INNOVATION LAB FOR WEARABLE AND UBIQUITOUS COMPUTING

MACHINE LEARNING AND DATA ANALYTICS LAB

DEPARTMENT OF COMPUTER SCIENCE

Stress+

Student:
Jonas Schüll

Project partner: Chair of Health Psychology

Scrum master:
Michael Nissen and Matthias Zürl

Date: August 27, 2020

Contents

1 Introduction

This project aims to develop a tool called *Stress+* that simplifies the existing stress test process. Stress is one of the major problems, it often gets overlooked and plays a negative influence on the overall quality of life. Due to its significant public health concern, the demand for a stress test is increasing rapidly. The basic idea behind this project is to overcome the limitations of existing stress test which requires a lot of time, assistance, and effort. *Stress+* is a user-friendly tool that allows the participants to take the stress test with their own devices, anytime, anywhere. Based on the Montreal Imaging Stress Task (MIST), this app sets up a challenging environment that induces the psychological stress level of the participants by allowing them to take a mental arithmetic test which is supported by stress-inducing factors like time constraints, scores, feedback etc. The results are shared to doctors which aids them to diagnose and plan any further treatment. The *Stress+* application is also designed for doctors, physiotherapist, sports psychologist, stress counsellors and health insurance organizations to assist their process.

2 Architecture

This chapter describes the architecture of Stress+. The whole architecture was designed in a way that arbitrary stress tests with different modules can be created very easily. It was also considered that programming and adding new modules is very simple. Each stress test consists of a pipeline of screens which are shown to the patient successively in the specified order. Additionally, a stress test can contain overlays which are displayed on top of the screens.

