Höhere Technische Bundeslehranstalt Wien 3, Rennweg
IT & Mechatronik

HTL Rennweg :: Rennweg 89b
A-1030 Wien :: Tel +43 1 24215-10 :: Fax DW 18

Diplomarbeit

Capentory
Digitalisierung der Schulinventur

ausgeführt an der
Höheren Abteilung für Informationstechnologie/Ausbildungsschwerpunkt
Netzwerktechnik
der Höheren Technischen Lehranstalt Wien 3 Rennweg

im Schuljahr 2019/2020

durch

Josip Domazet
Mathias Möller
Hannes Weiss

unter der Anleitung von

DI Clemens Kussbach
DI August Hörandl

Wien, 2. April 2020

Kurzfassung

Aktuell ist eine Inventur an der HTL Rennweg überaus mühsam, da es dafür dreier separater Listen bedarf. Die erste Liste stammt direkt aus dem SAP-System und beinhaltet infolgedessen Informationen über alle Gegenstände in der Organisation. Diese Liste wird fortan als „Primäre Liste“ bezeichnet. Das SAP-System ist eine Datenbank, die durch gesetzliche Vorgaben an dieser Schule verwendet werden muss.

Bei der zweiten und dritten Liste handelt es sich um interne Listen, die sich auf die IT-bezogenen Gegenstände in der Organisation beschränken. Diese Listen werden fortan als „Sekundäre Liste“ bzw. „Tertiäre Liste“ bezeichnet. Die sekundären und tertiären Listen beinhalten zusätzliche Informationen. Sie geben z. B. Aufschluss über Gegenstände, die in der primären Liste lediglich durch einen einzelnen Eintrag abgebildet werden, jedoch in der Realität mehrere verschiedene Gegenstände sind. Weiters spezifizieren sie die Position eines Gegenstandes innerhalb eines Raumes (z. B. steht ein PC in einem Kasten, der wiederum in einem Raum steht). Allerdings sind diese Listen nicht synchron zueinander und führen daher zahlreiche Komplikationen herbei. Das vorliegende Projekt soll die Schulinventarisierung erheblich erleichtern, indem es die erwähnten Listen sinnvoll vereint.

Außerdem soll der Inventurvorgang selbst durch eine mobile Android-Applikation vereinfacht werden. Ausgedruckte Listen, die bisher dafür zum Einsatz kommen, sollen durch die Applikation ersetzt werden. Durch das Scannen von Barcodes können Gegenstände auf der App validiert werden - mit dem angenehmen Nebeneffekt, dass der Inventurvorgang massiv beschleunigt wird. Die Barcodes werden vom SAP-System für jeden Gegenstand generiert und sind in Form eines Aufklebers an den Gegenständen angebracht. Da alle Änderungen protokolliert werden, ist ein genauer Verlauf und damit eine genaue Zuordenbarkeit zu im Serversystem registrierten Benutzern möglich.

Abstract

The current inventory process at HTL Rennweg is extremly tedious due to the fact that three independent lists serve as data source for all items. The first list originates from the SAP-System and therefore, contains information about all items in our organization. Henceforth, this list will be labeled as „Primary Source“. The SAP-System is a database that our school is required to use by law.

The second and third lists are unofficial lists for internal usage that are limited to the IT-related items at our organization. Henceforth, these lists will be labeled as „Secondary Source“ and „Tertiary Source“ respectively. The secondary and tertiary sources contain additional information. For instance, they include items that are represented by only one entry in the primary source but that are multiple different items in reality. Moreover, they specify the position of an item within a room (e. g. a PC might be in a cabinet that itself is located in a room). However, these sources are not in sync with each other and are thus, the cause of countless complications. The aim of this diploma thesis is to simplify the inventory process tremendously by unifying said sources in a reasonable manner.

Furthermore, the current inventory process itself is to be simplified by a mobile Android app. Currently used printed lists are to be replaced by said app. By scanning barcodes, one can validate items on the application – with the pleasant side effect of speeding up the inventory process considerably. Mentioned barcodes are provided for each item by the SAP-System and are physically attached to the items as stickers. Due to the fact that all changes are being logged, there is an exact history and accountability to users that are registered in the server system.

Ehrenwörtliche Erklärung

Ich erkläre an Eides statt, dass ich die individuelle Themenstellung selbstständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Wien, am 2. April 2020

Josip Domazet

Mathias Möller

Hannes Weiss

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Ziele

1.1 Die Idee

Das vorliegende Projekt soll die Schulinventur erheblich erleichtern, indem die Listenproblematik in Angriff genommen wird. Die verschiedenen Listen werden durch einen Server sinnvoll vereint und verwaltet. Inventuren werden über eine App, die mit dem Server kommuniziert, abgewickelt. Bei einer Inventur arbeitet der Benutzer eine Gegenstandsliste für einen Raum ab, indem er die Barcodes der Gegenstände scannt. Die Barcodes entstammen dem SAP-System. Dieser Prozess wird solange wiederholt, bis alle Räume abgeschlossen sind.

Der Server basiert auf dem frei verfügbaren Anlagenverwaltungstool “Ralph”, das von der polnischen Internetauktionsplattform “Allegro” verwendet und entwickelt wird. Die App und der Server sind mit den ursprünglichen Komponenten von Ralph kompatibel. Dies wurde mit verschiedenen Datenbanksichten realisiert. Für die HTL Rennweg ist nur die sogenennante “HTL”-Datenbanksicht relevant.

Sowohl die App als auch der Server wurden unter derselben liberalen Lizenz wie Ralph veröffentlicht (Apache 2.0)[73].

1.1.1 Der Sponsor

Zebra [92] ist der Sponsor des vorliegenden Projektes. Zebra ist ein Hardware-Unternehmen, dass sich auf die Datenerfassung und Datenverarbeitung spezialisiert hat. Zebra hat sich im Laufe des Projektes als guter Kooperationspartner erwiesen. Das Projektteam hat ausschließlich positive Erfahrungen mit Zebra gesammelt und konnte sich glücklich schätzen, einen Partner wie Zebra für das Projekt gewonnen zu haben.

Zebra hat dem Projektteam ein Smartphone zur Verfügung gestellt, das mit einem dedizierten Barcodescanner ausgestattet ist. Damit kann das Scannen von Barcodes rapide vonstatten gehen und der für eine Inventur notwendige Zeitaufwand enorm reduziert werden (siehe Kapitel 5.1, Seite 39).

1.2 Hauptziele

Die folgenden Ziele wurden dem offiziellen Antrag entnommen. Jedes Ziel enthält einen Verweis auf die konkrete Implementierung.

Ziel M1: Online Inventurauflistung

Eine zentrale Auflistung aller in der vorliegenden Organisation enthaltenen und registrierten Gegenstände ist per Webbrowser erreichbar und verwaltbar. Dabei kann nach Eigenschaften der Gegenstände gesucht, gefiltert und sortiert werden. Außerdem wird dabei zwischen Einträgen, die aus der primären Liste und jenen, die aus der sekundären Liste stammen, unterschieden. Einträge der beiden Quellen sind miteinander, sowie mit einem zugehörigen Raum verlinkt. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 7.1, Seite 55.

Ziel M1a: Verlauf auf Gegenstandsbasis

Ein Gegenstand verfügt in der Datenbank über eine Geschichte, die beschreibt, wie sich der Gegenstand durch die verschiedenen Inventuren verändert hat. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 7.1.1.3, Seite 57.

Ziel M1b: Inventurverlauf

Der Server speichert jede vorgenommene Inventur. Somit ist eine Versionsgeschichte an Inventuren abrufbar. Falls eine Inventur gelöscht wird, wird diese nicht unmittelbar verworfen, sondern vorerst in ein Archiv verschoben. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 7.2, Seite 64.

Ziel M1c: Entity-Relationship-Modell

Die Datenbankstruktur inkl. Verlaufsfunktion ist durch ein Entity-Relationship-Modell definiert. Vor der Implementierung wurde die Datenbank skizziert und vorerst auf Papier modelliert. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 7.2, Seite 64.

Ziel M2: Datenimport/-export

Daten von vorhandenen Inventarlisten (Primär und Sekundär) sind in die Datenbank importierbar, wobei Fehler des importierten Datensatzes erkannt und angezeigt werden. Das Importformat ist dabei .csv und .xlsx. Die in der zentralen Datenbank enthaltenen Daten sind in das .csv-, sowie das .xlsx-Format exportierbar. Dabei richtet sich die Formatierung der exportierten Daten an jene der importierten Daten. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 7.1.4, Seite 60.

Ziel M3: Statussystem

Ein Statussystem zeigt Diskrepanzen (etwa Gegenstände, die nur in einer der beiden Importquellen existieren, siehe 2) zwischen durchgeführten Inventuren und den importierten Daten in Form eines speziellen Status-Feldes auf. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 7.1.1, Seite 55.

Ziel M4: Infrastruktur

Das System ist vorbereitet, aus dem Schulnetz per HTTPS erreichbar zu sein. Die nötigen Schritte zur Inbetriebnahme der vorliegenden Infrastruktur sind dokumentiert. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 8.1, Seite 83.

Ziel M4a: Betriebsbereiter Server

Ein Server inkl. Betriebssystem ist betriebsbereit. Die dazu benötigten Ressourcen sind erfasst und stehen dem Serversystem – insofern hardwaretechnisch realisierbar — zur Verfügung. Dieses Ziel wird in folgenden Kapiteln näher erläutert: Kapitel 8.1.1, Seite 83, Kapitel 8.1.2, Seite 84 und Kapitel **??**, Seite **??**.

Ziel M4b: Applikationskonfiguration

Der einsatzfähige Server ist mit einer Konfiguration für den Produktivbetrieb ausgestattet. Die Konnektivität innerhalb des Schulnetzes ist getestet, sofern das Netzwerk dazu fähig ist. Dieses Ziel wird in folgenden Kapiteln näher erläutert: Kapitel 8.1.4.1, Seite 86 und Kapitel 8.1.5, Seite 87.

Ziel M4c: Virtualisierung

Die verschiedenen Komponenten (i.e. Datenbank, Webserver) des Servers sind containervirtualisiert. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 8.1.6, Seite 89.

Ziel M4d: Server-Dokumentation

Die Maßnahmen zur Installation und Inbetriebnahme des Gesamtsystems ist dokumentiert. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 8.1.9, Seite 102.

Ziel M5: Inventur per Android-App

Ein Administrator ist in der Lage mit einer mobilen Android-Applikation eine Inventur durchzuführen. Dabei scannt dieser den Barcode eines Gegenstandes oder gibt den Code manuell ein und hat dann die Möglichkeit diesen Gegenstand zu validieren. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 3, Seite 13.

Ziel M5a: Scanvarianten

Gegenstände werden per integriertem Zebra-Scanner — insofern vorhanden -– gescannt. Alternativ wird dem Benutzer die Möglichkeit angeboten, den Barcode mit der Handykamera — insofern vorhanden -– zu scannen. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 5, Seite 39.

Ziel M5b: Kommunikationsschnittstelle am Server

Die App verfügt über die Fähigkeit, den Server anzusprechen und die benötigten Inventurinformationen zu erhalten. Zusätzlich können auch Inventurveränderungsvorschläge über diese Weise vorgenommen werden. Dieses Ziel wird in folgenden Kapiteln näher erläutert: Kapitel 3.2, Seite 13 und Kapitel 3.4.1, Seite 17.

Ziel M5c: Inventurverhalten auf Raumbasis

Der Benutzer arbeitet im Rahmen einer Inventur eine Liste an Räumen ab, die ihm von der mobilen Applikation angezeigt werden. Hierbei scannt und validiert er die vorhandenen Gegenstände, die ihm angezeigt werden, sowie etwaige unbekannte/unerwartete

Gegenstände (denen im Anschluss entsprechende Einträge zugewiesen werden). Dieses Ziel wird in folgenden Kapiteln näher erläutert: Kapitel 3.3, Seite 14 und Kapitel 3.4.3, Seite 19.

Ziel M6: Serversicherung

Der Server, der die oben beschriebenen Features zur Verfügung stellt, ist selbst von verschiedenen Angriffsszenarien, die in der Schule realistischerweise stattfinden könnten, geschützt, insofern dies technisch umsetzbar ist. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 8.1.7, Seite 97.

1.3 Optionale Ziele

Ziel O1: Texterkennung

Für den Fall, dass der Barcode beschädigt sein sollte, verfügt die Applikation zusätzlich über die Fähigkeit, Text zu scannen. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 5.3.2, Seite 44.

Ziel O2: Bilder am Server

Der Server speichert Bilder zu Gegenständen. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 7.1.1.4, Seite 57.

Ziel O3: Bilder in der App

Die App zeigt Bilder, die am Server Gegenständen zugeordnet wurden, an. Außerdem kann der Benutzer per App Bilder bestimmten Gegenständen zuordnen und diese Bilder am Server hochladen. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 3.3, Seite 16.

Ziel O4: Benutzerdefinierte Felder

Der Server speichert auf Gegenstandsbasis benutzerdefinierte Felder, die wiederum von der App angezeigt werden können. Dieses Ziel wird in folgendem Kapitel näher

erläutert: Kapitel 3.3, Seite 15.

Ziel O5: Anhänge

Anhänge können Gegenständen zugewiesen werden. Die Speicherung erfolgt hierbei so, dass ein Anhang mehreren Gegenständen zugewiesen werden kann und somit keine Duplikate gespeichert werden müssen. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 7.1.1.4, Seite 57.

Ziel O6: Ralph-Pull-Request

Es wird versucht, die entwickelte Inventurfunktion in den offiziellen Ralph-Code einzuführen. Dies wird auf Versionsverwaltungssystemen mit einem `Pull-Request` realisiert. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 7.2.7, Seite 81.

Ziel O7: Automatische Importkorrektur

Der Benutzer hat die Möglichkeit nach dem Import vordefinierte Änderungen anzuwenden. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 7.1.4, Seite 60.

Ziel O8: Servermonitoring

Der Server ist für Monitoring vorbereitet. Dieses Ziel wird in folgendem Kapitel näher erläutert: Kapitel 8.1.8, Seite 99.

1.4 Gesamtübersicht

Die vorliegende Diplomarbeit kann vereinfacht durch die Abbildung 1.1, Seite 7 ausgedrückt werden. Die vorliegende Diplomarbeit kann dabei in drei Bereiche unterteilt werden. Die restlichen Kapitel erläutern die konkrete Implementierung und Funktionsweise dieser Bereiche:

- Die Appsoftware (Kapitel 2, Seite 9 bis inkl. Kapitel 5, Seite 39)
- Die Serversoftware (Kapitel 6, Seite 47 bis inkl. Kapitel 7, Seite 55)
- Die Infrastruktur, auf der die Serversoftware ausgeführt wird (Kapitel 8, Seite 83)

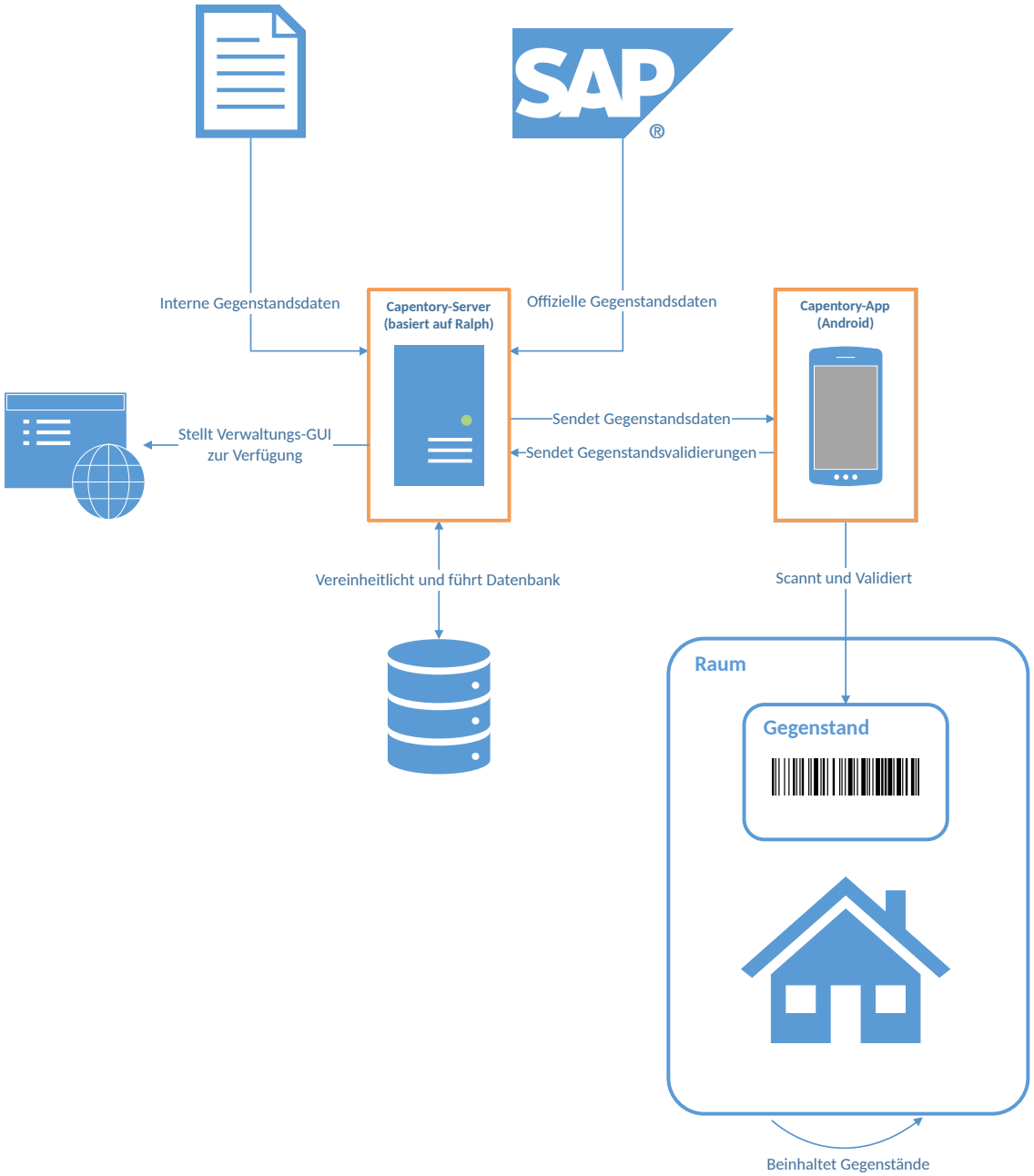


Abbildung 1.1: Gesamtübersicht

2 Einführung in die App

Das Ziel der Diplomarbeit ist es, eine App zu entwickeln, mit der man in der Lage ist, eine Inventur durchzuführen. Um zu verstehen, wieso sich das Projektteam für eine native App entschieden hat, muss man zwischen zwei Begriffen unterscheiden [42]:

- Native App
- Web-App

2.1 Nativ versus Web

Unter einer nativen App versteht man eine App, die für ein bestimmtes Betriebssystem geschrieben wurde [5]. Eine Web-App hingegen basiert auf HTML und wird per Browser aufgerufen. Sie stellt nichts anderes als eine für mobile Geräte optimierte Website dar.

2.2 Begründung für die Wahl der nativen App

Das Projektteam hat sich für eine native App entschieden. Um diese Entscheidung nachvollziehen zu können, ist ein tieferer Einblick in den gegebenen Use-Case erforderlich.

Das Ziel ist es nicht, möglichst viele Downloads im Play Store zu erzielen oder etwaige Marketingmaßnahmen zu setzen. Es soll stattdessen mit den gegebenen Ressourcen eine Inventurlösung entwickelt werden, die die bestmögliche Lösung für die vorliegende Schule darstellt. Eine native App wird eine Web-App hinsichtlich Qualität und User Experience immer klar übertreffen.

Im vorliegenden Fall wäre es sicherlich möglich, eine Inventur mittels Web-App durchzuführen, allerdings würde diese vor allem in den Bereichen Performanz und Verlässlichkeit Mängel aufweisen. Diese zwei Bereiche stellen genau die zwei Problembereiche dar, die es mit der vorliegenden Gesamtlösung bestmöglich zu optimieren gilt. Des Weiteren bieten sich native Apps ebenfalls für komplexe Projekte an, da Web-Apps aktuell

noch nicht in der Lage sind, komplexe Aufgabenstellungen mit vergleichbar geringem Aufwand zu inkorporieren [87]. Die vorliegende Arbeit ist dabei bereits allein aufgrund der in Betracht zu ziehenden Sonderfälle als komplexe Aufgabenstellung einzustufen.

Unter Berücksichtigung dieser Gesichtspunkte wurde also der Entschluss gefasst, eine native Applikation zu entwickeln, da diese ein insgesamt besseres Produkt darstellen wird. Es sei gesagt, dass es auch hybride Apps gibt. Diese sind jedoch einer nativen App in denselben Aspekten wie eine Web-App klar unterlegen.

2.2.1 Auswahl der nativen Technologie

Folgende native Alternativen waren zu vergleichen:

- Flutter [43]
- Xamarin [89]
- Native IOS
- Native Android (Java/Kotlin)

Flutter ist ein von Google entwickeltes Framework, dass eine gemeinsame Codebasis für Android und IOS anbietet. Eine gemeinsame Codebasis wird oftmals unter dem Begriff `cross-platform` zusammengefasst und bedeutet, dass man eine mit Flutter entwickelte native App sowohl mit Android-Geräten als auch mit IOS-Geräten verwenden kann. Flutter ist eine relativ neue Plattform – die erste stabile Version wurde erst im Dezember 2018 veröffentlicht [41]. Außerdem verwendet Flutter die Programmiersprache `Dart`, die Java ähnelt. Diese Umstände sind ein Segen und Fluch zugleich. Flutter wird in Zukunft sicherlich weiterhin an Popularität zulegen, allerdings ist die Anzahl an verfügbarer Dokumentation für das junge Flutter im Vergleich zu den anderen Optionen immer noch weitaus geringer.

Xamarin ist ebenfalls ein cross-platform Framework, das jedoch in C# geschrieben wird und älter (und damit bewährter) als Flutter ist. Weiters macht Xamarin von der proprietären .NET-Platform Gebrauch. Infolgedessen haben alle Xamarin-Apps Zugriff auf ein umfassendes Repertoire von .NET-Libraries [90]. Da Xamarin und .NET Microsoft angehören, ist eine leichtere Azure-Integration oftmals ein Argument, das von offzielen Quellen verwendet wird. Xamarin wird – anders als die restlichen Optionen – bevorzugterweise in Visual Studio entwickelt [91].

Native IOS wird nur der Vollständigkeit halber aufgelistet, stellte allerdings zu keinem Zeitpunkt eine wirkliche Alternative dar, weil IOS-Geräte einige Eigenschaften besitzen, die für eine Inventur nicht optimal sind (z. B. die Akkukapazität). Außerdem haben in etwa nur 20% aller Geräte [3] IOS als Betriebssystem und die Entwicklung einer IOS-App wird durch strenge Voraussetzungen äußerst unattraktiv gemacht. So kann man beispielsweise nur auf einem Apple-Gerät IOS-Apps entwickeln.

2.2.1.1 Begründung: Natives Android (Java)

Die Entscheidung ist schlussendlich auf natives Android (Java) gefallen. Es mag zwar vielleicht nicht die innovativste Entscheidung sein, stellt aber aus folgenden Gründen die bewährteste und risikoloseste Option dar:

- Natives Android ist eine allbekannte und weit etablierte Lösung. Die Wahrscheinlichkeit, dass die Unterstützung durch Google eingestellt wird, ist also äußerst gering.
- Die App wird in den nächsten Jahren immer noch am Stand der Technik sein.
- Natives Android hat mit großem Abstand die umfassendste Dokumentation.
- An der Schule wird Java unterrichtet. Das macht somit eventuelle Modifikationen nach Projektabschluss durch andere Schüler viel einfacher möglich.
- Dadurch, dass Kotlin erst seit 2019 [53] offiziell die von Google bevorzugte Sprache ist, sind die meisten Tutorials immer noch in Java.
- Sehr viele Unternehmen haben viele aktive Java-Entwickler. Dadurch wird die App attraktiver, da die Unternehmensmitarbeiter (von z. B. allegro) keine neue Sprache lernen müssen, um Anpassungen durchzuführen.
- Das Projektteam hat im Rahmen eines Praktikums bereits Erfahrungen mit nativem Java gesammelt.

Aus den Projektzielen hat sich in Absprache mit den Betreuern ergeben, dass die App nicht auf jedem "Steinzeitgerät" zu funktionieren hat. Das minimale API-Level der App ist daher 21 – auch bekannt als Android 5.0 "Lollipop".

2.3 Einführung in natives Java

Um eine Basis für die folgenden Kapitel zu schaffen, werden hier die Basics der Android-Entwicklung mit nativem Java näher beschrieben.

2.3.1 Grundsätzliches

Das Layout einer App wird in XML Dateien gespeichert, während der Programmcode in der Programmiersprache Java erstellt wird. Damit ist die Entwicklung einer Android-App objektorientiert.

2.3.2 Single-Activity-App

Als Einstiegspunkt in eine App dient eine sogenannte `Activity`. Eine Activity ist eine normale Java-Klasse, der durch Vererbung UI-Funktionen verliehen werden.

Bis vor kurzem war es üblich, dass eine App mehrere Activities hat. Das wird bei den Benutzern dadurch bemerkbar, dass die App z. B. bei einem Tastendruck ein weiteres Fenster öffnet, das das bisherige überdeckt. Das neue Fenster ist eine eigene Activity. Google hat sich nun offiziell für sogenannte Single-Activities ausgesprochen [48]. Das heißt, dass es nur eine Activity und mehrere `Fragments` gibt. Ein Fragment ist eine Teilmenge des UIs bzw. einer Activity. Anstatt jetzt beim Tastendruck eine neue Activity zu starten, wird einfach das aktuelle Fragment ausgetauscht. Dadurch, dass keine neuen Fenster geöffnet werden, ist die User Expierence (UX) um ein Vielfaches besser – die Performanz leidet nur minimal darunter. Die vorliegende App ist aus diesen Gründen ebenfalls eine Single-Activity-App.

3 Die Inventurlogik auf der App

Die genaue Bedienung der App ist dem App-Handbuch zu entnehmen. Dieses Kapitel befasst sich mit der Logik hinter einer Inventur.

3.1 Allgemeiner Ablauf einer Inventur

Eine Inventur läuft immer wie folgt ab:

1. Der Benutzer wählt die Inventur aus, an der er arbeiten will.
2. Der Benutzer wählt die Datenbanksicht aus. Dieser Schritt ist erforderlich, da die App auch mit der Ralph-Datenbanksicht kompatibel ist. An der Schule ist die HTL-Datenbanksicht auszuwählen.
3. Der Benutzer begibt sich zu einem Raum und wählt ihn auf der App aus.
4. Der Benutzer erhält die Gegenstandsliste für diesen Raum. Diese Gegenstandsliste gilt es abzuarbeiten.
5. Der Benutzer scannt die Barcodes der Gegenstände und arbeitet sie dadurch ab. Nicht aufgelistete Gegenstände werden automatisch ergänzt.
6. Der Benutzer kann auf Gegenstandsbasis Änderungen an den Feldern eines Gegenstandes vornehmen.
7. Wenn der Benutzer der Meinung ist, alle Gegenstände in diesem Raum erfasst zu haben, sendet er seine Validierungen an den Server.
8. Damit ist der ausgewählte Raum abgeschlossen und der Benutzer kann sich dem nächsten Raum annehmen.

Alle Informationen, die die App bezieht, stammen vom Server.

3.2 Die Modelle

Um die Antworten des Servers abzubilden, wurden mehrere Modell-Klassen erstellt. Folgende Modell-Klassen wurden angelegt:

- `Stocktaking`: Stellt eine Inventur dar.

- `SerializerEntry`: Stellt eine Datenbanksicht dar.
- `Room`: Stellt einen Raum dar.
- `MergedItem`: Stellt einen Gegenstand dar.
- `MergedItemField`: Stellt ein dynamisches Feld dar.
- `Attachment`: Stellt einen Anhang dar.

3.2.1 Designgrundsätze

Eine Modell-Klasse verwaltet etwaige Statusinformationen immer selbst. So weiß beispielsweise nur ein Gegenstand selbst, dass er ursprünglich aus einem anderen Raum stammt. Dies verbessert die Lesbarkeit und Wartbarkeit des Codes massiv, da diese Informationen abstrahiert sind und nicht mehrmals an verschiendenen Stellen im Quellcode schlummern.

3.3 Die Fragments

Eine Inventur wird auf der App durch folgendene Fragments abgewickelt, die gleichzeitig als Phasen verstanden werden können:

- `StocktakingFragment`
- `RoomsFragment`
- `ViewPagerFragment`
  - `MergedItemsFragment`
  - `ValidatedMergedItemsFragment`
- `DetailedItemFragment`
- `AttachmentsFragment`

StocktakingFragment ist das Fragment, in dem der Benutzer die aktuelle Inventur auswählt. Die Inventur kann nur vom Administrator am Server angelegt werden.

Außerdem wählt der Benutzer hier die Datenbanksicht (“den Serializer”) aus. Die App kommuniziert ausschließlich mittels REST API mit dem Server. Diese Schnittstelle kann verschiedene Quellen haben. Die Quellen sind abhängig von der ausgewählten Datenbanksicht.

Durch das Bestätigen eines langegezogenen blauen Buttons gelangt man immer zur nächsten Inventurphase. In diesem Fall gelangt man zum `RoomsFragment`.

RoomsFragment ist das Fragment, in dem der Benutzer den aktuellen Raum über eine DropDown auswählt. Anstatt die DropDown zu verwenden, kann er alternativ auch die Suchleiste verwenden. Als zusätzliche Alternative hat der Benutzer die Möglichkeit den Barcode eines Raumes zu scannen. Nach der Auswahl eines Raumes gelangt man zum `ViewPagerFragment`.

ViewPagerFragment ist das Fragment, das als Wrapper für das `MergedItemsFragment` und das `ValidatedMergedItemsFragment` dient. Die einzige Aufgabe dieses Fragments ist es, die zwei vorher genannten Fragments als Tabs anzuzeigen. Dies wurde mit der neuen `ViewPager2`-Library realisiert [85].

MergedItemsFragment & ValidatedMergedItemsFragment sind die Fragments, die die Gegenstandsliste eines Raumes verwalten und sie dem Benutzer anzeigen. Das `MergedItemsFragment` zeigt dem Benutzer die noch zu validierenden Gegenstände an. Das `ValidatedMergedItemsFragment` erfüllt nur den Zweck, dem Benutzer bereits validierte Gegenstände anzuzeigen und ihm die Möglichkeit zu geben, validierte Gegenstände zurück zu den nicht-validierten Gegenständen in `MergedItemsFragment` zu verschieben. Daher trägt der Tab für das `MergedItemsFragment` die Beschriftung "TODO", währenddessen der Tab für das `ValidatedMergedItemsFragment` die Beschriftung "DONE" trägt.

Beide Fragments verwenden eine `RecyclerView`, um dem Benutzer die Gegenstände anzuzeigen [76]. Eine RecyclerView generiert pro Eintrag ein Layout, hält aber nur die aktuell angezeigten Einträge inkl. Layout im RAM.

Die Gegenstände werden einzeln validiert. Durch das Scannen des Barcodes eines Gegenstands (beziehungsweise durch das Klicken auf seine GUI-Repräsentation) gelangt der Benutzer zum `DetailedItemFragment`.

DetailedItemFragment ist das Fragment, das zur Validierung eines einzelnen Gegenstands dient. Der Benutzer hat hier die Möglichkeit, etwaige Eigenschaften des Gegenstandes (beispielsweise den Anzeigenamen) zu ändern. Ein Formular, dass die Felder eines Gegenstandes beinhaltet, wird einmal angefordert und anschließend für die gesamte Lebensdauer der App gespeichert. Anhand dieses Formulars wird dann eine GUI-Repräsentation dynamisch erstellt.

Das Formular kann `ExtraFields` beinhalten. Das sind Felder, die als nicht essentiel angesehen werden und infolgedessen standardmäßig eingeklappt sind. Dazu gehören auch benutzerdefinierte Felder – sognannte `CustomFields`. `ExtraFields` weisen ansonsten dasselbe Verhalten wie herkömmliche Felder auf.

Folgende GUI-Komponenten wurden statisch implementiert, da sie Felder repräsentieren, die unabhängig von der ausgewählten Datenbanksicht immer vorhanden sind und daher nicht dynamisch sind:

- Ein read-only Textfeld für die Gegendstandsbeschreibung
- Ein read-only Textfeld für den Barcode
- Eine Checkbox “Erst später entscheiden” (siehe Kapitel 7.2.4, Seite 66)
- Eine DropDown für Subrooms (siehe Kapitel 3.4.3.2, Seite 19)

Man braucht für die gesamte Validierung eines Raumes – insofern keine Sonderfälle auftreten – keine Verbindung zum Server. Der Benutzer kann die Gegenstandsliste an einer Lokalität mit einer guten Verbindung anfordern, den Raum mit schlechter Verbindung betreten und alle Gegenstände validieren. Anschließend kann er den Raum verlassen und seine Validierungen an den Server senden. Damit wird der Bedarf an Netzwerkanfragen in Räumen mit schlechter Netzwerkverbindung minimiert.

Der Benutzer kann zusätzlich zur Validierung auch Anhänge für einen Gegenstand definieren, dazu landet er beim `AttachmentsFragment`.

AttachmentsFragment ist das Fragment, das die Anhänge eines Gegenstandes verwaltet. Der Benutzer sieht hier die bereits vorhandenen Anhänge mit Beschriftungen und kann weitere Anhänge hinzufügen. Bilder werden direkt angezeigt. Andere Dateien werden hingegen als Hyperlink dargestellt. Der Benutzer kann diese per Browser runterladen. Zum Hochladen eigener Anhänge greift die App auf den Standard-Dateibrowser des Systems zurück.

Das Outsourcen auf Webbrowser und Dateibrowser bietet den massiven Vorteil, dass man sich die Entwicklung eigener Download-Manager bzw. File-Manager erspart und auf Apps setzen kann, die von namenhaften Herstellern entwickelt werden. Fast jedes System hat bereits beide Komponenten vorinstalliert, daher treten bezüglich der Verfügbarkeit keine Probleme auf.

Beim dem Hochladen von Bildern komprimiert die App jene zuvor. Zur Kompression wird das `WEBP`-Format verwendet, das dem mittlerweile veralteten `JPG`-Standard überlegen ist [86]. Die Qualität des Bildes ist einstellbar:

- 100 % (keine Kompression)
- 95 %
- 85 %
- 75 %

Der Server speichert den gesendeten Anhang nur einmal (Dateien werden anhand von Hashes unterschieden). Wenn ein Benutzer allen PCs in einem EDV-Saal dasselbe Bild

zuweist, wird es nur einmal am Server hinterlegt. Die Anzahl der Anhänge ist nicht limitiert.

3.4 Validierungslogik

`MergedItemsFragment` & `DetailedItemFragment` sind die Fragments, die den Großteil einer Inventur ausmachen. Der Benutzer scannt alle SAP-Barcodes, die sich in einem Raum befinden. Im Idealfall entspricht diese Menge exakt der Menge der Gegenstände, die dem Benutzer im `MergedItemsFragment` angezeigt wird. Im Normalfall wird dies durch etwaige Sonderfälle jedoch nicht gegeben sein. Nach dem Scannen eines Gegenstandes werden die Felder des Gegenstandes dem Benutzer im `DetailedItemFragment` angezeigt. In diesem Fragment hat der Benutzer zwei Buttons, mit denen er den Gegenstand validieren kann:

- **Grüner Button**: Mit diesem Button wird signalisiert, dass sich der Gegenstand im Raum befindet und der Gegenstand wird mitsamt etwaigen Änderungen an seinen Attributen/Feldern übernommen. Dazu wird ein `ValidationEntry` erstellt. Alternativ kann der Benutzer das Klicken dieses Buttons mit dem Schütteln des Gerätes ersetzen. Die Schüttel-Sensibilität ist über die Einstellungen konfigurierbar (und auch deaktivierbar).
- **Roter Button**: Mit diesem Button wird signalisiert, dass sich der Gegenstand nicht im Raum befindet. Dieser Button wird im Normalfall nie betätigt werden, da ein Gegenstand, der sich nicht in diesem Raum befindet, nicht gescannt werden kann und daher dieses Fenster nie geöffnet werden wird. Der Button hat trotzdem einen Sinn, da der Benutzer damit die "TODO"-Liste über das GUI verkleinern kann, um sich einen besseren Überblick zu verschaffen.

3.4.1 Der ValidationEntry

Ein `ValidationEntry` beinhaltet sämtliche Informationen, die der Server benötigt, um die Datensätze eines Gengenstandes entsprechend anzupassen, und stellt eine Gegenstandsvalidierung dar. Ein `ValidationEntry` beinhaltet immer den Primary Key eines Gegenstandes und die Felder, die sich geändert haben. Ein `ValidationEntry` hat daher eine Liste an Feldern `List<Field>`. Wenn sich der Wert eines Feldes geändert hat, wird diese Liste um einen Eintrag erweitert.

Da die Felder wie erwähnt dynamisch (und dadurch generisch) sind, wurden Java Generics eingesetzt [46], um diese abbilden zu können. `Field` ist eine innere Klasse in `ValidationEntry`:

```
public static class Field<T> {
    private String fieldName;
    private T fieldValue;
    ...
```

Ein Feld besteht also immer aus einem Feldnamen und einem generischen Feldwert.

3.4.1.1 Sendeformat

Das `MergedItemsFragment` (bzw. das `MergedItemsViewModel` siehe Kapitel 4, Seite 21) verfügt über eine `HashMap` (`Map<MergedItem, List<ValidationEntry>>`), die alle `ValidationEntries` beinhaltet. Einem Gegenstand ist eine Liste an `ValidationEntries` zugeordnet, da Subitems eigene `ValiditionEntries` bekommen können (siehe Kapitel 3.4.3, Seite 19).

Wenn ein Raum abgeschlossen ist, werden alle `ValidationEntries` in einer Liste vereint und anschließend in eine JSON-Darstellung transformiert. Dieses JSON wird dem Server gesandt und damit ist der aktuelle Raum abgeschlossen. Der Benutzer kann sich nun den restlichen Räumen annehmen. Die genaue Kommunikationsarchitektur wird im Kapitel “Das Repository im Detail” beschrieben (siehe Kapitel 4.3.1, Seite 22).

Der Server erstellt auf Basis der gesendeten `ValidationEntries` Änderungsvorschläge – sogenannte `Change Proposals` (siehe Kapitel 7.2.5, Seite 67). Das `POST`-Format wird im Kapitel “JSON-Schema” beschrieben (siehe Kapitel 7.2.6.5, Seite 74).

3.4.2 QuickScan

Der häufigste Fall einer Inventur wird jener sein, dass ein Gegenstand im richtigen Raum ist und der Benutzer ohne weiteren Input auf den grünen Button drückt. Da dies einen unnötigen Overhead darstellt, wurde die App um den `QuickScan`-Modus erweitert. Hierbei wird sofort nach dem Scannen ein `ValidationEntry` erstellt, ohne dass zuvor das `DetailedItemFragment` geöffnet wird. Dieser Modus ist durch einen Button im `MergedItemsFragment` aktivierbar/deaktivierbar.

