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, 5. Februar 2020

Ignore this content for now

Kurzfassung

Abstract

Ignore this content for now

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 5. Februar 2020

Josip Domazet

Mathias Möller

Hannes Weiss

Inhaltsverzeichnis

Tabellenverzeichnis

kann entfallen falls (fast) leer

Abbildungsverzeichnis

1 Ziele

Ignore this content for now

Das erste Kapitel stellt die Ziele der DA (inkl. individuelle Ziele aller Mitarbeiter) dar.

Mögliche Gliederung (nach [30])

- Einleitung
- Zielsetzung und Aufgabenstellung des Gesamtprojekts
- individuelle Zielsetzung und Aufgabenstellung mit Terminplan der einzelnen Teammitglieder
- Grundlagen und Methoden (Ist-Situation, Lösungsansätze, konkrete Vorgehensweise)
- Bearbeitung der Aufgabenstellung (technische Beschreibungen, Berechnungen)
- Ergebnisse (Ergebnisdarstellung, kritische Gegenüberstellung mit der Zielsetzung und der gewählten Vorgehensweise)

2 Einführung in die App-Architektur

Wie im vorherigen Kapitel angesprochen, ist das Ziel der Diplomarbeit, eine App zu entwickeln, mit der man in der Lage ist, eine Inventur durchzuführen. Doch wieso eine App und wieso überhaupt Android? Um diese Frage zu klären, muss man zwischen zwei Begriffen unterscheiden [**?**]:

- Native App
- Web-App

2.1 Nativ vs. Web

Unter einer nativen App versteht man eine App, die für ein bestimmtes Betriebssystem geschrieben wurde [**?**]. Die Definition ist allerdings nicht ganz eindeutig, da Frameworks wie Flutter und Xamarin nativer Funktionalität sehr nahekommen, obwohl sie mehrere verschiedene Betriebssysteme unterstützten. Eine Web-App hingegen basiert auf HTML und wird per Browser aufgerufen. Sie stellt nichts anderes als eine für mobile Geräte optimierte Website da.

Am Markt zeichnet sich in letzter Zeit ein klarer Trend ab – native Apps sterben allmählich aus [**?**] und werden durch mobile Webseiten ersetzt. Das ist dadurch erklärbar, dass mobile Webseiten immer funktionsreicher werden. Mittlerweile haben Web-Apps nur noch geringfügig weniger Möglichkeiten als native Apps. Aus wirtschaftlicher Betrachtungsweise amortisieren sich Web-Apps de facto um einiges schneller und sind auch dementsprechend lukrativer.

2.2 Begründung: Native App

Das Projektteam hat sich dennoch 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 unsere Schule darstellt. Eine native App wird eine Web-App immer hinsichtlich Qualität und User Experience 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 da, 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. Ein typisches Beispiel für eine Web-App stellt eine mobile Website einer Tageszeitung (eventuell auch mit Kommentaren, Bewertungssystemen etc.) da. 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

Nun gilt es zu klären, warum die App für natives Android (in Java) entwickelt wurde. Folgende nativen Alternativen galt es abzuwägen:

- Flutter [**?**]
- Xamarin [**?**]
- 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 – das erste stabile Release wurde erst im Dezember 2018 veröffentlicht [**?**]. 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 ähnelt Flutter in den soeben aufgezählten Aspekten stark. Es ist ebenfalls ein cross-platform Framework, das jedoch in C# geschrieben wird.

Native IOS wird nur der Vollständigkeit Halber aufgelistet, stellte allerdings zu keinem Zeitpunkt eine wirkliche Alternative da, weil IOS-Geräte einige Eigenschaften besitzen,

die für eine Inventur nicht optimal sind (Sprichwort: Akkukapazität). Außerdem haben in etwa nur 20% aller Geräte [**?**] 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. All dies hat zum Entschluss geführt, IOS aus dem Spiel zu lassen und uns auf Android zu fokussieren.

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 da:

- 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 größte 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 [**?**] 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.

