System Requirement Specification

Edit New Page Jump to bottom

Jannik Schwarz edited this page 16 hours ago · 12 revisions

Table of contents

- Change log
- Introduction
  - Product Environment
  - Use Cases
    - <UC.001> Search Devices
    - <UC.002> Display detailed data of device
    - <UC.003> Save changes to device configuration
    - The Use Case Diagram
- Product Requirements
  - /LF10/
  - /LF20/
- Product Data
  - /LD10/ User
  - /LD20/ Frontend
  - /LD30/ Backend
  - /LD40/ Devices
- Non-functional Requirements
- References
- Glossary

Change log

Date	Author	Comment
01.11.2019	Jannik Schwarz	Created the Wiki-Page
03.11.2019	Jannik Schwarz	Added use cases, the first issue of the product environment, product data

Date	Author	Comment
04.11.2019	Jannik Schwarz	Added Product Requirements
04.11.2019	Sinan Yurttadur	Added non-functional requirements

Introduction

Product Environment

The application consists of two large separated modules. The frontend, which will display all the information to the user in a web browser. And the backend, which will feed information to the frontend. The frontend is created in Angular. It will rely on the API provided by the backend to display information about the individual Profinet-devices in the local network. The backend of the application is a NodeJS server. It will provide a JSON-based REST API implementation maturity level 2. The application will be installed on Windows 10 machines with an installer. The installer guides you through the installation process. The application has a manual, describing the features and how to use the application. Steps like initialization (if needed) or a simple use case will be shown and explained.

The target environment for the application to run in is Windows 10. An optional goal is to make the application work on Linux as well.

Use Cases

The use cases of the application are the searching of devices in the local network and returning a list of them to the user (<UC.001>) and also displaying detailed information about a chosen device out of the list (<UC.002>).

Optional use cases are the configuration of devices from within the client (<UC.003>).

<UC.001> Search Devices

Descriptor	Description
Use Case Objective	The server handles the frontend request over an API-request. The server will start to listen for all Profinet-devices in the network. Then all found devices will be send back to the frontend in form of a list or array
System Boundary	The system is bound to the local network of the server

Descriptor	Description
Precondition	The connection to the network is required
Involved Users	Angular Frontend, NodeJS Server
Triggering Event	A click on the "Search Devices" button