Falls ein Sonderfall auftreten sollte, vibriert das Gerät zweimal und öffnet doch das `DetailedItemFragment`. Damit wird gewährleistet, dass der Benutzer nicht irrtümlich mit dem Scannen weitermacht. Er muss diesen Sonderfall händisch validieren. Haptisches Feedback ist für Sonderfälle reserviert.

3.4.3 Sonderfälle auf der App

Ein zentrales Thema der vorliegenden Diplomarbeit ist die Behandlung der Sonderfälle.

3.4.3.1 Subitems

Wenn sich mehrere physische Gegenstände einen Barcode teilen, werden sie in der primären Liste lediglich durch einen einzelnen Eintrag abgebildet. Ein Eintrag des Serversystems kann also mehrere Subitems zusammenfassen.

Der Server inkludiert in seiner Antwort für jeden Gegenstand einen Zähler, der die nötigen Informationen zur Behandlung dieses Sonderfalles beinhaltet. Der Zähler wird in weiterer Folge `times_found_last`-Zähler genannt. Falls ein Gegenstand aus mehreren Subitems bestehen sollte, ist der `times_found_last`-Zähler in der Antwort des Servers größer als 1. Dieser Counter wird dem Benutzer in folgender Form angezeigt: `[Anzahl aktuell gefunden] / [Anzahl zuletzt gefunden]`.

Bei der Validierung eines Subitems wird ein eigener `ValidationEntry` erstellt und die Anzahl der aktuellen Funde erhöht. `Change Proposals` (siehe Kapitel 7.2.5, Seite 67), die auf Basis dieser `ValidationEntries` erstellt werden, werden dem echten "Parent"-Gegenstand zugeordnet und können wahlweise angewandt werden.

Falls ein Gegenstand mehrmals gescannt wird, wird – nach Bestätigung durch den Benutzer – die Anzahl der aktuellen Funde erhöht und wiederum einen `ValidationEntry` erstellt, selbst wenn es sich bei dem Gegenstand aktuell nicht um ein Subitem handelt.

3.4.3.2 Subrooms

Subrooms sind logische Räume in einem Raum (z. B. steht ein PC in einem Kasten, der wiederum in einem Raum steht). Subrooms werden dem Benutzer im `MergedItemsFragment` als einklappbare Zwischenebenen, denen Gegenstände zugeordnet sind, angezeigt. Die Subroom-Zugehörigkeit kann auf Gegenstandsbasis über eine DropDown geändert werden. Die Subroom-Zugehörigkeit wird in einem `ValidationEntry` immer gesetzt, auch wenn sie sich nicht geändert hat.

Die Gegenstandsliste des `MergedItemsFragment` beinhaltet in Wahrheit auch die Subrooms. Dies ist notwendig, da die `RecyclerView`, die dazu genutzt wird dem Benutzer die Gegenstände anzuzeigen, keine Möglichkeit bietet, eine Hierarchie bzw. Zwischenebenen darzustellen. Daher implementieren das `Room`-Modell und das `MergedItem`-Modell das Interface `RecyclerViewItem` und die RecyclerView erhält eine Liste an `RecyclerViewItems` - dies ist ein typisch polymorpher Ansatz. Abhängig vom Typen

des aktuellen Listenelements baut die RecyclerView entweder ein Gegenstandslayout oder ein Raumlayout auf. Die Anzahl der Subrooms ist weder in der Tiefe noch in der Breite limitiert.

3.4.3.3 Unbekannte Gegenstände

Falls ein Gegenstand, der sich nicht in der aktuellen Gegenstandsliste befindet, gescannt wird, muss der Server dazu befragt werden. Es gibt zwei mögliche Antwortszenarien. Die `ValidationEntries` für diese Sonderfälle unterscheiden sich nicht von den bisherigen.

Neuer Gegenstand Der Gegenstand befindet sich überhaupt nicht in der Datenbank. Im "DONE"-Tab haben solche Gegenstände eine blaue Hervorhebung.

Gegenstand aus anderem Raum Der Gegenstand befindet in der Datenbank und stammt ursprünglich aus einem anderen Raum. Im "DONE"-Tab haben solche Gegenstände eine orange Hervorhebung.

4 Die App-Architektur

Die App muss ein verlässliches und vorhersehbares Verhalten aufweisen. Die vom Benutzer erstellten Validierungen dürfen beispielsweise nicht einfach verschwinden. Um das zu gewährleisten, ist eine durchdachte App-Architektur vonnöten. Da die App-Architektur ein sehr zeitintensiver und zentraler Aspekt der vorliegenden Diplomarbeit ist, der als Fundament für die eigentlichen Ziele dient, folgt nun eine ausführliche Erläuterung.

4.1 Separation of Concerns

In Android ist es eine äußerst schlechte Idee, sämtliche Logik in einer Activity oder einem Fragment zu implementieren. Das softwaretechnische Prinzip `Separation of Concerns (SoC)` hat unter Android einen besonderen Stellenwert. Dieses Prinzip beschreibt im Wesentlichen, dass eine Klasse nur einer Aufgabe dienen sollte. Falls eine Klasse mehrere Aufgaben erfüllt, so muss diese auf mehrere logische Komponenten aufgeteilt werden. Beispiel: Eine Activity bzw. ein Fragment hat immer die Verantwortung, die Kommunikation zwischen UI und Benutzer abzuwickeln. Bad Practice wäre es, wenn eine Activity ebenfalls dafür verantwortlich ist, Daten von einem Server abzurufen.

Das Prinzip verfolgt das Ziel, die `God Activity Architecture (GAA)` möglichst zu vermeiden [4]. Eine God-Activity ist unter Android eine Activity, die die komplette Business-Logic beinhaltet und `SoC` in jeglicher Hinsicht widerspricht. God-Activities gilt es dringlichst zu vermeiden, da sie folgende Nachteile mit sich bringen:

- Refactoring wird kompliziert
- Wartung und Dokumentation werden äußerst schwierig
- Automatisiertes Testing (z. B. Unit-Testing) wird nahezu unmöglich gemacht
- Größere Bug-Anfälligkeit
- Im Bezug auf Android gibt es oftmals massive Probleme mit der Konsistenz einer Activity – da eine Activity und ihre Daten schnell vernichtet werden können (z. B. wenn der Benutzer sein Gerät rotiert und das Gerät den Bildschirmmodus wechselt, siehe Kapitel 4.3.2, Seite 27)

God-Activities sind ein typisches Beispiel für Spaghetticode.

Als Reaktion auf eine Vielzahl von Apps, die Probleme mit God-Activites aufwiesen, hat Google Libraries veröffentlicht, die klar auf eine MVVM-Architektur abzielen [62]. Daher fiel die Wahl der App-Architektur auf MVVM.

4.2 Designgrundlagen von MVVM

MVVM steht für Model-View-Viewmodel [88]. Wie man am Namen bereits erkennt, gilt es zwischen drei Komponenten/Ebenen zu unterscheiden [60].

Model Model beschreibt die Ebene der Daten und wird daher oftmals auch als Datenzugriffsschicht bezeichnet. Diese Ebene beinhaltet so viel Anwendungslogik wie möglich.

View View beschreibt die graphische Ebene und umfasst daher das GUI. Diese Ebene soll so wenig Logik wie möglich beinhalten.

ViewModel Das ViewModel dient als Bindeglied zwischen dem Model und der View. Die Logik der View wird in diese Ebene hinaufverschoben.

4.3 MVVM in Android

Mit der Einführung der `Architecture Components` hat Google Android-Entwicklern eine Vielzahl an Libraries zur Verfügung gestellt, um MVVM leichter in Android implementieren zu können [1]. Die konkrete Implementierung in Android ist in Abbildung Abbildung 4.1, Seite 23 ersichtlich.

In dem vorliegenden Fall ist unser `Fragment` die `View`, das `Repository` das `Model` und das `ViewModel` ist in Android namensgleich. MVVM ist streng hierarchisch. Wie in der Abbildung zu erkennen ist, kommuniziert jede Ebene nur mit der hierarchisch nächsten Ebene.

4.3.1 Das Repository im Detail

Wie in der Abbildung 4.1, Seite 23 veranschaulicht, ist das Repository alleinig dafür zuständig, Daten vom Server anzufordern. Beispielsweise wird die Gegenstandsliste

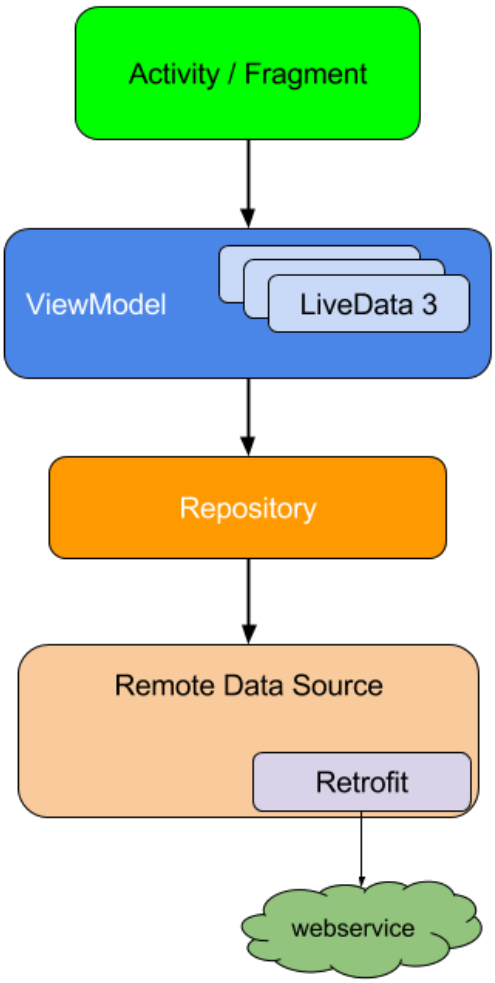


Abbildung 4.1: MVVM in Android nach Google [62] [61]

für einen Raum vom Repository angefordert – und von keiner anderen Ebene. Die Kommunikation zwischen der App und dem Server findet ausschließlich im `JSON`-Format statt. JSON ist ein text-basiertes und kompaktes Datenaustauschformat [34].

Das Repository fordert JSON an und instanziiert anschließend anhand der Serverantwort Objekte der Modell-Klassen (siehe Kapitel 3.2, Seite 13). Das Repository ist die einzige Ebene, die mit den rohen JSON-Antworten des Servers arbeitet. Die restlichen Ebenen arbeiten mit den abstrahierten Objekten, also z. B. mit `MergedItems` oder `Rooms`.

Die Kommunikation zwischen dem Server und der App wird mit zwei Libraries abgewickelt:

- Volley
- Retrofit

4.3.1.1 JsonRequest (Volley)

Android bietet Entwicklern eine Out-of-the-box Netzwerklibrary namens `Volley` an, mithilfe derer man unter anderem JSON-Anfragen verarbeiten kann [2]. Da diese für die vorliegenden Zwecke nicht komplett geeignet war, hat das Diplomarbeitsteam die gegebene Library durch den Einsatz von Vererbung und einer Wrapper-Klasse modifiziert. Die Library wurde in folgenden Punkten angepasst:

- Im Falle eines Fehlers wird die Anfrage wiederholt (Ausnahme: Zeitüberschreitungsfehler). Die maximale Anzahl an Wiederholungen ist limitiert.
- Die maximale Timeout-Dauer wurde erhöht.
- Leere Antworten werden von der App als valide Antwort behandelt und können ohne Fehler verarbeitet werden.
- Im Header der Anfrage wird der Content-Type der Anfrage auf JSON festgelegt.
- Im Header wird das zur Authentifikation notwendige API-Token mitgeschickt. Die Authentifizierung ist über einen Parameter deaktivierbar.
- Im Header wird die Systemsprache des Clients als standardisierter ISO-639-Code mitgesendet. Der Server passt seine Antwort auf die verwendete Sprache an [50]. Die Bezeichnung der Felder, die dem Benutzer auf Gegenstandsbasis angezeigt werden, ist beispielsweise abhängig von der Systemsprache.
- Zum Zeitpunkt der Anfrage ist nicht bekannt, ob die Antwort als `JSONArray` (z. B. für eine Gegenstandsliste) oder als `JSONObject` (z. B. für einen explizit angefragten Gegenstand) erfolgen wird. Da das Backend abhängig von der Anfrage sowohl mit einem `JSONArray` als auch einem `JSONObject` antworten kann, ist der Rückgabewert der Netzwerkanfrage immer ein String. Die Umwandlung erfolgt erst im Repository. Dies führt zu keinen Perfomance-Problemen, da die vorgefertigte Android Library den String zwar zu einem früheren Zeitpunkt aber auf dieselbe Weise umwandeln würde.

Diese Anpassungen wurden aus zwei Gründen vorgenommen:

- Um eine robustere Netzwerklibrary zu erhalten, die es ermöglicht in Räumen mit Netzwerkproblemen dennoch eine Inventur durchzuführen.
- Um eine spätere Abstraktion der Netzwerkanfragen durchführen zu können (siehe Kapitel 4.3.3, Seite 36).

Sämtliche Netzwerkanfragen (die Raumliste, die Gegenstandsliste etc.) werden mit der modifizierten `Volley`-Library durchgeführt. Die einzige Ausnahme sind hierbei die Anhänge (siehe Kapitel 4.3.1.3, Seite 27).

4.3.1.2 JsonRequest – Beispiel

Eine Anfrage wird nie direkt, sondern immer über einen Wrapper ausgeführt. Der Konstruktor ist wie folgt aufgebaut:

- `Context context`: Ist eine Schnittstelle, die globale Information über die App-Umgebung zur Verfügung stellt [12] und von Android zur Verfügung gestellt wird. Jede UI-Komponente (z. B. Textfelder, Buttons, Fragments, etc.) verfügt über einen Context. Ein besonderer Context ist der globale Application-Context. Dieser ist einzigartig und ist ein `Singleton`. Ein Singleton bedeutet, dass von einer Klasse nur ein (globales) Objekt besteht [78].
- `int method`: Ist die HTTP-Methode. Die App verwendet `GET`, `OPTIONS` und `POST`.
- `String url`: Ist die URL, die eine JSON-Antwort liefern soll.
- `@Nullable String requestBody`: Eventuelle Parameter, die an den Server gesendet werden sollen. Dieser Paramter ist für einen `POST`-Request wichtig.
- `NetworkSuccessHandler successHandler`: Funktionales Interface
- `NetworkErrorHandler errorHandler`: Funktionales Interface

Ein Request kann wie folgt ausschauen:

```
RobustJsonRequestExecutioner robustJsonRequestExecutioner =
 new RobustJsonRequestExecutioner(context, Request.Method.GET,
  "https://www.beispiel.org/", null,
  payload -> {
         // TODO: Antwort verarbeiten
         // -> Anhand der Antwort Modell-Objekte instanziieren
     },
  error -> {
         // TODO: Fehler verarbeiten
     });

 robustJsonRequestExecutioner.launchRequest();
```

Wie man sehen kann, sind die letzten beiden Parameter funktionale Interfaces, die dazu dienen, Methoden als Parameter übergeben zu können. Ein Interface mit einer einzigen abstrakten Methode ist als funktionales Interface zu bezeichnen [54]. Da Android Studio Java 8 Language Features unterstützt, verwendet die App mehrheitlich Lambda-Ausdrücke [51]. Lambda-Ausdrücke sind im Wesentlichen dazu da, funktionale Interfaces in vereinfachter Schreibweise verwenden zu können. Da es nur eine einzige abstrakte Methode gibt, kann die Schreibweise simplifiziert werden, weil klar ist, von welcher Methode die Rede ist. Damit fallen – wie im obigen Beispiel zu sehen – redundante Informationen wie Rückgabetyp und Methodenkörper vollständig weg. Das obige Beispiel würde ohne Lambda-Ausdrücke wie folgt ausschauen:

```
RobustJsonRequestExecutioner robustJsonRequestExecutioner =
 new RobustJsonRequestExecutioner(context, Request.Method.GET,
 "https://www.beispiel.org/", null,
     new NetworkSuccessHandler() {
         @Override
         public void handleSuccess(String payload) {
             // TODO: Antwort verarbeiten
             // -> Anhand der Antwort Modell-Objekte instanziieren
         }
     },
     new NetworkErrorHandler() {
         @Override
         public void handleError(Exception error) {
             // TODO: Fehler verarbeiten
         }
     });

robustJsonRequestExecutioner.launchRequest();
```

Lambdas sind eine Option, um `Callbacks` in Java zu implementieren. Ein Callback ("Rückruffunktion") ist eine Methode, die einer anderen Methode als Parameter übergeben werden kann. Die soeben erwähnten funktionalen Interfaces sind typische Callbacks [8]. Es gibt zwei Arten von Callbacks:

- Synchrone Callbacks: Die Ausführung der übergebenen Methode erfolgt sofort.
- Asynchrone Callbacks: Die Ausführung der übergebenen Methode erfolgt zu einem späteren Zeitpunkt.

In diesem Fall wird die Methode `handleSuccess` aufgerufen, sobald der Client die Antwort erhalten hat. Damit handelt es sich um ein asynchrones Callback. Callbacks werden in der App sehr häufig eingesetzt.

4.3.1.3 Retrofit

`Retrofit` ist eine weitere Netzwerk-Library (bzw. Libraries), die das Projektteam eingesetzt hat. Retrofit wurde nur zum Senden von Dateien eingesetzt, weil dies mit Volley nur erschwert möglich ist. Mit dieser Library wurde die Anhang-Funktion realisiert.

Das Projektteam hat bei dem einzigen die bereits bekannte Callback-Logik verwendet:

```
Call<String> call = prepareCall(args);
call.enqueue(new Callback<String>() {
    @Override
    public void onResponse(Call<String> call, Response<String> response) {
        // TODO: Antwort verarbeiten
        // -> Anhand der Antwort Modell-Objekte instanziieren
    }
    @Override
    public void onFailure(Call<String> call, Throwable t) {
        // TODO: Fehler verarbeiten
    }
});
```

In Retrofit werden API-Endpunkte über Interfaces definiert:

```
public interface AttachmentAPI {
    @Multipart
    @POST(SerializerEntry.attachmentUrl)
    Call<String> addFile(@Header("authorization") String auth,
                         @Part MultipartBody.Part file,
                         @Part("description") String description);
}
```

Dies führt zu einem besseren Überblick als bei Volley, da man pro API-Endpunkt des Backends ein Interface hat. Damit ist sofort ersichtlich, mit welchen Backend-Endpunkten ein Repository kommuniziert.

4.3.2 Das Fragment und das ViewModel im Detail

Ein Fragment durchlebt im Laufe seines Daseins eine Vielzahl an Zuständen/Phasen – man spricht von einem `Lifecycle`. Wenn der Benutzer zum Beispiel sein Gerät rotiert, führt dies dazu, dass das Fragment *zerstört* wird und das Fragment durch erneutes

Durchleben alter Zustände wiederaufgebaut wird – dies führt zu einer Zerstörung des aktuellen UIs des Fragments sowie sämtlicher Referenzen, die das Fragment besitzt. Fast alle GUI-Komponenten (Fragments, Textfelder etc.) sind in Android an einen Lifecycle gebunden.

Eine Gerätrotierung gehört zur Kategorie der `Configuration Changes` [84]. Der Grund hierfür liegt darin, dass Android das aktuelle Layout ändert, da beispielsweise andere Layouts (XML-Files) für den Landscape-Modus zur Verfügung stehen [11]. In der Literatur wird der Begriff *zerstören* verwendet, da dabei das Callback `onDestroy` in einer Activity aufgerufen wird.

Folgende Probleme, die eine Inventur massiv erschweren würden, könnten durch Lifecycle-Probleme auftreten:

- Die App stürzt ab, wenn eine Methode ausgeführt wird, die eine Referenz auf ein zerstörtes Objekt hat.
- `Memory Leaks` entstehen, da Referenzen auf zerstörte Objekte vom Gargabe Collector nicht freigegeben werden können. In Android wird die Minimierung des Speicherbedarfs der App einzig und allein vom Garbage Collector übernommen. Falls dieser Objekte nicht freigeben kann, führt dies dazu, dass die App immer mehr und mehr Arbeitsspeicher benötigt. Je nach Größe des Memory Leaks kann dies zu kleineren Verzögerungen bis zu einem Absturz der App führen.
- Nach einem `Configuration Change` gehen die aktuellen Daten verloren und der Benutzer muss das Problem selbst lösen.
- Wenn die aktuellen Daten verloren gehen, verhält sich eine App oftmals unvorhersehbar.

4.3.2.1 ViewModel als Lösung

Bei genauerer Betrachtung der Abbildung 4.2, Seite 29 wird ersichtlich, welche Phasen eine Activity bei einer Gerätrotierung durchlebt:

- Activity wird zerstört:
  - `onPause`
  - `onStop`
  - `onDestroy`
- Activity wird wieder aufgebaut:
  - `onCreate`
  - `onStart`
  - `onResume`

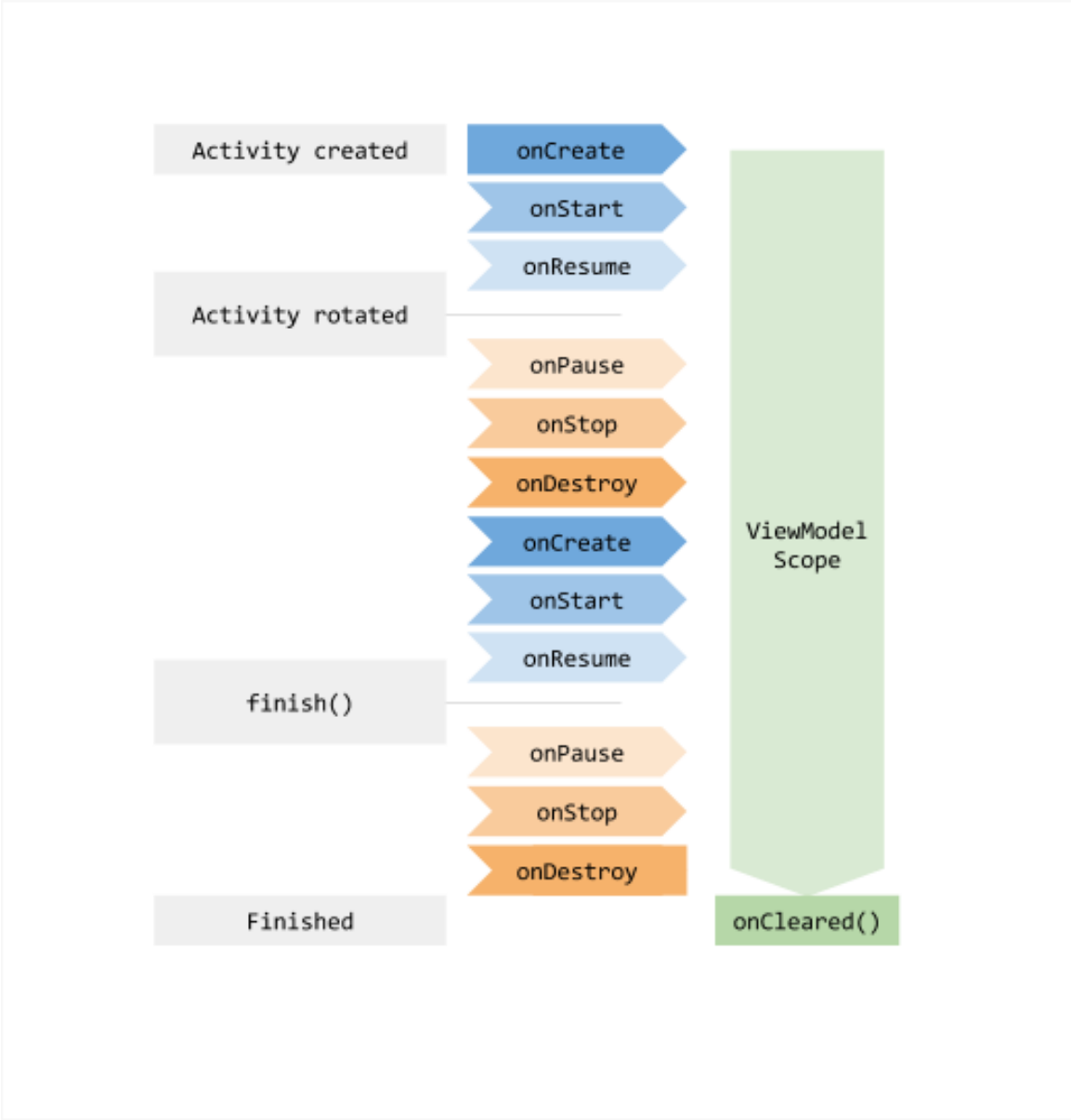


Abbildung 4.2: Zustände einer Activity im Vergleich zu den Zuständen eines ViewModels, Fragments haben einen ähnlichen Lifecycle [44] [84] [55]

Wie in der Abbildung 4.2, Seite 29 zu sehen ist, stellt ein ViewModel eine Lösung für diese Probleme dar. Ein ViewModel ist von einem `Configuration Change` nicht betroffen und kann dem UI damit stets die aktuellen Daten zur Verfügung stellen. Die Daten erhält das ViewModel vom Repository. Der gegebene Sachverhalt trifft genauso auf Fragments zu. Diese haben einen leicht veränderten Lifecycle, sind allerdings genauso von Configuration Changes betroffen wie Activities. Das ViewModel ermöglicht es dem Benutzer also eine Inventur konsistent – ohne unvorhersehbares Lifecycle-Verhalten – durchzuführen.

Anmerkung: Man kann das Zerstören & Wiederaufbauen von Activities/Fragments manuell blockieren. Dies ist jedoch kein Ersatz für eine wohlüberlegte App-Architektur und führt in den meisten Fällen zu unerwünschten Nebenwirkungen, da man sich nun auch manuell um das Wechseln der Konfiguration (Layouts etc.) kümmern muss und dies weitaus komplizierter ist, als auf ViewModels zu setzen [38].

Folgende Details sind bei der Verwendung eines ViewModels zu beachten [82]:

- Ein ViewModel sollte bei einem `Configuration Change` keine neue Netzwerkanfrage starten, da es bereits über die aktuellen Daten verfügt. Dies lässt sich mit einer `If`-Anweisung beheben.
- Referenzen zu Objekten, die an einen Lifecycle gebunden sind, sind ein absolutes NO-GO. Objekte mit Lifecycle haben ein klares Schicksal – wenn ihr Host vernichtet wird, müssen sie ebenfalls vernichtet werden. **Folgendes Szenario**: Ein ViewModel hat eine `TextView`-Variable (= ein Textfeld). Dreht der Benutzer sein Gerät wird das aktuelle Fragment inklusive `TextView` vernichtet. Das ViewModel überlebt den `Configuration Change` und hat nun eine Referenz auf eine invalide `TextView`. Dies ist ein `Memory Leak`.
- ViewModel überleben ein Beenden des App-Prozesses nicht. Wenn das BS aktuell wenig Ressourcen zur Verfügung hat, kann es sein, dass Apps kurzzeitig beendet werden. Falls man diesen Sonderfall behandeln will, ist dies mit Extra-Aufwand verbunden [83].
- ViewModels sollen nicht zu "God-ViewModels" werden. Das `SoC`-Prinzip ist anzuwenden.

Wie gelangen die Daten wie z. B. die Raumliste ins UI, wenn das ViewModel keine Referenzen auf das UI haben darf? Die Antwort lautet `LiveData`.

4.3.2.2 LiveData

`LiveData` ist eine observierbare Container-Klasse. Observierbar heißt, dass bei Änderungen des enkapsulierten Objektes ein Callback aufgerufen wird. `LiveData` ist (wie ein ViewModel) `lifecycle-aware`. Daher wird `LiveData` immer nur aktive Komponenten mit Daten versorgen. Eine `TextView`, die bereits zerstört wurde, erhält dementsprechend auch keine Updates mehr.

Dieses (angepasste) offizielle Beispiel veranschaulicht die Funktionsweise sehr gut [56]. Im Beispiel soll ein Benutzername angezeigt werden:

```
public class NameViewModel extends ViewModel {

    // LiveData-Objekt, das einen String beinhaltet
    private MutableLiveData<String> currentName;

    public LiveData<String> getCurrentName() {
        if (currentName == null) {
            currentName = new MutableLiveData<>();
        }
        // Benutzernamen bekannt geben
        currentName.postValue("Max Mustermann");

        return currentName;
    }

    // Rest des ViewModels...
}
```

Der Unterschied zwischen `MutableLiveData` und `LiveData` besteht darin, dass letzteres nicht veränderbar ist. Mit der `postValue`-Methode kann einer MutableLiveData-Instanz, die ja als Container-Objekt dient, ein neuer Wert zugewiesen werden. Dadurch werden etwaige Callbacks aufgerufen (siehe nächster Code-Ausschnitt). Man sollte bei öffentlichen Methoden immer nur `LiveData` als Rückgabewert verwenden, damit auf der Ebene der View keine Modifikationen der Daten des ViewModels vorgenommen werden können. Im Realfall stammt `LiveData` ursprünglich aus dem Repository.

```
public class NameFragment extends Fragment {
 @Override
    public void onViewCreated(@NonNull View view,
    @Nullable Bundle savedInstanceState) {
        ...

        // Mit dieser Anweisung wird ein ViewModel erstellt
        model = new ViewModelProvider(this).get(NameViewModel.class);

        model.getCurrentName().observe(getViewLifecycleOwner(),
         currentName -> {
            // Diese Methode wird bei Änderungen aufgerufen.
            // currentName ist ein String.
            // nameTextView ist ein TextFeld,
            // das den aktuellen Benutzernamen anzeigt.
            // Mit .setText(String) kann der angezeigte Text
```

```
            // geändert werden.
            nameTextView.setText(currentName);
        });
}
```

Hier kommt wieder die vorher angesprochene Lambda-Syntax zum Einsatz. Die Methode wird einmal beim erstmaligen Registrieren aufgerufen und wird danach bei jeder weiteren Änderung aufgerufen. Im Callback arbeitet man direkt mit dem eigentlichem Datentypen – in diesem Fall mit einem String –, da `LiveData` nur ein Container-Objekt ist. Im Fragment befindet sich damit relativ wenig Logik. Das Fragment hört nur auf eventuelle Änderungen und aktualisiert das UI in Abhängigkeit von den Änderungen. Ein Großteil der Logik (wie z. B. Validierungen oder Gegenstandslisten) befindet sich also im ViewModel.

Das LiveData-Objekt ist an `getViewLifeycleOwner()` gebunden. Wenn der `LifecycleOwner` inaktiv wird, werden keine Änderungen mehr entsandt. Man könnte auch `this` als Argument übergeben (`this` wäre in diesem Fall das Fragment, das ebenfalls über einen Lifecycle verfügt). `getViewLifeycleOwner()` hat jedoch den Vorteil, dass der Observer automatisch entfernt wird, sobald der LifecycleOwner zerstört wird.

4.3.2.3 Angepasste LiveData-Klasse

Für den vorliegenden Usecase reichen jedoch die Nutzdaten allein nicht. Es sind weitere Informationen über den Status der Nutzdaten erforderlich. **Beispiel:** Wenn eine Anfrage zehn Sekunden benötigt, um am Client anzukommen, muss dem Benutzer mittels Ladebalken (= `ProgressBar`) signalisiert werden, dass er auf das Backend zu warten hat.

Man könnte jetzt im ViewModel `LiveData<Boolean> isFetching` verwenden und im Fragment dieses LiveData-Objekt observieren. Falls der aktuelle Wert `true` ist, wird die ProgressBar angezeigt. Falls der Wert auf `false` geändert wird, werden stattdessen die nun zur Verfügung stehenden Nutzdaten angezeigt.

Bei mehreren Netzwerkanfragen wird dies bald unübersichtlich, da mehrere LiveData-Objekte vonnöten sind, die aus logischer Perspektive zu einem bereits bestehenden LiveData-Objekt gehören – den Nutzdaten. Daher hat das Diplomarbeitsteam – wie von Google [47] und von einem StackOverflow-Thread [80] empfohlen – die Nutzdaten in einer Wrapper-Klasse (`StatusAwareData`) enkapsuliert:

```
public class StatusAwareData<T> {

    // Status der Nutzdaten
    @NonNull
```

```
    private State status;

    // Nutzdaten
    @Nullable
    private T data;

    // Eventueller Fehler
    @Nullable
    private Throwable error;

    ...

    @NonNull
    public State getStatus() {
        return status;
    }

    @Nullable
    public T getData() {
        return data;
    }

    @Nullable
    public Throwable getError() {
        return error;
    }

    // Enum, das die legalen Status definiert
    public enum State {
        INITIALIZED,
        SUCCESS,
        ERROR,
        FETCHING
    }
}
```

Diese Wrapper-Klasse speichert Nutzdaten eines generischen Typen. Der Status der Daten wird durch ein `Enum` abgebildet. Folgende Status können Nutzdaten haben:

- `INITIALIZED`: Dieser Status bedeutet, dass das Objekt soeben erstellt wurde. Wird nur bei der erstmaligen Instanziierung verwendet.
- `SUCCESS`: Dieser Status bedeutet, dass die Nutzdaten `data` bereit sind.
- `ERROR`: Dieser Status bedeutet, dass eine Netzwerkanfrage fehlgeschlagen ist (Ob am Client oder am Server spielt keine Rolle). In diesem Fall ist die `error`-Variable gesetzt.

- `FETCHING`: Dieser Status bedeutet, dass auf eine Netzwerkanfrage gewartet wird.

In Android sollte man Enums vermeiden, da diese um ein Vielfaches mehr Arbeitsspeicher und persistenten Speicher benötigen als ihre Alternativen. Als Alternative kann man auf Konstanten zurückgreifen. Dieser Code-Ausschnitt stellt das einzige Enum des gesamten Projektes dar [39].

Um diese Wrapper-Klasse elegant benutzen zu können, hat das Projektteam `MutableLiveData` durch Vererbung angepasst:

```
public class StatusAwareLiveData<T>
    extends MutableLiveData<StatusAwareData<T>> {

    public void postFetching() {
        // Instanziiert StatusAwareData-Objekt mit FECHTING als
        // aktuellen Status.
        // Setzt das StatusAwareData-Objekt anschließend
        // als Wert der LiveData-Instanz.
        postValue(new StatusAwareData<T>().fetching());
    }

    public void postError(Exception exception) {
        // Instanziiert StatusAwareData-Objekt mit ERROR als
        // aktuellen Status.
        // Mit der übergebenenen Exception wird die error-Variable
        // initialisiert.
        // Setzt das StatusAwareData-Objekt anschließend
        // als Wert der LiveData-Instanz.
        postValue(new StatusAwareData<T>().error(exception));
    }

    public void postSuccess(T data) {
        // Instanziiert StatusAwareData-Objekt mit SUCCESS als
        // aktuellen Status.
        // Mit dem übergebenenen Objekt wird die data-Variable
        // initialisiert.
        // Setzt das StatusAwareData-Objekt anschließend
        // als Wert der LiveData-Instanz.
        postValue(new StatusAwareData<T>().success(data));
    }

}
```

Damit entfällt der Bedarf, selbst neue StatusAwareData-Objekte zu instanziieren, da diese bereits über die `postFetching`-, `postError`- und `postSuccess`-Methoden – mit

korrektem Status – instanziiert werden. Infolgedessen ist `StatusAwareData` abstrahiert und im ViewModel genügt es, mit den modifizierten LiveData-Instanzen zu arbeiten. Damit ändert sich das vorherige "Beispiel" wie folgt:

```
// Im ViewModel:

// StatusAwareLiveData-Objekt, das einen String mit Status beinhaltet
private StatusAwareLiveData<String> currentName;

public LiveData<StatusAwareData<String>> getCurrentName() {
    if (currentName == null) {
        currentName = new StatusAwareLiveData<>();
    }
    // Benutzernamen bekannt geben, hier wird eine
    // erfolgreiche Netzwerkanfrage simuliert.
    currentName.postSuccess("Max Mustermann");

    return currentName;
}


// Im Fragment:

model.getCurrentName().observe(getViewLifecycleOwner(),
 statusAwareData -> {
      switch (statusAwareData.getStatus()) {
          case SUCCESS:
              // TODO: Nutzdaten anzeigen
              nameTextView.setText(statusAwareData.getData());
              break;
          case ERROR:
              // TODO: Fehlermeldung anzeigen
              ...
              break;
          case FETCHING:
              // TODO: Ladebalken anzeigen
              ...
              break;
      }
   });
```

4.3.3 Konkrete MVVM-Implementierung

Im vorliegenden Anwendungsfall zeigt das Fragment immer mindestens einen Datensatz an, der für das Fragment (und die anderen Ebenen) namensgebend ist. Der Room-Screen (also die Anzeige mit einer DropDown zur Raumauswahl) setzt sich beispielsweise aus folgenden Komponenten zusammen:

- Der `RoomsRepository`-Klasse
- Der `RoomsViewModel`-Klasse
- Der `RoomsFragment`-Klasse
- Der `fragment_rooms.xml`-Datei, die das UI-Layout definiert

Das `RoomsRepository` ist dafür verantwortlich, die Raumliste vom Backend anzufordern und in Objekte der Modell-Klasse `Room` umzuwandeln. Das `RoomsViewModel` ist dafür verantwortlich, dem Fragment LiveData-Objekte zur Verfügung zu stellen. Das `RoomsFragment` ist dafür verantwortlich, dem Benutzer die Räume anzuzeigen, indem es `LiveData` observiert. Alternativ zeigt es Fehlermeldungen bzw. einen Ladebalken an.

Bei genauerer Betrachtung wird klar, dass fast jeder Screen dieselbe Aufgabe hat:

- Das Repository fordert Daten vom Backend an und wandelt die JSON-Antwort in Modell-Objekte um.
- Das ViewModel abstrahiert Logik und stellt der View `LiveData` zur Verfügung.
- Das Fragment zeigt entweder Nutzdaten, eine Fehlermeldung oder einen Ladebalken an.

Hier greift das softwaretechnische Prinzip `Do not repeat yourself (DRY)` [32]. Anstatt `Boilerplate-Code` für jeden einzelnen Screen kopieren zu müssen, hat das Projektteam diese sich wiederholende Logik abstrahiert. Boilerplate-Code sind Code-Abschnitte, die sich immer wieder wiederholen [79]. Wiederholende Logik sollte immer in eine Superklasse abstrahiert werden. Das Projektteam hat demnach drei abstrakte Klassen definiert, die die Menge an Boilerplate-Code signifikant reduzieren:

- `NetworkRepository`
- `NetworkViewModel`
- `NetworkFragment`

Alle Komponenten von inventurrelevanten Screens erben von diesen drei Klassen. Damit hat das Projektteam folgende Vorteile aggregiert:

- Abstrahierte Fehlerbehandlung
- Abstrahierte Ladeanzeige
- Abstrahierter Netzwerkzugriff

- Abstrahiertes Refreshverhalten (durch nach unten wischen wie z. B. bei YouTube)

Das Refactoring, das dies realisierte, war zwar zeitintensiv, hat sich jedoch mittlerweile mehr als rentiert. Das Hinzufügen von neuen Screens benötigt nur mehr einen Bruchteil des ursprünglichen Codes und jeder neue Screen hat automatisch die aufgezählten Features inkludiert. Infolgedessen wird das Erweitern der App um neue Features enorm erleichtert.