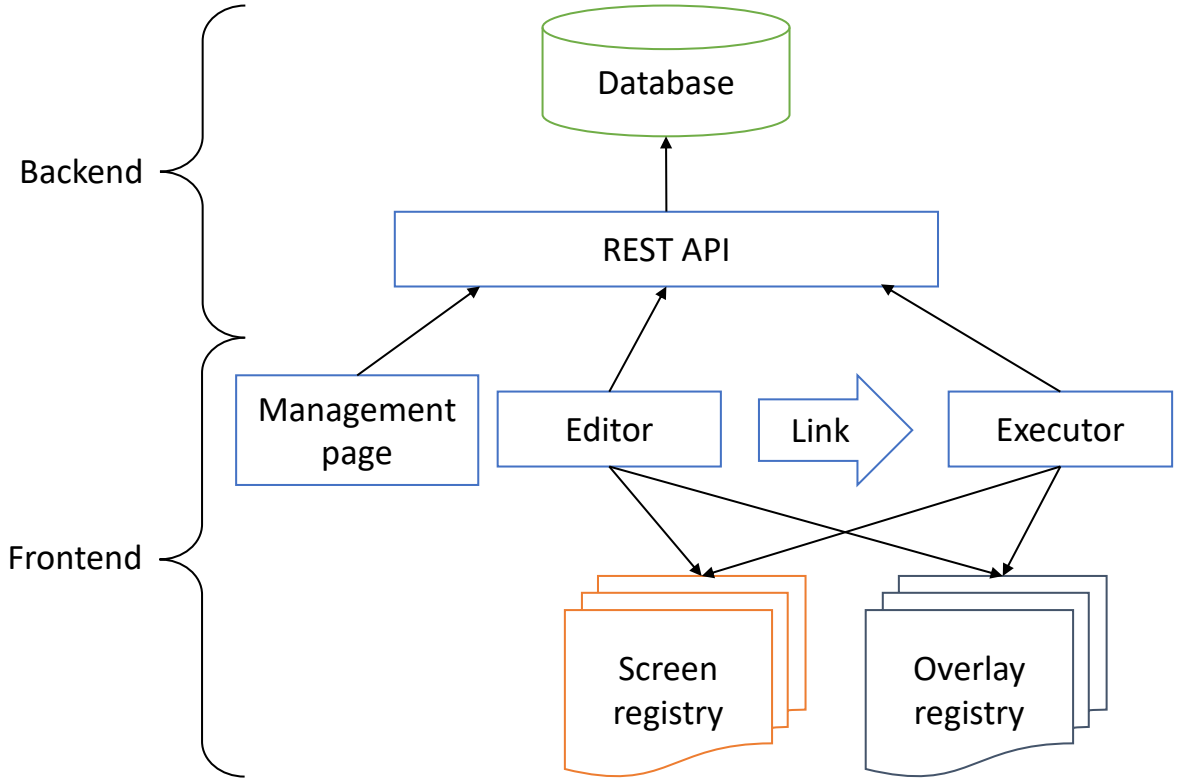

Figure 1: Software architecture

The figure 1 shows the software architecture of Stress+. It is a browser-based application and consists of a frontend and a backend. The main frontend components of Stress+ are the *Editor* for creating and editing stress tests and the *Executor* for executing the stress test. The third frontend component is the *Management page*, on which all available stress tests can be managed. To be able to easily add new modules in the future all available screens and overlays are registered in the *Screen registry* and *Overlay registry*. Therefore, the *Editor* and *Executor* are developed generically without the knowledge of the available modules. They must query the registries to know which modules are currently present. After saving a stress test in the *Editor* a link is generated, which can be sent to the patient so he can execute the stress test.
The backend is responsible for saving the stress test configurations and the statistics on how the patient performed. Therefore, it consists of a REST API, through which the database can be accessed.

2.1 Frontend

The frontend is a single-page application written in JavaScript utilizing the React framework. The following sections describe the different frontend components in more detail.

Screen

The stress test consists of a list of screens that will be displayed successively to the patient. A screen will be displayed full screen inside the user's browser. Each screen has its own settings, which can be adjusted inside the editor. All available screens can be found in chapter 5.

Overlay

The stress test can be equipped with overlays that are displayed on top of the current screen. Because all overlays are displayed simultaneously during the whole stress pipeline execution, they do not have an order. Each overlay has its own settings, which can be adjusted inside the editor. All available overlays can be found in chapter 6.

Management page

On the management page, all available stress tests are displayed. From there you can open a stress test in the editor, delete one or create a new test. Further details about the management page can be found in chapter 3.

Editor

The stress tests can be created and edited in the editor. Also, every setting of the screens and overlays can be adjusted within the editor. Each stress test has a unique id that is generated when it is saved for the first time. With this id a link is generated that can be sent to patients so they can execute the stress test. From the editor, users can also download all recorded statistics for the current stress test. Further details about the editor can be found in chapter 4.

Executor

The executor is responsible for executing the stress test specified by the link. It will display each screen successively to the patient and show all overlays simultaneously during the whole stress test run. The executor also collects records from the screens and persist them in the database. Further details can be found in chapter 7.

2.2 Backend

The backend consists of the Database and the REST API

Database

To save the stress test configurations and the results of a stress test executions a database is used. The data does not have a clear structure, as arbitrary modules can be composed together in a stress test. Therefore, the document-oriented NoSQL database MongoDB is used instead of a relational database.

REST API

The REST API acts as the connection between the frontend and the database. It can be accessed via HTTP and uses JSON documents for transferring the data. The REST API endpoints are written in JavaScript on the NodeJS platform.

3 Management page

To go to the management page, navigate to the home page of Stress+ and click the "Go to the management page" button. A screenshot of the management page can be found in figure 2. All previously saved stress tests are displayed on this page. You can open one of the tests in the editor by clicking on it. Clicking on the trash icon will delete that stress test. To create a new stress test, click the "New" button, which will open the editor with a new empty stress test.

