TUI - Test- und Integrationssystem für SCADA-Software

von Stefan Volz & Yannis Köhler

4. Januar 2019

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abk.	Beschreibung
AMQ	Atomic Message Queue
API	Application Programming Interface
CLI	Common Language Infrastructure
DI	Digital Input
DLR	Dynamic Language Runtime
DO	Digital Output
ERD	Entity-Relationship-Diagramm
ERM	Entity-Relationship-Modell
HMAC	Keyed-Hash Message Authentication Code
IPC	Industrie-PC
JIT	Just-in-time (bei Bedarf)
JRE	Java Runtime Environment
JVM	Java Virtual Machine
KW	Kalenderwoche
CV	Computer Vision
OAEP	Optimal Asymmetric Encryption Padding
ORM	Object Relational Mapper
PBKDF2	Password-Based Key Derivation Function 2
PEP	Python Enhancement Proposal
PKCS #1	Public-Key Cryptography Standards First Family
PSF	Python Software Foundation
SCADA	Supervisory Control and Data Acquisition
SHA	Secure Hash Algorithm
SV	Stefan Volz
UI	User Interface
VM	Virtual Machine / Virtuelle Maschine
YK	Yannis Köhler

Teil I
Einführung

Sämtliche Diagramme, Bilder, etc. sind, sofern nicht anders angegeben, selbst erstellt. Programmlistings wurden teils notwendigerweise implizit mittels „\“ umgebrochen. Dies ist zwar valide Python 3 Syntax, jedoch nicht in jedem Fall konform zu gängigen Format-Konventionen.

1 Die Firma Padcon

Die Firma Padcon GmbH, ein Tochterunternehmen des deutschen Netzbetreibers RWE, ist ein weltweit agierendes IT-Unternehmen mit Sitz in Kitzingen. Zu ihren Produkten gehören Systeme zur Überwachung von Groß-Photovoltaikanlagen. Diese fallen in die Kategorie der SCADA-Systeme. Besonders nennenswert ist der „Pavagada Solar Park“ in Indien, welcher, nach Bauabschluss, mit knapp 2GW die aktuell größte Photovoltaikanlage der Welt darstellt.

2 Aufgabenstellung

Für die Firma Padcon GmbH soll ein Test- und Integrationssystem erstellt werden. Dies umfasst die Dimensionierung und Installation von vier sogenannten Testplätzen in einem dafür vorgesehenen Raum. Hierfür muss auch eine neue Unterverteilung gebaut werden. An den Testplätzen wird man verschiedene Hardware wie IPC‘s, Router, Switches, etc. testen können. Ebenfalls sollen neue Versionen der firmeneigenen SCADA-Software, welche die Photovoltaik-Anlagen überwacht und Daten ausliest/aufzeichnet, getestet werden können, bevor diese an den Kunden ausgeliefert werden. Desweiteren soll eine Inventarsoftware erstellt werden, welche ein Einscannen von Barcodes ermöglicht und somit eine schnelle Zuordnung von Geräten zu Testplatz und zugehörigem Verantwortlichen zulässt. [sv]

Teil II

Test- und Integrationsraum

3 Abschnitt 1 Raum

Teil III

TUInventory

4 Begriffsdefinitionen

Begriff	Definition
Duck-Typing	Der Typ einer Instanz wird dadurch beschrieben/festgelegt welche Member sie besitzt (Von eng.: “If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck.”)
Dynamisch Typisiert	Typüberprüfung zur Laufzeit
Frame	Einzelnes Bild einer Videosequenz, hier synonym für Bildmatrix eingesetzt
Function Decorator	Ist Funktionsdefinition mit @decorator vorgestellt - Stellt im Prinzip eine Definition der Funktion f dar, der ein f = decorator (f) nachgestellt ist. Hierbei wird eine Funktion i.d.R. um weitere Funktionalität (z.B. ein Cache oder die Möglichkeit Vektoren zu verarbeiten) erweitert
Immutabel / immutable	Unveränderbar
Kontext-Manager / Context Manager	Klasse, die __enter__ und __exit__ implementiert. Erlaubt es Objekte auch im Fehlerfall sauber zu deinitialisieren, bzw. zu schließen (entspricht einem try-finally-Block).
Lazy evaluation	Die Auswertung eines Ausdrucks erfolgt nur soweit sie gerade nötig ist.
Magic Method	Eine i.d.R. implizit aufgerufene Methode, welche einer Klasse besondere Fähigkeiten verleiht[1].
Mutabel / mutable	Veränderbar
Mutex	Gegenseitig ausschließend (von engl. mutual exclusion)
ORM	Programmiertechnik um Daten zwischen inkompatiblen Typsystemen (hier Datenbank in SQLite3 und Python) zu konvertieren und somit eine virtuelle Objektdatenbank zu schaffen
Pythonic	Idiomatisch im Bezug auf Python
Race-Condition	Im Falle einer Race-Condition ist das Ergebnis einer Operation nicht deterministisch, es wird beeinflusst durch äußere Gegebenheiten wie z.B. Prozessorlast Der Name stammt von der Vorstellung, dass Signale wettlaufen um die Ausgabe als erstes zu beeinflussen[2]. Probleme einer Race-Condition sind beispielsweise, dass sie oftmals verschwindet wenn das Programm mittels Debugger betrachtet wird.

[1] [**?**, S. 369]
[2] [**?**, 24.12.18 - 01:25 Uhr]

Salt	Zufällige Zeichenfolge die in der Kryptografie u.A. bei Hash-Funktionen eingesetzt wird um die Entropie der Eingabe zu erhöhen.
Singleton	Ein Singleton ist ein Entwurfsmuster bei dem sichergestellt ist, dass nur eine einzige Instanz einer Klasse existiert (z.B. in Python *None*)
Thread	(dt.: Faden, Strang) Threads sind im Grunde genommen leichtgewichtige Prozesse (Ein Prozess kann mehrere Threads besitzen). In CPython (erläutert in Abschnitt **??**) ist der GIL (Global Interpreter Lock) zu beachten: Threads steigern hier nicht die Performance!
Thread-Safe	(dt.: Threadsicherheit) Mehrere Threads können gleichzeitig auf eine Komponente zugreifen ohne sich gegenseitig zu behindern oder race-conditions auszulösen

5 Python 3

Bei Python 3 handelt es sich um eine universell einsetzbare, plattformübergreifende, dynamisch typisierte Hochsprache, welche in verschiedensten Implementierungen als Open Source Projekt entwickelt wird. Ursprünglich wurde sie von Guido van Rossum 1991 mit dem Ziel entwickelt, eine möglichst lesbare und dennoch mächtige Programmiersprache zu schaffen. Auf der Website der PSF steht dazu: “Python is a programming language that lets you work quickly and integrate systems more effectively.”[3]

In der Standardimplementierung CPython wird der Quellcode zuerst in einen Bytecode kompiliert, welcher dann von einer in C geschriebenen VM interpretiert wird. Andere bekannte Implementierungen sind z.B. PyPy (Interpreter in RPython, Vorteil: sehr viel schneller als CPython, JIT-Compiler), Jython (Ausführung mittels JVM, Vorteil: Interoperabilität mit JRE) oder IronPython (Ausführung mittels DLR - also .NET, Vorteil: Interoperabilität mit .NET bzw. der CLI und Integration in Visual Studio).[4]

Alle Ausführungen dieser Arbeit beziehen sich, sofern nicht anders angegeben, auf CPython in Version 3.7.

5.1 Funktionen und Methoden in Python

Eine Besonderheit Pythons ist, dass alles, sei es Klasse, Instanz oder Funktion, ein Objekt ist. Dies bringt eine extreme Flexibilität mit sich.

Die Definitionen der Begriffe *Funktion* und *Methode* sind in Python anders als man es eventuell aus anderen Programmiersprachen kennt, daher seien sie hier kurz erläutert[5]:

	kein decorator	@classmethod	@staticmethod
Klasse	function	bound method	function
Instanz	bound method	bound method	function

Neben Methoden und Funktionen ist auch auch möglich andere Objekte in Python aufrufbar zu machen, hierzu müssen sie die *magic method* *__call__* implementieren.

[3] [**?**, 18.12.18 - 16:32 Uhr]

[4] [**?**, 18.12.18 - 17:08 Uhr]

[5] Neben den *bound methods* gab es außerdem noch die sogenannten *unbound methods*, die einer Methode entsprechen, welche ursprünglich zu einer Klasse gehört hat und somit eine Instanz dieser als ersten Parameter erwartet [**?**, 18.12.18 - 17:24 Uhr], allerdings von der Klasse losgelöst wurde. Jedoch wurde dieses Konzept mit Python 3.0 verworfen; was früher eine *unbound method* war, ist nun ebenfalls eine *function* [**?**, 18.12.18 - 17:23 Uhr].

Algorithmus 1 Die verschiedenen Funktionstypen

```
class MyClass():
    def my_method(self):
        pass

    @classmethod
    def my_classmethod(this):
        pass

    @staticmethod
    def my_static_method():
        pass

is_a_function_now = MyClass.my_method
```

5.2 Formatierung

Ein Grundsatz der Quellcodeformatierung ist, dass der Code wesentlich öfter gelesen als geschrieben wird. Dementsprechend sollte hier mit großer Vorsicht und einer gut durchdachten Systematik vorgegangen werden. In Python gibt es als grundlegende Quelle, wie man seinen Code gut lesbar und idiomatisch schreibt und formatiert z.B. das “Zen of Python”, welches in Python über *import this* eingesehen werden kann (und als PEP 20 zu finden ist), oder aber die Python Enhancement Proposals PEP 8 und PEP 254.

PEP 8 trägt hier den Titel “Style Guide for Python Code” und umfasst alle Formatfragen von generellem Codelayout über Kommentare bis hin zu Namensgebungskonventionen[6]. PEP 257 hingegen beschäftigt sich mit den sogenannten Docstrings und deren Format [7]. Docstrings sind eine Form von Inline-Dokumentation, die z.B. die Schnittstelle und Funktion einer Methode beschreibt. Sie werden vom Python-Interpreter berücksichtigt und sind dann unter dem Magic Member *__doc__* zu finden. Über Docstrings könnte man aus dem Quellcode auch automatisch die Dokumentation generieren lassen, dies ist in unserem Fall jedoch nicht gewünscht, da die Dokumentation im Rahmen dieser Arbeit stattfindet (im Anhang sind sie dennoch zu finden ({modulname}.html)). Als weitere Referenz für das genutzte Format galten ein von Google veröffentlichtes Dokument[8], sowie der Talk “Beyond PEP 8 -- Best practices for beautiful intelligible code” von Raymond Hettinger, einem der Core Developer von CPython[9].

Abschließend ist es jedoch ratsam, bei all diesen Formatforschriften nochmal Bezug auf

[6] [**?**, 18.12.18 - 16:17 Uhr]
[7] [**?**, 18.12.18 - 16:17 Uhr]
[8] [**?**, 18.12.18 - 17:36 Uhr]
[9] [**?**, 18.12.18 - 17:40 Uhr]

den Anfang von PEP 8 zu nehmen: “A Foolish Consistency is the Hobgoblin of Little Minds”[10] - man sollte also wissen, wann es die bessere Entscheidung ist mit den Vorschriften zu brechen.

5.3 Vorwort und Übersicht über die Anwendung

„All computers are now parallel... Parallel programming *is* programming.“
- Michael McCool[11]

Wie die meisten modernen Anwendungen, setzt auch TUInventory an vielen Stellen auf Multithreading. Daher sei hier eine Grafik vorangestellt, die Kontroll- (solider Pfeil) und Informationsfluss (gestrichelter Pfeil) visualisiert:

Abbildung 5.1: Übersicht über Verhältnisse von Threads zueinander

Desweiteren sei eine Übersicht über alle Module und deren Klassen, sowie Top-Level-Functions und Module-Level-Instanzen gegeben (siehe hierzu auch die automatisch generierte Dokumentation im Anhang):

barcodereader.py

- **VideoStream** class, thread
- **LazyVideoStream** class, thread
- **Camera** class

classes.py

- **BigInt** class
- **setup_context_session** method
  - **ContextSession** class
- **Producer** class
- **Article** class
- **Device** class
- **PhoneNumber** class
  - **NoNumberFoundWarning** class

[10] Siehe Fußnote **??** (PEP8)
[11] [**?**, , S. 1]

Location class

User class

Responsibility class

Timeout class, thread

keys.py

generate_key method

read_keys method

logger.py

logger RootLogger instance

main.py

VideoStreamUISync class, thread

main method

qr_generator.py

generate_qr method

slots.py

save_to_db method

update_user_dependant method

create_user method

create_admin method

login method

logout method

create_device method

create_location method

create_producer method

generate_password method

reset_password method

reset_admin_password method

ui.py

MainDialog class

LoginDialog class

utils.py

absolute_path method

_ParallelPrint class, thread

parallel_print put method of _ParallelPrint instance

6 SQLAlchemy

Bei SQLAlchemy handelt es sich um ein Open-Source SQL Toolkit und einen objektrelationalen Mapper für Python.

Lesenswert im Bezug auf die Grundidee, bzw. Architekturentscheidungen von SQLAlchemy ist hierzu, das von Michael Bayer, dem/einem Entwickler von SQLAlchemy, in *The Architecture of Open Source Applications (Volume 2)*[12] veröffentlichte Kapitel über SQLAlchemy. Hieraus geht hervor, dass SQLAlchemy so entworfen wurde, dass man bei der Entwicklung damit gewillt sein muss, die relationalen Strukturen seiner Daten zu bedenken, die Implementierung dieser jedoch durch eine High-Level-Schnittstelle erfolgen sollte. Dieser Abstraktionslayer ermöglicht es SQLAlchemy mit den verschiedensten Datenbanken (SQLite, PostgreSQL, Oracle,...) zu arbeiten, ohne große (wenn überhaupt) Änderungen am Code vorzunehmen. Diese verschiedenen Implementierungen werden alle über dieselbe High-Level API angesprochen und im SQLAlchemy-Jargon “dialect” genannt. Ein weiteres grundsätzliches Merkmal ist, dass SQLAlchemy grob in ein Core-Modul und ein darauf aufsetzendes ORM-Modul aufgeteilt werden kann. Viele Firmen bauen auf dem Core basierend einen eigenen ORM auf. Die Details dieser Struktur sind der umfangreichen SQLAlchemy-Dokumentation zu entnehmen.

6.1 Das Datenbankmodell

Das mit SQLAlchemy umgesetzte Datenbankmodell wurde zunächst wie folgt geplant:

[12] [**?**, 18.12.18 - 17:26 Uhr]

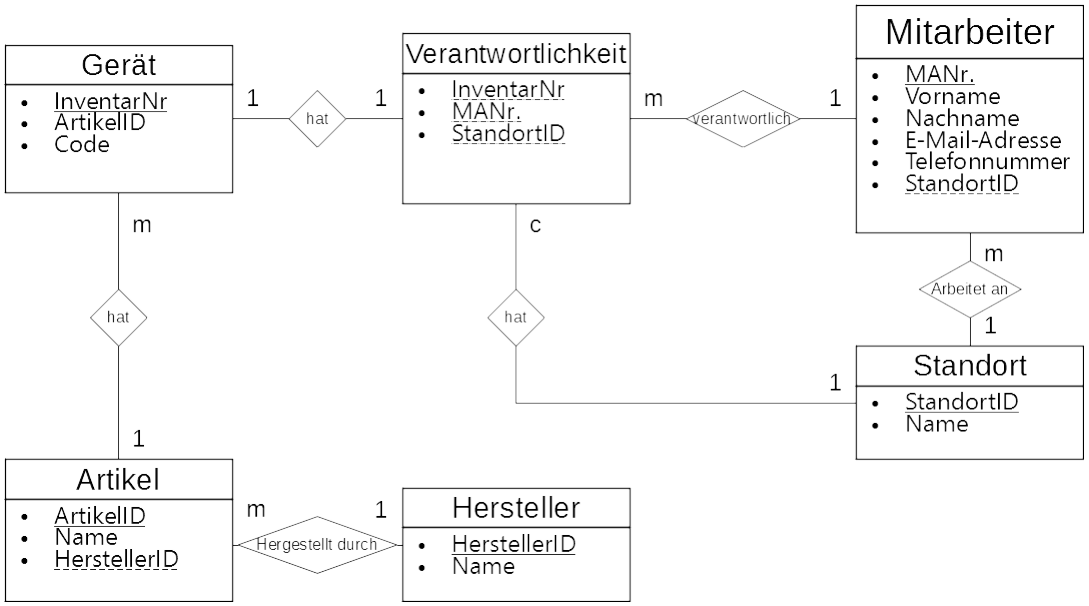


Abbildung 6.1: Entity-Relationship-Diagramm

Es sollte also Geräte geben, deren Standort verfolgt wird und die immer einen Mitarbeiter zugewiesen haben, der für sie verantwortlich ist. Um eine einfachere Pflege der Datenbank zu ermöglichen, wurden der Artikel eines Geräts sowie dessen Hersteller ausgekapselt (3. Normalform).

6.2 Implementierung des Datenbankmodells

Auf Basis dieses ERM wurden Klassen wie folgt erstellt:

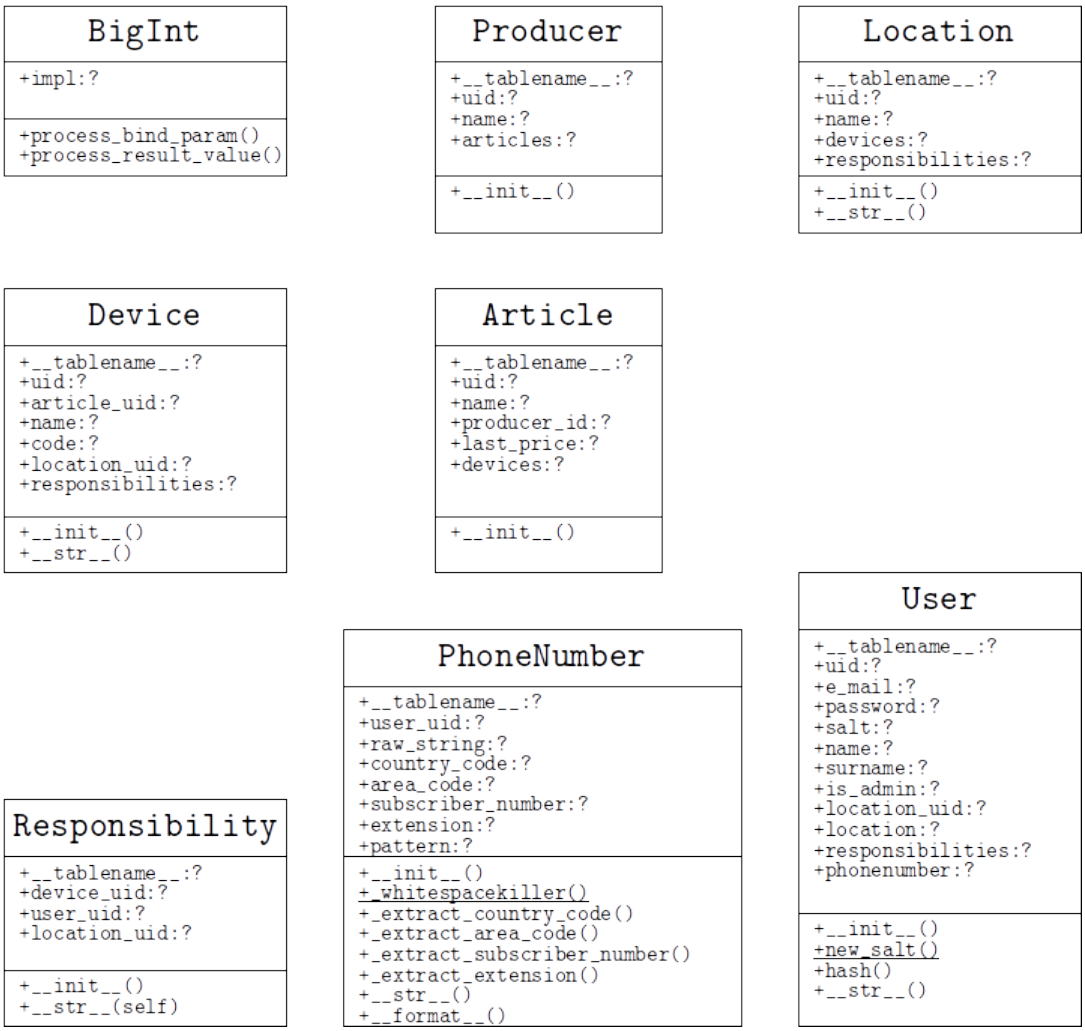


Abbildung 6.2: UML-Diagramm

Diese Klassen erben alle von einer abstrakten Basisklasse *Base*, die von SQLAlchemy bereitgestellt wird, und besitzen jeweils einen eigenen Datenbank-Table. Auch die Beziehungen der Klassen werden auf Basis des Quellcodes automatisch von SQLAlchemy erzeugt.