5 Das Scannen

Das Scannen ist ein vitaler Aspekt der vorliegenden Diplomarbeit. Gegenstände an der vorliegenden Organisation sind mit Barcodes ausgestattet. Der Benutzer wird den Großteil seiner Zeit damit verbringen, die Gegenstandsliste durch das Scannen von Barcodes abzuarbeiten. Um die Produktivität maximal zu steigern, muss die App in der Lage sein, diese Barcodes möglichst schnell zu erfassen. Es werden folgende Scanvarianten angeboten:

- Zebra-Scan
- Kamerascan
- Manuelle Eingabe (für den Fall, dass ein Scan fehlschlägt)

5.1 Der Zebra-Scan

Der hervorragende Sponsor dieser Diplomarbeit – Zebra – hat dem Diplomarbeitsteam einen `TC56` ("Touch Computer") zur Verfügung gestellt. Dieser verfügt über einige Eigenschaften, die für eine Inventur vom Vorteil sind [81]:

- Ein in das Smartphone integrierter Barcodescanner reduziert die Scanzeiten drastisch.
- Ein Akku mit über 4000 mAh ermöglicht mehrstündige Inventuren.
- Viel Arbeitsspeicher und ein leistungsstarker Prozessor ermöglichen ein flüssiges App-Verhalten.
- Dank robuster Bauart hält das Gerät auch physisch anspruchsvollere Phasen einer Inventur aus.

5.1.1 Zebra-Scan: Funktionsweise

Die App kommuniziert nicht direkt mit dem Scanner. Auf dem Zebra-Gerät läuft im Hintergrund immer die DataWedge-Applikation. Dies ist eine App, die die Behandlung des tatsächlichen Scans abwickelt und das Ergebnis auf mehrere Arten aussendet [14]. Beispielsweise wird das Ergebnis an die Tastatur geschickt, aber auch als `Broadcast` an das Betriebssystem [7].

Die App registriert sich beim Betriebssystem und hört auf den Broadcast, der den Barcode enthält und automatisch von DataWedge entsandt wird. Broadcasts werden durch eine String-ID unterschieden, die über DataWedge konfiguriert wird. Derartige Ansätze werden als Publish–subscribe-Model bezeichnet \cite{publish–subscribe}.

Dies bietet folgende Vorteile:

- Die Hardware wird komplett abstrahiert. Die vorliegende App "sieht" den Scanner zu keinem Zeitpunkt.
- Durch die Abstrahierung des Scanners wird die eigene Codebasis kleiner und weniger kompliziert.
- DataWedge wird von Zebra entwickelt. Damit hat man eine gewisse Sicherheit, dass der Scanner verlässlich funktioniert.

Bei weiterer Überlegung kommt man außerdem zur Erkenntnis, dass der Broadcast nicht von DataWedge stammen muss. Wenn eine beliebige andere App, einen Broadcast mit derselben ID ausschickt, wird die App dies als Scan werten. In weiterer Folge ist es zumindest theoretisch möglich, beispielsweise einen Bluetooth-Scanner mit einem regulären Android-Gerät zu verwenden und bei einem Resultat einen Broadcast mit derselben ID auszuschicken. Die App würde keinen Unterschied bemerken und somit auch mit dem Bluetooth-Scanner funktionieren. Dieses Einsatzgebiet wurde vom Diplomarbeitsteam jedoch nicht getestet.

5.1.2 Zebra-Scan: Codeausschnitt

Broadcast können in Android durch `BroadcastReceiver` ausgelesen werden. Auch hier wurde das DRY-Prinzip angewandt. Jedes Fragment, das Scanergebnisse braucht, verwendet nahezu denselben BroadcastReceiver. Daher hat das Team einen eigenen BroadcastReceiver erstellt, der die gemeinsamen Eigenschaften zusammenführt. Der gesamte Code für den Zebra-Scan konnte damit relativ kompakt in einer Klasse eingebunden werden:

```
public class ZebraBroadcastReceiver extends BroadcastReceiver {
    ...

    public static void registerZebraReceiver(
                        Context context,
                        ZebraBroadcastReceiver zebraBroadcastReceiver,
                        ErrorHandler errorHandler) {
        ...
        // Der BroadcastReceiver muss beim BS registriert werden.

        IntentFilter filter = new IntentFilter();
```

```
        filter.addCategory(Intent.CATEGORY_DEFAULT);

        // R.string.activity_intent_filter_action definiert
        // die konfigurierte ID des Broadcasts.
        filter.addAction(
            context.getResources()
            .getString(R.string.activity_intent_filter_action));

        // Hier wird der BroadcastReceiver mit der ID,
        // auf die er zu hören hat, registriert.
        context.registerReceiver(zebraBroadcastReceiver, filter);
    }


    public static void unregisterZebraReceiver(
                    Context context,
                    ZebraBroadcastReceiver zebraBroadcastReceiver) {
         // Falls ein BrodcastReceiver nicht mehr gebraucht wird,
         // sollte er immer beim BS abgemeldet werden.

        context.unregisterReceiver(zebraBroadcastReceiver);
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        // Falls ein Broadcast eintrifft,
        // wird diese Methode aufgerufen.

        String action = intent.getAction();

        // R.string.activity_intent_filter_action definiert
        // die konfigurierte ID des Broadcasts.
        // Somit kann festgestellt werden,
        // ob es sich um den richtigen Broadcast handelt.
        if (action.equals(context.getResources()
           .getString(R.string.activity_intent_filter_action))) {

            // R.string.datawedge_intent_key_data
            // definiert die ID des Scanergebnises selbst.
            // Der Scan liefert auch andere Ergebnisse,
            // wie z.B. das Barcodeformat.
            // Relevant ist nur der Barcode.
            String barcode = intent.getStringExtra(
                context.getResources()
                .getString(R.string.datawedge_intent_key_data));
```

```
                // Das Scanergebnis wird nun per
                // funktionalem Interface an die Fragments weitergegeben.
                scanListener.handleZebraScan(barcode);
            }
        }
    }
```

5.2 Der Kamerascan

Für Geräte, die nicht dem Hause Zebra entstammen, bietet die App die Möglichkeit eines Kamerascans an. Im Hintergrund wird dafür die Google Mobile Vision API verwendet, die unter anderem auch Texterkennung oder Gesichtserkennung anbietet [58].

Hierbei wird ein Barcode mittels der Gerätekamera erfasst, ohne dass zuvor ein Bild gemacht werden muss. Dem Benutzer wird eine Preview angezeigt und die Kamera schließt sich, sobald ein Barcode erfasst wurde. Um die Performanz zu maximieren, hat das Diplomarbeitsteam folgende Optimierungen vorgenommen:

- Die API wurde um Blitzfunktionalitäten ergänzt. Dazu wurde eine OpenSource-Variante der Library modifiziert, da die offizielle proprietäre Version keinen Blitz unterstützt. Der Blitz ist über einen `ToggleButton` sofort deaktivierbar oder aktivierbar. Um einen Klick einzusparen, kann der Benutzer über die Einstellungen den Blitz gleich beim Start des Kamerascans aktivieren lassen. Für eine optimale Erkennung darf man den Blitz jedoch nicht direkt in Richtung des Barcode anvisieren, sondern sollte den Blitz entweder höher oder niedriger als den Barcode halten, um optische Reflexionen zu vermeiden.
- Über die Einstellungen kann der Benutzer die Barcodeformate einschränken. Dies führt ebenfalls zur schnelleren Barcodeerkennung und vermeidet zudem noch das Auftreten von `false positives`. Wenn der Benutzer die Kamera nicht auf den gesamten Barcode hält, kann es unter Umständen dazu kommen, dass der abgeschnittene Barcode fälschlicherweise als anderes Format interpretiert wird und der Scan daher einen Barcode liefert, der nicht existiert. Offiziell wird in der vorliegenden Organisation nur das `Code_93`-Format eingesetzt.
- Interne Test haben ergeben, dass ein Aspect Ratio von 16:9 das Schnellste ist. Daher wird die Preview-Größe statisch auf 1920x1080 Pixel festgelegt. Die Preview verwendet jedoch tatsächlich die Größe, die am nähesten zu 1920x1080 ist und vom Gerät unterstützt wird.
- Falls ein Scan erfolgreich ist, wird ein Piepston abgespielt, der als akustisches Feedback fungiert.

5.2.1 Kamerascan: Codeausschnitt

Für die Scanfeatures wird bezüglich der Single-Activity-Architektur eine Ausnahme gemacht. Da diese Screens navigationstechnisch unabhängig sind und im Vollbildmodus gestartet werden, macht es mehr Sinn, sie als Activities anstatt als Fragments zu implementieren. Die Fragments, die einen Kamerascan verwenden, können ihn mit `startActivityForResult` aufrufen. Sie starten also eine neue Activity, um ein Ergebnis zu erhalten:

```
Intent intent = new Intent(getContext(), ScanBarcodeActivity.class);
// 0 ist der Request Code, der die Aktivität identifiziert.
startActivityForResult(intent, 0);
```

Mit der Vision API erstellt man einen `BarcodeDetector` dem ein `Processor` zugewiesen wird. Falls ein Barcode erkannt wird, wird `receiveDetections` automatisch aufgerufen. Die App nimmt sich den ersten erkannten Barcode heraus, setzt ihn als Ergebnis dieser Aktivität und beendet die Aktivität.

```
barcodeDetector.setProcessor(new Detector.Processor<Barcode>() {

     @Override
    public void receiveDetections(
                        Detector.Detections<Barcode> detections) {
            ...
            // Barcode einlesen
            String barcode = barcodeSparseArray.valueAt(0).rawValue;

            //Barcode bekanntgeben
            Intent intent = new Intent();
            intent.putExtra("barcode", barcode);
            setResult(CommonStatusCodes.SUCCESS, intent);
            finish();
            ...
    }
    ...
}
```

Das Ergebnis kann dann wiederum im Fragment wie folgt abgefangen und weiterverarbeitet werden:

```
@Override
public void onActivityResult(int requestCode, int resultCode,
                             @Nullable Intent data) {
    if (requestCode == 0) {
```

```
        if (resultCode == CommonStatusCodes.SUCCESS) {
            ...
            // Barcode einlesen
            String barcode = data.getStringExtra("barcode");
            // Anhand des Barcodes werden dann
            // weitere Aktionen gesetzt
            launchDetailedItemFragmentFromBarcode(barcode);
        }
        ...
}
```

5.3 Die manuelle Eingabe

Gegenstände können durch Scannen, aber auch durch manuelles Klicken auf ihre GUI-Darstellung validiert werden. Es kann durchaus Sinn machen, auf einen Scan zu verzichten, wenn:

- der Scan langsam oder unmöglich ist (z. B. bei Barcodes in unerreichbarer Höhe oder beschädigten Barcodes).
- man einen Gegenstand auch ohne Barcode identifizieren kann.
- man mit einer manuellen Vorgehensweise schneller ist, als mit dem Kamerascan.

5.3.1 Suche

Die Gegenstandsliste, die dem Benutzer angezeigt wird, ist durch eine Suchleiste (`SearchBar`) filterbar. Der Benutzer kann nach Barcode und Gegenstandsbeschreibung filtern. Dadurch ergibt sich auch die Möglichkeit, eine Voice-Tastatur – insofern das Gerät eine hat – einzusetzen.

5.3.2 Textscan

Falls der Benutzer weder Voice noch Tastatur verwenden will, kann er den Textscan verwenden. Hierbei wird per Google Mobile Vision API der aufgedruckte Text auf einem Barcode – jedoch NICHT der Barcode selbst – gescannt [59]. Dem Benutzer wird das aktuelle Scanergebnis laufend angezeigt.

Da der Out-Of-The-Box-Scan für die vorliegenden Zwecke nicht unbedingt geeignet ist, wurden folgende Modifikationen vorgenommen:

- Die API wurde um Blitzfunktionalitäten ergänzt.
- Der Scanner gibt grundsätzlich alles was er sieht wieder. Das Projektteam hat dies mit `Regex`-Ausdrücken kombiniert um sinnvolle Ergebnisse zu erlangen. Es gibt daher verschiedene Texterkennungsmodi:
  - Kein Filter. Hier wird keine Regex verwendet und der Benutzer sieht den ungefilterten Text, den der Scanner erkannt hat.
  - Alphanumerisch.
  - Alphabetisch.
  - Numerisch (Barcodes).
  - IP-Adressen.

Jeder Modus verwendet die Regex, die am besten für seine Kategorie geeignet ist. Die Library bietet nur geringfügige Möglichkeiten an, den Scan selbst zu beeinflussen. Man kann lediglich einen Fokus auf ein Scanergebnis setzen. Daher wird der Fokus auf ein Scanergebnis gesetzt, sobald dieses durch die aktuelle Regex ausgedrückt werden kann. Die API beschränkt sich allerdings nicht auf das Ergebnis, auf das der Fokus gesetzt wurde, sondern kann jederzeit den Fokus selbst wieder aufheben. Unabhängig davon werden alle Zeichen des Ergebnisses entfernt, die nicht durch die Regex erfasst wurden.

Die Erkennung von Barcodes (also der Numerische Modus) hat zusätzlich folgende Optimierungen erhalten:

- Gescannte Ergebnisse werden durch die Regex optimiert und gespeichert. Das Ergebnis, das zuerst dreimal vorkam, wird eingelockt. Das heißt, dass alle restlichen Scans ignoriert werden.
- Zeichen die häufig vom Scanner verwechselt werden (z. B. wird die Ziffer “0” häufig als Buchstabe “O” erkannt), werden durch ihr numerisches Gegenstück ersetzt.
- Längere Zeichenketten erhalten eine höhere Priorität und werden dem Benutzer vorzugsweise angezeigt. Damit werden abgeschnittene Ergebnisse benachteiligt. Falls ein kürzeres Ergebnis jedoch dreimal vorkommt, wird nicht mehr die längste Zeichenkette, sondern die häufigste bevorzugt und dem Benutzer angezeigt.
- Ergebnisse, die weniger als sieben Zeichen enthalten, werden nicht gespeichert und können damit auch nicht eingelockt werden.

Der Benutzer kann das aktuell angezeigte Ergebnis in seine Zwischenablage kopieren und anschließend die Suchleiste nutzen. Die Kommunikation zwischen den Fragments und dem Textscan erfolgt analog zum Kamerascan, da der Textscan aus denselben Gründen als Activity implementiert wurde.

6 Einführung in die Server-Architektur

Die Inventur- sowie Gegenstandsdaten der HTL Rennweg sollen an einem zentralen Ort verwaltet und geführt werden. Der für diesen Zweck entwickelte Server muss also folgende Anforderungen erfüllen:

- eine einfache Datenbankverwaltung und -verbindung
- das Führen einer Historie aller Zustände der Inventar-Gegenstände
- eine Grundlage für eine Web-Administrationsoberfläche
- die Möglichkeit für Datenimport und -export, etwa als *.xlsx* Datei
- eine Grundlage für die Kommunikation mit der Client-Applikation
- hohe Stabilität und Verfügbarkeit

Angesichts der Programmiersprachen, auf die das Projektteam spezialisiert ist, stehen als Backend-Lösung vier *Frameworks* zur öffentlichen Verfügung, zwischen denen gewählt wurde:

- Django [64]
- Pyramid [71]
- Web2Py [72]
- Flask [69]

Alle genannten Alternativen sind *Frameworks* in der Programmiersprache Python. Sie wurden anhand der Faktoren Funktionsumfang, verfügbarer Dokumentation und Anzahl aktiver EntwicklerInnen verglichen. Django übertrifft alle anderen Alternativen in jedem dieser Punkte und gilt daher als der Standard eines solchen *Frameworks*. Die zu Django ähnlichste Alternative bezüglich Funktionsumfang ist Pyramid. Die komplexere Integration von kundenspezifischen Funktionen, der geringere Umfang öffentlich verfügbarer Dokumentation und die wesentlich geringere Anzahl aktiver Entwickler sind klare Nachteile gegenüber Django. Web2Py kann mittlerweile als veraltet eingestuft werden. Die Dokumentation entspricht nicht dem aktuellen Stand der Technik und die Entwicklerbasis ist größtenteils auf andere *Frameworks* umgestiegen. Flask fällt unter die Kategorie der Microframeworks und besitzt daher nur einen sehr geringen Funktionsumfang, der entweder durch Einsatz von Fremdsoftwarepaketen oder eigene Implementierung auf das benötigte Niveau gehoben werden muss. Dadurch wird

eine hohe Anpassbarkeit und hohe Performanz versprochen, jedoch ist das Endresultat meist in Django effizienter implementierbar.

Aufgrund des vorliegenden Vergleichs und einer bestehenden und frei verfügbaren Inventarverwaltungsplattform "Ralph" [73], die auf Django aufbaut, wurde die Alternative Django gewählt.

6.1 Begründung der Wahl von Django und Ralph

Django bietet eine weit verbreitete Open-Source Lösung für die Entwicklung von Web-Diensten. Bekannte Webseiten, die auf Django basieren sind u. a. Instagram, Mozilla, Pinterest und Open Stack [6]. Django zeichnet sich besonders durch die sog. *"Batteries included"* Mentalität aus. Das heißt, dass Django bereits die gängigsten *Features* eines Webserver-Backends standardmäßig innehat. Diese sind (im Vergleich zu Alternativen wie etwa Flask) u. a.:

- Authentifikation und Autorisierung, sowie eine damit verbundene Benutzerverwaltung
- Schutz vor gängigen Attacken (wie *SQL-Injections* oder *CSRF* [74]), siehe Kapitel 6.2.5, Seite 51.

Zusätzlich bietet Ralph bereits einige *Features*, die die grundlegende Führung und Verwaltung eines herkömmlichen Inventars unterstützen (beispielsweise eine Suchfunktion mit automatischer Textvervollständigung). Ralph ist daher eine angemessene Basis für die vorliegende Diplomarbeit.

6.2 Kurzfassung der Funktionsweise von Django und Ralph

6.2.1 Einleitung

Django ist ein in der Programmiersprache Python geschriebenes Webserver-*Framework*. Ralph ist eine auf dem Django-*Framework* basierende *DCIM* und *CMDB* Softwarelösung. Haupteinsatzgebiet dieser Software sind vor allem Rechenzentren mit hoher Komplexität, die externe Verwaltungsplattformen benötigen. Zusätzlich können aber auch herkömmliche Inventardaten von IT-spezifischen Gegenständen in die Ralph-Plattform aufgenommen werden.

Ralph wurde von der polnischen Softwarefirma "Allegro" entwickelt und ist unter der Apache 2.0 Lizenz öffentlich verfügbar. Dies ermöglicht auch Veränderungen und Erweiterungen. Zu Demonstrationszwecken bietet Allegro eine öffentlich nutzbare Demo-Version [16] von Ralph an.

Die vorliegende Diplomarbeit bietet eine Erweiterung des Ralph-Systems.

In diesem Kapitel wird die Funktionsweise des Django-*Frameworks*, sowie Ralph beschrieben. Ziel dieses Kapitels ist es, eine Wissensbasis für die darauffolgenden Kapitel zu schaffen.

6.2.2 Datenbank-Verbindung, Pakete und Tabellen-Definition

Folgende Datenbank-Typen werden von Django unterstützt:

- PostgreSQL
- MariaDB
- MySQL
- Oracle
- SQLite

Die Konfiguration der Datenbank-Verbindung geschieht unter Standard-Django in der Datei `settings.py` und unter Ralph in der jeweiligen Datei im Verzeichnis `settings`. Eine detaillierte Anleitung zur Verbindung mit einer Datenbank ist in der offiziellen Django-Dokumentation [20] zu finden.

Die verschiedenen Funktionsbereiche des Servers sind in Pakete bzw. Module gegliedert. Jedes Paket ist ein Ordner, der verschiedene Dateien und Unterordner beinhalten kann. Die Dateinamen-Nomenklatur eines Pakets ist normiert [19]. Der Name eines Pakets wird fortan "App-Label" genannt. Standardmäßig ist dieser Name erster Bestandteil einer *URL* zu einer beliebigen graphischen Administrationsoberfläche des Pakets. Pakete werden durch einen Eintrag in die Variable `INSTALLED_APPS` innerhalb der o. a. Einstellungsdatei registriert. Beispiele sind die beiden durch die vorliegende Diplomarbeit registrierten Pakete `ralph.capentory` und `ralph.stocktaking`

Ist ein Python-Paket erfolgreich registriert, können in der Datei `models.py` Datenbank-Tabellen als Python Klassen[1] definiert werden. Diese Klassen werden fortan als "Modell" bezeichnet. Tabellenattribute werden als Attribute dieser Klassen definiert und sind jeweils Instanzen der Klasse `Field`[21][2]. Datenbankeinträge können demnach als Instanzen der Modellklassen betrachtet und behandelt werden. Standardmäßig besitzt jedes Modell ein Attribut `id`, welches als *primärer Schlüssel* dient. Der Wert des `id`

[1] erbend von der Superklasse `Model`[21]
[2] oder davon erbende Klassen

Attributs ist unter allen Instanzen eines Modells einzigartig. Die Anpassung dieses Attributs wird in der offiziellen Django-Dokumentation genauer behandelt [21].

Jedes Modell benötigt eine innere Klasse `Meta`. Sie beschreibt die *Metadaten* der Modellklasse. Dazu gehört vor allem der von Benutzern lesbare Name des Modells `verbose_name` [24].

6.2.3 Administration über das Webinterface

Um die Administration von Modelldaten über das Webinterface des Servers zu ermöglichen, werden grundsätzlich zwei Ansichten der Daten benötigt: Eine Listenansicht aller Datensätze und eine Detailansicht einzelner Datensätze.

Die Listenansicht aller Datensätze eines Modells wird in der Datei `admin.py` als *Subklasse* von `ModelAdmin`[3] definiert. Attribute dieser Klasse beeinflussen das Aussehen und die Funktionsweise der Weboberfläche. Durch das Setzen von `list_display` werden beispielsweise die in der Liste anzuzeigenden Attribute definiert.

Die Detailansicht einzelner Datensätze wird grundsätzlich durch die `ModelAdmin` Klasse automatisch generiert, kann aber durch Setzen dessen `form` Attributs auf eine eigens definierte *Subklasse* von `ModelForm`[4] angepasst werden. Diese Klassen werden in der Datei `forms.py` definiert und besitzen, ähnlich der `Model` Klasse, auch eine innere Klasse `Meta`.

Um die `ModelAdmin` *Subklassen* über eine *URL* erreichbar zu machen, müssen diese registriert werden. Dies geschieht durch den `register` *Dekorator*. Dieser Dekorator akzeptiert die zu registrierende Modellklasse, die zu dem `ModelAdmin` gehört, als Parameter. Die Listenansicht einer registrierten `ModelAdmin` Subklasse ist standardmäßig unter der *URL*

`/<App-Label>/<Modell-Name>/`

erreichbar und die Detailansicht einer Modellinstanz unter der *URL*

`/<App-Label>/<Modell-Name>/<Modellinstanz-ID>/` .

Die Dokumentation der Administrationsfeatures von Django ist auf der offiziellen Dokumentationswebseite von Django [17] zu finden.

[3] Unter Ralph steht hierfür die Klasse `RalphAdmin` zur Verfügung [65].
[4] Unter Ralph steht hierfür die Klasse `RalphAdminForm` zur Verfügung.

6.2.4 API und DRF

Um Daten außerhalb der graphischen Administrationsoberfläche zu bearbeiten, wird eine *API* benötigt. Eine besondere und weit verbreitete Form einer API ist eine *REST-API* [49], die unter Django durch das integrierte *DRF* implementiert wird [70]. API Definitionen werden unter Django in einem Paket in der Datei `api.py` getätigt.

Um den API-Zugriff auf ein Modell zu ermöglichen werden üblicherweise eine `APIView`[5] *Subklasse* und eine `Serializer`[6] *Subklasse* definiert. `APIView` Klassen sind zuständig für das Abarbeiten von Anfragen mithilfe einer `Serializer` Klasse, die die Daten aus der Datenbank repräsentiert und in das gewünschte Format konvertiert. Durch `APIView` Klassen werden Berechtigungen und sonstige Attribute definiert, die sich auf das wahrgenommene Erscheinungsbild des Servers auf einen Client auswirken. Beispiel dafür ist die Art der *Paginierung* [70]. Die erstellten `APIView` Klassen können dann mithilfe einer `Router`[7] Instanz registriert werden. Anleitungen zur Erstellung dieser API-Klassen sind auf der offiziellen Webseite des DRF [70] und der offiziellen Ralph-Dokumentationsseite [66] zu finden.

6.2.5 Views

Schnittstellen, die keiner der beiden o. a. Kategorien zugeordnet werden können, werden in der Datei `views.py` definiert. Bei diesen *generischen* Schnittstellen handelt es sich entweder um *Subklassen* der Klasse `View`[8] [22] oder vereinzelte Methoden mit einem `request`[9] Parameter [31]. Diese Schnittstellen werden fortan Views genannt.

Soll ein View als Antwort auf eine Anfrage HTML-Daten liefern, so sollte dazu ein *Template* verwendet werden. Mithilfe von Templates können Daten, die etwa durch Datenbankabfrage entstehen, zu einer HTML Antwort aufbereitet werden. Besonders ist hierbei die zusätzlich zu HTML verfügbare Django-Template-*Syntax* [29]. Damit können HTML Elemente auf den Input-Daten basierend dynamisch generiert werden. So stehen beispielsweise `if` Statements direkt in der Definition des Templates zur Verfügung. Die Benutzung von Templates schützt standardmäßig gegen Attacken, wie *SQL-Injections* oder *CSRF* [74] und gilt daher als besonders sicher. Durch das Diplomarbeitsteam wurden weitere Möglichkeiten zur Sicherung des Serversystems [27] implementiert und alle Sicherheitsempfehlungen der Entwickler von Django [27] eingehalten.

[5] Unter Ralph steht hierfür die Klasse `RalphAPISerializer` zur Verfügung [66].
[6] Unter Ralph steht hierfür die Klasse `RalphAPIViewSet` zur Verfügung [66].
[7] Unter Ralph steht hierfür die globale `RalphRouter` Instanz `router` zur Verfügung [66].
[8] die ebenfalls Superklasse der Klasse `ApiView` ist
[9] zu Deutch: Anfrage; entspricht den empfangenen Daten

Da reguläre Views nicht automatisch registriert werden, müssen sie manuell bekanntgegeben werden. Dies geschieht durch einen Eintrag in die Variable `urlpatterns` in der Datei `urls.py` [30].

6.2.6 Datenbankabfragen

Datenbankabfragen werden in Django durch `Queryset`-Objekte getätigt. Das Definieren eines `Queryset`-Objekts löst nicht sofort eine Datenbankabfrage aus. Erst, wenn Werte aus einem `Queryset`-Objekt gelesen werden, wird eine Datenbankabfrage ausgelöst. So kann ein `Queryset`-Objekt beliebig oft verändert werden, bevor davon ausgelesen wird. Ein Beispiel hierfür ist das Anwenden der `filter()` Methode.

In dem folgenden Code-Auszug[10] werden aus dem Modell `Entry` bestimmte Einträge gefiltert:

```
# Erstelle ein Queryset aller Entry-Objekte,
# dessen Attribut "headline" mit "What" beginnt.
q = Entry.objects.filter(headline__startswith="What")

# Filtere aus dem erstellten Queryset alle Entry-Objekte,
# dessen Attribut "pub_date" kleiner oder gleich
# dem aktuellen Datum ist.
q = q.filter(pub_date__lte=datetime.date.today())

# Schließe aus dem erstellten Queryset alle Entry-Objekte,
# dessen Attribut "body_text" den Text "food" beinhaltet, aus.
q = q.exclude(body_text__icontains="food")

# Ausgabe des erstellten Querysets.
# Erst hier kommt es zu der ersten Datenbankabfrage!
print(q)
```

Weitere Beispiele und Methoden sind der offiziellen Django-Dokumentation zu entnehmen [25].

6.3 Designgrundlagen

Designgrundlagen für Django-Entwickler sind auf der offiziellen Dokumentationsseite von Django [63] abrufbar. Die Erweiterung von Django durch die vorliegende Diplom-

[10] entnommen aus der offiziellen Django Dokumentation [25]

arbeit wurde anhand dieser Grundlagen entwickelt.

Das Konzept des `Mixin`s wird von der Ralph-Plattform besonders häufig genutzt. `Mixin`s sind Klassen, die anderen von ihnen erbenden Klassen bestimmte Attribute und Methoden hinzufügen. Manche `Mixin`s setzen implizit voraus, dass die davon erbenden Klassen ebenfalls von bestimmten anderen Klassen erben. Beispiel ist die Klasse `AdminAbsoluteUrlMixin`, die eine Methode `get_absolute_url` zur Verfügung stellt. Diese Methode liefert die *URL*, die zu der Detailansicht der Modellinstanz führt, die die Methode aufruft. Voraussetzung für das Erben einer Klasse von `AdminAbsoluteUrlMixin` ist daher, dass sie ebenfalls von der Klasse `Model` erbt.

7 Die zwei Erweiterungsmodule des Serversystems

Die vorliegende Diplomarbeit erweitert das Ralph-System um 2 Module. Dabei handelt es sich um die beiden Pakete “Capentory” und “Stocktaking”. Das Paket “Capentory” behandelt die Führung der Inventardaten und wurde speziell an die Inventardaten der HTL Rennweg angepasst. Das Paket “Stocktaking” ermöglicht die Verwaltung der durch die mobile Applikation durchgeführten Inventuren. Dazu zählen Aufgaben wie das Erstellen der Inventuren, das Einsehen von Inventurberichten oder das Anwenden der aufgetretenen Änderungen.

Dieses Kapitel beschreibt die grundlegende Funktionsweise der beiden Module. Eine Anleitung zur Bedienung der *Weboberfläche* ist dem Handbuch zum Server (siehe Kapitel A, Seite 105) zu entnehmen.

7.1 Das Capentory-Modul

Das Capentory-Modul beherbergt drei wichtige Modelle:

1. HTLItem
2. HTLRoom
3. HTLItemType

Die wichtigsten Eigenschaften der Modelle und damit verbundenen Funktionsweisen werden in diesem Unterkapitel beschrieben.

7.1.1 Das HTLItem Modell

Das `HTLItem`-Modell repräsentiert die Gegenstandsdaten des Inventars der HTL Rennweg. Es sind typische Merkmale aus dem *SAP ERP* System vertreten. Die Attribute `anlage`, `asset_subnumber` und `company_code` werden direkt aus dem *SAP ERP* System übernommen.

7.1.1.1 Die wichtigsten Attribute

Zu den wichtigsten Attributen des `HTLItem` Modells zählen:

- `anlage` und `asset_subnumber`: Diese Attribute bilden den Barcode eines Gegenstandes.
- `barcode_prio`: Wenn dieses Attribut gesetzt ist, überschreibt es den durch die Attribute `anlage` und `asset_subnumber` entstandenen Barcode.
- `anlagenbeschreibung`: Dieses Attribut repräsentiert die aus dem *SAP ERP* System entnommene Gegenstandsbeschreibung und kann nur durch den Import von Daten direkt aus dem *SAP ERP* System geändert werden.
- `anlagenbeschreibung_prio`: Dieses Attribut dient als interne Gegenstandsbeschreibung, die auch ohne einen Import aus dem *SAP ERP* System geändert werden kann.
- `room`: Dieses Attribut referenziert auf ein `HTLRoom` Objekt, in dem sich ein `HTLItem` Objekt befindet.
- `is_in_sap`: Der Wert dieses *Boolean*-Attributs ist `Wahr`, wenn der `HTLItem`-Datensatz aus dem *SAP ERP* System importiert wurde. Umgekehrt ist der Wert dieses Attributes `Falsch`, wenn der `HTLItem`-Datensatz aus einer anderen Quelle entstanden ist. Ein manuell hinzugefügter `HTLItem`-Datensatz hat für dieses Attribut den Wert `Falsch`.
- `item_type`: Dieses Attribut referenziert auf das `HTLItemType` Objekt, das einem `HTLItem` Objekts zugeordnet ist. Es repräsentiert die Gegenstandskategorie eines `HTLItem` Objekts.

7.1.1.2 Einzigartigkeit von HTLItem Objekten

Bezüglich der Einzigartigkeit von `HTLItem` Objekten gelten einige Bestimmungen.

Sind die Attribute `anlage`, `asset_subnumber` und `company_code` je mit einem nichtleeren Wert befüllt, so repräsentieren sie ein `HTLItem` Objekt eindeutig. Es dürfen keine 2 `HTLItem` Objekte denselben Wert dieser Attribute haben. Um diese Bedingung erfüllen zu können, muss eine eigens angepasste Validierungslogik implementiert werden. Standardverfahren wäre in diesem Anwendungsfall, die Metavariable `unique_together` [24] anzupassen:

```
unique_together = [["anlage",  "asset_subnumber", "company_code"]]
```

Dieses Verfahren erfüllt die geforderte Bedingung nur in der Theorie. Praktisch werden leere Werte von Attributen dieser Art nicht als `None` (Python) bzw. `null` (MySQL), sondern als Leerstrings `""` gespeichert. Um diese Werte ebenfalls von der Regel auszuschließen, muss die `validate_unique()` Methode [23] überschrieben werden. Die Methode wird unter Normalzuständen immer vor dem Speichern eines Objekts des

Modells aufgerufen wird. Wirft sie einen Fehler auf, wird das Objekt nicht gespeichert und der Benutzer erhält eine entsprechende Fehlermeldung.

Ist das `barcode_prio` Attribut eines `HTLItem` Objekts gesetzt, darf dessen Wert nicht mit jenem eines anderen `HTLItem` Objekts übereinstimmen. Standardverfahren wäre in diesem Anwendungsfall das Setzen des `unique` Parameters des Attributes auf den Wert `True`. Da dieses Verfahren ebenfalls das o. a. Problem aufwirft, muss die Logik stattdessen in die `validate_unique()` Methode aufgenommen werden. Zusätzlich darf der Wert des `barcode_prio` Attributs nicht mit dem aus den beiden Attributen `anlage` und `asset_subnumber` generierten Barcode übereinstimmen. Um diese Bedingung zu erfüllen kann nur die `validate_unique()` Methode herbeigezogen werden.

7.1.1.3 Änderungsverlauf

Besonders für das `HTLItem` Modell ist es von besonderer Wichtigkeit, ein Objekt auf den Zustand vor einer unbeabsichtigten Änderung zurücksetzen und gelöschte Objekte wiederherstellen zu können. Durch das bereits in Ralph inkludierte Paket `django-reversion` können die aktuellen Zustände von Datenbankobjekten gesichert werden, um später darauf zugreifen zu können [35]. Das Paket bietet die Funktion, der graphischen Administrationsoberfläche entsprechende Funktionen zur Wiederherstellung oder Zurücksetzung einzelner Objekte hinzuzufügen.

Um einen Gegenstand zu entinventarisieren, kann er gelöscht werden. Referenzen auf den gelöschten Gegenstand, die durch eine Inventur entstehen, bleiben in einem ungültigen Zustand erhalten. Der gelöschte Gegenstand kann zu einem späteren Zeitpunkt vollständig wiederhergestellt werden. Die Referenzen auf den wiederhergestellten Gegenstand werden damit wieder gültig.

7.1.1.4 Speichern von Bildern und Anhängen

Die in Ralph verfügbare Klasse `AttachmentsMixin` wird verwendet, um Instanzen der Modellklasse `HTLItem` über dessen graphische Administrationsoberfläche diverse Anhänge zuzuordnen. Ein Anhang ist eine ordinäre Datei, die auf den Server hochgeladen werden kann. Alle hochgeladenen Dateien werden vor dem Speichern anhand deren *Hash-Werten* verglichen. Ist eine Datei mit dem *Hash-Wert* einer hochgeladenen Datei bereits vorhanden, wird die Datei nicht erneut gespeichert und ein Verweis auf die vorhandene Datei wird erstellt.

7.1.2 Das HTLRoom Modell

Das `HTLRoom`-Modell repräsentiert die Raumdaten der HTL Rennweg. Es sind typische Merkmale aus dem *SAP ERP* System vertreten. Die Attribute `room_number`, `main_inv` und `location` werden direkt aus dem *SAP ERP* System übernommen.

7.1.2.1 Die wichtigsten Attribute

Zu den wichtigsten Attributen des `HTLRoom` Modells zählen:

- `room_number`: Dieses Attribut bildet den Barcode eines Raumes.
- `barcode_override`: Wenn dieses Attribut gesetzt ist, überschreibt es den durch das Attribut `room_number` gebildeten Barcode.
- `internal_room_number`: Dieses Attribut repräsentiert die schulinterne Raumnummer und ist somit von den Daten aus dem *SAP ERP* System unabhängig.
- `is_in_sap`: Der Wert dieses *Boolean*-Attributs ist `Wahr`, wenn der `HTLRoom`-Datensatz einen aus dem *SAP ERP* System vorhandenen Raum repräsentiert.
- `children`: Mit dieser Beziehung können einem übergeordneten `HTLRoom` Objekt mehrere `HTLRoom` Objekte untergeordnet werden. Anwendungsfall für dieses Attribut ist die Definition von Schränken oder Serverracks, die je einem übergeordneten Raum zugeteilt sind, selbst aber eigenständige Räume repräsentieren.
- `type`: Dieses Attribut spezifiziert die Art eines `HTLRoom` Objekts. So kann ein `HTLRoom` Objekt einen ganzen Raum oder auch nur einen Kasten in einem übergeordneten Raum repräsentieren.
- `item`: Dieses Attribut kann gesetzt werden, um ein `HTLRoom` Objekt durch ein `HTLItem` Objekt zu repräsentieren. Anwendungsbeispiel ist ein Schrank, der sowohl als `HTLRoom` Objekt als auch als `HTLItem` Objekt definiert ist. Sind die beiden Objekte durch das `item` Attribut verbunden, ist der Barcode des `HTLRoom` Objekts automatisch jener des `HTLItem` Objekts.

7.1.2.2 Einzigartigkeit von HTLRoom Objekten

Bezüglich der Einzigartigkeit von `HTLRoom` Objekten gelten ähnliche Bestimmungen wie zu jener von `HTLItem` Objekten.

Die Attribute `room_number`, `main_inv` und `location` sind gemeinsam einzigartig. Leere Werte sind von dieser Regel ausgeschlossen. Gleichzeitig darf der Wert des `barcode_override` Attribut nicht mit dem Wert des `room_number` Attributes eines anderen `HTLRoom` Objekts übereinstimmen und vice versa. Beide Bedingungen müssen wie im Falle des `HTLItem` Modells durch Überschreiben der `validate_unique()` Methode [23] implementiert werden. Die Methode wird unter Normalzuständen immer

vor dem Speichern eines Objekts des Modells aufgerufen wird. Wirft sie einen Fehler auf, wird das Objekt nicht gespeichert und der Benutzer erhält eine entsprechende Fehlermeldung.

7.1.2.3 Subräume

Wie im Abschnitt "Die wichtigsten Attribute" festgehalten, können einem `HTLRoom` Objekt mehrere `HTLRoom` Objekte untergeordnet werden. Diese untergeordneten Räume werden "Subräume" genannt. Bei "Subräumen" handelt es sich beispielsweise um einen Kasten, der als eigenständiger Raum in einem ihm übergeordneten Raum steht. Logisch betrachtet ist der Kasten auch nur ein Raum, in dem sich Gegenstände befinden. Ob der Raum ein Klassenraum oder ein Kasten in einem Klassenraum ist, hat keine logischen Auswirkungen auf seine Eigenschaften als "Standort von Gegenständen".

