Accessibility

- https://eduardz1.github.io/La-Nuit-de-L-Info/

We used as many standard components as possible, to make the website more familiar to the user and avoid building custom solutions that might not have been though out in all the details.

Colors are contrasty, most of the text is black on white. Buttons are clearly labeled and conform to the standards of the web.

The website is responsive and can be used on mobile devices.

The website is also navigable using the keyboard, with the tab key moving the focus between the different elements of the page.

We used Modals when necessary to avoid cluttering the page with too much information.

Error Handling

We took special care for error handling. For the website we developed we chose to try to follow the Web Sustainability Guidelines, in particular the ones defined in https://w3c.github.io/sustyweb/. Our website is minimal and uses only standard components, this makes it lighter to compile (through component reuse) and esasier on the user.

There are no actions that are being run in the background (minus the graphics animations), everything is explicitly triggered by the user.

Even in the few automatic animations we have, we chose to utilize emojis instead of SVG images to reduce the load on the website to only the single UNICODE character.

The image of the human body was translated to vector graphics and manually optimized to remove unnecessary points and reduce the file size. Animations on the human model are done as simple CSS animations, which are more efficient than using JavaScript.

Our website gets straight to the point, without wasting the user's time.

Error Handling

In our website we handle errors by using an `ErrorBoundary`, this makes it so the user never has no experience a crash. In particular our Main `App` component is wrapped in the `ErrorBoundary` component:

```javascript
const App = () => {
  const { message, setError, setInfo, setWarning, setMessage } =
    useMessageContext();

  return (
    <ErrorBoundary>
      <MessageContext.Provider
        value={{ message, setMessage, setInfo, setError, setWarning }}
      >
        <MessageToast />
        <Routes>
          <Route path="*" element={<Page />}></Route>
        </Routes>
      </MessageContext.Provider>
    </ErrorBoundary>
  );
};

```

```
export default App;
```

Where the `ErrorBoundary` component is defined as:

```JavaScript
class ErrorBoundary extends Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false, error: null, errorInfo: null };
  }

  componentDidCatch(error, errorInfo) {
    this.setState({
      hasError: true,
      error: error,
      errorInfo: errorInfo,
    });
  }

  render() {
    if (this.state.hasError) {
      return (
        <div>
          <h1>Something went wrong.</h1>
          <details style={{ whiteSpace: "pre-wrap" }}>
            {this.state.error && this.state.error.toString()}
            <br />
            {this.state.errorInfo.componentStack}
          </details>
        </div>
      );
    }
    return this.props.children;
  }
}

ErrorBoundary.propTypes = {
  children: PropTypes.node.isRequired,
};

export default ErrorBoundary;
```

Furtermore we also handle errors as a context, we can see the `MessageContext` component that is used to display messages to the user:

```JavaScript
import React from "react";

/**
 * Context used to display messages of different kind to the user as Toasts.
 * @see MessageToast
 */
const MessageContext = React.createContext({
  message: "",
  setMessage: () => {},
  setInfo: () => {},
  setError: () => {},
  setWarning: () => {},
});

```

```
export default MessageContext;
```

Which is displayed using a special Toast:

```javascript
import { useContext } from "react";
import Toast from "react-bootstrap/Toast";
import ToastContainer from "react-bootstrap/ToastContainer";
import MessageContext from "./MessageContext";

/**
 * Toast used to display all the messages, has different behavior for errors.
 * @see MessageContext
 */
const MessageToast = () => {
  const { message, setMessage } = useContext(MessageContext);

  return (
    <ToastContainer className="p-3" position="top-center">
      <Toast
        show={message.msg !== ""}
        autohide={message.type !== "error"}
        onClose={() => setMessage({ msg: "", type: "" })}
        delay={1000}
        // I prefer having them labeled as "error" instead of danger throughout the app
        bg={message.type === "error" ? "danger" : message.type}
      >
        {message.type === "error" && (
          <Toast.Header>
            <strong className="me-auto"> Error </strong>{" "}
          </Toast.Header>
        )}
        <Toast.Body className={message.type === "error" && "text-white"}>
          {message.msg}
        </Toast.Body>
      </Toast>
    </ToastContainer>
  );
};

export default MessageToast;
```

And classifies the errors in multiple categories, to let the user know the severity of the error, without throwing an exception:

```javascript
import { useState } from "react";

/**
 * A hook that provides functions to set different kinds of messages.
 * @see MessageContext
 */
const useMessageContext = () => {
  const [message, setMessage] = useState({ msg: "", type: "" });

  const setError = (err) => {
    setMessage({ msg: err.message || "Unknown Error", type: "error" });
  };

  const setInfo = (msg) => {
```

```
    setMessage({ msg: msg, type: "info" });
  };

  const setWarning = (msg) => {
    setMessage({ msg: msg, type: "warning" });
  };

  return { message, setError, setInfo, setWarning, setMessage };
};

export default useMessageContext;
```

We can see an example of usage in the function that queries the google generative AI API:

```JavaScript
async function aiRun(query) {
  setLoading(true);

  try {
    const result = await model.generateContent(query);
    const response = result.response;
    const text = response.text();
    setResponse(text);
  } catch (error) {
    setError(error);
  } finally {
    setLoading(false);
  }
}
```