Figure 2: Screenshot of the management page

4 Editor

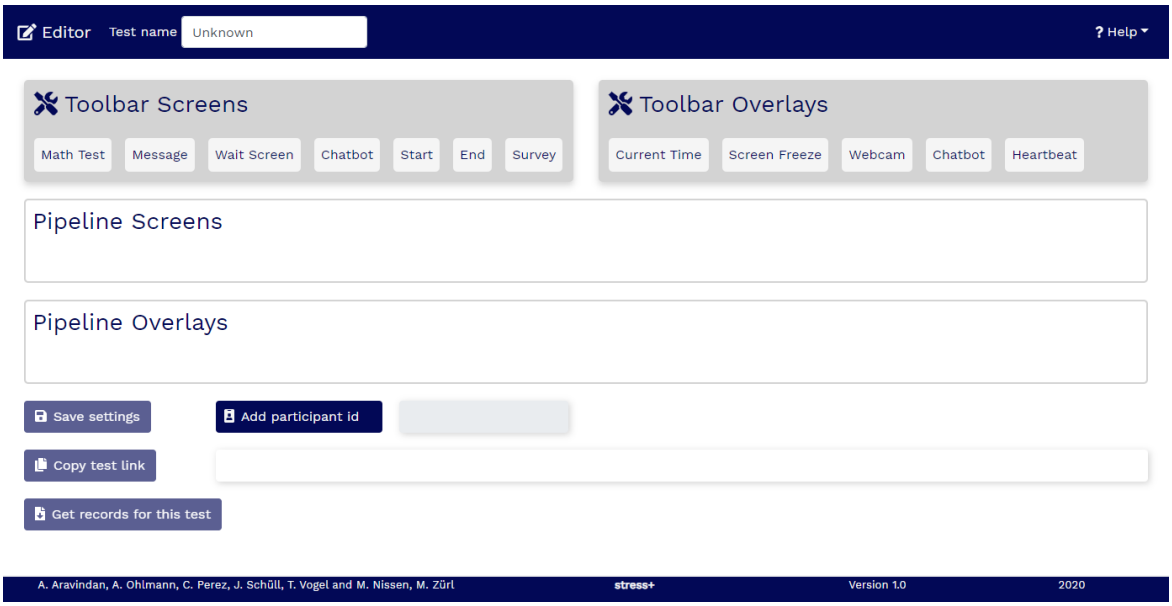

Figure 3: Screenshot of the editor after creating a new stress test

Figure 3 shows a screenshot of the editor with an empty stress test. With the text field on top of the editor, the name of the stress test can be changed. All available screens and overlays are displayed in the corresponding toolbars. From the toolbars, the screen and overlay items can be moved onto the corresponding pipeline with drag and drop. All items in the pipeline will be used when executing this stress test. The stress test can be saved with the "Save settings" button if at least one screen item is present. Each stress test has a unique id that is generated when it is saved for the first time. With this id a link is generated that can be sent to patients so they can perform the stress test. This link is displayed next to the "Copy test link" button, which can be clicked to copy the link to the clipboard.
Figure 4 shows a screenshot of the editor with a simple stress test with the name "My stress test". It consists of a "Start", "Math test" and "End" screen and uses the "Webcam" overlay. By clicking the trash icon in the top right corner of a pipeline item, the item can be deleted from the pipeline. Each pipeline item has a "Settings" button which can be clicked to open the settings of the corresponding screen or overlay.
Clicking on the "Get records for this rest" button will download all collected test statistics of this test in JSON format. The file will contain an array of test executions. To differentiate test executions a participant id can be added in the text field next to the "Add participant id" label. This participant id will be added to the test link and the test execution result, so executions from different participants can be identified easily.