Schlussendlich muss noch die Frage der unterstützten Android-Versionen geklärt werden. Das minimale API-Level der App ist 21 - auch bekannt als Android 5.0 'Lollipop'. Android 4.0 hat sehr viele nützliche Libraries hervorgebracht. So zum Beispiel die *Mobile Vision API* von Google, dank derer man in der Lage ist, Barcodes in akzeptabler Zeit mit der Kamera des Geräts zu scannen. Die Wahl ist auf 5.0 gefallen, da somit ein Puffer zur Verfügung steht und in etwa 90% aller Android-Geräte ohnehin auf 5.0 oder einer neueren Version laufen [**?**].

2.3 Einführung zu nativem Java

Um eine Basis für die folgenden Kapitel zu schaffen, werden hier die Basics der Android-Entwicklung mit nativem Java näher beschrieben. Das Layout einer App wird in XML Files gespeichert, währenddessen das wirkliche Programmieren mit Java erfolgt.

2.3.1 Single-Activity-App

Als Einstiegspunkt in eine App dient eine sogenannte *Activity*. Eine Activity ist eine normale Java-Klasse, der durch Vererbung UI-Funktionen verleiht 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 [**?**]. 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.

2.3.2 Seperation of Concerns

In Android ist es eine äußerst schlechte Idee, sämtliche Logik in einer Activity oder einem Fragment zu implementieren. Das softwaretechnische Prinzip *Seperation of Concerns (SoP)* 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 gilt es diese auf mehrere logische Komponenten aufzuteilen. Beispiel: Eine Activity hat immer die Verantwortung, die Kommunikation zwischen UI und Benutzer abzuwickeln. Bad Practice wäre es, wenn jene Activity ebenfalls dafür verantwortlich ist, Daten von einem Server abzurufen. Das Prinzip verfolgt das Ziel, die *God Activity Architecture (GAA)* möglichst zu vermeiden [**?**]. Eine God-Activity ist unter Android eine Activity, die die komplette Business-Logic beinhaltet und SoP in jeglicher Hinsicht widerspricht. God-Activities gilt es dringlichst zu vermeiden, da sie folgende Nachteile mit sich bringen:

- Refactoring wird kompliziert
- Wartung und Dokumentierung wird 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 dem *Lifecycle* 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)

God-Activities sind ein typisches Beispiel für Spaghetticode. Es bedarf also einer wohlüberlegten und strukturierten Architektur, um diese Probleme zu unterbinden. Im nächsten Kapitel wird dementsprechend die Architektur der App im Detail erklärt.

3 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, in die das Projektteam spezialisiert ist, stehen als Backend-Lösung vier *Frameworks* zur öffentlichen Verfügung, zwischen denen gewählt wurde:

- Django [19]
- Pyramid [26]
- Web2Py [27]
- Flask [24]

Alle genannten Alternativen sind *Frameworks* der Programmiersprache Python. Gewählt wurde "Django" aufgrund einer bestehenden und frei verfügbaren Inventarverwaltungsplattform "Ralph" [28], die auf Django aufbaut und durch die vorliegende Diplomarbeit hinreichend erweitert wird.

3.1 Django und Ralph

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 [4] von Ralph an.

Die vorliegende Diplomarbeit bietet eine Erweiterung des Ralph-Systems.

3.1.1 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. [1] 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*[29]), siehe Views

Zusätzlich bietet Ralph bereits einige *Features*, die die grundlegende Führung und Verwaltung eines herkömmlichen Inventars unterstützen (beispielsweise eine Suchfunktion mit automatischer Textvervollständigung).

3.2 Kurzfassung der Funktionsweise von Django und Rlaph

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

3.2.1 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` , unter Ralph in der jeweiligen Datei im Verzeichnis `settings`. Eine detaillierte Anleitung zur Verbindung mit einer Datenbank ist in der offiziellen Django-Dokumentation [7] 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 Packets ist normiert.[6] 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`[8][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` Attributs ist unter allen Instanzen eines Modells einzigartig. Die Anpassung dieses Attributs wird in der offiziellen Django-Dokumentation genauer behandelt. [8]

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`. [11]

3.2.2 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

[1] erbend von der Superklasse `Model`[8]
[2] oder davon erbende Klassen

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, die Detailansicht einer Modellinstanz unter der URL

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

. Somit repräsentiert die URL der Listenansicht gleichzeitig den Pfad, unter der sie definiert wurde.

Die Dokumentation der Administrationsfeatures von Django ist auf der offiziellen Dokumentationswebseite von Django [5] zu finden.

