Development of a web-based application for the degradation configuration of IoT components

Project Report

Steffen Knoblauch `steffen.knoblauch@uni-ulm.de`

July 11, 2021

Contents

1 Introduction

More and more IoT (Internet of Things) devices are finding their way into our daily lives, making it possible to combine a wide variety of end devices or sensors and design new composite systems. Such systems can encounter internal problems, for example, when a component no longer operates properly. The system as such must react to this and, for example, cease functionality or operate in another way.

This is where the SORRIR project comes in to develop a self-organising, resilient execution platform for IoT services. In order to make such a system resilient, i.e. resistant to errors of the subcomponents of the system, different levels were introduced in which such a system can operate.

A system that is composed of several subcomponents may be forced to change the level if, for example, one of these subcomponents is no longer functional. In the worst case, up to the complete deactivation of the system. In order to prevent a system from going directly into the deactivated state whenever an error occurs, various intermediate levels, the so-called degradation levels, can be defined. For example, a degradation level can continue to provide some of the original functionality, but with some limitations depending on the non-functional subcomponent.

An example of this is a smart barrier in a parking garage, which in the ideal case, i.e. when all subcomponents are functioning, recognizes a car by means of a camera and is opened automatically. If, for some reason, the camera failed, the system would no longer be able to function. However, if one uses the degradation levels, a part of the functionality could be guaranteed. For example, by opening the barrier manually with an ID card. To configure such degradation levels a json file is required which defines the different transitions between the levels. For example, what should happen when a certain subcomponent is no longer functional.

This is the starting point of this project, in which the goal is to create a graphical interface in form of a web application that allows to configure a system with the different degradation levels and finally to export it as Json. In order to change or create a configuration, several components for the web application are required, like the definition of subcomponents or the import of an existing configuration.

This report starts with a short description of the technology stack, in section 2, that was used to implement the web interface. It is followed by the architecture of the application, in section3, that includes a description of the general components, their usage, and some additional information about the implementation. The last section is the conclusion that contains a short summary as well as potential further enhancements.

2 Technology Stack

The goal of this project is the development of a web application and due to the large number of different frameworks and libraries, the selection was relevant from the beginning. To develop a web application you can work without a framework or library, but nowadays there are also various frameworks available to make the work easier, such as React, Vue or Angular.

The choice for this application fell on React, because React is not only extremely popular [3], but it also allowed me to learn the required fundamentals and use it for the first time. React is a JavaScript library to create single-page applications, but also provides typings to use it with TypeScript [2]. In order to facilitate future development and enhancements for this applcation, the choice fell on TypeScript. In addition to this, TypeScript allowed to define interfaces that help to understand the model of the application.

With this technologies as base, the next step is the decision is related to the build system. It would be possible to use a custom build system based on webpack or similar bundlers, but for React there is an officially supported way: create-react-app. This is a npx package that will setup a single-page application with all the required configuration, like the bundler or the typescript compiler. It is possible to later exchange and adapt parts of the configuration or the build system in general, but for this project it was not required to further adapt the default setup.

In addition to react some additional libraries were used:

- **material-ui**: a library that offers standard material design components
- **react-syntax-highlighter**: a syntax highlighter that is used to display the configuration as Json in the import and export
- **ajv**: a validator that is used in the import to validate that the Json fits the configuration schema
- **react-dnd**: a drag and drop library for react

3 Architecture

This section will give an overview of the general concepts and will show with examples how to use the application.

3.1 General concepts

The complete code was structured with too general ideas in mind. On the one hand it should be modular in the sense that the application consists of components that can be reused on multiple occasions. For example a dialog for the deletion can be always the same and only the handling and the description text have to be adapted based on the usage. The modularity will be introduced implicitly by React, because the general idea is the creation of components that will be updated internally by react.

Another relevant aspect was the future extension of the application, in order to be able to integrate more steps for the creation of the configuration. For example adding additional properties for the subcomponent configuration.

3.2 App-Component

The App is the central component that will contain all the relevant components for the application. The App keeps track of the current configuration and provides the access for the components via a context provider. This allows the access and update of the configuration from all the components, which will also trigger the React lifecycle to re-render a component.

