htlperg
innovativ | kreativ | praxisnah

Hardwareinformationen ITSM Software

DIPLOMARBEIT

Höhere Abteilung für Informatik

01/07/2024 – 01/04/2025

Projektmitglieder: Jakob Lettner
David Tober

Betreuer:in: Prof. Ing. Patrick Praher, MSc

Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Perg, 03.04.2025

Ort, Datum

Unterzeichner, David Tober

Perg, 03.04.2025

Ort, Datum

Unterzeichner, Jakob Lettner

Gendererklärung

Im Sinne der besseren Lesbarkeit werden in dieser Diplomarbeit personenbezogene Bezeichnungen, die sich zugleich auf Frauen und Männer beziehen, generell nur in der im Deutschen üblichen maskulinen Form angeführt. Dies soll jedoch keinesfalls eine Geschlechterdiskriminierung oder eine Verletzung des Gleichheitsgrundsatzes zum Ausdruck bringen.

Danksagung

Wir möchten uns herzlich bei allen bedanken, die uns aufgrund ihrer Unterstützung unsere Diplomarbeit ermöglicht haben.

Daher möchten wir uns besonders bei unserem Auftraggeber, der Firma CountIT, welche uns nicht nur diese Diplomarbeit in der Zusammenarbeit mit ihnen, sondern auch ein vierwöchiges Praktikum im Unternehmen ermöglicht haben, bedanken.

Besonders bedanken möchten wir uns bei unseren Betreuern Matthias Eder, welcher uns diese Diplomarbeit ermöglicht hat, sowie Elias Leonhardsberger, der uns nicht nur bei sämtlichen Anliegen mit Rat und Tat zur Seite gestanden ist und uns auch unser Praktikum so schön wie möglich gestaltet hat.

Ebenfalls möchten wir uns bei Matthias Furtlehner bedanken, welcher uns durch seine tatkräftige Unterstützung, sicher durch unser Praktikum begleitet hat.

Weiters gilt ein besonderer Dank unserem Diplomarbeitsbetreuer, Herrn Professor Ing. Patrick Praher, MSc, der uns bei all unseren Fragen, im Bezug auf das Verfassen der Diplomarbeit, zur Seite stand.

Auch gilt ein SEHR großer

Auch gilt unser Dank all unseren Familien, Freunden und Bekannten, insbesondere Karl C. G., welcher uns stets durch sein enormes Wissen bei dieser Diplomarbeit effektiv unterstützt hat.

Abstract

The basic idea of our diploma thesis is the automatic provision of hardware information to an **IT-Service-Management Software**. The aim is to develop an web application that is able to automatically read all computers with the corresponding information and display the information.

To achieve this goal, 2 main components have been created:

The graphical user interface, GERI, on which all information is displayed. The source of the data is the hardware information of the devices from our specially developed service, AGNES.

The backend, AGNES, which gathers the hardware information of the respective device fully automatically and then sends it via a REST API to a CountIT backend server. There, the data is stored and processed.

As a result, CountIT's **Technical Service** saves a lot of time by eliminating the need to manually maintain the respective desktop and notebook computers. In addition, the TS receives an intuitive and user-friendly display via a web application, which further simplifies the otherwise very complex and time-consuming work. This also makes it even easier for the **Technical Service** to find out which devices are currently in use and which are not, which are too old and need to be replaced, and so on.

Zusammenfassung

Unsere Diplomarbeit basiert auf der Idee, Hardwareinformationen automatisch an eine **ITSM-Software** bereitzustellen. Es soll eine Webanwendung entwickelt werden, die alle Hardwareinformationen aller verwendeten Computer des Unternehmens vollautomatisch einliest und sie übersichtlicht anzeigt.

Zur Erreichung dieses Ziels wurden zwei Hauptkomponenten entwickelt:

- Die grafische Benutzeroberfläche, **GERI**, auf der sämtliche Informationen dargestellt werden. Die Quelle dieser Daten stellen die Hardwareinfos der Geräte von unserem eigens entwickelten Service, **AGNES**, dar.
- Das Backend, **AGNES**, welches vollautomatisiert die Hardwareinfos über das jeweilige Gerät sammelt und danach über eine **REST-API** an einen Backend-Server der CountIT sendet. Dort werden die Daten anschließen auch abgespeichert und verarbeitet.

Durch die von uns erstellte Webanwendung erspart sich das CountIT Technical Service viel Zeit, da die jeweiligen Standcomputer und Notebooks nicht mehr manuell gewartet werden müssen. Darüber hinaus erhält der **TS** eine intutive und benutzerfreundliche Darstellung mittels einer Webanwendung, welche die sonst sehr aufwendige und zeitintensive Routinearbeit noch mehr vereinfacht. Außerdem kann der **TS** somit noch einfacher herausfinden, welche Geräte eventuell gerade benutzt werden und welche nicht, welche ausgetauscht werden müssen, etc.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die Firma CountIT besitzt zahlreiche Computer und Notebooks, welche natürlich auch vom jeweiligen Technical Service, der für die Verwaltung der Hardware im Unternehmen zuständig ist, dementsprechend verwaltet und gewartet werden müssen. Dieser Routineprozess ist kosten- und vorallem zeitintensiv und bedeutet für die Mitarbeiter des TS nur zusätzliche Arbeit. Deshalb hat uns die Firma CountIT die Möglichkeit gegeben, genau dieses Problem in Zusammenarbeit mit ihnen zu lösen. Unsere Anwendung soll es dem TS, mithilfe einer Webanwendung erleichtern, genau diese kosten- und zeitaufwändigen Routinearbeiten zu vereinfachen.

1.2 Zielsetzung

Das Ziel unserer Diplomarbeit ist die Entwicklung einer Anwendung, welche dem TS die Arbeit nicht nur angenehmer macht, sondern diese auch dramatisch vereinfacht. Dabei sollen alle Standcomputer und Notebooks vollautomatisiert eingelesen und übersichtlich in einer Webanwendung dargestellt werden. Auch soll es möglich sein, selbst Geräte anzulegen, zu konfigurieren und natürlich danach eventuell auszuweisen beziehungsweise zu entfernen.

Die Daten, die für die Darstellung in der Webanwendung benötigt werden, sollen mithilfe eines vollautomatischen, auf den Geräten der Mitarbeiter der CountIT ausgeführten Programmes, erfasst werden. Mithilfe einer API sollen diese Daten danach an das Backend gesendet werden, wo diese verarbeitet und korrekt in einer Datenbank mit relationalem Schema erfasst werden. Diese Geräteinformationen sollen nach der Verarbeitung an die Webanwendung gesendet werden, welche diese übersichtlich darstellt.

1.3 Projektinhalt - Überblick

Ausgangslage unserer Arbeit war ein altes Schulprojekt von Schülern unserer Schule, der HTL Perg. Dieses Projekt, auf dem diese Arbeit aufbaute, war zum Großteil noch nicht vorhanden beziehungsweise noch in einem Zustand, der nicht zur Zufriedenheit des Unternehmens war. Deshalb bat das Unternehmen uns, dieses Projekt zu übernehmen, zu vervollständigen und durch einen komplett neuen Teil zu erweitern. Diese Arbeit soll aus 3 Bereichen bestehen:

- dem Frontend
- dem Backend
  und
- AGNES

Das Frontend für diese Arbeit war zu kleinen Teilen schon vorhanden, aber in einem nicht fertigen Zustand und wird von uns so erweitert beziehungsweise fertiggestellt, dass es über sämtliche geforderte Funktionalitäten verfügt. Das Backend zu dieser Arbeit war ebenfalls zu kleinen Teilen vorhanden und soll durch unseren Beitrag fertiggestellt und um den komplett neuen Teil, AGNES, erweitert werden.

1.3.1 Frontend

Das Frontend dieser Arbeit wird aus insgesamt 6 Seiten bestehen, die alle unterschiedlichen Zwecken dienen.

Dashboard

Diese Seite soll lediglich eine Übersicht mit dem Logo des Unternehmens, einem Schnellzugriff zur Unternehmenswebseite sowie zur wohl wichtigsten Seite dieser Arbeit: der Geräteverwaltungsseite, bieten.

Geräteverwaltung

Diese Seite soll die Hauptseite für die gesamte Applikation sein. Darauf soll eine Übersicht über alle Geräte, die im Unternehmen im Einsatz sind, angezeigt werden. Darüber hinaus soll diese Seite sich in 2 Unterseiten unterteilen:

- Aktive Geräte
- Ausgewiesene Geräte

Diese Seiten unterscheiden sich dadurch, dass auf der Aktiven Geräteseite, nur die Geräte, die aktuell im Einsatz sind, dargestellt werden. Auf der Ausgewiesenen Geräteseite, sollen nur die nicht verwendeten und somit ausgewiesenen Geräte dargestellt werden.

Alle Geräte, die aktiven sowie die ausgewiesenen, sollen in einer Tabelle dargestellt werden, die wiederum in Spalten unterteilt ist. Sämtliche Spalten sollen einzeln sortier- und filterbar sein. Ebenfalls soll die Seite über eine Suchleiste verfügen, die serverseitig, sämtliche Geräte nach Namen durchsucht.

Vorlagen

Auf dieser Seite sollen sogenannte Vorlagen erstellt, gespeichert und dargestellt werden können. Eine Vorlage ist ein Bauplan, der einem neu erstellten Gerät zugewiesen werden kann. Beispielsweise hat die Vorlage die Eigenschaften: Marke mit dem Wert Lenovo, CPU mit dem Wert AMD, etc.. Diese Vorlage kann somit einem neu erstellten Gerät zugewiesen werden. Somit hat dieses Gerät vollautomatisch die Werte der Vorlage. Dies ist nützlich, um die manuelle Erstellung von Geräten zu erleichtern, da die Werte nicht mehr einzeln zugewiesen werden müssen.

Genau wie die Geräteseite, sollen die Daten in einer Tabelle mit dementsprechenden Spalten, die sortier- und filterbar sind, dargestellt werden.

Auch eine Suchleiste, die die Vorlagen entsprechend dem Namen serverseitig durchsuchen, soll auf der Seite sein.

Benutzer

Diese Seite soll dazu genutzt werden, eine Übersicht über sämtliche Benutzer des Unternehmens zu bieten. Natürlich sollen auf dieser Seite auch neue Benutzer erstellt werden können.

Wie alle anderen Seiten auch, soll die Benutzerübersicht auch über eine Suchleiste, welche die Benutzer serverseitig durchsucht, verfügen.

Eigenschaften

Hier sollen die Eigenschaften eines Gerätes dargestellt werden. Eine Eigenschaft eines Gerätes kann beispielsweise die CPU, der RAM, die Marke, etc.. sein.

Genauso wie alle anderen Seiten, so soll diese Seite auch über eine Suchleiste verfügen.

Gerätetypen

Auf dieser Seite soll ein Überblick über die unterschiedlichen Gerätetypen gegeben werden. Ein Gerätetyp kann beispielsweise ein Laptop, ein Stand-PC, ein Server, etc.. sein.

Ebenfalls sollen sämtliche Gerätetypen mittels einer Suchleiste natürlich serverseitig durchsuchbar sein.

1.3.2 Backend

Das Backend für diese Arbeit ist größtenteils bereits vorhanden und soll von uns in einen produktionsreifen Zustand gebracht und um den großen neuen Teil, dem automatischen Einlesen von Hardwaredaten, AGNES erweitert werden.

Unteranderem soll das Backend auch die Benutzervalidierung handhaben. Um festzustellen, ob eine Person Eigentümer eines Computers ist, wird zu jedem Zeitpunkt, zu dem ein Benutzer eine Anmeldung an einem Gerät vornimmt, die Anmeldung in eine Liste gespeichert. Überschreitet die Anzahl der Anmeldungen eines Benutzers an dem jeweiligen Gerät innerhalb eines festgelegten Zeitraumes 17, so wird der eingeloggte Benutzer als der derzeitige Eigentümer des Computers angenommen.

Um unser Ziel zu erreichen, muss das Backend, insbesondere die Datenbank, geringfügig geändert werden.

1.3.3 AGNES

AGNES soll den Hauptaspekt und komplett neuen Teil unserer Arbeit darstellen. Mithilfe dieses Programms soll es unter anderem ermöglicht werden, sämtliche Hardwareinformationen eines Computers auszulesen. Die ausgelesenen Informationen sollen an einen Server gesendet werden, welcher für die Auswertung dieser Daten zuständig ist. Ebenfalls soll dieses Programm es ermöglichen, mithilfe des eingeloggten Benutzers auf dem jeweiligen Gerät zu ermitteln, welchem Benutzer das jeweilige Gerät gehört. Um dieses Ziel zu erreichen, soll AGNES nicht

nur die jeweiligen Hardwaredaten eines Gerätes auslesen, sondern auch den Namen des auf dem Gerät eingeloggten Benutzers.

1.4 Projektumfeld

Diese Diplomarbeit ist in Zusammenarbeit mit dem Unternehmen CountIT entstanden. Weiters sind mehrere Personen der HTL Perg beteiligt, welche im nachfolgenden Abschnitt angeführt werden.

1.4.1 Projektteam

Das Projektteam besteht aus 2 Schülern der HTL Perg:

Jakob Lettner **David Tober**

Jakob Lettner ist aufgrund seiner Begeisterung und Begabung im Webdesign für das Designen der Webanwendung und die Implementierung des Frontends verantwortlich gewesen.
David Tober ist aufgrund seines Interesses an Datenbanken für das Einlesen der Gerätedaten sowie das Backend verantwortlich gewesen.

1.4.2 Betreuung

Die gesamte Diplomarbeit wurde von Prof. Ing. Patrick Praher, MSc betreut. Er ist zugleich auch unser Professor im Unterrichtsfach Programmieren mit der Programmiersprache Java. Auch bei Fragen rund um das Schreiben der Diplomarbeit stand er uns unterstützend zur Seite.

1.4.3 Auftraggeber CountIT

Abbildung 1: Logo der Firma CountIT

Auftraggeber dieser Diplomarbeit ist das Unternehmen CountIT GmbH. Hauptaspekt des Unternehmens ist die Entwicklung und Wartung von Individualsoftware. Wir haben unsere Diplomarbeit im Rahmen eines vierwöchigen Praktikums beginnen und teilweise vollenden können.

2 Grundlagen und Methoden

2.1 Verwendete Technologien

2.1.1 C#

Unsere Diplomarbeit wurde mit der Programmiersprache C# entwickelt. C# ist eine objektorientierte Programmiersprache, die von Anders Hejlsberg im Auftrag von Microsoft im Jahre 2000 entwickelt wurde. C# greift Konzepte von Java, C, C++, Haskell und Delphi. Im Gegensatz zu anderen Programmiersprachen wie C, C++ verfügt C# über einen "Garbage Collector", welcher durch eine automatische Speicherverwaltung, Speicherlecks im Code reduziert. Ebenso ist diese Programmiersprache sehr flexibel in ihren Einsatzgebieten und benutzerfreundlich.[1]

Abbildung 2: Logo von C#

2.1.2 Blazor

Um im Zuge unserer Diplomarbeit die Weboberfläche zu entwickeln, haben wir das Frontend-Framework Blazor verwendet. Blazor wird von Microsoft entwickelt und erlaubt es Entwicklern, HTML und CSS mit C# zu vereinen und somit Benutzeroberflächen für Webanwendungen zu entwickeln. [2]

Abbildung 3: Logo von Blazor

2.1.3 Blazor WebAssembly

Um unsere Webanwendung nun mit entsprechenden Funktionen zu versehen, wird Blazor WASM, eine spezielle Variante von Blazor, mit der C#-Code zuerst in WebAssembly kompiliert und danach Clientseitig im Browser ausgeführt wird, verwendet. [3]

Abbildung 4: WebAssembly

2.1.4 REST-API

Eine REST-API ist eine Schnittstelle, welche es ermöglicht, dass verschiedenste Anwendungen über ein Netzwerk miteinander kommunizieren.[4] Bei unserer Arbeit ist unser Backend-Teil, AGNES mit einem Backend-Server verbunden, welcher die gesendeten Daten verarbeitet. Auch ist unser Frontend über eine REST-API mit dem Backend-Server verbunden, um die gesamten Daten abzufragen.

Abbildung 5: REST

2.1.5 Git und GitLab

Git ist eine Software zur Versionsverwaltung, welche es ermöglicht, gemeinsam an Projekten zu arbeiten und den Verlauf des Projekts nachzuvollziehen. Im Unterschied zu anderen Versionsverwaltungssystemen speichert Git bei Änderungen nicht die gesamte Datei, sondern nur die Änderung und für die bereits vorhandenen Dateien wird ein Verweis auf die ursprüngliche Datei erstellt. [5]

GitLab bietet für Git die dazu passende Webanwendung, um diesen Prozess zu vereinfachen. Ebenfalls bietet es Entwicklern desweiteren die Möglichkeiten, Issues, Meilensteinlisten sowie Merge-Requests zu erstellen und zu verwalten.

Abbildung 6: Logo von Git

Abbildung 7: Logo von GitLab

2.1.6 Azure DevOps

Azure DevOps ist eine, von Microsoft bereitgestellte Plattform, die den gesamten Softwareentwicklungsprozess unterstützt. So bietet es unter anderem ein Board, mit dem Tickets verwaltet werden können, sowie ganze Git-Repos erstellt/verwaltet werden können. Auch gibt es die Möglichkeit, sogenannte Pipelines zu erstellen, die dabei helfen, die Testworkflows zu vereinfachen und zu automatisieren. [6]

Abbildung 8: Logo von Azure DevOps

2.1.7 Microsoft Entra ID

Microsoft Entra ID ist ein, von Microsoft zur Verfügung gestellter, cloudbasierter Identitäts- und Zugriffverwaltungsdienst. In unserer Arbeit wird es dazu verwendet, die Authentifizierung der Benutzer durchzuführen. Dadurch wird nur den richtigen Benutzern des Unternehmens Zugriff auf die Anwendung gewährt.

Abbildung 9: Logo von Microsoft Entra ID

2.1.8 Microsoft SQL Server

Der Microsoft-SQL Server ist ein von Microsoft entwickeltes Managementsystem, für relationale Datenbanken. In unserer Arbeit wurde im Backend auch ein Microsoft SQL Server verwendet, um die gesamten Daten zu sichern.

Abbildung 10: Logo von Microsoft SQL Server

2.2 Verwendete Entwicklungssysteme

2.2.1 Visual Studio

Visual Studio ist eine von Microsoft entwickelte integrierte Entwicklungsumgebung. Damit wird es Anwendern ermöglicht, Programme in den Programmiersprachen C, C++,C#, SQL Server, TypeScript sowie HTML, CSS, Python zu entwickeln. Vorrangig wird Visual Studio dazu verwendet, Webanwendungen, Desktop-Apps und Anwendungen mit dem .Net-Framework zu entwickeln. [7]

Visual Studio verfügt über Features wie IntelliSense-Codecompletation, WSL-Integration, Git sowie Live-Share-Session, Azure Integration und starken Profiling Tools für CPU, Datenbanken usw. [8]

Abbildung 11: Logo von Visual Studio

2.2.2 SQL Server Management Studio

SQL Server Management Studio ist ein von Microsoft entwickeltes Tool, um mit sämtlichen SQL-Strukturen von SQL Server bis hin zur Azure SQL-Datenbank zu arbeiten. SSMS stellt dem Benutzer Tools zur Verfügung, um Abfragen und Skripts, sowie die Datenbankkomponenten bereitzustellen, zu überwachen und zu aktualisieren. [9]

Abbildung 12: Logo von SQL Server Management Studio

2.3 Verwendete Bibliotheken und Plug-Ins

2.3.1 .NET Core

.NET Core ist ein Open-Source Framework zur Entwicklung von Webanwendungen, APIs und Microservives. **.NET Core** ist eine Weiterentwicklung von ASP.NET. Hauptmerkmale von .NET Core ist die Plattformunabhängigkeit, die verbesserte Performance im Gegensatz zum Vorgänger, sowie das einheitliche .NET-Ökosystem. [10]

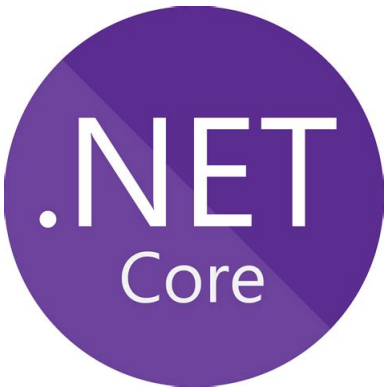


Abbildung 13: .NET Core

2.3.2 ADO.NET

ADO.NET ist eine von Microsoft's Technologien für den Datenzugriff auf diverse Datenquellen, wie den Microsoft SQL Server. ADO.NET ist Teil des .NET Frameworks. ADO.NET beinhaltet bereits vorgefertigte Klassen, die für die Durchführung der CRUD-Operationen aus Datenquellen verwendet werden können.[11]

2.3.3 MudBlazor

MudBlazor ist eine Bibliothek für Blazor, die das aufwendige, händische Erstellen von Benutzeroberflächen für Webanwendungen erleichtert. MudBlazor stellt den Entwicklern vorgefertigte UI-Komponenten zur Verfügung. [12] Um in unserer Arbeit das Frontend zu erschaffen, wurde auf Komponenten von MudBlazor zurückgegriffen.

Abbildung 14: Logo von MudBlazor

2.3.4 Swagger

Swagger ist ein Open-Source-Framework, mit dem es ermöglicht wird, über eine Benutzeroberfläche im Browser, RESTful-APIs zu dokumentieren, zu strukturieren und zu visualisieren. Auch in unserem Projekt wurde Swagger dafür verwendet.[13]

Abbildung 15: Logo von Swagger

2.4 Sonstige verwendete Software

2.4.1 SonarQube Cloud

SonarQube Cloud (früher: SonarCloud) ist eine cloudbasierte Lösung zur statischen Codeanalyse. Mithilfe dieses Tools wird die Codequalität kontinuierlich überprüft und verbessert. In unserem Projekt wurde jeder von uns erstellte Code vor der Integration in den Main-Branch durch eine Überprüfung von diesem Tool, dank Integration in **Azure DevOps**, validiert.[14]

Abbildung 16: Logo von SonarCloud

2.4.2 Clockify

Clockify ist eine kostenlose Software für Zeiterfassung und Zeitmanagement. Mithilfe von Clockify ist es möglich, Arbeitszeiten genau zu protokollieren, einzelne Projekte zu verwalten, sowie detaillierte Berichte zu erstellen.[15] In dieser Arbeit wurde Clockify dazu verwendet, die jeweiligen Arbeitszeiten pro Person mit zu protokollieren, sowie die Gesamtarbeitszeit an dieser Arbeit aufzuzeichnen.