3.2.3 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* [17], die unter Django durch das integrierte *DRF* implementiert wird.[25] 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

[3] Unter Ralph steht hierfür die Klasse `RalphAdmin` zur Verfügung.[20]
[4] Unter Ralph steht hierfür die Klasse `RalphAdminForm` zur Verfügung.
[5] Unter Ralph steht hierfür die Klasse `RalphAPISerializer` zur Verfügung. [21]
[6] Unter Ralph steht hierfür die Klasse `RalphAPIViewSet` zur Verfügung. [21]

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* [25]. 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 [25] und der offiziellen Ralph-Dokumentationsseite [21] zu finden.

3.2.4 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] [9] oder vereinzelte Methoden mit einem `request`[9] Parameter. [16] 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* [14]. 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 gängige Attacken, wie *SQL-Injections* oder *CSRF* [29] und gilt daher als besonders sicher.

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`. [15]

3.2.5 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.

[7] Unter Ralph steht hierfür die globale `RalphRouter` Instanz `router` zur Verfügung. [21]
[8] die ebenfalls Superklasse der Klasse `ApiView` ist
[9] zu Deutch: Anfrage; entspricht dem empfangenen Packet, meist als HTML.

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. [12]

3.3 Designgrundlagen

Designgrundlagen für Django-Entwickler sind auf der offiziellen Dokumentationsseite von Django abrufbar. [18] Die Erweiterung von Django durch die vorliegende Diplomarbeit wurde anhand dieser Grundlagen entwickelt.

Das Konzept des `Mixins` wird von der Ralph-Plattform besonders häufig genutzt. `Mixins` sind Klassen, die anderen von ihnen erbende Klassen, bestimmte Attribute und Methoden hinzufügen. Manche `Mixins` 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.

[10] entnommen aus der offiziellen Django Dokumentation [12]

4 Die 2 Erweiterungsmodule

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 Funktionsweise der beiden Module. Die Bedienung der *Weboberfläche* ist dem Handbuch zum Server zu entnehmen. Add reference

4.1 Das “Capentory” Modul

Das “Capentory” Modul beherbergt 3 wichtige Modelle:

1. HTLItem
2. HTLRoom
3. HTLItemType

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

4.1.1 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.

4.1.1.1 Die wichtigsten Attribute

Zu den wichtigsten Attributen des `HTLItem` Modells zählen u.a.:

- `anlage` und `asset_subnummer`: Diese Attribute bilden den Barcode eines Gegenstandes.
- `barcode_prio`: Wenn dieses Attribut gesetzt ist, überschreibt es den durch die Attribute `anlage` und `asset_subnummer` 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.

4.1.1.2 Einzigartigkeit von HTLItem Objekten

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

Sind die Attribute `anlage`, `asset_subnummer` 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` [11] anzupassen:

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

Dieses Verfahren erfüllt nicht 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()`[1] Methode [10] überschrieben werden.

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_subnummer` generierten Barcode übereinstimmen. Um diese Bedingung zu erfüllen kann nur die `validate_unique()` Methode herbeigezogen werden.

4.1.2 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.

4.1.2.1 Die wichtigsten Attribute

Zu den wichtigsten Attributen des `HTLRoom` Modells zählen u.a.:

- `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. Anwendungsbeisipel ist ein Schrank, der sowohl als `HTLRoom` Objekt als auch als `HTLItem` Objekt definiert ist. Sind die beiden

[1] Eine Methode einer Modell-Klasse, die unter Normalzuständen immer vor dem Speichern eines Objekts des Modells aufgerufen wird. Wirft sie einen Fehler auf, kann das Objekt nicht gespeichert werden.

Objekte durch das `item` Attribut verbunden, ist der Barcode des `HTLRoom` Objekts automatisch jener des `HTLItem` Objekts.

4.1.2.2 Einzigartigkeit von HTLRoom Objekten

Bezüglich der Einzigertigkeit 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()`[2] Methode [10] implementiert werden.

4.1.2.3 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 Speicherforgängen, die nicht direkt durch die Administrationsoberfläche initiiert werden [3], muss `clean_children()` manuell aufgerufen werden.