3.3 Welcome page

The welcome page is the first component that will be displayed when you start the application. In figure 1 you can see that there are two buttons that allow to start the configuration. There are two choices on how to start the configuration, the start with an import and without an import, based on the choice a user makes the wizard (compare section 3.4) will be started with different steps.

3.4 Wizard

The creation of the configuration can be divided into several steps. To facilitate the configuration, a wizard has been developed that allows the step-by-step configuration. In addition to that it is also possible to navigate arbitrary back and forth between the different steps of the configuration.

The wizard component allows to dynamically configure what steps will be shown and in what order. The wizard consist in general of a menu-bar, a stepper to show navigate between the steps and the content it will show which is internally called a view.

Figure 2 shows the menu bar and the stepper for the configuration with an import. You can navigate through the different steps by clicking on the steps in the stepper,

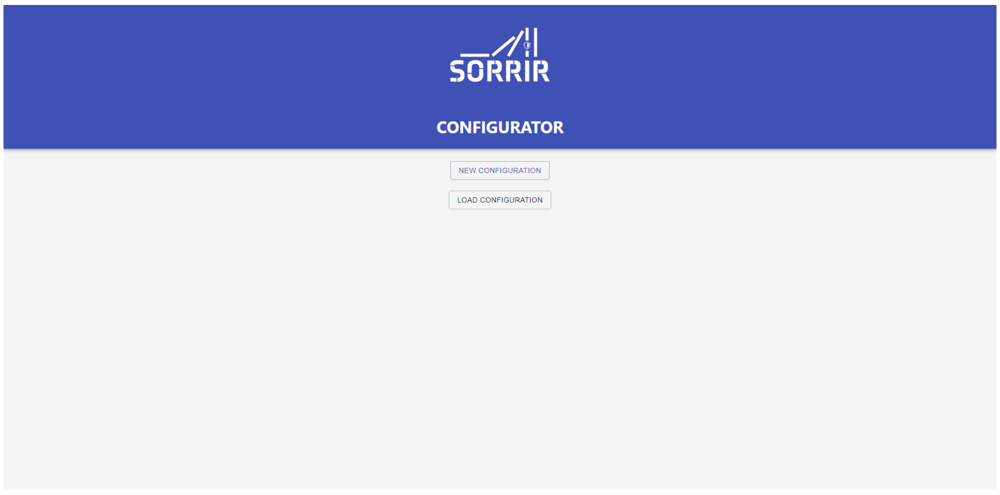


Figure 1: Welcome page

whereby the active step is highlighted. The menu bar will update its label with the label that was specified for the current step, for example in the figure the label was set to "Import".

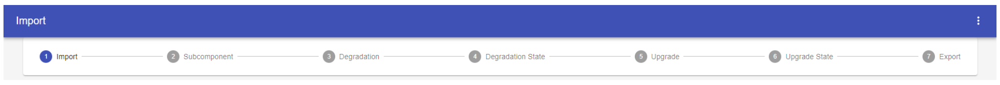


Figure 2: Wizard

In addition to the navigation between the different steps there is a menu on the top right. At the moment this only has one menu item that allows to restart the configuration, but it is possible to further add more functionality. The restart will show a dialog to confirm the reset of the current configuration and will, after the confirmation, send the user back to the welcome page.

In order to extend the wizard with additional steps following actions have to take place:

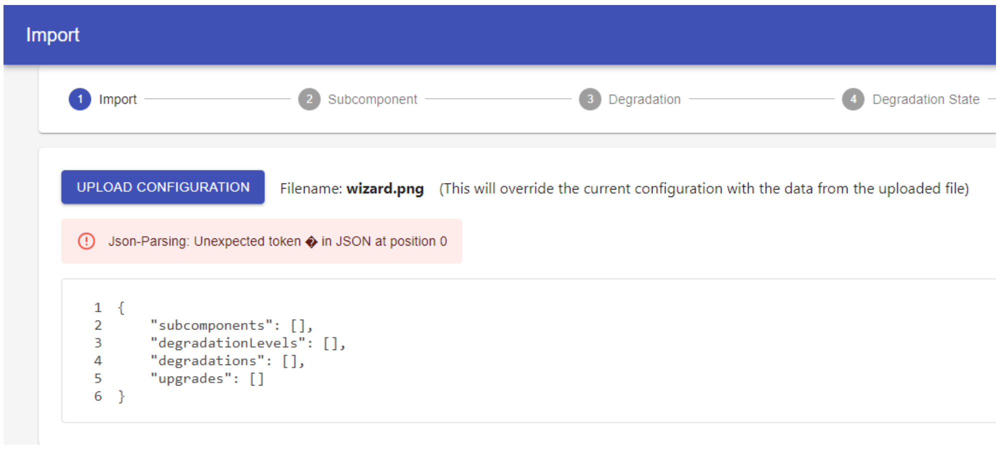
- Extension of the view enum (a enum that contains all available views in the application) ("src\util\Views.ts")
- Setting a label for the new view ("src\util\ViewLabelResolver.ts")
- Adding the view that will be displayed for the step in the wizard. This has to be a react component ("src\components\wizard\ViewSelector\ViewSelector.tsx")

