MOD01: Server connection

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

The AAS web client is a display interface for presenting the data taken from an AASX server. Therefore, correct connection and proper JSON processing are very important.

2 Glossary

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

3 Module Requirements

3.1 User View

The user has a selection of AASX servers, which they can choose from through a dropdown menu. By clicking on the URL, the connection should be established in the background, and the data should be visually available for the user to view.

3.2 Requirements

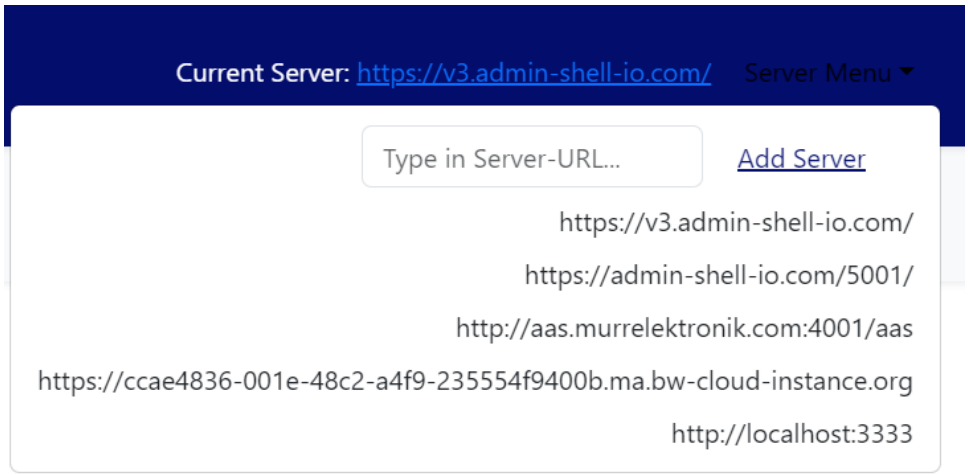
REQ/ Adding AASX-Servers via URL (including Server-Validation)

4 Analysis

The task of the server connection is to take the JSON from the AASX server and process it correctly. The previous project had an issue with the map function, causing the connection to continuously load for the user and outputting in the console that the map function is not working. This needs to be addressed to ensure correct processing.

5 Design

When the User clicks on the Dropdownmenu, a list of each available server will be shown:



6 Implementation

In the file backend.js with the function getFullShellData the JSON from the AASX-server is loaded and processed.
In the previous project, the response was supposed to be processed using the Map function. However, the issue was that the response wasn't an array object. Thus, the Object.entries function was used to create a two-dimensional array. The information about the assets was located at a specific index in the array. Once this index was found, the data could be processed for visualization.

Codesnippet:
```
const entries= Object.entries(response)
return entries[0][1].map(element => {
```

The getFullShellData function returns for each Asset in the detailed mode this format:
```
return {
    idShort: element.idShort,
    id: id,
    idEncoded: btoa(id),
    apiVersion: apiVersion,
    globalAssetId: element.assetInformation.globalAssetId,
    Assetsubmodels: submodelIds
};
```

Sometimes the Server is down. So for safety reasons we implemented an own API. We took the JSON from the AASX-Server, copied it on a txt-file. We load the content of the txt and provide it via express.js as REST-API in the localhost for the Webclient.

7 Module Tests

Testing for this module is conducted through usability tests. In this process, a server is selected. The expected outcome is that the data from the selected server reaches the web client without causing a crash.