4.1.3 Das HTLItemType Modell

Das `HTLItemType` Modell repräsentiert Kategorien von Gegenständen. Das Modell besteht aus 2 Eigenschaften. Die Eigenschaft `item_type` beschreibt ein `HTLItemType`

[2] Eine Methode einer Modell-Klasse, die unter Normalzuständen immer vor dem Speichern eines Objekts des Modells aufgerufen wird. Wirft sie einen Fehler auf, kann das Objekt nicht gespeichert werden.
[3] etwa das automatisierte Speichern beim Datenimport

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 Abschnitt “Designgrundlagen”) `WithCustomFieldsMixin` bietet die Funktion der `custom_fields_inheritance`. Die Funktion ermöglicht das Erben von allen *Custom-Field* Werten eines bestimmten Objekts an ein anderes. Dieser 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` Methode genutzt, die 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 Methoden werden je beim Aufkommen eines `post_save` Signals [13] ausgeführt, dessen `sender` ein `HTLItemType` oder `HTLItem` ist. Die Methoden rufen jeweils eine weitere Methode auf, welche die *Custom-Fields* entsprechent aggregiert.

5 Einführung in die Infrastruktur

5.1 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 Organisieren eines Servers
- das Aufsetzen eines Betriebssystemes
- die Konfiguration der notwendigen Applikationen
- die Konfiguration der Netzwerkschnittstellen
- das Testen der Konnektivität im Netzwerk
- die Einrichtung des Produktivbetriebes der Applikation
- das Verfassen einer Serverdokumentation
- die Absicherung der Maschine
- die Überwachung des Netzwerks

In den folgenden Kapitel wird deutlich gemacht, wie die oben angeführten Punkte, im Rahmen der Diplomarbeit “Capentory” abgearbeitet wurden.

5.1.1 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.

5.1.1.1 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 gebaut und gesicherten Virtual Private Network (VPN), sodass nur mittels eines eingerichteten Tools auf den virtualisierten Server zugegriffen werden kann.

5.1.1.2 FortiClient

FortiClient ermöglicht es, eines 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 Diplomarbeit "Capentory" jedoch Erreichbarkeit im Schulnetz verlangt, muss die Maschine in einem Ausmaß abgesichert werden, damit sie ohne Bedenken in das Schulnetz gehängt werden kann. Dafür müssen folgende Punkte gewährleistet sein:

- Konfiguration beider Firewalls (siehe Punkt **Absicherung der virtuellen Maschine**)
- Wohlüberlegte Passwörter und Zugriffsrechte

5.1.2 Wahl des Betriebssystems

Neben den physischen Hardwarekomponenten wird für einen funktionierenden und leicht bedienbaren Server logischerweise auch ein Betriebsystem benötigt. Die erste Entscheidung, welche Art von Betriebssystem für die Diplomarbeit "Capentory" in Frage kam wurde rasch beantwortet: Linux. Weltweit basieren die meisten Server und andere Geräte auf Linux. Jedoch gibt es selbst innerhalb des OpenSource-Hersteller zwei gängige Distributionen, die das Diplomarbeitsteam während deren Schulzeit an der Htl Rennweg kennenlernen und Übungen darauf durchführen durfte:

- Linux CentOS
- Linux Ubuntu

5.1.2.1 Linux CentOS

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

5.1.2.2 Linux Ubuntu

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

5.1.2.3 Vergleich und Wahl

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 “Capentory” klarerweise das Betriebsystem Ubuntu zu verwenden, vorallem auch weil der Hersteller deren Serversoftware die Verwendung von diesem Betriebsystem empfiehlt. Anschließend wird die Installation des Betriebsystemes genauer erläutert und erklärt.

5.1.3 Installation des Betriebsystemes

Wie bereits unter Punkt ""“Anschaffung des Servers”"" erwähnt, wird uns von der Schule ein eigener Servercluster mit virtuellen Maschinen zur Verfügung gestellt. Durch die