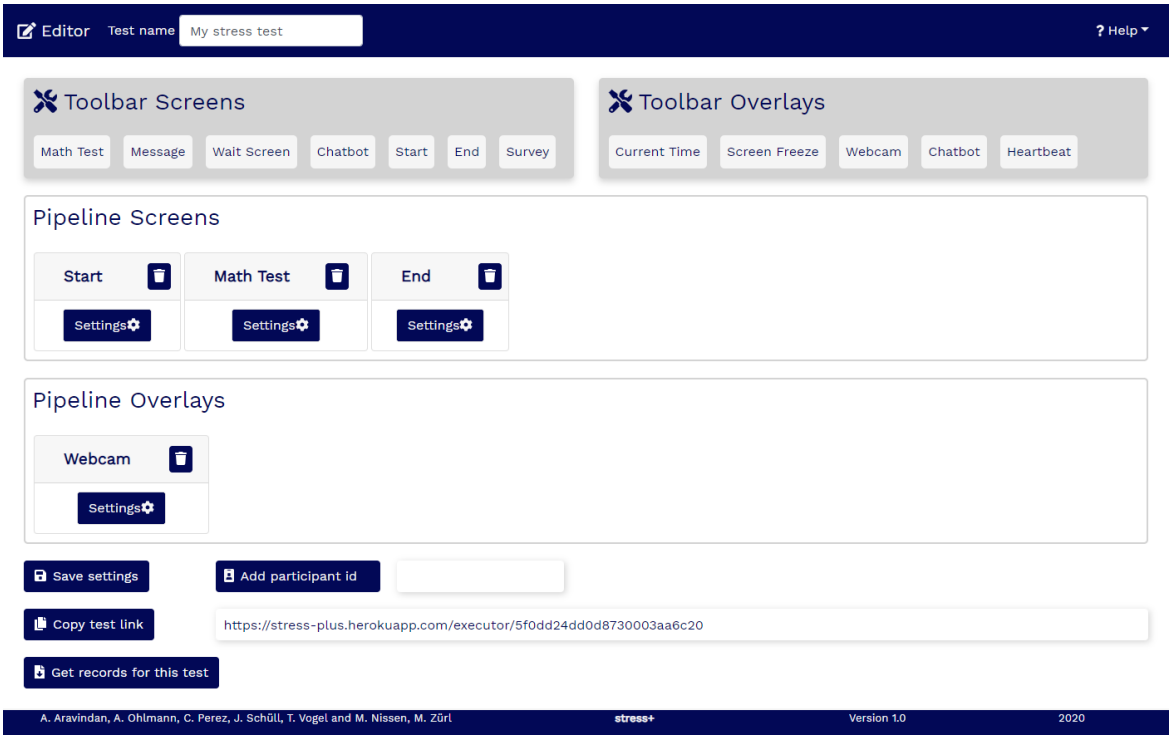

Figure 4: Screenshot of the editor with a simple stress test

5 Screens

This chapter describes all the available screens.

5.1 Math test

The math test screen is the main stress-inducing factor of the stress test. The stress is induced by mathematic questions. For each question, the user has only a limited time to answer the question. If the time is over or the user has given the wrong answer, a negative feedback message is displayed. If the user has answered the math question correctly a positive feedback message is displayed. The feedback message is visible for a short amount of time. After this short pause, a new question is displayed. Math questions are taken out of a pool of 40 arithmetic expressions. Before every math test, the array of questions is mixed randomly.