After these steps the wizard can be configured to display the new view as an additional step.

3.5 Import

The import view is only inserted as a step when the wizard is started with the "LOAD CONFIGURATION" button on the welcome page. The import page allows to import an existing configuration file and will update the current configuration in the app. The import will also override all previous changes in the configuration. The configuration will be displayed with syntax highlighter that will be updated with the new configuration after a successful import.

There are several cases when the import has to fail to enforce a correct state of the configuration and therefore it will be validated that file contains a valid configuration. If this is not the case the import will display all errors that occurred during the import validation. The first part of the validation is to check if the file contains a string that can be parsed to a JSON object. if this is not the case an error message similar to the message in figure 3 will be displayed.



Filename: **wizard.png** (This will override the current configuration with the data from the uploaded file)

Json-Parsing: Unexpected token � in JSON at position 0

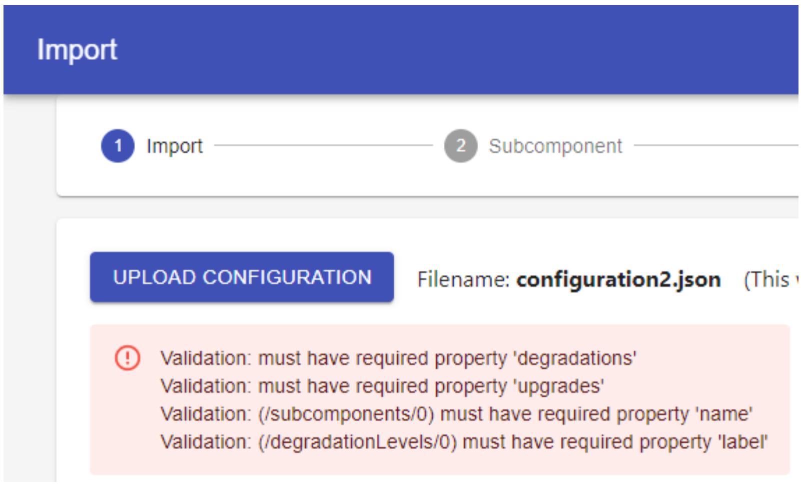
```
{
    "subcomponents": [],
    "degradationLevels": [],
    "degradations": [],
    "upgrades": []
}
```

Figure 3: Import invalid file

The second part of the validation is the validation of the JSON data itself. The JSON data requires several properties to be a valid configuration. Details on this part of the validation can be found in section 3.6. The found problems of the second part of the validation will also be displayed. An example can be seen in figure 4 where there are several problems with the imported object. If a problem occurs for an element of an array, for example in the subcomponents, the index of the element will also be added to the error message. This will help a user to identify the problem with the imported file and update it accordingly.

3.6 Validation

The validation of the configuration at the import is a central part of the application, because it enforces a valid configuration. Another validation is not necessary because



Filename: **configuration2.json** (This

Validation: must have required property 'degradations'
Validation: must have required property 'upgrades'
Validation: (/subcomponents/0) must have required property 'name'
Validation: (/degradationLevels/0) must have required property 'label'

Figure 4: Import invalid JSON

all current configuration steps will always update the configuration correct and end up in a valid state.

For the validation the *ajv JSON schema validator*[1] was used, because it has several advantages over a custom implementation. A custom implementation requires not only to write the complete parsing logic, but also has to be adapted if changes to the configuration schema are required in the future. The configuration schema is a description of all the properties a JSON data has to have to be a valid configuration.