Um eine valide Hierarchie beizubehalten, muss diese Beziehung bei jedem Speicherprozess eines `HTLRoom` Objekts überprüft werden. Das geschieht durch die Methode `clean_children()`, die beim Speichern durch die graphische Administrationsoberfläche automatisch aufgerufen wird. Bei Speichervorgängen, die nicht direkt durch die Administrationsoberfläche initiiert werden [1], muss `clean_children()` manuell aufgerufen werden.

7.1.3 Das HTLItemType Modell

Das `HTLItemType` Modell repräsentiert Kategorien von Gegenständen. Das Modell besteht aus 2 Eigenschaften. Die Eigenschaft `item_type` beschreibt ein `HTLItemType` kurz, die Eigenschaft `description` bietet Platz für Kommentare.

Durch das Setzen eines `HTLItemType` Objekts für ein `HTLItem` Objekt durch seine Eigenschaft `item_type` werden dem `HTLItem` Objekt alle *Custom-Fields* des `HTLItemType` Objekts zugewiesen. Anwendungsbeispiel ist das Setzen eines *Custom-Fields* namens "Anzahl Ports" für den `HTLItemType` namens "Switch". Jedes `HTLItem` Objekt mit dem `item_type` "Switch" hat nun ein *Custom-Field* namens "Anzahl Ports".

Das für *Custom-Fields* erforderliche Mixin (siehe Kapitel 6.3, Seite 52) `WithCustomFieldsMixin` bietet die Funktionalität der `custom_fields_inheritance`. Sie ermöglicht das Erben von allen *Custom-Field* Werten eines bestimmten Objekts an ein anderes. Diese Funktion macht sich das `HTLItem` Modell zunutze. Um beim Speichern automatisch vom `HTLItemType` Objekt unabhängige *Custom-Field* Werte zu erstellen, die sofort vom Benutzer bearbeitet werden können, muss der Speicherlogik eine Funktion hinzugefügt werden. Dazu wird eine `@receiver` Funktion genutzt, die

[1] etwa das automatisierte Speichern beim Datenimport

automatisch bei jedem Speichervorgang eines `HTLItemType` oder `HTLItem` Objekts aufgerufen wird:

```
@receiver(post_save, sender=HTLItemType)
def populate_htlitem_custom_field_values(sender, instance, **kwargs):
    populate_inheritants_custom_field_values(instance)

@receiver(post_save, sender=HTLItem)
def populate_htlitemtype_custom_field_values(sender, instance, **kwargs):
    populate_with_parents_custom_field_values(instance)
```

Die beiden angeführten Funktionen werden je beim Aufkommen eines `post_save` Signals [28] ausgeführt, dessen `sender` ein `HTLItemType` oder `HTLItem` ist. Die Funktionen rufen jeweils eine weitere Funktion auf, welche die *Custom-Fields* entsprechend aggregiert.

7.1.4 Datenimport

Um die Gegenstands- und Raumdaten des Inventars der HTL Rennweg in das erstellte System importieren zu können, muss dessen standardmäßig verfügbare Importfunktion entsprechend erweitert werden. Dazu sind 4 spezielle Importverhalten notwendig. Die Datenquellen sind das *SAP ERP* System der HTL Rennweg (siehe Kapitel 7.1.4.1, Seite 62) und und intern von Lehrkräften und Inventarverantwortlichen geführte Listen (siehe Kapitel 7.1.4.2, Seite 62).

Implementiert wird das Importverhalten nicht innerhalb der entsprechenden Modellklasse, sondern in deren verknüpften `ModelAdmin` Klasse. Durch das Attribut `resource_class` wird spezifiziert, mithilfe welcher Python-Klasse die Daten importiert werden. Um für ein einziges Modell mehrere `resource_class` Einträge zu setzen, müssen mehrere `ModelAdmin` Klassen für Proxy-Modelle [21] des eigentlichen Modells definiert werden. Ein Proxy-Modell eines Modells verweist auf dieselbe Tabelle in der Datenbank, kann aber programmiertechnisch als unabhängiges Modell betrachtet werden. Die Daten, die durch das Proxy-Modell ausgelesen oder eingefügt werden, entsprechen exakt jenen des eigentlichen Modells.

```
# Definition eines Proxy-Modells zu dem Modell "HTLItem"
class HTLItemSecondaryImportProxy(HTLItem):
    class Meta:
        proxy = True

# Diese Datenbankabfragen liefern beide dasselbe Ergebnis:
print(HTLItem.objects.all())
print(HTLItemSecondaryImportProxy.objects.all())
```

```
# Für das Proxy-Modell kann eine ModelAdmin-Klasse definiert werden.
# Diese bekommt eine eigene "resource_class".
@register(HTLItemSecondaryImportProxy)
class HTLItemSecondaryImportProxyAdmin(HTLItemSecondaryImportMixin, RalphAdmin):
    resource_class = HTLItemSecodaryResource
```

Das Importverhalten wird in der Klasse, die in das Attribut `resource_class` eingetragen wird, programmiertechnisch festgelegt. Es werden alle Zeilen der zu importierenden Datei nacheinander abgearbeitet. Bei sehr großen Datenmengen oder aufwändigem Importverhalten kann es zu Performanceverlusten kommen. Da nahezu jedes durch das Diplomarbeitsteam erstellte Importverhalten die zu importierenden Daten überprüfen oder anderweitig speziell behandeln muss, kann es hier besonders zu Performanceengpässen kommen. Beispielsweise muss beim Import von Daten aus dem *SAP ERP* System auch geprüft werden, ob der Raum eines Gegenstandes existiert. Die oft sehr großen Datenmengen, die aus dem *SAP ERP* System importiert werden müssen, sorgen ebenfalls für Performanceverluste.

Die in den zu importierenden Dateien vorhandenen Überschriften werden vor dem Importprozess auf Modelleigenschaften abgebildet. Manche Werte der zu importierenden Datensätze müssen in Werte gewandelt werden, die in der Datenbank gespeichert werden können. In der Datei `import_settings.py` sind diese Abbildungen bzw. Umwandlungen als "*Aliases* " deklariert. In dem folgenden Beispiel werden die Überschriften "Erstes Attribut" und "Zweites Attribut" auf die zwei Modelleigenschaften `field_1` und `field_2` abgebildet. Importierte Werte für "Zweites Attribut" werden von den Zeichenketten "Ja" und "Nein" auf die *Boolean*-Werte `True` und `False` übersetzt.

```
# Beispielhafte Definition von Aliases für einen Import
ALIASES_HTLITEM = {
    "field_1": (["Erstes Attribut"], {

    }),
    "field_2": (["Zweites Attribut"], {
      "Ja": True,
      "Nein": False
    }),
```

Der Datenimport wird immer zweimal durchlaufen. Zuerst werden die importierten Daten zwar generiert, aber nicht gespeichert und dem Benutzer nur zur Validierung vorgelegt. Nach einer Bestätigung des Benutzers werden die Daten ein weiteres Mal von Neuem generiert und gespeichert.

Details über das Format einer zu importierenden Quelldatei sind dem Handbuch zum Server (siehe Kapitel A, Seite 105) zu entnehmen.

7.1.4.1 Import aus dem SAP ERP System

Die Daten aus dem *SAP ERP* System der Schule können in ein gängiges Datenformat exportiert werden. Das üblich gewählte Format ist eine Excel-Tabelle (Dateiendung "`.xlsx`").

Die aus dem *SAP ERP* System exportierten Daten haben grundsätzlich immer volle Gültigkeit. Bereits im "Capentory" System vorhandene Daten werden überschrieben.

Zu importierende Datensätze werden anhand der Werte "BuKr", "Anlage" und "UNr." aus der Quelldatei verglichen. Diese Werte werden auf die `HTLItem` Modellattribute `company_code`, `anlage` und `asset_subnumber` abgebildet. Zusätzlich wird auch das `barcode_prio` Attribut mit dem zusammengefügten Wert der "Anlage" und "UNr." Felder der Quelldatei verglichen. Diese Felder repräsentieren den Barcode eines `HTLItem` Objekts. Stimmt ein `HTLItem` Datensatz aus dem "Capentory" System mit einem zu importierenden Datensatz aufgrund einer der beiden verglichenen Wertepaare überein, wird dieser damit assoziiert und überschrieben.

Vor dem Verarbeiten der Daten des `HTLItem` Objekts werden die Daten des zugehörigen `HTLRoom` Objekts verarbeitet. Es wird nach einem existierenden `HTLRoom` Objekt mit einem übereinstimmenden `room_number` Attribut gesucht. Das "Raum" Feld der Quelldatei wird auf das `room_number` Attribut abgebildet. Sollten mehrere `HTLRoom` Objekte übereinstimmen[2], wird jener mit übereinstimmenden `main_inv` und `location` Attributen ausgewählt. Die "Hauptinven" und "Standort" Felder der Quelldatei werden auf die `main_inv` und `location` Attribute abgebildet. Ein gefundenes `HTLRoom` Objekt wird mit dem importierten `HTLItem` Objekt verknüpft. Sollte kein `HTLRoom` Objekt gefunden werden, wird es mit den entsprechenden Werten erstellt. In beiden Fällen wird das `is_in_sap` *Boolean*-Attribut des `HTLRoom` Objekts gesetzt.

Es gibt eine Ausnahme der absoluten Gültigkeit der Daten aus dem *SAP ERP* System. Stimmt das gefundene `HTLRoom` Objekt nicht mit jenem aktuell verknüpften `HTLRoom` Objekt eines `HTLItem` Objekts überein, wird es grundsätzlich aktualisiert. Sollte das aktuell verknüpfte `HTLRoom` Objekt ein "Subraum" des gefundenen Objekts sein, wird dieses nicht aktualisiert. So wird verhindert, dass Gegenstände durch den Import aus einem Subraum in den übergeordneten Raum wandern. In dem *SAP ERP* System existieren die "Subräume" grundsätzlich nicht.

7.1.4.2 Import aus sekundärer und tertiärer Quelle

Bei der sekundären und tertiären Quelle handelt es sich um schulinterne Inventarlisten. Beide Quellen enthalten Informationen über den Raum, in dem sich ein bestimmter Gegenstand befindet. Die sekundäre Quelle enthält Informationen über die Kategorie

[2] Dieser Fall sollte bei einem einzigen Schulstandort nicht auftreten.

eines EDV-spezifischen Gegenstands. Die tertiäre Quelle enthält Informationen über "Subräume" und welche Gegenstände sich darin befinden.

Beide Quellen besitzen allerdings keine absolute Gültigkeit wie der Import aus dem *SAP ERP* System. Aus diesem Grund werden alle Änderungen von Eigenschaften eines `HTLItem` Objekts in Änderungsvorschläge einer Inventur ausgelagert und nicht direkt angewendet. Die Änderungen können zu einem späteren Zeitpunkt eingesehen, bearbeitet und schlussendlich angewendet werden. Grund für dieses spezielle Importverhalten ist die Vertrauenswürdigkeit der Informationen, die der sekundären bzw. tertiären Quelle entnommen werden. Die Listen haben offiziell kein einheitliches Format und können daher bei sofortigem Übernehmen der Änderungen zu ungewollten Fehlinformationen führen. Der Import einer sekundären oder tertiären Quelle kann wie eine eigenständige Inventur angesehen werden. Es können durch den Import auch neue `HTLItem` Objekte hinzugefügt werden, wenn ein Datensatz mit keinem bestehenden Objekt assoziiert werden kann.

Das Importverhalten für die sekundäre Quelle erstellt zusätzlich durch die Methode `get_or_create_item_type()` der Klasse `HTLItemSecodaryResource` definierte `HTLItemType` Objekte. Die erstellten `HTLItemType` Objekte werden den `HTLItem` Objekten indirekt über Änderungsvorschläge zugewiesen.

Die bereits erwähnten Informationen über "Subräume" der tertiären Quelle werden sofort auf die entsprechenden `HTLRoom` Objekte angewendet. Es bedarf keiner weiteren Bestätigung, um die "Subräume" zu erstellen und den übergeordneten `HTLRoom` Objekten zuzuweisen.

7.1.4.3 Import der Raumliste

Die Importfunktion der Raumliste dient zur Verlinkung von interner Raumnummer (`HTLRoom`-Attribut `internal_room_number`) und der Raumnummer im *SAP ERP* System (`HTLRoom`-Attribut `room_number`). Es werden dadurch bestehenden `HTLRoom` Objekten eine interne Raumnummer und eine Beschreibung zugewiesen, oder gänzlich neue `HTLRoom` Objekte anhand aller erhaltenen Informationen erstellt. Der Import geschieht direkt, ohne Auslagerung von Änderungen in Änderungsvorschläge.

7.1.5 Datenexport

Die Daten aller `HTLItem` Gegenstände, die aus dem *SAP ERP* System importiert wurden können unter spezieller Verarbeitung exportiert werden. Die exportierten Daten können in das *SAP ERP* System importiert werden und beinhalten u. a. Raumänderungen, die durch Inventuren aufgetreten sind. Bei der Verarbeitung der zu exportierenden Daten wird eine Funktion des `reversion` Pakets [35] genutzt. Diese Funktion besteht

darin, den Zustand eines Gegenstandes zum Zeitpunkt des letzten Imports aus dem *SAP ERP* System abzufragen. Es wird der Zustand zu dem angegebenen Zeitpunkt mit dem aktuellen Zustand verglichen. Anhand der erkannten Änderungen wird die Export-Datei erstellt. Weitere Informationen zum Datenexport sind dem Handbuch zum Server (siehe Kapitel A, Seite 105) zu entnehmen.

7.2 Das Stocktaking-Modul

Das *Stocktaking*-Modul ermöglicht das Erstellen und Verwalten von Inventuren. Durch das neuartige Konzept der Änderungsvorschläge wird die Verwaltung von Inventuren erheblich erleichtert, indem Änderungen effizient angenommen, verworfen und rückverfolgt werden können (siehe Kapitel 7.2.5, Seite 67).

Um das Datenbankmodell möglichst modular und übersichtlich zu gestalten, werden diverse Modelle miteinander hierarchisch verknüpft. Die oberste Ebene der Modellhierarchie bildet das `Stocktaking` Modell. In der Abbildung 7.1, Seite 65 ist die Hierarchie in Form eines Klassendiagramms abgebildet. Das Klassendiagramm wurde mithilfe der Erweiterungen `django_extensions` und `pygraphviz` erstellt. Folgendes Kommando wurde dafür aufgerufen [36]:

```
dev_ralph graph_models stocktaking \
-X ItemSplitChangeProposal,MultipleValidationsChangeProposal,
   ValueChangeProposal,TimeStampMixin \
-g -o stocktaking_klassendiagramm.png
```

Um die *Subklassen* der Klasse `ChangeProposalBase` auszublenden, wurden diese mit der Option `-X` exkludiert.

7.2.1 Das Stocktaking Modell

Eine Instanz des `Stocktaking` Modells repräsentiert eine Inventur.

Zu den wichtigsten Attributen des `Stocktaking` Modells zählen:

- `name`: Durch dieses Attribut kann eine Inventur benannt werden. Dieser Name erscheint auf der mobilen Applikation oder dem Inventurbericht.
- `user`: Dieses Attribut referenziert einen hauptverantwortlichen Benutzer einer Inventur.
- `date_started` und `time_started`: Diese Attribute sind Zeitstempel und werden automatisch auf den Zeitpunkt der Erstellung einer `Stocktaking` Instanz gesetzt. Eine Inventur beginnt zum Zeitpunkt ihrer Erstellung.

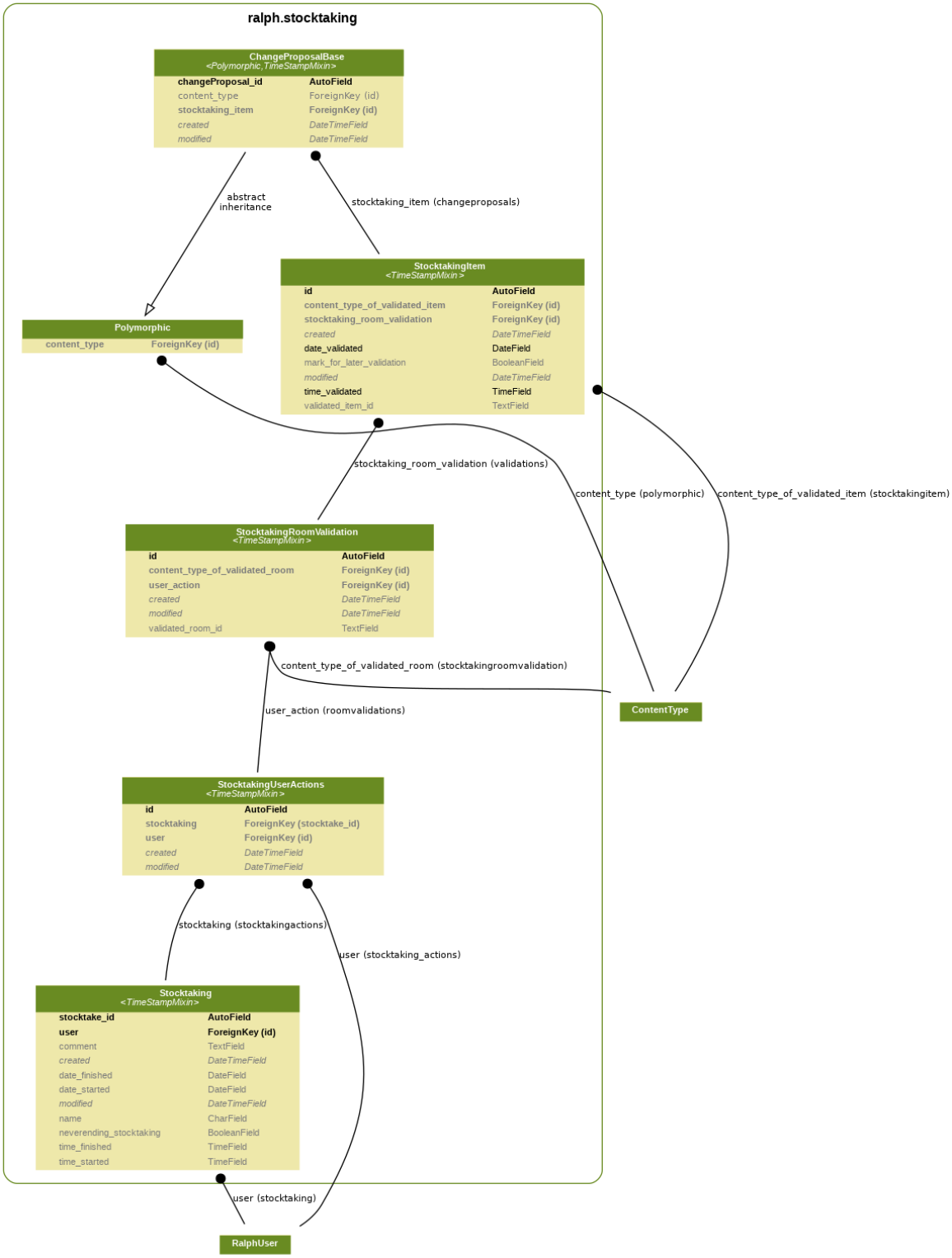


Abbildung 7.1: Das automatisch generierte Klassendiagramm der Modelle des Stocktaking-Moduls.

- `date_finished` und `time_finished`: Diese Attribute sind Zeitstempel und werden gesetzt, wenn ein Administrator eine Inventur beenden möchte. Wenn beide Attribute mit einem Wert befüllt sind, werden keine über die mobile Applikation empfangene Validierungen verarbeitet. Der Zeitpunkt, der sich aus den beiden Attributen ergibt, darf nicht vor dem Zeitpunkt aus `date_started` und `time_started` liegen.
- `neverending_stocktaking`: Wenn dieses *Boolean* Attribut gesetzt ist, hat der Wert der `date_finished` und `time_finished` Attribute keine Bedeutung. Es werden alle Validierungen der mobilen Applikation verarbeitet. Ob in einem Raum bereits einmal validiert wurde oder die Inventur beendet ist, wird nicht mehr überprüft.

7.2.2 Das StocktakingUserActions Modell

Das `StocktakingUserActions` Modell ist die Verbindung zwischen einer Inventur und den Benutzern, die zu dieser Inventur beigetragen haben. Das Attribut `stocktaking` verlinkt je eine bestimmte `Stocktaking` Instanz. Das Attribut `user` verlinkt je eine bestimmte `RalphUser` Instanz, die einem angemeldeten Benutzer entspricht. Validierungen, die ein Benutzer während einer Inventur tätigt, sind mit der entsprechenden `StocktakingUserActions` Instanz verknüpft. Pro Inventur kann es nur eine `StocktakingUserActions` Instanz für einen bestimmten Benutzer geben. Diese Bedingung wird durch die `unique_together` Metavariable [24] festgelegt:

```
unique_together = [["user", "stocktaking"]]
```

7.2.3 Das StocktakingRoomValidation Modell

Das `StocktakingRoomValidation` Modell ist die Verbindung zwischen einer `StocktakingUserActions` Instanz und einer bestimmten Raum-Instanz. Die Raum-Instanz kann dank der `GenericForeignKey` Funktionalität [18] von jedem beliebigen Modell stammen. In der Implementierung einer Client-Schnittstelle wird ein konkretes Modell spezifiziert. Zum Zweck der Inventur an der HTL Rennweg ist das `HTLRoom` Modell in der entsprechenden Client-Schnittstelle spezifiziert. Eine Instanz des `StocktakingRoomValidation` Modells repräsentiert das Validieren von Gegenständen in einem bestimmten Raum.

7.2.4 Das StocktakingItem Modell

Das `StocktakingItem` Modell repräsentiert die Validierung einer Gegenstands-Instanz während einer Inventur. Die Gegenstands-Instanz kann dank der `GenericForeignKey` Funktionalität [18] von jedem beliebigen Modell stammen. In der Implementierung einer Client-Schnittstelle wird ein konkretes Modell spezifiziert. Zum Zweck der Inventur an

der HTL Rennweg ist das `HTLItem` Modell in der entsprechenden Client-Schnittstelle spezifiziert. Durch die Beziehungen zu den Modellen `StocktakingRoomValidation` und dadurch zu `StocktakingUserActions` und `Stocktaking` ist erkennbar, in welchem Raum durch welchen Benutzer im Rahmen welcher Inventur ein Gegenstand validiert wurde. Durch das `ChangeProposalBase` Modell sind die Änderungsvorschläge einer Gegenstandsvalidierung verknüpft.

Zu den wichtigsten Attributen des `StocktakingItem` Modells zählen:

- `date_validated` und `time_validated`: Diese Attribute bilden den Zeitstempel der Gegenstandsvalidierung. Durch die Client-Schnittstelle erstellten `StocktakingItem` Instanzen wird der Wert der aktuellen Systemzeit zum Zeitpunkt des Datenempfangs zugewiesen. Wird ein Gegenstand mehrmals validiert[3], ist der Zeitstempel jener der jüngsten Validierung.
- `mark_for_later_validation`: Dieses *Boolean* Attribut dient zur erleichterten Verifikation der Validierungen durch einen Administrator. Ist das Attribut gesetzt, repräsentiert es eine unvertrauenswürdige Gegenstandsvalidierung. Das Attribut wird gesetzt, wenn der Benutzer durch die mobile Applikation angibt, sich bei einer Gegenstandsvalidierung nicht sicher zu sein und daher das "Später entscheiden" Feld setzt. Innerhalb des Inventur-Berichts werden `StocktakingItem` Instanzen, dessen `mark_for_later_validation` Attribut gesetzt ist, besonders markiert.

7.2.5 Änderungsvorschläge

Änderungsvorschläge beziehen sich auf eine bestimmte `StocktakingItem` Instanz. Ein Änderungsvorschlag ist eine Instanz einer *Subklasse* von `ChangeProposalBase`. Ein Änderungsvorschlag besagt, dass der aktuelle Zustand eines Datensatzes in der Datenbank nicht der Realität entspricht.

Um während einer Inventur nicht sofort jegliche erkannte Änderungen von Gegenstandseigenschaften anwenden zu müssen, werden sie in Änderungsvorschläge ausgelagert. So kann Fehlern, die während einer Inventur entstehen können, vorgebeugt werden. Der Benutzer der mobilen Applikation ist nur damit beauftragt, die aktuellen Inventardaten aus dessen Sicht zu erheben. Es ist die Aufgabe eines Administrators, Änderungsvorschläge als vertrauenswürdig einzustufen und diese tatsächlich anzuwenden.

Als Basis für Änderungsvorschläge dient die Klasse `ChangeProposalBase`. Sie beschreibt nicht, welche Änderung während einer Inventur festgestellt wurde. Sie erbt von der Klasse `Polymorphic`. Die Klasse `Polymorphic` ermöglicht die Vererbung von Modellklassen auf der Datenbankebene. So können alle *Subklassen* der

[3] Dieser Fall tritt bei Inventuren auf, dessen `neverending_stocktaking` Attribut gesetzt ist. So kann derselbe Gegenstand Monate nach der ersten Validierung erneut während derselben Inventur validiert werden.

Klassen `ChangeProposalBase` einheitlich betrachtet werden. Über die Verbindung von `StocktakingItem` zu `ChangeProposalBase` in Abbildung 7.1, Seite 65 sind nicht nur alle verknüpften `ChangeProposalBase` Instanzen, sondern auch alle Instanzen der *Subklassen* von `ChangeProposalBase`, verbunden. Eine *Subklasse* von `ChangeProposalBase` beschreibt eine Änderung, die von einem Benutzer während einer Inventur festgestellt würde. Es sind 4 *Subklassen* von `ChangeProposalBase` und damit 4 Arten von Änderungsvorschlägen implementiert:

1. `ValueChangeProposal`
2. `MultipleValidationsChangeProposal`
3. `ItemSplitChangeProposal`
4. `SAPExportChangeProposal`

Weitere Informationen über die Arten von Änderungsvorschlägen und deren Handhabung sind dem Handbuch zum Server (siehe Kapitel A, Seite 105) zu entnehmen.

7.2.6 Die Client-Schnittstelle

Jede Art der Inventur benötigt eine eigene Client-Schnittstelle. Eine Art der Inventur unterscheidet sich von anderen durch das Gegenstands- und Raummodell, gegen welches inventarisiert wird. Ein Beispiel ist die Inventur für die Gegenstände der HTL Rennweg, durch die gegen die `HTLItem` und `HTLRoom` Modelle inventarisiert wird.

Um die Kommunikation zwischen Server und mobiler Client-Applikation für eine Art der Inventur zu ermöglichen, werden 2 `APIView` Klassen erstellt (siehe Kapitel 6.2.5, Seite 51):

- Eine Klasse, die von der Klasse `BaseStocktakeItemView` erbt.
- Eine Klasse, die von der Klasse `BaseStocktakeRoomView` erbt.

In den *Subklassen* wird u. a. definiert, um welche Gegenstands- und Raummodelle es sich bei der Art der Inventur handelt. Die beiden vordefinierten Klassen `BaseStocktakeItemView` und `BaseStocktakeRoomView` wurden dazu konzipiert, möglichst wenig Anpassung für die Implementierung einer Art der Inventur zu benötigen. Um das Anpassungspotenzial allerdings nicht einzuschränken, sind die Funktionalitäten der Klassen möglichst modular. Eine Funktionalität wird durch eine Methode implementiert, die von den *Subklassen* angepasst werden kann. In dem unten angeführten Beispiel repräsentiert jede Methode eine Anpassung gegenüber dem von `BaseStocktakeItemView` und `BaseStocktakeRoomView` definierten Standardverhalten.

Zu Demonstrationszwecken wurde eine voll funktionstüchtige Client-Schnittstelle für die Inventur des Gegenstandsmodells `BackOfficeAsset` und Raummodells `Warehouse`

in weniger als 30 Code-Zeilen erstellt.

```
class BackOfficeAssetStocktakingView(StocktakingGETWithCustomFieldsMixin,
                                     ClientAttachmentsGETMixin,
                                     BaseStocktakeItemView):
    item_model = BackOfficeAsset
    pk_url_kwarg = None
    slug_url_kwarg = "barcode_final"
    slug_field = "barcode_final"

    display_fields = ("hostname", "status", "location", "price", "provider")
    extra_fields = (
        "custom_fields", "office_infrastructure", "depreciation_rate",
        "budget_info", "property_of", "start_usage"
    )
    explicit_readonly_fields = ("hostname", "price")

    def get_queryset(self, request=None, previous_room_validation=None):
        # annotate custom barcode as either "barcode" or "sn"
        return super(BackOfficeAssetStocktakingView, self).get_queryset(
            request=request, previous_room_validation=previous_room_validation
        ).annotate(
            barcode_final=Case(
                When(Q(barcode__isnull=True) | Q(barcode__exact=""),
                     then=(F("sn"))),
                default=F("barcode"), output_field=CharField()))

    def get_room_for_item(self, item_object):
        return str(item_object.warehouse)

    def item_display_name_getter(self, item_object):
        return str(item_object)

    def item_barcode_getter(self, item_object):
        return str(
            item_object.barcode_final
        ) or "" if hasattr(
            item_object, "id"
        ) else ""


class WarehouseStocktakingView(StocktakingPOSTWithCustomFieldsMixin,
                               ClientAttachmentsPOSTMixin,
                               BaseStocktakeRoomView):
    room_model = Warehouse
```

```
item_detail_view = BackOfficeAssetStocktakingView()

item_room_field_name = "warehouse"
```

Die Modelle `BackOfficeAsset` und `Warehouse` sind in dem unveränderten Ralph-System vorhanden.

Die Klassen `StocktakingGETWithCustomFieldsMixin`, `StocktakingPOSTWithCustomFieldsMixin`, `ClientAttachmentsGETMixin` und `ClientAttachmentsPOSTMixin` ermöglichen das Miteinbeziehen von *Custom-Fields* und Anhängen (siehe Kapitel 7.1.1.4, Seite 57). Weitere Informationen zu den `Mixin`-Klassen kann der im Source-Code enthaltenen Dokumentation entnommen werden.

7.2.6.1 Kommunikationsformat

Die Kommunikation zwischen Server und mobilem Client erfolgt über HTTP Abfragen. Das Datenformat wurde auf das JSON-Format [34] festgelegt. Grund dafür ist die weit verbreitete Unterstützung und relativ geringe Komplexität des Formats.

7.2.6.2 Modell-Voraussetzungen

Um eine Client-Schnittstelle für die Inventarisierung bestimmter Gegenstands- und der dazugehörigen Raummodelle zu implementieren, müssen diese bestimmte Voraussetzungen erfüllen. Die Voraussetzungen werden in diesem Abschnitt beschrieben.

Raum-Gegenstand-Verknüpfung

Das Gegenstandsmodell muss ein Attribut der Klasse `ForeignKey` besitzen, das auf das Raummodell verweist. Der Name dieses Attributs wird in der Klasse, die von `BaseStocktakeRoomView` erbt, in dem Attribut `item_room_field_name` angegeben. Zusätzlich kann in der von `BaseStocktakeItemView` erbenden Klasse durch die `get_room_for_item()` Methode der Raum einer Gegenstandsinstanz als *String* zurückgegeben werden. Ein Implementierungsbeispiel ist in der o. a. angegebenen Client-Schnittstelle zu finden.

String-Repräsentation

Gegenstands- und Rauminstanzen müssen eine vom Menschen lesbare Repräsentation ermöglichen. Standardmäßig wird dazu die `__str__()` Methode der jeweiligen Instanz aufgerufen. Das Verhalten kann durch das Überschreiben der Methoden `item_display_name_getter()` (aus der Klasse `BaseStocktakeItemView`) und `room_display_name_getter()` (aus der Klasse `BaseStocktakeRoomView`) angepasst werden.

Barcode eines Gegenstandes

Es muss für eine Instanz des Gegenstandsmodells ein Barcode generiert werden können. Diese Funktion wird in der Methode `item_barcode_getter()` der von `BaseStocktakeItemView` erbenden Klasse definiert. Um über den von der mobilen Applikation gescannten Barcode auf einen Gegenstand schließen zu können, müssen die Attribute `slug_url_kwarg` und `slug_field` auf den Namen eines Attributes gesetzt werden, das den Barcode eines Gegenstandes repräsentiert. Dieses Attribut muss für jeden Datensatz, der durch die `get_queryset()` Methode entsteht, vorhanden sein. In dem in Kapitel 7.2.6, Seite 68 angegebenen Beispiel wird dieses Attribut in der `get_queryset()` Methode durch `annotate()` [26] hinzugefügt.

7.2.6.3 Einbindung und Erreichbarkeit der APIView-Klassen

Um die für eine Art der Inventur implementierten Klassen über das *DRF* ansprechbar zu machen, werden diese in die Variable `urlpatterns` [30] eingebunden. Die Einbindung erfolgt beispielsweise in der Datei `api.py` eines beliebigen Pakets. Bei der Einbindung müssen den jeweiligen Schnittstellen zur späteren Verwendung interne Namen vergeben werden. Folgendes Implementierungsbeispiel bindet die oben definierten Klassen `BackOfficeAssetStocktakingView` und `WarehouseStocktakingView` ein:

```
urlpatterns = [
    url(
        r"^warehousesforstocktaking/(?:(?P<pk>[\w/]+)/)?$",
        WarehouseStocktakingView.as_view(),
        name="warehousesforstocktaking"),
    url(
        r"^backofficeassetsforstocktaking/(?:(?P<barcode_final>[\w/]+)/)?$",
        BackOfficeAssetStocktakingView.as_view(),
        name="backofficeassetsforstocktaking"),
]
```

Die Namen der Schnittstellen werden auf `"warehousesforstocktaking"` und `"backofficeassetsforstocktaking"` gesetzt. In der Einbindung der Klasse `BackOfficeAssetStocktakingView` wird definiert, dass der Name des Barcode-Attributes eines Gegenstandes `barcode_final` lautet. Das `barcode_final` Attribut wird in der Definition von `BackOfficeAssetStocktakingView` hinzugefügt (s. auch Kapitel 7.2.6.2, Seite 71).

Die Klassen `BackOfficeAssetStocktakingView` und `WarehouseStocktakingView` sind dadurch über die *URLs* `/api/backofficeassetsforstocktaking/` und `/api/warehousesforstocktaking/` erreichbar.

Für die Implementierung der Inventur der HTL Rennweg wurden die *URLs* `/api/htlinventoryitems/` und `/api/htlinventoryrooms/` definiert.

7.2.6.4 Statische Ausgangspunkte

Der mobilen Client-Applikation müssen notwendige Informationen der dynamisch erweiterbaren Inventuren und Inventurarten über statisch festgelegte *URLs* mitgeteilt werden:

Verfügbare Inventuren

Der mobilen Client-Applikation muss mitgeteilt werden, welche Inventuren zurzeit durchgeführt werden können. Für das Modell `Stocktaking` ist eine *API*-Schnittstelle implementiert (siehe Kapitel 6.2.4, Seite 51). Über diese Schnittstelle können durch folgende Abfrage-URL mittels eines HTTP GET-Requests [33] alle verfügbaren Inventuren und dessen einzigartige *IDs* abgefragt werden:

```
/api/stocktaking/?date_finished__isnull=True&time_finish_isnull=True&format=json
```

Durch `&format=json` wird festgelegt, dass der Server eine Antwort im JSON-Format [34] liefert. Die Antwort des Servers auf die Abfrage sieht beispielsweise wie folgt aus (Die Daten wurden zwecks Lesbarkeit auf die wesentlichsten Attribute gekürzt.):

```
[
    {
        "stocktake_id": 1,
        "name": "Inventur 1",
        "comment": "Diese Inventur endet nie!",
        "neverending_stocktaking": true,
    },
    {
```

```
        "stocktake_id": 2,
        "name": "Inventur 2",
        "comment": "",
        "neverending_stocktaking": false,
    }
]
```

Laut der Antwort des Servers sind aktuell 2 Inventuren – “Inventur 1” und “Inventur 2” -verfügbar. Die mobile Client-Applikation benötigt den Wert des Attributs `stocktake_id`, um bei späteren Abfragen festzulegen, welche Inventur von einem Benutzer ausgewählt wurde.

Verfügbare Arten der Inventur

Der mobilen Client-Applikation muss mitgeteilt werden, welche Arten der Inventur verfügbar sind und über welche *URLs* die jeweiligen `APIView` Klassen ansprechbar sind. Unter der *URL* `/api/inventoryserializers/` werden je eine Beschreibung der Inventurart und die benötigten *URLs* ausgegeben. Bei korrekter Implementierung der Inventurarten für die Gegenstandsmodelle `HTLItem` und `BackOfficeAsset` liefert der Server folgende Antwort im JSON-Format [34]:

```
{
    "HTL": {
        "roomUrl": "/api/htlinventoryrooms/",
        "itemUrl": "/api/htlinventoryitems/",
        "description": "Inventur der HTL-Items"
    },
    "BackOfficeAsset-Inventur": {
        "roomUrl": "/api/warehousesforstocktaking/",
        "itemUrl": "/api/backofficeassetsforstocktaking/",
        "description": "Demonstrations-Inventur fuer BackOfficeAssets"
    }
}
```

Um eine Art der Inventur in diese Ausgabe miteinzubeziehen, muss in der Datei `api.py` des Stocktaking-Moduls die Variable `STOCKTAKING_SERIALIZER_VIEWS` entsprechend erweitert werden. Dazu wird der Name der Schnittstelle verwendet, wie er in der Variable `urlpatterns` gesetzt wurde. Die zu dem angeführten Beispiel gehörige `STOCKTAKING_SERIALIZER_VIEWS` Variable ist wie folgt definiert:

```
STOCKTAKING_SERIALIZER_VIEWS = {
    "HTL": (
        "htlinventoryrooms",
```

```
        "htlinventoryitems",
        "Inventur der HTL-Items"
    ),
    "BackOfficeAsset-Inventur": (
        "warehousesforstocktaking",
        "backofficeassetsforstocktaking",
        "Demonstrations-Inventur fuer BackOfficeAssets"
    )
}
```

`"BackOfficeAsset-Inventur"` ist der Name der Inventurart für das Gegenstandsmodell `BackOfficeAsset`. `"warehousesforstocktaking"` ist der Name der Schnittstelle für die Klasse `WarehouseStocktakingView`. `"backofficeassetsforstocktaking"` ist der Name der Schnittstelle für die Klasse `BackOfficeAssetStocktakingView`. `"Demonstrations-Inventur fuer BackOfficeAssets"` ist eine Beschreibung der Inventurart.

7.2.6.5 JSON Schema

Die Daten, die durch die implementierten `APIView` Klassen gesendet oder empfangen werden, müssen einer bestimmten Struktur folgen. In diesem Abschnitt werden die durch die Klassen akzeptierten HTTP-Methoden [33] aufgezählt und deren JSON Schema anhand mehrerer Beispiele dargestellt. Die Beispiele können zwecks Lesbarkeit gekürzt sein. Gekürzte Bereiche werden mit `[...]` markiert.

‘BaseStocktakeItemView‘ GET-Methode

Die Klasse `BaseStocktakeItemView` akzeptiert 2 unterschiedliche Abfragen der GET-Methode.