Abbildung 17: Logo von Clockify

2.4.3 Microsoft Teams

Microsoft Teams ist eine Kommunikationssoftware, mit der man die Kommunikation in Teams oder Unternehmen vereinfacht.[16] Während des Praktikums bei CountIT wurde Microsoft Teams als unser Hauptkommunikationsweg eingesetzt.

Abbildung 18: Logo von Microsoft Teams

2.4.4 Overleaf

OverLeaf ist ein cloudbasierter Online-Editor für LaTeX. Dieser wurde speziell für das Schreiben, Bearbeiten und Veröffentlichen von wissenschaftlichen Arbeiten entworfen. Diese Arbeit wurde mithilfe von OverLeaf verfasst.[17]

Abbildung 19: Logo von OverLeaf

3 Implementierung

3.1 Datenbank

Die Datenbank hat in unserem Produkt die Aufgabe, sämtliche Informationen über die Computer, sowie den Namen der jeweiligen Benutzer zu speichern. Diese gespeicherten Daten werden in der Webanwendung entsprechend dargestellt.

3.1.1 Datenmodel

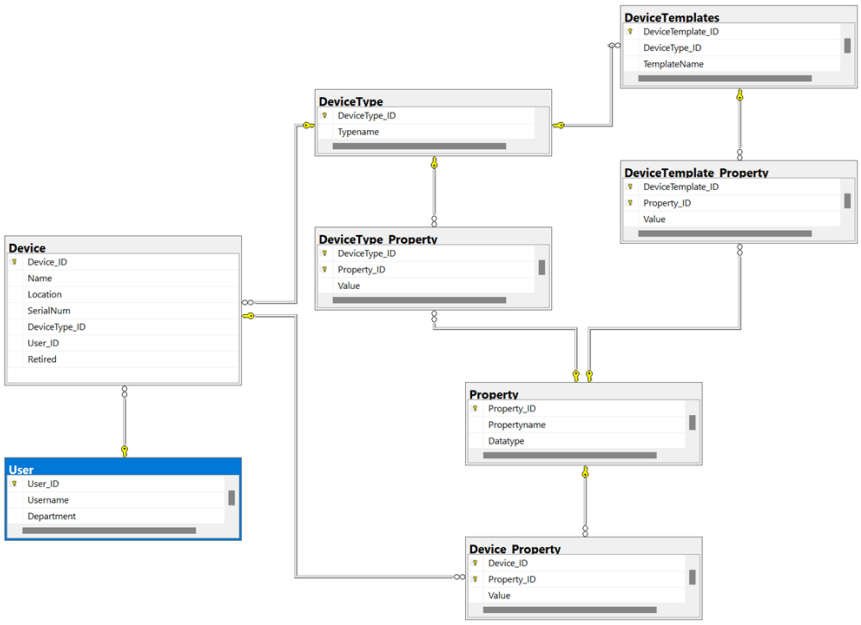


Abbildung 20: Datenbankschema

Unser Datenbankschema (Siehe Abb. 20) ist in einem normalisierten relationalen Schema, mit Fokus auf Konsitenz und Normalisierung, gehalten. Die Entitäten in der Datenbank wurden auf Wunsch des Unternehmens in Englisch gehalten.

Entität *device*

Hier werden sämtliche Informationen über ein Gerät, wie der Name des Gerätes, die Seriennummer und so weiter gespeichert. Diese Tabelle ist mittels einem Primärschlüssel auf die Geräte ID, über 1:n Beziehungen, mit den Tabellen Entität *devicetype* und Entität *user* verknüpft. Dies

bedeutet, dass ein Gerät genau zu einem Gerätetyp gehört, sowie ein Gerät zu genau einem Benutzer.

Entität *device property*

Hier werden in einer Verbindungstabelle sämtliche Geräte mit den jeweiligen Eigenschaften verknüpft. Da ein Gerät mehrere Eigenschaften hat und eine Eigenschaft zu mehreren Geräten gehören kann, wird hier eine *n:m*-Beziehung zwischen Entität *device* und Entität *property* verwendet.

Entität *device templates*

Diese Tabelle speichert die Vorlagen für ein Gerät. Verknüpft ist diese Tabelle mithilfe einer *1:n*-Beziehung mit Entität *devicetype* und in einer *n:m*-Beziehung mit Entität *device template property*. Somit ist jede Vorlage einem Gerätetyp zugeordnet, jede Vorlage kann über mehrere Eigenschaften verfügen und umgekehrt.

Entität *device template property*

Diese Tabelle ist eine Zwischentabelle zwischen Entität *device templates* und Entität *property*, welche die Vorlagen mit den Eigenschaften speichert.

Entität *devicetype*

Hier werden die unterschiedlichen Gerätetypen definiert und gespeichert. Da ein Gerätetyp mehrere Geräte haben kann, aber ein Gerätetyp nur ein Gerät, ist hierfür eine einfache *1:n*-Beziehung ausreichend. Ein Gerätetyp kann allerdings mehrere Geräteeigenschaften haben und umgekehrt, somit muss eine *n:m*-Beziehung verwendet werden.

Entität *devicetype property*

Hier werden die Verknüpfungen zwischen Gerätetypen und den entsprechenden Eigenschaften gespeichert. Ein Gerätetyp kann über mehrere Eigenschaften und umgekehrt verfügen, deshalb wird eine *n:m*-Beziehung verwendet.

Entität *property*

Diese Tabelle speichert die Eigenschaften, welche den Geräten oder Typen zugewiesen werden können. Verknüpft ist diese Tabelle *n:m* mit Entität *device property*, Entität *devicetype property* und Entität *device template property*. Somit kann, wie schon in Entität *device* erwähnt, ein Gerät über mehrere Eigenschaften verfügen und Eigenschaften mehreren Geräten zugewiesen werden. Ebenso kann ein Entität *devicetype* mehrere Eigenschaften besitzen und umgekehrt. Eine Vorlage ebenso über mehrere Eigenschaften und umgekehrt.

Entität *user*

Hier werden die Informationen über die Benutzer, die die Geräte verwenden, gespeichert. Diese Tabelle ist mittels einer *1:n*-Beziehung mit der Tabelle Entität *device* verknüpft, da ein Benutzer mehrere Geräte besitzen kann, ein Gerät aber nur einen Benutzer haben kann.

3.2 Frontend Implementierung

3.2.1 Layout

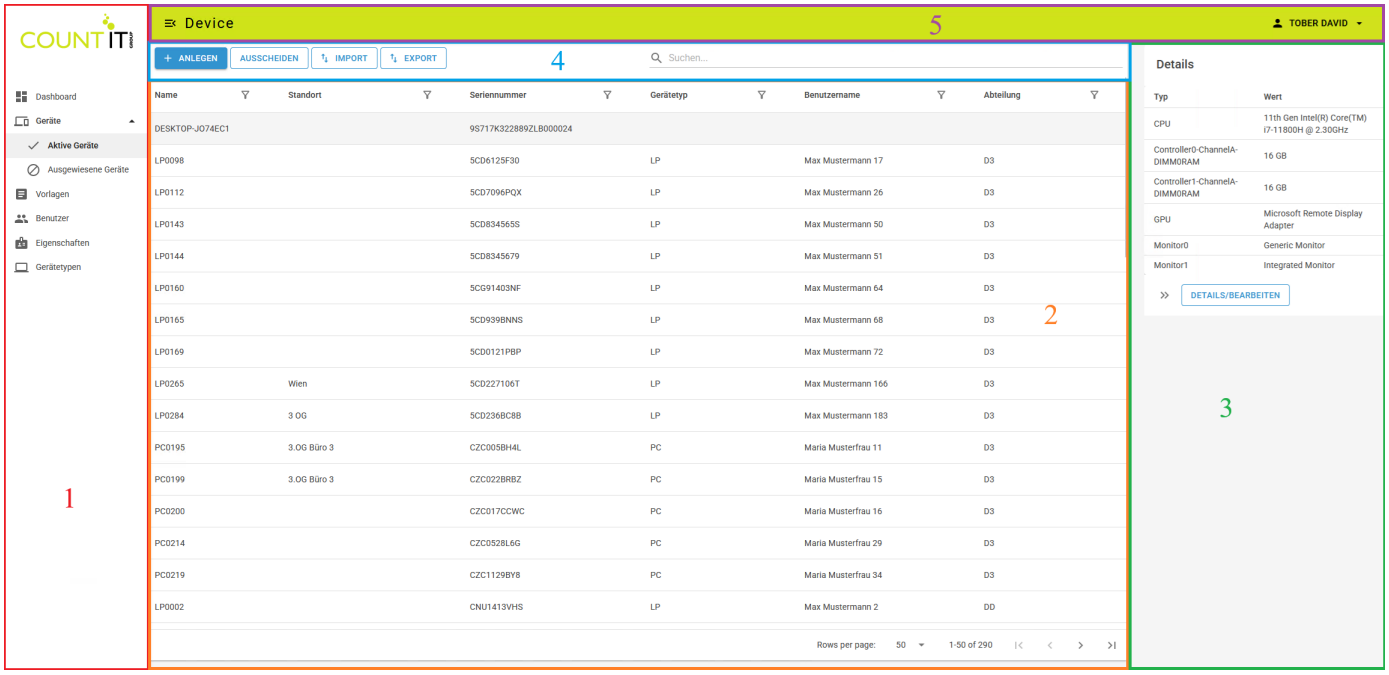


Abbildung 21: Layout des Frontends

Das Gesamtlayout des Frontend wurde in 5 Hauptbereiche unterteilt:

1. der Menüleiste,
2. der Hauptanzeige,
3. Der Sidebar,
4. der Toolbar,
5. der Titelleiste

Menüleiste

Listing 1: Code der Menüleiste

```
<MudNavMenu Bordered Style="margin-top: 30px;">
  <MudNavLink Href="/" Match="NavLinkMatch.All"
      Icon="@Icons.Material.Filled.Dashboard">Dashboard</MudNavLink>
  <MudNavGroup Title="Geraete" Icon="@Icons.Material.Filled.Devices"
      onclick="@NavigateToActiveDevices">
    <MudNavLink Href="/devices" Icon="@Icons.Material.Filled.Check"
        Match="NavLinkMatch.Prefix">Aktive Geraete</MudNavLink>
    <MudNavLink Href="/retired" Icon="@Icons.Material.Filled.Block"
        Match="NavLinkMatch.Prefix">Ausgewiesene Geraete</MudNavLink>
  </MudNavGroup>
  <MudNavLink Href="/templates" Icon="@Icons.Material.Filled.Article"
      Match="NavLinkMatch.Prefix">Vorlagen</MudNavLink>
  <MudNavLink Href="/users" Icon="@Icons.Material.Filled.People"
      Match="NavLinkMatch.Prefix">Benutzer</MudNavLink>
  <MudNavLink Href="/properties" Icon="@Icons.Material.Filled.Badge"
      Match="NavLinkMatch.Prefix">Eigenschaften</MudNavLink>
  <MudNavLink Href="/devicetypes" Icon="@Icons.Material.Filled.Laptop"
      Match="NavLinkMatch.Prefix">Geraetetypen</MudNavLink>
</MudNavMenu>
```

Dieser Code (Siehe Listing 1) definiert durch die '**MudNavMenu**' - Komponente das Navigationsmenü in der Menüleiste. Dieses Menü macht es für die Benutzer möglich, durch sämtliche Seiten der Anwendung einfach und übersichtlich zu navigieren. Diese Funktionalität wird dadurch erreicht, dass Navigationslinks zu den verschiedenen Seiten über die '**MudNavLink**'-Komponente bereitgestellt werden.

Die Verwendung der '**MudNavGroup**' - Komponente ermöglicht es, eine Gruppe von Navigationslinks zu erstellen. Dadurch werden mehrere Navigationslinks zu Seiten der Anwendung in eine Gruppe gebündelt und mittels eines Dropdown-Menüs dargestellt. Genau diese Funktionalität wird benötigt, da die Geräte-Seite aus 2 Unterseiten, der Aktiven- und der Ausgewiesenen-Geräteseite besteht.

Die Navigation zu den beiden Unterseiten der Geräteseiten – der Aktiven Geräteseite und der Ausgewiesenen Geräteseite – erfolgt über den sogenannten *NavigationManager* (Siehe Listing 2). Dieser ist Teil der Routing-Funktionalität von .NET Core.

Listing 2: Code des NavigationsManagers

```
[Inject]
NavigationManager NavigationManager { get; set; }

private void NavigateToActiveDevices()
{
  NavigationManager.NavigateTo("/devices");
}
```

Hauptanzeige

Der folgende Codeabschnitt definiert den Hauptinhalt der Anwendung und dessen Anordnung:

Listing 3: Grundlegende Struktur der Hauptseite

```
1       <div class="cit-main-content">
2           <MudGrid Container="true">
```

Der äußere *div*-Container mit der Klasse *cit-main-content* dient im Wesentlichen als übergeordnete Struktur für den gesamten Anzeigebereich. Das *MudGrid*-Element ist Bestandteil der MudBlazor-Komponentenbibliothek und ermöglicht es, UI-Elemente flexibel anzuordnen und wird außerdem dazu genutzt, um die zwei Hauptbereiche der Anzeige darzustellen:

1. Die Haupttabelle zu Anzeige der Gerätedaten,
2. Die Sidebar, die eine Zusammenfassung der Hardwareinformationen zu einem ausgewählten Gerät bereitstellt

Listing 4: Darstellung der Daten in einem MudDataGrid

```
1       <Item Class="datagrid-container" Style="width: calc(100% - @_sidebarWidth - 10px);
            transition: width 0.3s ease-out;">
2       <div>
3           <MudDataGrid T="DeviceContainer"
4                        @ref="_grid"
5                        Hover="true"
6                        ServerData="buildDeviceContainers"
7                        Dense="false"
8                        Hideable="true"
9                        FixedHeader="true"
10                       FixedFooter="false"
11                       Filterable="true"
12                       Height="calc(100vh - 12rem)"
13                       RowClick="@RowClicked"
14                       MultiSelection="false">
15                       .
16                       .
```

Der Item-Container enthält die Komponente der *MudDataGrid*-Komponente, die für die Darstellung und Verwaltung der anzuzeigenden Daten dient.

Sidebar

Die Sidebar bietet lediglich eine Zusammenfassung aller wichtigen Daten beziehungsweise Eigenschaften eines Gerätes, wie Vorlage, Eigenschaft, etc.. Die Detailsseite bietet hierfür eine wesentlich genauere Übersicht.

Implementiert wird die Sidebar durch folgenden Code:

Listing 5: Code der Sidebar

```
1      <div class="sidebar-container @(_selectedDevice != null ? "show" : "")">
2    <div class="sidebar-content">
3      <MudText Typo="Typo.h6" Class="sidebar-header">Details</MudText>
4      <div style="height:70%">
5        <MudTable
6          Items="@propertyData"
7              Style="flex: 1;"
8          Dense="true">
9          <HeaderContent>
10           <MudTh>Typ</MudTh>
11           <MudTh>Wert</MudTh>
12         </HeaderContent>
13         <RowTemplate>
14           <MudTd>@context.Propertyname</MudTd>
15           <MudTd>@context.Value @context.Datatype</MudTd>
16         </RowTemplate>
17       </MudTable>
18
19     </div>
20     <div class="detailsbuttondiv">
21       @if (_selectedDevice != null)
22       {
23         <MudIconButton Icon="@Icons.Material.Filled.KeyboardDoubleArrowRight"
24                OnClick="HideSidebar"></MudIconButton>
25         <MudButton @onclick="ShowDetailsSite" Variant="Variant.Outlined"
26                Color="Color.Secondary">
27           Details/Bearbeiten
28         </MudButton>
29         @*<MudIconButton Icon="@Icons.Material.Filled.FilterNone"
               OnClick="Duplicate"></MudIconButton>*@
30       }
31     </div>
32   </div>
33 </div>
```

Wie durch die Abfrage

```
1      @(_selectedDevice != null ? "show" : "")
```

ersichtlich, wird die Sidebar nur dann eingeblendet, wenn *selectedDevices* nicht null ist.

Um die Sidebar nun zu verstecken, wenn gerade kein Gerät in der Haupttabelle ausgewählt wurde, wird die CSS-Klasse *show* (Siehe Listing 6)verwendet.

Listing 6: CSS show Methode

```
1  .sidebar-container.show {
2     transform: translateX(0);
3  }
```

Mithilfe des *transform*-Befehls wird die Sidebar außerhalb des sichtbaren Bereichs verschoben.

Toolbar

Listing 7: Code für die Toolbar

```
1       <ToolBarContent>
2           @if (editmode)
3           {
4             /*Anlegen Button*/
5             <MudButton Variant="Variant.Filled"
6               .
7                           .
8               OnClick="CreateDevice">Anlegen</MudButton>
9
10            /*Ausscheiden Button*/
11            <MudButton Variant="Variant.Outlined"
12                          .
13                          .
14                          OnClick="Retire">Ausscheiden</MudButton>
15            /*Import Button*/
16            <MudButton Variant="Variant.Outlined"
17                          .
18                          .
19                          OnClick="ImportDialog">Import</MudButton>
20            /*Export Button*/
21            <MudButton Variant="Variant.Outlined"
22                          .
23                          .
24                          OnClick="ExportDialog">Export</MudButton>
25            }
26          <MudTextField @bind-Value="_searchString"
27       .
28       .
29      </MudTextField>
30  </ToolBarContent>
```

Mit diesem Code wird die Toolbar mit den entsprechenden Buttons definiert. Diese Toolbar enthält verschiedene Buttons zur Steuerung der Funktionalitäten der Hauptanzeige.

Listing 8: Bedingte If-Abfrage

```
1       @if (editmode)
2       {
3           ...
4       }
```

Um nun zu gewährleisten, dass die Buttons nur von einem Admin-Benutzer gesehen werden dürfen, wurde (siehe Listing 8), eine If-Abfrage eingeführt, die überprüft, ob der angemeldete Benutzer auch wirklich zu den Administratoren gehört.

Listing 9: Code zur Überprüfung der Berechtigung

```
1       editmode = authenticationState.User.IsInRole("admin") ||
            authenticationState.User.IsInRole("contributor");
```

Ist der '*authenticationsState*' nicht gleich '**Admin**', so wird die Variable *editmode* auf **false** gesetzt und der angemeldete Benutzer bekommt die Buttons nicht zu Gesicht. Innerhalb der bedingten If-Abfrage werden die 4 Buttons mittels der MudBlazor-Komponente **MudButton**, sowie die Suchleiste mithilfe der Komponente **MudTextField** definiert.

Listing 10: Definition der Buttons

```
            /*Anlegen Button*/
        <MudButton Variant="Variant.Filled"
                   .
                   .
          OnClick="CreateDevice">Anlegen</MudButton>

        /*Ausscheiden Button*/
        <MudButton Variant="Variant.Outlined"
                    .
                    .
                    Disabled="_selectedDevice == null"
                    OnClick="Retire">Ausscheiden</MudButton>
        /*Import Button*/
        <MudButton Variant="Variant.Outlined"
                     .
                     OnClick="ImportDialog">Import</MudButton>
        /*Export Button*/
        <MudButton Variant="Variant.Outlined"
                    .
                    .
                    OnClick="ExportDialog">Export</MudButton>
        }
        <MudTextField @bind-Value="_searchString"
          .
          .
          @onkeydown="@(async e => await HandleSearchKeyPress(e))">
        </MudTextField>
```

Titelleiste

Die Titelleiste wird mittels folgendem Code definiert:

Listing 11: Code der Titelleiste

```
<MudAppBar Elevation="2" Style="z-index: 1000">
    <MudIconButton Icon="@DrawerIcon()" Color="Color.Inherit" Edge="Edge.Start"
        OnClick="@((e) => DrawerToggle())" />
    <MudText Typo=Typo.h5 Style="letter-spacing: 2px; user-select:
        none">@LayoutState.PageTitle</MudText>
    <MudSpacer />
    <LoginDisplay />
</MudAppBar>
```

Um den entsprechenden Titel endgültig der Variable '*PageTitle*' zuzuweisen, wird auf jeder Seite der Titel entsprechend dem Listing gesetzt.

Listing 12: Setzen des Titels der Titelleiste

```
1       LayoutState.PageTitle = "Device";
```

Ebenfalls beinhaltet die Titelleiste eine Anzeige über den jeweiligen Benutzer, welcher aktuell eingeloggt ist.

```
1  <MudMenu FullWidth AnchorOrigin="Origin.BottomCenter" Color="Color.Inherit"
       TransformOrigin="Origin.TopCenter">
2       <ActivatorContent>
3           <MudButton StartIcon="@Icons.Material.Filled.Person" Size="Size.Large"
                ButtonType="MudBlazor.ButtonType.Button" Color="Color.Inherit"
                EndIcon="@Icons.Material.Filled.ArrowDropDown">
4                @context.User.Identity?.Name
5           </MudButton>
6       </ActivatorContent>
7       <ChildContent>
8           <MudMenuItem OnClick="BeginLogout">
9               Abmelden
10          </MudMenuItem>
11      </ChildContent>
12 </MudMenu>
```

Mittels der Methode *BeginLogout* Methode (siehe Listing13) wird eine LogOut-Funktion zur Verfügung gestellt, mit dieser sich der eingeloggte Benutzer jederzeit wieder ausloggen kann.

Listing 13: Logout Methode

```
1   public void BeginLogout(MouseEventArgs _)
2   {
3       Navigation.NavigateToLogout("authentication/logout");
4   }
```

3.2.2 UI

Zur Realisierung des Frontends für diese Arbeit wurde sich für die Third-Party Bibliothek MudBlazor entschieden. Wie bereits im Abschnitt MudBlazor beschrieben, handelt es sich hierbei um eine Frontend-Bibliothek zum Erstellen von UI's. MudBlazor wurde auf Wunsch des Unternehmens verwendet, da nahezu sämtliche Applikationen des Unternehmens mithilfe dieser Bibliothek entwickelt wurden und man nicht den Standard des Unternehmens brechen wollte.