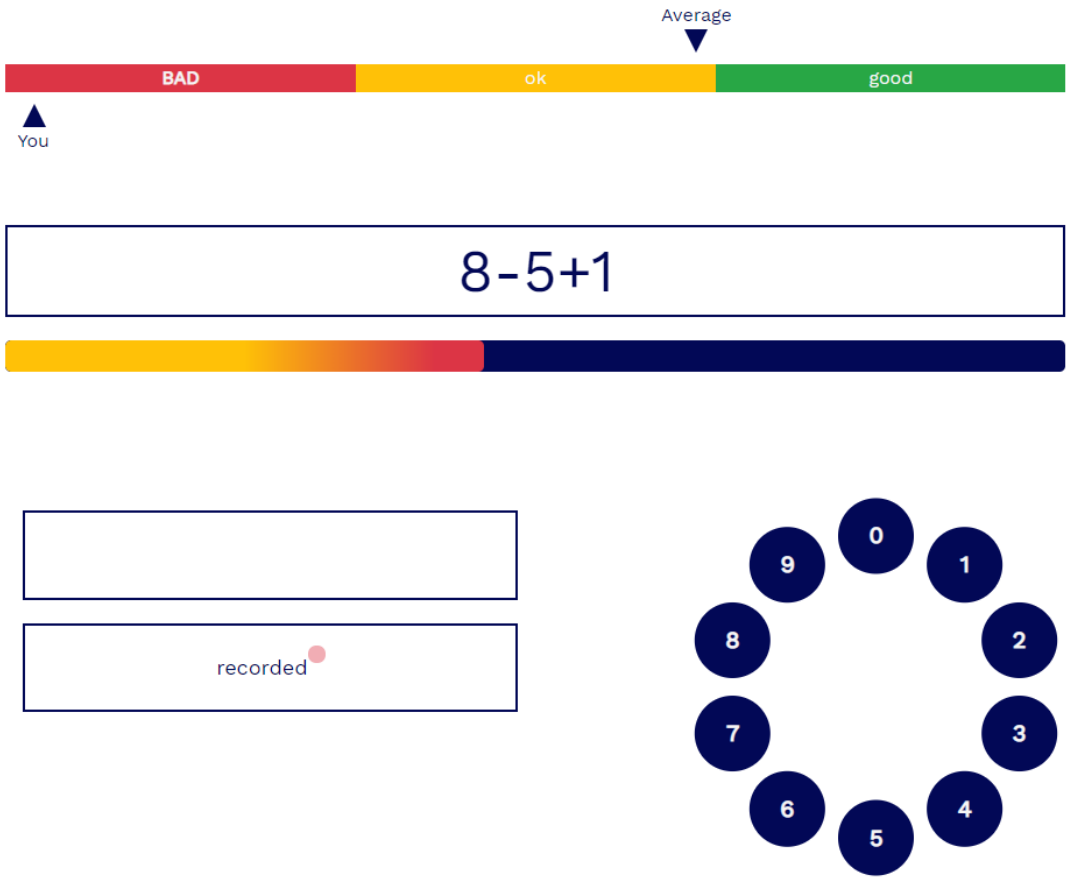

Figure 5: Screenshot of the math test screen

Figure 5 shows a screenshot of the math test screen. At the top, the level bar is displayed, which shows two indicators "You" and "Average". The "You" indicator shows the real performance of the participants. The "Average" indicator is faked. It is increasing if the participant has given a correct answer and is decreasing if the participant has given an incorrect answer. Below the math questions, a progress bar

is displayed. It shows how long the participant has time for the current time. The user must give their answers with the dial pad at the right bottom of the screen. The feedback is given inside the blue box on the left of the screen.

Settings of the math screen

Enable sound If enabled, feedback will be given also via sound. To induce more stress an annoying background sound is played while the user has time to answer the math question. Note: users can turn their volume down or mute their speakers, so do not rely on the sound to be heard.

Activate Control This option enables the control mode. During control mode, the level bar and progress bar are hidden. Also, the annoying background sound will not be played.

Total test time The total duration in seconds of the math test.

Answer timeout The duration in seconds for answering one math question.

Time between questions The duration in seconds of the short pause after the feedback of the current question has been given to the user.

Difficulty The initial difficulty of the math questions. The table 1 lists all possible difficulty levels. The difficulty level will be increased by one level if the participant has answered three questions correctly in a row. The level is decreased by one level if the participant has answered three questions incorrectly in a row.

level	number of operands	possible operators	example
0	2	+, −	$7-2$
1	3	+, −	$2-5+6$
2	3	+, −, *	$3*5-9$
3	4	+, −, *	$2*4-2*3$
4	4	+, −, *, /	$3*6/2-2$

Table 1: Difficulty levels for the math test

5.2 Message

The message screen shows a simple message to the user. The user can go to the next screen by clicking a button on the screen. The title, the message and the button text of this screen can be adjusted.

5.3 Wait Screen

The wait screen shows a simple message to the user, like the Message screen. But the user must wait a specific time until the next screen is displayed. The user cannot manually go to the next screen. The title, the message, and the waiting duration in seconds of this screen can be adjusted.

5.4 Chatbot

This screen provides a very basic chatbot functionality. Stress test creators can define the chat messages the chatbot will show to the participant. After the chatbot has sent a message to the participant, he must send an answer to the chatbot. After the last message from the chatbot, a "Click to continue" button will be displayed, which will take the participant to the next screen. The answer a participant has given will be added to the statistics of this test execution.

5.5 Start

This screen can be used as the first screen. It displays the Stress+ logo and can show a custom message.

5.6 End

This screen can be used as the last screen. It displays the Stress+ logo and can show a custom message.
Note: The end screen has no mechanism to go to the next screen. Therefore, it should not be placed in the middle of the pipeline. If the end screen is not present a blank page will be shown as the last screen.

5.7 Survey

The survey screen can display other websites inside the stress test. This can be useful to display for example a Google Form inside a stress test. A button will always be displayed at the top of the screen to navigate to the next screen.

6 Overlays

This chapter describes all the available screens. For easy positioning of the overlays, an overlay can have the optional property `position` which controls the position of the overlay on the screen.

6.1 Current Time

This overlay just displays the current time with hours, minutes, and seconds.