Eine Abfrage über die *URL* ohne weitere Zusätze liefert eine Liste aller Gegenstände und deren Eigenschaften:

`GET /api/htlinventoryitems/` liefert:

```
{
    "items": [{
            "itemID": 2,
            "displayName": "Tischlampe",
            "displayDescription": "",
            "barcode": "46010000",
            "room": "111 (Klassenraum)",
```

```
            "fields": {
                "anlagenbeschreibung": "Tischlampe",
                [...]
            },
            "attachments": []
        },
        [...]
    ]
}
```

Eine Abfrage über die *URL* mit Zusatz des Barcodes eines Gegenstandes liefert eine Liste aller Gegenstände mit dem entsprechenden Barcode und dessen Eigenschaften:

`GET /api/htlinventoryitems/46010000/` liefert:

```
{
    "items": [{
            "itemID": 2,
            "displayName": "Tischlampe",
            "displayDescription": "",
            "barcode": "46010000",
            "room": "111 (Klassenraum)",
            "fields": {
                "anlagenbeschreibung": "Tischlampe",
                [...]
            },
            "attachments": []
        }
    ]
}
```

'BaseStocktakeItemView' OPTIONS-Methode

Der `OPTIONS` Request wird von der mobilen Client-Applikation benötigt, um ein Formular für Gegenstandsdaten zu erstellen. Dafür werden die Eigenschaften eines Gegenstandes nach Relevanz in `displayFields` (hohe Priorität; wichtige Eigenschaften) und `extraFields`(niedrige Priorität; unwichtige Eigenschaften) geteilt. Welche Eigenschaften welcher Kategorie zugeordnet werden ist in der Implementierung der `BaseStocktakeItemView` *Subklasse* durch die Variablen `display_fields` und `extra_fields` definiert. Jede Eigenschaft wird anhand folgender Kenngrößen beschrieben:

- `verboseFieldName`: Eine vom Menschen lesbare Beschreibung der Eigenschaft.

- `readOnly`: Dieses *Boolean* Feld ist `true`, wenn die Eigenschaft durch Eintragen in die `explicit_readonly_fields` Variable der `BaseStocktakeItemView` *Subklasse* als schreibgeschützt markiert wurde.
- `type`: Dieses Feld gibt die Art der Eigenschaft an. Beispiele sind `"string"`, `"boolean"`, `"integer"`, die jeweils eine Zeichenkette, *Boolean* oder ganzzahlige Nummer identifizieren. Besondere Eigenschaftsarten sind `"choice"` und`"dict"`.
- `choices`: Dieses Feld ist nur angeführt, wenn das `type` Feld den Wert `"choice"` hat. Es beinhaltet eine Liste aller möglichen Werte inkl. vom Menschen lesbare Repräsentation der Werte für die Eigenschaft.
- `fields`: Dieses Feld ist nur angeführt, wenn das `type` Feld den Wert `"dict"` hat. Es beinhaltet eine Kollektion an Eigenschaften, je mit eigenen `verboseFieldName`, `readOnly` und `type` Feldern. Dadurch können Felder rekursiv verschachtelt werden. Ein Beispiel ist unten angeführt.

`OPTIONS /api/htlinventoryitems/` liefert:

```
{
    "displayFields": {
        "anlagenbeschreibung_prio": {
            "verboseFieldName": "interne Gegenstandsbeschreibung",
            "readOnly": false,
            "type": "string"
        },
        "item_type": {
            "verboseFieldName": "Kategorie",
            "readOnly": false,
            "type": "choice",
            "choices": [
                {
                    "value": null,
                    "displayName": "--- kein Wert ---"
                },
                {
                    "value": 1,
                    "displayName": "IT-Infrastruktur"
                },
                [...]
            ]
        },
        [...]
    },
    "extraFields": {
        "anlagenbeschreibung": {
            "verboseFieldName": "SAP Anlagenbeschreibung",
            "readOnly": true,
```

```
            "type": "string"
        },
        "custom_fields": {
            "verboseFieldName": "Custom Fields",
            "readOnly": false,
            "type": "dict",
            "fields": {
                "Beispiel-Custom-Field": {
                    "verboseFieldName": "Beispiel-Custom-Field",
                    "type": "string",
                    "readOnly": false
                }
            }
        },
    }
}
```

'BaseStocktakeRoomView' GET-Methode

Die Klasse `BaseStocktakeRoomView` akzeptiert 2 unterschiedliche Abfragen der GET-Methode. Beide Abfragen benötigen die *ID* der ausgewählten Inventurinstanz als *URL* -Parameter `stocktaking_id`.

Eine Abfrage über die *URL* ohne weitere Zusätze liefert eine Liste aller Räume:

`GET /api/htlinventoryrooms/?stocktaking_id=1` liefert:

```
{
    "rooms": [{
            "roomID": 1,
            "displayName": "111",
            "displayDescription": "Klassenraum",
            "barcode": "11111111",
            "itemID": null
        }, {
            "roomID": 2,
            "displayName": "222",
            "displayDescription": "Labor",
            "barcode": "22222222",
            "itemID": null
        },
    ]
}
```

Eine Abfrage über die *URL* mit Zusatz der *ID* einer Rauminstanz[4] liefert die Details der angegebenen Rauminstanz inkl. aller Gegenstände, die sich in dem Raum befinden sollten, deren Eigenschaften und allen Subräumen der Rauminstanz:

`GET /api/htlinventoryrooms/1/?stocktaking_id=1` liefert:

```
{
    "roomID": 1,
    "displayName": "111",
    "displayDescription": "Klassenraum",
    "barcode": "11111111",
    "itemID": null,
    "subrooms": [],
    "items": [{
            "times_found_last": 1,
            "itemID": 2,
            "displayName": "Tischlampe",
            "displayDescription": "",
            "barcode": "46010000",
            "room": "111 (Klassenraum)",
            "fields": {
                "anlagenbeschreibung": "Tischlampe",
                [...]
            },
            "attachments": []
        }
    ]
}
```

‘BaseStocktakeRoomView‘ POST-Methode

Mit dieser Methode sendet die mobile Client-Applikation den aufgezeichneten Ist-Zustand der in einem Raum befindlichen Gegenstände. Eine Abfrage kann nur über die *URL* mit Zusatz der *ID* einer Rauminstanz[5] getätigt werden. Die *ID* der ausgewählten Inventurinstanz muss als Teil der übermittelten Daten im Feld `stocktaking` angegeben werden. Unter dem Feld `validations` werden alle validierten Gegenstände mit mindestens deren *ID* als Feld `itemID` angegeben. Zusätzlich können alle Gegenstandseigenschaften spezifiziert werden. Die Eigenschaften jedes Gegenstandes werden von der Klasse `BaseStocktakeRoomView` oder der davon erbenden Klasse verarbeitet und

[4] Diese ID wird etwa der Antwort auf die Abfrage ohne URL-Zusatz entnommen. Die entsprechende Eigenschaft ist `roomID`

[5] Diese ID wird etwa der Antwort auf die Abfrage ohne URL-Zusatz entnommen. Die entsprechende Eigenschaft ist `roomID`

mit dem aktuell in der Datenbank eingetragenen Wert verglichen. Bei einer Differenz wird automatisch ein Änderungsvorschlag für diesen Gegenstand erstellt.

`POST /api/htlinventoryrooms/1/` mit folgenden Daten

```
{
    "stocktaking": 1,
    "validations": [
        {
            "itemID": 1
        },
        {
            "itemID": 2
        }
    ]
}
```

liefert die Antwort:

```
{
    "success:": true
}
```

Tritt ein Fehler während der Verarbeitung der Daten auf, wird der Zustand der Datenbank vor dem Empfangen der Daten wiederhergestellt. In diesem Fall teilt der Server dem Client in seiner Antwort mit, welcher Fehler aufgetreten ist.

`POST /api/htlinventoryrooms/1/` mit folgenden Daten

```
{
    "stocktaking": 100,
    "validations": [
        {
            "itemID": 1
        },
        {
            "itemID": 2
        }
    ]
}
```

liefert die Antwort:

```
{
```

```
    "errors": [
        "No stocktaking with ID 100"
    ]
}
```

Um eine Gegenstandsvalidierung zur erneuten Validierung zu einem späteren Zeitpunkt durch einen Administrator zu markieren, kann das Feld `mark_for_later_validation` auf `true` gesetzt werden:

```
{
    "stocktaking": 1,
    "validations": [
        {
            "itemID": 1,
            "mark_for_later_validation": true
        }
    ]
}
```

Dadurch wird die *Boolean*-Eigenschaft der erstellten `StocktakingItem` Instanz gesetzt (siehe Kapitel 7.2.4, Seite 66).

Um einen Gegenstand in einem Subraum (siehe Kapitel 7.1.2.3, Seite 59) zu validieren, kann eine der folgenden 3 Maßnahmen gesetzt werden:

1. Eine separate POST-Abfrage auf die *URL* mit Zusatz der *ID* des Subraums mit den Daten der entsprechenden Gegenstände. Für einen `HTLRoom`-Subraum mit *ID* 100 ist die anzuwendende *URL* `/api/htlinventoryrooms/100/`.
2. Das Setzen des `room` Feldes auf die *ID* des Subraums innerhalb einer Gegenstandsvalidierung. Ein Beispiel bietet die folgende Abfrage. Die *ID* des Subraums ist 100. Die *ID* des übergeordneten Raumes ist 1.

`POST /api/htlinventoryrooms/1/` mit folgenden Daten:

```
{
    "stocktaking": 1,
    "validations": [
        {
            "itemID": 1,
            "room": 100
        }
    ]
}
```

3. Das Auslagern der Gegenstandsvalidierungen in das Feld `subroomValidations` wie in folgendem Beispiel. Das Ergebnis gleicht dem Beispiel aus Alternative 2:

`POST /api/htlinventoryrooms/1/` mit folgenden Daten:

```
{
    "stocktaking": 1,
    "validations": [
    ],
    "subroomValidations": [{
            "roomID": 100,
            "validations": [{
                    "itemID": 1
                }
            ],
            "subroomValidations": [
            ]
        }
    ]
}
```

`subroomValidations` können rekursiv definiert werden. In dem Beispiel aus Alternative 3 muss darauf geachtet werden, dass der Raum mit *ID* 100 direkter "Subraum" des Raums mit *ID* 1 ist.

Weitere Details zur Funktionsweise und Anpassung der Client-Schnittstelle ist der im Source-Code enthaltenen Dokumentation zu entnehmen.

7.2.7 Pull-Request

Es wurde versucht, das `Stocktaking` Modul durch einen *Pull-Request* in das offizielle Ralph-System einzubinden. Durch die in Kapitel 7.2.3, Seite 66 und Kapitel 7.2.4, Seite 66 behandelte `GenericForeignKey` Funktionalität [18] ist es möglich, das Stocktaking-Modul für jegliche Modellklassen zu verwenden. Es gibt keine Beschränkung auf die durch das Diplomarbeitsteam implementierten Klassen.

Die erstellte Lösung wurde im Entwicklungsforum der Ralph Plattform vorgestellt [40] [93]. Die Vorbereitungen für einen Pull-Request wurden getätigt. Das Entwicklerteam der Ralph Plattform hat entschieden, für die erstellte Lösung ein eigenes Git-Repository anzulegen. Das Repository ist dazu vorgesehen, optionale Module zu beherbergen. Ein Pull-Request erfolgt, sobald das Repository von den Entwicklern erstellt wurde.

8 Einführung in die Infrastruktur

8.1 Technische Umsetzung: Infrastruktur

Um allen Kunden einen problemlosen Produktivbetrieb zu gewährleisten, muss ein physischer Server aufgesetzt werden. Auf diesem können dann alle Komponenten unseres Git-Repositories geklont und betriebsbereit installiert werden. Dafür gab es folgende Punkte zu erfüllen:

- das Beschaffen eines Servers
- das Aufsetzen eines Betriebssystems
- die Konfiguration der notwendigen Applikationen
- die Konfiguration der Netzwerkschnittstellen
- das Testen der Konnektivität im Netzwerk
- die Einrichtung des Produktivbetriebs der Applikation
- das Verfassen einer Serverdokumentation
- die Absicherung der Maschine
- die Überwachung des Netzwerks

8.1.1 Anschaffung des Servers

Den 5. Klassen wird, dank gesponserter Infrastruktur, im Rahmen ihrer Diplomarbeit ein Diplomarbeitscluster zur Verfügung gestellt. Damit können sich alle Diplomarbeitsteams problemlos Zugang zu ihrer eigenen virtualisierten Maschine verschaffen. Die Virtualisierung dieses großen Servercluster funktioniert mittels einer ProxMox-Umgebung.

8.1.1.1 ProxMox

Proxmox Virtual Environment (kurz PVE) ist eine auf Debian und KVM basierende Virtualisierungs-Plattform zum Betrieb von Gast-Betriebssystemen. Vorteile:

- läuft auf fast jeder x86-Hardware

- frei verfügbar
- ab 3 Servern Hochverfügbarkeit

Jedoch liegen alle Maschinen der Diplomarbeitsteams in einem eigens gebauten und gesicherten Virtual Private Network (VPN), sodass nur mittels eines eingerichteten Tools auf den virtualisierten Server zugegriffen werden kann.

8.1.1.2 FortiClient

FortiClient ermöglicht es, VPN-Konnektivität anhand von IPsec oder SSL zu erstellen. Die Datenübertragung wird verschlüsselt und damit der enstandene Datenstrom vollständig gesichert über einen sogenannten "Tunnel" übertragen.

Da die vorliegende Diplomarbeit die Erreichbarkeit des Servers im Schulnetz verlangt, muss die Maschine in ausreichendem Ausmaß abgesichert werden, damit sie ohne Bedenken in das Schulnetz gehängt werden kann. Dafür müssen folgende Punkte gewährleistet sein:

- Firewallkonfiguration (siehe Kapitel 8.1.7, Seite 97)
- wohlüberlegte Passwörter und Zugriffsrechte

8.1.2 Wahl des Betriebssystems

Neben den physischen Hardwarekomponenten wird für einen funktionierenden und leicht bedienbaren Server selbstverständlich auch ein Betriebsystem benötigt. Die erste Entscheidung, welche Art von Betriebssystem für die Diplomarbeit in Frage kam, wurde rasch beantwortet: Linux. Während der Schulzeit an der HTL Rennweg durfte das Diplomarbeitsteam auf zwei verschiedenen Linux-Distributionen, die auch für den Serverbetrieb der Diplomarbeit in Frage kamen, Übungen durchführen:

- Linux CentOS
- Linux Ubuntu

8.1.2.1 Linux CentOS

CentOS ist eine frei verfügbare Linux Distribution, die auf Red-Hat [77] aufbaut. Hinter Ubuntu und Debian ist CentOS die am dritthäufigsten verwendete Linux-Plattform und wird von einer offenen Gruppe von freiwilligen Entwicklern betreut, gepflegt und weiterentwickelt.

8.1.2.2 Linux Ubuntu

Ubuntu ist die am meisten verwendete Linux-Betriebssystemsoftware für Webserver. Auf Debian basierend, ist das Ziel der Ubuntu-Entwickler, ein einfach zu installierendes und leicht zu bedienendes Betriebssystem mit aufeinander abgestimmter Software zu schaffen. Hauptsponsor des Ubuntu-Projektes ist der Software-Hersteller Canonical [9], der vom südafrikanischen Unternehmer Mark Shuttleworth [57] gegründet wurde.

8.1.2.3 Vergleich und Wahl [37]

CentOS

- Kompliziertere Bedienung
- Keine regelmäßigen Softwareupdates
- Weniger Dokumentation vorhanden

Ubuntu

- Leichte Bedienung
- Wird ständig weiterentwickelt und aktualisiert
- Zahlreich brauchbare Dokumentation im Internet vorhanden
- Wird speziell von "Ralph" empfohlen

Aus den angeführten Punkten entschied sich das Diplomarbeitsteam, aufgrund der auf der Hand liegenden Vorteile, das Betriebsystem Ubuntu zu verwenden, vor allem auch weil der Hersteller der Serversoftware "Ralph" die Verwendung dieses Betriebsystems empfiehlt. Anschließend wird die Installation des Betriebsystems genauer erläutert und erklärt.

8.1.3 Installation des Betriebsystems

Wie bereits unter Punkt "Anschaffung des Servers" (siehe Kapitel 8.1.1, Seite 83) erwähnt, wird uns von der Schule ein eigener Servercluster mit virtuellen Maschinen zur Verfügung gestellt. Durch die ProxMox-Umgebung und diversen Tools, ging die Installation der Ubuntu-Distribution rasch von der Hand. In der Virtualisierungsumgebung der Schule musste nur ein vorhandenes Linux-Ubuntu 18.04 ISO-File gemountet und anschließend eine gewöhnliche Betriebsysteminstallation für Ubuntu durchgeführt werden. Jedoch kam es beim ersten Versuch zu Problemen mit der Konfiguration der Netzwerkschnittstellen, die im nächsten Punkt genauer erläutert werden.

8.1.4 Internetkonnektivität der Maschine

8.1.4.1 Konfiguration

Im Rahmen des Laborunterrichts an der HTL Rennweg bekamen die Schüler für diverse Unklarheiten ein sogenanntes Cheat-Sheet [10] für Linux-Befehle zur Verfügung gestellt. In diesem Cheat-Sheet finden sich unter anderem Anleitungen für die Konfiguration einer Netzwerkschnittstelle auf einer CentOS/RedHat sowie Ubuntu/Debian-Distribution. Den Schülern der fünften Netzwerktechnikklasse sollte diese Kurzkonfiguration jedoch schon leicht von der Hand gehen, da sie diese Woche für Woche benötigen.

Eine Netzwerkkonfiguration mit statischen IPv4-Adressen für eine Ubuntu-Distribution könnte wie folgt aussehen:

In `/etc/network/interfaces`:

```
auto ens32
iface ens32 inet static
address 192.168.0.1
netmask 255.255.255.0
```

Eine Netzwerkkonfiguration mit Verwendung eines IPv4-DHCP-Servers für eine Ubuntu-Distribution könnte wie folgt aussehen:

In `/etc/network/interfaces`:

```
auto ens32
iface ens32 inet dhcp
```

8.1.4.2 Topologie des Netzwerkes

Unter Abbildung 8.1 ist der Netzwerkplan veranschaulicht. Auf der linken Seite ist der Servercluster des Diplomarbeitsteams dargestellt, worauf die virtuelle Maschine der Diplomarbeit gehostet wird. In der Mitte ist die FortiGate-Firewall zu sehen, die nicht nur als äußerster Schutz vor Angriffen dient, sondern auch die konfigurierte VPN-Verbindung beinhaltet und nur berechtigten Teammitgliedern den Zugriff gewährleistet. Desweiteren ist die moderne Firewall auch für die Konnektivität der virtuellen Maschine im Schulnetz zuständig, aber dazu später (unter Punkt “Absicherung der virtuellen Maschine”) mehr.

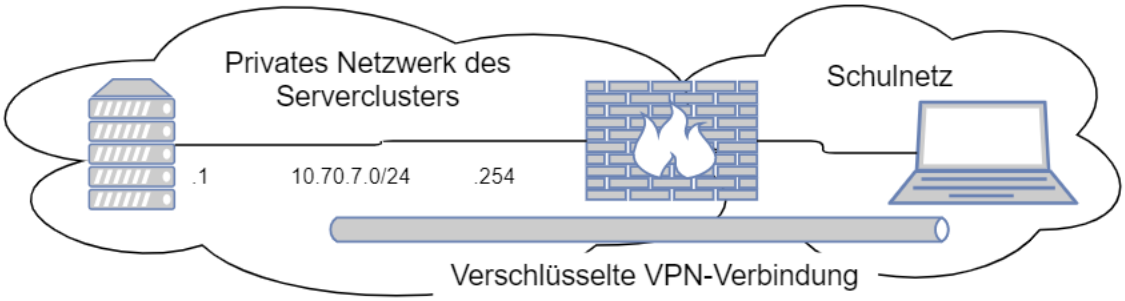


Abbildung 8.1: Netzwerkplan

8.1.5 Installation der notwendigen Applikationen

Damit die Ubuntu-Maschine für den Produktivbetrieb startbereit ist, müssen im Vorhinein noch einige wichtige Konfigurationen durchgeführt werden. Die wichtigste (Netzwerkkonfiguration) wurde soeben ausführlich erläutert, doch ohne der Installation von diversen Applikationen, wäre das System nicht brauchbar.

8.1.5.1 Advanced Packaging Tool

Mit diesem Tool werden auf dem System die notwendigen Applikationen heruntergeladen, extrahiert und anschließend installiert. Insgesamt stehen einem 18 apt-get commands zur Verfügung. Genauere Erklärungen zu den wichtigsten commands folgen.

apt-get update Update liest alle in `/etc/apt/sources.list`, sowie in `/etc/apt/sources.list.d/` eingetragenen Paketquellen neu ein. Dieser Schritt wird vor allem vor einem upgrade-command oder nach dem Hinzufügen einer neuen Quelle empfohlen, um sich die neusten Informationen für Pakete ansehen zu können.

apt-get upgrade Upgrade bringt alle bereits installierten Pakete auf den neuesten Stand.

apt-get install Install lädt das Paket bzw. die Pakete inklusive der noch nicht installierten Abhängigkeiten (und eventuell der vorgeschlagenen weiteren Pakete) herunter und installiert diese. Außerdem besteht die Möglichkeit, beliebig viele Pakete auf einmal anzugeben, indem sie mittels eines Leerzeichens getrennt werden.

apt-get remove Mit apt-get remove werden nicht mehr benötigte Pakete von einem Ubuntu-System vollständig entfernt.

8.1.5.2 Installierte Pakete

NGINX NGINX ist einer der am häufigsten verwendeten OpenSource-Webserver unter Linux für diverse Webanwendungen. Große Unternehmen wie Cisco, Microsoft, Facebook oder auch IBM verwenden diesen Webserver für deren Zwecke. Unter anderem wird NGINX auch als Reverse-Proxy, HTTP-Cache und Load-Balancer verwendet. Wie genau NGINX für den Produktivbetrieb funktioniert, wird in einem eigenen Punkt (siehe Kapitel 8.1.6.6, Seite 92) erläutert.

Docker Docker ist eine frei verwendbare Software, die die Erstellung und den Betrieb von Linux Containern ermöglicht. Wie genau dies funktioniert, wird etwas später (siehe Kapitel 8.1.6.6, Seite 92) genauer beschrieben und erklärt.

docker-compose Die Verwaltung und Verlinkung von mehreren Containern kann auf Dauer sehr nervenaufreibend sein. Die Lösung dieses Problems nennt sich docker-compose. Wie docker-compose jedoch genau funktioniert, wird ebenfalls, wie das Grundkonzept von Docker, (siehe Kapitel 8.1.6.6, Seite 92) präziser erläutert.

MySQL MySQL ist ein OpenSource-Datenverwaltungssystem und die Grundlage für die meisten dynamischen Websiten. Darauf werden die Inventurdatensätze der HTL Rennweg gespeichert. Nähere Informationen finden sich ebenfalls während der Erklärung der Funktionsweise des Produktivbetriebs (siehe Kapitel 8.1.6.6, Seite 92) wieder.

Redis Redis ist eine In-Memory-Datenbank mit einer Schlüssel-Wert-Datenstruktur (Key Value Store). Wie auch MySQL handelt es sich um eine OpenSource-Datenbank.

Nagios Nagios ist ein Monitoring-System, mit dem sich verschiedene Geräte und auch laufende Dienste (oder auch Eigenschaften) überwachen lassen. Ziel ist es dabei, schnell Ausfälle festzustellen und diese dem zuständigen Administrator mitzuteilen, sodass dieser dann schnell darauf reagieren kann.

virtualenv Bei virtualenv handelt es sich um ein Tool, mit dem eine isolierte Python-Umgebung erstellt werden kann. Eine solche isolierte Umgebung besitzt eine eigene

Installation von diversen Services und teilt ihre libraries nicht mit anderen virtuellen Umgebungen (im optionalen Fall greifen sie auch nicht auf die global installierten libraries zu). Dies bringt vor allem den großen Vorteil, dass im Testfall virtuelle Umgebungen aufgesetzt werden können, um nicht die globalen Konfigurationen zu gefährden.

Python Python ist einer der Hauptbestandteile auf dem Serversystem der Diplomarbeit. Das Backend (=Serveranwendung) basiert, wie bereits erwähnt, auf dem Python-Framework "Django". Um dieses Framework auf dem System installieren zu können wird jedoch noch ein weiteres "Packaging-Tool", speziell für Python-Module, benötigt.

pip Dieses Tool nennt sich Pip. Pip ist ein rekursives Akronym für **P**ip **I**nstalls **P**ython und ist, wie bereits erwähnt, das Standardverwaltungswerkzeug für Python-Module. Die Funktion sowie Syntax kann relativ gut mit der von apt verglichen werden.

uWSGI Das eigentliche Paket, mit dem der Produktivbetrieb schlussendlich gewährleistet wurde, nennt sich uWSGI. Speziell wurde es für die Produktivbereitstellung von Serveranwendungen (wie eben der Django-Server der vorliegenden Diplomarbeit) entwickelt und harmoniert eindrucksvoll mit der Webserver-Software NGINX. Die grundlegende Funktionsweise von uWSGI, sowie eine Erklärung warum schlussendlich diese Software und nicht Docker verwendet wurde, wird in einem eigenen Teil (siehe Kapitel 8.1.6.2, Seite 91) veranschaulicht.

Django Django ist ein in Python geschriebenes Webframework, auf dem unsere Serveranwendung basiert. Genauere Informationen wurden jedoch schon übermittelt. (siehe Kapitel 6.2, Seite 48)

runsslserver Runsslserver ist ein Python-Paket mit dem eine Entwicklungsumgebung über https erreichbar gemacht werden kann.

8.1.6 Produktivbetrieb der Applikation

Das Aufsetzen beziehungsweise die Installation der Produktivumgebung ist der wichtigste, aber auch aufwendigste, Bestandteil jeder Serverinfrastruktur. Eine Produktivumgebung soll von einer Testumgebung möglichst weit getrennt sein, damit die zu testende Software keinen Schaden für den produktiven Betrieb anrichten kann. Weiters soll durch den Produktivbetrieb der Server um einiges perfomanter sein, da

dieser, im Falle der vorliegenden Diplomarbeit, einen optimierten Webserver (NGINX) verwendet.

8.1.6.1 Entwicklungsumgebung

Die Entwickler des Grundservers "Ralph" haben eine eigene Entwicklungsumgebung für den Django-Server erstellt. Normalerweise findet sich in einem klassischen Python-Projekt oder einem Projekt, das auf einem Python-Framework (wie zum Beispiel Django) basiert, ein sogenanntes "`manage.py`"-File. Hierbei handelt es sich um ein Script, das das Management eines beliebigen Python-Projektes unterstützt. Mit dem Script ist man unter anderem in der Lage, den Webserver auf einem unspezifischen Rechner zu starten, ohne etwas Weiteres installieren zu müssen.

Jedoch wurde diese Datei im Vorgängerprojekt in eine eigene spezifische Entwicklungsumgebung umimplementiert. Der Server startet nach der eigenen Installation (welche ab Seite 4 im Dokument "Serverdokumentation Schritt für Schritt erklärt wird) nicht mehr mittels:

```
python manage.py runserver
```

sondern mit:

```
dev_ralph runserver 0.0.0.0:8000
```

Obwohl die Befehle dieses Serverstarts nicht wirklich ident aussehen, führen sie im Hintergrund aber die gleichen Unterbefehle aus, um eine sichere Verwendung der Entwicklungsumgebung zu gewährleisten.

Da das Diplomarbeitsteam diese Entwicklungsumgebung vorerst auf dem Produktivserver für Testzwecke verwendete, wurde ein eher unbekanntes Paket namens "runsslserver" für die Entwicklungsumgebung installiert und eingebaut. Mit diesem Tool kann eine Django-Entwicklungsumgebung mittels "https" gestartet werden. Zuerst musste, üblich um eine https-Verbindung einzurichten, beispielsweise mit dem Befehl

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048
-keyout /ssl/nginx.key -out /ssl/cert.crt
```

ein Zertifikat mit dem zugehörigen Schlüssel generiert werden.

Kurzerklärung der Befehlsoptionen

- `-x509` Gibt an, statt eines CSR, gleich ein selbstsigniertes Zertifikat auszustellen.
- `-nodes` Zertifikat wird nicht über ein Kennwort geschützt. Damit kann der Server ohne weitere Aktion (Eingabe des Kennworts) gestartet werden.
- `-days` Beschreibt die Gültigkeit des Zertifikats für 365 Tage.
- `-newkey rsa:2048` Generiert das Zertifikat und einen 2048-bit langen RSA Schlüssel.
- `-keyout` Gibt die Ausgabepfad und -datei für den Schlüssel an.
- `-out` Gibt Ausgabepfad und -datei des Zertifikats an.

Der Befehl "runsslserver" muss jetzt nur noch in den "Installed-Apps" des Projekts (im vorliegenden Projekt befinden sich diese in der Datei "`base.py`") hinzugefügt werden.

Dann kann der Server problemlos mit

```
dev_ralph runsslserver 0.0.0.0:443
```

gestartet werden.

8.1.6.2 Probleme der Produktivumgebung

Zu Beginn dachte das Diplomarbeitsteam, dass die Umsetzung des Django-Servers in eine Produktivumgebung nicht allzu kompliziert wäre, da von Ralph bereits Dockerfiles für den Produktivbetrieb vorlagen. Dadurch wurde zu Beginn der Arbeit versucht dieses bereits existierendes Dockersystem zu starten, jedoch war im Browser dann nicht der Diplomarbeitsserver, sondern die Basislösung des Vorgängers zu sehen. Der Infrastrukturverantwortliche analysierte anschließend die Docker-Dateien und musste feststellen, dass sie für die Zwecke der Diplomarbeit so tatsächlich nicht verwendet werden können. Ralph hat in Verbindung mit dem implementierten Dockersystem einige komplexe, aufeinander zugreifende Skripts, verwendet. Vorweg muss gesagt werden, dass die Vorgängerserverlösung als Service angeboten wird und in den vorliegenden Skripts ohne den Änderungen des Diplomarbeitsteam installiert wird. Daraufhin wird aus dem Webservice ein Docker-Container erstellt und anschließend mit den anderen Docker-Komponenten hochgefahren. Dadurch war im Webbrowser beim Serverabruf, statt der gewollten Serverlösung, die falsche zu sehen. Der Fortschritt des Arbeitspaketes der Produktivumgebung war somit wieder um einiges geschrumpft und es musste schnellstmöglich eine Alternative gefunden werden, da das Umschreiben der Dockerfiles, beziehungsweise der Skripts, ein riesiger Aufwand wäre.

Nach kurzer Recherche stieß man schließlich auf zwei verwendbare Alternativen, die nachfolgend genannt und miteinander verglichen werden.

8.1.6.3 Alternativen

Es wurde bereits erläutert, warum das Diplomarbeitsteam die Dockerlösung des Vorgängers nicht verwendet (siehe Kapitel 8.1.6.2, Seite 91). Allerdings musste rasch eine Alternative für die Umsetzung des Produktivbetriebs gesucht werden, um möglichst wenig Zeit zu verlieren. Zwei Alternativen, die für den Produktivbetrieb in Frage kamen, wurden genauer analysiert: uWSGI sowie Gunicorn.

8.1.6.4 uWSGI vs. Gunicorn

Bei beiden Alternativen handelt es sich um Schnittstellen-Spezifikationen, unter anderem für die Programmiersprache Python, die eine Schnittstelle zwischen Webservern und Webframeworks bzw. Web Application Servern festlegen, um die Portabilität von Webanwendungen auf unterschiedlichen Webservern zu fördern.

Gunicorn ist ein Pre-Fork-Worker-Modell[75], das aus Rubys Unicorn-Projekt portiert wurde. Der Gunicorn-Server ist weitgehend kompatibel mit verschiedenen Web-Frameworks, einfach implementierbar, spart Serverressourcen und ist recht schnell.

Das uWSGI-Projekt zielt darauf ab, einen vollständigen Stack für den Aufbau von Hosting-Diensten zu entwickeln.

8.1.6.5 Wahl

Da beide Alternativen sehr ähnlich sind, lag es am Infrastrukturverantwortlichen, welche für die Installation der Produktivumgebung ausgewählt wird. Zuerst wurde versucht, alles mithilfe von Gunicorn aufzusetzen. Da dies jedoch mehrmalig Probleme verursachte, entschied man sich für die Verwendung von uWSGI. Nach kurzer Recherche stieß der Infrastrukturverantwortliche auf ein vielversprechendes Tutorial im Internet[52], mit dem die Installation reibungslos verlief. Die genaue Installationsanleitung wurde bereits in der Serverdokumentation niedergeschrieben. Die Funktionsweise, also wie uWSGI genau arbeitet, wird nachfolgend (siehe Kapitel 8.1.6.6, Seite 92) anhand einer Grafik erklärt.

8.1.6.6 Funktionsweise der Produktivumgebung

Funktion von uWSGI Es wurde bereits des öfteren erklärt, warum die Dockerlösung des Vorgängers vom Diplomarbeitsteam nicht verwendet wird. (siehe Kapitel 8.1.6.2, Seite 91) Jedenfalls wurde nun die Einrichtung der Produktivumgebung mittels uWSGI

erfolgreich durchgeführt. In der folgenden Grafik wird die Funktion von uWSGI genauer dargestellt.

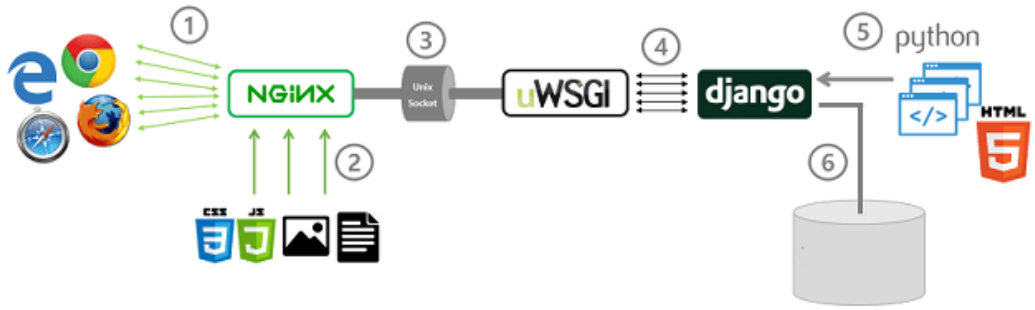


Abbildung 8.2: Funktionsweise von uWSGI
[45]

1. Ein beliebiger Benutzer eines Webbrowsers (zum Beispiel Google Chrome oder Mozilla Firefox) sendet einen sogenannten "Webrequest" auf den https-Port "443" (siehe Kapitel 8.1.6.2, Seite 91).
2. Ein beliebig gewählter, optimierter Webserver (im Fall der vorliegenden Diplomarbeit "NGINX") stellt Dateien wie Javascript, CSS oder auch Bilder bereit und macht diese für den Benutzer abrufbar.
3. Hier wird die Kommunikation zwischen dem hochperfomanten Webserver NGINX und dem Webinterface uWSGI, mit Verwendung eines klassischen Websockets, veranschaulicht. (Anm.: Bei einem Websocket handelt es sich um ein auf TCP basierendes Protokoll, das eine bidirektionale Verbindung zwischen einer Webanwendung und einem Webserver herstellt).
4. Das eigentliche Interface von uWSGi ist hier zu sehen. Diese Schnittstelle sorgt für die Kommunikation des oben genannten Websockets mit dem verwendeten Python-Framework.
5. Django reagiert nun auf die Anfrage des Benutzers und überlässt diesem (falls dessen Zugriffsrechte darauf es erlauben) die gewünschten Daten.
6. Die vom Benutzer gewünschten Daten werden in einer MySQL-Datenbank gespeichert.

Grundsätzlich kann die Grafik auch mittels

```
the web client <-> the web server <-> the socket <-> uwsgi <-> Django
```

als normale ASCII-Zeichenkette dargestellt werden.

Funktion von Docker Docker wird im Projekt ausschließlich für die Bereitstellung der Datenbank verwendet. Der Befehl