Dashboard

Zur Verwirklichung des Dashboards wurden folgende Komponenten verwendet:

Listing 14: Aufbau des Dashboards

```
<MudPaper Height="300px" Width="100%">
<MudContainer  Fixed="true" Style=" width = 80%; text-align: center; align-items:center;
    align-content: center ">
    <MudImage class="no-border" Src="images/ITSupplies-Transparent.png" Elevation="25"
        Height="300" Width="930" />
    <MudButtonGroup Color="Color.Primary" Variant="Variant.Filled"
        Style="margin-left:20px">
        <MudButton Href="https://www.countit.at" Variant="Variant.Filled" Style="width:
            600px" Color="Color.Primary">Count IT Homepage</MudButton>
        <MudButton Href="https://localhost:7040/devices" Style="width:
            600px">Devices</MudButton>
    </MudButtonGroup>
</MudContainer>
</MudPaper>
```

Desweiteren wird mittels **MudButtons** eine Weiterleitung zur Unternehmenswebseite sowie zur Geräteseite ermöglicht (Siehe Listing 15).

Listing 15: Buttons zum Weiterleiten auf die entsprechenden Webseiten

```
<MudButton Href="https://www.countit.at" Variant="Variant.Filled" Style="width: 600px"
    Color="Color.Primary">Count IT Homepage</MudButton>
<MudButton Href="https://localhost:7040/devices" Style="width: 600px">Devices</MudButton>
```

Durch

```
    @page "/"
```

wird sichergestellt, dass das Dashboard die Startseite, also die allererste Seite, die ein Benutzer beim Aufrufen unserer Seite sieht, ist.

Geräte

Ziel beim Entwickeln dieser Seite war es, dem Benutzer ein möglichst effizientes sowie benutzerfreundliches UI zur Verfügung zu stellen. Um dieses Ziel zu erreichen, wurde sich für die *DataGrid*-Komponente des MudBlazor-Frameworks entschieden.

Listing 16: Aufbau der Geräteseite

```
    <MudDataGrid T="DeviceContainer"
                 @ref="_grid"
                 Hover="true"
                 ServerData="buildDeviceContainers"
                 Dense = "false"
                 Hideable="true"
                 FixedHeader="true"
                 FixedFooter="false"
                 Filterable="true"
                 Height="calc(100vh - 12rem)"
                 RowClick="@RowClicked"
                 MultiSelection=false>
        <ToolBarContent>
                        .
                        .
                        .
        </ToolBarContent>

        <!-- Spaltendefinitionen -->
        <Columns>
                        <PropertyColumn Property="x=> x.Name"/>
                        <PropertyColumn Property="x=> x.Location" Title="Standort"/>
                        <PropertyColumn Property="x=> x.SerialNum" Title="Seriennummer"/>
                        <PropertyColumn Property="x=> x.Typename" Title="Geraetetyp"/>
                        <PropertyColumn Property="x=> x.Username" Title="Benutzername"/>
                        <PropertyColumn Property="x=> x.Department" Title="Abteilung"/>
        </Columns>
</MudDataGrid>
```

Durch *Filterable="true"* sind die in *<Columns>* definierten Spalten einzeln filterbar. Mittels *ServerData="buildDeviceContainers"* werden die Daten serverseitig geladen.

Listing 17: buildDeviceContainers Methode

```
private async Task<GridData<DeviceContainer>>
    buildDeviceContainers(GridState<DeviceContainer> gridState)
{
    if (!string.IsNullOrWhiteSpace(_searchString))
    {
        var searchResults = await DevicesApi.SearchDevices(_searchString);

        var deviceContainers = searchResults.Select(d => new DeviceContainer
        {
            DeviceId = d.DeviceId,
            Name = d.Name,
            Location = d.Location,
            SerialNum = d.SerialNum,
            Typename = d.DeviceType?.Typename,
            Username = d.User?.Username,
            Department = d.User?.Department
        }).ToList();

        return new GridData<DeviceContainer>
        {
            TotalItems = deviceContainers.Count,
            Items = deviceContainers
                .Skip(gridState.Page * gridState.PageSize)
                .Take(gridState.PageSize)
                .ToList()
        };
    }
    else
    {
        var listquery = gridState.ToListQuery();
        var deviceContainers = await DeviceContainerApi.GetActiveDevices(listquery);
        return new GridData<DeviceContainer>
        {
            TotalItems = deviceContainers.TotalCount,
            Items = deviceContainers.Items
        };
    }
}
```

Diese Methode wird nicht nur beim ersten Laden der Seite aufgerufen, sondern auch wenn der *MudDataGrid* Daten benötigt, beispielsweise beim Seitenwechseln, Filtern oder bei Suchanfragen über die Suchleiste. Der *MudDataGrid* kann aus zwei unterschiedlichen Datenquellen bestehen. Wenn eine Suchanfrage existiert, beispielsweise bei der Suche über die Suchleiste, werden im *MudDataGrid* die gefilterten Ergebnisse geladen und angezeigt. Wird hingegen nichts über die Suchleiste gesucht und die Seite ganz normal aufgerufen, so werden mittels einer API-Abfrage ans Backend alle Geräte abgefragt und angezeigt. Durch

```
1       .Skip(gridState.Page * gridState.PageSize)
```

wird sichergestellt, dass nur die aktuell benötigten Datensätze an das *MudDataGrid* zurückgegeben werden. Dies reduziert die Datenlast und die Ladezeiten, da nicht alle Datensätze auf einmal geladen werden müssen.

Durch *RowClick = "@RowClicked"* wird festgelegt, wann eine Zeile angeklickt worden ist. Dies wird benötigt, da bei Klick auf eine Zeile eine Sidebar angezeigt werden soll.

Listing 18: RowClicked Methode

```
1  async void RowClicked(DataGridRowClickEventArgs<DeviceContainer> args)
2  {
3      _searchString = string.Empty;
4    var selectedDevice = args.Item;
5
6    if (!sideBaropen)
7    {
8      ToggleSidebar();
9    }
10   else if (sideBaropen && _selectedDevice.DeviceId == selectedDevice.DeviceId)
11   {
12     ToggleSidebar();
13     _selectedDevice = null;
14     return;
15   }
16
17   if (selectedDevice != null)
18   {
19     int firstDeviceId = selectedDevice.DeviceId;
20     _selectedDevice = selectedDevice;
21     propertyData = await GetProperties(firstDeviceId);
22     StateHasChanged();
23   }
24 }
```

Die Sidebar wird je nach Zustand geschlossen oder geöffnet:

1. Wenn die Sidebar geschlossen und ein Gerät ausgewählt ist, wird *ToggleSidebar()* aufgerufen um die Sidebar zu öffnen.

2. Wenn die Sidebar **bereits geöffnet** ist und dasselbe Geräte **wieder angeklickt** wurde,

   ```
   1          else if (sideBaropen && _selectedDevice.DeviceId == selectedDevice.DeviceId)
   ```

   wird die Sidebar geschlossen und die Auswahl des ausgewählten Gerätes zurückgesetzt.

3. Wird ein neues Gerät ausgewählt, wenn die Sidebar geöffnet ist, so wird das neu ausgewählte Gerät übernommen.

```
1           _selectedDevice = selectedDevice;
```

Nach dem übernehmen des neuen Gerätes werden die Infos von diesem geladen und diese entsprechend in die Sidebar geladen.

```
1           propertyData = await GetProperties(firstDeviceId);
2       StateHasChanged();
```

Um beispielsweise ein neues Gerät manuell anzulegen, wird ein Dialogfenster geöffnet, in diesem die entsprechenden Werte eingegeben werden müssen. Dies wird durch die *MudDialog*-Komponente von MudBlazor ermöglicht.

Listing 19: Dialogfenster für das Erstellen eines neuen Gerätes

```
1  <MudDialog>
2      <DialogContent>
3          <MudTextField Label="Name *" @bind-Value="Device.Name" For="@(() => Device.Name)"
               />
4          <MudTextField Label="Standort" @bind-Value="Device.Location" For="@(() =>
               Device.Location)" />
5          <MudTextField Label="Seriennummer" @bind-Value="Device.SerialNum" For="@(() =>
               Device.SerialNum)" />
6          .
7          .
8  </MudDialog>
```

Die Benutzerzuweisung eines manuell erstellten Gerätes erfolgt über ein Dropdown-Menü.

Listing 20: Benutzerauswahl mittels Dropdown

```
1  <MudSelect T="UserContracts" @bind-Value="@selectedUser" ToStringFunc="@converterUser"
       Label="Benutzer *" Variant="Variant.Text" AnchorOrigin="Origin.BottomCenter"
       Dense="true" Style="margin-right: 10px">
2      @foreach (UserContracts p in users)
3      {
4          <MudSelectItem Value="@p"> @p.Username (@p.Department) </MudSelectItem>
5      }
6  </MudSelect>
```

Dabei werden alle in der Datenbank gespeicherten Benutzer über einen API-Aufruf in die Variable *users* gespeichert und ein bestimmter Benutzer kann somit gesucht und dem Gerät zugewiesen werden.

Wenn eine Vorlage ausgewählt werden soll, so ist dies mittels einem Dropdown-Menü (siehe Listing 21) möglich.

Listing 21: Vorlagenauswahl mittels Dropdown

```
1   <MudSelect T="DeviceTemplateContracts" ValueChanged="OnTempValueChanged"
        Value="@selectedTemplate" ToStringFunc="@converterTemplate" Label="Vorlage
        auswaehlen" Variant="Variant.Text" AnchorOrigin="Origin.BottomCenter" Dense="true"
2               Style="margin-right: 10px">
3       @foreach (DeviceTemplateContracts p in deviceTemplates)
4       {
5           <MudSelectItem Value="@p" />
6       }
7
8   </MudSelect>
```

Es werden alle aktuell vorhanden Vorlagen mittels API-Aufruf abgefragt und in dem Dropdown-Menü angezeigt.

Vorlagen

Die Vorlagenseite besteht genau wie die Geräteseite ebenfalls aus einer *MudDataGrid*-Komponente des MudBlazor-Frameworks.

Listing 22: Aufbau der Vorlagenseite

```
<MudDataGrid T="TemplateContainer"
             Items="@templateContainer"
             Hover="true"
             Loading="_isLoading"
             Dense="false"
             Hideable="true"
             FixedHeader="true"
             FixedFooter="false"
             Filterable="true"
             Height="calc(100vh - 12rem)"
             RowClick="@RowClicked"
             MultiSelection=true>
            .
            .
            .
</MudDataGrid>
```

Um eine neue Vorlage anlegen zu können, wird ein Dialogfenster geöffnet, auf dem die entsprechenden Werte eingegeben werden müssen.

Listing 23: Dialogfenster zur Erstellung einer neuen Vorlage

```
    <MudDialog>
    <DialogContent>
        <MudTextField Label="Name * " @bind-Value="template.TemplateName" For="@(() =>
            template.TemplateName)" />

        <MudSelect T="DeviceTypeContracts"
                   @bind-Value="@selectedDeviceType">
            @foreach (DeviceTypeContracts p in deviceTypes)
            {
                <MudSelectItem Value="@p" />
            }
        </MudSelect>
    </DialogContent>
    <DialogActions>
        .
        .
        .
    </DialogActions>
</MudDialog>
```

Dieses Dialogfenster besteht, ebenso wie das Dialogfenster zum Erstellen eines neuen Gerätes (siehe Listing 26), aus der MudBlazor-Komponente *MudDialog*. Das Dialogfenster besteht aus einem Textfeld für den Namen der Vorlage

```
    <MudTextField Label="Name * " @bind-Value="template.TemplateName" For="@(() =>
        template.TemplateName)" />
```

, sowie einem Dropdown-Menü (siehe Listing 24), aus dem die bereits vorhandenen Gerätetypen ausgewählt werden können.

Listing 24: Dropdown-Menü zur Auswahl der Gerätetypen

```
<MudSelect T="DeviceTypeContracts"
           @bind-Value="@selectedDeviceType"
           ToStringFunc="@converterDeviceType"
           Label="Geraetetyp * "
           Variant="Variant.Text"
           AnchorOrigin="Origin.BottomCenter"
           Dense="true"
           Style="margin-right: 10px">
    @foreach (DeviceTypeContracts p in deviceTypes)
    {
        <MudSelectItem Value="@p" />
    }
</MudSelect>
```

Um zu erreichen, dass alle Gerätetypen korrekt in dem Dropdown-Menü angezeigt werden, wird ein API-Aufruf ins Backend gemacht, mit diesem alle Gerätetypen geholt und in die Variable *selectedDeviceType* gespeichert werden.

```
    DeviceTypeContracts selectedDeviceType = new DeviceTypeContracts();
    .
    .
    var dtresult = await DeviceTypeApi.Get(template.DeviceTypeId ?? default(int));
    if (dtresult.StatusCode != System.Net.HttpStatusCode.NoContent)
    {
        selectedDeviceType = dtresult.Content;
    }
```

Benutzer

Ebenso wie bisher alle anderen Seiten basiert die Darstellung der Benutzer auf einem *MudGrid*:

Listing 25: Aufbau der Benutzerseite

```
<MudGrid Container="true">
  <div style="max-height: calc(100vh - 8rem); flex: auto">
    <MudDataGrid T="UserContracts"
                 Items="@users"
        .
                 @bind-SelectedItems="_selectedUsers"
                 MultiSelection="true"
                 Virtualize="true"
                 style="box-shadow: none; border: none;">
                 .
                 .
 </MudGrid>
```

Erwähnenswerter Unterschied zu den anderen Seiten ist die Verwendung folgender Codezeile:

```
    Virtualize="true"
```

Durch diese Zeile wird das sogenannte *virtuelle Scrolling* im *MudDataGrid* aktiviert. Dadurch werden nur die Einträge im *MudDataGrid* gerendert, die der Benutzer aktuell angezeigt bekommt. Diese Änderung wurde benötigt, da das *MudDataGrid* mit dem Anzeigen von allen Benutzern auf einmal überfordert war und die Ansicht somit beim Scrollen ruckelte. Dadurch wird eine ruckelfreie Bedienung dieser Seite ermöglicht.

Um die Erstellung eines neuen Benutzers zu ermöglichen, wurde wieder ein Dialogfenster implementiert. Der Aufbau eben dieses Dialogfensters (siehe Listing 26) zur Erstellung eines neuen Benutzers ist im Grundsatz gleich zu den bisher beschriebenen Dialogfenstern der Geräte- und Vorlagenseite.

Listing 26: Beispiel für ein Dialogfenster mit Eingabefelder

```
<MudDialog>
    <DialogContent>
        <MudTextField Label="Name" @bind-Value="User.Username" For="@(() =>
            User.Username)" Required= "true" />
        <MudTextField Label="Abteilung" @bind-Value="User.Department" For="@(() =>
            User.Department)" />

    </DialogContent>
    <DialogActions>
        <MudButton OnClick="Cancel">Abbrechen</MudButton>
        <MudButton Color="MudBlazor.Color.Secondary"
                   Variant="Variant.Filled"
                   DisableElevation="true"
                   ButtonType="ButtonType.Submit">@(User.UserId == default(int) ?
                       "Anlegen" : "Aktualisieren")</MudButton>
    </DialogActions>
</MudDialog>
```

Der größte Unterschied zu den anderen Seiten ist, dass dieses Dialogfenster über kein Dropdown-Menü verfügt, sondern lediglich über 2 Textfelder, mit denen die Eingabe der Daten ermöglicht wird.

Eigenschaften

Diese Seite ist genauso aufgebaut, wie sämtliche bisherigen Seiten.

Listing 27: Aufbau der Eigenschaftenseite

```
<MudGrid Container="true">
    <div style="max-height: calc(100vh - 8rem); flex: auto">
        <MudDataGrid T="PropertyContracts"
                     .
                     .
                     @bind-SelectedItems="_selectedProperties" MultiSelection="true"
                     style="box-shadow: none; border: none;">
</MudGrid>
```

Es wird wieder eine *MudDataGrid*-Komponente verwendet (siehe Listing 27), in dieser sämtliche Eigenschaften, die ein Gerät annehmen kann, gespeichert und auch angezeigt werden. Um eine neue Eigenschaft anzulegen, wurde ebenso wie schon auf anderen Seiten, ein entsprechendes Dialogfenster(siehe Listing 26) mit den entsprechenden Eingabefeldern implementiert.

Gerätetypen

Auch diese Seite ist im Aufbau grundsätzlich ident zu sämtlichen bisher beschriebenen Seiten.

Listing 28: Aufbau der Gerätetypenseite

```
<MudGrid Container="true">
    <div style="max-height: calc(100vh - 8rem); flex: auto">
        <MudDataGrid T="DeviceTypeContracts"
            Items="@templateContainer"
            Hover="true"
            Loading="_isLoading"
            Dense="false"
            Hideable="true"
            FixedHeader="true"
            FixedFooter="false"
            Filterable="true"
            Height="calc(100vh - 12rem)"
            RowClick="@RowClicked"
            MultiSelection=true>
            .
            .
            .
 </MudDataGrid>
```

Zum manuellen Erstellen eines neuen Gerätetypen wird ein Dialogfenster geöffnet, welches über zwei Textfelder zur Eingabe der Daten verfügt.

Listing 29: Dialogfenster zur Erstellung eines neuen Gerätetypen

```
<MudDialog>
    <DialogContent>
        <MudTextField Label="Name" @bind-Value="Property.Propertyname" For="@(() =>
            Property.Propertyname)" Required="true" />
        <MudTextField Label="Datentyp" @bind-Value="Property.Datatype" For="@(() =>
            Property.Datatype)" />
        .
    </DialogContent>
</MudDialog>
```

Details

Diese Seite wurde als eine Erweiterung zu der Sidebar auf der Geräteseite implementiert. Auf der Sidebar werden, wie im Kapitel Aktive Geräte schon beschrieben, sämtliche Daten zu einem Gerät, Eigenschaften, etc.. zusammengefasst angezeigt. Diese Detailsseite ist nun eine wesentlich umfangreichere Version dieser Sidebar. Hier ist es zum Beispiel auch möglich, etwa Eigenschaften von Geräten umzubenennen, zu löschen, nachträglich noch hinzuzufügen und vieles mehr. Aufgebaut ist diese Seite etwas unterschiedlich zu den bisherigen Seiten:

Listing 30: Aufbau der Detailsseite

```
<div class="cit-main-content">
  <div>
    .
        .
  </div>
    .
    .
</div>
```

Zuerst verfügt diese über zwei verschachtelte * *-Elemente, welche die Buttons für das Zurückgehen, zum Ausscheiden eines Gerätes, sowie einen Button zur Aktivierung des Bearbeitungsmodus beinhalten. In diesem Bearbeitungsmodus verwandeln sich sämtliche Textfelder in Eingabefelder, sodass beispielsweise der Name des Gerätes einfach geändert werden kann.

Listing 31: Aufbau des DataGrid

```
<MudDataGrid T="PropertyContainer"
             Items="@ElementsProp"
             Filterable="false"
             Dense="true"
             Hideable="true"
             ReadOnly="@_canEdit"
             EditMode="DataGridEditMode.Cell"
             CommittedItemChanges="@CommittedPropChanges"
             QuickFilter="@_quickFilter"
             Class="fixed-header-datagrid">
    <Columns>
        .
        .
    </Columns>
</MudDataGrid>
```

Ermöglicht wird dieser Bearbeitungsmodus, indem bei Klick des Buttons *Bearbeiten* eine Variable *canEdit* auf **true** gesetzt wird, sodass im *MudDataGrid* das *ReadOnly*-Attribut

```
ReadOnly="@_canEdit"
```

nun auf *false* gesetzt wird, wodurch nun auch die Eigenschaften in den Eingabefeldern bearbeitet werden können, da das *ReadOnly*-Attribut dafür zuständig ist, den gesamten **MudDataGrid** *ReadOnly*, also nur lesbar zu halten.

3.3 Backend Implementierung

3.3.1 Authentifizierung

Ablauf der Authentifizierung:

Der Vorgang der Authentifizierung eines Users wird mittels Azure AD vorgenommen.

1. **Starten der Anwendung / Zugriff auf die Webanwendung:**
   Der Benutzer greift auf die Anwendung zu / startet sie.
2. **Anmeldung bei Azure AD:**
   Weiterleitung zur Azure AD-Anmeldeseite
3. **Anmeldung bei Azure AD:**
   Benutzer gibt seine Anmeldedaten ein
4. **Azure AD sendet Benutzerdaten zurück:**
   Eingebenene Benutzerdaten werden an uns zurückgesendet
5. **Verarbeitung der Anmeldedaten:**
   Anmeldedaten des Benutzers werden verarbeitet.
6. **Benutzer ist authentifiziert:**
   Zugriff auf die Anwendung wird freigegeben.

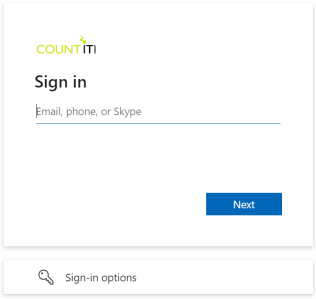


Abbildung 22: Azure-AD Loginfenster der Anwendung

3.3.2 Grundstruktur/Projekte in der Solution

Das gesamte Backend wurde in der Programmiersprache C# entwickelt und besteht aus folgenden Projekten:

- CIT.GERI.Contracts
- CIT.GERI.API
- CIT.GERI.AGNES
- Solution Items

3.3.3 CIT.Geri.API

In diesem Abschnitt wird die Struktur des Projekts CIT.GERI.API und dessen Klassen mit ihren Funktionalitäten beschrieben. Die API stellt die zentrale Schnittstelle zwischen der Datenbank, den Geschäftslogiken und externen Systemen dar. Sie verarbeitet eingehende Anfragen und kommuniziert mit den verschiedenen Services. Damit wird eine strukturierte und effiziente Datenverarbeitung möglich gemacht.

Services

Die Services sind wichtig für die Geschäftslogik des Programms. Sie bieten Methoden zur Verwaltung von Geräten, Benutzern und deren Eigenschaften. Diese Methoden interagieren direkt mit dem Datenbankkontext. Dabei kapseln sie komplexe Operationen voneinander ab, um eine saubere Trennung zwischen Datenbankzugriff und Anwendungslogik sicherzustellen. Diese Struktur erleichtert die Wartung, Erweiterung und Wiederverwendbarkeit des Programms.

Folgende Services sind im Programm enthalten:

1. **DeviceService:**
   Ist für die Verarbeitung von den Devices (Geräten) zuständig. Der DeviceService bietet folgende Funktionalitäten:

   CreateDevice: Erstellt ein neues Gerät.

   GetActiveDevices: Gibt alle aktiven Geräte zurück.

2. **PropertyService:**
   Der PopertyService ist für die Verarbeitung von den Properties (Eigenschaften) und der Devices (Geräten) zuständig. Die Klasse PropertyService bietet folgende Funktionalitäten:

GetProperties: Ruft alle Eigenschaften aus der Datenbank ab.