6.2 Screen Freeze

The screen freeze overlay will block all user input for a configurable amount of time to induce more stress. The freeze will start after a configurable amount of time. The timer will run as soon as the first screen has started and does not consider how long a user may wait until he clicks the next button on the Start or Message screen. Therefore, it cannot be controlled on which screen the freeze happens if a screen with a next button is in the stress test.

6.3 Webcam

The Webcam overlay will display the live recordings of the user's webcam on the screen. To be able to access the webcam the user must give the Stress+ website the permission to do so. Some operating systems might also restrict access to the webcam. It is also generally not possible that two programs access the webcam simultaneously. If the webcam cannot be accessed, the stress test will continue without displaying the live recordings of the webcam.

6.4 Chatbot

This is an overlay version of the Chatbot screen, that can be opened and closed via a button. For further details consult the Chatbot screen documentation.

6.5 Heartbeat

This overlay will display the heart rate next to a heart image with a bumping animation. The heartrate is completely faked and will slightly increase over time. This overlay can be displayed delayed by specifying a duration in seconds after which it will become visible.

7 Executor

The executor is responsible for handling the execution of a stress test. The id of the stress test to be executed is contained in the link generated by the editor. With this id, the executor loads the stress test configuration from the backend and starts displaying the screens and overlays.
The executor is also responsible for collecting and saving all records for the statistics. The records are batched together for each screen. They are all sent together to the backend when the screen changes to the next one. Therefore, records for the end screen will not be sent and saved in the database.
Currently, there are three types of records. The math test screen will add records for each answer of the participant. The chatbot screen will add a record containing the messages from the participant. Thirdly, the mouse cursor position will be added to the records once per second.

8 Deployment

The easiest way to deploy Stress+ is using a Docker container. To build the docker image a `Dockerfile` is present in the root directory. The `Dockerfile` will build the frontend and configure the backend to also serve the frontend. Run the following command in the root directory to build the docker image with the name `stress-plus`.

```
docker build -t stress-plus .
```

After the docker image is built it can be started with:

```
docker run
  --detach
  --name stress-plus
  --publish 80:80
  --env MONGODB_URI=<mongodb-uri>
  stress-plus
```

Stress+ is now running on port 80 of the host machine. By default, the backend will listen on port 80 of the container, but this can be changed by setting the `PORT` environment variable. Because the backend requires a running MongoDB database, you must set the `MONGODB_URI` environment variable. The value should have the following schema:

```
mongodb://<username>:<password>@<host>:<port>/<database-name>
```

For more information on how to run docker containers refer to the docker documentation.

9 Development

This section describes how to develop new screen and overlay modules. All modules have a unique name, a React component that displays the module during the execution of the stress test and a React component to adjust the settings of the module in the editor. Every module must be registered in the corresponding registry and should have its own folder inside the `src/overlays` or `src/screens` directory.

9.1 Setup development environment

Both backend and frontend can be started on your local machine in development mode. Before starting them, make sure that you have installed NodeJS and have executed `npm install` in the `Code/frontend` and the `Code/backend` directory.
In development mode the backend uses the MongoDB database referenced by the `MONGODB_URI` variable in the `.env` file in the backend directory. Please make sure this database is up and running or change the variable to a running MongoDB database. Run `npm run start` in the backend directory to start the backend on port 4000.
To start the frontend run `npm run start` in the frontend directory. Your browser should now be opened automatically and show the start page of Stress+. If that's not happening, navigate to `localhost:3000` in your browser to open Stress+. You can now start developing or play around with Stress+.

9.2 Screen Component

Now, you will learn how to code a screen's main component. This React component is responsible for displaying the screen during the execution of the stress test. This component will receive its `settings` and an `onFinished` callback inside its props. The `settings` object contains the settings configured for this screen. The `onFinished` callback must be called when the screen is finished performing its task, so the pipeline executor can show the next screen. This callback must only be called once. A very simple screen component might look like this:

```
export default function ScreenComponent(props) {
  const {settings, onFinished} = props;
  return (<Button onClick={onFinished}>{settings.text}</Button>);
}
```

9.3 Overlay Component

In this section, you will learn how to code an overlay's main component. This React component is responsible for displaying the overlay during the execution of the stress test. It will receive only its settings inside the props. A very simple overlay that displays a message might look like this:

```
export default function OverlayComponent(props) {
  return (<div>{props.message}</div>);
```

```
}
```

By default, an overlay is configured to not receive any click events, because the current screen must handle them. If an overlay component wants to receive click events it can enable them by adding the `pointer-events: all;` CSS property to an overlay HTML element. But be careful as this can make the underlying screen components unclickable.