```
docker-compose -f docker/docker-compose-dev.yml up -d
```

im Projektstammverzeichnis erstellt und startet die in der folgenenden Grafik zu sehenden Container, die das Datenbanksystem darstellen:

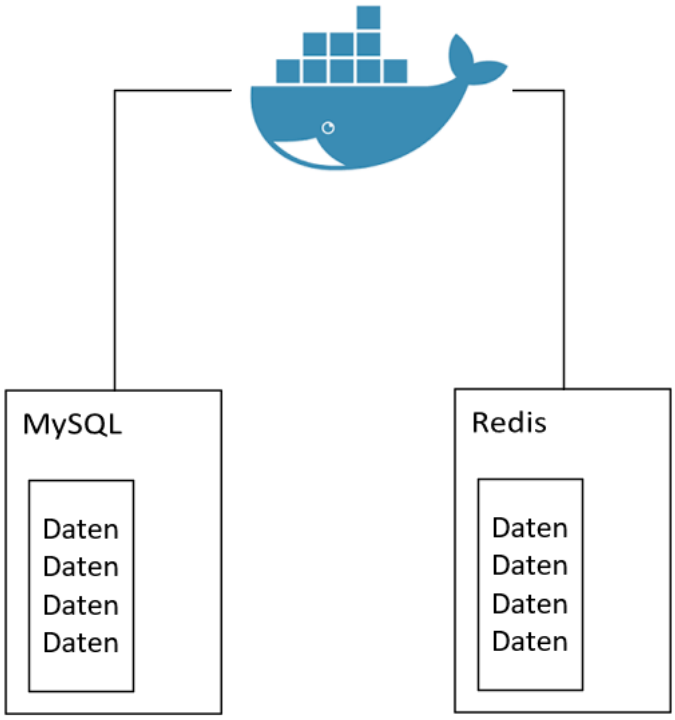


Abbildung 8.3: Datenbanksystem mit Docker

Die erstellten Container enthalten die Datenbankanwendungen, sowie deren Ressourcen und speichern beziehungsweise stellen eine dauerhafte Verfügbarkeit der erfassten Inventurdaten bereit.

8.1.6.7 Erreichbarkeit mittels HTTPS

Weiters wurde NGINX mit einem sogenannten Self Signed Certificate ausgestattet, um eine sichere Verbindung des Benutzers mit dem Webserver zu gewährleisten. Die Konfiguration von NGINX für den Gebrauch von uWSGI mit HTTPS ist auf Seite 9 der Serverdokumentation zu finden. Im Webbrowser “Microsoft Edge” würde der Aufruf des Servers nun wie folgt aussehen:

Der Zertifikatsfehler, der am Bild deutlich zu sehen ist, bedeutet jedoch nichts anderes als dass das Zertifakt des Produktivservers der Diplomarbeit nicht von einer offiziellen

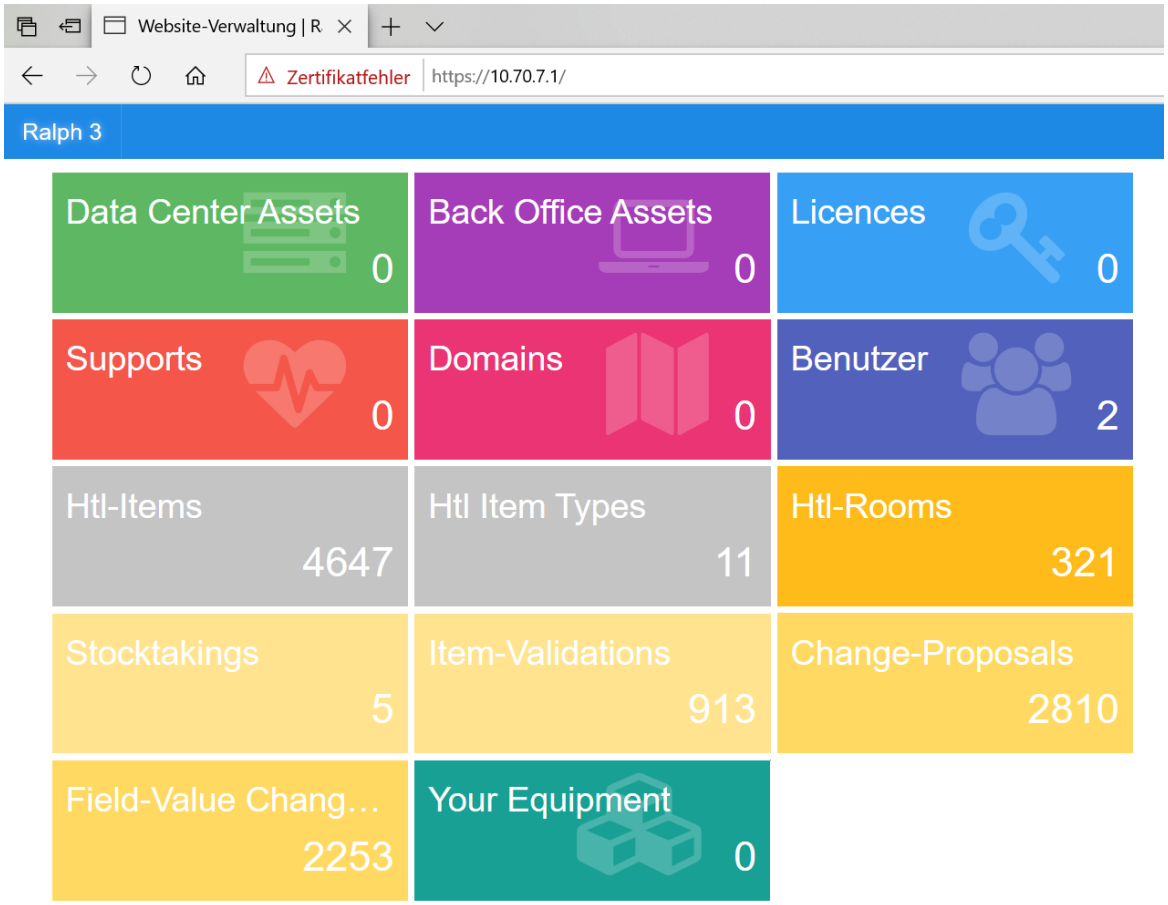


Abbildung 8.4: Aufruf des Servers über HTTPS

Zertifizierungsstelle signiert wurde. Da der Server sowieso nur im Schulnetz der HTL 3 Rennweg beziehungsweise über einen VPN-Tunnel erreichbar ist, war es nicht nötig, sich an solch eine Zertifizierungsstelle zu wenden.

8.1.6.8 Verwendung einer .ini-Datei

Bei einem File mit einer .ini-Endung handelt es sich um eine Initialisierungsdatei. Diese beinhaltet beispielsweise Konfigurationsmöglichen die zum Start eines bestimmten Dienstes benötigt werden. Glücklicherweise funktioniert so ein File auch wunderbar mit uWSGI. Durch die Implementierung einer ralph.ini-Datei wurden die Befehlsoptionen des `uwsgi`-Befehls ausgelagert und dieser somit für den Serverstart verkürzt.

Ein kleiner Vergleich:

```
uwsgi --socket mysite.sock --module mysite.wsgi --chmod-socket=664 --proceses 10

uwsgi --ini ralph.ini
```

Beide Befehle liefern schlussendlich das gleiche Ergebnis, jedoch ist der unter Befehl (Verwendung einer .ini-Datei) um einiges kürzer und erspart somit nervernaufreibende Schreibarbeit.

8.1.6.9 Neustartverhalten mittels Service

Für den Gebrauch von uWSGi gibt es einige Lösungen, um den automatischen Neustart, beispielsweise bei Eintritt eines Stromausfalls, zu gewährleisten. Das Team hat sich mit der Implementierung eines eigenen "Ralph-Service" für den klassischen Linuxweg entschieden.

Systemd Systemd gibt es seit 2010 und wurde ursprünglich von Lennart Poettering (Red Hat Inc.) entwickelt. Es ist heutzutage der de-facto Standard, daher in nahezu allen Distros standardmäßig inkludiert und hat damit System-V-Init abgelöst (ist aber noch zu 99% mit System-V-Init Skripten abwärtskompatibel). Die ursprüngliche Überlegung war ein schnellerer Systemstart durch Parallelisierung und Verwenden von kompiliertem C-Code statt Shell-Boot-Skripten. Mittlerweile ist Systemd nicht NUR Service- sondern ganzer System-Manager und daher viel mächtiger als Init-V. Systemd wird als erster Prozess vom Linux-Kernel gestartet und hat daher die PID 1.

.service-Datei

```
[Unit]
Description=ralphserver
After=network.target

[Service]
ExecStart=/usr/bin/uwsgi /home/capentory/capentory-prod/
          capentory-ralph/ralph/ralph.ini

Restart=on-failure
RestartSec=60s

[Install]
WantedBy=multi-user.target
```

- **Description**: Kurzbeschreibung des Dienstes
- **After=network.target**: Der Dienst wird gestartet sobald eine Netzwerkverbindung besteht
- **ExecStart**: Der auszuführende Shell-Befehl (in diesem Fall wird der uWSGI-Server gestartet)

- **Restart und RestartSec**: Hier wird der Dienst beispielsweise bei einem unsauberen Beenden des Prozesses, einem Timeout, oder ähnlichem neugestartet. Es ist auch möglich, einen Dienst neu zu starten, wenn der Watchdog einen Fehler meldet.
- **WantedBy=multi-user.target**: Normalbetrieb

Start des Service Nach dem korrekten Erstellen der `ralph.service`-Datei kann dieser als ganz normaler Linux-Service behandelt werden. Mit