GetPropertyID: Sucht die ID einer Eigenschaft basieren auf den Namen der Eigenschaft.

CreateProperty: Erstellt eine neue Eigenschaft in der Datenbank, falls diese noch nicht existiert.

3. **UserService:**
   Ist für die Verarbeitung von den Benutzern (User) zuständig. Der UserService bietet folgende Funktionalität:

   CheckorCreateUser: Prüft ob ein Benutzer bereits in der Datenbank vorhanden ist. Wenn er vorhanden ist wird keine Aktion durchgeführt. Wenn dieser Benutzer nicht vorhanden ist, dann wird dieser in der Datenbank als User gespeichert.

4. **DevicePropertyService:**
   Verarbeitet die Verknüpfung zwischen Geräten und Eigenschaften. Der DeviceProperty-Service bietet folgende Funktionalität:

   CheckOrCreateDeviceProperty: Die Methode CheckOrCreateDeviceProperty ist eine zentrale Funktionalität im Backend, die dazu dient, Verknüpfungen zwischen Geräten und Eigenschaften (DeviceProperty) in der Datenbank zu prüfen und gegebenenfalls zu aktualisieren oder neue Verknüpfungen hinzuzufügen. Es handelt sich also um eine sogenannte 'Upsert'-Operation (Update or Insert), die sicherstellt, dass die Datenbank den aktuellen Stand der Verknüpfungen veranschaulicht.

Interface für Services

Für jede Serviceklasse wurde ein Interface entwickelt, um die Struktur und die Funktionalität von Diensten zu definieren.

Folgender Code zeigt eines dieser Interfaces

Listing 32: Code vom Interface IUserService

```
using CIT.Geri.Contracts;

namespace CIT.Geri.API.Interfaces
{
    public interface IUserService
    {
        Task<List<UserContracts>> GetUsers();
        Task<UserContracts?> GetUserById(int id);
        Task CreateUser(UserContracts user);
        Task UpdateUser(UserContracts user);
        Task DeleteUser(int id);
        Task CheckOrCreateUser(List<UserContracts> users);
    }
}
```

3.3.4 DataProcessing

Die Klasse DataProcessing ist eine der wichtigsten Klassen von der CIT.GERI.API. Ihre Hauptaufgabe besteht darin, eingehende Gerätedaten zu verarbeiten, Benutzer-Geräte-Zuweisungen zu verwalten und sicherzustellen, dass alle relevanten Informationen in der Datenbank gespeichert werden.

Sie agiert als Kernmodul zwischen der Datenaufnahme und der permanenten Speicherung in der Datenbank, indem sie Services aufruft, die für die Speicherung und Aktualisierung von Geräten, Benutzern und deren Eigenschaften zuständig sind.

Wie arbeitet die DataProcessing-Klasse?

1. **Eingabe der Daten:**
   Die DataProcessing-Klasse bekommt die Daten in strukturierter Form über ein HardwareInformations-Objekt.

   Dieses Objekt enthält alle relevanten Informationen zu einem Gerät, einschließlich:

   a) **Allgemeine Daten:** Gerätetyp, Gerätename, Seriennummer, aktueller Benutzer.

   b) **Komponenteninformationen:** Prozessor, Speicher, Festplatten, Grafikkarten, USB-Geräte und Monitore.

   Jede dieser Kategorien wird in einem passenden Dictionary bereitgestellt.

Ablauf der Methode ProcessData() in der DataProcessing-Klasse:
Die Methode **ProcessData()** verarbeitet die eingehenden Gerätedaten schrittweise und sorgt dafür, dass alle relevanten Informationen korrekt gespeichert werden.

Zuerst werden allgemeine Gerätedaten eingelesen. Dazu gehören der **Gerätename**, die **Seriennummer** und der **aktuelle Benutzer**.

Anschließend werden die Hardware-Komponenten des Geräts analysiert. Dabei werden die **CPU-Daten** verarbeitet, der **Arbeitsspeicher** ausgelesen, die **Festplatten** analysiert, die **GPU** erkannt, angeschlossene **USB-Geräte** erfasst und die **vorhandenen Monitore** registriert.

Nach der Verarbeitung der Gerätedaten wird die automatische Benutzerzuweisung durchgeführt. Dabei wird geprüft, ob der Benutzer bereits existiert oder neu angelegt werden muss.

Zum Abschluss werden die Eigenschaften des Geräts gespeichert. Dazu gehören alle relevanten Hardware-Informationen als auch die Zuordnung der gespeicherten Gerätedaten in der Datenbank.

Dadurch wird sichergestellt, dass sowohl das Gerät als auch dessen Eigenschaften und der zugehörige Benutzer korrekt in der Datenbank gespeichert werden.

Folgende Methoden wurden für die zuvor beschriebenen Anwendungen entwickelt:

1. **ProcessGeneralInfo()**
2. **ProcessProcessorInfo()** – Speichert CPU-Details.
3. **ProcessMemoryInfo()** – Speichert RAM-Daten.
4. **ProcessDiskInfo()** – Speichert Festplattendetails.
5. **ProcessVideoControllerInfo()** – Speichert GPU-Informationen.
6. **ProcessUSBDeviceInfo()**
7. **ProcessMonitorInfo()**
8. **CheckOrCreateDeviceProperty()**

Siehe Funktionen in der Abbildung 33

Listing 33: Methodenaufrufe ProcessData()

```
public async Task ProcessData(HardwareInformations hardwareInfo)
{
    try
    {
        await ProcessGeneralInfo(hardwareInfo.GeneralInfo);
        await ProcessProcessorInfo(hardwareInfo.ProcessorInfo);
        await ProcessMemoryInfo(hardwareInfo.MemoryInfo);
        ProcessDiskInfo(hardwareInfo.DiskInfo);
        await ProcessVideoControllerInfo(hardwareInfo.VideoControllerInfo);
        ProcessUSBDevicesInfo(hardwareInfo.USBDevicesInfo);
        await ProcessMonitorInfo(hardwareInfo.MonitorInfo);

        Console.WriteLine("[INFO] Daten erfolgreich verarbeitet.");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error in ProcessData: {ex.Message}");
    }
}
```

Detaillierte Beschreibung der Methode ProcessGeneralInfo()

Diese Methode verarbeitet Gerätedaten, die unter anderem über den Namen des PCs, die Seriennummer und den aktuell angemeldeten Benutzer verfügen. Dabei wird der Gerätename direkt gespeichert. Die Seriennummer dient zur eindeutigen Identifikation in der Datenbank und der Benutzername wird in der **'User-Queue'** eingetragen, um später den tatsächlichen Besitzer des Geräts zu bestimmen. Nebenbei wird mithilfe der Seriennummer überprüft, ob das Gerät bereits existiert oder neu erstellt werden muss.

Detaillierte Beschreibung der Methode ProcessProcessorInfo()

Die Methode **ProcessProcessorInfo()** verarbeitet die **CPU-Daten** eines Geräts. Sie liest den Namen des Prozessors aus, speichert ihn als Eigenschaft und verknüpft ihn mit dem Gerät. Falls der Prozessor noch nicht in der Datenbank existiert, wird er neu erstellt.

Detaillierte Beschreibung der Methode ProcessMemoryInfo()

Hier wird der Arbeitsspeicher eines Geräts analysiert. Die Methode erfasst die Anzahl der **RAM-Module** sowie deren Kapazität und speichert diese als Geräteeigenschaften. Jede Speichereinheit wird einzeln verarbeitet und dem dazugehörenden Gerät zugeordnet.

Detaillierte Beschreibung der Methode ProcessDiskInfo()

Diese Methode liest die Festplattendaten aus. Dazu gehören Modellname, Speicherkapazität und Typ **(HDD oder SSD)**. Falls mehrere Festplatten vorhanden sind, werden alle als separate Eigenschaften gespeichert.

Detaillierte Beschreibung der Methode ProcessVideoControllerInfo()

Hier werden Informationen zur Grafikkarte eines Geräts erfasst. Die Methode liest den Namen der **GPU** aus, speichert ihn als Geräteeigenschaft und stellt sicher, dass die Informationen mit dem richtigen Gerät in der Datenbank gespeichert werden.

Detaillierte Beschreibung der Methode ProcessUSBDeviceInfo()

Diese Methode analysiert die angeschlossenen USB-Geräte. Auch wenn in der aktuellen Implementierung keine aktiven Daten gespeichert werden, ist die Methode vorbereitet, um in Zukunft USB-spezifische Informationen zu verarbeiten. Die Methode liest die angeschlossenen USB-Geräte aus dem Dictionary aus. Aktuell bleibt die Funktion jedoch leer und dient als Platzhalter für eine spätere Erweiterung.

Detaillierte Beschreibung der Methode ProcessMonitorInfo()

Diese Methode verarbeitet Informationen über angeschlossene Monitore. Jeder Monitor wird dabei als separates Objekt erfasst. Für jeden Monitor wird der Name gespeichert und dem Gerät als Eigenschaft zugewiesen. Die Monitorbezeichnungen werden aus dem mitgelieferten Dictionary gelesen und verarbeitet in die Datenbank überführt.

Detaillierte Beschreibung der Methode CheckOrCreateDeviceProperty()

Die Methode **CheckOrCreateDeviceProperty()** stellt sicher, dass die Eigenschaften eines Geräts korrekt in der Datenbank gespeichert werden. Sie überprüft, ob bereits existierende Geräteeigenschaften *(DeviceProperty)* vorhanden sind, und speichert neue Eigenschaften nur dann, wenn sie noch nicht existieren.

Ablauf der Methode CheckOrCreateDeviceProperty()

1. **Datenbankabruf:** Die Methode ruft zuerst alle bereits gespeicherten *DeviceProperty-Einträge* aus der Datenbank ab. Dadurch wird vermieden, dass doppelte Eigenschaften für ein Gerät gespeichert werden.
2. **Vergleich mit neuen Eigenschaften:** Für jede neue Eigenschaft, die in *_deviceProperties* gespeichert ist, wird geprüft, ob diese bereits in der Datenbank existiert. Falls eine Eigenschaft bereits vorhanden ist, wird sie nicht erneut gespeichert.
3. **Speicherung neuer Eigenschaften:** Falls eine Eigenschaft noch nicht existiert, wird diese in die Datenbank hinzugefügt und gespeichert. Dadurch wird sichergestellt, dass nur neue und relevante Eigenschaften gespeichert werden.
4. **Datenbank-Aktualisierung:** Nachdem alle neuen Eigenschaften hinzugefügt wurden, werden die Änderungen mit *SaveChangesAsync()* gespeichert.

Folgender Code repräsentiert die Methode CheckOrCreateDeviceProperty()

Listing 34: Code von CheckOrCreateDeviceProperty()

```
1  public async Task CheckOrCreateDeviceProperty()
2  {
3      var existingProperties = await
           _devicePropertyService.GetDeviceProperties(_device.DeviceId);
4
5      foreach (var property in _deviceProperties)
6      {
7          if (!existingProperties.Any(p => p.PropertyId == property.PropertyId && p.DeviceId
               == property.DeviceId))
8          {
9              await _devicePropertyService.CreateDeviceProperty(property);
10         }
11     }
12 }
```

3.3.5 Controller

Die Controller sind verantwortlich für die Verarbeitung von HTTP-Anfragen. Sie dienen als Brücke zwischen dem Client und den eigentlichen Diensten, die die Hauptgeschäftslogik beinhalten. Es gibt grundlegende Regeln, nach denen das System arbeitet, damit die Anfragen klar strukturiert und effizient verarbeitet werden können.

1. **Hauptaufgabe: Vermittler zwischen Client und Geschäftslogik**

   Die Verantwortung der Controller besteht darin, die eingehende HTTP-Anfrage vom Client zu bearbeiten. Sie verwalten das Routing und leiten die Anfragen im Wesentlichen an den richtigen Endpunkt weiter. (wie z.B.: *api/users*, *api/device*, etc.).

   Code Beispiel für einen dieser Endpunkte 35:

Listing 35: Code-Beispiel Endpunkt User

```
1       [HttpGet]
2       [Route("api/users")]
3       public async Task<ActionResult<IEnumerable<UserContracts>>> GetUsers()
4       {
5           var users = await _userService.GetUsers();
6           return Ok(users);
7       }
```

   Die übermittelten Daten werden überprüft, bevor die Anfrage verarbeitet wird. Dies umfasst sowohl den JSON-Body als auch Abfrageparameter. Diese Überprüfung stellt sicher, dass die bereitgestellten Details korrekt und vollständig sind. Für die Validierung werden .NET DataAnnotations[18] und FluentValidation[19] verwendet, um sicherzustellen, dass die eingehenden Anfragen die jeweiligen Anforderungen erfüllen.

   In den Controllern gibt es keine Geschäftslogik. Vielmehr leiten sie die Anfragen zur Verarbeitung an die richtigen Dienste weiter, wo die eigentliche Geschäftslogik liegt und wo Datenbankeinträge erstellt oder aktualisiert werden. Dies hält den Code sauber, modular und leicht erweiterbar.

2. **Verarbeitung der HTTP-Anfrage**

   Controller in CIT.GERI.API sind darauf ausgelegt, verschiedene HTTP-Methoden zu verarbeiten. Je nach Art der Anfrage führt der Controller eine der folgenden Aktionen aus:

   - **GET**

     Wird dazu verwendet, um Daten abzurufen, beispielsweise eine Liste aller Benutzer.

   - **POST**

     Dient dazu, neue Datensätze zu erstellen, etwa wenn ein Benutzer registriert wird.

   - **PUT**

     Ermöglicht das Aktualisieren bestehender Daten, beispielsweise das Ändern von Benutzerinformationen.

   - **DELETE**

     Wird dafür genutzt, um bestimmte Einträge aus der Datenbank zu entfernen, wie das Löschen eines Benutzerkontos.

Nach der Verarbeitung sendet der Controller eine HTTP-Antwort an den Client zurück. Die Antwort erfolgt im JSON-Format und enthält alle relevanten Informationen, die für den Client wichtig sind. Neben den angeforderten Daten, wie beispielsweise Benutzerprofilen, enthält die Antwort auch einen Statuscode, der den Erfolg oder mögliche Fehler zurückgibt. Typische Statuscodes sind *200 OK* für eine erfolgreiche Anfrage oder *400 Bad Request*, falls die übermittelten Daten fehlerhaft sind. Wenn ein ein Problem auftritt, so wird zusätzlich eine detaillierte Fehlermeldung zurückgegeben.

3. **Eigenschaften der Controller im Projekt**
   Die Controller in der API nutzen attributbasiertes Routing. Das heißt, dass jede Methode mit einem Attribut wie [HttpGet] oder [HttpPost] gekennzeichnet und direkt mit einer bestimmten Anfrageart (GET, POST usw.) verbunden ist.

   Durch Dependency Injection werden externe Dienste wie z.B. *IUserService* oder *IPropertyService* automatisch in die Controller eingefügt. Die Controller müssen diese Dienste also nicht selbst erstellen. Deswegen bleibt der Code sauber und jeder Teil hat nur eine klar definierte Aufgabe.

   Ein weiterer Hauptpunkt ist die Fehlerbehandlung: Wenn zum Beispiel eine Ressource nicht gefunden wird, antwortet der Controller mit einem „*404 Not Found*“. Nicht abgefangene Fehler führen zu einem „*500 Internal Server Error*“.

   Für bessere Leistung arbeiten die Methoden asynchron (mit async/await). Das bedeutet, dass der Server während längerer Aufgaben (z.B. bei Datenbankabfragen) nicht stehen bleibt und deshalb andere Anfragen weiter bearbeiten kann. Das macht die Anwendung schneller und besser skalierbar.

4. **Zusammenspielt mit anderen Schichten**
   Die Controller in CIT.GERI.API sind Teil einer mehrschichtigen Architektur, die eine klare Trennung der Verantwortlichkeiten sicherstellt. Dabei arbeiten die Controller mit anderen Komponenten des Backends zusammen, um die Daten effizient zu verarbeiten und weiterzugeben.

   Die Modelle definieren die Datenstrukturen, die zwischen dem Client und dem Backend ausgetauscht werden. Sie enthalten alle relevanten Informationen, die für die Kommunikation zwischen API und Client erforderlich sind, und sorgen damit für eine reibungslose Datenübertragung.

   Die Services enthalten die eigentliche Geschäftslogik der Anwendung. Während die Controller lediglich für die Entgegennahme und Weiterleitung von Anfragen zuständig sind,

führen die Services die notwendigen Verarbeitungen durch. Sie validieren Daten, führen Umwandelungen der Daten aus und kommunizieren mit den darunterliegenden Schichten.

Die Repositories übernehmen schließlich den direkten Zugriff auf die Datenbank. Services greifen auf diese Repositories zu, um Datensätze abzurufen, zu speichern oder zu aktualisieren. Dadurch bleibt die Datenbanklogik sauber von der Geschäftslogik getrennt, was die Wartung und Skalierbarkeit der Anwendung verbessert.

Diese klare Schichtentrennung sorgt für eine übersichtliche, modularisierte und gut wartbare Architektur, bei der jede Komponente eine spezifische Aufgabe erfüllt.

3.3.6 Program.cs in CIT.GERI.API

Die Datei Program.cs ist die Startdatei der API, die die Middleware-Komponenten einrichtet und Dependency Injection, API-Sicherheit usw. konfiguriert. Folgende Einstellungen werden unter anderem vorgenommen:

- **Dependency Injection:**
  Registriert Dienste, z. B. *IUserService* oder *IDeviceService*.
- **Middleware:**
  CORS-Regeln, Fehlerbehandlung, API-Sicherheit.
- **Swagger-Integration:**
  Ermöglicht die API-Dokumentation für Entwickler.
- **Routing- und Endpunktkonfiguration:**
  Bestimmt, wie HTTP-Anfragen verarbeitet werden.

Folgender Code Zeigt einen teil der Program.cs 36

Listing 36: Code von Program.cs

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddScoped<IDeviceService, DeviceService>();
builder.Services.AddScoped<IPropertyService, PropertyService>();
builder.Services.AddScoped<IDevicePropertyService, DevicePropertyService>();
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseAuthorization();
app.MapControllers();
await app.RunAsync();
```

3.3.7 launchSettings.json

Die **launchSettings.json** definiert, wie die Anwendung beim Start funktioniert. In ihr wird festgelegt, auf welchen Ports die API läuft, welche Umgebung verwendet wird und ob beim Start automatisch Swagger im Browser geöffnet werden soll.Auch können noch mehr verschiedene Startoptionen für die API festgelegt werden.

Fehlerverarbeitung: Die API kann in verschiedenen Umgebungen (Entwicklung, Produktion) eingesetzt werden, sodass Logging und Fehlerverarbeitung abhängig von der Umgebung sind.

Folgender Codeausschnitt zeigt, dass in der **launchSettings.json**, Swagger automatisch im Browser geöffnet wird, sobald das Programm gestartet wird:

Listing 37: Code launchSettings.json

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:5000",
      "sslPort": 44300
    }
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "CIT.Geri.API": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "http://localhost:5001;https://localhost:5002",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

3.3.8 CIT.Geri.AGNES

Die CIT.GERI.AGNES besteht aus zwei *.cs* Programmen:

1. HardwareInformations.cs
2. Program.cs

HardwareInformations.cs

In diesem Programm werden Objekte definiert, die im Program.cs verwendet werden.

Code der Objekte 38

Listing 38: Code Hardwareinformations

```
public Dictionary<string, string?> GeneralInfo { get; set; }
public Dictionary<string, string?> ProcessorInfo { get; set; }
public Dictionary<string, Dictionary<string, string>?> MemoryInfo { get; set; }
public Dictionary<string, Dictionary<string, string>?> DiskInfo { get; set; }
public Dictionary<string, Dictionary<string, string>?> VideoControllerInfo { get; set; }
public Dictionary<string, Dictionary<string, string>?> USBDevicesInfo { get; set; }
public Dictionary<string, Dictionary<string, string>?> MonitorInfo { get; set; }
```

Program.cs

Das *Program.cs* ist das Herzstück unseres Systems und wird auf jedem Rechner der Firma CountIT gespeichert. Es liest beim Start des Rechners folgende Hardwaredaten des Gerätes automatisiert ein:

1. **Allgemeine Informationen:**
   PC-Name, Benutzername, Hersteller, Modell, Seriennummer.
2. **Prozessor:**
   Name, Anzahl der Kerne, Taktgeschwindigkeit.
3. **Arbeitsspeicher (RAM):**
   Kapazität, Geschwindigkeit, Hersteller.
4. **Festplatten:**
   Modell, Speichergröße, Schnittstelle.
5. **Grafikkarten (GPU):**
   Name, VRAM, Treiberversion.
6. **USB-Geräte:**
   Name, Geräte-ID, Beschreibung.
7. **Monitore:**
   Name und Beschreibung.

Nach dem Einlesen der Hardwaredaten, werden diese als JSON-Payload an die API gesendet. Dabei wird die Windows-Management-Instrumentation (WMI) über die Bibliothek System.Management genutzt, um Hardware-Details auszulesen. Dort übernimmt das Modul **Data-Processing** die empfangenen Daten und verarbeitet sie weiter, indem es die JSON-Struktur analysiert und die Informationen in eine logisch strukturierte Form zerlegt. Dies ermöglicht eine einfache Weiterverarbeitung und Speicherung der Daten.

Folgende Abbildung zeigt die JSON-Datei der Hardwareinformationen 39

Listing 39: Code JSON Datei - Hardwareinformationen