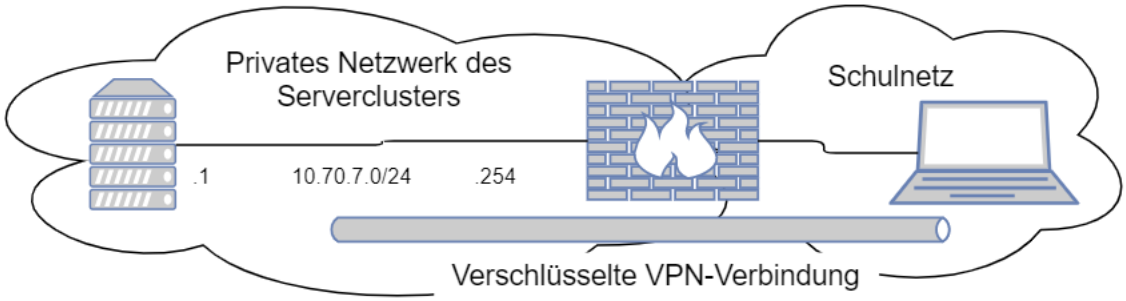
ProxMox-Umgebung und diversen Tools, ging die Installation der Ubuntu-Distribution rasch von der Hand. In der Virtualisierungsumgebung der Schule musste nur ein vorhandenes Linux-Ubuntu 18.04 ISO-File gemountet und anschließend eine gewöhnliche Betriebsysteminstallation für Ubuntu durchgeführt werden. Jedoch kam es beim ersten Versuch zu Problemen mit der """Konfiguration der Netzwerkschnittstellen""", die im nächsten Punkt genauer erläutert werden.

5.1.4 Konfiguration der Netzwerkschnittstellen

Um eine funktiorierende Internetverbindung zu erstellen, durfte die Konfiguration der Schnittstellen nicht erst "später durchgeführt" werden, da mit dem Aufschub der Schnittstellen-Konfiguration das Paket "NetworkManager" nicht installiert wurde. Der "NetworkManager" ist verantwortlich für den Zugang zum Internet und der Netzwerksteuerung auf dem Linuxsystem. Und da im Nachhinein dieses Paket nicht installiert war (und aufgrund fehlender Internetverbindung nicht installiert werden konnte), half auch die fehlerfreie Interface-Konfiguration nicht, um eine Konnektivität herzustellen. Dadurch musste ein zweiter Installationsdurchgang durchgeführt werden, worauf dann alles fehlerfrei und problemlos lief.

5.1.4.1 Topologie des Netzwerkes

Unter Abbildung 5.1 wird der Netzwerkplan veranschaulicht.



5.1.5 Konfiguration der notwendigen Applikationen

5.1.6 Testen der Konnektivität im Netzwerk

5.1.7 Produktivbetrieb der Applikation

5.1.8 Verfassen einer Serverdokumentation

5.1.9 Absicherung der virtuellen Maschine

5.1.10 Überwachung des Netzwerks

6 Planung

A Anhang 1

was auch immer: technische Dokumentationen etc.

Zusätzlich sollte es geben:

- Abkürzungsverzeichnis
- Quellenverzeichnis (hier: Bibtex im Stil plaindin)

Index

Literaturverzeichnis

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

[2] *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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

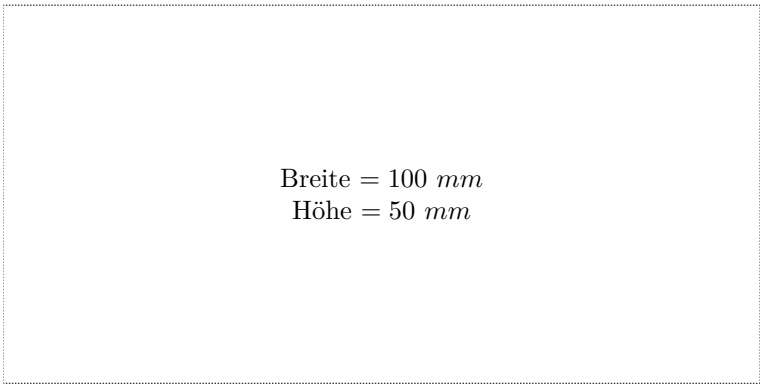
[27] *Die offizielle Web2Py-Website*. http://www.web2py.com/, Abruf: 2020-01-01

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

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

[30] *Reife- und Diplomprüfung, Abschlussprüfung an technischen, gewerblichen und kunstgewerblichen Lehranstalten*. https://moodle.htl.rennweg.at/MoodleKurs-Matura/HTL_RDP-AP_Leitfaden.pdf, Abruf: 2018-04-24

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —

Diese Seite nach dem Druck entfernen!