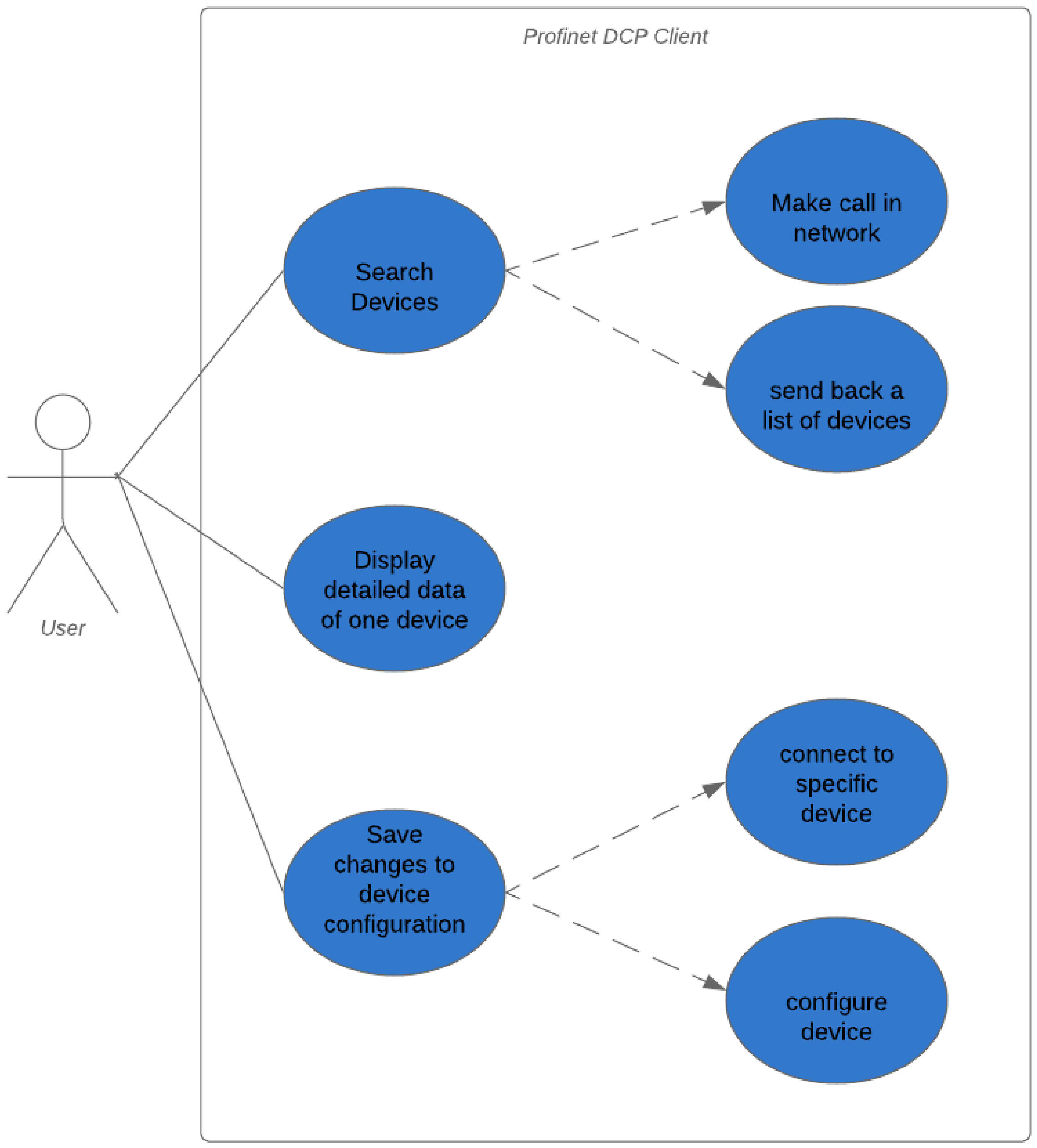
<UC.002> Display detailed data of device

Descriptor	Description
Use Case Objective	The frontend displays the information of one of the found devices. Information like the IP-address and name of the device are shown.
System Boundary	The frontend is limited to the list of devices it received by the server.
Precondition	A list of devices must have been sent back to the frontend
Involved Users	Angular Frontend, NodeJS Server
Triggering Event	Selecting a device