```
{
    "GeneralInfo": {
        "PC Name": "Workstation-01",
        "Benutzername": "Philipp",
        "Hersteller": "Dell",
        "Modell": "XPS 15"
    },
    "ProcessorInfo": {
        "Name Processor": "Intel(R) Core(TM) i7-10875H",
        "Kerne": 8,
        "Threads": 16
    },
    "MemoryInfo": {
        "Slot 1": {
            "Kapazitaet": "16 GB",
            "Taktung": "3200 MHz"
        },
        "Slot 2": {
            "Kapazitaet": "16 GB",
            "Taktung": "3200 MHz"
        }
    }
}
```

3.4 Herausforderungen und Lösungen

3.4.1 Herausforderung 1: Einarbeitung in das bestehende Projekt

Das Einarbeiten in ein bestehendes Projekt war eine der größten Herausforderungen in dieser Arbeit. Die Grundstruktur des Programms war anfangs gegeben, jedoch war diese Struktur nur schwer verständlich. Zudem gab es auch viele Funktionen, die nicht verwendet worden sind. Auch gab es Funktionalitäten, die mehrmals implementiert waren. Somit mussten wir uns durch ein Gestrüpp, bestehend aus für uns logischen und unlogischen Implementierungen, schlagen.

3.4.2 Herausforderung 2: Neue Technologien

Das Projekt basiert auf .NET-Core für das Backend und nutzt Technologien wie Entity Framework für den Datenbankzugriff sowie Dependency-Injections für die Verwaltung von Services. Ohne viel Vorerfahrung in diesen Bereichen, brauchte das Einarbeiten in das bestehende Projekt natürlich länger. In weiterer Folge wurde die Frontend-Technologie, Microsoft Blazor, auch zu einer großen Hürde. Das Arbeiten mit Blazor wird von vielen mit Schmerzen verbunden. Diese Art von Schmerzen bekamen wir während der Arbeit an unserem Projekt nicht gerade selten zu spüren. Entweder hat ein neues Update alte Funktionen vernichtet, oder das Frontend machte nach langen und intensiven Stunden des Programmierens einfach nicht, was es sollte.

3.4.3 Herausforderung 3: Datenbank

Die größte Herausforderung bei unserer Datenbank war das Einfügen der Flag zum Setzen der '*Ausgewiesen*'-Flag. Dabei kam es in Verbindung mit dem Frontend zu Problemen mit dem Setzen eben dieses Flags im Frontend, da die Datenbank das Setzen nicht korrekt übernahm. Bis die Lösung nach etlichen kopfzerreissenden Stunden des Programmierens, für dieses Problem zur Verfügung stand, verging wertvolle Zeit. Diese verlorene Zeit hätten wir für andere, mindestens genauso wichtige Dinge, verwenden können.

3.4.4 Herausforderung 4: DevOps

Das DevOps führte zu weiteren Komplikationen. Diese Komplikationen beziehen sich spezifisch auf das gemeinsame Entwickeln an einem Projekt. Mehrmals kam es beim Hochladen eines fertigen Branch zu einem Merge-Konflikt mit dem Main-branch. Bittererweise wurde so manche Funktion, aufgrund unbekannter Ursachen, nicht in dem Main-Branch übernommen. Somit verschwendeten wir wertvolle Zeit beim Auseinandersetzen mit dem Lösen der Merge-Konflikte.

4 Ergebnis

4.1 Ergebnis Frontend

4.1.1 Grundstruktur

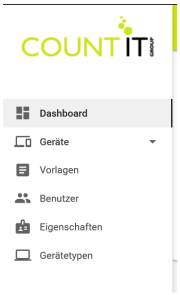


Abbildung 23: Grundstruktur GERI

Das gesamte Frontend wurde in der Programmiersprache C# in Verbindung mit dem Webframework Blazor entwickelt.
Das Frontend der Webanwendung unterteilt sich, wie im Abschnitt Frontend schon gelistet, in folgende Funktionalitäten:

- Authentifizierung
- Dashboard
- Geräte
- Vorlagen
- Benutzer
- Eigenschaften
- Gerätetypen

Diese Struktur wurde mit der Firma CountIT in einem gemeinsamen Meeting entwickelt, um eine reibungslose und fehlerfreie Bedienung der Webanwendung zu ermöglichen.

4.1.2 Authentifizierung

Dieser Teil der Webanwendung ist per se keine eigenständige Seite der Grundanwendung. Diese ist lediglich dazu da, um den Benutzer auf der Webseite einzuloggen und die Berechtigungen zu überprüfen.

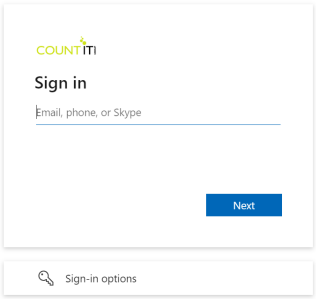


Abbildung 24: Azure-AD Loginfenster der Anwendung

Sollte der User keine Berechtigungen besitzen, um auf die Webanwendung zuzugreifen, so wird folgende Fehlermeldung ausgegeben:

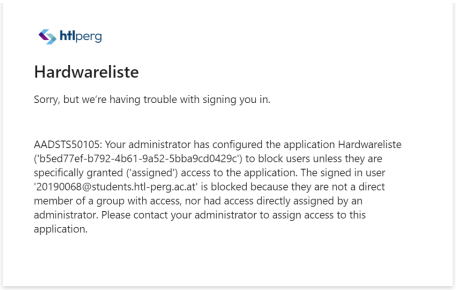


Abbildung 25: Benutzer hat keine Rechte, um auf die Webanwendung zuzugreifen

Auf der Fehlermeldung (Siehe Abbildung 25) sieht man nun, dass der eben eingeloggte Benutzer nicht über die erforderlichen Berechtigungen verfügt, um auf die Webanwendung zuzugreifen.

4.1.3 Dashboard

Abbildung 26: Dashboard der Anwendung

Diese Seite ist die erste Seite, auf die ein Benutzer nach erfolgreicher Authentifizierung landet. Das Dashboard (Siehe Abbildung 26) beginnt mit einer Titelleiste, auf der der aktuelle Seitenname angezeigt wird, sowie einer Darstellung des Benutzernamen, des aktuell eingeloggten Benutzers. In der Mitte des Dashboards kommt das Logo der internen Abteilung der CountIT, den IT Supplies zum Einsatz. Darunter werden 2 Buttons zur weiteren Navigation eingesetzt:

- **CountIT Homepage:**
  Dieser Button leitet den Benutzer bei Klick auf die Firmenwebseite weiter. Dies war ein Wunsch der Firma.
- **Devices:**
  Dieser Button leitet auf den eigentlichen Star der Anwendung weiter: die Geräteverwaltungsseite. Dies war von uns so beabsichtigt, da auf einem Dashboard nur die wichtigsten Sachen dargestellt werden sollten.

4.1.4 Geräte

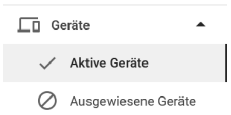


Abbildung 27: Aufteilung der Geräteseite

Grundsätzlich unterteilt sich die Hauptseite unserer Anwendung, die Geräteverwaltungsseite, in zwei Unterseiten:

- **Aktive Geräte**
- **Ausgewiesene Geräte**

Diese Seiten sind für eine genauere Differenzierung zwischen Geräten, die aktuell noch im Einsatz sind und Geräten, welche nicht mehr verwendet werden.

Aktive Geräte

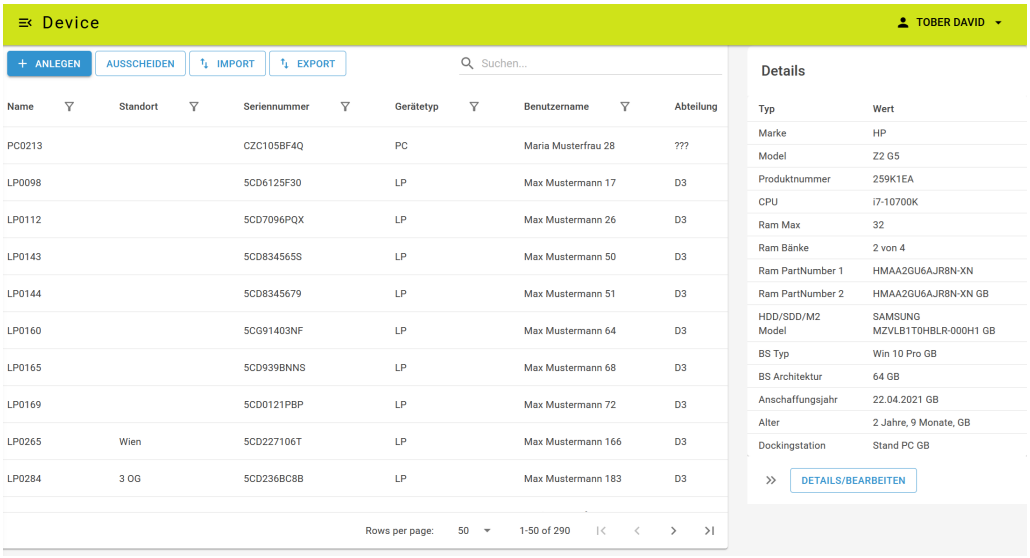


Abbildung 28: Übersicht der aktive Geräteseite

Auf dieser Seite (Siehe Abbildung 28) werden alle zurzeit verwendeten Geräte angezeigt. Sollte ein Gerät nun angeklickt werden, wird die Gesamtübersicht nach links geschoben und eine sogenannte Sidebar, also eine kurze Übersicht der wichtigsten Informationen über das angeklickte Gerät, kommt zum Vorschein.

Diese Webseite besteht zum einen aus einer Toolbar mit 4 Buttons und einer Suchleiste.

Abbildung 29: Toolbar mit den Buttons

Diese 4 Buttons der Toolbar sind:

1. Anlegen
2. Ausscheiden
3. Import
4. Export
5. Suchleiste

Anlegen

Der erste Button in der Toolbar ist der 'Anlegen' - Button. Dieser ist dafür zuständig, um ein neues Gerät manuell anlegen zu können. Wird der Button nun gewählt, so öffnet sich ein Dialogfenster (Siehe Abbildung 30) in dem man die jeweiligen Daten für ein Gerät eingeben kann.

Abbildung 30: Gerät Anlegen Dialogfenster

Das Dialogfenster besteht aus sechs unterschiedlichen Eingabefeldern:

1. Name
2. Standort
3. Seriennummer
4. Gerätetyp
5. Benutzer
6. Vorlage auswählen

Werte wie der Name, Gerätetyp und Benutzer sind zum Erstellen des Gerätes zwingend notwendig und somit durch einen '*' gekennzeichnet.

Ausscheiden

Der zweite Button in der Toolbar ist der 'Ausscheiden' - Button. Dieser wird dazu verwendet, um ausgewählte Geräte in den 'ausgewiesen' Zustand zu versetzen, d.h. das Gerät wird aktuell nicht verwendet.

Import Button

Der dritte Button ist der 'Import' - Button. Damit kann eine eigene CSV-Datei eingelesen werden und damit alle Geräte in dieser CSV-Datei automatisch hinzugefügt werden.

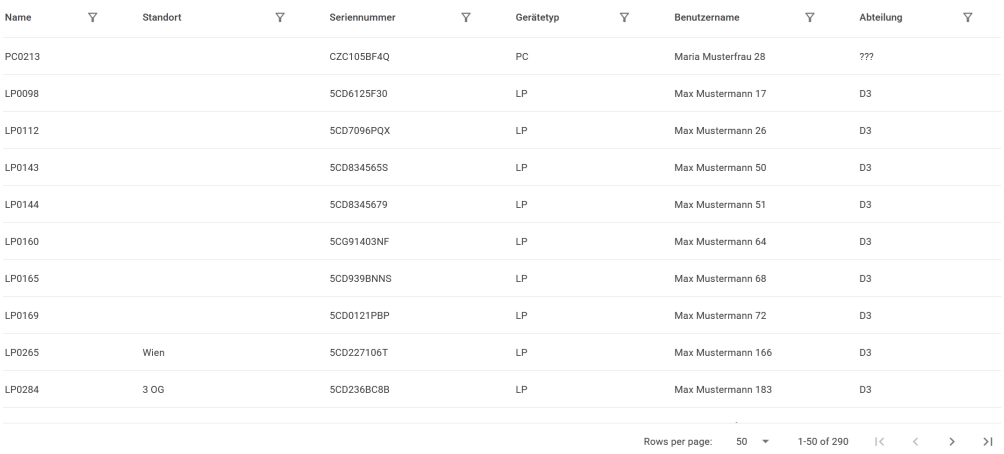
Export Button

Der letzte Button ist der 'Export' - Button. Damit ist es möglich, die gesamte Liste der Geräte in eine CSV-Datei zu exportieren und danach herunterzuladen.

Suchleiste

Auch gibt es eine Suchleiste, auf der man, ohne den aufwendigen Umweg über die Filter der Übersichtsliste gehen zu müssen, nach Namen, etc. suchen kann.

Natürlich gibt es auf dieser Seite auch eine generelle Übersichtsliste (Siehe Abbildung 31) über alle aktiven Geräte.



Name	Standort	Seriennummer	Gerätetyp	Benutzername	Abteilung
PC0213		CZC105BF4Q	PC	Maria Musterfrau 28	???
LP0098		5CD6125F30	LP	Max Mustermann 17	D3
LP0112		5CD7096PQX	LP	Max Mustermann 26	D3
LP0143		5CD834565S	LP	Max Mustermann 50	D3
LP0144		5CD8345679	LP	Max Mustermann 51	D3
LP0160		5CG91403NF	LP	Max Mustermann 64	D3
LP0165		5CD939BNNS	LP	Max Mustermann 68	D3
LP0169		5CD0121PBP	LP	Max Mustermann 72	D3
LP0265	Wien	5CD227106T	LP	Max Mustermann 166	D3
LP0284	3 OG	5CD236BC8B	LP	Max Mustermann 183	D3

Rows per page: 50 1-50 of 290

Abbildung 31: Generelle Übersicht über alle aktiven Geräte

Diese Übersicht wird in folgende Spalten unterteilt:

- **Name**
  Der Name des Gerätes, bspw.:PC0213
- **Standort**
  Der Standort des jeweiligen Gerätes, bspw.: 3.OG
- **Seriennummer**
  Die Seriennummer des Gerätes
- **Gerätetyp**
  Der Gerätetyp, bspw.: LP (Laptop)
- **Benutzername**
  Der Name des derzeitigen Benutzers des Gerätes
- **Abteilung**
  Die Abteilung, in der das jeweilige Gerät verwendet wird

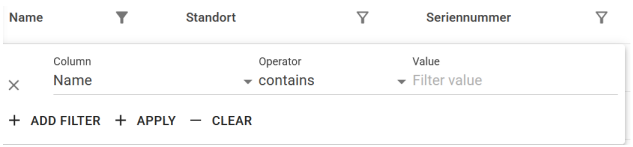


Abbildung 32: Filter Optionen für die Spalten

Sämtliche Spalten sind natürlich auch einzeln filterbar, so kann man beispielsweise die Spalte '*Name*' nach einem bestimmten Namen filtern (Siehe Abbildung 32).

Ausgewiesene Geräte

Diese Seite (Siehe Abbildung 33) wird dafür verwendet, um dem TS einen Überblick über alle nicht mehr verwendeten und somit ausgeschiedenen Geräte zu geben. Grundsätzlich besitzt die Seite über die ausgewiesenen Geräte denselben Aufbau wie die Seite über die aktiven Geräte.

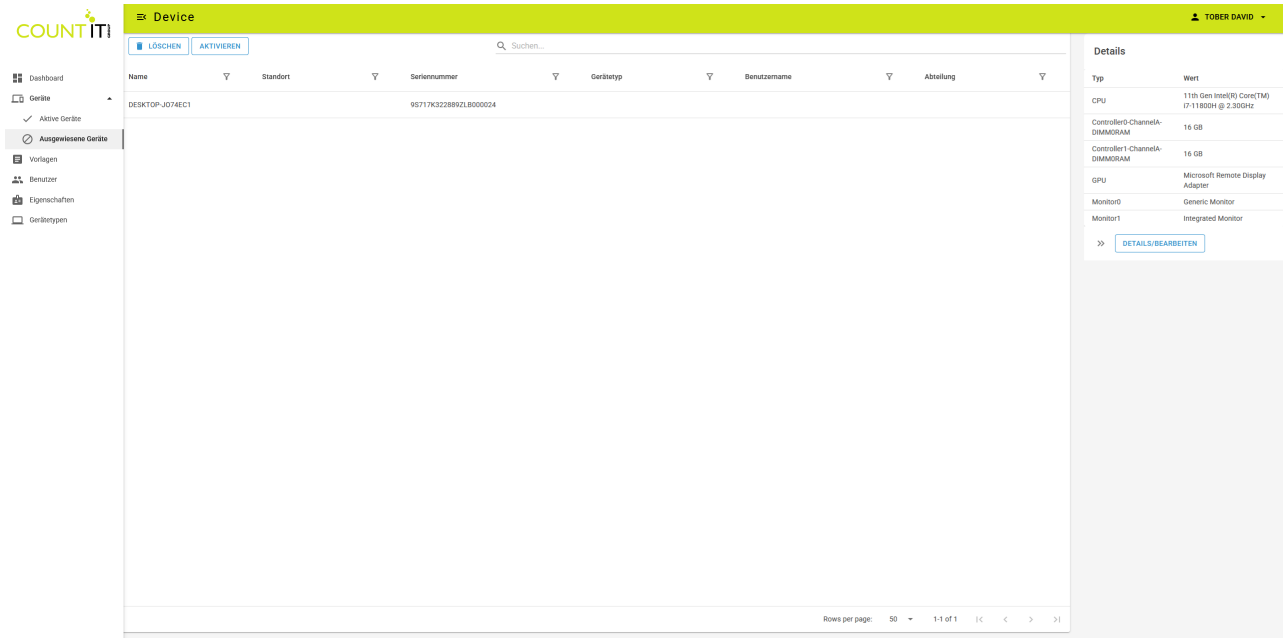


Abbildung 33: Übersicht der ausgewiesene Geräte

Wird ein Gerät angeklickt, so wird ebenso wie bei der Aktiven Geräteseite, die Gesamtübersicht nach links geschoben und eine Sidebar mit den wichtigsten Infos über das angeklickte Gerät kommt zum Vorschein.

Lediglich zwei Buttons unterscheiden die beiden Seiten voneinander.
Zum einen ein Button, mit dem die ausgewiesenen Geräte endgültig entfernt werden können (Siehe Abbildung 34).

Abbildung 34: Button zum endgültigen Entfernen der ausgewiesenen Geräte

Zum anderen ein Button (Siehe Abbildung 35), mit dem man die bereits ausgewiesenen Geräte wieder aktivieren kann, womit sie automatisch wieder auf die aktive Geräteseite verschoben werden.

Abbildung 35: Button zum Aktivieren von Geräten

4.1.5 Vorlagen

Grundidee dieser Seite (Siehe Abbildung 36) ist es, dem TS die manuelle Erstellung von Geräten zu vereinfachen. Dabei werden sog. Vorlagen ("Templates") erstellt, welche Gerätetypen zugewiesen werden können, die schon Eigenschaften enthalten.
Dadurch wird die mühevolle Arbeit des manuellen Zuweisens von Geräteeigenschaften zu Geräten vereinfacht, da nun einfach eine Vorlage erstellt werden kann und alle entsprechenden Geräteeigenschaften diesem neuen Gerät mittels dieser Vorlage automatisch zugewiesen werden.