There are two different schema languages that allow the definition of JSON data. On the one hand there is the *JSON schema*[2] and on the other hand there is the *JSON Type Definition*[3]. Both schema description languages are supported by ajv, but the decision was made in favor of the JSON schema. Although there is an RFC for the JSON Type definition and only a draft for JSON Schema, industry support is not yet entirely in place [1]. Since both are supported by ajv, the decision was made to use JSON Schema because of the better support, but this can easily be changed to the JSON Type Definition.

The change from the JSON Schema to the JSON Type Definition is for this application quiet simple, because you just have to replace the schema file and adapt the configuration of ajv, which can be found in src\util\

3.6.1 JSON schema

This section provides a short overview of the configuration schema, but the complete definition can be found in the source code.

[1] `https://github.com/ajv-validator/ajv`
[2] `https://json-schema.org/`
[3] `https://datatracker.ietf.org/doc/html/rfc8927`

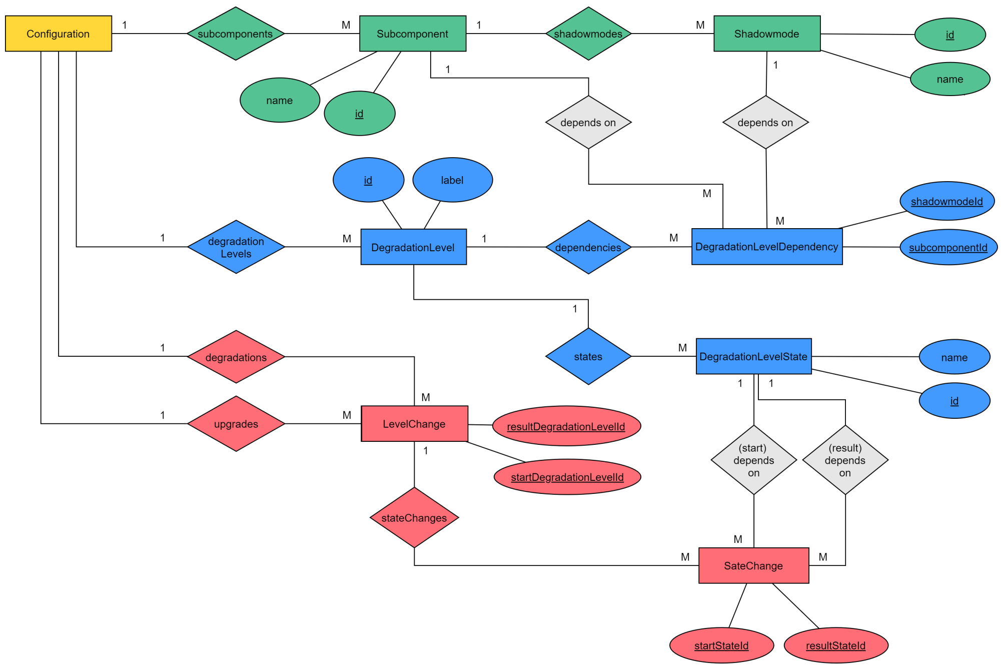


Figure 5: Configuration diagram

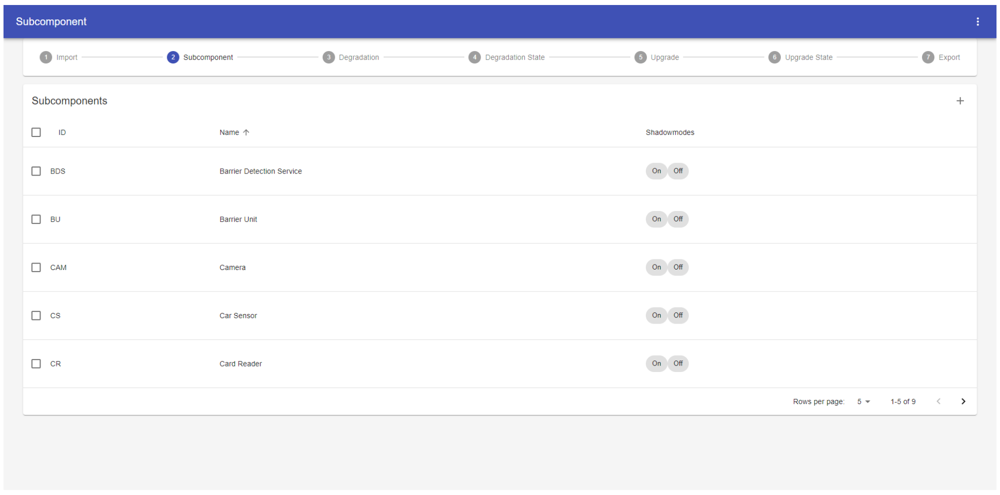
The first part of the configuration are the subcomponents. A system that has to be configured consists of several subcomponents, these are stored in an array in the configuration. The subcomponents are highlighted in green in figure 5 a together with the shadowmodes. Each subcomponent has a set of shadowmodes that are all the operational states a subcomponent can have. For example the typical states are an *on*-state or an *off*-state.