Position of overlays

Overlays can be positioned on the whole screen. For easier positioning of overlays, overlays can define an optional `position` property in its settings. This property accepts the following values: `center`, `top`, `bottom`, `right`, `left`, `top-left`, `top-right`, `bottom-left`, `bottom-right`. To make the position property adjustable an OverlayPositionInput component exists. Please refer to the Settings Component and Input Components section for further information.

9.4 Settings Component

In this section you will learn how to create the settings React component for screen and overlay modules. The settings component for screen and overlay modules are working the same way. The settings component will receive the following properties in its props.

- `id`: The unique id of this pipeline element
- `dndType`: The drag and drop type (either `screen` or `overlay`)
- `type`: The name of this module
- `updateSettings` callback: This function must be called to update a settings value. It takes the id of the pipeline element, the name of the settings value and the new value
- all current settings

There are many predefined input components available. For details about them refer to the Input Components section. An example settings component with a number input might look like this:

```
export default function SettingsComponent(props) {
  const onChange =
    (name, value) => props.updateSettings(props.id, name, value);
  return (
    <NumberInput
      name="test"
      label="..."
      value={props.test}
```

```
      onChange={onChange}
    />
  );
}
```

9.5 How to register a module

Each module must be registered in the corresponding registry so it can be used. Screen modules must be registered in the `screen-registry.js` file and overlay modules in the `overlay-registry.js`. Each registry exports a map of modules where the key is the unique module name and the value is the definition of the module. This definition has the following properties:

- `component`: The overlay's or screen's main React component
- `settingsComponent`: The overlay's or screen's settings React component
- `initialSettings`: An object containing the initial/default settings

Here is an example registry file with one module named `moduleName`:

```
export default {
  moduleName: {
    component: ModuleMainComponent,
    initialSettings: {
      test: "Foo"
    },
    settingsComponent: ModuleSettingsComponent,
  },
};
```

9.6 Input Components

Input component can be used to create settings components for the modules. One input component can be used to change one settings property. There are some input components already defined and they all have the following properties:

- `name`: the name of the settings property this input component controls
- `value`: the current value of the settings property
- `onChange`: callback that is called when the value changed. The callback will be called with two arguments. The first is the name of the property (passed via the `name` property) and the second is the new value.
- `label`: The description of the settings property controlled by this input component.

In the following example for a NumberInput, you can see that the name `testTotalTime` matches the corresponding setting name read from `props.testTotalTime`:

```
<NumberInput
  name="testTotalTime"
  label={t("settings.totalTime")}
  value={props.testTotalTime}
  onChange={onChange}
/>
```

Input validation

To validate the input of the users, validation attributes can be added to the input component. If the value of the input is not valid, an error message will be displayed and the input component will get surrounded by a red box. The following validation attributes are available:

- `required`: The value cannot be empty.
- `minLength`: Only for text values: The text length must be greater or equal to the attribute's value.
- `maxLength`: Only for text values: The text length must be lower or equal to the attribute's value.
- `pattern`: Only for text values: The text must match the given regular expression.
- `min`: Only for number values: The number must be greater or equal to the attribute's value.
- `max`: Only for number values: The number must be lower or equal to the attribute's value.

The following example shows the usage of the input validation attributes. This NumberInput cannot be empty and only integers between -5 and 5 (both inclusive) are considered as valid. Other attributes of the input component are omitted for brevity.

```
<NumberInput required min={-5} max={5} />
```

Available input components

Checkbox Input component for a boolean value.

MultilineTextInput Input component for a multiline text value. There also exists the TextInput component which accepts only single line text.

NumberInput Input component for an integer value. It also accepts negative integers.

SelectInput Input component to select one value from a list of given values. The array of possible values must be passed to the select input via the `values` property.

TextInput Input component for a single-line text value. It does not accept the newline character. There also exists the MultilineTextInput component, which accepts multiline text values.