6.3 Benutzerdefinierte Datentypen

Da SQLAlchemy im Hintergrund eine tatsächliche SQL-Datenbank einsetzt, ist es natürlich an deren Datentypen gebunden. Daher ist es hier, ähnlich einer traditionellen DBAPI, nötig, Adapter zu schreiben, um eigene Datentypen ablegen zu können. Konkret geht es darum, das *salt* eines Nutzers in der Datenbank zu speichern. Das Salt ist eine sehr große Zahl. Daher

wird ein eigener Datentyp benötigt, welcher den Python-seitigen Wert beim Speichern in die Datenbank zu einem String konvertiert und beim Abrufen die umgekehrte Umwandlung durchführt. SQLAlchemy stellt hierzu die abstrakte Klasse *TypeDecorator* zur Verfügung, von dieser leiten wir eine Klasse *BigInt* ab. Als Klassenvariable legen wir über *impl* fest, dass die datenbankseitige Implementierung als string erfolgt, die beiden Methoden *process_bind_param* und *process_result_value* nehmen die Umwandlung vor. Es ist auch möglich, je nach *dialect* der Datenbank, verschieden umzuwandeln, dies ist hier allerdings nicht nötig.

6.4 Beziehungen herstellen in SQLAlchemy

6.4.1 Grundprinzip und 1-zu-n-Beziehungen

In SQLAlchemy werden Beziehungen zwischen Tables mittels der *relationship* Methode hergestellt. Die Variante, für die wir uns entschieden haben, nutzt hierbei *backref*:

Algorithmus 2 Beispiel einer 1-zu-n Beziehung

```
class Parent(Base):
    __tablename__ = "parents"
    id = Column(Integer, primaryKey=True)
    children = relationship("Children", backref="parent")


class Child(Base):
    __tablename__ = "children"
    parent_id = Column(Integer, ForeignKey("parents.uid", primaryKey=\
        True))
```

Es wird also in einer Klasse lediglich ein Fremdschlüssel festgelegt. In der anderen Klasse wird das *relationship* gesetzt. Dieses *relationship* stellt hierbei bei jeder Instanz von *Parent* eine Liste aller *Children*-Instanzen zur Verfügung. Umgekehrt legt *backref* bei *Children* den Member *parent* an, welcher auf die zugehörige Parent-Instanz verweist. Hier gilt zu beachten, dass es egal ist, auf welcher Seite der Beziehung der Fremdschlüssel hinterlegt ist.

Außerdem ist das erste Argument von *relationship* nicht die Klasse *Child*, sondern ein String, der ihren Namen enthält. Dies ermöglicht es, Beziehungen anzulegen, ohne dass, die Partnerklasse bereits angelegt wurde bzw. bekannt ist, was das Arbeiten wesentlich komfortabler macht.

6.4.2 1-zu-1-Beziehung

Es lassen sich mit SQLAlchemy jedoch nicht nur 1-zu-n-Beziehungen anlegen. Durch Erweiterung der *relationship* Methode mit dem *uselist* Parameter lässt sich auf folgende Weise an beiden Seiten des relationships eine Einzelinstanz hinterlegen.

Algorithmus 3 Beispiel einer 1-zu-1 Beziehung

```
class Parent(Base):
    __tablename__ = "parents"
    id = Column(Integer, primaryKey=True)
    children = relationship("Children", backref=backref("parent", \
        uselist=False))


class Child(Base):
    __tablename__ = "children"
    parent_id = Column(Integer, ForeignKey("parents.uid", primaryKey=\
        True))
```

Prinzipiell ist es auch möglich, n-zu-n Beziehungen-herzustellen, dies ist bei uns jedoch nicht nötig und wird daher nicht aufgeführt.

6.5 Arbeiten mit SQLAlchemy

Beim Arbeiten mit SQLAlchemy gibt es ein sehr prominentes Konstrukt: *Session*s

Über diese erfolgt die komplette Interaktion mit der Datenbank, sie erledigt im Hintergrund jedoch auch Dinge wie “State-Management” (Konsistenzstatus zwischen Objekt und Datenbank), was später noch eine wichtige Rolle spielt. Eine Session wird zunächst an eine *Engine* (dt. Datenbanktreiber) angebunden; diese Engine verwaltet z.B. den bereits erwähnten Dialect und stellt das Interface zur DBAPI und damit auch zur Datenbank dar. Über ihre Lebenszeit kann man einer Session Objekte manuell oder über Abfragen hinzufügen, sie entfernen und Änderungen an der Datenbank vornehmen. Oftmals, so auch in unserem Fall, gibt es jedoch viele Stellen, von denen aus auf eine Session zugegriffen werden muss. Da stellt man sich natürlich die Frage, wann man denn eine neue Session aufmacht und schließt etc.. Hierzu steht in der Dokumentation von SQLAlchemy:

“When do I construct a Session, when do I commit it, and when do I close it? [...] Make sure you have a clear notion of where transactions begin and end, and keep transactions

short, meaning, they end at the series of a sequence of operations, instead of being held open indefinitely."[13]

Dies und der Fakt, dass eine Session stets ordnungsgemäß initialisiert und deinitialisiert werden muss, führt uns daher ein weiteres Mal zu einem Kontext-Manager. Dieser Kontext-Manager verfügt über die Engine, zu der alle Sessions verbunden werden, sowie einen sog. *sessionmaker*. Dieser Sessionmaker kümmert sich um die Konfiguration der erzeugten Sessions. Da es jedoch potentiell wünschenswert ist, mehrere Datenbanken zu haben (z.B. lokale Datenbank für Geräte und remote Datenbank für Nutzerdaten), wird im Hinblick auf Zukunftssicherheit und Maintainability, eine Variante gewählt, die es erlaubt, mehrere Kontext-Manager für Sessions auf verschiedenen Engines zu haben. Damit steht eine Fabrikmethode (*setup_context_session*) zur Verfügung, der eine Engine übergeben wird und die den Kontext-Manager als Klasse zurückgibt. Die zurückgegebene Klasse kümmert sich dann um eventuelle Rollbacks im Fehlerfall etc.. Es ist wichtig zu beachten, dass Methoden, die im Hintergrund zu *INSERT*, DELETE oder UPDATE evaluiert werden, erst beim Aufrufen der Methoden *flush* oder *commit* der Session tatsächlich geschrieben werden, sofern kein *autocommit* eingestellt ist.

6.5.1 Neue Einträge in der Datenbank vornehmen

Nachdem die Vorarbeit mit Sessions und Klassen geleistet ist, ist es nun sehr einfach, neue Einträge in der Datenbank vorzunehmen. Das folgende Beispielprogramm zeigt, wie einige bereits erzeugte Objekte in der Datenbank abgelegt werden.

Algorithmus 4 Objekte in Datenbank ablegen

```
from classes import engine, setup_context_session
# Ausserdem werden benoetigte Klassen etc. eingebunden

CSession = setup_context_session(engine)
# ... Instanziieren der Klassen zu user1 und user2 sowie location1
with CSession() as session:
    session.add(user1)
    session.add_all([user2, location1])
```

Wie zu sehen ist, können sehr komfortabel entweder einzelne Objekte oder eine ganze Reihe von ihnen auf einmal geschrieben werden. Commits, Rollbacks etc. werden automatisch im Hintergrund gehandled.

[13] [**?**, 22.12.18 - 16:41 Uhr]

6.5.2 Abfragen

Abfragen können auf zwei verschiedenen Wegen erfolgen:

- direkt über SQL Querys
- über die *query*-Methode einer *Session* Instanz

Bei TUInventory wurde ausschließlich der objektorientierte Abfrageansatz eingesetzt. Als Beispiel dient hier die Abfrage (wenn auch nicht in finaler Version), die im Zusammenhang des Einloggens eines Users auftritt:

Algorithmus 5 Abfrage aus Datenbank anhand eines tatsächlichen Beispiels

```
def login(e_mail, password):
    """Log user into application
    Checks if there's a user of given name in the database,
    if the given password is correct and returns the user if both is \
        the case
    """
    e_mail = e_mail.lower()
    with CSession() as session:
        try:
            user = session.query(classes.User).filter_by(e_mail=e_mail\
                ).first()
            user_at_gate = classes.User(e_mail, password, salt=user.\
                salt)
            if compare_digest(user_at_gate.password, user.password):
                update_user_dependant(user)
                session.expunge(user)
                logger.info(f"Successfully␣logged␣in␣as␣{user.uid}")
                return user
            else:
                logger.info(f"Attempted␣login␣with␣wrong␣password␣for␣\
                    {e_mail}")
                return None
        except ValueError as e:
            logger.info(f"Attempted␣login␣from␣unknown␣user␣{e_mail}")
```

Die wichtigen Punkte sind hier die Zeilen 6 und 10. In Zeile 6 wird über die *query*-Methode eine Abfrage auf dem Table der *User* vorgenommen. Das Ergebnis dieser Abfrage wird anschließend mittels *filter_by* anhand der Spalte *e_mail* in der Datenbank mit der übergebenen E-mail Adresse gefiltert. Dies gibt uns ein *Query* Objekt. Da die Spalte *e_mail* mit *unique* gekennzeichnet ist, wird nur ein Objekt abgefragt, dieses erhalten wir mit *first*. Nun wird dieses Objekt verarbeitet bis zu Zeile 10, hier wird die Methode *expunge*, zu deutsch auslöschen der

Session genutzt, um das Objekt von der Session zu trennen, sodass es auch außerhalb von ihr seine Gültigkeit behält. Ein Grund, wieso dies nötig ist, ist die Tatsache, dass SQLAlchemy hier, wie auch in vielen anderen Bereichen sehr „lazy“ ist (siehe *Lazy Evaluation*). Tatsächlich fragt es nämlich nicht direkt das ganze Objekt ab, sondern erstmal nur das Attribut *e_mail*, da dieses gerade gebraucht wird. Im Verlauf bis Zeile 10 werden noch *salt* und *password* just-in-time abgefragt und damit der Klasse hinzugefügt. Wenn die Instanz jedoch mit *expunge* von der Session getrennt wird, erfolgt eine Abfrage aller restlichen Attribute, sodass das Objekt vollständig ist. Wer mit SQL vertraut ist, hat sicherlich bereits erkannt, dass die Methoden dieselben Namen wie einige Operationen in SQL tragen. Analog stehen auch zu weiteren SQL Operationen Methoden bereit, um komplexere Abfragen zu realisieren.

6.5.3 Aktualisieren und Löschen einer Instanz

Möchte man eine Instanz in der Datenbank ändern, ist dies ebenfalls mit einer Session einfach umzusetzen. So muss man lediglich die Instanz in einer Session verfügbar machen (entweder sie ist ohnehin schon darin oder wird über eine Abfrage ermittelt), seine Änderungen vornehmen und den Kontext der aktuellen *ContextSession* verlassen, bzw. die Methode *commit* der *Session aufrufen*. Im Fall der Löschung ist es möglich eine Instanz über die Methode *delete* der Session als gelöscht zu markieren, tatsächlich ist sie aber nicht direkt gelöscht, sondern auch hier erst, sobald der Kontext verlassen wird oder *flush* oder *commit* aufgerufen wird.

7 Frameworkfreie Implementierung

7.1 Utilities

Bei der Entwicklung wurde ein kleines Hilfsmodul geschrieben, um einige Sachen zu vereinfachen. Dieses Hilfmodul ist nicht abhängig von einem der anderen Module.

7.1.1 Absolute Pfade

Python bietet grundsätzlich die Möglichkeit an, mit relativen Pfaden zu arbeiten. Diese sind jedoch so umgesetzt, dass sie nicht relativ zur Quelldatei interpretiert werden, sondern stattdessen relativ zum Aufruf arbeiten. Führe ich das Python Programm in der Konsole also von einem anderen Ordner aus aus, so wird auf eine andere Datei zugegriffen. Daher wurde eine Funktion entwickelt, die ermittelt, wo sich die Quellcodedatei befindet und ein *pathlib* Objekt zum übergebenen relativen Pfad zurückgibt. Die Pfad-Objekte des Moduls *pathlib* sind plattformunabhängig und sehr komfortabel in der Nutzung (Beispielsweise: Erweiterung mittels Verwendung des überladenen /-Operators).

7.1.2 Paralleles Printen

Beim Debuggen ist es oftmals hilfreich sich kleinere Nachrichten an Schlüsselpunkten ausgeben zu lassen. Da diese Nachrichten oftmals jedoch aus verschiedenen Threads kommen, kann sich hier eine race-condition auf der Konsole abbilden, bei der mehrere Nachrichten gleichzeitig ausgegeben werden, wodurch sie schwer bis garnicht zu verwerten sind. Daher wird ein Singleton-Thread geöffnet, der über eine Queue Nachrichten empfangen und auf der Konsole ausgeben kann. Um eine sauberere Schnittstelle zu haben, wird die *put*-Methode seiner Queue von der Module-Level-Referenz *parallel_print* gespiegelt, über die dann aus allen Modulen sauber zugegriffen werden kann. Die Umsetzung des Singletons ist so, dass es einem Anwender von außerhalb des Moduls nicht möglich ist, eine weitere Instanz der Klasse *_ParallelPrint* zu erzeugen (Zumal das „_“ im Namen nach Konvention angibt, dass es sich um eine Klasse handelt, die in der API nicht aus Gründen der Nutzung von außen vorhanden ist[14]), indem die Referenz von *_ParallelPrint* nicht mehr auf der Klasse, sondern auf einer Instanz dieser liegt. Probiert er nun, eine Instanz zu erzeugen, wird stattdessen die Magic Method *__call__* der Instanz aufgerufen, welche eine *Warning* wirft. Sofern ein Nutzer des Moduls zwischen

[14] Ein Nutzer kann eine mit „_“-benannte Klasse/Methode/Instanz etc. zwar dennoch verwenden, sollte dabei allerdings die internen Mechanismen im Hinterkopf behalten und ist sogesagt im Fehlerfall auf sich alleine gestellt

Klassendefinition und Umleitung der Referenz im Modulkörper selbst eine weitere Instanz erzeugen würde, könnten zwei Instanzen erzeugt werden. Um dies zu verhindern, existiert eine Klassenvariable *_created*, welche bei der ersten Instanziierung gesetzt wird und bei jeder weiteren eine Warnung hervorruft. Sollte ein Nutzer diese Klassenvariable manuell zurücksetzen, liegt dies außerhalb der zu erwartenden Nutzung und wird nicht mehr abgefangen.

7.2 Logging

Beim Logging werden an interessanten Punkten (Datenbankinteraktion, Nutzeranmeldung, Fehler etc.) in der Anwendung, Nachrichten in eine Datei geschrieben. Die Implementierung erfolgt mittels des *logging*-Moduls aus Pythons Standardbibliothek. Dieses wurde so konfiguriert, dass die geschriebenen Datensätze folgende Informationen enthalten:

- Zeitstempel nach ISO 8601 im Format YYYY-MM-DDThh:mm:ss
- Logtyp (Fehler, Debugnachricht, Information, kritischer Fehler o.ä.)
- Dateiname
- Funktionsname, aus dem der Log kommt
- Nachricht

Somit sollte es im Fehlerfall möglich sein, leicht herauszufinden, wo eine Fehlerquelle liegt. Der Aufbau eines Log-Datensatzes ist dabei so, dass rechts die am meisten relevante und links die am wenigsten relevante Information liegt und jeder Abschnitt eine konstante Breite aufweist. Eine Beispieldatei könnte dann so aussehen:

```
1  2018-12-18T19:12:09 [ERROR   ] from          logger.<module>     "testerror"
2  2018-12-18T19:12:18 [ERROR   ] from            main.test         "testerror in main"
3  2018-12-18T19:12:38 [ERROR   ] from          logger.<module>     "testerror"
4  2018-12-18T19:12:38 [CRITICAL] from            main.info         "System shut down due to missing database"
5
```

Abbildung 7.1: Log-Datei Beispiel

Es ist zu sehen, dass sich die Datei Spaltenweise gut lesen lässt und leicht zurückverfolgbar ist, woher eine Nachricht stammt. Bei Nachricht 1 und 3 ist ein *<module>* im Funktionsslot zu sehen; dies bedeutet, dass die Nachricht nicht aus einer Unterfunktion sondern vom Modul selbst stammt. Die beiden Nachrichten aus *main* hingegen kamen aus den Funktionen *test* und *info*. Ebenfalls zu sehen ist, dass der Leitgedanke, des von rechts nach links absteigend relevanten Informationslevels sich sehr gut mit ISO 8601 kombinieren lässt, da die Sekunden

die wohl wichtigste Information der Zeit darstellen. Auf die Darstellung der Millisekunden (bzw. ggf. Microsekunden etc.) wurde verzichtet, da der Nutzer die Anwendung eher im Sekundentakt bedient und sich somit die Zeile nicht unnötig überlädt. Aus demselben Grund wäre es denkbar, auf die Jahreszahl zu verzichten.

7.3 Barcode Reader - die Klassen VideoStream und LazyVideoStream

Um Geräte identifizieren und markieren zu können, wird für jedes Gerät ein einzigartiger QR-Code erzeugt. Um diese einzulesen, wird *OpenCV* mit *ZBar* verwendet. Die Funktionalität des Einlesens ist im Modul *barcodereader* gekapselt.

Der Barcode Reader wird anfangs als Prozedur geschrieben. Sobald die Funktion gegeben ist, wird er in eine Klasse umgewandelt und für die später benötigte Parallelisierung vorbereitet[15]. Dabei gibt es grundsätzlich zwei Optionen:

1. Die zuerst implementierte: Der Thread arbeitet im Polling-Betrieb und stellt kontinuierlich das aktuellste Frame an seiner Schnittstelle bereit. Hierbei wird auf *Mutex* gesetzt um Threadsicherheit zu gewährleisten.

2. Beim Thread kann ein Frame angefragt werden, er antwortet anschließend mit einem Frame - realisiert wird dieser Vorgang durch *Atomic-Message- Queues*[16], welche ein weiterer Weg zur Threadsicherheit sind.

Beide Varianten setzen dabei auf einen sog. *Daemon-Thread* - einen Thread, der nach seinem Starten niemals mit seiner “Aufgabe” fertig ist (es gibt immer wieder neue Frames zum Auswerten).

Die Vermutung beim Auswählen des Prozesses ist, dass die Polling-basierte Variante schneller, jedoch rechenintensiver ist. Demgegenüber steht die Queue-basierte Variante, welche nur so viele Bilder wie nötig verarbeitet, jedoch nach Anfrage länger braucht, um ein Bild zurückzugeben. Daher wird mit einer einfachen Funktion eine statistische Gegenüberstellung der beiden Methoden durchgeführt. Die Funktion erhält hierbei einen Wert, inkrementiert diesen und meldet ihn zurück. Durchgeführt wurde dies für den Bereich [0:20e3].