The next part of the configuration are the degradation levels which represent the levels in which the system can be. In figure 5 the blue highlighted parts are all relevant parts that are related to the degradation levels. The degradation level dependencies are a set of all the states the subcomponents have to be in for a degradation level or in other words the dependencies of a degradation level. Only the subcomponents and the states that are relevant for a level are stored in this array and the *don't care* cases will be ignored to save storage space. The degradation level states are the internal states of each degradation level. They describe the states the state machine of each degradation level can be in.

The central part of the configuration are the upgrade and degradation of the system, in figure 5 highlighted in red. Both the upgrade and the downgrade from on degradation level to another degradation level can be represented as level change. A level change object consists of the degradation level where the change starts and where it results in. A simple example could be a system that starts at the off-state and after an upgrade results in the on-state. In addition to the level change, it is also required to define in what internal state the start and the result degradation level has to be. In order to map the states each level change contains a set of mapping of the states of the start and result degradation level.

3.7 Subcomponent configuration

The subcomponent configuration is the first step when you start the wizard without importing an existing configuration. Otherwise it is the second step that allows the adjustment of the imported subcomponents. This component offers a table that displays all defined subcomponents and allows to sort and navigate them and in addition to this the general creation, update or deletion of them. In figure 6 an example is shown with several defined subcomponents.



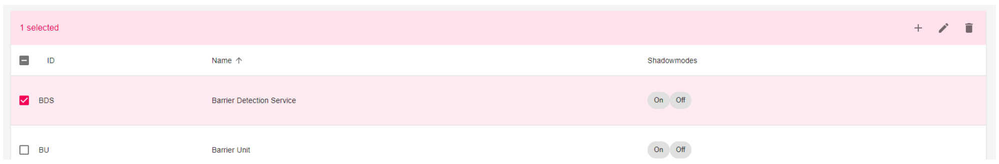
Subcomponents

ID	Name ↑	Shadowmodes
BDS	Barrier Detection Service	On Off
BU	Barrier Unit	On Off
CAM	Camera	On Off
CS	Car Sensor	On Off
CR	Card Reader	On Off

Rows per page: 5 1-5 of 9

Figure 6: Subcomponent view

The table allows to select subcomponents and depending on the amount of selected subcomponents different actions are possible. If only one subcomponent is selected it is possible to either delete, edit the selected one or create a new subcomponent (compare figure 7 on the top right corner of the table).

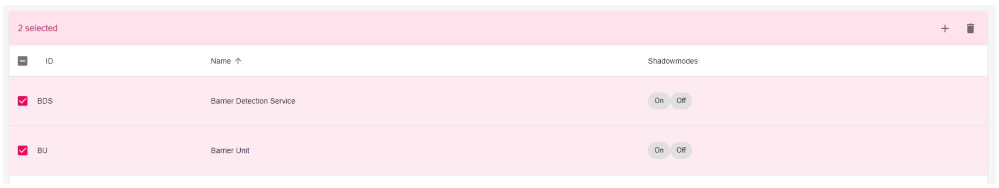


1 selected

	ID	Name ↑	Shadowmodes
✓	BDS	Barrier Detection Service	On Off
	BU	Barrier Unit	On Off