```
systemctl enable ralph
systemctl start ralph
```

wird der angelegte Service nun immer, wenn die virtuelle Maschine gestartet wurde, ausgeführt.

8.1.7 Absicherung der virtuellen Maschine

Ein weiterer wichtiger und sensibler Teil der Serverinfrastruktur ist die Absicherung der virtuellen Maschine gegen Angriffe. Da es sich im Rahmen der Diplomarbeit um geheime Daten der Schulinventur handelt, war es ein Anliegen der Verantwortlichen, dass mit diesen Daten verantwortungsvoll und vorsichtig umgegangen wird.

8.1.7.1 Firewall

Weiter oben (siehe Kapitel 8.1.4.2, Seite 86) wird der Plan des Netzwerkes veranschaulicht. Die in der Mitte liegende FortiGate-Firewall stellt, wie bereits in oben genannten Punkt erwähnt, die Verfügbarkeit des Servers im Schulnetz bereit. Dadurch dass der Server nur über eine VPN-Verbindung konfiguriert werden kann, denkt man sich bestimmt, dass die virtuelle Maschine schon genug gesichert sei. Jedoch ist es sinnvoll den Server doppelt abzusichern, daher wurden auf der Linux-Maschine ebenfalls noch Firewallregeln konfiguriert.

Erlauben einer SSH-Verbindung Die virtuelle Maschine wurde aufgrund der nicht-vorteilhaften Konsole von ProxMox immer über SSH mit einer beliebigen externen Konsole (beispielsweise Putty) konfiguriert. Dies ist wegen der FortiGate-Konfiguration nur mit VPN-Verbindung möglich. Direkt auf dem Server wurde daher eine Firewallregel für die Erlaubnis von SSH implementiert.

Regel: `sudo ufw allow ssh`

oder: `sudo ufw allow 22`

Erlauben des Datenaustausches mittels HTTP Obwohl der Webserver den Datenaustausch mit HTTPS bevorzugt, wurde auf der zweiten Ebene auch eine Regel für die HTTP-Verbindung konfiguriert.

Regel: `sudo ufw allow http`

oder: `sudo ufw allow 80`

Erlauben des Datenaustausches mittels HTTPS Für den Datenaustausch über HTTPS musste ebenso eine Regel aktiviert werden.

Regel: `sudo ufw allow https`

oder: `sudo ufw allow 443`

Verbieten aller restlichen Verbindungen Zuguterletzt müssen alle restlichen (die nicht von Administratoren gebrauchten) Verbindungen deaktiviert werden, um Angriffslücken zu schließen.

Regel: `ufw default deny`

8.1.7.2 Mögliche Angriffsszenarien

Da der Server im Schulnetz erreichbar ist, gelten unter anderem auch die Schüler der HTL 3 Rennweg als potenzielle Angreifer.

Malware Bei Malware handelt es sich um Schadsoftware, zu dem unter anderem **Viren**, **Würmer** und **Trojaner** zählen.

Angriffe auf Passwörter Neben dem Raten und Ausspionieren von Passwörtern ist die Brute Force Attacke weit verbreitet. Bei dieser Attacke versuchen Hacker mithilfe einer Software, die in einer schnellen Abfolge verschiedene Zeichenkombinationen ausprobiert, das Passwort zu knacken. Je einfacher das Passwort gewählt ist, umso schneller kann dieses geknackt werden.

Vorbeugung des Diplomarbeitsteams: Die festgelegten Passwörter sind komplex aufgebaut und sicher in den Köpfen der Mitarbeiter gespeichert.

Man-in-the-middle Attacken Bei der „Man-in-the-Middle“-Attacke nistet sich ein Angreifer zwischen den miteinander kommunizierenden Rechnern ein. Diese Position ermöglicht es ihm, den ausgetauschten Datenverkehr zu kontrollieren und zu manipulieren. Er kann z. B. die ausgetauschten Informationen abfangen, lesen, die Weiterleitung kappen, usw.. Von all dem erfährt der Empfänger aber nichts.

Vorbeugung des Diplomarbeitsteams: Der Datenaustausch verläuft über HTTPS und ist somit verschlüsselt.

Sniffing Unter Sniffing (Schnüffeln) wird das unberechtigte Abhören des Datenverkehrs verstanden. Dabei werden oft Passwörter, die nicht oder nur sehr schwach verschlüsselt sind, abgefangen. Andere Angreifer bedienen sich dieser Methode, um herausfinden zu können, welche Teilnehmer über welche Protokolle miteinander kommunizieren. Mit den so erlangten Informationen können die Angreifer dann den eigentlichen Angriff starten.

Vorbeugung des Diplomarbeitsteams: Der Datenaustausch verläuft über HTTPS und ist somit verschlüsselt.

8.1.8 Überwachung des Netzwerks

Sollte der Fall eintreten, dass der Server nicht mehr erreichbar ist, wurde ein eigener Monitoring-Server im Netzwerk eingehängt und installiert. Dadurch wird der Produktivserver rund um die Uhr überwacht. Weiters wird das Diplomarbeitsteam bei etwaigen Komplikationen per E-Mail benachrichtigt, damit das aufgetretene Problem, beziehungsweise die aufgetretenen Probleme, möglichst schnell behoben werden können.

8.1.8.1 Topologieänderung

Im Netzwerk wurde ein zweiter Server installiert, der mit dem OpenSource-Programm “Nagios” als Monitoring-Maschine für den Produktivserver dient.

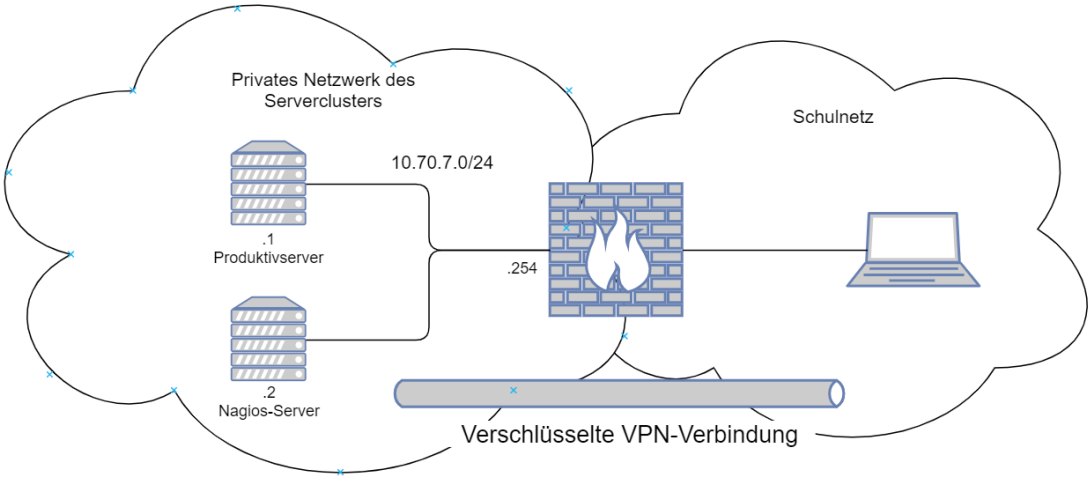


Abbildung 8.5: Ergänzter Netzwerkplan

8.1.8.2 Nagios

Worum es sich bei diesem tollen Tool handelt wurde bereits bei den installierten Paketen (siehe Kapitel 8.1.5.2, Seite 88) erklärt. Nun folgt ein vertiefender Einblick in diese Monitoring-Software.

Hosts Hosts sind bei Nagios definierte virtuelle Maschinen, die überwacht werden sollen. Die vorliegende Diplomarbeit überwacht nicht nur den Produktivserver, sondern auch den aufgesetzten Nagios-Server selbst. Falls dieser Probleme aufweist, wird das Team ebenfalls per E-Mail benachrichtigt. Dies wird in einem eigenen Punkt (siehe Kapitel 8.1.8.2, Seite 102) näher erläutert.

Im Webbrowser sehen die zu überwachenden Hosts wie folgt aus:

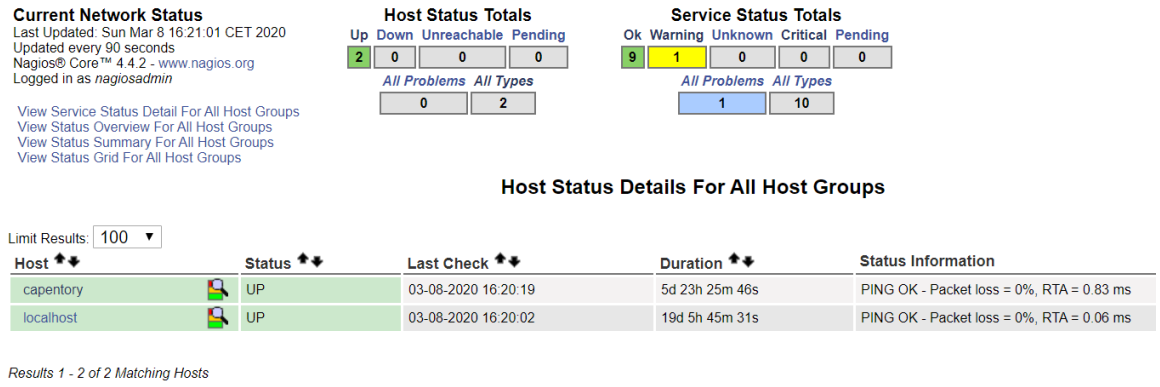


Abbildung 8.6: Die definierten Hosts der Diplomarbeit

Capentory ist der Name des Hosts des Produktivservers.

Localhost heißt der Host des Nagios-Servers.

Momentan scheint alles ohne Probleme zu funktionieren. Jedoch kann sich dies schlagartig ändern:

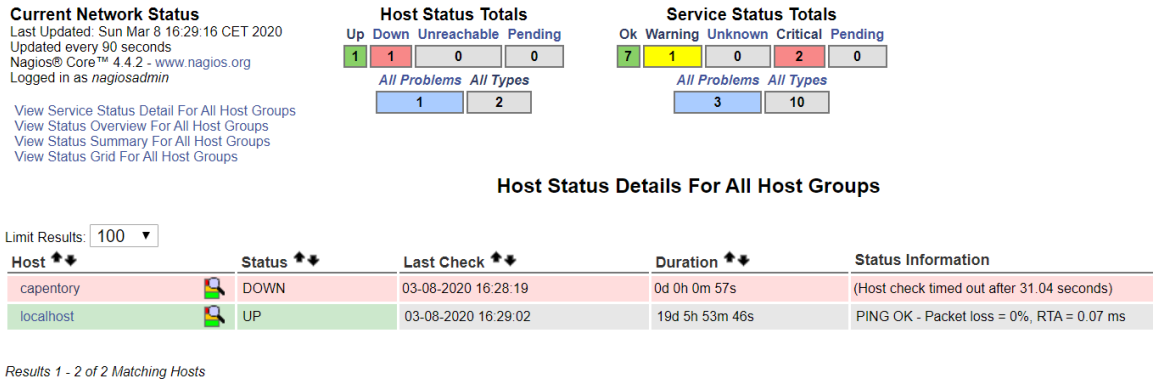


Abbildung 8.7: Der heruntergefahrene Produktivserver

Hier wird deutlich gemacht, dass der Produktivserver abgestürzt ist.

Services Nagios-Services sind, wie der Name schon verrät, die einzeln zu überwachenden Dienste eines Hosts. Zurzeit wird am Nagios-Server aus Testzwecken eine Menge an Services überwacht. Auf dem Produktivserver hingegen werden nur Probleme, die mit der Verbindung über Port 22 (SSH) oder Port 443 (HTTPS) zu tun haben, erfasst.

So sehen die überwachten Services am Admin-Dashboard des Nagios-Servers aus:

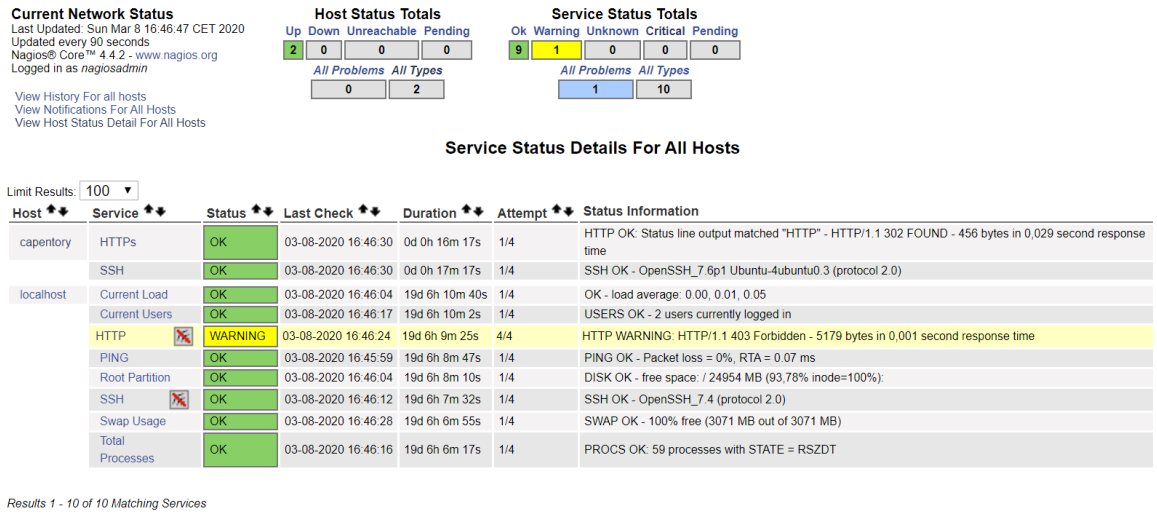


Abbildung 8.8: Die überwachten Services

Im Moment weist kein Service der beiden Hosts ein kritisches Problem auf. Auf dem Nagios-Server ist der HTTP-Service zwar gelb markiert, hierbei handelt es sich jedoch nur um eine harmlose Warnung.

Wenn auf dem Produktivserver hingegen der Webserver NGINX ausfällt, sieht das Dashboard jedoch wiederum anders aus:

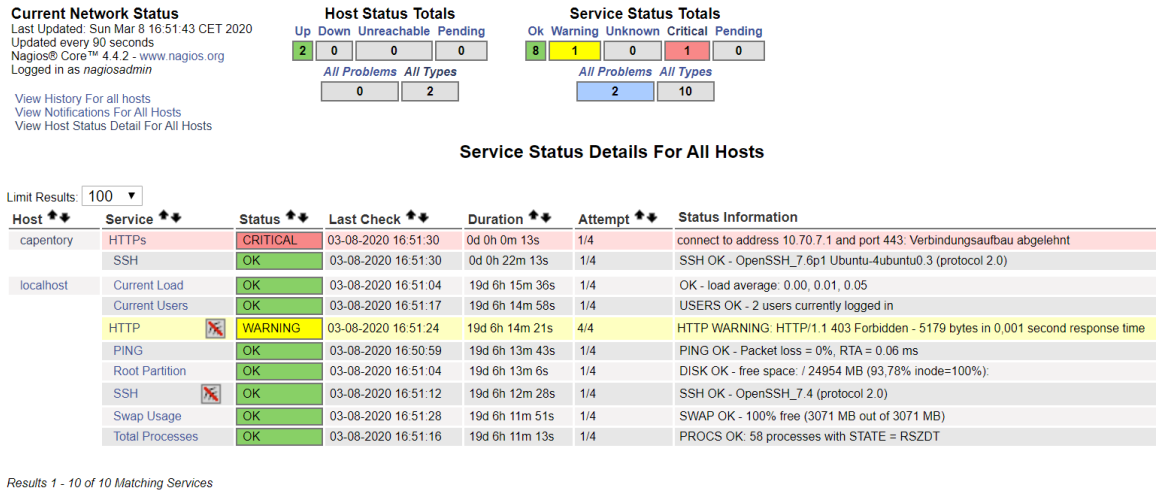


Abbildung 8.9: Die überwachten Services

Notifications Zuguterletzt gibt es noch das Feature der Notifications. Falls ein Host oder ein Service der Hosts ausfällt, benachrichtigt der Nagios-Server das Diplomarbeitsteam per E-Mail über die aufgetretenen Probleme.

8.1.9 Verfassen einer Serverdokumentation

Die Funktionsweise (siehe Kapitel 8.1.6, Seite 89) wurde bereits erklärt, allerdings war das letzte Ziel der Infrastruktur, den Installationsvorgang der Entwicklungs- sowie der Produktivumgebung zu dokumentieren. Somit kann nun jeder die vorliegende Inventurlösung verwenden und anhand der Serverdokumentation Schritt für Schritt selbständig aufsetzen. Die elf Seiten der Serverdokumentation sind jedoch nicht im Anhang beigelegt.

9 Planung

Die vorliegende Diplomarbeit hatte sehr vielen Stakeholder, die oftmals sehr viele Vorschläge oder Wünsche eingebracht haben. Um dies verwalten zu können, wurde diese Diplomarbeit mit einer agilen Projektmanagementmethode, die an SCRUM angelehnt war, durchgeführt. Die agile Arbeitsweise hat eine möglichst schnelle Reaktionszeit ermöglicht. Das zentrale Planungsdokument war hierbei der Backlog, der digital jederzeit einsehbar war. Ebenso wurden alle zwei Wochen Statusberichte ausgesandt. Weiters wurden insgesamt drei offizielle Probeinventuren mit einigen Stakeholdern durchgeführt.

A Anhang 1: Serverhandbuch

Das Serverhandbuch erläutert die Handhabung der graphischen Oberfläche des Servers mittels Bildschirmaufnahmen und geht in die Details der produktiven Administrationsumgebung ein. Es beinhaltet allgemeine Informationen bezüglich Layout, Benutzerverwaltung Importformaten. Zusätzlich wird die Architektur der beiden Server-Module in gekürzter Form beschrieben. Das Handbuch behandelt nur praktische Aspekte und beinhaltet keinen Programmcode oder sonstige programmiertechnische Details.

htl3 rennweg
IT & MECHATRONIK

Capentory

Handbuch zum Server

Projekttitel:	**Capentory**
Betreuer:	**DI Clemens Kussbach, DI August Hörandl**
Auftraggeber:	**DI Christian Schöndorfer**
Auftragnehmer:	**Josip Domazet (PL), Mathias Möller (PL Stv.), Hannes Weiss**
Schuljahr:	**2019/20** Klasse: **5CN**

VERSION	DATUM	AUTORIN/AUTOR	ÄNDERUNG
v1.0	28.01.2020	Mathias Möller	Erstellung des Dokuments

Inhalt

1 Grundlegendes

Das Serversystem, das im Rahmen der Diplomarbeit Capentory entwickelt wurde, ermöglicht die Verwaltung von Inventardaten und über die mobile Applikation durchgeführten Inventuren. Dieses Handbuch beschreibt die Bedienung der Weboberfläche des Servers. Die Bedienung der mobilen App wird in einem eigens dafür verfassten Handbuch erläutert. Zur besseren Lesbarkeit werden technische Details stark vereinfacht dargestellt.

Anmerkung: In den Abbildungen werden ausschließlich zufällig-generierte Daten verwendet.
Anmerkung: Das Serversystem basiert auf einer öffentlich verfügbaren Serverlösung namens „Ralph“ von dem Unternehmen Allegro, weshalb dieser Name des Öfteren im Webinterface auftaucht.

2 Systemanforderungen

Ein gängiger[1] Internetbrowser wird benötigt, um das Webinterface zu erreichen.

3 Erreichen der Administrationsoberfläche

Dazu öffnen Sie den Internetbrowser Ihrer Wahl und geben die Adresse des Capentory-Servers ein. Ist dieser erreichbar, werden Sie aufgefordert, sich einzuloggen:

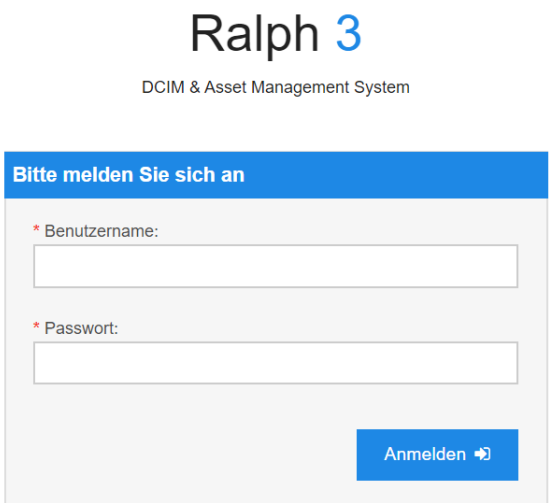


Abbildung 1: Der Login-Bereich fordert Sie auf, Ihren Benutzernamen und Passwort einzugeben.

[1] getestet wurden: Google Chrome, Firefox und Edge in ihrer aktuellsten Version (Stand: 28.01.2020)

htl3 rennweg
IT & MECHATRONIK

Capentory

4 Administration

4.1 Allgemeine Informationen zu Layout und Funktion

4.1.1 Startseite

Die Startseite ist von allen Seiten ausgehend durch Klick des "Ralph" Buttons (Abbildung 2 [1]) erreichbar. Hier werden die wichtigsten Tabellen/Seiten verlinkt, wie etwa die Tabelle aller Htl-Räume (Abbildung 2 [2]). Außerdem werden die letzten Aktionen des angemeldeten Benutzers angezeigt (Abbildung 2 [3]).

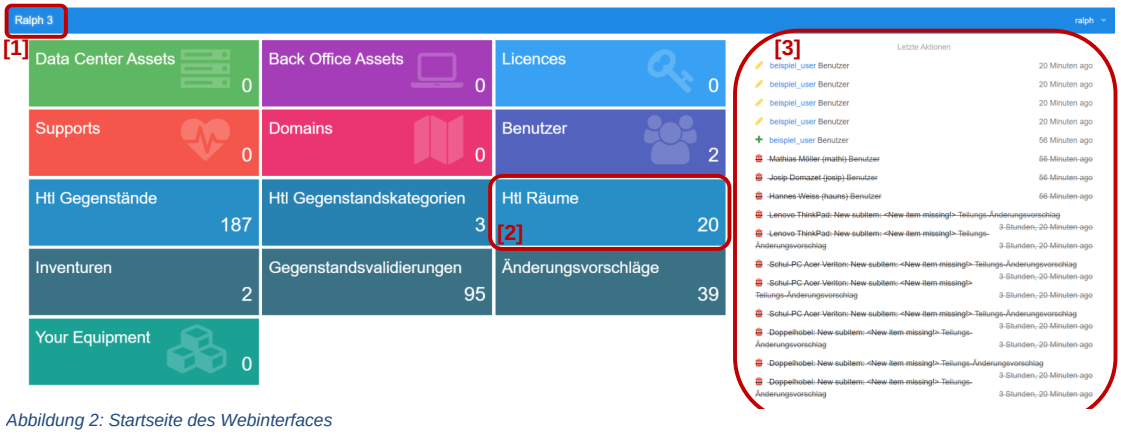


Abbildung 2: Startseite des Webinterfaces

4.1.2 Tabellenansichten

Die Tabellenansicht der Htl-Räume kann beispielsweise durch Klicken des Buttons in **Fehler! V erweisquelle konnte nicht gefunden werden.** [2] erreicht werden.

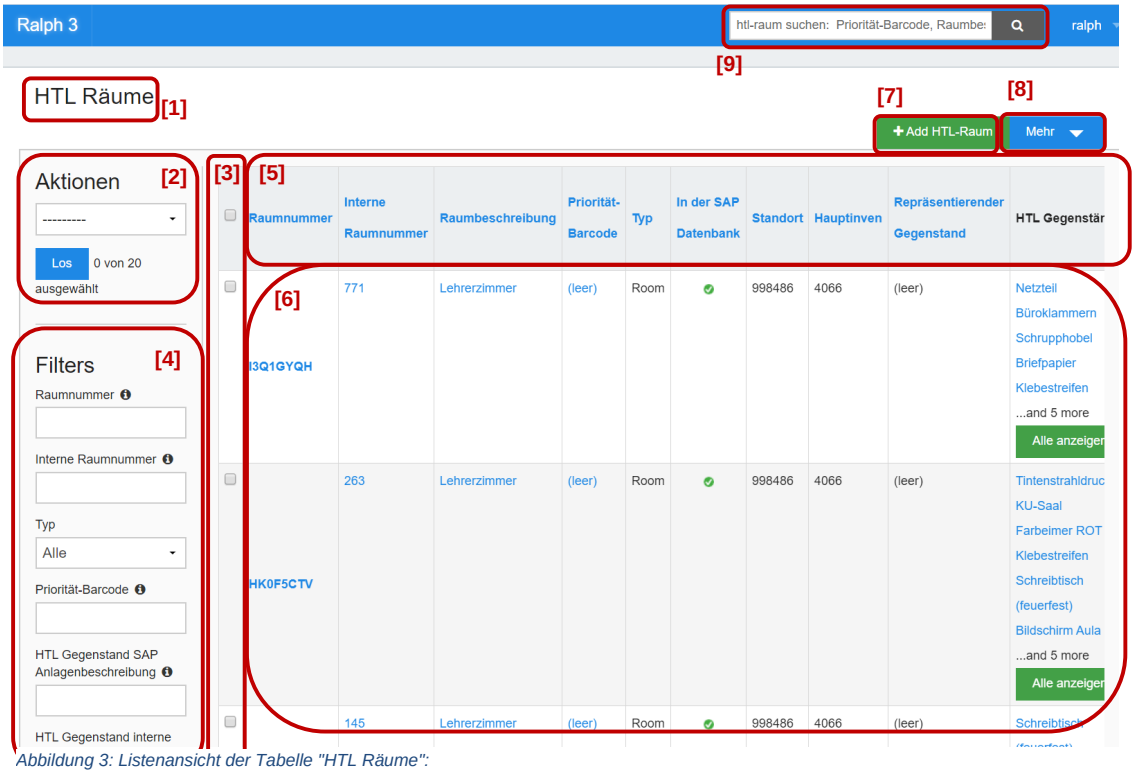


Abbildung 3: Listenansicht der Tabelle "HTL Räume":

Legende zu Abbildung 3:

[1]	Name der Tabelle
[2]	Aktionen: Hier können diverse Aktionen für die markierten Elemente ausgeführt werden. Standardmäßig können die ausgewählten Elemente gelöscht werden und Inventuränderungen angewendet werden. Es muss eine Aktion ausgewählt werden und dann auf „Los“ geklickt werden, um eine Aktion für alle markierten Elemente auszuführen.
[3]	Hier können die einzelnen Elemente für das Miteinbeziehen beim Ausführen von Aktionen ausgewählt werden. Die erste Checkbox wählt alle Elemente der aktuellen Seite aus (Standardmäßig enthält eine Seite 100 Elemente; zwischen mehreren Seiten kann am Ende einer Seite geblättert werden)
[4]	Filter: Hier können Gegenstände nach beliebigen Eigenschaften gefiltert werden. Um die ausgewählten Filter anzuwenden, klicken Sie „Filter“ am Ende des Abschnittes.

[5]	Die Überschriften der Tabelle: Sie repräsentieren meist Eigenschaften der einzelnen Elemente, oder auch Eigenschaften verknüpfter Elemente. Ist eine Überschrift blau markiert, kann nach der entsprechenden Eigenschaft mit einem Klick darauf sortiert werden. Erneutes Klicken kehrt die Sortierreihenfolge um.
[6]	Die einzelnen Zeilen / Elemente der Tabelle: Das sind die Eigenschaften der jeweiligen Elemente. Ist eine Eigenschaft blau markiert, kann durch Klicken auf die Detailansicht des Elements gewechselt werden.
[7]	Durch diesen Button kann ein neues Element erstellt werden.
[8]	Dieser Button ermöglicht u.a. den Import und Export von Daten, sowie das Wiederherstellen gelöschter Elemente.
[9]	Suchfunktion: Es können Elemente nach bestimmten Eigenschaften gesucht werden.

4.1.3 Detailansichten

Die Detailansicht eines HTL-Raumes kann beispielsweise durch Klicken eines blau markierten Buttons in Abbildung 3 [6] erreicht werden.

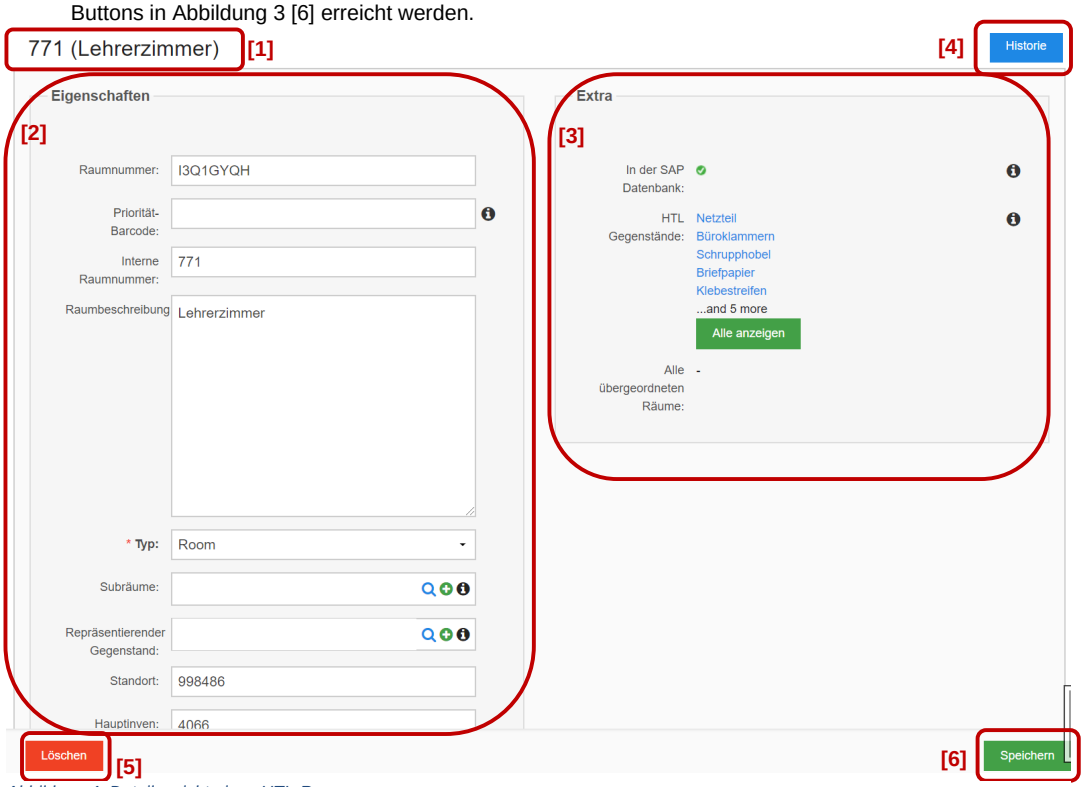


Abbildung 4: Detailansicht eines HTL-Raumes

Legende zu Abbildung 4:

[1]	Name des Elements (meist eine Kombination aus mehreren Eigenschaften)

[2]	Eigenschaften: Hier können die Eigenschaften eines Gegenstandes bearbeitet werden. Bei Referenzen auf Elemente aus anderen Tabellen kann mit dem Symbol die Liste aller Elemente zur Auswahl geöffnet werden, oder mit dem Symbol ein neues Element erstellt werden, welches beim Speichern sofort verlinkt wird.
[3]	Das ist ein weiterer Abschnitt der Elementeigenschaften. Die Detailansicht der HTL-Räume ist so gegliedert, dass links alle änderbaren Eigenschaften zu finden sind, rechts alle Eigenschaften, die sich aus div. Verknüpfungen ergeben oder nicht änderbar sind.
[4]	Historie: Hier können Änderungen rückgängig gemacht werden und auf ältere Versionen des Elements zurückgegriffen werden.
[5]	Löschen: Dieser Button ermöglicht die Löschung eines Elements (es kann später noch wiederhergestellt werden, siehe Abbildung 3 [8])
[6]	Speichern: Bei Klick dieses Buttons werden alle Änderungen, so wie sie in den Bereichen [2] und [3] erscheinen, übernommen und gespeichert. Hierbei können eventuell Fehler auftreten, auf die Sie hingewiesen werden (Beispielsweise eine ungültige Raumnummer, da diese bereits in Verwendung ist.).

4.1.4 Custom Fields

Manche Detailansichten ermöglichen es, Elementen bestimmte benutzerdefinierte Eigenschaften zu verleihen:

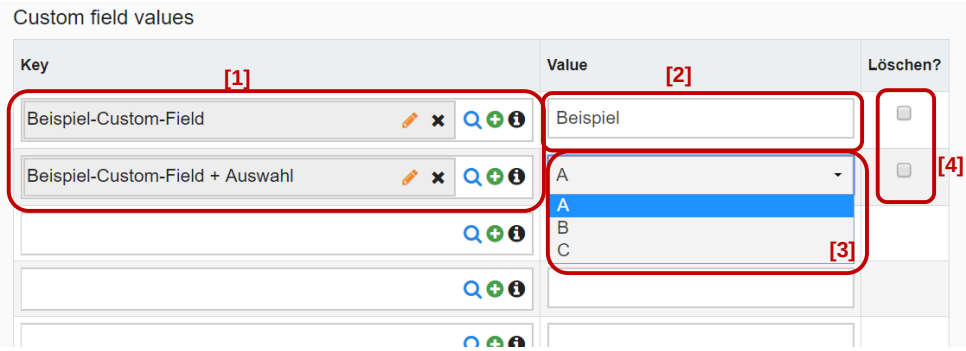


Abbildung 5: Custom-Fields

Für ein Element (wie z.B. ein HTL-Gegenstand) können beliebig Custom-Fields gesetzt werden. Dazu muss erst eine Custom-Field „Art“ bzw. „Typ“ (Abbildung 5 [1]) erstellt ( ) oder ausgewählt ( )werden. Je nachdem, wie der gewählte Custom-Field-Typ definiert wurde, kann eine bestimmte Art von Wert (Abbildung 5 [2]) festgelegt werden (Beispiele: Nummer, Zeichenkette, Datum). Zusätzlich können auch in der Definition der Custom-Field-Art Werte definiert werden, unter denen gewählt werden kann (Abbildung 5 [3]). Wenn die Checkbox in Abbildung 5 [4] angehakt ist, wird das Custom-Field für das Element (etwa den HTL-Gegenstand) beim nächsten Speichervorgang entfernt. Der Custom-Field-Typ bleibt erhalten.

4.2 Benutzereinstellungen

Die Benutzereinstellungen für den angemeldeten Benutzer können über die Schaltfläche rechts oben auf einer beliebigen Seite (Abbildung 6 [1]) erreicht werden:

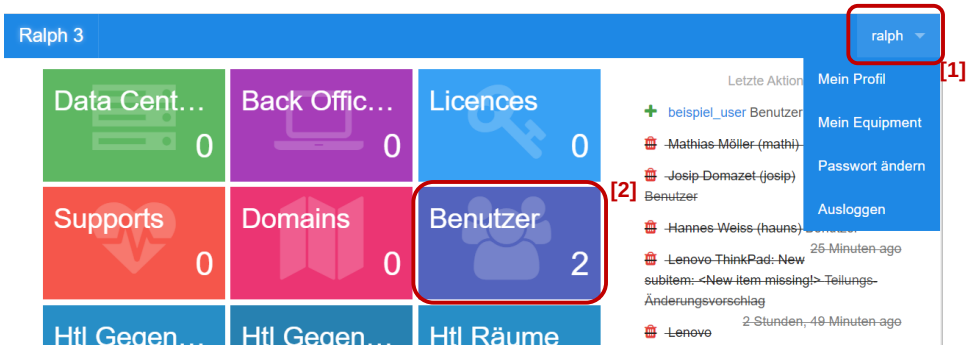


Abbildung 6: Die Startseite des Administrationsinterfaces. Unter dieser Schaltfläche können Sie unter "Mein Profil" Ihr Benutzerprofil bearbeiten oder sich unter "Ausloggen" vom Server abmelden.

Unter „Mein Profil“ haben Sie u.a. die Möglichkeit, ihre bevorzugte Sprache zu ändern. Alle von der Diplomarbeit „Capentory“ erstellten Elemente[2] sind in Deutsch, sowie Englisch verfügbar.

Unter „Benutzer“ (Abbildung 6 [2]) können von einem Administrator alle Benutzerkonten eingesehen und verändert werden. Besonders wichtig ist hierbei die Rechtevergabe.

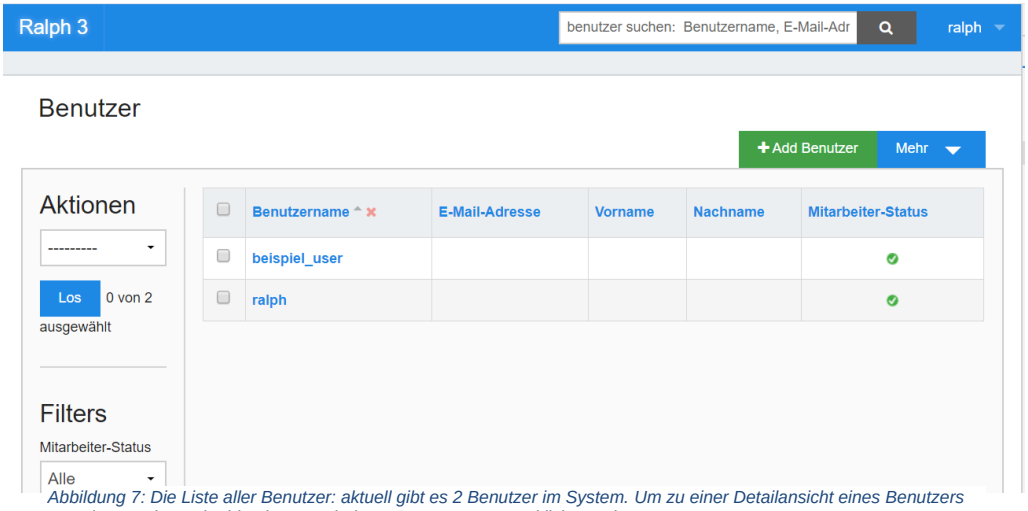


Abbildung 7: Die Liste aller Benutzer: aktuell gibt es 2 Benutzer im System. Um zu einer Detailansicht eines Benutzers zu gelangen, kann der blau hervorgehobene Benutzername geklickt werden.

[2] Das sind nicht alle Elemente des Systems, wie etwa einige Bezeichnungen in der Benutzerverwaltung selbst.

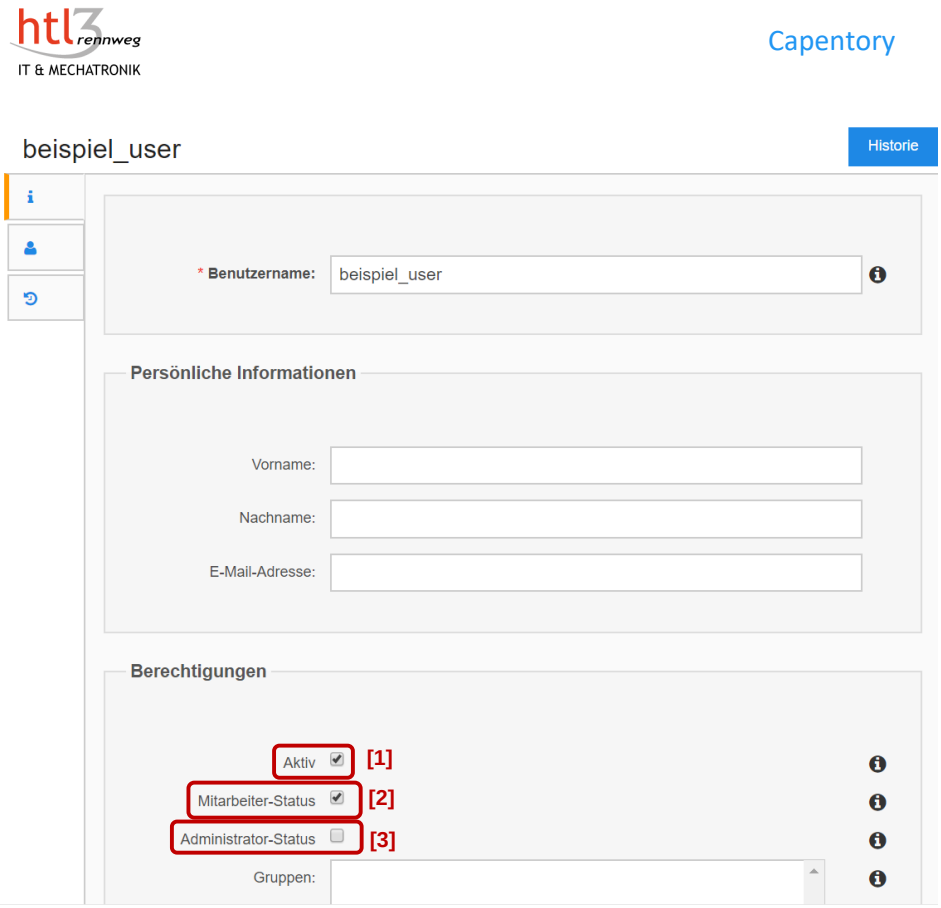


Abbildung 9: Detailansicht des Benutzers "beispiel_user"

Wichtig: Um einen Benutzer verwenden zu können muss „Aktiv“ (Abbildung 9 [1]) gesetzt sein! Um sich mit einem Benutzer anmelden zu können, muss „Mitarbeiter-Status“ (Abbildung 9 [2]) gesetzt sein! Weiters werden hier alle Berechtigungen des Benutzers auf per-Tabellen-Basis vergeben (siehe Abbildung 8), es sei denn, der Benutzer hat Administrator-Status (Abbildung 9 [3]). In diesem Fall hat der Benutzer alle Lese-/ und Schreibrechte.

Anmerkung: Man beachte, dass der Inhalt in Abbildung 8 *nicht übersetzt wird. Dies ist leider das Standardverhalten des Servers und nicht von externen Paketen (wie etwa dem Capentory-Paket) beeinflussbar.*

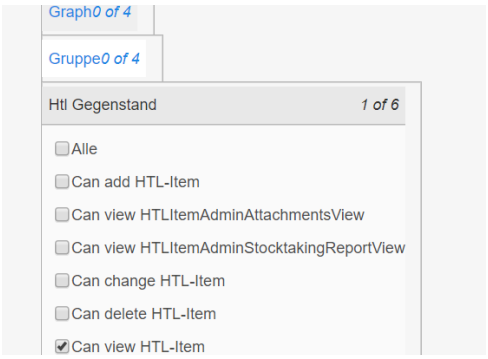


Abbildung 8: Dieser Benutzer darf die Tabelle "Htl Gegenstand" nur lesen. "HTLItemAdminAttachmentsView" und "HTLItemAdminStocktakingReportView" sind hierbei zusätzliche Ansichten eines Htl-Gegenstandes, die der Benutzer ebenfalls nicht lesen darf.

Zusätzlich können einem Benutzer Gruppen zugewiesen werden, woraufhin der Benutzer alle Berechtigungen der Gruppe erhalt. So kann beispielsweise eine Gruppe „Inventarisierer“ und „Inventar-Admins“ erstellt werden. Es wird empfohlen, für die Gruppe „Inventarisierer“ folgende Mindestberechtigungen zu vergeben:

Tabelle / Elementart	empfohlene Mindestberechtigungen
Attachment	Add, Change, View
Attachment Item	Add, Change, View
Custom Field	View
Custom Field Value	View
Gegenstandsvalidierung	Add, View
Htl Gegenstand	Add, View (optional: View HTLItemAdminAttachmentsView) (optional: View HTLItemAdminStocktakingReportView)
Htl Gegenstandskategorie	View
Htl Raum	View
Inventur	View (optional: View StocktakingAdminStocktakingReportView)
Inventur-Aktion	Add, View
Raumvalidierung	Add, View
Änderungsvorschlag	Add, View

„Inventar-Admins“ können auf alle o.a. Tabellen/Elementarten volle Zugriffsberechtigungen zugeteilt werden, bzw. gleich als „Administratoren“ (Abbildung 9 [3]) festgelegt werden.

Falls es nicht möglich ist, die Gruppeneigenschaften in der Detailansicht zu bearbeiten, können diese unter der URL [IP bzw. Name des Servers]/auth/group/ eingesehen und geändert werden.

Vergessen Sie nicht, nach dem Bearbeiten auf „Speichern“ zu klicken!

Abbildung 10: Ende einer Detailansicht-Seite mit "Löschen" und "Speichern" Buttons

5 Das HTL-Paket

Als „HTL-Paket“ stehen jegliche Funktionen rund um folgende Tabellen/Elementarten zur Verfügung:

- **HTL Gegenstände**
- **HTL Gegenstandskategorien**
- **HTL Räume**

Diese werden in diesem Kapitel erläutert.

5.1 HTL Gegenstände

5.1.1 Eigenschaften

Name der Eigenschaft	Beschreibung
Anlage [1] [2]	Die Anlagennummer aus SAP
UNr. [1] [2]	Das UNr. Feld aus SAP
BuKr [1]	Das BuKr Feld aus SAP (repräsentiert den Schulstandort)
Priorität-Barcode [3]	Der Barcode eines Gegenstandes, sollte er nicht mit der Anlage+Unr. übereinstimmen. (Anwendungsfall: Gegenstände, die nicht aus SAP importiert wurden, aber trotzdem einen Barcode haben.)
Anlagenbeschreibung	Die Anlagenbeschreibung aus SAP
Interne Gegenstandsbeschreibung	Eine „interne Anlagenbeschreibung“, die Priorität über jene aus SAP hat, aber nicht beim Export/Import berücksichtigt wird.
Raum	Ein HTL-Raum in dem sich der Gegenstand aktuell befindet, siehe Abschnitt „HTL Räume“
Gegenstand ist in der SAP Datenbank	Gibt Auskunft, ob der Gegenstand aus SAP importiert wurde oder aus einer anderen Quelle entstanden ist
Zeit des letzten Import von SAP	Wenn der Gegenstand aus SAP importiert wurde gibt diese Eigenschaft Auskunft, wann dies zuletzt geschehen ist

Kategorie	Eine HTL-Gegenstandskategorie, die dem Gegenstand zugewiesen ist, siehe Abschnitt „HTL Gegenstandskategorien“
Besitzer	Ein Benutzer im System, der als Besitzer des Gegenstandes gilt.
Sponsor	Ein Textfeld für die Spezifikation von Sponsor-Informationen
Akt.Invnr	Die akt. Invnr. aus SAP
Bish.Invnr	Die bish. Invnr. aus SAP
Anmerkung	Ein Textfeld für Anmerkungen
Schutzklasse	Die Schutzklasse des Gegenstandes (siehe: https://de.wikipedia.org/wiki/Schutzklasse_(Elektrotechnik))

Anmerkungen:
[1]: Diese 3 Eigenschaften identifizieren gemeinsam einen Gegenstand eindeutig. Es kann also nur ein HTL-Gegenstand gleichzeitig dieselbe Anlage, UNr. und BuKr haben. Leere Werte sind von dieser Regel ausgenommen.
[2]: Diese 2 Eigenschaften bilden den Barcode, sollte der Priorität-Barcode nicht gesetzt sein.
[3]: Dieser Barcode ist ähnlich wie die 3 Eigenschaften in [1] eindeutig. Ein Priorität-Barcode darf außerdem nicht mit einem in [2] definierten Barcode übereinstimmen (und vice versa).

5.1.2 Inventur-Report

Für einen einzelnen Gegenstand kann über die Schaltfläche „Inventur-Report“ bzw. „Stocktaking Report“ ein Bericht generiert werden:

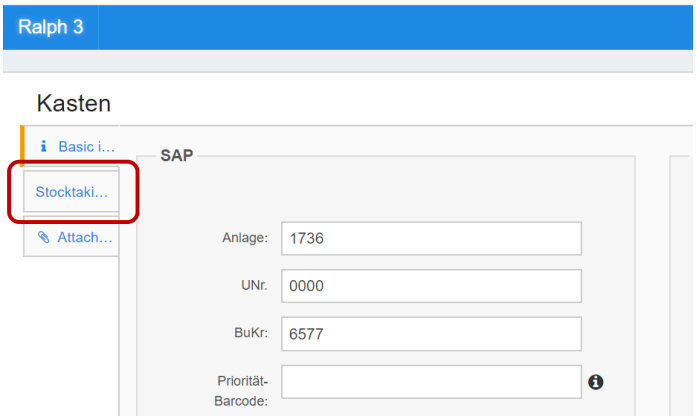


Abbildung 12: Inventur-Bericht Schaltfläche

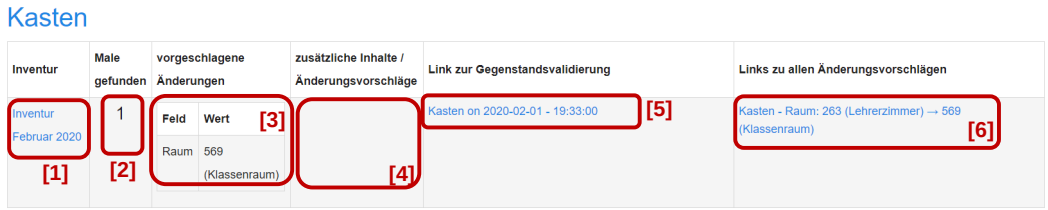


Abbildung 11: Inventur-Bericht eines Gegenstandes

Legende zu Abbildung 11

[1]	Name der Inventur, in der der Gegenstand mind. einmal gefunden wurde.
[2]	Wie oft der Gegenstand insgesamt während der Inventur gefunden wurde.
[3]	Die Änderungsvorschläge, die während der Inventur aufgetreten sind und erstellt wurden und sich auf Eigenschaften des Gegenstands beziehen.
[4]	Sonstige Änderungsvorschläge (Beispiel: „Der Gegenstand benötigt ein neues Etikett“)
[5]	Links zu den einzelnen Gegenstandsvalidierungen
[6]	Links zu allen Änderungsvorschlägen

5.1.3 Anhänge

Für einen HTL-Gegenstand können beliebig viele Anhänge gespeichert werden. Ein „Anhang“ ist eine ordinäre Datei, unter anderem können auch Bilder gespeichert werden, die auf der mobilen Applikation angezeigt, verändert und hinzugefügt werden können. Die Übersicht über alle Anhänge findet sich im „Attachments“ Untermenü eines HTL-Gegenstandes:

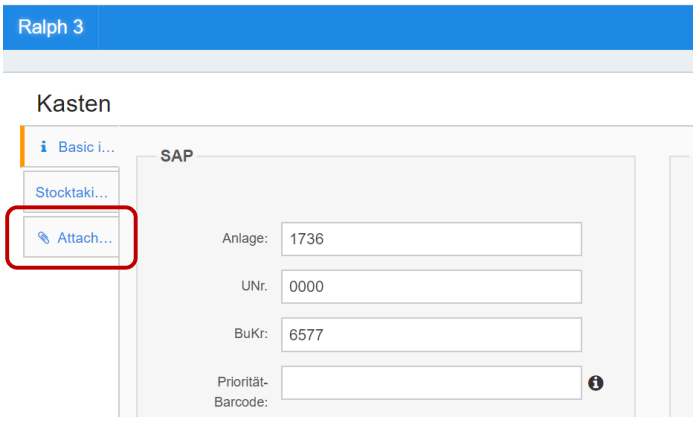


Abbildung 13: Anhang-Schaltfläche

5.1.4 Import

Daten können über die in Abbildung 3 [8] markierte Schaltfläche importiert werden. Dazu muss in einem Untermenü die zu importierende Datei und dessen Format gewählt werden.

Es gibt 4 Import-Funktionen für HTL-Gegenstände:

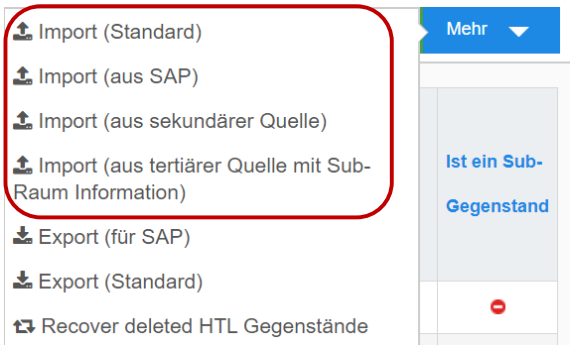


Abbildung 14: Import-Funktionen für HTL-Gegenstände

Für schulinterne Zwecke kann der „Import (Standard)“ vernachlässigt werden.

Es ist empfohlen, beim allerersten Import die angegebene Reihenfolge zu befolgen und noch vor dem Import von Gegenständen bereits die Raumliste zu importieren (siehe HTL-Räume – Import).

Bei jedem Importvorgang werden grundsätzlich die Attributnamen aus der importierten Datei auf datenbankinterne Attributnamen abgebildet. Wird ein Attributname der importierten Datei geändert, muss diese Änderung ebenfalls in der Datei `import_settings.py` vorgenommen werden, um die Attribute weiterhin abbilden zu können.

- **Import (aus SAP)**

Dies ist der Standard-Export von SAP, der in das Capentory-System aufgenommen werden kann. Damit können auch Daten im Capentory-System aktualisiert werden; sie müssen also nicht immer neu sein. Das Mindestformat sieht (anhand eines Beispieldatensatzes) wie folgt aus (zusätzliche Spalten werden ignoriert):

BuKr	Anlage	UNr.	Akt.Invnr	Anlagenbez	Hauptinven	Standort	Raum	Bemerkung	Bish.Invnr
1234	0987654321	9876	xxxxxxxxxx	Kasten	5656	787878	F4A7210		yyyyyyyyyy

Nach dem Einlesen der Datei werden Sie gebeten, die zu importierenden Daten zu bestätigen.
Verifizieren Sie die zu importierenden Daten bitte sorgfältig.

Die Vorschau des o.a. Beispieldatensatz sieht so aus:

Abbildung 15: SAP-Import Vorschau

Legende zu Abbildung 15:

[1]	Button zum Bestätigen des Imports
[2]	Vorschau aller HTL-Gegenstände
[3]	Vorschau aller HTL-Räume (Anmerkung: bei korrektem Import der Raumliste vor Import der SAP-Liste sollten keine neuen Räume erstellt werden müssen.

- **Import (aus sekundärer Quelle)**

Diese Import-Funktion ändert die HTL-Gegenstände nicht direkt, sondern erstellt eine Inventur mit Änderungsvorschlägen, die sich aus den Daten der angegebenen Liste und dem aktuellen Zustand der am Server gespeicherten Gegenstandsdaten ergeben. Außerdem werden beim Import vorgefertigte HTL-Gegenstandskategorien erstellt und diese in Änderungsvorschlägen den einzelnen HTL-Gegenständen zugewiesen.

Mindestformat ist (anhand eines Beispieldatensatzes, bezogen auf den o.a. Datensatz):

Anlage	**Anlagenbez**	**Raum**	**Raum-Lang**	**neus_Raumbuch**	Inventaraufkleber	**Standort**	**Infrastruktur**	**akt_Komponente**	**Telefon**	**Drucker**	**PC**	**Lehrerrechner**	**Schutzklasse**	**in_Domäne**	**WLAN**	**Anmerkung**
0987654321	Kasten	105	Klassenraum	F4A7210					Nein							

Mögliche Werte für:

Inventaraufkleber	OK, <Leer>

Infrastruktur	X, <Leer>
akt_Komponente	X, <Leer>
Telefon	Nein, X, <Leer>
Drucker	X, <Leer>
PC	X, <Leer>
Lehrerrechner	X, <Leer>
Schutzklasse	1, 2, 3, <Leer>
in_Domäne	Ja, Nein, X, <Leer>
WLAN	X, <Leer>

„X“ wird als „Ja“ bzw. „Wahr“ interpretiert, ein leeres Feld als „Nein“ bzw. „Falsch“
Nach dem Import können die Änderungen sofort angewendet werden. Der Inventur-Bericht der eben importierten Liste sieht wie folgt aus:

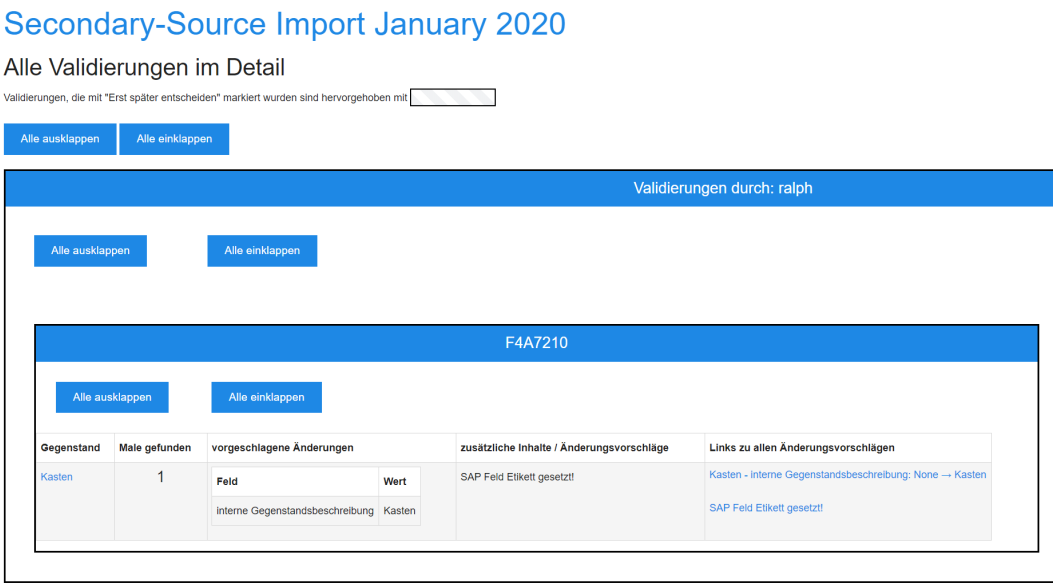


Abbildung 16: Inventur-Bericht der durch den Import erstellten Inventur. Eine Änderung der internen Gegenstandsbeschreibung wird vorgeschlagen, sowie ein Änderungsvorschlag erstellt, der signalisiert, dass der Gegenstand ein neues Etikett benötigt (da „Inventaraufkleber“ nicht gesetzt wurde.

- **Import (aus tertiärer Quelle (mit Sub-Raum Information)**

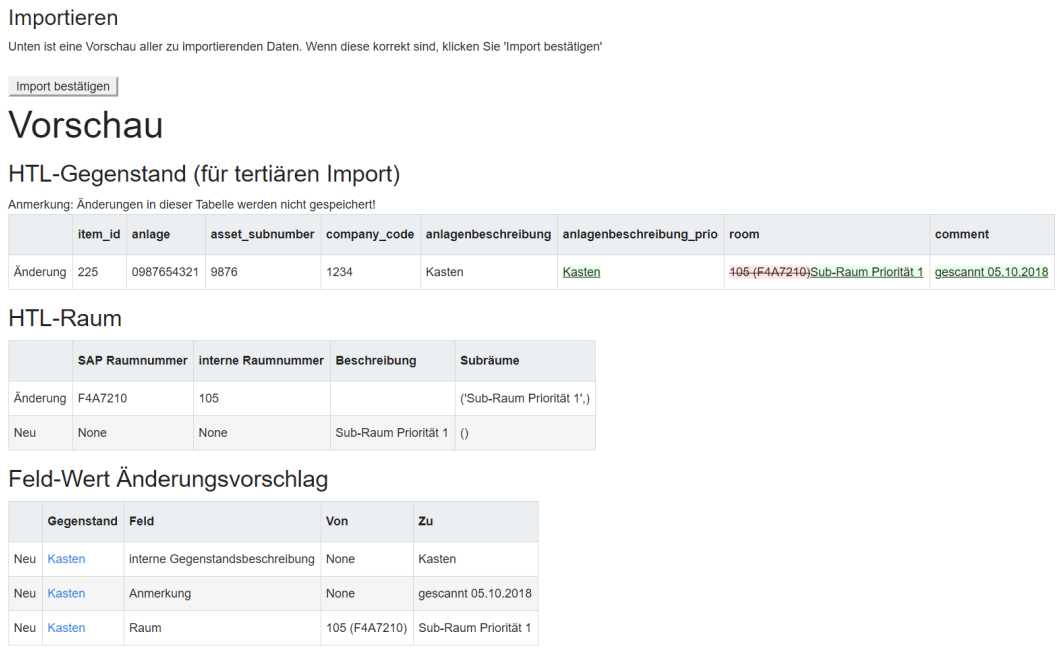
Diese Import-Funktion ändert die HTL-Gegenstände nicht direkt, sondern erstellt eine Inventur mit Änderungsvorschlägen, die sich aus den Daten der angegebenen Liste und dem aktuellen Zustand der am Server gespeicherten Gegenstandsdaten ergeben. Außerdem werden beim Import Sub-Räume erstellt und (bestenfalls bereits existierenden) HTL-Räumen untergeordnet. Bei den Sub-Räumen handelt es sich schlussendlich ebenfalls um HTL-Räume. Sie sind also in derselben Liste zu finden, wie die übergeordneten Räume.

Mindestformat ist (anhand eines Beispieldatensatzes, bezogen auf den o.a. Datensatz):

Anlage	Anlagenbez	Raum	o.Ä.	Etikett	Bish.Invnr	Überprüft durch
Sub-Raum Priorität 2						
0987654321	Kasten	105	Sub-Raum Priorität 1	Pickerl geklebt		gescannt 28.01.2020

Ab der 2. Zeile wird angenommen, dass sich weitere Gegenstände in „Sub-Raum Priorität 2“ befinden, sollte im Feld „o.A.“ kein Sub-Raum eingetragen sein, bis zu jenem Zeitpunkt, an dem eine weitere Zeile mit verbundenen Spalten angibt, um welchen Sub-Raum es sich handelt.

Da sich kein Gegenstand in „Sub-Raum Priorität 2“ befindet, wird dieser auch nicht erstellt:



Importieren

Unten ist eine Vorschau aller zu importierenden Daten. Wenn diese korrekt sind, klicken Sie 'Import bestätigen'

Import bestätigen

Vorschau

HTL-Gegenstand (für tertiären Import)

Anmerkung: Änderungen in dieser Tabelle werden nicht gespeichert!

	item_id	anlage	asset_subnumber	company_code	anlagenbeschreibung	anlagenbeschreibung_prio	room	comment
Änderung	225	0987654321	9876	1234	Kasten	Kasten	~~105 (F4A7210)~~Sub-Raum Priorität 1	gescannt 05.10.2018

HTL-Raum

	SAP Raumnummer	interne Raumnummer	Beschreibung	Subräume
Änderung	F4A7210	105		('Sub-Raum Priorität 1',)
Neu	None	None	Sub-Raum Priorität 1	()

Feld-Wert Änderungsvorschlag

	Gegenstand	Feld	Von	Zu
Neu	Kasten	interne Gegenstandsbeschreibung	None	Kasten
Neu	Kasten	Anmerkung	None	gescannt 05.10.2018
Neu	Kasten	Raum	105 (F4A7210)	Sub-Raum Priorität 1

Abbildung 17: Vorschau des tertiären HTL-Gegenstand-Imports

Der Inventur-Bericht der durch den Import erstellten Inventur würde so aussehen:

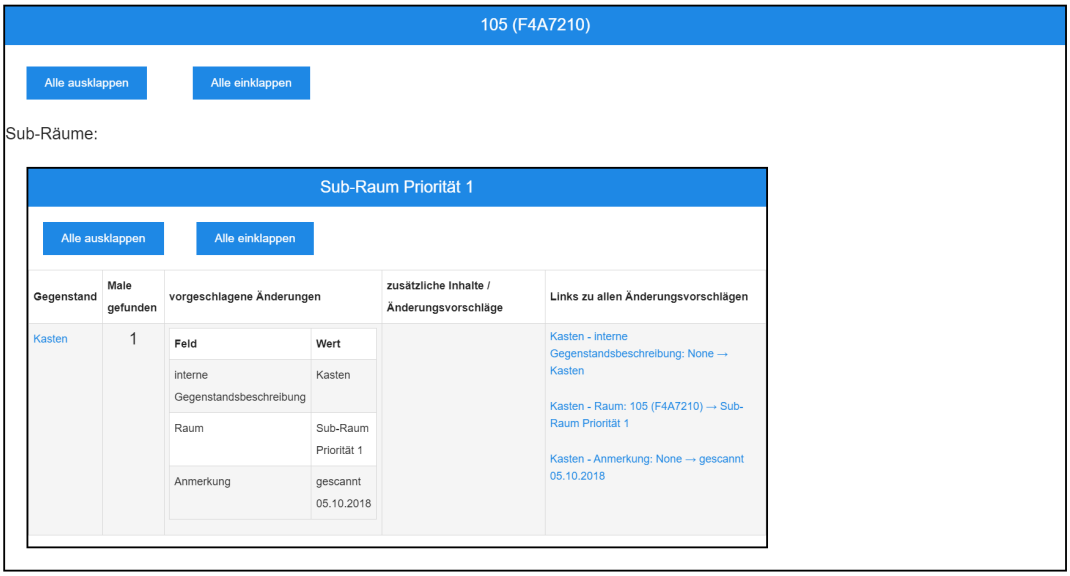


Abbildung 18: Abschnitt des Inventur-Berichtes für Raum „105 (F4A7210)“. Eine Änderung der internen Gegenstandsbeschreibung wird vorgeschlagen, sowie ein Änderungsvorschlag des Raumes auf den erstellten Sub-Raum und ein Änderungsvorschlag der Anmerkung

5.1.5 Export

Daten können über die in Abbildung 3 [8] markierte Schaltfläche exportiert werden. Dazu muss in einem Untermenü das Export-Format gewählt werden.

Es gibt 2 Export-Funktionen für HTL-Gegenstände:

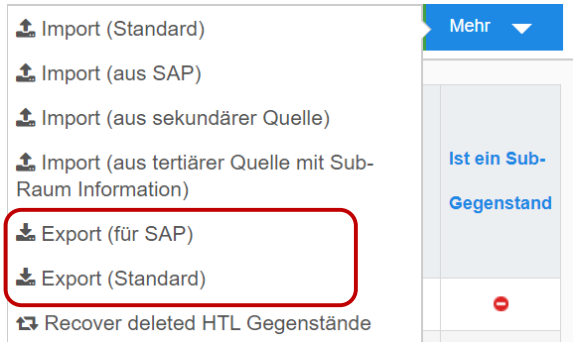


Abbildung 19: Import-Funktionen für HTL-Gegenstände

Für schulinterne Zwecke kann der „Export (Standard)“ vernachlässigt werden.

1. **Export (für SAP)**

Diese Export-Funktion dient zum Import der Datenänderungen in das SAP-System. Hierbei werden nur Gegenstände exportiert, die in den angegebenen Inventuren von den angegebenen Benutzern validiert wurden (Abbildung 20)