[15] Eine genaue Erklärung einiger Prozesse des parallelen Programmierens erfolgt im Abschnitt **??**

[16] Eine Erklärung dieser Konstrukte, von Raymond Hettinger, der sie entwickelt/implementiert hat, gab es u.a. auf der PyCon Russia 2016 [**?**, 20.12.18 - 23:03 Uhr]

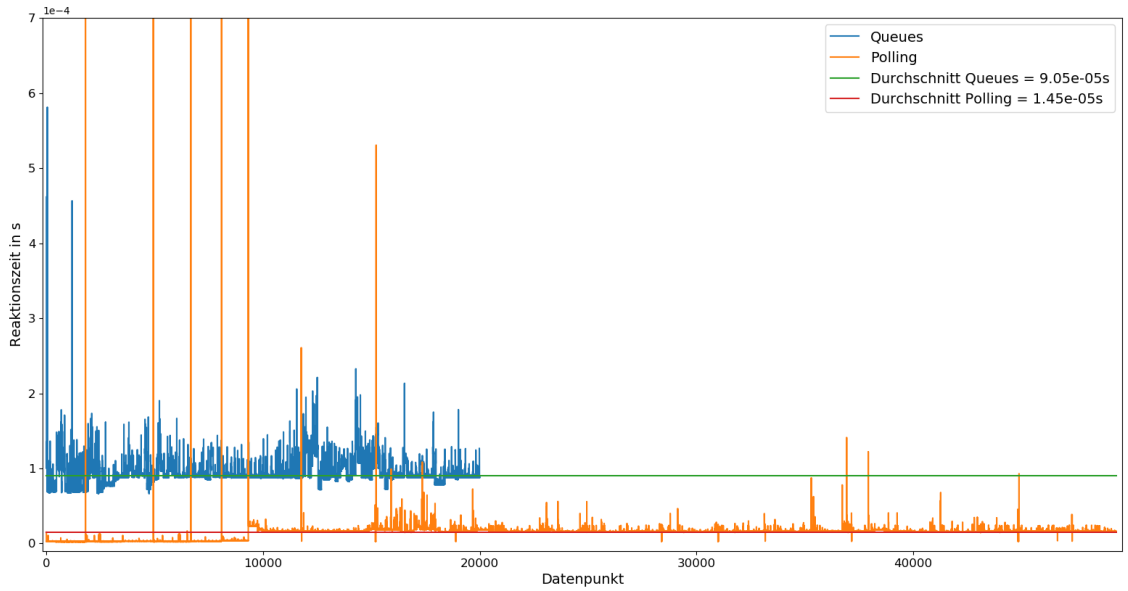


Abbildung 7.2: Gegenüberstellung von Polling und Queues

Wie in Abbildung **??** zu sehen ist, war die ursprüngliche Vermutung korrekt. Polling bietet wesentlich bessere Response Times, da es nicht reagiert, sondern proaktiv die Schnittstelle aktualisiert. Was jedoch auch zu erkennen ist: Bis das Ergebnis erreicht war, wurden bei der Queue-Variante lediglich die 20000 minimalen Lesezugriffe (Auf unseren use-case Übertragen also Aktualisierungen des Video-Feeds in der UI) durchgeführt - beim Polling fanden mehr als doppelt so viele Lesezugriffe statt (über mehrere Versuche ließ sich der tatsächliche Wert auf ~44e3 bis ~48e3 festmachen); es wurden konsekutiv mehrfach die gleichen Werte abgefragt. Es liegt also bei mehr Rechenzeit kein tatsächlicher Informationszuwachs vor. Betrachtet man die beiden Verfahren als Signale und bildet deren SNR, gilt:

$$SNR_{AMQ} = 1$$

$$SNR_{Polling} \approx 0,5$$

Bei diesen Zahlen ist selbstverständlich zu beachten, dass diese lediglich auf einer sample size von 1 beruhen und somit nur bedingt aussagekräftig sind.

Generell ist es bei einer Echtzeit-Kamera-Anwendung wünschenswert, bessere Response Times zu haben - im Praxisversuch zeigt sich jedoch, dass die Message Queues auf den Testgeräten ein flüssiges Bild liefern, daher wurde diese gewählt. Jedoch steht auch die Klasse für

Polling weiterhin bereit und ist wegen ihrer ähnlichen Schnittstelle, durch geringe Quellcodeänderung einzubinden oder alternativ für eine andere Anwendung einsetzbar.

In der konkreten Implementierung wird zum Anfragen eines Frames eine Queue genutzt; es stellt sich die Frage, wieso hierzu kein *Event* genutzt wurde. Solange lediglich ein Consumer auf den *LazyVideoStream* zugreift, hätte ein Event genauso funktioniert, jedoch treten bei mehreren Consumern Probleme auf. Wenn beispielsweise ein Thread, der die Barcodes auswertet, einen Frame anfragt, gleichzeitig allerdings die UI ein neues Frame zum Anzeigen anfragt, würden beide das Event starten, was beide Male intern das gleiche Flag auf True setzt; im Kontrast könnten beide separate Anfragen in der Request-Queue anlegen. Also würde der VideoStream auf das Event antworten, indem er ein Frame in seine Antwort-Queue legt. Einer der Konsumenten könnte dieses Frame herausnehmen, der andere jedoch würde leer ausgehen; bei der Anfragen-Queue-Variante hingegen würde er beide Anfragen mit einem Frame beantworten.

7.4 Datenbanksynchronisation

Im Modul *slots* existieren Fabrikfunktionen für die meisten Entitätsklassen der Datenbank. Um diese direkt bei ihrer Erzeugung mit der Datenbank zu synchronisieren, wird die recht simple Funktion *save_to_db* implementiert. Diese Funktion wird zunächst am Ende jeder Funktion, die eine neue Instanz erzeugt, aufgerufen. Dies ist zwar grundsätzlich nicht falsch, bedeutet jedoch, dass ein Programmierer, beim späteren Pflegen des Programms, sich eventuell nur die Funktionsschnittstelle ansieht und dabei übersieht, dass die Instanz direkt synchronisiert wird. Daher wurde im Zuge des Refactoring der *Function-Decorator synchronized* geschrieben, welcher direkt bei Funktionsdefinition klarmacht, dass diese Funktion synchronisierte Instanzen erzeugt.

Der Function-Decorator sei hier kurz erläutert:

Algorithmus 6 Definition des Function-Decorators

```
from functools import partial

def synchronized(function, decorated=False, *args, **kwargs):
    """Function-decorator to automatically add the instance a function\
        returns to DB"""
    if not decorated:
        return partial(synchronized, function, True)
    else:
        instance = function(*args, **kwargs)
        save_to_db(instance)
        return instance
```

Ein Function-Decorator ist so aufgebaut, dass er mit der @-Syntax, die ihm nachgestellte Funktion als ersten positionellen Parameter übergeben bekommt und diese mit seiner Rückgabe zum Definitionszeitpunkt der dekorierten Funktion redefiniert. In der Schnittstelle gibt es außerdem noch *decorated*, ein standardmäßig mit *False* vorbelegtes Flag, das später wichtig wird, sowie die Parameter **args* und ***kwargs*. Diese Parameter haben eine besondere Bedeutung, welche sie nicht durch ihren Namen, sondern durch die vorgestellten Asteriske erhalten (die Bezeichner nach * und ** sind grundsätzlich egal, jedoch sind *args* und *kwargs* gängig für „Durchreichungen“, da sie in diesem Funktionskontext keine weniger abstrakte Bedeutung haben); **args* ist ein Parameter, welcher stellvertretend für beliebig viele positionelle Argumente steht - **kwargs hingegen steht für beliebig viele Schlüsselwortparameter (daher der Name kwargs, von keyword arguments). Im Funktionskörper stellen diese dann ein Tupel bzw. ein Dictionary dar.

Wird diese Funktion nun das erste Mal aufgerufen, nämlich wenn die dekorierte Funktion definiert wird, defaulted das *decorated*-Flag auf *False*, was zur Folge hat, dass der erste if-Block direkt betreten wird. In diesem wird nun die Funktion definiert, welche bei späterem Aufruf der Originalfunktion an Stelle dieser aufgerufen wird. Die Funktion *partial* des Moduls *functools* der Standardbibliothek erlaubt die partielle Evaluation einer Funktion, dabei übergibt man ihr zuerst die Funktion, welche partiell evaluiert werden soll, und anschließend alle Parameter, welche sozusagen „vorbelegt“ werden. Hierbei sagen wir also, dass die zurückgegebene Funktion der Decorator selbst ist, welcher bei jedem Aufruf die dekorierte Funktion übergeben bekommt. Desweiteren setzen wir das decorated-Flag auf True.

Wird nun die dekorierte Funktion (in ihrem dekorierten Zustand) aufgerufen, wird die Funktion *function* (also die dekorierte Funktion in undekoriertem Zustand) aufgerufen, wobei als Parameter nach der Partiellevaluation nur noch *args* und *kwargs* übrig sind. Hier wird beim Aufruf explizites *Tuple Unpacking* für *args* und *Dictionary Unpacking* durch die *- bzw. **-Syntax eingesetzt, was dazu führt, dass die Originalfunktion effektiv einen „normalen“ Funktionsaufruf „erlebt“. Die dadurch erzeugte Instanz kann nun weiterverbeitet werden, bevor sie an den ursprünglichen Funktionsaufruf zurückgegeben wird.

8 Benutzerverwaltung

Da es in der Anwendung unter anderem darum geht, festzulegen, wer für welche Geräte verantwortlich ist, haben wir die Entscheidung getroffen, dass es nicht möglich sein sollte, die Verantwortlichkeiten eines anderen Mitarbeiters bearbeiten zu können. Dies erfordert ein Nutzersystem mit entsprechender Nutzerverwaltung. Aufgrund dieser Nutzerverwaltung wurden

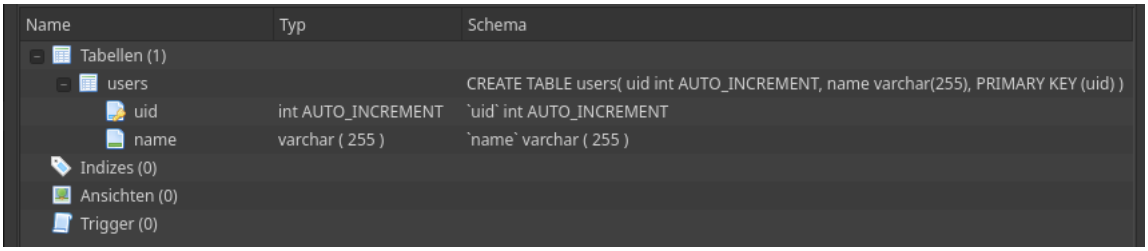
wir mit dem Problem konfrontiert, dass die Benutzer sich authentifizieren können müssen, hierfür wurde, klassischerweise, ein Passwort-System gewählt. Dies bringt das Problem der Passwortspeicherung mit sich, das, auch heute noch, in vielen Systemen eine Schwachstelle für potentielle Angriffe Dritter darstellt. Während es sich bei diesem System nicht um eine wirklich sicherheitskritische Anwendung handelt, sollte eine Anwendung unserer Ansicht nach dennoch so gestaltet sein, dass sie keine zu offensichtlichen Schwachstellen hat. Ein weiteres Problem entsteht dadurch, dass eine SQL-Datenbank im Hintergrund mit vom Nutzer zur Laufzeit eingegebenen Daten befüllt wird. Dies könnte, wenn nicht richtig abgefangen zur SQL-Injection genutzt werden.

8.1 SQL

Das zweite Problem lässt sich mit Python vergleichsweise leicht umgehen, indem man beim Nutzen von *sqlite3* die mitgelieferte ?-Syntax einsetzt, oder aber, wie wir, das bereits näher erläuterte SQLAlchemy nutzt, welches intern eine Validierung der Werte durchführt, die es in seinen SQL-Befehlen nutzt.

Um das Problem/Prinzip der SQL-Injection aufzuzeigen, folgt hier ein kurzes Beispiel.

Gegeben sei eine sehr einfache Datenbank, die Nutzer mit Namen speichert:



Wobei es möglich sein soll neue Nutzer über eine Kommandozeileneingabe des Namens hinzuzufügen.

Algorithmus 7 Beispiel von SQL-Injection

```
import sqlite3

connection = sqlite3.connect("test.db")
cursor = connection.cursor()

def insert_user_insecure(name):
    cursor.executescript(f"INSERT␣INTO␣users(name)␣VALUES␣({name})")

def insert_user_secure(name):
    cursor.execute("INSERT␣INTO␣users(name)␣VALUES␣(?)", (name,))

insert_user_insecure(input("Benutzername␣eingeben:"))
```

Wird nun über die Kommandozeile als Name z.B. '"); DROP TABLE users;--' eingegeben, wird die Tabelle aller Nutzer gelöscht. Korrekt umgesetzt ist das Ganze in *insert_user_secure* dargestellt. In diesem Fall wirft dieselbe Eingabe einen Fehler und die Datenbank wird nicht kompromittiert.

8.2 Kryptografie

Für die Passwörter gestaltet sich das Ganze nicht ganz so unkompliziert. Aufgrund dessen, dass der Quellcode offen ist und auch die Datenbank lokal liegt, ist es einem Angreifer ein Leichtes, jede eventuelle einfache Verschlüsselung zu umgehen. Daher werden bei unserer Lösung die Passwörter gar nicht gespeichert - stattdessen wird ein Hash-Algorithmus (zu Deutsch: Streuwertfunktion) eingesetzt und dieser Hash gespeichert. Ein Hash-Algorithmus ist eine Funktion, welche, idealerweise, in eine Richtung sehr schnell durchzuführen ist, in die andere dagegen einen sehr hohen Rechenaufwand benötigt. Im Idealfall ist dieser Rechenaufwand so hoch, dass sich eine "Einwegfunktion" ergibt, die nicht umzukehren ist. Ein klassisches Beispiel für einen simplen Hash-Algorithmus ist die einstellige Quersumme, hier wird rekursiv die Quersumme einer Zahl gebildet, bis nur noch eine Stelle übrig ist.

8.2.1 Benutzer anlegen

Das genaue Verfahren ist im folgenden Diagramm ersichtlich, der zugehörige Quellcode ist in der Klasse *User* als Methode *hash* zu finden.

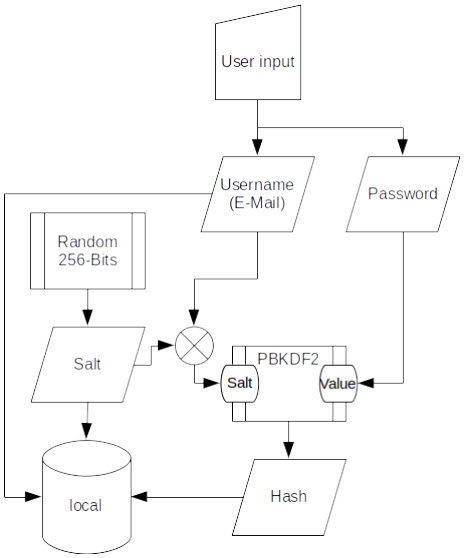


Abbildung 8.1: Passwort-Speicher-Vorgang

Es werden also aus den vom Nutzer getätigten Eingaben der Benutzername und das Passwort ausgewählt (bzw. werden diese in der Instanz einer Nutzerklasse abgelegt und daraus wieder abgerufen, dies wird allerdings später näher erläutert). Außerdem wird eine kryptografisch starke (im Gegensatz zu den, nicht für Kryptografieanwendungen geeigneten Zufallszahlen des normalen Pythonmoduls *random*) 256-Bit Zufallszahl generiert. Diese stellt ein erstes Salt dar. Im ersten Schritt wird nun die E-Mail-Adresse - ein String also - zu einer Abfolge aus Bytes codiert, welche dann als Ganzzahl interpretiert wird. Diese Zahl wird anschließend über ein logisches XOR mit dem generierten Salt verknüpft und stellt unser finales Salt dar. Die daraus resultierende Zahl sowie das Passwort werden dann wieder zu einer Folge aus Bytes konvertiert. Im zweiten Schritt werden nun diese beiden *bytes*-Instanzen über einen PBKDF2-Algorithmus miteinander verknüpft. In unserem Fall nutzt dieser intern einen HMAC-, welcher wiederum einen SHA-512-Algorithmus nutzt. Dieser Hash-Vorgang wird nun 9600-mal durchgeführt. Der daraus entstehende Hash ist unser “Endergebnis” und wird zusammen mit der Anfangs generierten Zufallszahl in der Datenbank abgelegt.

Der Hintergrund zu der vergleichsweise komplexen Erzeugung des finalen Salts ist Folgender:

- Wenn man kein Salt einsetzt, haben zwei Nutzer in der Datenbank denselben Hash, wenn sie dasselbe Passwort haben, was einem Angreifer im Bereich Social Engineering einen Angriffspunkt liefern würde (à la.: “Was haben diese Nutzer gemeinsam, dass sie eventuell als Passwort nutzen könnten”), bzw. lässt auf ein gängiges Passwort schließen.

- Wenn man nur die Zufallszahl als Salt nutzt, ist es theoretisch möglich, dass zwei Nutzer dieselbe Zufallszahl erhalten (Auch wenn diese Wahrscheinlichkeit praktisch vernachlässigbar klein ist, bei 2^{256} Werten, die die Zufallszahl annehmen kann und einer zu erwartenden Nutzerzahl ≤ 50) was im selben Problem wie der zuvor genannte Punkt münden würde.
- Wenn man nur den Benutzernamen als Salt einsetzt läuft man Gefahr, dass ein Nutzer einen sehr kurzen Benutzernamen wählt, wodurch ein vergleichsweise schnelles Durchprobieren/ Brute-Forcen durch alle Salts möglich ist.
- Durch die XOR-Verknüpfung von Nutzername und Zufallszahl hat man somit ein für jeden Nutzer garantiert einzigartiges Salt. Ein Angreifer kann dadurch eine Bruteforce Attacke nur für einen einzigen Nutzer auf einmal ausführen.

8.2.2 Benutzer anmelden

Beim Anmeldevorgang wird grundsätzlich derselbe Vorgang wie beim Erzeugen eines Nutzers durchgeführt, nur dass hier aus der Datenbank abgefragt, anstatt gespeichert wird.

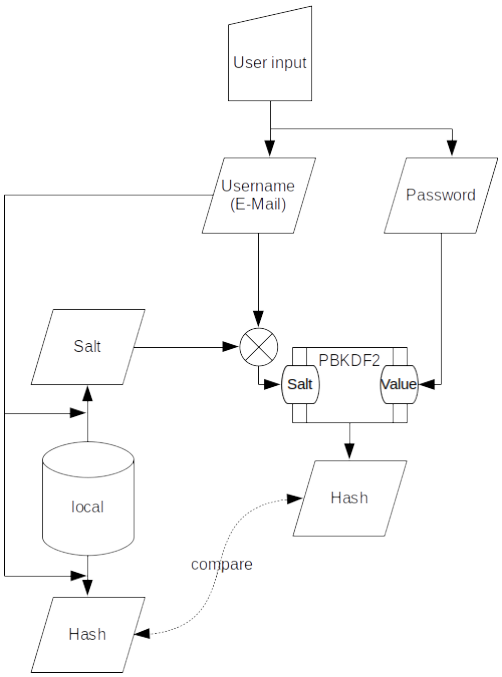


Abbildung 8.2: Anmelde-Vorgang

Der Nutzer gibt also wieder seine Daten ein. Anhand des Nutzernamens wird nun geprüft,

ob ein solcher Nutzer in der Datenbank vorhanden ist - wenn ja, wird dieser ausgelesen. Nun wird mit dem ausgelesenen Salt analog zum Passwort-Speicher-Vorgang ein Hash erzeugt. Stimmt dieser Hash mit dem des Nutzers aus der Datenbank überein, wird der Benutzer eingeloggt.