Figure 7: Subcomponent single selection

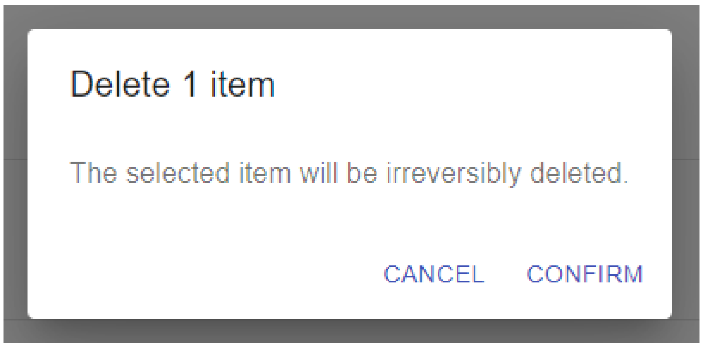
If a user selects multiple subcomponents it is only possible to delete all of them or to create a new one. (compare figure 8).


2 selected

	ID	Name ↑	Shadowmodes
✓	BDS	Barrier Detection Service	On Off
✓	BU	Barrier Unit	On Off

Figure 8: Subcomponent multi selection

For the deletion, the creation and the update of a subcomponent different dialogs are displayed. The deletion dialog is shown in figure 9 and shows the amount of deleted subcomponents. The deletion of one or more subcomponent will also delete the references that exist in other components. For example a degradation level depends on a subcomponent and therefore has a degradation level dependency object that will be deleted too.


Delete 1 item

The selected item will be irreversibly deleted.

Figure 9: Subcomponent delete dialog

The create and edit dialog will allow to create or edit a subcomponent. If no name is set for a subcomponent the dialog will autofill the id with the value of the name. In addition to this the dialog will validate if there are empty fields and if another subcomponent already exists with the id set in the dialog. This can be the case when either a new component is created with an existing id or the id of an existing one is changed. Changing the id of an existing subcomponent will also update the dependencies of the degradation levels that depend on this subcomponent. An example of the create dialog is shown in figure 10. For each subcomponent it is required to define one or more shadowmode and therefore a custom input was designed. This input allows to enter a value in the text field and after the "enter" button is pressed the value will be added as chip on the right side. For example the *On* or *Off* shadowmodes in figure 10. Deleting a shadowmode will also update the degradation level dependencies that rely on this shadowmode.

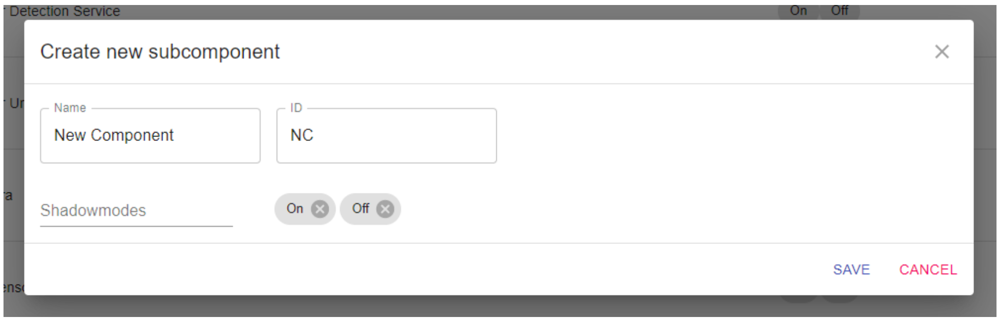


Figure 10: Subcomponent create dialog

3.8 Degradation Level Management

The creation, update and deletion of degradation levels is combined with the functionality to define upgrades and degradations, as described in section 3.9. The degradtion level tree allows to select single and multiple degradation levels and depending on that there are different actions possible. If only one is selected there are three possible actions: creating a new degradation level, edit the selected one or delete the selected one (compare figure 11). The selection of multiple degradation levels (compare figure 12) allows only the deletion of the selected levels or the creation of a new level. The deletion for both cases will display a dialog similar to the one shown in figure 9.