Abbildung 20: HTL-Gegenstands-Export Einstellungen

Neben den Standard-Formaten ist hier auch das Format „gezipptes xlsx (mit limitierter Dateigröße)“ verfügbar. Es limitiert die Zeilenanzahl einer exportierten Excel-Datei auf 100 und aggregiert alle dadurch erstellten Excel-Dateien in einen ZIP-komprimierten Ordner.

Beim Export für SAP werden 2 Bereiche berücksichtigt:

1. **Noch nicht angewendete Änderungsvorschläge der angegebenen Inventuren (hat Priorität; die jüngsten Änderungen werden miteinbezogen, bleiben aber „unangewendet“)**
2. **Der Zustand der HTL-Gegenstände direkt nach dem letzten Import aus SAP**

Dadurch ergibt sich eine Art „Delta“, welches beim Erstellen der Export-Dateien miteinbezogen wird.
Der Export, der durch die von den beiden Imports angefertigten Inventuren entsteht, sieht wie folgt aus (an die Seitenverhältnisse angepasst formatiert):

htl3 rennweg IT & MECHATRONIK

Capentory

BuKr	Anlage	UNr.	Akt.Invnr	Anlagenbez	Hauptinven	Standort	Raum	o.Ä.	Etikett	Verschrott	Verkauf	Unent.Über	SGÜ	HV-Transf.	VM-Transf.	Haupti.neu	Stand.neu	Raum_neu	Bemerkung	Bish.Invnr
1234	09876543 21	9876	xxxxxxxx xx	Kasten	5656	787878	F4A7210												gescannt 05.10.201 8	yyyyyyyy yy

Wie erkennbar ist, hat sich der Raum des Gegenstandes nicht verändert, da er nur in einen Sub-Raum des in SAP eingetragenen Raumes „gewandert“ ist. Zusätzlich ist „o.Ä“ (=“ohne Änderung“) nicht gesetzt, da sich die Bemerkung geändert hat.
Außerdem ist „Etikett“ ebenfalls nicht gesetzt, da die jüngste Inventur (=“tertiärer Import“) keinen entsprechenden Änderungsvorschlag spezifiziert. Das wird als „das Etikett wurde zwischen der Inventur „sekundärer Import“ und „tertiärer Import“ geklebt und wird nun nicht mehr benötigt“ interpretiert.

5.2 HTL-Gegenstandskategorien

5.2.1 Eigenschaften

Name	Beschreibung
Kategorie	Eine Kurzbeschreibung der Kategorie
Beschreibung	Eine ausführliche Beschreibung der Kategorie

HTL-Gegenstandskategorien besitzen ebenfalls die Fähigkeit, Custom-Fields zu beherbergen (siehe Kapitel „Custom Fields“). Besonders ist hierbei, dass jegliche HTL-Gegenstände einer Kategorie immer mindestens alle Custom-Fields der Kategorie beherbergen. Als Standardwert für die einzelnen HTL-Gegenstände wird der Wert der Kategorie herbeigezogen. Beispielsweise hat also eine Kategorie „Switch“ das Custom-Field „Anzahl Ports“ mit dem Wert 0. Automatisch haben dann alle HTL-Gegenstände der Kategorie „Switch“ das Custom-Field „Anzahl Ports“. Jeder Gegenstand hat einen unabhängigen Wert. Sollte kein Wert eingetragen worden sein, hat ein HTL-Gegenstand die „Anzahl Ports“ 0.

5.3 HTL-Räume

5.3.1 Eigenschaften

Name	Beschreibung
Raumnummer [1] [2]	Die SAP-Raumnummer
Priorität-Barcode [3]	Der Barcode eines Gegenstandes, sollte er nicht mit der Raumnummer übereinstimmen. (Anwendungsfall: Eigens angepasste Raum-Barcodes)
Interne Raumnummer	Eine intern geführte Raumnummer, die primär zur Repräsentation des Raumes dient

Raumbeschreibung	Eine Beschreibung des Raumes (z.B. „Klassenraum“)
Typ	Der Typ eines Raumes, für die Abgrenzung zwischen Sub-Raum und übergeordnetem Raum (z.B. Raum oder Kasten)
Subräume	Alle untergeordneten Räume dieses Raumes. Das sind HTL-Räume von beliebigem Typen, also beispielsweise Kästen oder Räume. Diese müssen in einer hierarchischen Beziehung zueinander sein (Sollte das nicht der Fall sein wird eine Warnung ausgegeben).
Repräsentierender Gegenstand	Ein Raum kann durch einen Gegenstand repräsentiert werden, dessen Barcode dann automatisch zum Barcode des Raumes wird, wenn der Priorität-Barcode nicht gesetzt ist. Beispielsweise kann ein Kasten gleichzeitig ein Gegenstand, wie auch ein (Sub-) Raum sein.
Standort [1]	Das SAP-Standort Feld
Hauptinven [1]	Das SAP-Hauptinven Feld
In der SAP Datenbank	Gibt Auskunft, ob der Raum (auch) aus SAP importiert wurde oder aus einer anderen Quelle entstanden ist.
HTL Gegenstände	Links zu allen Gegenständen in diesem Raum, exklusive Sub-Räumen
Alle übergeordneten Räume	Links zu allen übergeordneten Räumen, die diesen Raum als Sub-Raum verlinkt haben (oder als Sub-Sub-Raum, etc.)

Anmerkungen:
[1]: Diese 3 Eigenschaften identifizieren gemeinsam einen Raum eindeutig. Es kann also nur ein HTL-Raum gleichzeitig dieselbe „Raumnummer“, „Standort“ und „Hauptinven“ haben. Leere Werte sind von dieser Regel ausgenommen.
[2]: Diese Eigenschaft bildet den Barcode, sollte der Priorität-Barcode nicht gesetzt sein. (Dadurch können theoretisch auch mehrere Räume denselben Barcode haben, da die Raumnummer allein nicht einzigartig ist. Praktisch ist das für einen Standort allerdings nicht der Fall!)
[3]: Dieser Barcode ist ähnlich wie die 3 Eigenschaften in [1] eindeutig. Ein Priorität-Barcode darf außerdem nicht mit einem in [2] definierten Barcode übereinstimmen (und vice versa).

5.3.2 Import

Grundsätzlich sollten die HTL-Räume vor allen anderen Daten importiert werden, da nicht in allen Listen die benötigten Informationen enthalten sind, um interne Bezeichnung mit SAP-Bezeichnung zu verknüpfen. Daher sollte so früh wie möglich diese „Verknüpfung“ entstehen. Dazu können Raumdaten über die in Abbildung 3 [8] markierte Schaltfläche importiert werden. Das Format sieht dabei wie folgt aus:

Raum_ALT	Raum_Bezeichnung	Raum_Neu
110	Klassenraum	J7E2377

Raum_ALT ist hierbei die interne Raumnummer, Raum_Bezeichnung ist die Raumbeschreibung und Raum_Neu die SAP-Raumnummer.

Die Eigenschaften „Standort“ und „Hauptinven“, sowie „In der SAP Datenbank“ werden bei einem HTL-Gegenstand-Import aus SAP ergänzt bzw. aktualisiert.

6 Das Inventur-Paket

Als „Inventur-Paket“ stehen jegliche Funktionen rund um folgende Tabellen/Elementarten zur Verfügung:

- **Inventuren**
- **Inventur-Aktionen**
- **Raumvalidierungen**
- **Gegenstandsvalidierungen**
- **Änderungsvorschläge („Change Proposals“)**

Der Einfachheit halber werden in diesem Kapitel nur die wichtigsten Aspekte einer Inventur behandelt. Der Großteil der Inventur-, Inventur-Aktionen, etc. -objekte werden vom System automatisch anhand der Daten, die von der mobilen Client-Applikation empfangen werden, generiert. Theoretisch ist es möglich, eine vollständige Inventur inkl. Validierungen manuell über das Webinterface zu erstellen. Dies ist allerdings etwas umständlich und wird daher nicht in diesem Benutzerhandbuch behandelt.

6.1 Übersicht

Um eine Kurzübersicht aller Aspekte des Inventur-Pakets zu erhalten werden in der u.a. Tabelle alle Tabellen des Pakets kurz erläutert:

Tabellen-/Elementname	Beschreibung	Besitzt keine, eine oder mehrere…
Inventur	Die Basis einer Inventur.	Inventur-Aktionen
Inventur-Aktion	Repräsentiert alle Validierungen eines Benutzers während einer Inventur. URL zur Listenansicht (, da diese nicht über Startseite erreichbar ist): [IP bzw. Name des Servers] /stocktaking/stocktakinguseractions/	Raumvalidierungen
Raumvalidierung	Repräsentiert einen Raum, in der ein Benutzer während einer Inventur Gegenstände validiert hat und verweist auf den validierten Raum. Grundsätzlich kann jede Datenbanktabelle als „Raum“ definiert werden. Es hängt aber an der Implementierung der Client-Schnittstelle ab, welche Raumarten der mobilen Applikation zur Verfügung gestellt werden. URL zur Listenansicht (, da diese nicht über Startseite erreichbar ist): [IP bzw. Name des Servers] /stocktaking/stocktakingroomvalidation/	Gegenstandsvalidierungen

Gegenstandsvalidierung	Repräsentiert das Scannen eines Gegenstandes in einem Raum durch einen Benutzer während einer Inventur und verweist auf den gescannten Gegenstand. **Grundsätzlich kann jede Art von Gegenstand validiert werden. Es gibt keine Beschränkung auf HTL-Gegenstände.** Es hängt aber an der Implementierung der Client-Schnittstelle ab, welche Gegenstandstypen durch die mobile Applikation validiert werden können.	Änderungsvorschläge
Änderungsvorschläge	Besagt grundsätzlich, dass der Zustand des Gegenstandes in der verknüpften Gegenstandsvalidierung in der Datenbank falsch ist und eine Änderung benötigt. Dazu gibt es 4 relevante Änderungsvorschlag-Typen.	-

Es gibt 4 Arten/Typen von Änderungsvorschlägen:

1. **Feld-Wert Änderungsvorschlag:**
   Besagt, dass der Wert einer konkreten Eigenschaft (Bsp.: Kommentar, Raum, Gegenstandskategorie) des Gegenstandes geändert werden soll.
   Beispiel: „Der Wert des „Kommentarfeldes“ von Gegenstandes X soll von „ohne Schäden“ auf „hat eine Delle“ geändert werden.
2. **SAP-Änderungsvorschlag**
   Besagt, dass ein gewisses Feld des HTL-Gegenstandes beim Export (für SAP) gesetzt werden soll. Die Felder sind „Etikett“, „Verschrott“ und „Verkauf“
3. **Mehrfachvalidierung-Änderungsvorschlag**
   Besagt, dass ein Gegenstand mehrfach in einem Raum gescannt wurde.
   Für interne Gegenstände, die nicht aus SAP stammen, deutet es darauf hin, dass diese in mehrere, eigenständige Gegenstände geteilt werden sollten. Für Gegenstände, die aus SAP stammen, die physikalisch aber mehr als ein einziger Gegenstand, je mit eigenem aber demselben Barcode, sind dient es rein zu Informationszwecken. (Beispiel: ein 10er-Pack Telefone ist in SAP ein einziger Gegenstand, physikalisch existieren aber 10 eigenständige Telefone mit demselben Barcode)
4. **Teilungs-Änderungsvorschlag**
   Verweist auf einen neuen Gegenstand, der entweder:
   - ein neuer Gegenstand ist, den ein Benutzer während einer Inventur erstellt hat.
   - ein Sub-Gegenstand eines übergeordneten Gegenstandes ist (dieser Fall wird aktuell nicht von der Client-Schnittstelle erstellt und stattdessen durch einen Mehrfachvalidierung-Änderungsvorschlag ersetzt. Theoretisch ist dieser Fall aber durch einen Administrator produzierbar.

6.2 Inventuren

6.2.1 Eigenschaften

Name	Beschreibung
Name	Ein beliebiger Inventur-Name, z.B. „Inventur Februar 2020“
Benutzer / User	Der hauptverantwortliche Benutzer einer Inventur
Anmerkungen	Ein Textfeld für Anmerkungen / Kommentare zur Inventur als Ganzes
Startdatum	Das Datum, an dem die Inventur begonnen hat (Wird automatisch zu jenem Zeitpunkt, an dem die Inventur erstellt wurde)
Startzeit	Die Uhrzeit, zu der die Inventur begonnen hat (Wird automatisch zu jenem Zeitpunkt, an dem die Inventur erstellt wurde)
Enddatum	Ein Datum, ab dem keine Validierungen mehr getätigt werden dürfen und somit das Ende der Inventur signalisiert.
Endzeit	Eine Uhrzeit, ab der keine Validierungen mehr getätigt werden dürfen und somit das Ende der Inventur signalisiert.
Inventur endet nie	Wenn gesetzt, wird die Inventur zu einer „laufenden Inventur“. Das bedeutet: - Es kann derselbe Raum mehrmals validiert werden. - Das Enddatum und die Endzeit spielen keine Rolle.
Alle Validierungen	Zeigt die Links zu allen Gegenstandsvalidierungen, die in dieser Inventur getätigt wurden.

6.2.2 Inventur-Bericht

Über die Schaltfläche „Inventur-Bericht“ bzw. „Stocktaking Report“ in der Detailansicht einer Inventur kann ein interaktiver Inventur-Bericht generiert werden:

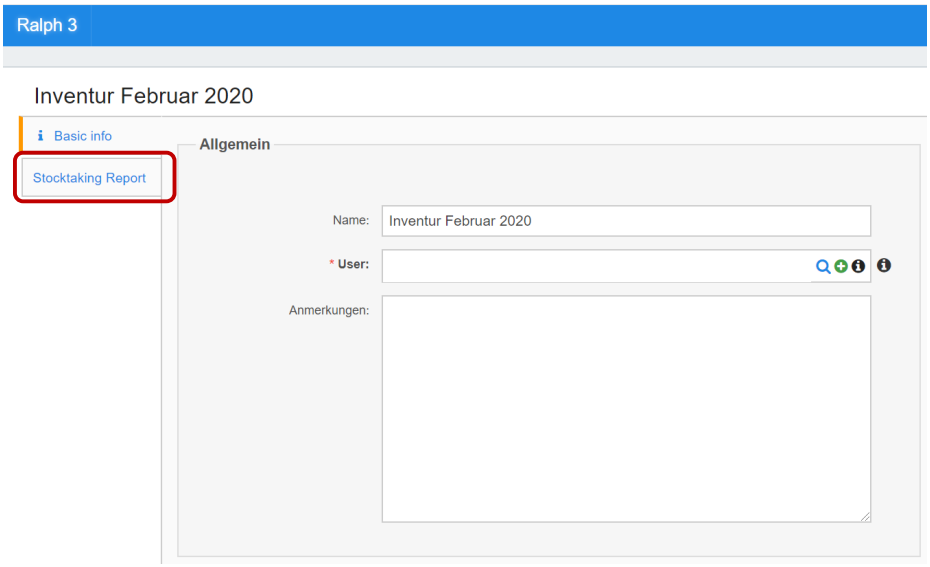


Abbildung 21: "Stocktaking Report" Schaltfläche in der Detailansicht einer Inventur

Ein Inventur-Bericht sieht beispielsweise wie folgt aus:

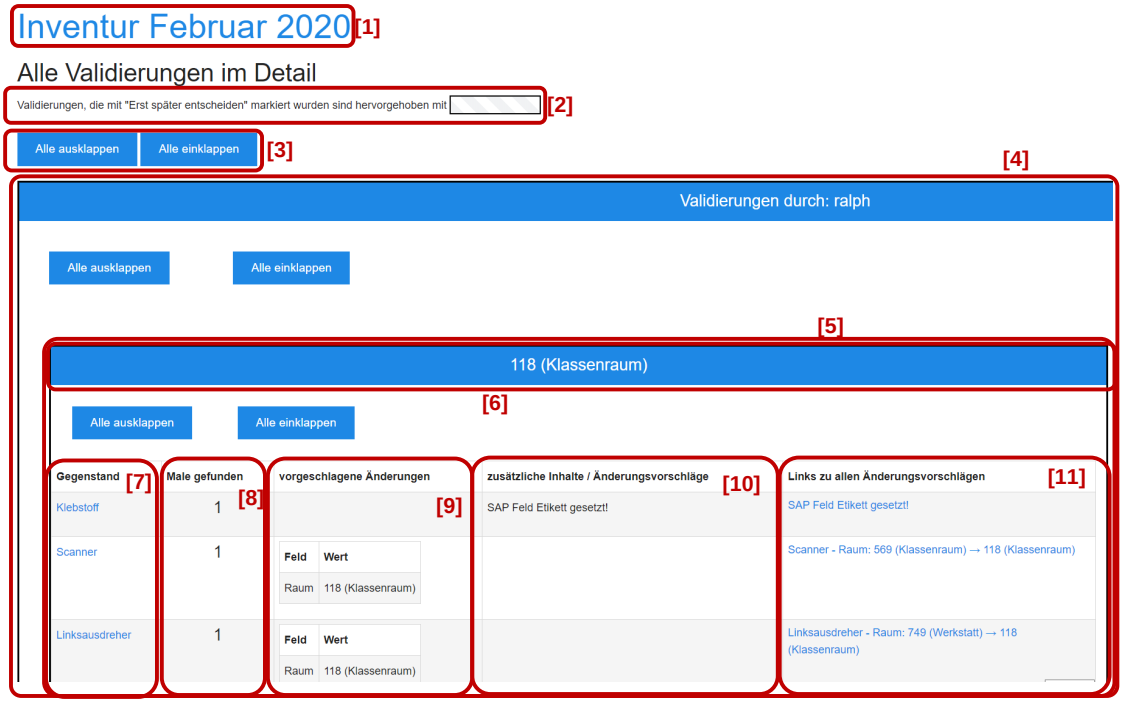


Abbildung 22: Inventur- Bericht

Legende zu Abbildung 22:

[1]	Name der Inventur (gleichzeitig auch Link zur Detailansicht der Inventur)
[2]	Alle Validierungen, die in der mobilen Applikation mit „erst später entscheiden“ markiert wurden, werden in einem schraffierten Stil ( ) hinterlegt.
[3]	Mit diesen Schaltflächen können alle Inventur-Aktionen [4] und Raum-Validierungen [5] „ausgeklappt“ und geladen werden.
[4]	Die Schaltfläche zum Ein- und Ausklappen einer Inventur-Aktion eines bestimmten Benutzers.
[5]	Der Gesamtbereich einer Raum-Validierung. Hier sind alle Informationen zu Gegenstandsvalidierungen aufgelistet, sowie Informationen zu nicht gefundenen Gegenständen und Validierungen in einem Sub-Raum.
[6]	Die Schaltfläche zum Ein- und Ausklappen einer Raumvalidierung, die durch den Benutzer in [4] vorgenommen wurden.
[7]	Alle gefundenen Gegenstände (gleichzeitig auch Link zur Detailansicht der Gegenstände)
[8]	Wie oft ein Gegenstand in dieser Inventur in diesem Raum gefunden wurde.

[9]	Eine Zusammenfassung aller Änderungsvorschläge, die sich direkt auf die Eigenschaften eines Gegenstandes beziehen.
[10]	Informationen zu zusätzlichen Änderungsvorschlägen (Beispiel: „Der Gegenstand benötigt ein neues Etikett“)
[11]	Links zu allen Änderungsvorschlägen. Wollen Sie einen Änderungsvorschlag löschen, können Sie dies machen, indem Sie auf den entsprechenden Link klicken, und den Änderungsvorschlag in seiner Detailansicht löschen. Den neuen Tab/Das neue Fenster, das beim Klick auf den Link entsteht, können Sie nach dem Löschen wieder schließen. Beachten Sie, dass diese Änderung erst im Bericht ersichtlich wird, wenn dieser neu geladen wird. Dazu können Sie auch nur den Bereich der Raum-Validierung [5] durch Klick auf [6] ein- und wieder ausklappen.

6.2.3 Löschen einer Inventur

Wird eine Inventur gelöscht, so werden alle mit der gelöschten Inventur verknüpften Elemente (Inventur-Aktionen, Raumvalidierungen, Gegenstandsvalidierungen und Änderungsvorschläge) ebenso gelöscht. Um eine Inventur vollständig wiederherzustellen müssen alle verknüpften Elemente einzeln wiederhergestellt werden. Dabei muss die Objekt-Hierarchie beachtet werden (Wiederherstellungsreihenfolge: Inventur – Inventur-Aktionen - Raumvalidierungen – Gegenstandsvalidierungen -Änderungsvorschläge)

6.2.4 Anwenden von Änderungen einer Inventur

Da die Änderungen von Eigenschaften eines Gegenstandes, die auf der mobilen Applikation vorgenommen werden, nicht sofort übernommen, sondern vorerst als Änderungsvorschläge gespeichert werden, muss irgendwann ein Administrator mit Schreibrechten auf die Gegenstandstabelle (etwa HTL-Gegenstände) die Änderungen anwenden. Dies ist über 2 Vorgehensweisen möglich:

1. Anwenden auf Inventurbasis
2. Anwenden auf Gegenstandsbasis

Anmerkung: Es werden nur Feld-Wert Änderungsvorschläge berücksichtigt!

Um alle Änderungsvorschläge einer oder mehrerer Inventuren anzuwenden, wählen Sie die gewünschten Inventuren in der Listenansicht aus (Abbildung 23 [2]), wählen Sie die Aktion „Änderungsvorschläge anwenden!“ (Abbildung 23 [1]) und klicken „Los“.

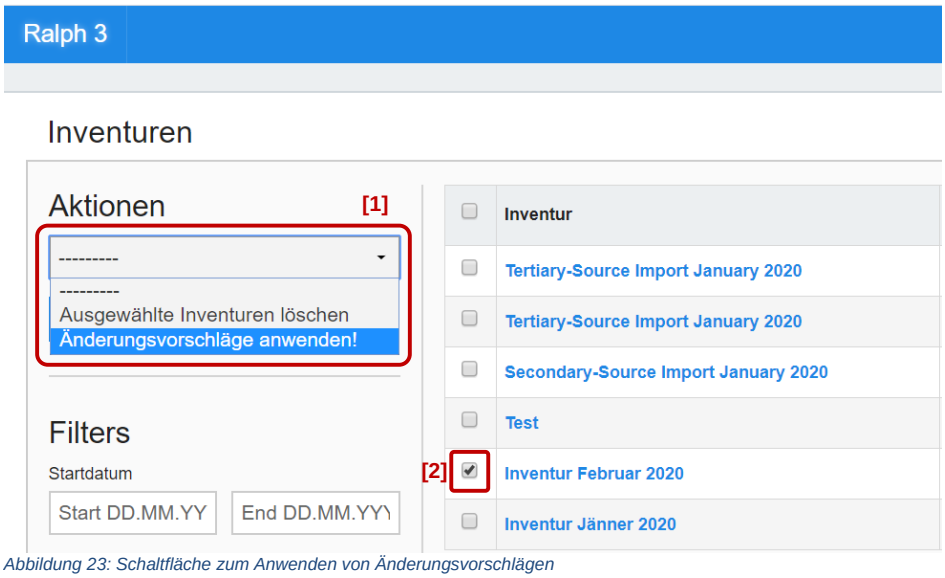


Abbildung 23: Schaltfläche zum Anwenden von Änderungsvorschlägen

Um alle Änderungsvorschläge bestimmter Gegenstände anzuwenden, wählen Sie die gewünschten Gegenstände in der Listenansicht aus (wie in Abbildung 23 [2]), wählen Sie die Aktion „Änderungsvorschläge anwenden!“ (wie in Abbildung 23 [1]) und klicken „Los“.

Sie gelangen dadurch auf eine Zwischenseite, auf der Sie diverse Parameter zur Anwendung der Änderungsvorschläge anpassen können. Diese Zwischenseite zeigt auch alle Änderungen, die angewendet werden:

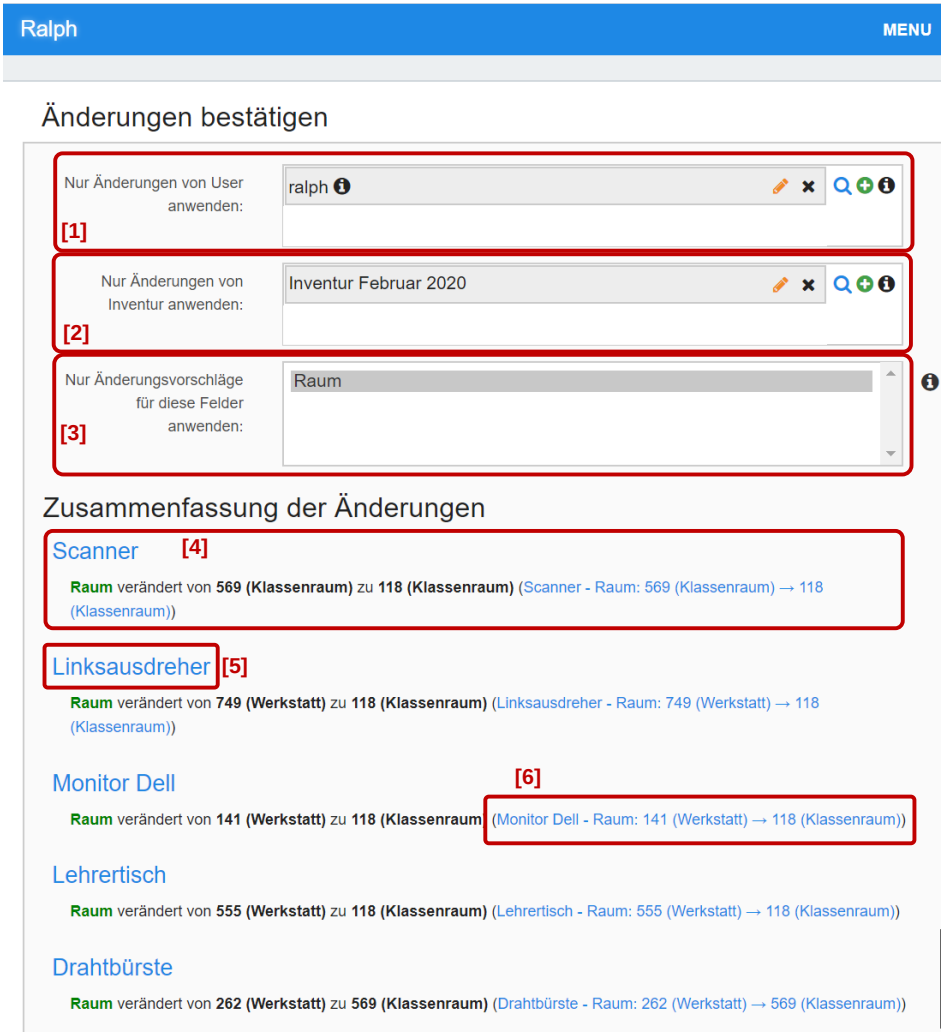


Abbildung 24: Bestätigung der Anwendung von Änderungsvorschlägen

Legende zu Abbildung 24:

[1]	Hier können Sie die Anwendung auf Änderungsvorschläge, die von bestimmten Benutzern erstellt wurden, beschränken. Standardmäßig sind alle Benutzer, die zu den Inventuren beigetragen haben, ausgewählt.
[2]	Hier können Sie die Anwendung auf Änderungsvorschläge von bestimmten Inventuren beschränken (nützlich für die Anwendung auf Gegenstandsbasis). Standardmäßig sind alle Inventuren, in der der Gegenstand/die Gegenstände vorkommt/vorkommen, ausgewählt.
[3]	Hier können Sie die Anwendung auf Änderungsvorschläge bestimmter Felder beschränken. Standardmäßig sind alle Felder, die zu mindestens einer Änderung führen, ausgewählt.
[4]	Die Zusammenfassung der Änderungen für einen Gegenstand.
[5]	Der Link zu einem Gegenstand. Über diesen Link gelangen Sie zur Detailansicht eines Gegenstandes.
[6]	Der Link zu dem beschriebenen Änderungsvorschlag. Über diesen Link gelangen Sie zur Detailansicht des Änderungsvorschlages und können ihn dort auch löschen.

Sollte es dazu kommen, dass mehrere Änderungsvorschläge dieselbe Eigenschaft eines Gegenstandes ändern würden, wird dies als Warnung rot markiert:

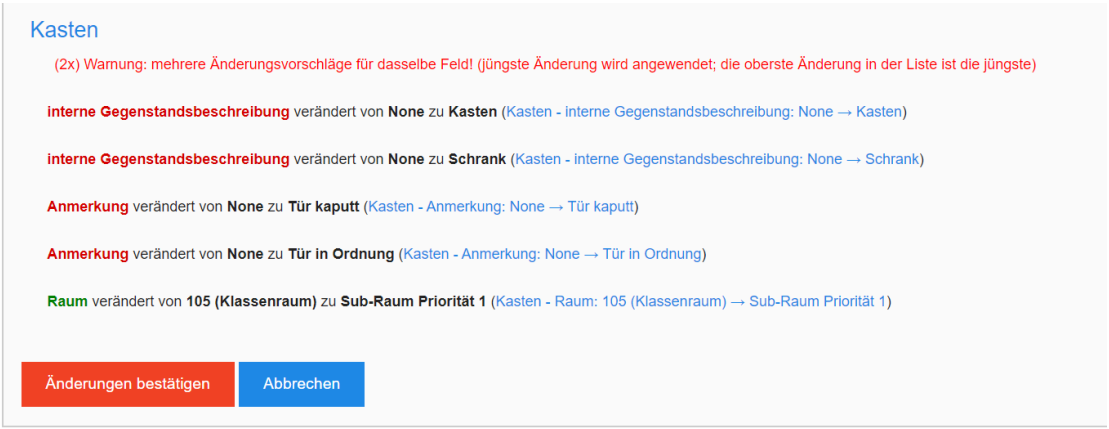


Abbildung 25: Warnungen bei Anwenden von Änderungsvorschlägen: In diesem Fall wird das Endresultat von dem jeweils obersten Änderungsvorschlag bestimmt!

Wenn Sie mit den angeführten Änderungen zufrieden sind, Klicken Sie „Änderungen bestätigen“ am Ende der Seite (Abbildung 25). Sollten Fehler beim Anwenden auftreten, werden Sie durch einen rot hinterlegten Warnbereich benachrichtigt und alle bereits angewendeten Änderungen werden zurückgesetzt.

Bei weiteren Fragen schreiben Sie bitte eine E-Mail an mathias.moeller@htl.rennweg.at oder josip.domazet@htl.rennweg.at

Frohes Inventarisieren wünscht Capentory!

Index

Literaturverzeichnis

[1] *Android Architecture Components.* https://developer.android.com/topic/libraries/architecture, Abruf: 2020-02-09

[2] *Android Volley Library.* https://developer.android.com/training/volley/request, Abruf: 2020-02-09

[3] *Anteile der weltweiten mobilen Betriebssysteme.* https://gs.statcounter.com/os-market-share/mobile/worldwide, Abruf: 2020-01-08

[4] *Artikel zu God Activity Architecture (Sarkastisch gehaltener Artikel!).* https://medium.com/@taylorcase19/god-activity-architecture-one-architecture-to-rule-them-all-62fcd4c0c1d5, Abruf: 2020-01-08

[5] *Begriffserkärung Native App.* https://de.ryte.com/wiki/Native_App, Abruf: 2020-01-08

[6] *Überblick von Django auf der offiziellen Website.* https://www.djangoproject.com/start/overview/, Abruf: 2020-01-01

[7] *Broadcasts - Offizielle Dokumentation.* https://developer.android.com/guide/components/broadcasts, Abruf: 2020-03-03

[8] *Callbacks.* https://stackoverflow.com/questions/824234/what-is-a-callback-function/7549753#7549753, Abruf: 2020-02-10

[9] *Canonical.* https://de.wikipedia.org/wiki/Canonical, Abruf: 2020-03-15

[10] *Cheat-Sheet.* https://netzwerktechnik.htl.rennweg.at/~zai/Fachschule/NWT_Stamm_Foerderkurs/Survival_Guide_Linux_Network_Configuration.pdf, Abruf: 2020-03-15

[11] *Configuration Changes - Offizielle Dokumentation.* https://developer.android.com/guide/topics/resources/runtime-changes, Abruf: 2020-02-10

[12] *Context - Offizielle Beschreibung von Android.* https://developer.android.com/reference/android/content/Context, Abruf: 2020-02-10

[13] *Datacenter-Insider Webseite mit Informationen über CMDB.* https://www.datacenter-insider.de/was-ist-eine-configuration-management-database-cmdb-a-743418/, Abruf: 2020-01-02

[14] *DataWedge - Offizielle Dokumentation.* https://www.zebra.com/content/dam/zebra_new_ia/en-us/solutions-verticals/product/Software/Mobility%20Software/datawedge/spec-sheet/dwandroid-specification-sheet-en-us.pdf, Abruf: 2020-03-03

[15] *Definition von "Syntax" im Duden.* https://www.duden.de/rechtschreibung/Syntax, Abruf: 2020-01-02

[16] *Demo-Webseite des Ralph-Systems von Allegro (Login-Daten: Benutzername: ralph/ Passwort: ralph).* https://ralph-demo.allegro.tech/, Abruf: 2020-01-02

[17] *Django Admin-Dokumentation (Django Version 1.8).* https://docs.djangoproject.com/en/1.8/ref/contrib/admin/, Abruf: 2020-01-01

[18] *Django ContentTypes-Dokumentation (Django Version 1.8).* https://docs.djangoproject.com/en/1.8/ref/contrib/contenttypes/, Abruf: 2020-02-07

[19] *Django Dateinamen-Nomenklatur.* https://streamhacker.com/2011/01/03/django-application-conventions/, Abruf: 2020-01-01

[20] *Django Datenbank-Dokumentation (Django Version 1.8).* https://docs.djangoproject.com/en/1.8/ref/databases/, Abruf: 2020-01-01

[21] *Django Datenbank-Modell-Dokumentation (Django Version 1.8).* https://docs.djangoproject.com/en/1.8/topics/db/models/, Abruf: 2020-01-01

[22] *Django Dokumentation von klassenbasierten Views (Django Version 1.8).* https://docs.djangoproject.com/en/1.8/topics/class-based-views/, Abruf: 2020-01-02

[23] *Django Model-Instance-Dokumentation (Django Version 1.8).* https://docs.djangoproject.com/en/1.8/ref/models/instances/, Abruf: 2020-02-05

[24] *Django Model-Options-Dokumentation (Django Version 1.8).* https://docs.djangoproject.com/en/1.8/ref/models/options/, Abruf: 2020-01-01

[25] *Django Query-Dokumentation (Django Version 1.8).* https://docs.djangoproject.com/en/1.8/topics/db/queries/, Abruf: 2020-01-01

[26] *Django Queryset-Dokumentation (Django Version 1.8).* https://docs.djangoproject.com/en/1.8/ref/models/querysets/, Abruf: 2020-01-01

[27] *Django Security-Dokumentation (Django Version 1.8).* https://docs.djangoproject.com/en/1.8/topics/security/, Abruf: 2020-02-16

[28] *Django Signal-Dokumentation (Django Version 1.8).* https://docs.djangoproject.com/en/1.8/topics/signals/, Abruf: 2020-02-05

[29] *Django Template-Dokumentation (Django Version 1.8).* https://docs.djangoproject.com/en/1.8/topics/templates/, Abruf: 2020-01-01

[30] *Django URL-Dokumentation (Django Version 1.8).* https://docs.djangoproject.com/en/1.8/topics/http/urls/, Abruf: 2020-02-09

[31] *Django View-Dokumentation (Django Version 1.8).* https://docs.djangoproject.com/en/1.8/topics/http/views/, Abruf: 2020-01-02

[32] *Do not repeat yourself.* https://deviq.com/don-t-repeat-yourself/, Abruf: 2020-02-10

[33] *Dokumentation der HTTP Methoden.* https://restfulapi.net/http-methods/, Abruf: 2020-02-09

[34] *Dokumentation des JSON-Formats.* json.org/json-de.html, Abruf: 2020-02-09

[35] *Dokumentation von Django-Reversion (Version 1.8).* https://django-reversion.readthedocs.io/en/release-1.8/, Abruf: 2020-03-05

[36] *Dokumentation zu Django-Extensions: Graph-Models.* https://django-extensions.readthedocs.io/en/latest/graph_models.html, Abruf: 2020-01-02

[37] *Eigenschaften: CentOS vs. Ubuntu.* https://thishosting.rocks/centos-vs-ubuntu-server/, Abruf: 2020-03-22

[38] *Eingriff in den Lifecycle.* https://stackoverflow.com/questions/19219458/fragment-on-screen-rotation, Abruf: 2020-02-10

[39] *Enums sollten vermieden werden.* https://android.jlelse.eu/android-performance-avoid-using-enum-on-android-326be0794dc3, Abruf: 2020-02-10

[40] *Erster Beitrag im Entwicklungsforum von Ralph.* https://ralph.discourse.group/t/ralph-inventory-extension/131, Abruf: 2020-03-05

[41] *Fakten rund um Flutter.* https://en.wikipedia.org/wiki/Flutter_(software), Abruf: 2020-01-08

[42] *Flutter.* https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/, Abruf: 2020-01-08

[43] *Flutter.* https://flutter.dev/, Abruf: 2020-01-08

[44] *Fragments - Offizielle Dokumentation.* https://developer.android.com/guide/components/fragments, Abruf: 2020-02-10

[45] *Funktionsweise von uWSGI.* https://www.vndeveloper.com/django-behind-uwsgi-nginx-centos-7/, Abruf: 2020-03-15

[46] *Generics - Offizielle Dokementation.* https://docs.oracle.com/javase/tutorial/java/generics/types.html, Abruf: 2020-03-14

[47] *Google - State Wrapper.* https://github.com/android/architecture-components-samples/blob/master/GithubBrowserSample/app/src/main/java/com/android/example/github/vo/Resource.kt, Abruf: 2020-02-10

[48] *Google unterstützt offiziel Single-Activity-Apps.* https://android-developers.googleblog.com/2018/05/use-android-jetpack-to-accelerate-your.html?m=1, Abruf: 2020-01-08

[49] *Internet-Posting über die Funtkion von REST-APIs.* https://www.cloudcomputing-insider.de/was-ist-eine-rest-api-a-611116/, Abruf: 2020-01-01

[50] *ISO-639.* https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes, Abruf: 2020-02-09

[51] *Java 8 in Android Studio.* https://developer.android.com/studio/write/java8-support, Abruf: 2020-02-10

[52] *Konfiguration uWSGI mit DJANGO und NGINX.* https://uwsgi-docs.readthedocs.io/en/latest/tutorials/Django_and_nginx.html, Abruf: 2020-03-22

[53] *Kotlin offiziel von Google präferiert.* https://www.heise.de/developer/meldung/I-O-2019-Googles-Bekenntnis-zu-Kotlin-4417060.html, Abruf: 2020-01-09

[54] *Lambdas in Java.* https://www.geeksforgeeks.org/lambda-expressions-java-8/, Abruf: 2020-02-10

[55] *Lifecycle eines ViewModels im Vergleich zu Activities - Offizielles Bild.* https://developer.android.com/images/topic/libraries/architecture/viewmodel-lifecycle.png, Abruf: 2020-03-23

[56] *LiveData - Offizielle Dokumentation.* https://developer.android.com/topic/libraries/architecture/livedata, Abruf: 2020-02-10

[57] *Mark Shuttleworth.* https://de.wikipedia.org/wiki/Mark_Shuttleworth, Abruf: 2020-03-15

[58] *Mobile Vision API- Offizielle Dokementation.* https://developers.google.com/vision/android/barcodes-overview, Abruf: 2020-03-04

[59] *Mobile Vision API Text Recognition - Offizielle Dokementation.* https://developers.google.com/vision/android/text-overview, Abruf: 2020-03-04

[60] *MVVM Artikel.* https://proandroiddev.com/mvvm-architecture-viewmodel-and-livedata-part-1-604f50cda1, Abruf: 2020-02-09

[61] *MVVM nach Google, die Grafik wurde für den gegebenen Fall vereinfacht.* https://developer.android.com/topic/libraries/architecture/images/final-architecture.png, Abruf: 2020-03-22

[62] *Offiziele Google-Dokementation zu App-Architekturen.* https://developer.android.com/jetpack/docs/guide, Abruf: 2020-02-09

[63] *Offizielle Design-Grundlagen der Django-Entwicker.* https://docs.djangoproject.com/en/dev/internals/contributing/writing-code/coding-style/, Abruf: 2020-01-02

[64] *Die offizielle Django-Website.* https://www.djangoproject.com/, Abruf: 2020-01-01

[65] *Offizielle Dokumentationsseite der Ralph Admin-Klasse von Allegro.* https://ralph-ng.readthedocs.io/en/stable/development/admin/, Abruf: 2020-01-02

[66] *Offizielle Dokumentationsseite der Ralph-API von Allegro.* https://ralph-ng.readthedocs.io/en/stable/development/api/, Abruf: 2020-01-02

[67] *Offizielle Dokumentationsseite des Paginierungs-Feature im Django Framework.* https://docs.djangoproject.com/en/3.0/topics/pagination/, Abruf: 2020-01-02

[68] *Offizielle Dokumentationsseite für Python Dekoratoren.* https://wiki.python.org/moin/PythonDecorators, Abruf: 2020-01-02

[69] *Die offizielle Flask-Website.* https://palletsprojects.com/p/flask/, Abruf: 2020-01-01

[70] *Offizielle Infopage des Django-REST Frameworks.* https://www.django-rest-framework.org/, Abruf: 2020-01-01

[71] *Die offizielle Pyramid-Website.* https://trypyramid.com/, Abruf: 2020-01-01

[72] *Die offizielle Web2Py-Website.* http://www.web2py.com/, Abruf: 2020-01-01

[73] *Die offizielle Website von "Ralph" des Unternehmens "Allegro".* https://ralph.allegro.tech/, Abruf: 2020-01-01

[74] *OWASP-Weiseite mit Informationen über CSRF.* https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF), Abruf: 2020-01-01

[75] *preforking.* https://stackoverflow.com/questions/25834333/what-exactly-is-a-pre-fork-web-server-model, Abruf: 2020-03-15

[76] *RecyclerView - Offizielle Dokementation.* https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView, Abruf: 2020-03-14

[77] *Red-Hat.* https://de.wikipedia.org/wiki/Red_Hat_Enterprise_Linux, Abruf: 2020-03-15

[78] *Singletons.* https://de.wikibooks.org/wiki/Muster:_Java:_Singleton, Abruf: 2020-03-15

[79] *Stackoverflow - Boilerplate Code.* https://stackoverflow.com/questions/3992199/what-is-boilerplate-code, Abruf: 2020-02-10

[80] *StackOverflow - State Wrapper.* https://stackoverflow.com/questions/44208618/how-to-handle-error-states-with-livedata, Abruf: 2020-02-10

[81] *TC56 Spezifikationen.* https://www.zebra.com/us/en/products/spec-sheets/mobile-computers/handheld/tc51-tc56.html, Abruf: 2020-03-03

[82] *ViewModel Anti-Patterns.* https://medium.com/androiddevelopers/viewmodels-and-livedata-patterns-antipatterns-21efaef74a54, Abruf: 2020-02-10

[83] *ViewModel Process Death.* https://developer.android.com/topic/libraries/architecture/viewmodel-savedstate, Abruf: 2020-02-10

[84] *ViewModels in Android.* https://developer.android.com/topic/libraries/architecture/viewmodel, Abruf: 2020-02-10

[85] *ViewPager2 - Offizielle Dokementation.* https://developer.android.com/guide/navigation/navigation-swipe-view-2, Abruf: 2020-03-04

[86] *WEBP.* https://developers.google.com/speed/webp, Abruf: 2020-03-04

[87] *Weiterer Vergleich Native vs. Hybrid vs. Web-App.* https://mlsdev.com/blog/native-app-development-vs-web-and-hybrid-app-development, Abruf: 2020-02-08

[88] *Wikipedia-Artikel zu MVVM.* https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel, Abruf: 2020-02-09

[89] *Xamarin.* https://apiko.com/blog/flutter-vs-xamarin-the-complete-2019-developers-guide-infographics-included/, Abruf: 2020-01-08

[90] *Xamarin - Offizielle Beschreibung.* https://dotnet.microsoft.com/learn/xamarin/what-is-xamarin, Abruf: 2020-02-09

[91] *Xamarin in Visual Studio.* https://visualstudio.microsoft.com/de/xamarin/, Abruf: 2020-02-09

[92] *Zebra Homepage.* https://www.zebra.com/de/de.html, Abruf: 2020-03-23

[93] *Zweiter Beitrag im Entwicklungsforum von Ralph.* https://ralph.discourse.group/t/integration-of-ralph-inventory-extension/154, Abruf: 2020-03-05