OverlayPositionInput Input component to adjust the position property for overlays. This component acts as a SelectInput with the list of position values already configured.

10 Development Tools

This section describes all used third party tools and libraries. First all common tools, then the frontend libraries and finally all backend libraries are listed. Currently, the project uses the GitLab Server of the MaD Lab.

10.1 Common tools

Git repository Currently, the project uses the GitLab Server of the MaD Lab.

NodeJS NodeJS is a JavaScript runtime that must be installed to develop, test and build the project. It is also required for running the backend.

Linter ESLint statically analyzes our code to find common problems (for example accessing an undefined variable). ESLint is configured with the `.eslintrc.js` file. More rules can be added to ESLint via plugins. The following plugins are used:

- eslint-plugin-import: This plugin intends to support linting of ES2015+ (ES6+) import/export syntax and prevent issues with misspelling of file paths and import names. It also enforces the ordering of import statements.
- eslint-plugin-prettier: This plugin runs our Code Formatter prettier inside of ESLint. It also uses the eslint-config-prettier configuration to disable rules of ESLint which conflict with prettier.
- eslint-plugin-jsx-a11y: This plugin adds rules for React's JSX syntax.
- eslint-plugin-react: This plugin adds react specific rules.
- eslint-plugin-react-hooks: This plugin enforces React's Rules of Hooks.

Code Formatter Prettier is used to have a consistent code formatting in your code. To format the code run the `npm run lint-fix` command.

Continous Integration (CI) We are using GitLab CI to run the linter, the tests and the build on every push onto the GitLab Server. The jobs are configured inside the `.gitlab-ci.yml` file. Every job must have the tag `docker` to be executed.

10.2 Frontend libraries

Frontend Javascript framework The frontend is written in Javascript and uses the React framework. The project was initially created with the Create React App tool.

CSS framework Bootstrap is used as a CSS library for nice UI Elements. To use the bootstrap components like normal React components the library react-bootstrap is used.

Router React-router-dom is used to provide support for handling different routes in our frontend (e.g. `/` and `/editor`).

Icons Fontawesome provides lots of free icons.

HTTP client We use Axios as our HTTP client. It is used to access our REST API of the backend, to save and load stress test configurations.

Drag and drop react-beautiful-dnd provides beautiful and accessible drag and drop for lists. It is used extensively in our editor.

Translation I18next is an internationalization-framework written in and for JavaScript, which we use to provide translations. The react-i18next library adds React specific components like React Hooks for i18next. Currently, only English is available as language. The translations are specified inside `public/locales/en/stress-app.json` and are loaded via HTTP with the i18next-http-backend library.
The following example shows the usage of `useTranslation()` hook from react-i18next. This hook returns a `t` function, which takes a translation key as an argument and returns the translation.

```
import {useTranslation} from "react-i18next";
function Component() {
  const {t} = useTranslation();
  return <span>{t("title")}</span>;
}
```

Copy to clipboard The library react-copy-to-clipboard is used to easily copy content to the clipboard.

Generating ids The library uuid is used to create RFC4122 UUIDs. These unique ids are used for identifying each screen and overlay in a stress pipeline.

Tests For frontend tests, we use the tools that are preconfigured by create-react-app. It uses jest as a javascript testing framework and the testing library as a testing utility. @testing-library/react adds support for testing react components, @testing-library/jest-dom adds custom DOM element matchers for jest and @testing-library/user-event adds advanced browser interactions such as clicking and typing.
Tests should be placed in the same folder as the file to test and the file extension for tests must be `.test.js`.

precompress The precompress tool generates gzip and brotli compressed file for static web servers.

10.3 Backend libraries

Web framework The backend uses Express as the web framework for the REST API. The REST API uses JSON to exchange data. The body-parser library is used parse the JSON body of HTTP requests. The compression library is used to add gzip compression to responses to save bandwith. To be able to serve the pre-compressed static files of the frontend the express-static-gzip library is used.

Database To access the database the official mongodb driver for NodeJS is used. Also the mongodb-client-encryption library must be added.

Bundler The JavaScript module bundler rollup is used to create a single file, that can be executed. In development the @rollup/plugin-run is used to start a development server. In production the rollup-plugin-terser is used to execute the JavaScript mangler and compressor terser.