<UC.003> Save changes to device configuration

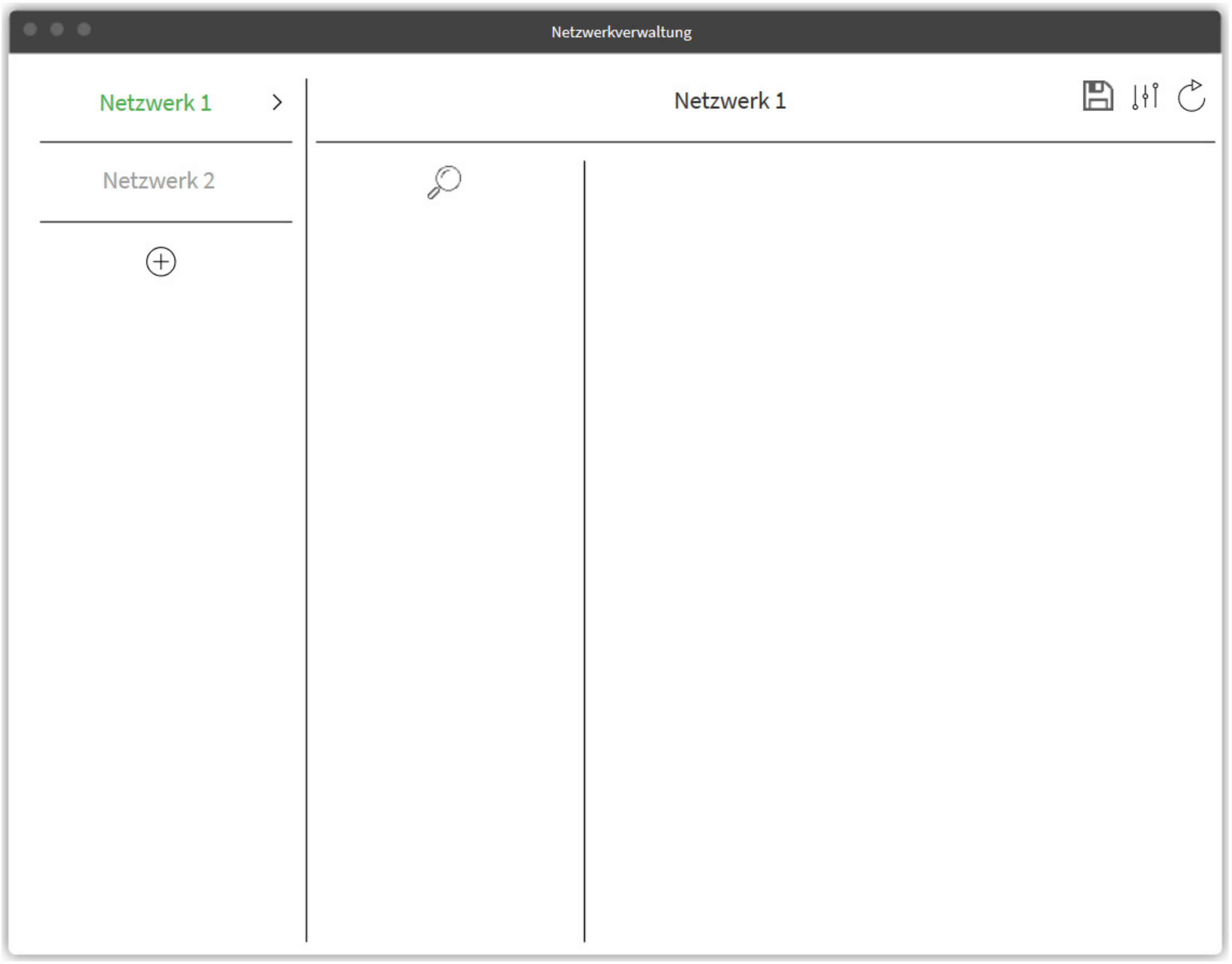
Descriptor	Description
Use Case Objective	The objective is to save the new configuration of a selected device. Changes like a different IP-address or name are to be used by the device from that point on.
System Boundary	The operation is limited to the selected device.
Precondition	A user must have selected a device, changed the configuration, and a connection to the device is needed.
Involved Users	Angular Frontend, NodeJS Server, the selected device
Triggering Event	A click on the "Save new configuration" button

The Use Case Diagram


Profinet DCP Client
User
Search Devices
Make call in network
send back a list of devices
Display detailed data of one device
Save changes to device configuration
connect to specific device
configure device