8.2.3 Benutzer hat sein Passwort vergessen

Wenn man ein System errichtet, bei dem ein Nutzer sich ein Passwort merken muss, ist die Wahrscheinlichkeit groß, dass er dieses vergisst. Daher gibt es einen Prozess, um das Passwort eines Nutzers zurücksetzen zu können. Da die Anwendung grundsätzlich offline laufen können sollte (nicht zuletzt aus sicherheitstechnischen Aspekten), wurde hier nicht die Variante des Rücksetzens per E-Mail-Token gewählt - stattdessen gibt es einen (oder auch mehrere) Admin-User die dem Nutzer ein automatisch generiertes Passwort zuweisen können, sodass er sich anmelden und selbst ein neues eingeben kann. Dieser Prozess lässt sich wie folgt visualisieren:

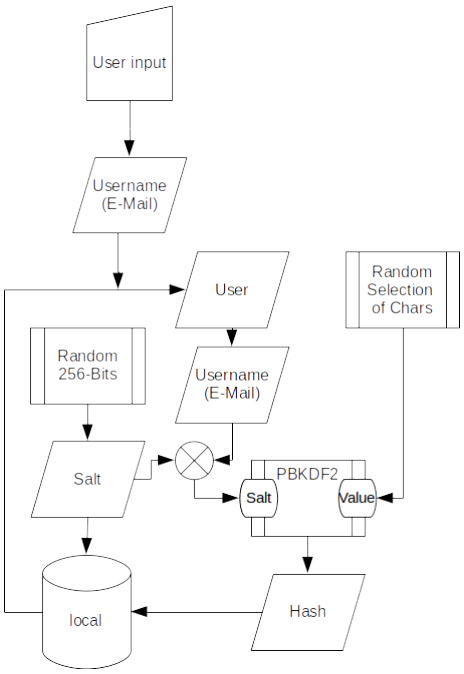


Abbildung 8.3: Rücksetzen eines Benutzerpassworts

Es wird also zunächst, über eine Nutzereingabe, der gewünschte User aus der Datenbank ausgelesen. Parallel dazu wird aus einer vordefinierten Zeichenfolge eine Auswahl von 15 Zeichen getroffen. Diese stellen das neue Passwort dar und werden in der UI angezeigt. Bei der

Zeichenfolge, aus der das Passwort generiert wird, wird darauf geachtet, dass jedes Zeichen klar lesbar ist. So wurden “O”, “0”, “I” und “l” ausgenommen da diese nicht immer klar zu unterscheiden bzw. leicht zu verwechseln sind. Abseits davon besteht die Zeichenfolge aus dem ganzen Alphabet in Groß- und Kleinschreibung, den Ziffern und einigen Sonderzeichen - insgesamt stehen somit 77 Zeichen zur Verfügung. Es stellt sich die Frage, ob es klüger wäre mehr Zeichen hinzuzufügen oder aber ein längeres Passwort zu wählen.

8.2.4 Mathematische Betrachtung der Passwort-Generation

Definiert man *n* als die Anzahl an verfügbaren Zeichen und *k* als Länge des Passwortes lässt sich die Anzahl an möglichen Passwörtern mit der Funktion

$$f(n,k) := n^k$$

darstellen. Möchte man nun wissen, ob es effizienter ist, weitere Zeichen hinzuzufügen oder das Passwort zu verlängern, kann man dies über eine Betrachtung der beiden partiellen Ableitungen

$$f_1(n,k) := \frac{\partial f}{\partial n} = k \cdot n^{k-1}$$

und

$$f_2(n,k) := \frac{\partial f}{\partial k} = n^k \cdot ln(n)$$

herausfinden. Diese beschreiben die Änderungsraten von $f(n,k)$ bei Änderung eines der beiden Parameter.

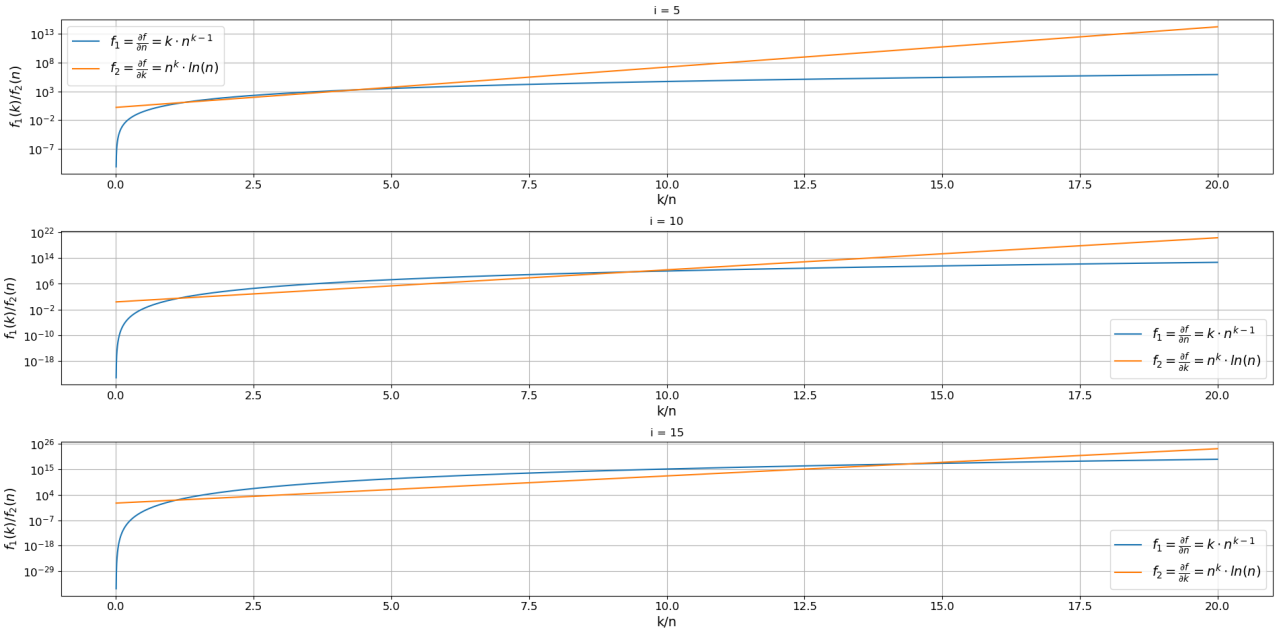


Abbildung 8.4: Vergleich von f_1 und f_2

Hier wurden die beiden Funktionen für die Werte 5, 10 und 15 für den Parameter, welcher

statisch gehalten wird (k bei f_1 und n bei f_2; dargestellt durch i), und in einem Bereich von 0 bis 20 des dynamischen Parameters geplottet. Wie man sieht, gibt es sowohl Bereiche, in denen f_1, wie auch welche, in denen f_2 effizienter (sprich größer) ist (siehe Schnittpunkte der Graphen). Daher sollte man eine allgemeinere Darstellung wählen. Verallgemeinert, sodass kein Parameter statisch gehalten wird, lässt sich die Funktion folgendermaßen darstellen:

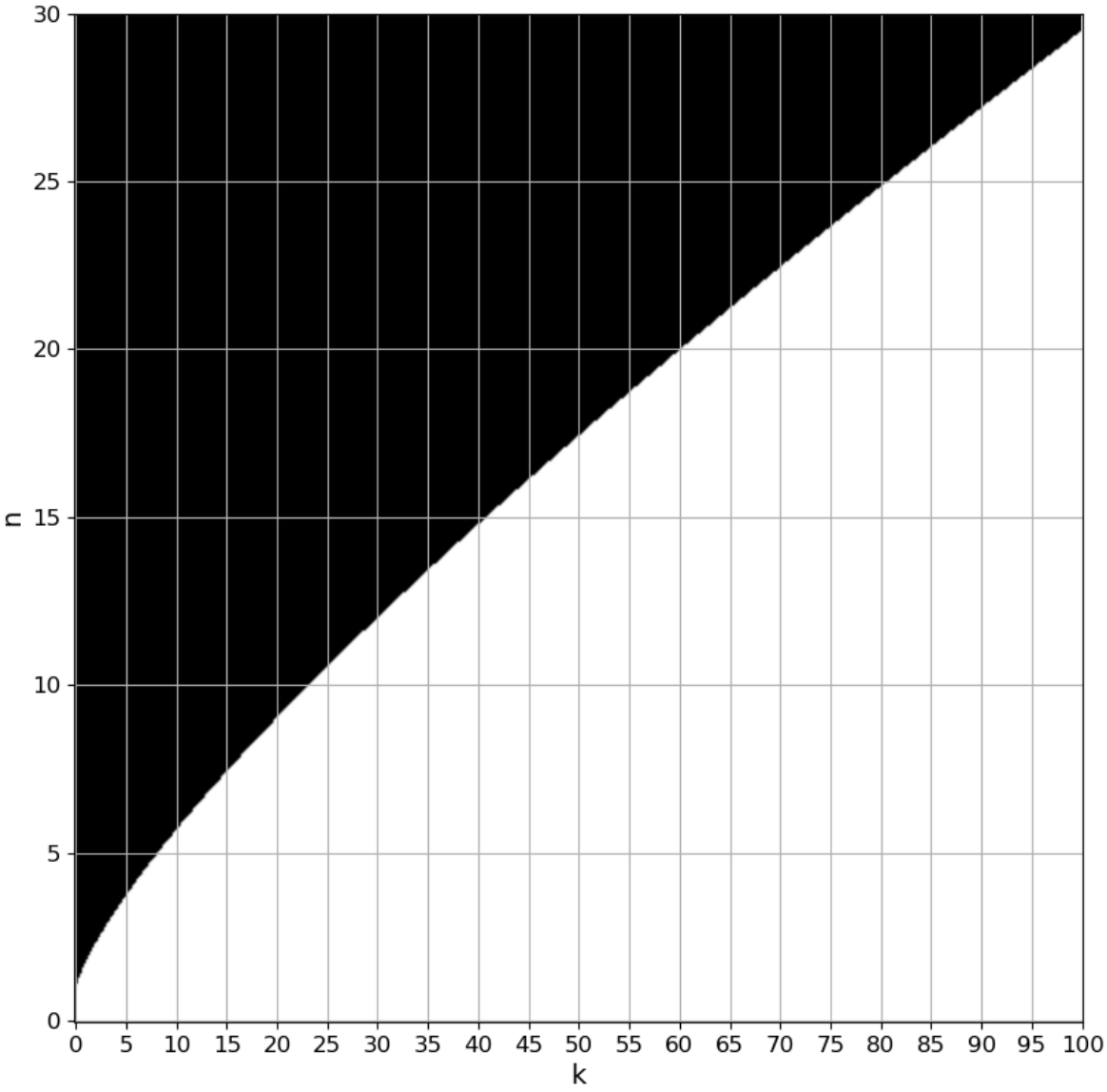


Abbildung 8.5: Heatmap aus f_1 und f_2

Wenn man eine neue Funktion $f_3(n,k)$ als

$$f_3(n,k) := f_1 - f_2 = \frac{\partial f}{\partial n} - \frac{\partial f}{\partial k} = k \cdot n^{k-1} - n^k \cdot ln(n)$$

definiert, und bei dieser alle Werte > 0 in Schwarz, die anderen in Weiß darstellt, hat man eine Karte, die angibt, ob bei einer bestimmten Koordinate die Erhöhung von k oder n die effizientere Wahl ist, um das generierte Passwort sicherer zu machen. Die Grenze zwischen den weißen und schwarzen Bereichen stellt die Nullstellen von f_3 dar. Setzt man also $f_3 = 0$ und löst dieses beispielsweise nach k auf ergibt sich

$$k(n) = n \cdot ln(n),$$

was ein einfacheres Überprüfen der Koordinaten zulässt.

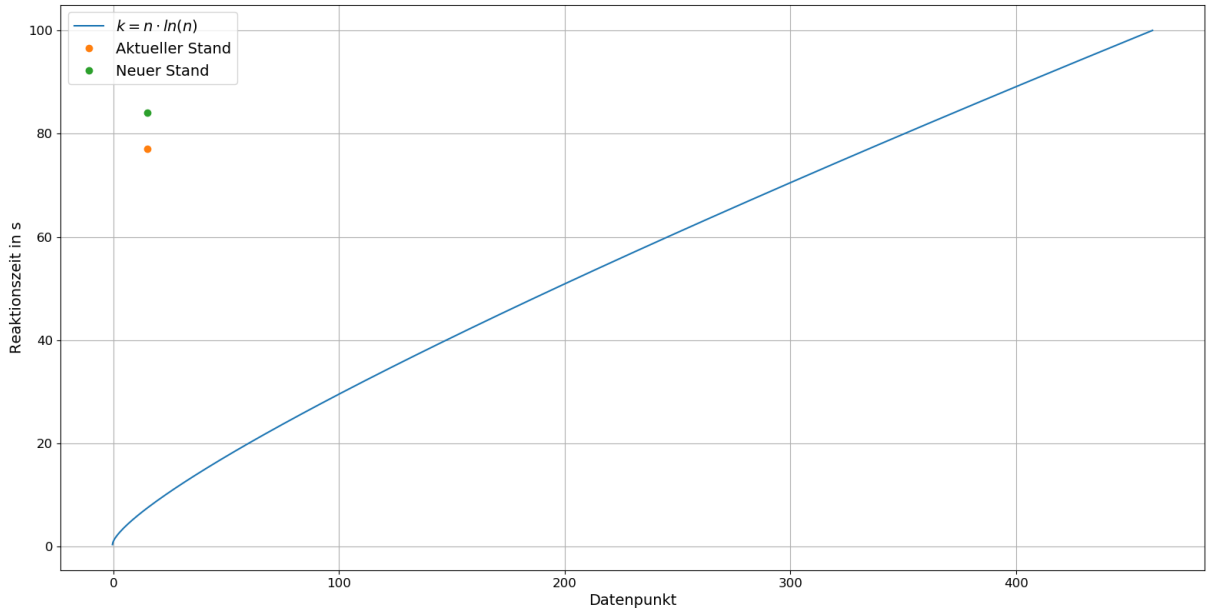


Abbildung 8.6: $k(n)$ aus f_3

Hier gilt, dass bei einem Punkt oberhalb der Kurve der Wert von f_3 positiv ist, wohingegen er unterhalb negativ ist. Daraus folgt, dass oberhalb der Kurve f_1 überwiegt, also die Rate der Änderung von k größer der von n ist.

Basierend auf diesen Daten bzw. Feststellungen wurde also der Zeichensatz um einige Zeichen erweitert. Es stehen nun 84 anstatt 77 Zeichen zur Verfügung, also 7 Zeichen mehr. Diese 7 Zeichen mehr geben bei gleicher Passwortlänge von 15 Zeichen $84^{15} - 77^{15} \approx 7,31 \cdot 10^{28} - 1.98 \cdot 10^{28} = 5.33 \cdot 10^{28}$ mehr Möglichkeiten. Oder anders ausgedrückt ist die Sicherheit des Passworts um den Faktor 3,69 erhöht worden.

Im Retrospekt ist zu erkennen, dass hierbei eine Funktion für die Schnittpunkte der beiden

partiellen Ableitungen gebildet wurde.

8.2.5 Admin hat sein Passwort vergessen

Allerdings kann es auch vorkommen, dass ein Admin sein Passwort vergisst - auch für so einen Fall steht ein Prozess zur Verfügung.

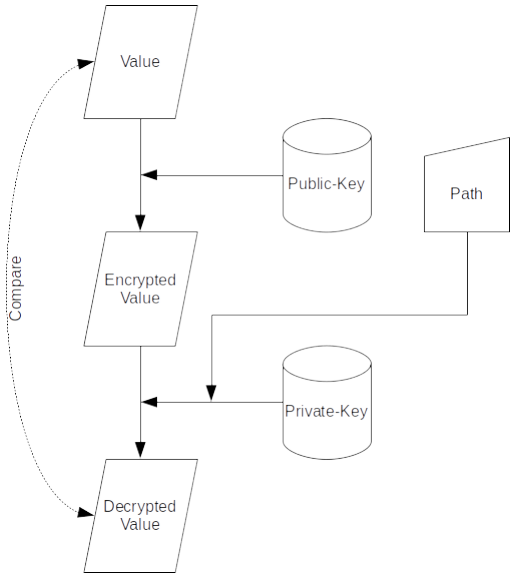


Abbildung 8.7: Verifizierung zum Rücksetzen eines Admin-Passworts

Wenn ein Admin sein Passwort zurücksetzen möchte, so erfordert dies eine weitere Sicherheitsstufe. Diese wird über eine asymmetrische Verschlüsselung mittels RSA, bzw. PKCS #1-OAEP realisiert. Das Verfahren läuft dabei wie folgt ab:

- Lokal ist ein Public-Key hinterlegt
- Mit diesem wird ein arbiträrer Wert (In der Implementierung ist “True” gewählt) verschlüsselt
- Anschließend wird probiert diesen Wert mit einem vom Nutzer gegebenen Schlüssel zu entschlüsseln
- Ist dies erfolgreich, wird ein neues Schlüsselpaar erzeugt und sowohl Public- wie auch Private-Key überschrieben - dies hat zur Folge, dass jedes Schlüsselpaar nur einmal gültig ist - so können andere Admins kontrollieren, ob ein Admin kürzlich sein Passwort zurückgesetzt hat

- Ab hier wird der normale Passwort-Rücksetz-Vorgang eines Users eingeleitet

Es ist so vorgesehen, dass ein Admin (bzw. Superadmin - je nachdem, wer Zugang zum Private-Key erhält) den Schlüssel auf einem USB-Stick o.ä. ablegt und diesen im Bedarfsfall einsteckt.

8.3 Automatisches Abmelden

Bei einem zentralen System, das mehreren Nutzern zugänglich ist, ist es wünschenswert, wenn ein Nutzer automatisch abgemeldet wird, sobald er eine bestimmte Zeit untätig ist. Dies beugt Missbrauch vor. Hierzu wurde die Klasse *Timeout* geschrieben.

Timeout ermöglicht es, einen Thread zu öffnen, welcher im Hintergrund mit einer bestimmten Abtastrate prüft, ob die eingestellte Zeit bereits verstrichen ist. Da es mit dieser Funktionalität an sich nur ein Timer wäre, implementiert die Klasse weiterhin die Methode *reset*, um den internen Timer immer dann zurückzusetzen, wenn ein bestimmtes Event aufgetreten ist. Jede *Timeout*-Instanz erhält außerdem ein aufrufbares Objekt und eine Liste mit positionellen Argumenten. Beim timeout wird diese Methode mit den übergebenen Argumenten ausgeführt und das Attribut *timed_out* auf *True* gesetzt. Bei den Argumenten ist zu beachten, dass *args* standardmäßig auf None gesetzt wird, zur Laufzeit des Konstruktors allerdings mit einer leeren Liste ersetzt wird, wenn dieser Standardwert vorhanden ist. Dies hat den Hintergrund, dass es generell eine schlechte Idee ist, eine leere Liste als Standardwert festzulegen. Standardwerte werden nämlich nicht beim Aufrufen einer Methode (oder Funktion oder Sonstigem), sondern sobald das zugehörige *def* ausgeführt wird, evaluiert. Bei immutablen Datentypen ist dies kein Problem, bei mutablen allerdings wird so stets dieselbe Instanz intern verwendet, was dazu führt, dass eine Änderung der Daten in einem Funktionsaufruf alle anderen mit beeinflusst.

Da die Klasse Timeout auf paralleler Programmierung basiert, sind hier einige Besonderheiten zu beachten. So wird auf *timer* sowohl von außerhalb, über *reset*, wie auch von innerhalb zugegriffen. Die Punkte, an denen diese Zugriffe stattfinden, bezeichnet man als *Critical Sections*, hier kann es zu sog. *Race Conditions* kommen[17]. Dies ist der Fall, wenn ein Thread gerade auf die Variable zugreift (z.B. liest) und dann der andere Thread die Kontrolle erhält und die Variable überschreibt und führt dazu, dass der lesende Thread einen falschen Wert erhält. Um diesem Problem vorzubeugen existiert die Klasse *Lock* des Moduls *threading.* Mithilfe dieser lassen sich Stellen, die nicht parallel verarbeitet werden dürfen, abriegeln (siehe Mutex). Dabei nimmt sich der Thread, welcher zuerst seine *Critical Section* erreicht, den *lock*, welcher als Kontext-Manager fungiert (intern werden die Methoden lock.acquire und lock.release aufgerufen) und führt den Code im *with*-Block aus. Der andere Thread kann hierbei weiterlaufen bis

[17] [**?**, S. 565]

er seine *Critical Section* erreicht, erst dann wird er blockiert, bis er selbst den *lock* akquirieren kann.