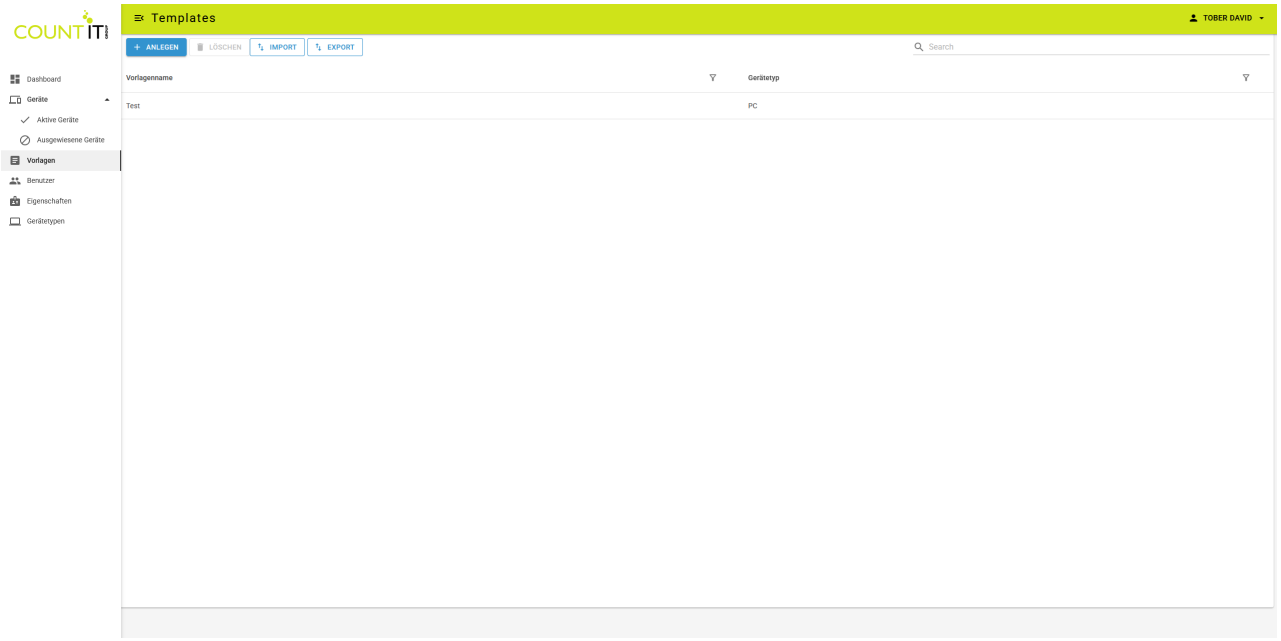


Abbildung 36: Übersicht über die erstellten Vorlagen

4.1.6 Benutzer

Auf dieser Seite (Siehe Abbildung 37) werden alle Benutzer, welche über ein Gerät verfügen, dargestellt.

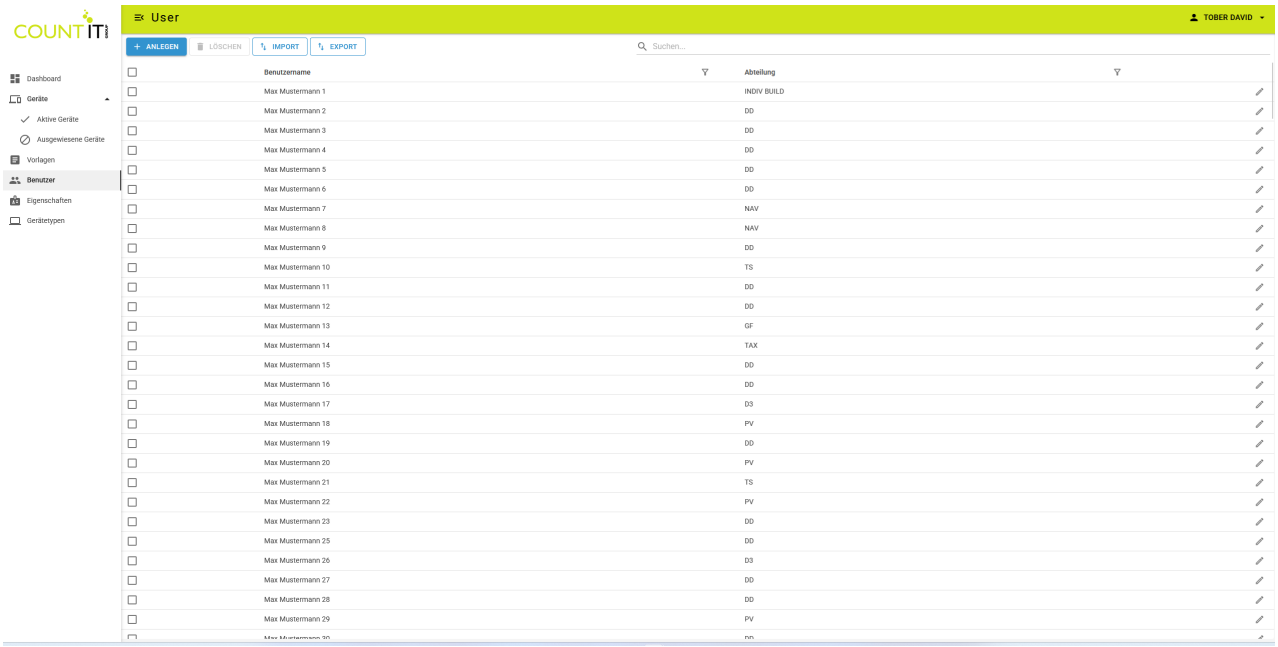


Abbildung 37: Übersicht über die Benutzer

Die Seite besteht aus folgenden Spalten:

- **Benutzername**
  Name des Benutzers
- **Abteilung**
  Abteilung in der der Benutzer tätig ist

Natürlich können sämtliche Benutzer auch beliebig manuell gelöscht und bearbeitet werden. (Siehe Abbildung 38)

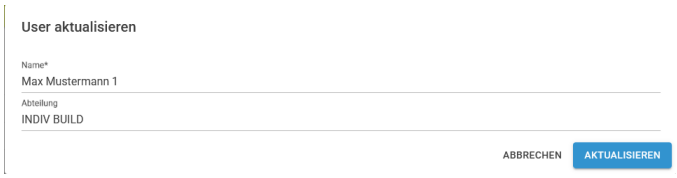


Abbildung 38: Bearbeiten eines Benutzers

4.1.7 Eigenschaften

Der Zweck dieser Seite (Siehe Abbildung 39) ist, einen Überblick über alle möglichen Eigenschaften eines Gerätes zu liefern.

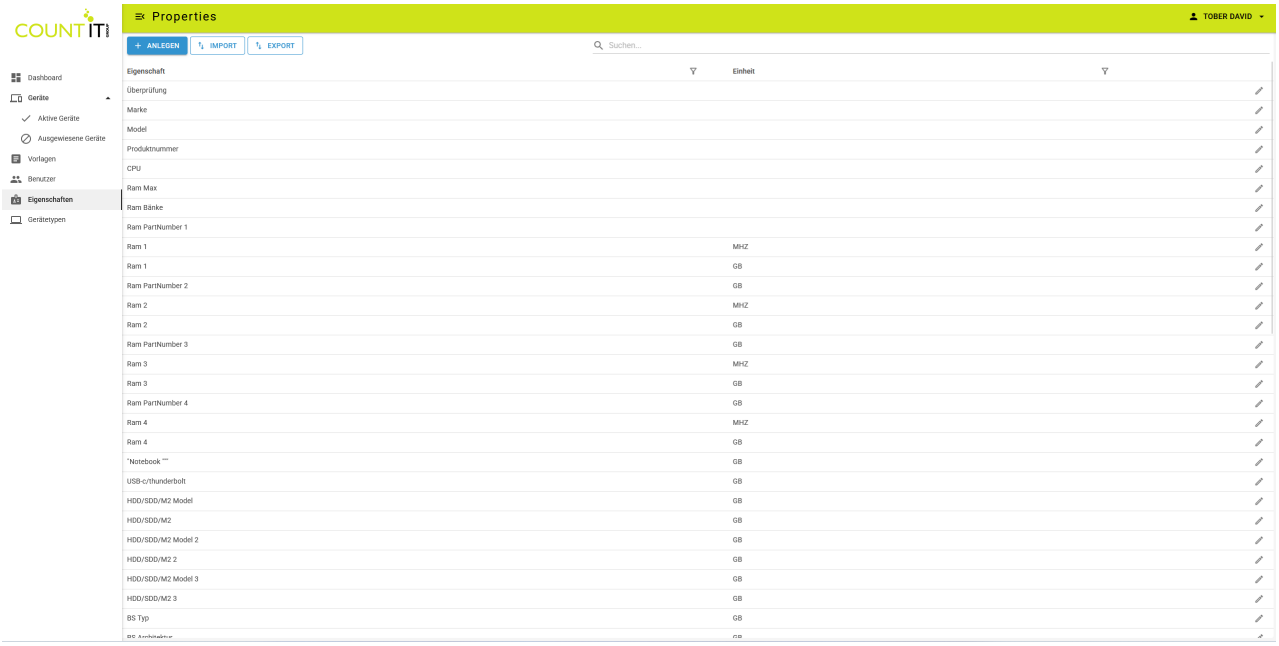


Abbildung 39: Überblick über die Eigenschaften

Die Seite teilt sich in folgende Spalten:

- **Eigenschaft**

  Name der Eigenschaft z.B.: Marke, Model, etc..

- **Einheit**

  Name der Einheit z.B.: GB, MB, MHz, etc..

Natürlich können sämtliche Eigenschaften auch beliebig bearbeitet werden.

4.1.8 Gerätetypen

Diese Seite (Siehe Abbildung 40) beinhaltet einen Überblick über sämtlich erstellte Gerätetyen. Gerätetypen sind Sammlungen von Eigenschaften. Dies bedeutet, dass ein Gerätetyp 'PC' angelegt werden kann, der über die Eigenschaften eines PCs, wie z.B.:

- **Marke**: Lenovo
- **Model**: ThinkStation

  etc...

verfügt.

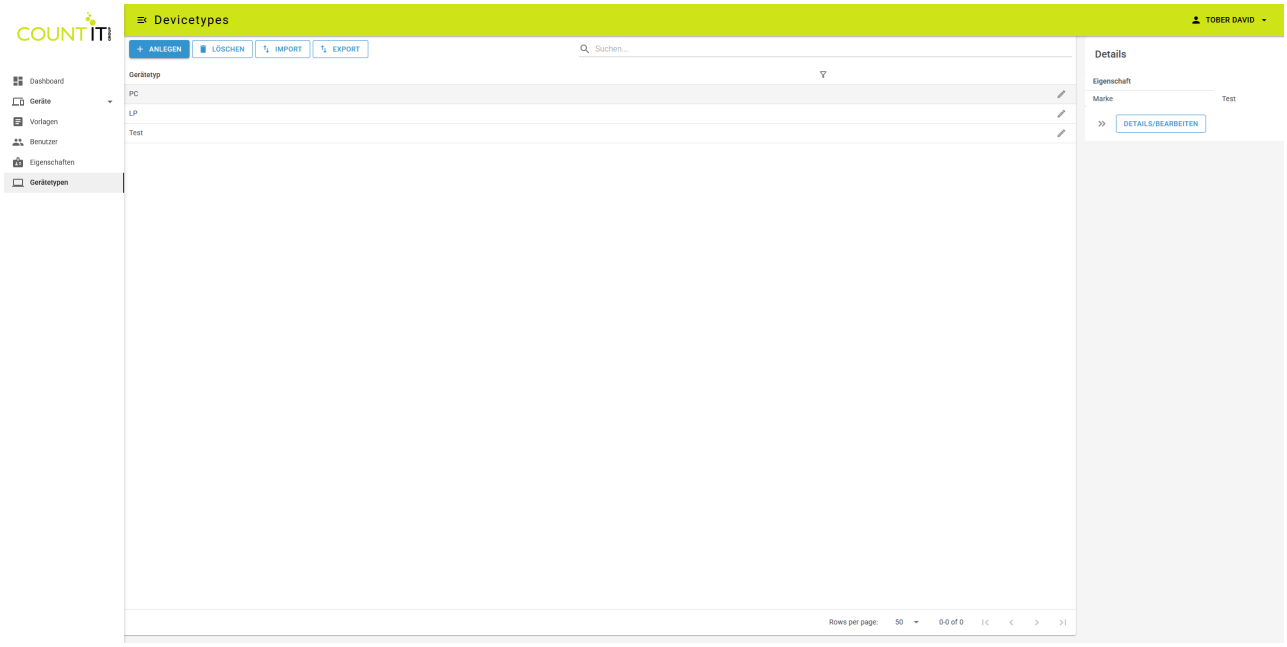


Abbildung 40: Überblick über die Eigenschaften

Klickt man einen Eintrag an, so wird, genau wie auf den Geräteseiten, die Gesamtübersicht nach links verschoben und eine Sidebar mit den entsprechenden Infos über den entsprechenden Gerätetyp kommt zum Vorschein.

4.2 Ergebnis Backend

In diesem Abschnitt wird erklärt, welche der gegebenen Anforderungen des Backends erfüllt wurden.

Das Backend bietet eine leistungsfähige API zur Verwaltung von Geräten und anderen Objekten. Über die erstellten Endpunkte können Geräte erstellt werden. Weiterhin kann man diese Geräte auch aktualisieren, löschen und nach spezifischen Geräten suchen. Für alle anderen Objekte wurden auch nötige Endpunkte erstellt.

Um die Beschreibung der Endpunkte zu visualisieren, werden nach jeder Erklärung, Screenshots pro Swagger-API-Endpunkt gezeigt.

4.2.1 Ergebnisbeschreibung für Devices

Das Backend von CIT.GERI.API liefert viele Endpunkte zum Verwalten von Geräten. Über die API können folgende Funktionen abgerufen werden:

- **erfassen**
- **aktualisieren**
- **löschen**
- **durchsuchen**

Weiters kann das System auch zwischen aktivierten und ausgewiesenen Geräten unterscheiden. Damit ist ein vollständiger Blick über die Hardware gewährleistet.

Mit dem Endpunkt **/Devices/getActiveDevices** können alle aktiven Geräte geladen werden. Im Gegensatz dazu kann man mit **/Devices/getRetiredDevices** eine Liste aller ausgewiesenen Geräte zurückliefern. Siehe Swagger-Abbildung 41

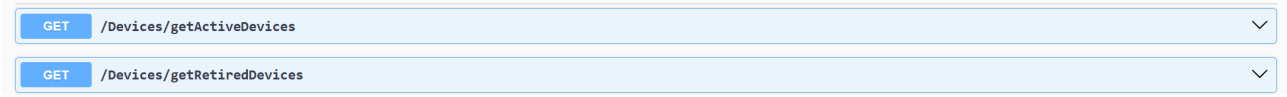


Abbildung 41: Retires/Active Devices Swagger

Falls spezifische Gerätedaten benötigt werden, bietet die API den Endpunkt **/Devices/id**. Durch das Aufrufen dieses Endpunktes, zusammen mit einer gültigen Geräte-ID, wird ein spezifisches Device (Gerät) geliefert. Zusätzlich ermöglicht **/Device/searchDevices** eine gezielte Suche nach Geräten anhand bestimmter Kriterien. Siehe Swagger-Abbildungen 42 43

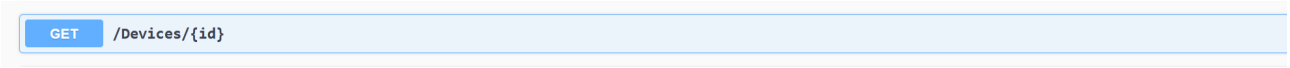


Abbildung 42: get Device by ID Swagger

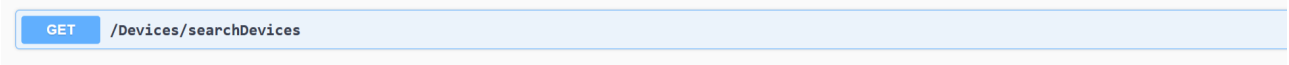


Abbildung 43: get Devices by criteria Swagger

Nicht nur das Abrufen von einem oder mehreren Geräten ist möglich, auch können Geräte erstellt werden. Bereits existierende Geräte können aktualisiert oder gelöscht werden. Der Endpunkt **/Device** nimmt über eine **POST-Anfrage** neue Gerätedaten an und speichert diese in der Datenbank ab. Siehe Swagger-Abbildungen 44 45 46

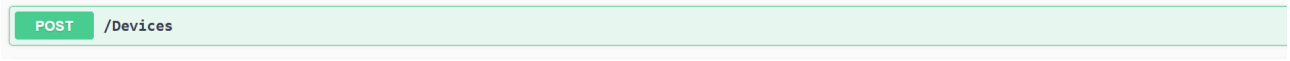


Abbildung 44: post Device Swagger

Über eine **PUT-Anfrage** an **/Devices** können Geräteinformationen geändert werden.

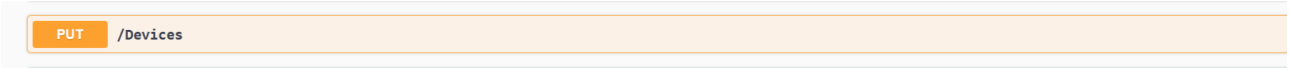


Abbildung 45: update Device Swagger

Falls ein Gerät nicht mehr benötigt wird, kann dieses per **DELETE-Anfrage** an **/Devices/id** gelöscht werden.

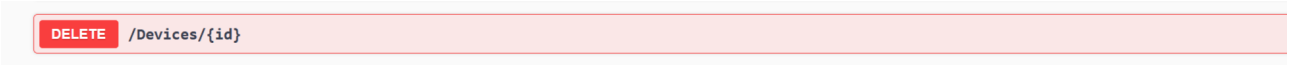


Abbildung 46: delete Device Swagger

Neben den Standardfunktionalitäten wie **POST, PUT, DELETE, ...**, kann man auch den Gerätestatus verwalten. Über den Endpunkt **/Devices/updateRetired** können Geräte als nicht aktiv gekennzeichnet werden. Dadurch bleiben die Geräte in der Datenbank, werden im Frontend jedoch nicht mehr als aktive Geräte dargestellt. Dies ermöglicht eine bessere Nachverfolgung und sorgt dafür, dass veraltete Hardwaredaten nicht versehentlich gelöscht werden. Siehe Swagger-Abbildung 47

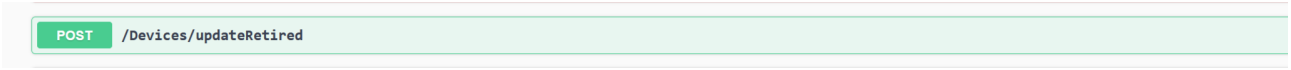


Abbildung 47: update retired Devices Swagger

Das Backend von CIT.GERI.API stellt eine leistungsfähige und flexible Lösung zur Verwaltung von Geräten bereit. Neue Geräte können einfach erfasst werden und genauso ist das Verwalten von bestehenden Geräten möglich. Durch die klare Strukturierung der Endpunkte ist die Verwaltung der Geräte einfach und übersichtlich.

4.2.2 Ergebnisbeschreibung für DeviceContaineR und DeviceData

Das Backend von CIT.GERI.API stellt eine API zur Verwaltung von Gerätecontainern zur Verfügung. Diese Container helfen dabei, aktive und archivierte Geräte zu organisieren und übersichtlich zu halten.

Mit der **GET-Anfrage** an **/DeviceContainer/getActiveDevices** können alle derzeit aktiven Geräte abgerufen werden. Falls nur die archivierten Geräte benötigt werden, ermöglicht der Endpunkt **/DeviceContainer/getRetiredDevices** eine entsprechende Abfrage. Diese Unterscheidung sorgt für eine bessere Übersicht und erleichtert die Verwaltung von Geräten, die nicht mehr aktiv genutzt werden. Siehe Swagger-Abbildung 48

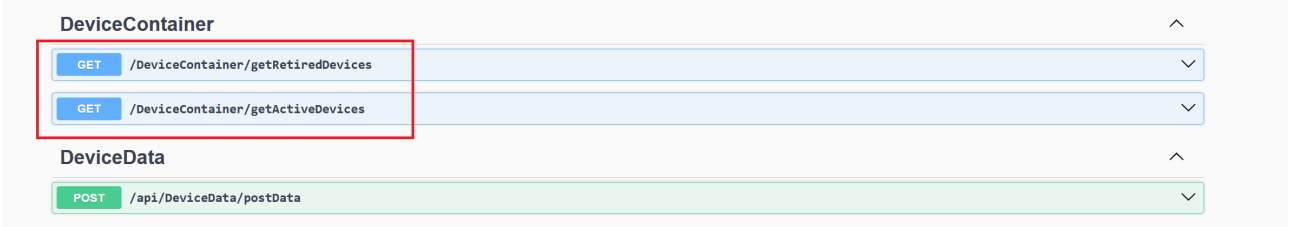


Abbildung 48: Retires Active Devices Swagger

Darüber hinaus bietet das Backend die Möglichkeit, neue Gerätedaten zu übermitteln und zu speichern. Über eine **POST-Anfrage** an **/api/DeviceData/postData** können gesammelte Gerätedaten in das System hochgeladen werden. Dies ist besonders nützlich für die automatische

Erfassung von Hardware-Informationen, die anschließend verarbeitet und in der Datenbank gespeichert werden. Siehe Swagger-Abbildung 49

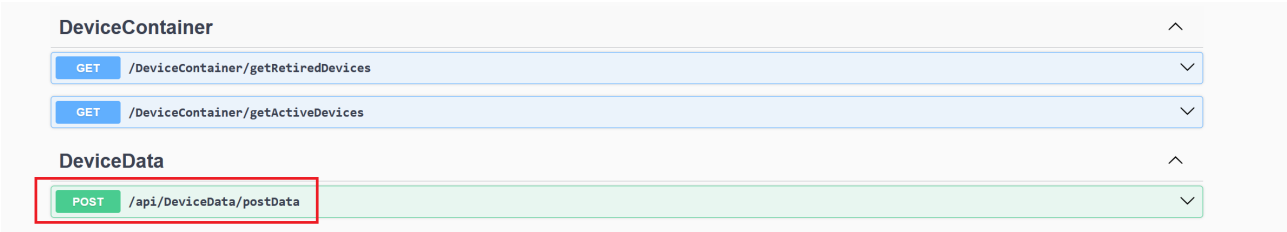


Abbildung 49: DeviceContainer/DeviceDate Swagger

Durch diese API-Funktionen können aktive und archivierte Geräte effizient verwaltet und neue Gerätedaten in das System eingefügt werden. Dadurch bleibt die Datenbank aktuell und alle relevanten Informationen stehen jederzeit zur Verfügung.

4.2.3 Ergebnisbeschreibung für Properties

Das Backend von **CIT.GERI.API** stellt eine API zur Verwaltung von Geräteeigenschaften (Properties) zur Verfügung. Über diese Schnittstelle können Eigenschaften aktualisiert, gelöscht und zu spezifischen Geräten zugeordnet werden. Durch diese Funktionalitäten können Gerätedetails zentral erfasst und verwaltet werden.

Mithilfe von verschiedenen Endpunkten können Geräteeigenschaften abgerufen werden. Mit der **GET-Anfrage** an **/Property** werden alle Eigenschaften aufgelistet. Falls nur die Eigenschaften für ein spezifisches Gerät benötigt werden, dann können diese Eigenschaften per **GET-Anfrage** an **/Property/by-device/deviceid** angefordert werden. Siehe Swagger-Abbildungen 50 51

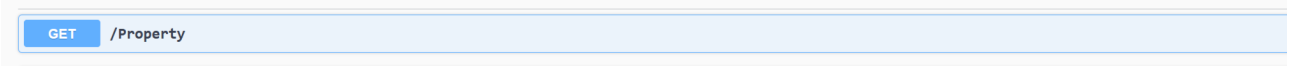


Abbildung 50: get Properties Swagger

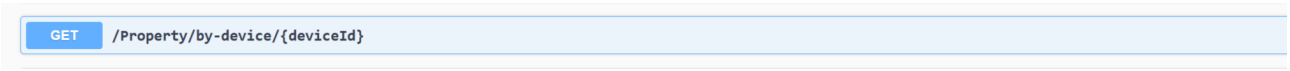


Abbildung 51: get Properties by DeviceID Swagger

Neben den Abrufen von Eigenschaften können über die API auch neue Eigenschaften angelegt werden. Weiters können auch bestehende Geräteeigenschaften aktualisiert oder auch gelöscht werden. Der Endpunkt **/Property** nimmt **POST-Anfragen** entgegen, um somit

neue Eigenschaften zu erstellen und in der Datenbank zu speichern. Falls sich Werte einer Eigenschaft ändern, kann dies über eine **PUT-Anfrage** an **/Property** oder **/Property/id** aktualisiert werden. Wenn eine Geräteeigenschaft nicht mehr benötigt wird, kann diese über eine **DELETE-Anfrage** an **/Property/id** beseitigt werden. Siehe Swagger-Abbildungen 52 53 54 55

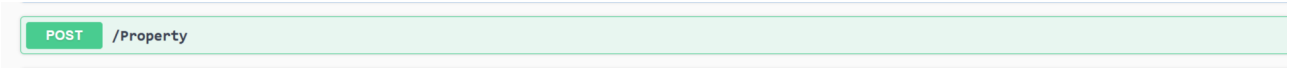


Abbildung 52: post Property Swagger

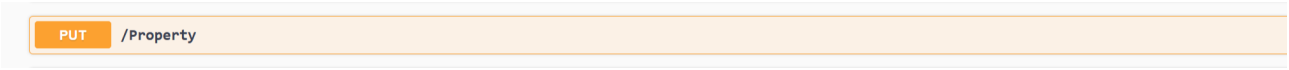


Abbildung 53: put Property Swagger

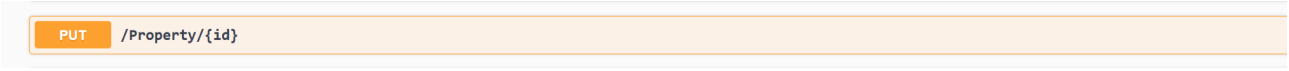


Abbildung 54: put Property by id Swagger

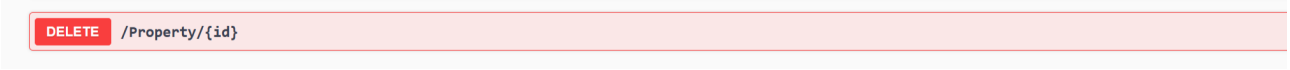


Abbildung 55: delete Property by id Swagger

Durch diese API-Funktionen wird eine flexible und effiziente Verwaltung von Hardware-Eigenschaften ermöglicht. Alle Änderungen werden zentral gespeichert und können jederzeit über die API abgerufen oder bearbeitet werden. Dies sorgt für eine vollständige und aktuelle Dokumentation aller Geräteeigenschaften.

4.2.4 Ergebnisbeschreibung für Property_Device

Das Backend ermöglicht die Verwaltung der Zuweisung von Eigenschaften zu Geräten. Dadurch wird sichergestellt, dass jedes Gerät die richtigen und aktuellen Eigenschaften hat.

Über eine **POST-Anfrage** an **/Property_Device** können neue Eigenschaften mit einem Gerät verknüpft werden. Falls eine Eigenschaft nicht mehr benötigt wird oder falsch zugewiesen wurde, kann sie über eine **DELETE-Anfrage** an **/Property_Device/deviceID/propertyID** entfernt werden.

Diese API ermöglicht eine effiziente Verwaltung von Geräteeigenschaften. Neue Eigenschaften können hinzugefügt und bestehende bei Bedarf gelöscht werden. Dies sorgt dafür, dass die

gespeicherten Gerätedaten stets aktuell sind und nur relevante Informationen gespeichert werden. Siehe Swagger-Abbildung 56

Abbildung 56: post/delete PropertyDevice Swagger

4.2.5 Ergebnisbeschreibung für Users

Neben dem Verwalten der Geräte und deren Eigenschaften, ist auch das Verwalten der Benutzer (User) wichtig. Benutzer können über die API:

- **erfasst**
- **aktualisiert**
- **gelöscht**

und mit anderen Geräten **zusammengeführt** werden.

Mit einer **GET-Anfrage** an **/User** können alle registrierten Benutzer abgerufen werden. Falls nur die Benutzer eines bestimmten Geräts benötigt werden, kann dies über den Endpunkt **/User/by-device/deviceId** erfolgen. Dadurch lassen sich Benutzer gezielt nach dem Gerät filtern, mit dem sie verknüpft sind. Siehe Swagger-Abbildungen 57 58

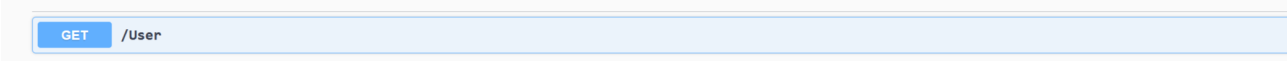


Abbildung 57: get User Swagger