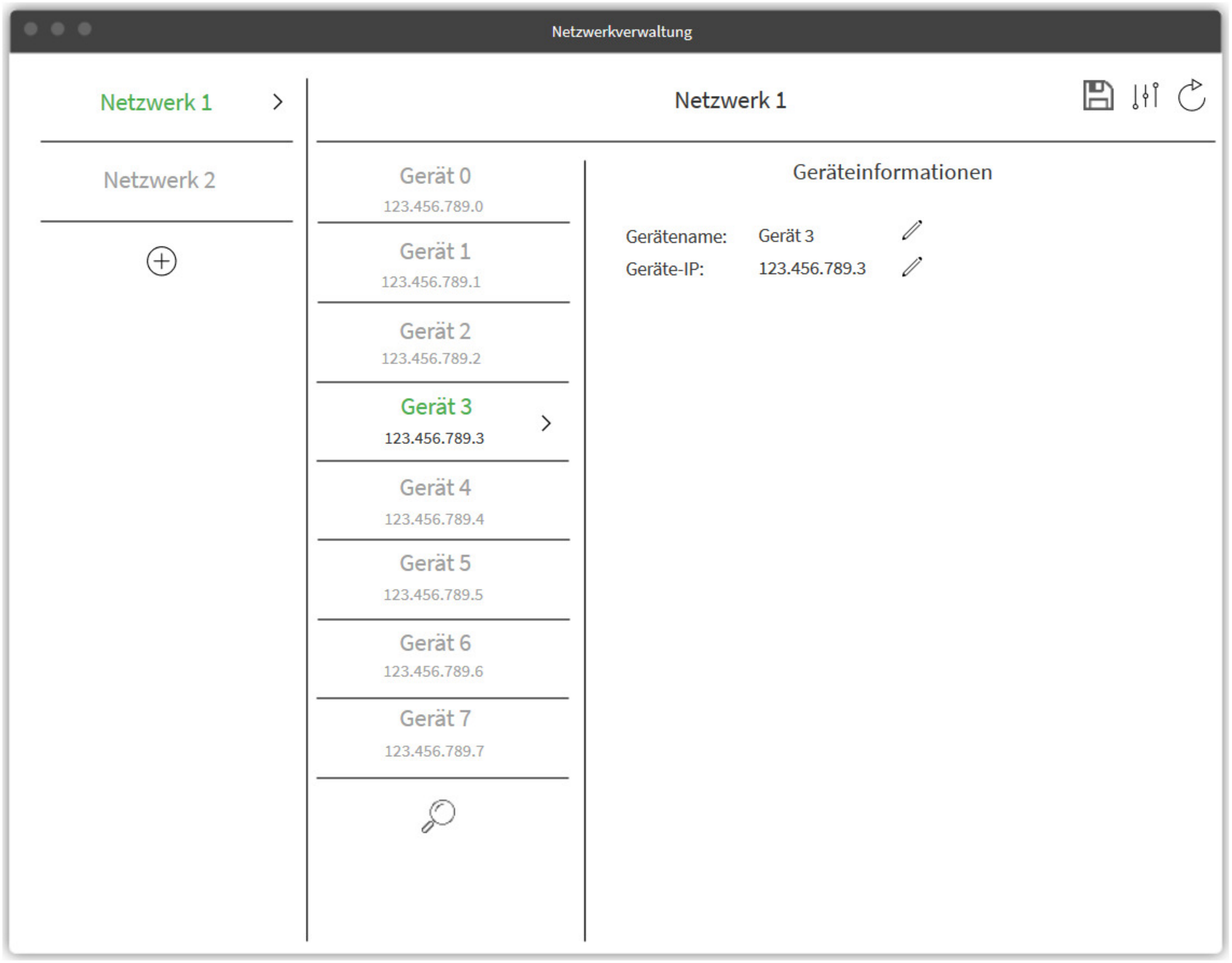
Product Requirements

/LF10/ Searching Devices



The magnifying glass icon shall send a request for all the devices in the network to the server to receive a list of all devices which will then be displayed.

