MOD03: JSON-Download

TINF22F, Software Engineering I
Praxisproject 2023/24

Project

AAS-Webclient

Customer

Markus Rentschler, Ivan Bogicevic

Lerchenstraße 1, 70178 Stuttgart

Supplier

Armin Taktar (Project Leader)

Lara Lorke (System Architect)

David Bauer (Product Manager)

Ümmühan Ay (Documentation)

Rafael Sancho Pernas (Developer)

Kyle Zieher (Test Manager)

Author

Rafael Sancho Pernas 06.05.2024

CONTENTS

1 Scope

According to the first functional requirement, the AAS-Webclient should provide the opportunity to download the JSON data of the asset with clicking on a button.

2 Glossary

- AAS: Asset Administration Shell
- AASX: file format to store an asset

3 Module Requirements

3.1 User View

When the user clicks on the asset the webclient should show a overview of the information of the asset. In this overview there should be a button for the user, where he can download the JSON of the asset.

3.2 Requirements

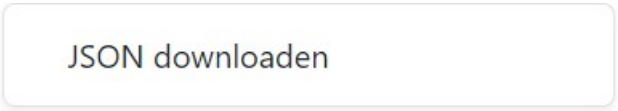
- FR01: Adding AASX-Servers via URL (JSON-Download)

4 Analysis

The function responsible for passing the data for the assets is called getFullShellData in the file backend.js. To make the JSON available for download from the asset, it needs to be returned in the background by this function so that the frontend does not process it. Regardless of the mode, this function forwards the JSON, and then it's made available for download upon button press.

5 Design

The button for the JSON download is positioned below the button for the assetsubmodels and has the same design:



6 Implementation

The first step was adding the button for the JSON Dowmload to the assetBody.js file. This file provides the overview of the asset data for the user.
The code for this looks like this:

```
<div className="d-flex flex-column navigation-buttons">
    <div onClick={jsonDownload} className="navigation-button my-2 shadow-sm
border rounded">JSON downloaden</div>
</div>
```

The JSON information is returned within the asset under ".hide". Initially, there was an issue with the backend.js file where inserting the assetJSON directly caused it to be rendered in the frontend by assetBody.js. Therefore, the JSON of the asset had to be structured in a way that there was a page specifically for reading, rendered by assetBody.js, which was found under ".read". The assetJSON was then hidden before assetBody.js and could be accessed for download under ".hide.assetJSON":

```
return {
    read:{
        idShort: element.idShort,
        id: id,
        idEncoded: btoa(id),
        apiVersion: apiVersion,
        globalAssetId: element.assetInformation.globalAssetId,
        Assetsubmodels: submodelIds
    },
    hide: {
        assetJSON:element
    }


};
```

The function of the download is based on the Blob-function of JavaScript and "link.download" to download the JSON.

```
function jsonDownload(){
    const jsonData = shellBody.hide.assetJSON;

    // JSON in einen Blob umwandeln
    const jsonBlob = new Blob([JSON.stringify(jsonData, null, 2)], {type:
'application/json'});

    // Blob als URL erstellen
    const url = window.URL.createObjectURL(jsonBlob);

    // Link erstellen und klicken, um den Download zu starten
    const link = document.createElement('a');
    link.href = url;
    link.download = 'data.json'; // Dateiname für den Download
    document.body.appendChild(link);
    link.click();

    // Link wieder entfernen
    document.body.removeChild(link);
    window.URL.revokeObjectURL(url);
}
```

7 Module Tests

The testing of this module is carried out through usability tests. When you press the button the expected outcome is the download the correct JSON of the asset. In the test you should compare the JSON of the Download to the JSON of the AASX-Server.