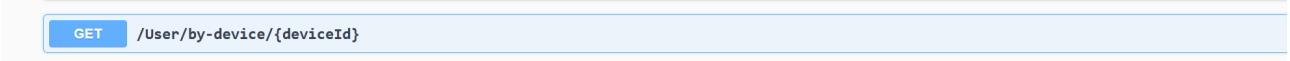


Abbildung 58: get User by DeviceID Swagger

Benutzer können nicht nur abgerufen, sondern auch über die API erstellt werden. Über eine **POST-Anfrage** an **/User** kann ein neuer Benutzer erstellt und in der Datenbank gespeichert werden. Falls bereits existierende Benutzer aktualisiert werden müssen, kann dies über die **PUT-Anfrage** an **/User** ausgeführt werden. Natürlich muss es möglich sein, einen Benutzer aus dem System zu löschen. Das kann über die **DELETE-Anfrage** an **/User/id** durchgeführt werden.

Folgende Grafik visualisiert die im vorigen Absatz genannten Funktionalitäten 59:

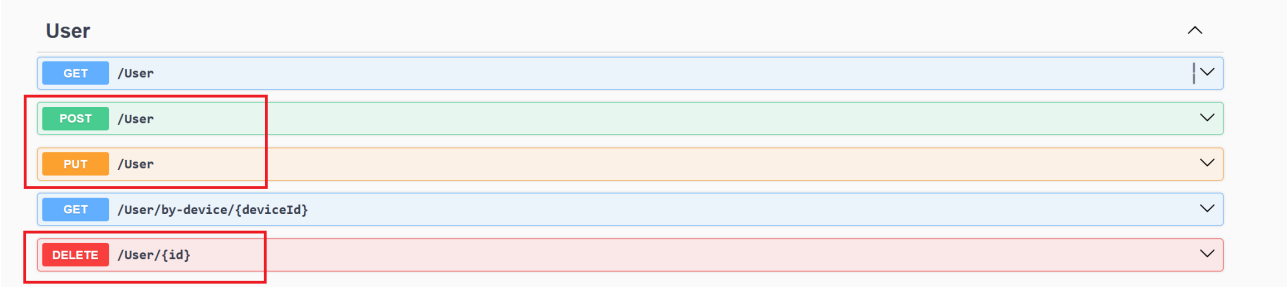


Abbildung 59: post/put/delete User Swagger

4.2.6 Ergebnisbeschreibung für DeviceTypes

Das Backend von CIT.GERI.API stellt eine API zur Verwaltung von Gerätetypen bereit. Über diese Schnittstelle können Gerätetypen erstellt, geändert, gelöscht und bestimmten Geräten zugewiesen werden. Damit wird sichergestellt, dass alle Geräte im System einer klar definierten Kategorie zugeordnet sind, was die Strukturierung und Verwaltung der Hardware vereinfacht.

Mit einer **GET**-Anfrage an **/DeviceType** können in der Datenbank gespeicherte Gerätetypen abgerufen werden. Weiters kann auch nur ein spezifischer Gerätetyp abgefragt werden. Falls nur ein spezifischer Gerätetyp benötigt wird, kann dies über den Endpunkt **/DeviceType/id** erfolgen, indem die zugehörige ID übergeben wird. Zusätzlich besteht die Möglichkeit, mit **/DeviceType/by-device/deviceId** alle Gerätetypen eines bestimmten Geräts abzurufen. Siehe Swagger-Abbildungen 60 61 62

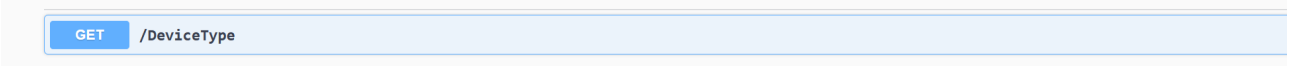


Abbildung 60: get DeviceType Swagger

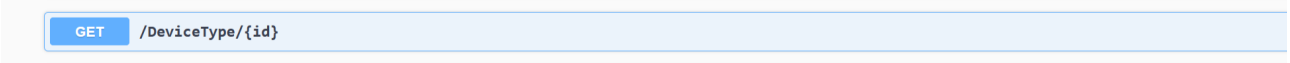


Abbildung 61: get DeviceType by id Swagger

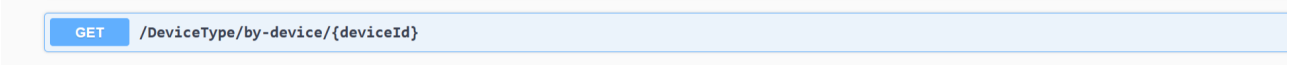


Abbildung 62: get DeviceType by Device id Swagger

Neben der reinen Abfrage von Daten ermöglicht die API auch das Erstellen, Bearbeiten und Löschen von Gerätetypen. Über die **POST-Anfrage** an **/DeviceType** können neue Typen erstellt werden.

Folgende Funktionalitäten sind noch für die Gerätetypen verfügbar:

- **PUT-Anfrage** an **/DeviceType**
- **PUT-Anfrage** an **/DeviceType/id**
- **DELETE-Anfrage** an **/DeviceType/id**

Siehe Swagger-Abbildung 63

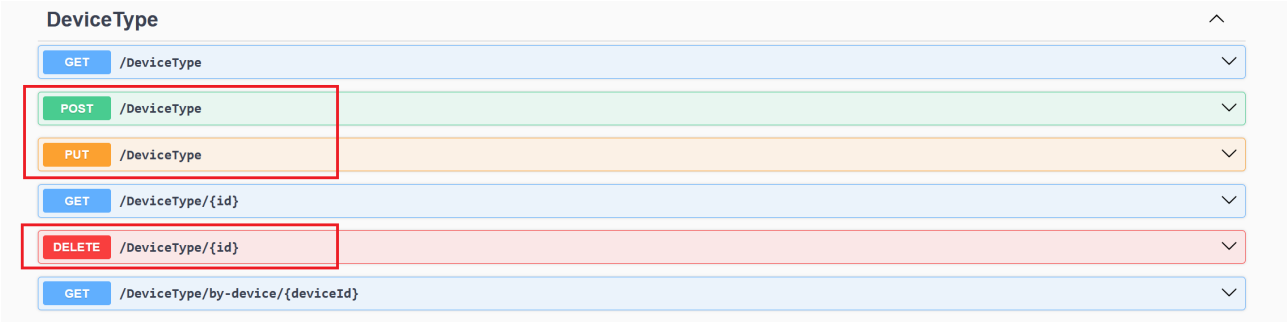


Abbildung 63: post/put/delete DeviceType Swagger

Die zuvor genannten Funktionalitäten ermöglichen eine klare und strukturierte Verwaltung von Gerätetypen. Jede Hardware kann einer Kategorie zugeordnet werden. Dadurch entsteht eine übersichtliche Datenhaltung.

4.2.7 Ergebnisbeschreibung für DeviceTypeProperty

Das Backend bietet eine API, mit der sich Eigenschaften für verschiedene Gerätetypen verwalten lassen. Mit dieser API können Gerätetypen bestimmte Eigenschaften zugewiesen, bestehende Werte geändert oder Verknüpfungen entfernt werden. So lassen sich Gerätetypen klar definieren und einfacher verwalten.

Mit einer **GET-Anfrage** an **/DeviceTypeProperty/deviceId** können alle Eigenschaften eines bestimmten Gerätetyps abgerufen werden. Falls eine neue Eigenschaft mit einem Gerätetyp verknüpft werden soll, kann dies über eine **POST-Anfrage** an **/DeviceTypeProperty/deviceId/propertyId/value** erfolgen. Falls eine Eigenschaft nicht mehr benötigt wird, kann sie über eine **DELETE-Anfrage** an **/DeviceTypeProperty/deviceId/propertyId** entfernt werden. Siehe Swagger-Abbildung 64

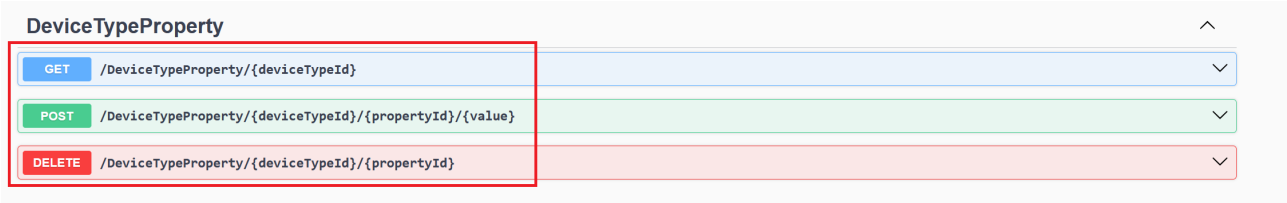


Abbildung 64: get/post/delete DeviceTypeProperty Swagger

Diese API stellt eine effiziente Verwaltung von Gerätetypen und ihren Eigenschaften zur Verfügung. Die zuvor beschriebenen Funktionalitäten sorgen für eine flexible und anpassbare Verwaltung von Gerätetypen im System.

4.2.8 Ergebnisbeschreibung für DeviceTemplates

Das Backend von CIT.GERI.API bietet eine API zur Verwaltung von Gerätetemplates. Diese Templates ermöglichen eine einheitliche Strukturierung und Verwaltung von Gerätedaten, indem vordefinierte Eigenschaften und Konfigurationen für bestimmte Gerätetypen festgelegt werden.

Die API stellt verschiedene Endpunkte zum Abrufen von Gerätetemplates zur Verfügung. Über eine **GET**-Anfrage an **/DeviceTemplate** können alle gespeicherten Templates abgerufen werden. Falls nur ein bestimmtes Template benötigt wird, kann es über **/DeviceTemplate/id** anhand seiner ID gezielt abgerufen werden. Zusätzlich ermöglicht **/DeviceTemplate/Container** eine Abfrage aller Gerätetemplates in einer Container-Ansicht, während **/DeviceTemplate/Container/id** ein spezifisches Template innerhalb dieses Containers ausgibt. Siehe Swagger-Abbildung 65

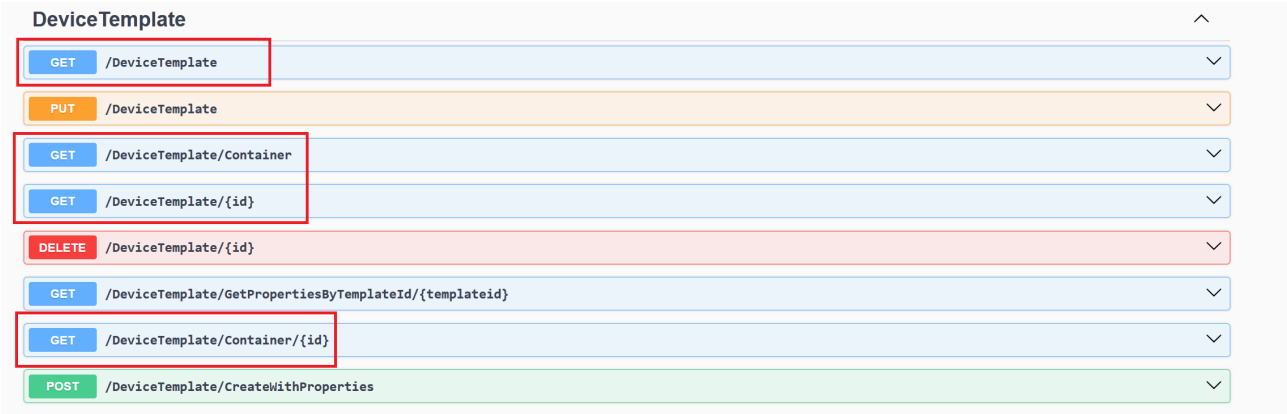


Abbildung 65: get DeviceTemplate Swagger

Neben der Abfrage von Templates bietet die API auch Funktionen zur Erstellung und Verwaltung von Gerätekonfigurationen. Über eine **POST-Anfrage** an **/DeviceTemplate/CreateWith-**

Properties können neue Templates erstellt und mit bestimmten Eigenschaften verknüpft werden. Falls bestehende Templates geändert werden müssen, kann dies über eine **PUT-Anfrage** an **/DeviceTemplate** erfolgen. Falls ein Template nicht mehr benötigt wird, kann es über eine **DELETE-Anfrage** an **/DeviceTemplate/id** entfernt werden. Siehe Swagger-Abbildung 66

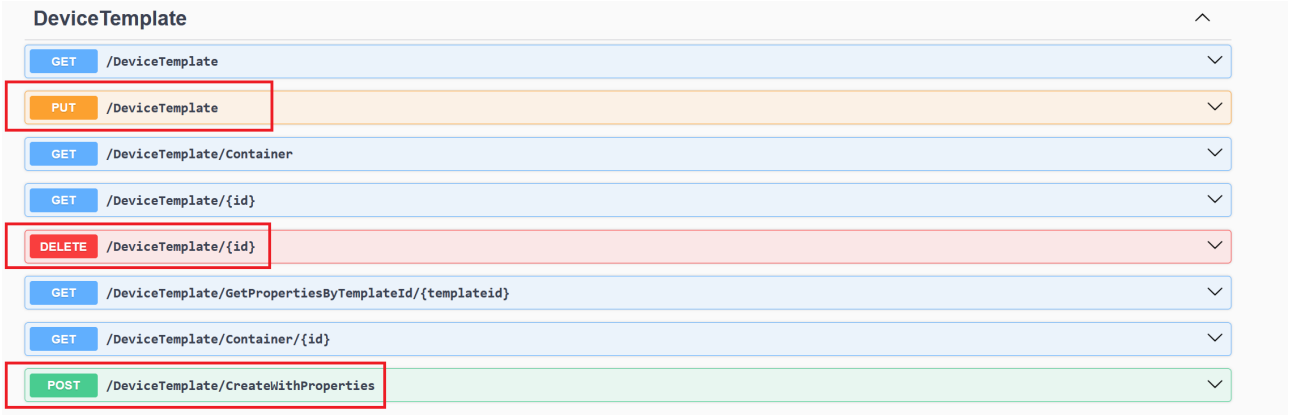


Abbildung 66: post/put/delete DeviceTemplate Swagger

Zusätzlich bietet die API eine Funktion zur Abfrage von Template-Eigenschaften. Mit dem Endpunkt **/DeviceTemplate/GetPropertiesByTemplateId/templateId** können die spezifischen Eigenschaften eines bestimmten Templates ausgelesen werden. Siehe Swagger-Abbildung 67

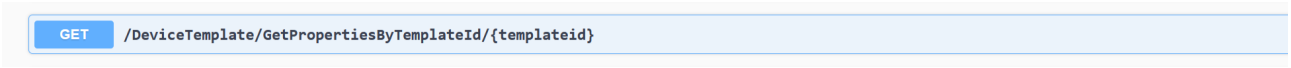


Abbildung 67: get DeviceTemplate Properties by Templade id Swagger

Durch diese API-Funktionen ermöglicht das Backend eine effiziente Verwaltung von Gerätetemplates. Neue Konfigurationen können einfach erstellt, bestehende angepasst und nicht mehr benötigte Vorlagen entfernt werden. Dies sorgt für eine klare und einheitliche Struktur bei der Verwaltung von Geräten und deren Eigenschaften.

4.2.9 Ergebnisbeschreibung für DeviceTemplates_Property

Das Backend enthält eine API zur Verwaltung der Verknüpfung zwischen Gerätetemplates und deren Eigenschaften. Folgende Funktionalitäten ermöglichen das Erstellen, sowie das Löschen und Anpassen von Eigenschaften zu einem bestimmten Template. Dadurch wird eine konsistente Verwaltung von Gerätetemplates und deren Konfigurationen gewährleistet.

Über eine GET-Anfrage an **/DeviceTemplates_Property** können alle bestehenden Verknüpfungen zwischen Templates und Eigenschaften abgerufen werden. Falls eine neue Eigenschaft mit einem Template verknüpft werden soll, kann dies über eine POST-Anfrage an **/DeviceTemplates_Property** erfolgen. Bereits bestehende Zuordnungen können über eine **PUT-Anfrage** an **/DeviceTemplates_Property/id** aktualisiert werden. Falls eine bestimmte Eigenschaft aus einem Template entfernt werden soll, kann dies über eine **DELETE-Anfrage** an **/DeviceTemplates_Property/DeviceTemplateId/PropertyId** erfolgen. Siehe Swagger-Abbildung 68

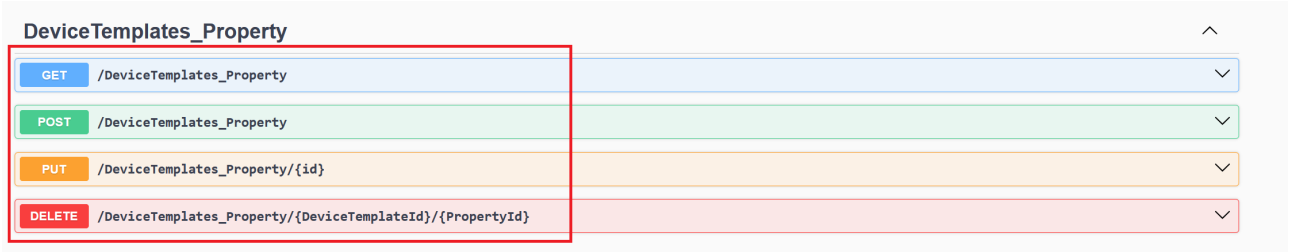


Abbildung 68: get/post/put/delete DeviceTemplate_Property Swagger

Durch diese API wird sichergestellt, dass Gerätetemplates flexibel konfigurierbar bleiben. Neue Eigenschaften können hinzugefügt werden. Weiters können bestehende Eigenschaften geändert werden, und nicht mehr benötigte gelöscht werden. Dies erleichtert die Verwaltung von Gerätekonfigurationen und stellt sicher, dass die jeweiligen Templates stets mit den richtigen Geräteeigenschaften verknüpft sind.

4.2.10 Zusammenfassende Beschreibung des Backends

Das Backend von CIT.GERI.API bildet das Rückgrat der gesamten Anwendung. Es ist für die vollständige serverseitige Verwaltung der Gerätedaten verantwortlich, die für das Endprodukt benötigt werden. Dabei agiert es nicht nur als reine Datenschnittstelle, sondern als Steuerungssystem, das zahlreiche Prozesse automatisiert und aufbereitet. Die erfassten Informationen betreffen allgemeine Gerätedetails wie Name, Standort und Seriennummer sowie technische Komponenten wie Prozessor, Arbeitsspeicher, Grafikkarte, Festplatten, USB-Geräte und Monitore.

Besonders wichtig für das Produkt ist die automatische Benutzerzuweisung. Hierfür wird eine intelligente Logik eingesetzt, bei der pro Gerät eine Warteschlange geführt wird. Diese Warteschlange (Queue) verfolgt, welcher Benutzer das Gerät verwendet, und weist bei entsprechender Häufung automatisch den Nutzer, der das Gerät am öftesten verwendet, dem Gerät zu. Die Zuordnung passiert ohne Eingreifen eines/einer Mitarbeiters/Mitarbeiterin und stellt sicher,

dass die Geräteverwaltung immer den tatsächlichen Zustand widerspiegelt. Dies erspart dem Unternehmen einiges an Arbeit.

Ein weiterer Bestandteil des Backends ist die Unterscheidung zwischen aktiven und nicht aktiven Geräten. Geräte, die nicht mehr im Einsatz sind, können als „*retired*“ markiert werden. Diese Funktion erlaubt es, ausgemusterte Geräte weiterhin nachzuvollziehen, ohne sie aus dem System löschen zu müssen. Damit wird eine langfristige Nachverfolgung der IT-Infrastruktur möglich gemacht.

Das Backend unterstützt auch die Verwaltung von Gerätetypen und Templates. Damit können Hardwarekonfigurationen erfasst werden, was besonders hilfreich ist, wenn viele Geräte nach ähnlichen Spezifikationen strukturiert werden müssen. Auch dieser Aspekt trägt zur Übersichtlichkeit der Datenbank bei.

Alle Funktionen des Backends sind über eine REST-API zugänglich. Das bedeutet, dass andere Systeme – wie z.B. eine grafische Verwaltungsoberfläche – direkt mit dem Backend kommunizieren und Informationen abrufen, bearbeiten oder hinzufügen können. Die Daten werden in einem einheitlichen Format übermittelt, sodass eine klare Trennung zwischen Backend-Logik und Frontend-Darstellung gewährleistet ist.

Für das Zusammenspiel mit CIT.GERI.AGNES ist das Backend ebenfalls vorbereitet. Sobald AGNES beim Start eines Rechners Hardwaredaten sammelt, sendet es diese strukturiert an das Backend. Dort werden sie automatisch verarbeitet, mit bestehenden Datensätzen verglichen und aktualisiert oder neu eingetragen. Das alles passiert abhängig davon, ob das Gerät bereits bekannt ist. Diese direkte Integration von AGNES sorgt dafür, dass die Erfassung und Benutzerzuweisung im Hintergrund läuft, ohne dass ein Eingreifen durch IT-Mitarbeiter erforderlich ist.