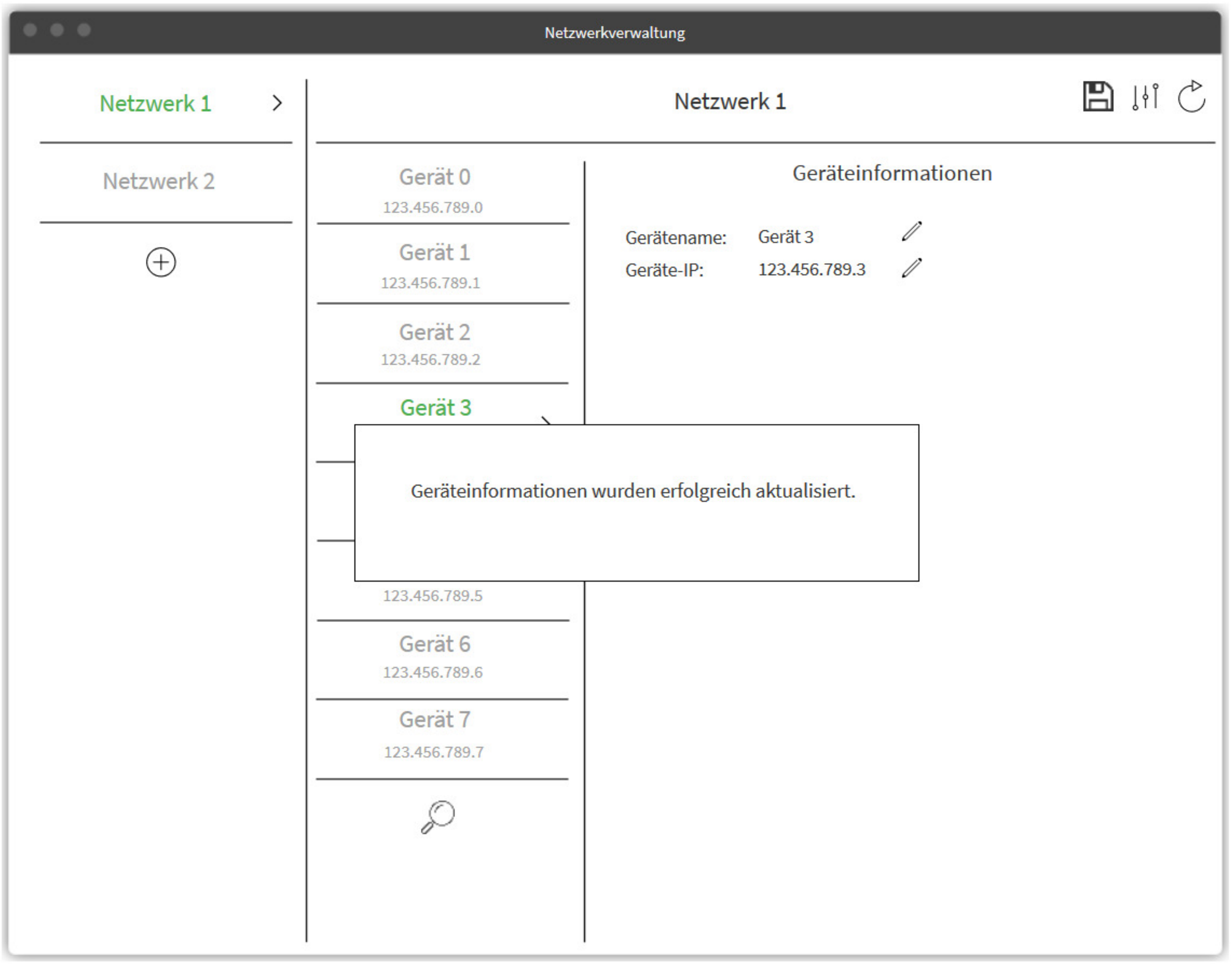
/LF20/ Display detailed data of device



The device name will expand the view to show the information about the device. The pen next to the attributes allows for a change of that attribute.

Input field	Value Range
Name	String with 0 to 50 letters
IP-Adress	String in IPv4-format "xxx.xxx.xxx.xxx"