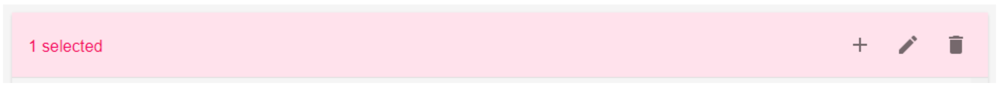


Figure 11: Degradation level single selection

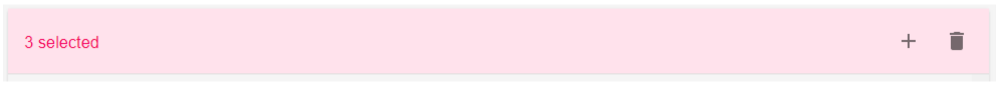
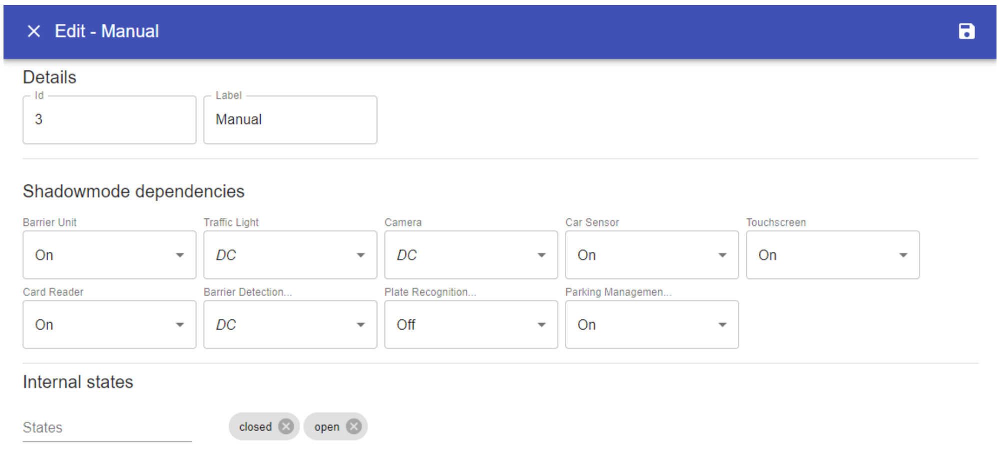


Figure 12: Degradation level multi selection

The creation and edit dialog (shown in figure 13) allows to either create or edit a degradation level. The dialog will validate the id to enforce unique ids for each degradation level in the configuration. Depending on the subcomponents that are already defined, the dropdown selection inputs will be generated dynamically. These dropdown selection inputs allow to define the dependencies of the degradation level by selecting a shadowmode for each subcomponent. If the shadowmode is set to *DC* (don't care) then no value will be created in the configuration (or an existing value will be deleted in the edit case). The internal states are a custom input that allows again to add a state for degradation level by entering a string and pressing enter. After the enter button was pressed the state will be added as chip on the right and can be deleted with the X-button. The deletion of the states of a degradation level will also trigger an update of the configuration and will remove all references that require the state (the state changes).


Edit - Manual

Details

Id: 3

Label: Manual

Shadowmode dependencies

Barrier Unit: On	Traffic Light: DC	Camera: DC	Car Sensor: On	Touchscreen: On
Card Reader: On	Barrier Detection...: DC	Plate Recognition...: Off	Parking Managemen...: On	

Internal states

States: closed, open

Figure 13: Degradation level dialog

3.9 Degradation and Upgrade configuration

- Custom Graph View for the Hierarchy (Recursive generation of the Graph)
- Drag and Drop Support to update the Graph (degradation level hierarchy)
- Off state as default node
- Explain model on how to save the hierarchy (LevelChange model)
- shows all levels even the ones that are not inserted in the hierarchy yet
- Upgrade: Inverse of the Degradation Graph - same functionality as Degradation Graph
- Both will be saved separately in the configuration and are also separate steps in the wizard

3.10 LevelChange configuration

- configure the state in which the level after the degradation or upgrade is
- separate step in the wizard for both degradation and uprade
- shows for all LevelChanges the states of the corresponding level and allows the selection of the state the level will be in after the degradation/upgrade

3.11 File structure

Explain in short the structure (folders/files/...) of the project

4 Conclusion

References

[1] ajv contributors. Choosing schema language, 2021. URL https://ajv.js.org/guide/schema-language.html#comparison. (last visited: 2021-07-09).

[2] Facebook. Adding typescript, 2021. URL https://create-react-app.dev/docs/adding-typescript/. (last visited: 2021-07-09).

[3] Stack Overflow. Stack overflow developer survey 2020, 2021. URL https://insights.stackoverflow.com/survey/2020. (last visited: 2021-07-09).