Zur Parallelisierung wurde ein Thread einem Prozess gegenüber bevorzugt, da der Overhead hier kleiner ist, es ist also weniger kostspielig einen neuen Thread zu öffnen. Außerdem ist der Datenaustausch zwischen Threads immens einfacher verglichen mit Prozessen. In unserem Fall können wir einfach Attribute der Klasse dafür benutzen; bei einem Prozess müsste hier auf Kommunikation mittels *sockets*, Serialisierung mit *pickle* oder temporäre Dateien zurückgegriffen werden (alternativ gibt es auch im Modul *multiprocessing* eine Klasse für *Atomic-Message Queues*).

Abschließend ist als wichtiger Punkt zu nennen, dass es für simple Timer-Aufgaben die Klasse *Timer* im Modul *threading* gibt. Diese unterstützt jedoch weder Funktionsargumente noch das Zurücksetzen des Timers, er kann lediglich gänzlich abgebrochen werden. Außerdem bietet *threading* weitere Klassen, welche in der Implementierung von Timer hätten genutzt werden können. So hätte man das Zurücksetzen beispielsweise mit einer *Event*-Instanz lösen können. Da dies jedoch keinen weiteren Nutzen oder bessere Performance bringt wurde darauf verzichtet.

8.4 Die Klasse *TelephoneNumber*

Beim Ablegen einer Nutzereingabe in einer Datenbank ist es ratsam diese Eingabe in irgendeiner Art und Weise zu standardisieren, sodass bei einer anderen Eingabe die jedoch dieselben Informationen enthält, erkannt wird, dass der Nutzer dasselbe meint. Bei einer E-Mail-Adresse oder einem Namen z.B. ist das eine sehr einfache Aufgabe, so kann man hier einfach die übergebene Zeichenkette komplett in Kleinbuchstaben umwandeln. Bei einer Telefonnummer hingegen ist diese Standardisierung eine nontriviale Aufgabe, da ein Nutzer eine Telefonnummer beispielsweise mit Länderkennzahl, Vorwahl, Durchwahl oder auch einfach nur als lokale Nummer eingeben könnte. Daher wird die Nutzereingabe intern umformatiert und eventuell fehlende Angaben (wie z.B. eine fehlende Länderkennzahl) extrapoliert, sodass jedesmal eine volle Nummer zur Verfügung steht. Sieht man sich die Norm DIN 5008 an, dann besteht eine vollständige Nummer aus Länderkennzahl (country-code), Vorwahl (area-code), Rufnummer (subscriber number) und Nebenstellennummer bzw. Durchwahl (extension)[18]. Es gilt in der Nutzereingabe ein Muster zu erkennen, anhand dessen sich feststellen lässt, welche Teile der Nummer er angegeben hat und wie diese lauten. Hierbei handelt es sich um ein Paradebeispiel für Pattern-Recognition (dt. Mustererkennung) mittels *regular expression* (dt. regulärer Ausdruck, kurz *Regex*). Diese kann man in Python mit dem Modul *re* der Standardbibliothek

[18] [**?**, 22.12.18 - 18:02 Uhr]

nutzen. Zunächst wird also ein Muster entwickelt, dass die einzelnen Teile erkennt. Der Muster-String ist:

r" (((\+\d{1,3})| (0)) ? ([1-9]+))? (\d+ ?)+ (-\d+)?"

Was zunächst wie eine Kette aus unzusammenhängenden Zeichen aussieht, symbolisiert alle syntaktischen Kombinationen die ein Nutzer zur Eingabe wählen kann, um dem System zu erlauben, die Telefonnummer korrekt zu erkennen. Der präfix *r* vor dem String gibt an, dass es sich um einen raw-String handelt - ein String ohne standard Escape-Sequenzen wie \n etc. also.

Eine Regex ist über die runden Klammern in Gruppen unterteilt, die jeweils eine Aufgabe unternehmen. Zu Beginn werden also drei Gruppen eröffnet, diese sind mit der Reihenfolge ihrer öffnenden Klammer nach als erste, zweite und dritte Gruppe benannt. Gruppe 3 enthält damit erste Unterpattern. Hierarchisch dargestellt ließe sich die Regex so aufbauen (Syntax in Anlehnung an XML):

Pattern

Gruppe_1

Gruppe_2

Gruppe_3

\+ steht für das Literal +

\d steht für eine beliebige Zahl zwischen 0 und 9

{1,3} steht dafür, dass der vorhergehende Ausdruck mindestens einmal und maximal 3 mal vorkommen darf

/**Gruppe_3**

| stellt ein logisches ODER zwischen Gruppen oder Ausdrücken dar

Gruppe_4

0 steht für das Literal 0

/**Gruppe_4**

/**Gruppe_2**

Leerzeichen steht für den String " ", also ein einzelnes Leerzeichen

? steht dafür, dass der vorhergehende Teil entweder 0- oder 1-mal, jedoch nicht öfter vorkommt; mit den bisherigen Gruppen lässt sich also feststellen, ob eine Eingabe mit Länderkennzahl oder eine Vorwahl mit beginnender 0 gewählt wurde oder keine von beidem vorhanden ist. Es kann auch sein, dass eine Leerstelle zwischen Länderkennzahl/ führender Null und dem Rest der Nummer steht

Gruppe_5

[1-9] steht für eine beliebige Zahl zwischen 1 und 9

+ steht dafür, dass der vorhergehende Ausdruck mindestens einmal, möglicherweise jedoch beliebig oft vorkommt

/Gruppe_5

Leerzeichen steht für den String " ", also ein einzelnes Leerzeichen

/Gruppe_1

? steht dafür, dass der vorhergehende Teil entweder 0- oder 1-mal, jedoch nicht öfter vorkommt; mit den bisherigen Gruppen lässt sich feststellen, ob eine Vorwahl angegeben wurde

Gruppe_6

\d steht für eine beliebige Zahl zwischen 0 und 9

+ steht dafür, dass der vorhergehende Ausdruck mindestens einmal, möglicherweise jedoch beliebig oft vorkommt

Leerzeichen steht für den String " ", also ein einzelnes Leerzeichen

? steht dafür dass der vorhergehende Teil entweder 0- oder 1-mal, jedoch nicht öfter vorkommt

/Gruppe_6

+ steht dafür, dass der vorhergehende Ausdruck mindestens einmal, möglicherweise jedoch beliebig oft vorkommt; bisher lässt sich feststellen, ob eine Vorwahl gegeben wurde und ob eine Rufnummer vorhanden ist

Gruppe_7

- steht für das Literal -

\d steht für eine beliebige Zahl zwischen 0 und 9

+ steht dafür, dass der vorhergehende Ausdruck mindestens einmal, möglicherweise jedoch beliebig oft vorkommt

/Gruppe_7

? steht dafür dass der vorhergehende Teil entweder 0- oder 1-mal, jedoch nicht öfter vorkommt; Hiermit wird geprüft ob eine Durchwahl angegeben wurde

/Pattern

Dieses Pattern ist als Klassenvariable der Klasse TelephoneNumber hinterlegt. Wird nun TelephoneNumber instanziiert, so wird über *re.match* zunächst ermittelt ob im übergebenen String

- also der Nutzereingabe - eine Telefonnummer ist, die auf das Pattern passt. Sofern dies nicht der Fall ist, wird ein Fehler geworfen. Andernfalls werden die einzelnen Gruppen ausgewertet und als Attribute der Instanz hinterlegt. So ist der Ländercode beispielsweise durch Gruppe 2 vertreten. Es wird überprüft, ob Gruppe zwei ein + enthält, wenn dies der Fall ist, wird der *_whitespacekiller* aufgerufen, eine statische Methode die alle Zeichen die keine Zahlen sind, aus der Kette entfernt (hier kommt eine weitere Regex zum Einsatz, diese ist jedoch so trivial, dass auf eine Erklärung verzichtet wird). Sofern kein + vorhanden ist, wird direkt 049 zurückgegeben, da davon ausgegangen werden kann, dass es sich in diesem Fall um eine deutsche Nummer handelt. Ähnlichen Verfahren folgend werden dann die Gruppen 5, 6 und 7 für Vorwahl, Rufnummer und Durchwahl ausgewertet.

Desweiteren implementiert die Klasse die Magic Methods *__str__* und *__format__*, welche ermöglichen eine Instanz als String zu evaluieren. Hierbei wird die Telefonnummer als vollständige DIN 5008-konforme Nummer zurückgegeben. Über *__format__* ist es möglich, die Instanz direkt in f-Strings o.Ä. einzubinden und die bekannten Formatspecifier für Strings zu nutzen.

Die Klasse besitzt außerdem eine von *Warning* abgeleitete Klasse, die sie emmitiert, wenn sie keine Nummer finden kann.

Durch diese ausgefeilte Implementierung des Telefonnummernextrahierens kann der Nutzer seine Telefonnummer, wenn er denn möchte, auch in einem Satz verpacken o.ä., sie sollte dennoch fehlerfrei erkannt werden.

9 PyQt5

Bei PyQt5 handelt es sich um ein GUI-Toolkit und Framework für Python. Grundsätzlich handelt es sich um einen Wrapper des Qt (lies “cute”, oftmals jedoch auch Q-T (englisch)) Frameworks für C++. Was Qt und PyQt seine Attraktivität, auch im professionellen, Bereich verleiht, ist vor allem, dass eine Qt-Anwendung grundsätzlich Cross-Platform ist und dabei auf allen Plattformen konsistent gut aussieht.

9.1 Darstellen der Verantwortlichkeiten als Baumstruktur

Aufgrund von guter Übersichtlichkeit haben wir uns dazu entschlossen die Verantwortlichkeiten als Baumstruktur darzustellen. PyQt5 stellt hierzu die Widgets QTreeView und QTreeWidget zur Verfügung, wobei QTreeView ein Model-View-Architektur- und QTreeWidget ein Item-basierendes Widget ist. Die MV-Architektur die PyQt über einige, teils abstrakte, Klassen bietet, ist dabei eine Variante der generell bekannten Model-View-Controller-Architektur, wobei der Controller mit dem View kombiniert wurde. Desweiteren wurde in PyQt der Delegate hinzugefügt, der managt, wie Daten im View gerendert werden und wie geänderte Daten der UI im Model abgelegt werden.

Da im Fall von TUInventory nicht gewünscht ist, dass Daten direkt im Baum geändert werden können und die Darstellung der Daten nur an dieser Stelle erfolgt, musste aufgrund von Kosten-Nutzen-Aspekten vorerst die Item-basierte Variante des *QTreeWidget* vorgezogen werden. Das Befüllen dieses Widgets erfolgt in der Methode *set_tree* der Klasse *MainDialog*.

Zu Beginn wird hier über den bereits erläuterten Session-Context-Manager eine neue Datenbank-Session geöffnet, in der wir nun alle *Responsibilit*ies abfragen können. Über diese wird nun iteriert, wobei in jeder Iteration für Ort, Benutzer und Gerät jeweils zuerst ein *QTreeWidgetItem* erzeugt wird. Dieses wird zum Einfügen in den Baum benötigt. Nun wird nach einem Siebprinzip nacheinander ermittelt, ob sich der Ort, Benutzer und das Gerät bereits im Baum befinden. Hierbei wird zuerst überprüft, ob sich im Wurzelverzeichnis *root* des Baums die *Location* der aktuellen *Responsibility* befindet. Sofern dies nicht der Fall ist, wird die Location mit all ihren untergeordneten Elementen eingefügt. Falls sich die *Location* bereits als sog. *TopLevelItem* im Baum befindet, wird überprüft, ob sich unter dieser *Location* bereits der *User* der aktuellen *Responsibility* befindet. Sofern dies nicht der Fall ist, wird der *User* unter seiner *Location* einsortiert und bekommt das aktuelle *Device* als Unterelement zugewiesen. Sollte der *User* bereits vorhanden sein, wird analog zur *Location* weiterverfahren. Dieser Prozess ist im folgenden Struktogramm dargestellt:

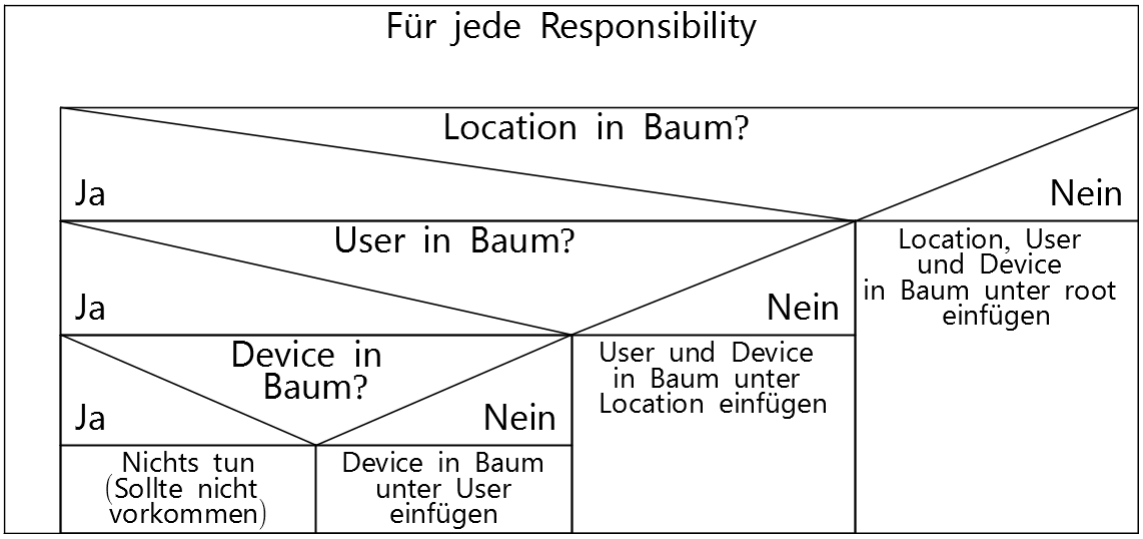


Abbildung 9.1: Struktogramm nach Nassi-Shneiderman zum Responsibility-Baum-Sieb

In diesem Teil des Programms werden oftmals Generator-Expressions und List-Comprehensions verwendet, daher sei im Folgenden kurz erläutert, wann man sich für welche entscheidet.

Der folgende Programmausschnitt zeigt einen Ausschnitt der interaktiven Konsole.

Algorithmus 8 Gegenüberstellung Generator Expression und List Comprehension

```
>>> from sys import getsizeof
>>> a = [i for i in range(10)]
>>> b = (i for i in range(10))
>>> getsizeof(a)
192
>>> getsizeof(b)
88
>>> type(a)
<class 'list'>
>>> type(b)
<class 'generator'>
>>> a[0]
0
>>> b[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'generator' object is not subscriptable
>>> sum(a)+sum(a)
90
>>> sum(b)+sum(b)
45
>>>
```

Zu Beginn wird hier die Funktion *getsizeof* des Moduls *sys* der Standardlibrary importiert, die es erlaubt, die Größe einer Instanz in Bytes zu ermitteln. Nun wird zunächst über eine List-Comprehension eine Liste mit den Zahlen 0 bis 9 definiert. Anschließend werden dieselben Zahlen mit einer Generator-Expression hinterlegt.

Vergleicht man hier die Größe der beiden Instanzen, wird klar, dass die Generator-Expression wesentlich kleiner ist. Führt man nun einen Typvergleich der Instanzen durch, sieht man, dass es sich bei *a* um eine Liste und bei *b* um einen Generator handelt. Damit lässt sich auch einfach der geringere Speicher-Footprint erklären: Bei einer List-Comprehension erfolgt die Auswertung sofort und somit liegt die ganze Liste im Speicher. Bei der Generator-Expression hingegen wird erst zum Zeitpunkt der Auswertung die Methode *__next__* der zugrundeliegenden Generatorinstanz aufgerufen, welche dann intern mittels *yield* den nächsten Wert ausgibt.

Jedoch kann man nicht immer Generatoren einsetzen, da diese einige entscheidende Nachteile haben. So ist es nicht möglich, mittels Index auf die Elemente eines Generators zuzugreifen. Außerdem ist ein Generator nach einer “Benutzung” “aufgebraucht”. Dies kann man sehen, wenn man zweimal die Summen unserer beiden Instanzen addiert. Für die List-Comprehension wird hier korrekterweise 90 ausgegeben ($\sum_{n=0}^{9} n = 45$), für die Generator-Expression jedoch nur 45, da sie beim zweiten Aufruf 0 zurückgibt. Hierzu sollte noch gesagt werden, dass, wenn

man einen Generator nicht über eine Generator-Expression, sondern manuell als Klasse/Funktion implementiert, dies teilweise umgangen werden kann.

Mit diesem Wissen ist nun auch leicht erklärbar, wann welche der Strukturen eingesetzt wurde:

- Generator-Expression immer, wenn sichergestellt ist, dass die Instanz nur einmal iteriert werden muss.
- List-Comprehension immer dann, wenn die Instanz mehrfach iteriert oder subscribed (dt. abonniert - also die [Index]-Notation) werden muss

9.2 Fazit

Es kann gesagt werden, dass PyQt (5) gegenüber anderen GUI-Paketen, wie z.B. tkinter aus der Python Standardbibliothek, klare Vorzüge hat. So ist der Code plattformunabhängig, die Bedienelemente sehen ansprechend aus und es ist einfach, auch größere Anwendungen noch gut zu strukturieren. Jedoch muss auch gesagt werden, dass PyQt5 teils nicht wirklich "pythonic" ist; so werden z.B. die Texte vieler Bedienelemente nicht über eine *property*/Zuweisung sondern über eine *setText*-Methode geändert. Da Qt eigentlich aus dem C++ Bereich kommt, ist dies verständlich, hätte im Wrapper jedoch geändert werden sollen.

10 Optimierung

Nachdem das Backend vollständig implementiert und größtenteils verknüpft ist, wird nach potentiellen Optimierungsstellen gesucht. Hierzu wird das Modul *cProfile* eingesetzt.

10.1 mouseMoveEvent

Beim Betrachten der Zeiten und Aufrufhäufigkeiten nach einer „Session" des Programms kann man z.B. sehen, dass sehr häufig das *mouseMoveEvent* des Hauptfensters aufgerufen wird. Diese Aufrufhäufigkeit lässt sich nicht optimieren[19], da das Event genutzt wird, um zu überprüfen, ob der Nutzer noch aktiv ist oder eventuell ausgeloggt werden sollte. Der Code, der dies übernimmt, ist sehr klein und nimmt auch nach einer mehrminütigen Session nur einige Millisekunden in Anspruch, dennoch wird hier eine Optimierung angestrebt.

[19] Außer durch ein anderes Event, wie z.B. Überprüfung, ob der Nutzer noch aktiv die UI bedient und Elemente anklickt

Algorithmus 9 Ursprünglicher Timeout reset

```
def mouseMoveEvent(self, QMouseEvent):
    if "timeout" in self.__dict__:
        self.timeout.reset()

""" ca. 100 Elemente
{'b_home_1': <PyQt5.QtWidgets.QCommandLinkButton object at 0\
    x7f48a0c9da68>,
 'b_search_places': <PyQt5.QtWidgets.QPushButton object at 0\
     x7f48982837 98>,
    [...]
 'ui': <__main__.MainDialog object at 0x7f48a17f3948>,
 'videoFeed': <PyQt5.QtWidgets.QLabel object at 0x7f48982839 48>}
 """
```

Es wird zu jedem *mouseMoveEvent* überprüft, ob aktuell ein Timer aktiv ist. Wenn dies der Fall ist, wird er zurückgesetzt. Diese Überprüfung ist notwendig, da andernfalls eine Exception geworfen wird, wenn die Klasse aktuell keine Instanz besitzt, die über die Methode *reset* verfügt. Außerdem ist ein Beispiel des Dictionaries einer *MainDialog-Instanz* abgebildet, wie es zum Zeitpunkt eines *mouseMoveEvents* aussehen könnte. Wie zu sehen ist, ist dieses Dictionary nicht unbedeutend klein (ca. 100 Elemente) was jedoch kein Problem darstellt, da ein Dictionary-Lookup i.d.R. eine Zeitkomplexität von O(1) (im worst case O(n), bei vielen Hash-Kollisionen etc.) besitzt. Dennoch gibt es hier eventuell eine effizientere Variante: die Funktion *hasattr*.

