coding Standards

This project's frontend will be programmed in Javascript.

1. Files

1.1. File Naming

File names must be written in `camelCase`. If they contain a React component, they must use `UpperCamelCase`.

1.2. File Structure

All React components must be in a folder called `components`. Redux states and action creators must be in a folder called `redux`. There may be multiple subfolders inside those two folders.

2. Formatting

2.1. Braces

Curly braces must be used even if they are not necessary

```
/* YES */
if(condition) {
  doSomething();
}

/* YES */
if(condition) {doSomething();}

/* NO */
if(condition) doSomething();
```

2.2. Statements

There can only be one statement per line. Statements can only have whitespace characters behind them.

```
/* YES */
if(condition) {
  doSomething();
}
else {
  doSomethingElse();
}

/* NO */
if (condition) {
  doSomething();
} else {
  doSomethingElse();
}
```

2.3. Maximum line length

A single line may only have up to 120 characters, counting whitespaces.

2.4. Line wrapping

It is heavily encouraged to only wrap lines at a higher syntactic level.

```
/* YES */
const answer = (168 / 4)
    * 2;

/* NO */
const answer = (168
    / 4) * 2;
```

2.5. Ternary operator with parenthesis

Should the ternary operator require parenthesis, the `:` symbol must be included in the same line as the first one closing and the second one opening. The line cannot consist of anything else.

```
/* YES */
const myComponent = someCondition ? (
  <div>
    <p>This comes out if true</p>
  </div>
) : (
  <div>
    <h1>This comes out if false</h1>
  </div>
);
```

3. React

3.1. Destructuring props and state

Both props and state must be destructured before usage. If a key has the same name in both the props and the state, it cannot be destructured.

3.2. Initializing the state

State must be initialized in the constructor.

3.3. Prop passing

Props must be properly indented when passed in multiple lines.

```
<MyComponent
  firstProp={firstValue}
  secondProp={secondValue}
  thirdProp={thirdValue}
/>
```

3.4. JSX

JSX code must always be in parenthesis

3.5 Dynamic JSX

Components passed dynamically must first be stored in a variable, and the variable can then be passed.

```
const percentage = document.getElementById("percentage-
input-field").value;

/* YES */
let body = null;
if(percentage > 100) {
  body = (
    <p>That's not a valid value!</p>
  );
}
else {
  body = (
    <React.Fragment>
      <h1>You have typed {percentage}%</h1>
      <p>That's a pretty good percentage!</p>
    </React.Fragment>
  );
}

return (
  <div>
    {body}
  </div>
);

/* NO */
return (
  <div>
    {
      percentage > 100 ? (
        <p>That's not a valid value!</p>
      ) : (
        <React.Fragment>
          <h1>You have typed {percentage}%</h1>
          <p>That's a pretty good percentage!</p>
        </React.Fragment>
      )
    }
  </div>
);
```

4. Objects and Arrays

4.1 Trailing commas

The use of trailing commas is heavily encouraged.

```
const myArray = [
  "value1",
  "value2",
];
```

4.2 Mixing quoted and unquoted object keys

Objects should only have either quoted or unquoted keys constantly throughout the project.

5. Naming & Declaring

5.1 Variables

Variables must be named using `camelCase`. Variables must not be abbreviated if unnecessary, and must be clearly understandable by anybody, even outside the project.

5.2 Usage of var

The usage of the `var` keyword is heavily discouraged. Use `let` and `const` instead, and declare the variables as needed to limit their scope.

5.3 Destructuring in functions

Destructuring in functions is not allowed directly in the arguments. The arguments may only be destructured within the function.

```
/* YES */
function myFunction(someObject) {
  const {someProperty, someOtherProperty} = someObject;
  // some code
}

/* NO */
function myFunction({someObject, someOtherProperty}) {
  // some code
}
```

5.4 Classes

Class names must be written in `UpperCamelCase`.

6. JSDoc

6.1 React props

Component props must be documented using the `@param` option, and the prop name must always be preceded by `prop.`.

```
/**
 * Shows a message
 *
 * @param {string} props.message The message to be shown
 */
class MessageShower extends React.Component {
  render() {
    const {message} = this.props;
    return (
      <p>{message}</p>
    );
  }
}
```

6.2 Redux props

Component props that originate from a Redux state may not be documented. Document the state itself instead.

6.3 Redux state

The initial state of the Redux state must be properly documented. Use the `@prop` option for the keys.

```
/**
 * The initial state of this state.
 *
 * @prop {boolean} initialized  Indicates if the state
is ready.
 * @prop {boolean} loading      Indicates if the state
is loading the data.
 */
const init = {
  initialized: false,
  loading: false,
};
```