Damit leistet das Backend einen wichtigen Beitrag zur Zielsetzung des Projekts, nämlich eine automatisierte, verlässliche und aktuelle Verwaltung aller IT-Geräte und ihrer zugehörigen Nutzer. Es übernimmt die eigentliche Verarbeitung, stellt alle relevanten Daten bereit und sorgt durch seine technische Architektur für eine effiziente und wartungsfreundliche Umsetzung.

5 Resümee

Diese Diplomarbeit ist unsere erste Erfahrung im Verfassen technischer und wirtschaftlicher Arbeiten.

Der Grundstein für diese Diplomarbeit wurde bereits am Tag der offenen Tür der HTL Perg gelegt. An diesem Tag bewarben wir uns direkt am Stand der Firma CountIT für eine Diplomarbeit. Kurz darauf wurde die Zusammenarbeit in einem persönlichen Meeting im Unternehmen offiziell besiegelt. Nach einigen klärenden Absprachen zum Umfang dieser Diplomarbeit, konnte die Reise bereits zu Beginn unseres Praktikums im Sommer losgehen. Somit konnte im Praktikum ein Großteil der Anforderungen unter bezahlten Umständen erledigt werden.

Während des Praktikums konnten viele hilfreiche Erfahrungen und Eindrücke über das spätere Arbeitsleben gesammelt werden. In weiterer Folge war die Zusammenarbeit in einem Projektteam für uns eine neuartige Erfahrung, die uns gezeigt hat, wie wichtig die interne Kommunikation im Team ist. Besonders positiv war die Zusammenarbeit mit unserem Projektleiter, Leonhartsberger Elias, im Unternehmen. Dieser half uns bei großen sowie kleinen Problemen. Die größte Hürde war das Weiterführen von einem bereits bestehenden Projekt.

Da der Backend-Teil die meiste Zeit in Anspruch nahm, wurde dieser sowie einzelne Aufgaben im Frontend erst nach dem Praktikum fertiggestellt. Trotz allem waren wir auch nach Praktikumsende noch mit unserem Projektleiter in Kontakt. Durch seine Hilfe wurden viele Fragen und aufkommende Probleme besprochen und gelöst.

Nach Vollendung des Backends, wurde sich auf das Verfassen der Diplomarbeit konzentriert. Nach stundenlangem und kopfzerreißendem Schreiben dieser Diplomarbeit, endete alles beim Resümee.

Glossar

AGNES **A**utomatische **G**eräte A**n**passung und **E**rstellung beim **S**tart

API Application Programming Interface

Azure AD Azure Active Directory

CPU **C**entral **P**rocessing **U**nit

CSS Cascading Style Sheets

Dictionary Key-Value Store in C#

GERI **Ge**räte**i**ndex

GPU **G**raphics **P**rocessing **U**nit

HTL Höhere Technische Lehranstalt

HTML Hyper Text Markup Language

HTTP **H**yper**t**ext **T**ransfer **P**rotocol

ITSM **IT** **S**ervice **M**anagement Software

JSON **J**ava**S**cript **O**bject **N**otation

RAM **R**andom **A**ccess **M**emory

REST **Re**presentational **S**tate **T**ransfer - Protokoll

TS **T**echnical **S**ervice

UI **U**ser **I**nterface

WSL **W**indows **S**ubsystem for **L**inux

Literaturverzeichnis

[1] Wikipedia, „C-Sharp." Online verfügbar: https://de.wikipedia.org/wiki/C-Sharp

[2] IT-Schulungen, „Was ist Microsoft Blazor und Blazor WebAssembly?" 2024. Online verfügbar: https://www.it-schulungen.com/wir-ueber-uns/wissensblog/was-ist-microsoft-blazor-und-blazor-webassembly.html

[3] ——, „Was ist Microsoft Blazor und Blazor Webassembly." Online verfügbar: https://www.it-schulungen.com/wir-ueber-uns/wissensblog/was-ist-microsoft-blazor-und-blazor-webassembly.html

[4] IBM, „Was ist eine REST-API?" Online verfügbar: https://www.ibm.com/de-de/topics/rest-apis

[5] Git, „Git - Was ist Git?" Online verfügbar: https://git-scm.com/book/de/v2/Erste-Schritte-Was-ist-Git%3F

[6] Microsoft, „Was ist Azure DevOps?" Online verfügbar: https://learn.microsoft.com/de-de/azure/devops/user-guide/what-is-azure-devops?view=azure-devops

[7] Wikipedia, „Visual Studio." Online verfügbar: https://de.wikipedia.org/wiki/Visual_Studio

[8] Microsoft, „Visual Studio IDE." Online verfügbar: https://visualstudio.microsoft.com/de/vs/

[9] ——, „SQL Server Management Studio." Online verfügbar: https://learn.microsoft.com/de-de/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16

[10] zeroshope, „Was ist .NET Core?" 2020. Online verfügbar: https://www.dev-insider.de/was-ist-net-core-a-f2280609b28ad1e3f1e9a3deca966ab7/

[11] dotnettutorials, „ADO.NET Tutorial for Beginners and Professionals." Online verfügbar: https://dotnettutorials.net/course/ado-net-tutorial-for-beginners-and-professionals/

[12] MudBlazor, „What ist MudBlazor." Online verfügbar: https://www.mudblazor.com/mud/introduction#first-step:-mudblazor-as-a-component-library-

[13] Swagger, „What ist Swagger." Online verfügbar: https://swagger.io/docs/specification/v2_0/what-is-swagger/

[14] SonarCloud, „SonarQube Cloud Documentation." Online verfügbar: https://docs.sonarsource.com/sonarqube-cloud/

[15] Clockify, „Features - Clockify." Online verfügbar: https://clockify.me/features/

[16] Microsoft, „Microsoft Teams." Online verfügbar: https://www.microsoft.com/de-at/microsoft-teams/group-chat-software/

[17] OverLeaf, „OverLeaf." Online verfügbar: https://de.overleaf.com/about/features-overview

[18] Microsoft, „System.ComponentModel.DataAnnotations Namespace." Online verfügbar: https://learn.microsoft.com/de-de/dotnet/api/system.componentmodel.dataannotations?view=net-8.0

[19] J. Skinner, „FluentValidation." Online verfügbar: https://docs.fluentvalidation.net/en/latest/

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang

A Aufgabenverteilung

In diesem Abschnitt wird festgehalten, wer welchen Teil der Diplomarbeit verfasst hat. Einige Teile wurden von beiden Mitglieder dieser Arbeit gemeinsam verfasst, da diese in den Aufgabenbereich beider Mitglieder fallen.

A.1 Jakob Lettner

1. **Header:**
    - Eidstattliche Erklärung
    - Gendererklärung
    - Danksagung
    - Abstract
    - Zusammenfassung
2. **Einleitung:**
    - Einleitung
    - Projektinhalt und Überblick
3. **Implementierung**
    - Frontend Implementierung
    - Datenbank
    - Herausforderungen und Lösungen
4. **Ergebnis**
    - Ergebnis Frontend
5. **Resumee**
    - Resumee

A.2 David Tober

1. **Header:**
    - Eidstattliche Erklärung
    - Gendererklärung
    - Danksagung
    - Abstract
    - Zusammenfassung
2. **Einleitung:**
    - Einleitung
    - Projektinhalt und Überblick
3. **Grundlagen und Methoden:**
    - Verwendete Technologien
    - Verwendete Entwicklungssystem
    - Verwendete Bibliotheken und Plug-Ins
    - Sonstige verwendete Software
4. **Implementierung:**
    - Backend Implementation
    - Herausforderungen und Lösungen
5. **Ergebnis**
    - Ergebnis Backend
6. **Resumee**
    - Resumee

B Projektstruktur

Um diese Diplomarbeit professionell durchführen zu können, wurde lange darüber debattiert, welche Projektstruktur am besten geeignet wäre. Es wurde sich dann für das Scrum-Modell entschieden.

Scrum bietet für uns klare Vorteile gegenüber den klassischen Projektmethoden, insbesondere bei Softwareprojekten mit dynamischen Anforderungen. Durch die Entwicklung in sogenannten „Sprints“ konnte flexibel auf Änderungswünsche der CountIT eingegangen werden. Zudem förderten wöchentliche Abstimmungen (*Weekly Scrums*) die enge Kommunikation im Team und halfen dabei, aufkommende Komplikationen frühzeitig zu erkennen und zu beseitigen.

B.1 Unterstützung der Struktur durch Azure DevOps

Durch die Verwendung von Azure DevOps wurde das Planen einzelner Meilensteine schon von Grund auf erleichtert. Durch die von der CountIT gestellten Tickets konnten wir die Zuteilung der Aufgaben bzw. das Erstellen der Meilensteine zügig und effizient abarbeiten.

B.2 Durchführung

Vor dem Start des Praktikums, beziehungsweise dem eigentlichen Start der Diplomarbeit, habe wir uns Gedanken über die Durchführung gemacht. Die theoretische Struktur des Durchführungsplans ist in drei Unterteilungen gegliedert:

- Planung
- Durchführung
- Ergebnis

Planung: Als Planung gilt der Zeitraum, indem wir uns für das Praktikum vorbereitet haben. In dieser Zeit wurden theoretische Grundlagen wie Meilensteine und Sonstiges gelegt.

Durchführung: Als Durchführung gilt der Zeitraum, indem wir die praktische Arbeit geleistet haben.

Ergebnis: Als Ergebnis haben wir den Abschnitt der Arbeit beschrieben, indem wir die schriftliche Arbeit verfassen und diese Präsentieren.

B.3 Zeitaufzeichnung

Um auf die Sekunde genau nachvollziehen zu können, wer, was, wann gearbeitet hat, wurde Clockify verwendet. Clockify ermöglichte uns eine einfache und effiziente Veranschaulichung der gearbeiteten Stunden.

Gearbeitete Stunden im Team: Gemeinsam wurden 413,18 Stunden an dem Projekt gearbeitet.

Clockify Dokumentation Jahr 2024 siehe Abbildung 69:

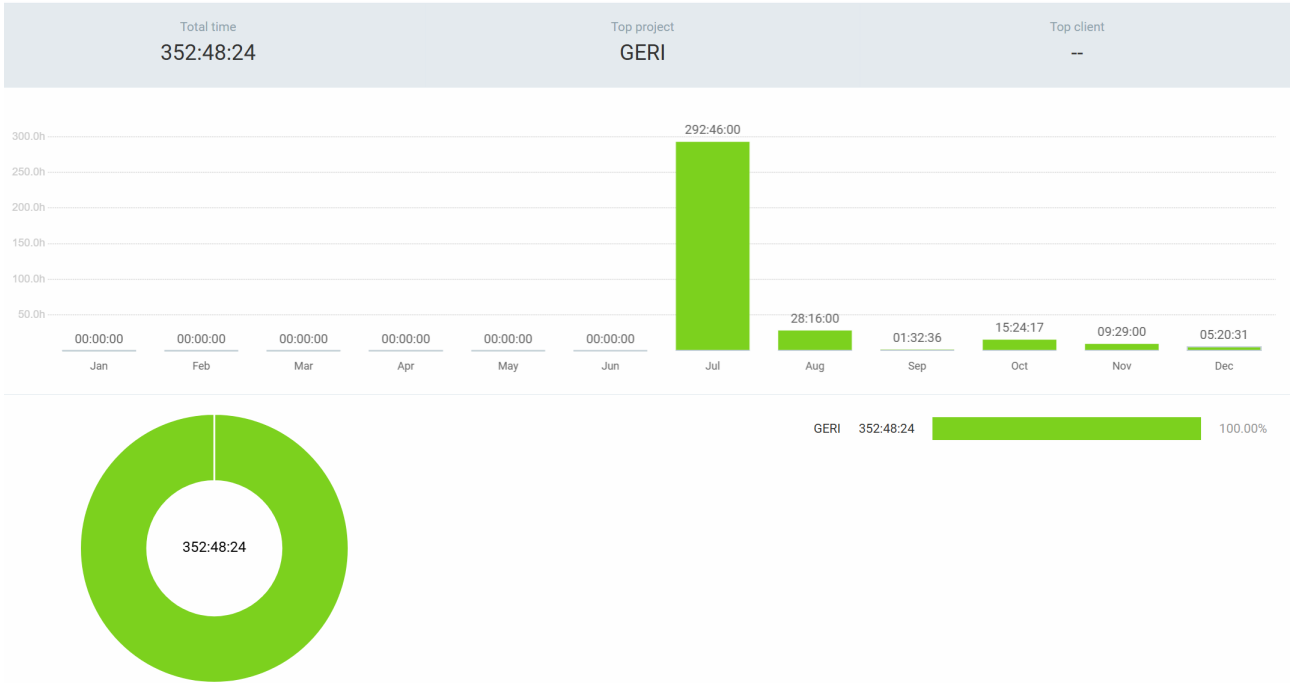


Abbildung 69: time year 2024

Clockify Dokumentation Jahr 2025 siehe Abbildung 70:

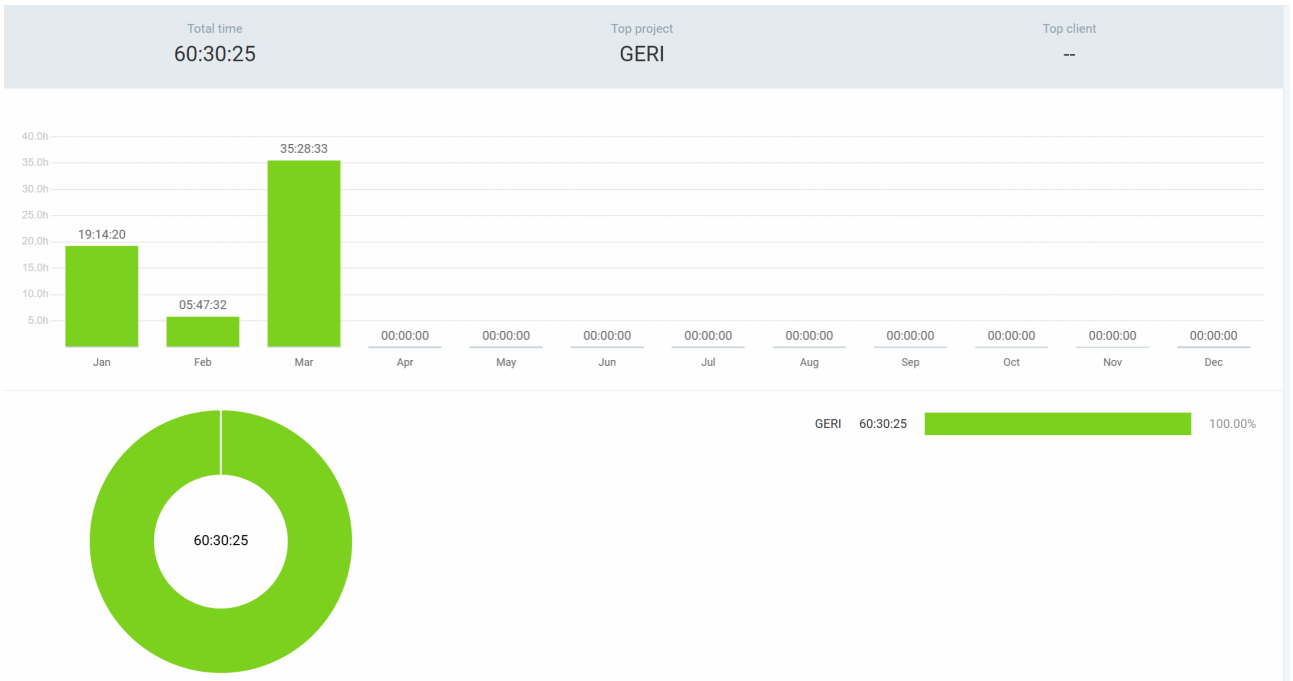


Abbildung 70: time year 2025

C Meilensteine

C.1 Meilensteine - David Tober

Nr	Erklärung	Datum
1	Einlesen in das bestehende Projekt	09.07.2024
2	Auseinandersetzung mit Microsoft Blazor	10.07.2024
3	Erfassen von Problemen/Bugs im bestehenden Programm	10.07.2024
4	Erste bugfreie Version des bestehenden Programms	12.07.2024
5	Besprechung der geplanten Features	15.07.2024
6	Auseinandersetzen mit den neuen Tools zum Auslesen der Hardwareinformation	19.07.2024
7	Beginn Implementierung AGNES	22.07.2024
8	Fertigstellung AGNES	14.12.2024
9	Implementierung des vollautomatischen User-Zu-Rechner-Zuweisungs-Systems	12.01.2025
10	Fertigstellung der Schriftlichen Arbeit	31.03.2025

Tabelle 1: Meilensteinliste David Tober

C.2 Meilensteine - Jakob Lettner

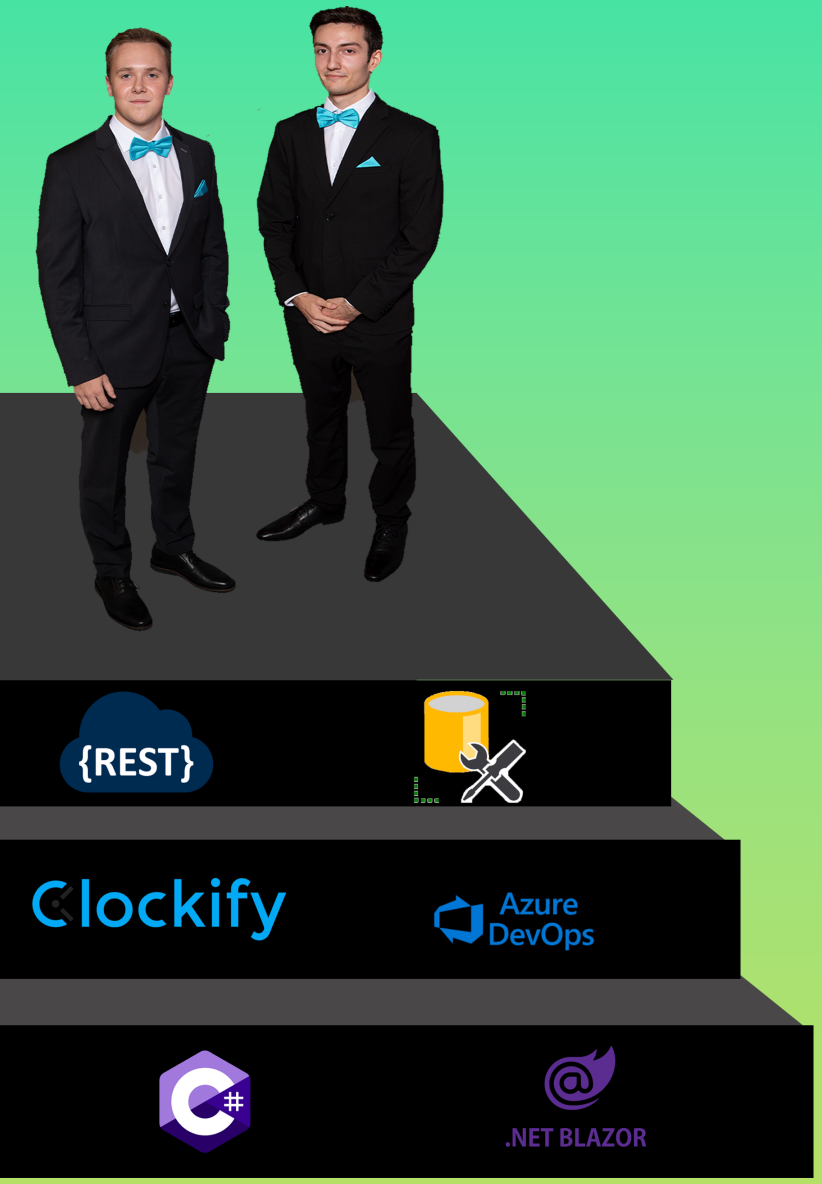
Nr	Erklärung	Datum
1	Einlesen in bestehendes Programm	09.07.2024
2	Auseinandersetzten mit Microsoft Blazor	10.07.2024
3	Erfassen von Problemen/Bugs im bestehenden Programm	10.07.2024
4	Erste Bugfreie Version des bestehenden Programms	12.07.2024
5	Besprechung der geplanten Features	15.07.2024
6	Frontend Design zum ausweisen von Geräten	26.07.2024
7	Gerätetyp wird nur gelöscht, wenn von keiner Vorlage mehr verwendet	30.07.2024
8	Nebenanzeige zur Darstellung der Eigenschaften eines Gerätes	30.07.2024
9	Suchleiste funktioniert Serverseitig	13.11.2024
10	Designänderungen	12.12.2024
11	Fertigstellung Frontend	06.01.2025
12	Fertigstellung Schriftliche Arbeit	31.03.2025

Tabelle 2: Meilensteinliste Jakob Lettner

D Diplomarbeitsplakat

COUNT IT GROUP - GERI

htlperg



GERI ist ein Webtool zum Verwalten der Computer der Firma CountIT. Mit diesem Tool wird es ermöglicht, sämtliche Hardwaredaten der Computer des Unternehmens vollautomatisch einzulesen, sowie einzelne Computer zu bearbeiten, ausweisen, etc.. Ziel dieser Diplomarbeit ist es, den Technical Service des Unternehmens bei seiner Arbeit zu unterstützen.