Algorithmus 10 Alternativer Timeout reset

```
def mouseMoveEvent(self, QMouseEvent):
    if hasattr(self, "timeout"):
        self.timeout.reset()
```

Ein Blick in den Python Sourcecode zeigt, dass diese in C implementiert ist[20], was einen Geschwindigkeitsschub in Aussicht stellt. Dies wird jedoch dadurch relativiert, dass auch Dictionaries in C implementiert sind[21].

Daher wird ein statistischer Versuch herangezogen, um zu sehen, ob eine Variante schneller ist als die andere:

[20] https://github.com/python/cpython/blob/master/Python/bltinmodule.c
[21] https://github.com/python/cpython/blob/master/Objects/dictobject.c

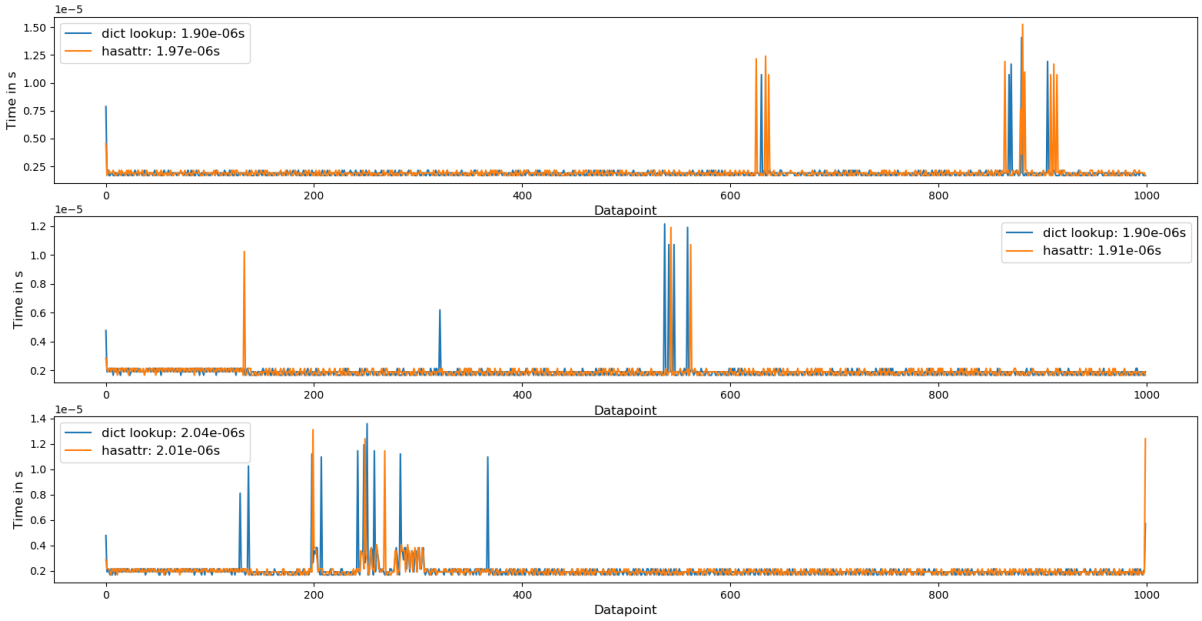


Abbildung 10.1: Vergleich von dict lookup und hasattr (jeweiliger Durchschnitt der Versuchsreihe in Legende)

Da hier zu sehen ist, dass beide Varianten praktisch gleich sind, könnte man hier bereits eine Entscheidung treffen, es wird jedoch noch das letzte Werkzeug herangezogen, um die Varianten zu vergleichen: Bytecode

Wie eingangs erwähnt wurde, ist Bytecode das Zwischenkompilat, welches tatsächlich vom Interpreter verwertet wird. Zugriff auf den Bytecode bekommt man entweder über das *__pycache__* oder den *__code__* Member.

Das Modul *dis* erlaubt es, diesen Bytecode in menschenlesbarer Form darzustellen.

Algorithmus 11 Bytecode zu beiden Varianten erzeugen

```
import dis


class B():
    def __init__(self):
        self.status = False

    def reset(self):
        self.status = True


class A():
    def __init__(self):
        self.timeout = B()
        for i in range(100):
            setattr(self, f"{i}", i)

    def dict_lookup_(self):
        if "timeout" in self.__dict__:
            self.timeout.reset()

    def hasattr_(self):
        if hasattr(self, "timeout"):
            self.timeout.reset()


a = A()


def hasattr_():
    if hasattr(a, "timeout"):
        a.timeout.reset()


def dict_lookup_():
    if "timeout" in a.__dict__:
        a.timeout.reset()


print(f"{'hasattr':-^50}")
print(dis.dis(hasattr_))
print("\n")
print(f"{'dict-lookup':-^50}")
print(dis.dis(dict_lookup_))
```

Algorithmus 12 Bytecode zu beiden Varianten

```
----------------------hasattr-----------------------
 29           0 LOAD_GLOBAL              0 (hasattr)
              2 LOAD_GLOBAL              1 (a)
              4 LOAD_CONST               1 ('timeout')
              6 CALL_FUNCTION            2
              8 POP_JUMP_IF_FALSE       20

 30          10 LOAD_GLOBAL              1 (a)
             12 LOAD_ATTR                2 (timeout)
             14 LOAD_ATTR                3 (reset)
             16 CALL_FUNCTION            0
             18 POP_TOP
        >>   20 LOAD_CONST               0 (None)
             22 RETURN_VALUE
None


--------------------dict-lookup--------------------
 34           0 LOAD_CONST               1 ('timeout')
              2 LOAD_GLOBAL              0 (a)
              4 LOAD_ATTR                1 (__dict__)
              6 COMPARE_OP               6 (in)
              8 POP_JUMP_IF_FALSE       20

 35          10 LOAD_GLOBAL              0 (a)
             12 LOAD_ATTR                2 (timeout)
             14 LOAD_ATTR                3 (reset)
             16 CALL_FUNCTION            0
             18 POP_TOP
        >>   20 LOAD_CONST               0 (None)
             22 RETURN_VALUE
None
```

Bytecode liest sich wie folgt (Spalten von links nach rechts aufsteigend nummeriert)[22]:

1. Zeilennummer des Quellcodes
2. Hier nicht vorhanden (Aktuelle Anweisung beim schrittweisen Ausführen)
3. Mögliches Sprungziel markiert mit >>
4. Adresse der Anweisung
5. Anweisungsname

[22] [**?**, 24.12.18 - 20:27 Uhr]

6. Anweisungsparameter

7. Interpretation der Parameter in Klammern

Um den Sprung in beiden Varianten direkt zu Anfang zu erklären: Wenn das *if* als *False* evaluiert, gibt die Funktion ein *None* zurück.

Ebenso können vorab die Zeilen 30 und 35 erklärt und gleichzeitig eine Einführung in Bytecode gegeben werden[23], da sie bei beiden gleich sind:

- Es wird zuerst die Globale *a* auf den *Evaluation-Stack* (im Folgenden E-Stack) geladen.
- Anschließend wird das TOS (Top of Stack)-Element, also *a*, mit der Anweisung *getattr (TOS, co_names[namei]) -> getattr (a, co_names[timeout])* ersetzt.
- Dasselbe geschieht nun mit *reset.*
- Nun liegt *reset* oben auf dem E-Stack und wird mit 0 positionellen Argumenten aufgerufen, der Rückgabewert wird auf den E-Stack gepusht.
- Die folgende Anweisung gibt an, dass der Rückgabewert nicht weiter verwertet und somit verworfen wird.
- Die letzen beiden Zeilen sind die bereits erklärten Anweisungen des impliziten Funktionsreturn.

Die Abfolge für hasattr ist wie folgt (Alle Ladeanweisungen erfolgen auf den E-Stack):

- Laden der Globalen *hasattr*
- Laden der Globalen *a*
- Laden der Konstante *'timeout'*
- Aufruf einer Funktion mit zwei positionellen Argumenten, dies ist der Aufruf von *hasattr* mit *a* und *'timeout'*
- Hier folgt der Sprung falls das TOS-Element *False* ist.

Wohingegen die Abfolge für den *dict-lookup* ist:

- Laden der Konstante *'timeout'*

[23] Einige Aspekte sind hier zum besseren Verständnis vereinfacht dargestellt (z.B. werden eigentlich *frame*-Objekte auf die Stacks gelegt)

- Laden Globale *a*
- Laden des Attributs *__dict__* von *a*
- Vergleichsoperation *in* auf *__dict__* und *'timeout'*
- Hier folgt der Sprung falls das TOS-Element *False* ist.

Der Unterschied besteht also darin, dass eimal eine Globale mehr geladen wird (globale Lookups können kostspielige Operationen sein) und ein Funktionsaufruf (generell auch eher kostspielig) stattfindet, wohingegen beim Anderen mal ein Attribut abgerufen wird und eine Vergleichsoperation stattfindet. Sieht man sich die Implementierung des Interpreters an[24], sieht man, dass beide Varianten mittels Branch-Prediction optimiert werden. In diesem konkreten Fall wird auch durch den Bytecode nicht klar, dass eine Variante generell schneller ist als eine andere (was durch die Messung unterstützt wird).

Schließlich kann hier keine Optimierung erfolgen, dennoch wird der Code dahingehend abgeändert, dass er *hasattr* einsetzt, da dies die leichter zu lesende und somit schlicht bessere Variante ist.

[24] https://github.com/python/cpython/blob/master/Python/ceval.c, 24.12.18 - 21:14 Uhr

A Quellcode

A.1 barcodereader.py

```
1  """Allows opening a camera feed and finding barcodes in it"""
2
3  from collections import Counter
4  from time import sleep
5  from sys import stderr
6  import threading
7  import queue
8
9  import cv2
10 import numpy as np
11 from pyzbar import pyzbar
12
13 from utils import parallel_print
14
15
16 class VideoStream(threading.Thread):
17     """Class for reading barcodes of all kinds from a video feed and \
           marking them in the image
18     Be sure you can't use the LazyVideoStream instead!
19     """
20     def __init__(self, target_resolution=None, camera_id=0):
21         """
22         Args:
23             target_resolution: Tuple of (width, height) to set the \
                   final resolution of the frames
24             camera_id: The id of the camera that's to be used (if your\
                    system only has one it's zero)
25         """
26         super().__init__()
27         self.camera = Camera(camera_id)
28         self.camera_id = camera_id
29         self.barcodes = []
30         self._mirror = False
31         self._abort = False
32         self.frame_lock = threading.Lock()
33         self.gp_lock = threading.Lock()
```

```
34          self._target_resolution = target_resolution
35          self._frame = np.zeros((1, 1))
36
37      def _set_frame(self, frame):
38          with self.frame_lock:
39              self._frame = frame
40
41      def _get_frame(self):
42          with self.frame_lock:
43              frame = self._frame
44          if self.mirror:
45              frame = cv2.flip(frame, 1)
46          if self.target_resolution:
47              frame = cv2.resize(frame, (self.target_resolution[0], self\
                    .target_resolution[1]))
48          return frame
49
50      def _set_mirror(self, mirror):
51          with self.gp_lock:
52              self._mirror = mirror
53
54      def _get_mirror(self):
55          with self.gp_lock:
56              return self._mirror
57
58      def _set_abort(self, abort):
59          with self.gp_lock:
60              self._abort = abort
61
62      def _get_abort(self):
63          with self.gp_lock:
64              return self._abort
65
66      def _set_target_resolution(self, resolution):
67          with self.gp_lock:
68              self._target_resolution = resolution
69
70      def _get_target_resolution(self):
71          with self.gp_lock:
72              return self._target_resolution
```

```
73
74      frame = property(fget=_get_frame, fset=_set_frame)
75      mirror = property(fget=_get_mirror, fset=_set_mirror)
76      abort = property(fget=_get_abort, fset=_set_abort)
77      target_resolution = property(fget=_get_target_resolution, fset=\
            _set_target_resolution)
78
79      @staticmethod
80      def rect_transformation(x, y, width, height):
81          """Transform rectangle of type "origin + size" to "two-point"
82          Args:
83              x (int): x coordinate of origin
84              y (int): y coordinate of origin
85              width (int): width of rectangle
86              height (int): height of rectangle
87          Returns:
88              Tuple of tuple of int with x-y-coordinate pairs for both \
                    points
89          """
90          return ((x, y), (x + width, y + height))
91
92      def find_and_mark_barcodes(self, frame):
93          barcodes = pyzbar.decode(frame)
94          found_codes = []
95          for barcode in barcodes:
96              barcode_information = (barcode.type, barcode.data.decode("\
                    utf-8"))
97              if barcode_information not in found_codes:
98                  found_codes.append(barcode_information)
99              poly = barcode.polygon
100             poly = np.asarray([(point.x, point.y) for point in poly])
101             poly = poly.reshape((-1,1,2))
102             cv2.polylines(frame, [poly] ,True, (0,255,0), 2)
103             cv2.rectangle(frame, *self.rect_transformation(*barcode.\
                    rect), (255, 0, 0), 2)
104             x, y = barcode.rect[:2]
105             cv2.putText(frame, "{}({})".format(*barcode_information), \
                    (x, y-10), cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1)
106         return frame, found_codes
107
```

```
108     def run(self):
109         with self.camera as camera:
110             while not self.abort:
111                 frame = camera.read()[1]
112                 marked_frame, found_codes = self.\
                        find_and_mark_barcodes(frame)
113                 self.frame = marked_frame
114                 if found_codes:
115                     self.barcodes.append(found_codes)
116
117
118 class LazyVideoStream(threading.Thread):
119     """Class for reading barcodes of all kinds from a video feed and \
            marking them in the image
120     This Lazy implementation only processes frames on demand
121
122         Args:
123             target_resolution: Tuple of (width, height) to set the \
                    final resolution of the frames
124             camera_id: The id of the camera that's to be used (if your\
                     system only has one it's zero)
125
126         Attributes:
127             camera: Instance of Camera with for given camera_id
128             camera_id: Given camera_id
129             _mirror: Set to True to mirror the frame
130             gp_lock: general purpose lock for property access
131             _target_resolution: Tuple of (width, height) to set the \
                    final resolution of the frames
132             request_queue: Queue that's used to request a frame, \
                    request by putting True
133             frame_queue: Queue that answer frames get pushed into
134
135         Properties:
136             mirror: gp_lock locked _mirror
137             target_resolution: gp_lock locked _target_resolution
138     """
139     def __init__(self, target_resolution=None, camera_id=0):
140         super().__init__()
141         self.camera = Camera(camera_id)
```

```
142         self.camera_id = camera_id
143         self._mirror = False
144         self.gp_lock = threading.Lock()
145         self._target_resolution = target_resolution # (width, height)
146         self.request_queue = queue.Queue()
147         self.frame_queue = queue.Queue()
148         with self.camera:
149             pass
150 
151     def _set_mirror(self, mirror):
152         with self.gp_lock:
153             self._mirror = mirror
154 
155     def _get_mirror(self):
156         with self.gp_lock:
157             return self._mirror
158 
159     def _set_target_resolution(self, resolution):
160         with self.gp_lock:
161             self._target_resolution = resolution
162 
163     def _get_target_resolution(self):
164         with self.gp_lock:
165             return self._target_resolution
166 
167     mirror = property(fget=_get_mirror, fset=_set_mirror)
168     target_resolution = property(fget=_get_target_resolution, fset=\
        _set_target_resolution)
169 
170     @staticmethod
171     def rect_transformation(x, y, width, height):
172         """Transform rectangle of type "origin + size" to "two-point"
173 
174             Args:
175                 x (int): x coordinate of origin
176                 y (int): y coordinate of origin
177                 width (int): width of rectangle
178                 height (int): height of rectangle
179 
180             Returns:
```

```
181                 Tuple of tuple of int with x-y-coordinate pairs for \
                        both points
182         """
183         return ((x, y), (x + width, y + height))
184
185     def find_and_mark_barcodes(self, frame):
186         """Find barcodes in given frame
187
188             Args:
189                 frame: Frame that barcodes should be detected in
190
191             Returns:
192                 Tuple of (frame where barcodes are marked, list of all\
                        found codes in frame)
193         """
194         barcodes = pyzbar.decode(frame)
195         found_codes = []
196         for barcode in barcodes:
197             barcode_information = (barcode.type, barcode.data.decode("\
                    utf-8"))
198             if barcode_information not in found_codes:
199                 found_codes.append(barcode_information)
200             poly = barcode.polygon
201             poly = np.asarray([(point.x, point.y) for point in poly])
202             poly = poly.reshape((-1,1,2))
203             cv2.polylines(frame, [poly] ,True, (0,255,0), 2)
204             cv2.rectangle(frame, *self.rect_transformation(*barcode.\
                    rect), (255, 0, 0), 2)
205             x, y = barcode.rect[:2]
206             cv2.putText(frame, "{}({})".format(*barcode_information), \
                    (x, y-10), cv2.FONT_HERSHEY_PLAIN, 1, (0, 0, 255), 1)
207         return frame, found_codes
208
209     def request(self):
210         """Convenience function to abstract the request_queue from the\
                user"""
211         self.request_queue.put(True)
212
213     def run(self):
214         """Start connection to camera and answer requests"""
```

```
215         with self.camera as camera:
216             while True:
217                 self.request_queue.get()
218                 frame = camera.read()[1]
219                 marked_frame, found_codes = self.\
                        find_and_mark_barcodes(frame)
220                 if self.target_resolution:
221                     marked_frame = cv2.resize(marked_frame, (self.\
                            target_resolution[0], self.target_resolution\
                            [1]))
222                 self.frame = marked_frame
223                 self.frame_queue.put((frame, found_codes))
224                 self.request_queue.task_done()
225
226
227 class Camera():
228     """Context Manager for video streams"""
229     def __init__(self, camera_id=0):
230         self.camera_id = camera_id
231
232     def __enter__(self):
233         self.camera = cv2.VideoCapture(self.camera_id)
234         if not self.camera.isOpened():
235             raise IOError(f"Failed to open camera {self.camera_id}")
236         while not self.camera.read()[0]:
237             pass
238         return self.camera
239
240     def __exit__(self, exc_type, exc_value, traceback):
241         self.camera.release()
242
243 if __name__ == "__main__":
244     lazy_feed = LazyVideoStream()
245     lazy_feed.start()
246     window = "window"
247     cv2.namedWindow(window)
248
249     while True:
250         lazy_feed.request_queue.put(True)
251         frame, codes = lazy_feed.frame_queue.get()
```

```
252             lazy_feed.frame_queue.task_done()
253             cv2.imshow(window, frame)
254             cv2.waitKey(1)
255             if codes:
256                 parallel_print(codes)
```