/LF30/ Save changes to device configuration

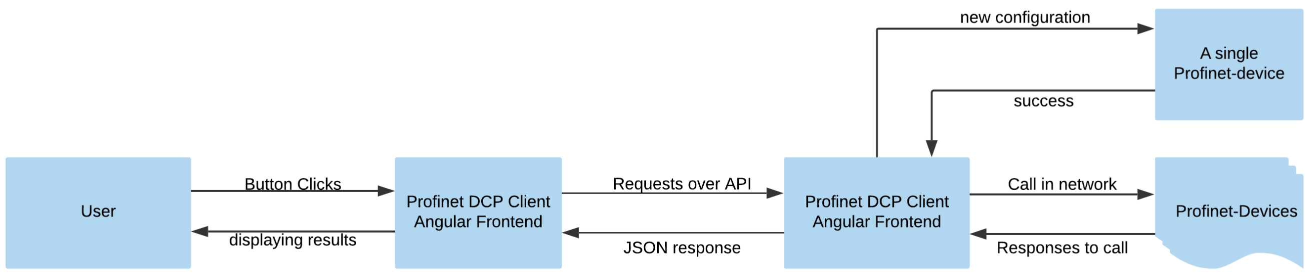


Geräteinformationen wurden erfolgreich aktualisiert.

Clicking the floppy-icon shall save the current configuration of that device by sending the information to the backend allowing it to apply the changes. An additional info box will appear, notifying the user whether or not the changes were successful.

Product Data

This section describes the various data the application and its interfaces uses.



/LD10/ User

The user provides the application with events, for the application to react. The only information the user generates is a new configuration for a device.

/LD20/ Frontend

The frontend supplies various buttons, like searching for devices or saving a new configuration, it responds to the user and handles all of the interaction. The frontend is dependent on a list of devices it receives from the backend. The data it receives is a list of JSON objects. Each object resembles a device. It can make a configuration request for a device over the API by making a patch-request.

/LD30/ Backend

The backend is handling all network activity and supplies the frontend with a list of devices. It also takes change requests when a new configuration for a device has to be made. The backend communicates with the local network as well as the frontend. It can be interacted with the JSON-based REST API.

/LD40/ Devices

The devices have their information stored locally which can be rewritten and read by the server.

Non-functional Requirements

Usability: Die Software muss den Anforderungen der Zielgruppe entsprechen. Folgende Kriterien wurden im Laufe der Recherche als wichtig empfunden:

- UX Design
- Bedienbarkeit
- Einfachheit
- Konfiguration und Einstellungsmöglichkeiten
- Hilfeanzeige Die Software wird nach den Standard der UNIX-Apps konzipiert und mit den gegebenen Anforderungen für Angular und Typscript ausgearbeitet.

Efficiency: Die Kommunikation der Schnittstellen sollte aufgrund der Nutzerfreundlichkeit nicht länger als 0,2 Sekunden benötigen. Außerdem sollte die Serveranfrage über die internen Speicher stattfinden.

Maintenance: Es soll sich im ersten Schritt über eine Prototyp Entwicklung handeln. Das heißt, es sollen Veränderungen mit einkalkuliert werden. Es können Anpassungen in der Funktionsweise sowie im Design gefordert werden. Außerdem ist es möglich, den Workflow zu verändern. Dieser muss nach den Test festgestellt werden.

Security: Die Daten sind im Firmeninternen Server hinterlegt und auf Zugriff von Dritten und Personen ohne Zugriffsrechte geschützt. Somit ist die Funktionsweise gewährleistet. Es können Änderungen nur vom Admin getroffen werden.

Regulatory: Das Produkt muss keine gesetzlichen Anforderungen erfüllen.

Änderungen vorbehalten.

References

General Information about Profinet:

- https://us.profinet.com/technology/profinet
- https://profinetuniversity.com

Communication with Profinet devices

- https://github.com/Eiwanger/profinet_scanner_prototype/blob/master/SendPacket/ReadMe.txt

Glossary

- Angular: A programming language primarily used to develop a fast, responsive and platform indipendent website
- NodeJS: A server-side JavaScript runtime for network applications
- JSON: JavaScript Object Notation
- API: Application Program Interface

Home

Manual

System Architecture Specification
System Requirement Specification

+ Add a custom sidebar

Clone this wiki locally

```
https://github.com/nicolasbreuni/Tinf18C_Team_2_Profinet_DCP_Client.wiki.git
```