A.2 classes.py

```
1  """All classes of the database connection reside here
2  This also handles database initialization
3  """
4
5  from collections import Counter
6  import hashlib
7  import re
8  from secrets import randbits
9  from threading import Lock, Thread
10 from time import sleep, time
11
12 import cv2
13 import sqlalchemy
14 from sqlalchemy import Boolean, Column, Float, Integer, LargeBinary, \
       String
15 from sqlalchemy.ext.declarative import declarative_base
16 from PyQt5.QtGui import QImage, QPixmap
17
18 from logger import logger
19 from utils import absolute_path
20
21 orm = sqlalchemy.orm
22
23 Base = declarative_base()
24 if __name__ == "__main__":
25     print("")
26     if not input("Warning! Do you really want to delete the database \
           and write some example data to it? [y/N]: ") in ("y", "Y"):
27         exit()
28     print("")
29     logger.info("Database cleared! This was authorized via a prompt")
30     with open(absolute_path("test.db"), "w") as f:
```

```
31          f.flush()
32  engine = sqlalchemy.create_engine(f"sqlite:///{absolute_path('test.db\
        ')}", echo=False)
33  # engine = sqlalchemy.create_engine("sqlite:///:memory:", echo=False)
34
35
36  class BigInt(sqlalchemy.types.TypeDecorator):
37      """SQLAlchemy datatype for dealing with ints that potentially \
            overflow a basic SQL Integer"""
38      impl = sqlalchemy.types.String
39      def process_bind_param(self, value, dialect):
40          """Gets called when writing to db"""
41          return str(value)
42
43      def process_result_value(self, value, dialect):
44          """Gets called when reading from db"""
45          return int(value)
46
47
48  def setup_context_session(engine):
49      """Factory for contextmanagers for Session objects
50      Initialize a ContextSession class
51
52          Args:
53              engine (sqlalchemy.engine.base.Engine): Engine that's \
                    bound to the sessionmaker
54
55          Example:
56              engine = sqlalchemy.create_engine('sqlite:///:memory:')
57              CSession = setup_context_session(engine)
58              with CSession() as session:
59                  session.add(user1)
60                  session.add(user2)
61      """
62      class ContextSession():
63          _engine = engine
64          _Session = orm.sessionmaker(bind=engine)
65          def __enter__(self):
66              self.session = self._Session()
67              return self.session
```

```
68          def __exit__(self, exc_type, exc_value, traceback):
69              if exc_value or exc_type or traceback:
70                  self.session.rollback()
71                  return False
72              else:
73                  self.session.commit()
74                  self.session.close()
75                  return True
76      return ContextSession
77
78
79  class Producer(Base):
80      """Represents a producer of articles"""
81      __tablename__ = "producers"
82      uid = Column(Integer, primary_key=True)
83      name = Column(String)
84      articles = orm.relationship("Article", backref="producer")
85      def __init__(self, name, uid=None):
86          self.uid = uid
87          self.name = name.title()
88
89
90  class Article(Base):
91      """Represents an article"""
92      __tablename__ = "articles"
93      uid = Column(Integer, primary_key=True)
94      name = Column(String)
95      producer_uid = Column(Integer, sqlalchemy.ForeignKey("producers.\
            uid"))
96      last_price = Column(Float(asdecimal=True))
97      devices = orm.relationship("Device", backref="article")
98      def __init__(self, name, producer=None, uid=None):
99          self.uid = uid
100         self.name = name
101         self.producer = producer
102
103
104 class Device(Base):
105     """Represents a device"""
106     __tablename__ = "devices"
```

```
107     uid = Column(Integer, primary_key=True)
108     article_uid = Column(Integer, sqlalchemy.ForeignKey("articles.uid"\
            ))
109     name = Column(String) # not currently used
110     code = Column(String)
111     #location_uid = Column(Integer, sqlalchemy.ForeignKey("locations.\
            uid"))
112     responsibilities = orm.relationship("Responsibility", backref="\
            device")
113     def __init__(self, code=None, uid=None):
114         self.uid = uid
115         self.code = code
116 
117     def __str__(self):
118         return f"{self.article.name} mit ID {self.uid}"
119 
120 
121 class PhoneNumber(Base):
122     """Get a Phone Number from raw input string
123     PhoneNumber can also be stored in database table
124 
125         Args:
126             raw_string (str): String that holds the number
127 
128         To do:
129             - config to set locale (subscriber number prefix and \
                    country code)
130               probably not possible due to precision limitations of \
                      system-locale
131     """
132     __tablename__ = "phone_numbers"
133     user_uid = Column(Integer, sqlalchemy.ForeignKey("users.uid"), \
            primary_key=True)
134     raw_string = Column(String)
135     country_code = Column(String)
136     area_code = Column(String)
137     subscriber_number = Column(String)
138     extension = Column(String)
139     pattern = r"(((\+\d{1,3})|(0)) ?([1-9]+) )?(\d+ ?)+(-\d+)?"
140     def __init__(self, raw_string):
```

```
141         self.raw_string = raw_string
142         if not self.raw_string:
143             self.country_code = ""
144             self.area_code = ""
145             self.subscriber_number = ""
146             self.extension = ""
147             self.match = None
148             return
149         self.match = re.match(self.pattern, self.raw_string)
150         if not self.match:
151             raise self.NoNumberFoundWarning(raw_string)
152         self.country_code = self._extract_country_code()
153         self.area_code = self._extract_area_code()
154         self.subscriber_number = self._extract_subscriber_number()
155         self.extension = self._extract_extension()
156 
157     @staticmethod
158     def _whitespacekiller(string):
159         """Remove all non-number characters from a string"""
160         return re.sub(r"\D", "", string)
161 
162     def _extract_country_code(self):
163         country_code = self.match.group(2)
164         if country_code and "+" in country_code:
165             return self._whitespacekiller(country_code)
166         else:
167             return "049"
168 
169     def _extract_area_code(self):
170         area_code = self.match.group(5) if self.match.group(5) else "\
                9321"
171         return self._whitespacekiller(area_code)
172 
173     def _extract_subscriber_number(self):
174         subscriber_number = self.match.group(6)
175         return self._whitespacekiller(subscriber_number)
176 
177     def _extract_extension(self):
178         extension = self.match.group(7)
179         extension = "" if not extension else extension
```

```
180         return self._whitespacekiller(extension)
181
182     def __str__(self):
183         """Build string of the telephone number based on DIN 5008"""
184         extension = f"-{self.extension}" if self.extension else ""
185         return f"+{self.country_code} {self.area_code} {self.\
                subscriber_number}{extension}"
186
187     def __format__(self, format_spec):
188         return f"{str(self):{format_spec}}"
189
190     class NoNumberFoundWarning(Warning):
191         def __init__(self, raw_string):
192             self.raw_string = raw_string
193         def __str__(self):
194             return f'No telephonenumber was found in: "{self.\
                    raw_string}"'
195
196
197 class Location(Base):
198     """Represents a physical location where a Device or User (or \
            Responsibility) may be located"""
199     __tablename__ = "locations"
200     uid = Column(Integer, primary_key=True)
201     name = Column(String, unique=True)
202     #devices = orm.relationship("Device", backref=orm.backref("\
            location", uselist=False))
203     responsibilities = orm.relationship("Responsibility", backref="\
            location")
204     def __init__(self, name="", uid=None):
205         self.name = name.title()
206         self.uid = uid
207
208     def __str__(self):
209         return self.name
210
211
212 class User(Base):
213     """Represents a user of the application"""
214     __tablename__ = "users"
```

```
215     uid = Column(Integer, primary_key=True)
216     e_mail = Column(String, unique=True)
217     password = Column(LargeBinary)
218     salt = Column(BigInt)
219     name = Column(String)
220     surname = Column(String)
221     is_admin = Column(Boolean)
222     location_uid = Column(Integer, sqlalchemy.ForeignKey("locations.\
            uid"))
223     location = orm.relationship("Location", backref=orm.backref("users\
            ", uselist=False))
224     responsibilities = orm.relationship("Responsibility", backref="\
            user")
225     phonenumber = orm.relationship("PhoneNumber", backref="user", \
            uselist=False)
226     def __init__(self, e_mail, password, name="", surname="", \
            phonenumber="", uid=None, salt=None):
227         self.uid = uid
228         self.e_mail = e_mail.lower()
229         self.salt = salt if salt else self.new_salt()
230         self.name = name.title()
231         self.surname = surname.title()
232         if isinstance(phonenumber, PhoneNumber):
233             self.phonenumber = phonenumber
234         else:
235             self.phonenumber = PhoneNumber(phonenumber)
236         self.hash(password)
237         self.is_admin = False
238 
239     @staticmethod
240     def new_salt():
241         return randbits(256)
242 
243     def hash(self, password):
244         """Hash a string with pbkdf2
245         The salt is XORd with the e_mail to get the final salt
246         """
247         salt = str(int.from_bytes(self.e_mail.encode(), byteorder="big\
                ") ^ self.salt).encode()
248         self.password = hashlib.pbkdf2_hmac(
```

```
249             hash_name="sha512",
250             password=password.encode(),
251             salt=salt,
252             iterations=9600)
253
254     def __str__(self):
255         return f"{self.name} {self.surname}"
256
257
258 class Responsibility(Base):
259     """Represents a responsibility a User has for a Device"""
260     __tablename__ = "responsibilities"
261     device_uid = Column(Integer, sqlalchemy.ForeignKey("devices.uid"),\
            primary_key=True)
262     user_uid = Column(Integer, sqlalchemy.ForeignKey("users.uid"), \
            primary_key=True)
263     location_uid = Column(Integer, sqlalchemy.ForeignKey("locations.\
            uid"), primary_key=True)
264
265     def __init__(self, device=None, user=None, location=None):
266         self.device = device
267         self.user = user
268         self.location = location
269
270     def __str__(self):
271         return f"{self.user} is responsible for device {self.device} \
                at {self.location}"
272
273
274 Base.metadata.create_all(bind=engine) # Database initialized
275 logger.info("Database initialized, tables verified")
276
277
278 class Timeout(Thread):
279     """Timer that runs in background and executes a function if it's \
            not refreshed
280     Important: This is different from the threading.Timer class in \
            that it can provide
281     arguments to a function as well as allows reseting the timer, \
            rather than canceling
```

```
282     completely. to cancel a Timeout set function to None, the thread \
            will then close itself
283     down on the next lifecycle check.
284
285         Args:
286             function: function that is executed once time runs out
287             timeout: time in seconds after which the timeout executes \
                    the function
288             args: arguments for function
289
290         Attributes:
291             timeout: time in seconds afters which the timer times out
292             function: function to be executed once time runs out
293             args: arguments for function
294             lock: mutex for various attributes
295             timed_out: boolean presenting wether the timer timed out
296             last_interaction_timestamp: unix timestamp of last refresh\
                    /initialization
297
298         Properties:
299             timer: Shows the time remaining until timeout
300     """
301
302     def __init__(self, timeout, function, args=None):
303         super().__init__()
304         if args is None:
305             args = []
306         self.timeout = timeout
307         self.function = function
308         self.args = args
309         self.lock = Lock()
310         self.timed_out = False
311         self.reset()
312
313     def _refresh_timer(self):
314         with self.lock:
315             return self.timeout - (time() - self.\
                        last_interaction_timestamp)
316
317     timer = property(fget=_refresh_timer)
```

```
318
319     def reset(self):
320         """Reset the internal timer"""
321         with self.lock:
322             if not self.timed_out:
323                 self.is_reset = True
324                 self.last_interaction_timestamp = time()
325
326     def run(self):
327         """Start the timer"""
328         if self.function is None:
329             logger.debug("Timeout␣thread␣closed␣because␣function␣was␣\
                    None")
330             return
331         with self.lock:
332             if self.is_reset:
333                 self.is_reset = False
334             else:
335                 self.timed_out = True
336                 self.function(*self.args)
337                 return
338             difference_to_timeout = self.timeout - (time() - self.\
                    last_interaction_timestamp)
339         sleep(difference_to_timeout)
340         self.run()
341
342
343 class VideoStreamUISync(Thread):
344     """Class to tie a LazyVideoStream to some canvas in Qt
345         Args:
346             canvas: canvas has to be able to take pixmaps/implement \
                    setPixmap
347             videostream: Instance of LazyVideoStream that supplies the\
                     frames
348             barcodes: Counter that holds all found barcodes USE \
                    barcode_lock WHEN ACCESSING!
349             barcode_lock: Lock for barcodes
350     """
351     def __init__(self, canvas, videostream):
352         super().__init__()
```

```
353         self.canvas = canvas
354         self.videostream = videostream
355         self.daemon = True
356         self.barcodes = Counter()
357         self.barcode_lock = Lock()
358 
359     @staticmethod
360     def _matrice_to_QPixmap(frame):
361         """Convert cv2/numpy matrice to a Qt QPixmap"""
362         height, width, channel = frame.shape
363         image = QImage(frame.data, width, height, 3 * width, QImage.\
                Format_RGB888)
364         return QPixmap(image)
365 
366     def get_most_common(self):
367         """Get barcode with highest occurence"""
368         with self.barcode_lock:
369             return self.barcodes.most_common(1)[0]
370 
371     def reset_counter(self):
372         with self.barcode_lock:
373             self.barcodes = Counter()
374 
375     def run(self):
376         """Start synchronization"""
377         while True:
378             self.videostream.request_queue.put(True)
379             frame, found_codes = self.videostream.frame_queue.get()
380             pixmap = self._matrice_to_QPixmap(frame)
381             self.canvas.setPixmap(pixmap)
382             if found_codes:
383                 self.barcodes.update(found_codes)
384             cv2.waitKey(1)
385 
386 
387 if __name__ == "__main__":
388     """Tests"""
389     """Timer Test
390     import threading
391     timeout = Timeout(timeout=10, function=print, args=["timed out"])
```

```
392     timeout.start()
393     reset_thread = threading.Thread(target=lambda: timeout.reset() if \
            input() else None) # function for testing - reset on input
394     reset_thread.start()
395     t = 0
396     delta_t = 1
397     t1 = time()
398     while not timeout.timed_out:
399         print(f"Not timed out yet - {t:.2f} seconds passed")
400         print(timeout.timer)
401         t += delta_t
402         sleep(delta_t)
403     print(time() - t1)
404     timeout.join()
405     reset_thread.join()
406     """
407
408     location1 = Location("GebĂude␣23")
409     location2 = Location("BĂźro")
410     location3 = Location("Lager")
411     user1 = User(e_mail="Karl@googlemail.com", password="123", name="\
            Karl", surname="KĂśnig", phonenumber="123456")
412     user2 = User(e_mail="hey@ho.com", password="passwort", name="Bob",\
             surname="KĂźnig", phonenumber="654321")
413     user3 = User(e_mail="Mail@mail.com", password="456", name="Bob", \
            surname="KĂźnig", phonenumber="21")
414     user4 = User(e_mail="Testo@web.de", password="456", name="testo", \
            surname="Testington", phonenumber="621")
415     user5 = User(e_mail="Jack@web.de", password="1234", name="Jack", \
            surname="von␣Teststadt", phonenumber="+49045␣1123")
416     user6 = User(e_mail="mymail@gmail.com", password="abc", name="\
            Billy", surname="Bob", phonenumber="651")
417     user1.location = location2
418     user2.location = location2
419     user3.location = location2
420     user4.location = location2
421     user5.location = location2
422     user6.location = location3
423
424     producer1 = Producer("Padcon")
```

```
425     producer2 = Producer("Intel")
426     article1 = Article("Prusa␣Mk3")
427     article2 = Article("Kopierpapier")
428     article3 = Article("Laptop␣123")
429     article4 = Article("i7␣4790k")
430     article1.producer = producer1
431     article2.producer = producer1
432     article3.producer = producer1
433     article4.producer = producer2
434
435     device1 = Device("1")
436     device1.article = article1
437     device1.location = location1
438
439     device2 = Device("2")
440     article1.devices.append(device2)
441     device2.location = location2
442
443     device3 = Device("3")
444     device3.article = article1
445     device3.location = location3
446
447     device4 = Device("4")
448     device4.article = article4
449     device4.location = location3
450
451     device5 = Device("5")
452     device5.article = article4
453     device5.location = location3
454
455     device6 = Device("6")
456     device6.article = article3
457     device6.location = location1
458
459     device7 = Device("7")
460     device7.article = article2
461     device7.location = location1
462
463     device8 = Device("8")
464     device8.article = article1
```

```
465     device8.location = location2
466
467     resp1 = Responsibility()
468     resp1.user = user1
469     resp1.location = location2
470     resp1.device = device1
471
472     resp2 = Responsibility()
473     resp2.user = user2
474     resp2.location = location3
475     resp2.device = device2
476
477     resp3 = Responsibility()
478     resp3.user = user5
479     resp3.location = location1
480     resp3.device = device3
481
482     resp4 = Responsibility()
483     resp4.user = user5
484     resp4.location = location1
485     resp4.device = device4
486
487     resp5 = Responsibility()
488     resp5.user = user3
489     resp5.location = location2
490     resp5.device = device5
491
492     resp6 = Responsibility()
493     resp6.user = user3
494     resp6.location = location1
495     resp6.device = device6
496
497     resp7 = Responsibility()
498     resp7.user = user1
499     resp7.location = location1
500     resp7.device = device7
501
502     resp8 = Responsibility()
503     resp8.user = user1
504     resp8.location = location1
```

```
    resp8.device = device8

    Session = orm.sessionmaker(bind=engine)
    session = Session()

    session.add(location1)
    session.add(location2)
    session.add(location3)
    session.add_all([user1, user2, user3, user4, user5, user6])
    session.add_all([producer1, producer2])
    session.add_all([article1, article2, article3, article4])
    session.add_all([device1, device2, device3, device4, device5, \
        device6, device7, device8])
    session.add_all([resp1, resp2, resp3, resp4, resp5, resp6, resp7, \
        resp8])

    print("-"*30)
    print(f"{article1.producer.name} produces the following articles")
    for art in producer1.articles:
        print(f"{' '*5}{art.name}")
        print(f"{' '*10}instances of these articles are:")
        for device in art.devices:
            print(f"{device.code:>20} stored at {device.location.name\
                :<20}")
    print("-"*30)

    del user1
    del user2
    del location1
    del location2

    try:
        print(user1.name)
    except Exception:
        print("There's no user1, this is wanted and good")
    print("-"*30)

    users = session.query(User).all()
    locations = session.query()

```

```
542     for user in users:
543         print(f"{user.uid:>5}:␣␣␣{user.e_mail:>40}␣|␣{user.phonenumber\
                :<20}␣|␣{user.location.name}")
544     print("-"*30)
545     print("Known␣responsibilities␣for␣these␣users␣are:")
546     for user in users:
547         for resp in user.responsibilities:
548             print(f"{resp.user.name:^15}|{resp.device.code:^15}|{resp.\
                    location.name:^15}")
549             print(resp.device.article.name)
550
551     session.commit()
552     session.close()
```

A.3 keys.py

```
1  """Generation and reading of RSA keys for asymmetric encryption"""
2
3  try:
4      from Cryptodome.Cipher import PKCS1_OAEP
5      from Cryptodome.PublicKey import RSA
6  except ImportError as e:
7      try:
8          from Crypto.Cipher import PKCS1_OAEP
9          from Crypto.PublicKey import RSA
10     except ImportError as err:
11         raise err
12 from pathlib import Path
13
14 def generate_key(path_public, path_private):
15     """Generate new RSA key-pair and save it to the given paths
16
17         Args:
18             path_public: Path where the public-key should be stored
19             path_private: Path where th private-key should be stored
20     """
21     key = RSA.generate(4096)
22     with open(path_public, "wb") as f:
23         f.write(key.publickey().exportKey())
24     with open(path_private, "wb") as f:
```

```
25          f.write(key.exportKey())
26
27
28  def read_keys(path_public, path_private):
29      """Read a key-pair from the given paths and build ciphers from it
30
31          Args:
32              path_public: Path where the public-key is stored
33              path_private: Path where the private-key is stored
34
35          Returns:
36              (PKCS1_OAEP-cipher from public-key, PKCS1_OAEP-decipher \
                    from private-key)
37      """
38      path_public = Path(path_public)
39      path_private = Path(path_private)
40      with open(path_public, "rb") as f:
41          publickey = RSA.importKey(f.read())
42      with open(path_private, "rb") as f:
43          privatekey = RSA.importKey(f.read())
44      cipher = PKCS1_OAEP.new(publickey)
45      decipher = PKCS1_OAEP.new(privatekey)
46      return cipher, decipher
```

A.4 logger.py

```
1  """logging interface for consistent formatting"""
2
3  import logging
4
5  from utils import absolute_path
6
7  handler = logging.FileHandler(filename=absolute_path("protocol.log"))
8  formatter = logging.Formatter(
9      fmt='{asctime} [{levelname:8}] from {module:>15}.{funcName:12} "{\
           message}"',
10     style="{",
11     datefmt="%Y-%m-%dT%H:%m:%S")
12
13 handler.setFormatter(formatter)
```

```
14 logger = logging.getLogger()
15 logger.addHandler(handler)
16 logger.setLevel(logging.INFO)
17
18 del handler
19 del formatter
```

A.5 main.py

```
1  """main module that ties everything together"""
2
3  import logging
4  import sys
5  from threading import Lock, Thread
6  from time import sleep
7
8  from PyQt5 import QtWidgets
9
10 from barcodereader import LazyVideoStream, VideoStream
11 from classes import VideoStreamUISync
12 from logger import logger
13 import ui
14 from utils import absolute_path
15
16
17 def main():
18     app = QtWidgets.QApplication(sys.argv)
19     dialog_main = ui.MainDialog()
20
21     try:
22         videostream = LazyVideoStream()
23         videostream.start()
24         logger.info(f"Camera␣{videostream.camera_id}␣succesfully␣\
               opened")
25         video_ui_sync = VideoStreamUISync(dialog_main.ui.videoFeed, \
               videostream)
26         video_ui_sync.start()
27         logger.info("Connected␣Camera␣to␣UI")
28     except IOError as e:
29         logger.error(str(e))
```

```
30
31      dialog_main.show()
32
33      sys.exit(app.exec_())
34
35
36  if __name__ == "__main__":
37      # Profiling via terminal
38      import cProfile
39      cProfile.run("main()", sort="time")
40
41      #main()
```

A.6 pydoc.sh

```
1  #!/bin/bash
2
3  pydoc -w barcodereader
4  pydoc -w classes
5  pydoc -w keys
6  pydoc -w logger
7  pydoc -w main
8  pydoc -w qr_generator
9  pydoc -w slots
10 pydoc -w ui
11 pydoc -w utils
```

A.7 qr_generator.py

```
1  """Handles QR-Code generation"""
2
3  import qrcode
4  import qrcode.image.svg as svg
5
6  def generate_qr(device):
7      """Generate the QR-Code for a given device and store it locally as\
           svg"""
8      qr = qrcode.QRCode(version=None,
9          error_correction=qrcode.constants.ERROR_CORRECT_H,
```

```
10          box_size=20,
11          border=6)
12      code = f"id={device.uid} name={device.article.name}"
13      qr.add_data(code)
14      qr.make(fit=True) # autosize according to data
15      img = qr.make_image(image_factory=svg.SvgPathFillImage)
16      img.save(f"{device.uid}_{device.article.name}.svg")
17
18  if __name__=="__main__":
19      class TestArticle():
20          def __init__(self, name):
21              self.name = name
22      class TestDevice():
23          def __init__(self, uid, code, article):
24              self.uid = uid
25              self.code = code
26              self.article = article
27      article = TestArticle("Article")
28      device = TestDevice(10, "", article)
29      generate_qr(device)
```

A.8 slots.py

```
1  from functools import partial
2  from secrets import choice, compare_digest
3  from string import ascii_letters, digits
4
5  import sqlalchemy
6
7  import classes
8  import keys
9  from logger import logger
10
11
12 CSession = classes.setup_context_session(classes.engine)
13
14
15 def synchronized(function, decorated=False, *args, **kwargs):
16     """Function-decorator to automatically add the instance a function\
           returns to DB"""
```

```
17      if not decorated:
18          return partial(synchronized, function, True)
19      else:
20          instance = function(*args, **kwargs)
21          save_to_db(instance)
22          return instance
23
24
25  def save_to_db(instance):
26      """Save instance to it's corresponding table
27      Needs testing: May need to expunge instance after commit
28      ToDo: Error handling if uid already exists
29      """
30      with CSession() as session:
31          session.add(instance)
32
33
34  def update_user_dependant(user):
35      pass
36
37
38  @synchronized
39  def create_user(e_mail, password, name, surname, location, phonenumber\
        ):
40      new_user = classes.User(e_mail, password, name, surname, \
            phonenumber)
41      new_user.location = location
42      return new_user
43
44
45  @synchronized
46  def create_admin(new_admin):
47      """Create a new admin"""
48      new_admin.is_admin = True
49      return new_admin
50
51
52  def login(e_mail, password):
53      """Log user into application
54      Checks if there's a user of given name in the database,
```

```
55      if the given password is correct and returns the user if both is \
            the case
56      """
57      e_mail = e_mail.lower()
58      with CSession() as session:
59          try:
60              user = session.query(classes.User).filter_by(e_mail=e_mail\
                    ).first()
61              user_at_gate = classes.User(e_mail, password, salt=user.\
                    salt)
62              if compare_digest(user_at_gate.password, user.password):
63                  update_user_dependant(user)
64                  session.expunge(user)
65                  logger.info(f"Successfully␣logged␣in␣as␣{user.uid}")
66                  return user
67              else:
68                  logger.info(f"Attempted␣login␣with␣wrong␣password␣for␣\
                        user␣{e_mail}")
69                  return None
70          except (AttributeError, ValueError) as e: #user not found \
                exception
71              logger.info(f"Attempted␣login␣from␣unknown␣user␣{e_mail}")
72              pass # show error message
73
74
75  def logout():
76      """Log user out of application"""
77      pass
78
79
80  @synchronized
81  def create_device(article):
82      new_device = classes.Device()
83      new_device.article = article
84      save_to_db(new_device)
85      return new_device
86
87
88  @synchronized
89  def create_location():
```

```
    pass


@synchronized
def create_producer():
    pass


def generate_password(len_=15):
    """Generate an human readable password of given length"""
    alphabet = ascii_letters
    without = list("0OIl")
    pw_chars = alphabet + digits + "!,;.-_+-*()[]{}$%=?â Ź#'˜Ă Â§&"
    pw_chars = "".join((letter for letter in pw_chars if letter not in\
         without))
    pw = "".join((choice(pw_chars) for i in range(len_)))
    return pw


def reset_password(user):
    user.salt = user.new_salt()
    password = generate_password()
    user.hash(password)
    return password


def reset_admin_password(user, path_public, path_private):
    #catch error if invalid key or no public key
    cipher, decipher = keys.read_keys(path_public, path_private)
    text = b"True"
    ciphertext = cipher.encrypt(text)
    if compare_digest(decipher.decrypt(ciphertext), text):
        keys.generate_key(path_public, path_private)
        new_password = reset_password(user)
        return new_password
    else:
        return None


if __name__ == "__main__":
```

```
129     user = login("schokoladenkÄśnig@googlemail.com", "12345\
            ibimsdaspasswort")
130     print(user.e_mail)
```

A.9 ui.py

```
1  """PyQt5 UI classes and linking to slots"""
2
3  import sys
4
5  from PyQt5 import uic, QtWidgets
6  from PyQt5.QtGui import QColor, QIcon, QPainter, QPen
7  from PyQt5.QtCore import Qt
8
9  import classes
10 from logger import logger
11 import slots
12 from utils import absolute_path, parallel_print
13
14 CSession = classes.setup_context_session(classes.engine)
15
16
17 class MainDialog(QtWidgets.QDialog):
18
19     def __init__(self, parent=None):
20         path = absolute_path("main_horizontal.ui")
21         super().__init__(parent)
22         self.ui = uic.loadUi(path, self)
23         self.logged_in_user = None
24         self.set_tree()
25
26         self.ui.b_user_login.clicked.connect(self.b_user_login_click)
27         self.ui.b_user_logout.clicked.connect(self.b_user_logout_click\
               )
28         self.ui.b_home_1.clicked.connect(self.b_home_1_click)
29
30         # tabs_click
31         self.ui.b_tab_1.clicked.connect(self.b_tab_1_click)
32         self.ui.b_tab_2.clicked.connect(self.b_tab_2_click)
33         self.ui.b_tab_3.clicked.connect(self.b_tab_3_click)
```

```
34          self.ui.b_tab_4.clicked.connect(self.b_tab_4_click)
35          self.ui.b_tab_5.clicked.connect(self.b_tab_5_click)
36          self.ui.b_tui_bottom.clicked.connect(self.b_tui_bottom_click)
37
38      ## set Colors
39          self.setAutoFillBackground(True)                    # \
                background / white
40          p = self.palette()
41          p.setColor(self.backgroundRole(), Qt.white)
42          self.setPalette(p)
43
44
45          palette1 = self.line_1.palette()                    # tab 1 / \
                blue
46          role1 = self.line_1.backgroundRole()
47          palette1.setColor(role1, QColor('blue'))
48          self.ui.line_1.setPalette(palette1)
49          self.ui.line_1.setAutoFillBackground(True)
50
51          palette2 = self.line_2.palette()                    # tab 2 / \
                blue
52          role2 = self.line_2.backgroundRole()
53          palette2.setColor(role2, QColor('blue'))
54          self.ui.line_2.setPalette(palette2)
55          self.ui.line_2.setAutoFillBackground(True)
56
57          palette3= self.line_3.palette()                     # tab 3 / \
                blue
58          role3 = self.line_3.backgroundRole()
59          palette3.setColor(role3, QColor('blue'))
60          self.ui.line_3.setPalette(palette3)
61          self.ui.line_3.setAutoFillBackground(True)
62
63          palette4 = self.line_4.palette()                    # tab 4 / \
                blue
64          role4 = self.line_4.backgroundRole()
65          palette4.setColor(role4, QColor('blue'))
66          self.ui.line_4.setPalette(palette4)
67          self.ui.line_4.setAutoFillBackground(True)
68
```

```
69          palette5 = self.line_5.palette()                    # tab 5 / \
                blue
70          role5 = self.line_5.backgroundRole()
71          palette5.setColor(role5, QColor('blue'))
72          self.ui.line_5.setPalette(palette5)
73          self.ui.line_5.setAutoFillBackground(True)
74
75          palette6 = self.line_6.palette()                    # \
                line_bottom / blue
76          role6 = self.line_6.backgroundRole()
77          palette6.setColor(role6, QColor('blue'))
78          self.ui.line_6.setPalette(palette6)
79          self.ui.line_6.setAutoFillBackground(True)
80
81          #palette7 = self.line_7.palette()                    # line_top\
                / blue
82          #role7 = self.line_7.backgroundRole()
83          #palette7.setColor(role7, QColor('blue'))
84          #self.ui.line_7.setPalette(palette7)
85          #self.ui.line_7.setAutoFillBackground(True)
86
87          palette8 = self.b_tui_bottom.palette()              # \
                button_bottom / blue
88          role8 = self.b_tui_bottom.backgroundRole()
89          palette8.setColor(role8, QColor('blue'))
90          self.ui.b_tui_bottom.setPalette(palette8)
91          self.ui.b_tui_bottom.setAutoFillBackground(True)
92
93          #palette9 = self.b_home_1.palette()                  # ist noch\
                hÄsslich, muss Äźberarbeitet werden
94          #role9 = self.b_home_1.backgroundRole()
95          #palette9.setColor(role9, QColor('blue'))
96          #self.ui.b_home_1.setPalette(palette9)
97          #self.ui.b_home_1.setAutoFillBackground(True)
98          #self.b_home_1.setStyleSheet("color: white")
99      #/#
100
101
102     #*# show tabs
103         self.ui.stackedWidget.setCurrentIndex(0)
```

```
104         self.ui.line_1.show()
105         self.ui.line_2.hide()
106         self.ui.line_3.hide()
107         self.ui.line_4.hide()
108         self.ui.line_5.hide()
109 
110     def b_home_1_click(self):                       # home
111         self.ui.stackedWidget.setCurrentIndex(0)
112         self.ui.line_1.show()
113         self.ui.line_2.hide()
114         self.ui.line_3.hide()
115         self.ui.line_4.hide()
116         self.ui.line_5.hide()
117 
118     def b_tui_bottom_click(self):                       # home
119         self.ui.stackedWidget.setCurrentIndex(0)
120         self.ui.line_1.show()
121         self.ui.line_2.hide()
122         self.ui.line_3.hide()
123         self.ui.line_4.hide()
124         self.ui.line_5.hide()
125 
126     def b_tab_1_click(self):
127         self.ui.stackedWidget.setCurrentIndex(0)
128         self.ui.line_1.show()
129         self.ui.line_2.hide()
130         self.ui.line_3.hide()
131         self.ui.line_4.hide()
132         self.ui.line_5.hide()
133 
134     def b_tab_2_click(self):
135         self.ui.stackedWidget.setCurrentIndex(1)
136         self.ui.line_1.hide()
137         self.ui.line_2.show()
138         self.ui.line_3.hide()
139         self.ui.line_4.hide()
140         self.ui.line_5.hide()
141 
142     def b_tab_3_click(self):
143         self.ui.stackedWidget.setCurrentIndex(2)
```

```
144         self.ui.line_1.hide()
145         self.ui.line_2.hide()
146         self.ui.line_3.show()
147         self.ui.line_4.hide()
148         self.ui.line_5.hide()
149 
150     def b_tab_4_click(self):
151         self.ui.stackedWidget.setCurrentIndex(3)
152         self.ui.line_1.hide()
153         self.ui.line_2.hide()
154         self.ui.line_3.hide()
155         self.ui.line_4.show()
156         self.ui.line_5.hide()
157 
158     def b_tab_5_click(self):
159         self.ui.stackedWidget.setCurrentIndex(4)
160         self.ui.line_1.hide()
161         self.ui.line_2.hide()
162         self.ui.line_3.hide()
163         self.ui.line_4.hide()
164         self.ui.line_5.show()
165     #/#
166 
167     def set_tree(self): # toDo make tree scrollable if it gets too big
168         with CSession() as session:
169             responsibilities = session.query(classes.Responsibility).\
                    all()
170             for resp in responsibilities:
171                 location = QtWidgets.QTreeWidgetItem([str(resp.\
                        location.name)])
172                 user = QtWidgets.QTreeWidgetItem([str(resp.user)]) # \
                        toDo: Set color if user is the currently logged in\
                         one and add this to update_user_dependant
173                 device = QtWidgets.QTreeWidgetItem([str(resp.device)])
174                 locations = [self.treeWidget.topLevelItem(i) for i in \
                        range(self.treeWidget.topLevelItemCount())] # has \
                        to be list-comprehension because used multiple \
                        times -> docu
175                 tree_text = (item.text(0) for item in locations) # can\
                         be generator expression because single-use -> \
```

```
                    docu
176                 if str(resp.location.name) not in tree_text:
177                     location.addChild(user)
178                     user.addChild(device)
179                     self.treeWidget.addTopLevelItem(location)
180                 else:
181                     for location in locations:
182                         if location.text(0) == str(resp.location):
183                             users = [location.child(i) for i in range(\
                                    location.childCount())]
184                             if str(resp.user) not in (user.text(0) for\
                                     user in users):
185                                 location.addChild(user)
186                                 user.addChild(device)
187                             else:
188                                 for user in users:
189                                     devices = (user.child(i) for i in \
                                            range(location.childCount()))
190                                     if str(resp.device) not in devices\
                                            :
191                                         user.addChild(device)
192
193     def b_user_login_click(self):
194         LoginDialog(self).exec() # show dialog_login as modal dialog \
                => blocks controll of main
195         self.update_user_dependant()
196         if self.logged_in_user:
197             logger.info(f"Logged␣in␣as␣{self.logged_in_user}")
198             self.timeout = classes.Timeout(5, self.timed_out)
199             self.timeout.start()
200
201     def b_user_logout_click(self):
202         self.logged_in_user = None # may want slots.logout if that \
                does something eventually
203         self.update_user_dependant()
204         self.timeout.function = None
205         del self.timeout
206
207     def timed_out(self):
```

```
208         logger.info(f"User␣{self.logged_in_user.uid}␣logged␣out␣due␣to\
                ␣inactivity")
209         self.b_user_logout_click()
210         """Crashes on windows for some reason, known Qt issue
211         messagebox = QtWidgets.QMessageBox()
212         messagebox.setIcon(QtWidgets.QMessageBox.Information)
213         messagebox.setWindowTitle("Automatisch ausgeloggt")
214         messagebox.setText("Sie wurden wegen InaktivitÄt automatisch \
                ausgeloggt!")
215         messagebox.setStandardButtons(QtWidgets.QMessageBox.Ok)
216         messagebox.exec_()"""
217
218     def update_user_dependant(self):
219         if self.logged_in_user:
220             self.ui.log_in_out.setCurrentIndex(0)
221             self.label.setText(str(self.logged_in_user).title())
222             if self.logged_in_user.is_admin:
223                 self.checkBox.visible = True
224             else:
225                 self.checkBox.visible = False
226         else:
227             self.ui.log_in_out.setCurrentIndex(1)
228             self.label.setText("")
229
230     def mousePressEvent(self, QMouseEvent):
231         pass
232
233     def mouseMoveEvent(self, QMouseEvent):
234         if hasattr(self, "timeout"):
235             self.timeout.reset()
236
237     def new_user(self): # not linked yet
238         if "" in (box.text for box in [self.lineEdit_2, self.\
                lineEdit_3, self.lineEdit_4, self.lineEdit_5, self.\
                lineEdit_6]) or self.comboBox.currentText() == "": # \
                change to final names
239             # show message that user needs to fill all boxes
240             return
241
242         args = [None for i in range(6)]
```

```
243         user = slots.create_user(*args) # add textboxes once names are\
                final and remove args
244         if self.checkBox.isChecked():
245             slots.create_admin(user)
246         # show message that creation was successful
247 
248     def reset_password(self):
249         """Set new password and salt for user, push it to db and show \
                it in UI"""
250         with CSession() as session:
251             # check if admin is logged in(though this dialog shouldn't\
                    show if no admin is logged in)
252             # query user from db via name
253             user = None
254             new_password = slots.reset_password(user)
255         # display password
256 
257 
258 class LoginDialog(QtWidgets.QDialog):
259 
260     def __init__(self, parent=None):
261         path = absolute_path("login.ui")
262         super().__init__(parent)
263         self.parent = parent
264         self.ui = uic.loadUi(path, self)
265         self.b_login.clicked.connect(self.b_login_click)
266 
267     def b_login_click(self):
268         username = self.t_username.text()
269         password = self.t_password.text()
270         self.parent.logged_in_user = slots.login(username, password)
271         self.close()
272 
273 if __name__ == "__main__":
274     app = QtWidgets.QApplication(sys.argv)
275     dialog_main = MainDialog()
276     dialog_login = LoginDialog()
277     dialog_main.show() # show dialog_main as modeless dialog => return\
            control back immediately
278 
```

```
279     sys.exit(app.exec_())
```

A.10 utils.py

```
1  """General utilities independet of the other modules"""
2
3  import pathlib
4  import os
5  from queue import Queue
6  from threading import Thread
7
8  def absolute_path(relative_path):
9      """Convert a path relative to the sourcefile to an absolute one"""
10     path = pathlib.Path(os.path.dirname(__file__))
11     return path / relative_path
12
13 class _ParallelPrint(Thread):
14     """Provides a threadsafe print, instantiation below"""
15     print_ = Queue()
16     _created = False
17     def __init__(self):
18         if self._created:
19             raise ResourceWarning(f"{self.__class__} should only be \
                   instantiated once!")
20         super().__init__()
21         self.daemon = True
22         self.__class__._created = True
23
24     @classmethod
25     def run(this):
26         while True:
27             val = this.print_.get()
28             print(val)
29             this.print_.task_done()
30
31     @classmethod
32     def __call__(this):
33         raise ResourceWarning(f"{this.__class__} should only be \
               instantiated once!")
34
```

```
35
36 _ParallelPrint = _ParallelPrint()
37 _ParallelPrint.start()
38 parallel_print = _ParallelPrint.print_.put
```

B Automatisch generierte Dokumentation

Als .html auf beiliegendem USB-Stick, alternativ über pydoc.sh aus dem Quellcode zu generieren.