

TUI - Test- und Integrationssystem für SCADA-Software

von Stefan Volz & Yannis Köhler

6. April 2019



Teil I

Einführung

[&&]

Sämtliche Diagramme, Bilder, etc. sind, sofern nicht anders angegeben, selbst erstellt. Flussdiagramme wurden nach DIN 66001 erstellt¹, mit der Erweiterung, dass Zylinder als lokaler Speicher des Rechners und Datenbank eingesetzt werden.

Programmlistings wurden teils notwendigerweise automatisch vom Texteditor explizit 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. Des Weiteren soll eine Inventarsoftware erstellt werden, welche ein Einscannen von Barcodes ermöglicht und somit eine schnelle Zuordnung von Geräten zu Testplatz und zugehörigem Verantwortlichen zulässt.

¹[KB18, S. 114f]

Markierungsschema

Zur Markierung, welcher Projektpartner welche Teile verfasst hat, werden Tags als Randnotizen im Inhaltsverzeichnis und an Überschriften genutzt. Dabei gilt, dass alle einem getagten Abschnitt hierarchisch untergeordneten Abschnitte das übergeordnete Tag tragen, sofern sie nicht selbst eines besitzen. Der tatsächliche Verfasser ist über die Initialen im Tag codiert (Volle Namen im Inhaltsverzeichnis). Ein mit `&&` getagpter Abschnitt wurde kollaborativ vom ganzen Projektteam erstellt.

Inhaltsverzeichnis

I Einführung	1	[&&]
1 Die Firma Padcon	1	
2 Aufgabenstellung	1	
II Test- und Integrationsraum	12	[yk]
3 Raum	13	
3.1 Allgemein	13	
3.2 Ursprüngliche Anforderungen	14	
3.3 Planung	15	
3.3.1 Iteration 1	16	
3.3.2 Iteration 2	18	
3.3.3 Iteration 3	20	
3.3.4 Iteration 4	22	
4 Elektrische Umsetzung	24	
4.1 Austattung der einzelnen Bereiche	24	
4.1.1 Testplatz 1-4	24	
4.1.2 Testplatz PCB	24	
4.1.3 Der Serverschrank	24	
4.1.4 Das Serverrack	24	
4.1.5 Die Unterverteilung	24	
4.2 Die Unterverteilung	25	
4.2.1 Zuleitung	25	
4.2.2 Niederspannung	26	
4.2.3 Kleinspannung	27	
4.2.4 Wärmeberechnung	28	
4.2.5 Klemmen	30	
4.2.6 Montage	31	
4.3 Das Netzwerk	31	

III TUIventory	32 [sv]
5 Begriffsdefinitionen	33
6 Einführung und Problemanalyse	35
6.1 Codierungssystem	35
6.2 Desktopanwendung und Kommandozeilen-Schnittstelle	36
7 Python 3	38
7.1 Funktionen und Methoden in Python	38
7.2 Formatierung	39
7.3 Vorwort und Übersicht über die Anwendung	40
8 Command Line Interface	42
9 SQLAlchemy	44
9.1 Das Datenbankmodell	44
9.2 Implementierung des Datenbankmodells	45
9.3 Benutzerdefinierte Datentypen	46
9.4 Beziehungen herstellen in SQLAlchemy	47
9.4.1 Grundprinzip und 1-zu-n-Beziehungen	47
9.4.2 1-zu-1-Beziehung	48
9.5 Arbeiten mit SQLAlchemy	48
9.6 <i>ContextSessions</i>	49
9.6.1 Neue Einträge in der Datenbank vornehmen	50
9.6.2 Abfragen	50
9.6.3 Aktualisieren und Löschen einer Instanz	52
10 Frameworkfreie Implementierung	53
10.1 Utilities	53
10.1.1 Absolute Pfade	53
10.1.2 Paralleles Printen	53
10.1.3 Dateitools - Pfadvalidierung und Normalisierung von Dateinamen	56
10.2 Logging	56
10.3 Barcode Reader - die Klassen VideoStream und LazyVideoStream	57
10.4 Auslesen von Barcodes und grundlegende Aufbereitung von Frames	60
10.5 Weitere Kapselung: VideostreamUISync	63
10.6 Datenbanksynchronisation	65

11 Benutzerverwaltung	67
11.1 SQL	67
11.2 Kryptografie	68
11.2.1 Benutzer anlegen	69
11.2.2 Benutzer anmelden	72
11.2.3 Benutzer hat sein Passwort vergessen	74
11.2.4 Mathematische Betrachtung der Passwort-Generation	76
11.2.5 Admin hat sein Passwort vergessen	80
11.3 Automatisches Abmelden	82
11.4 Die Klasse <i>TelephoneNumber</i>	84
12 PyQt5	90 [yk]
12.1 Signale und Slots	90
12.1.1 Signal	90
12.1.2 Slot	90
12.1.3 Signale und Slots	90
12.2 Darstellen der Verantwortlichkeiten als Baumstruktur	91 [sv]
12.3 Fazit	94 [sv]
13 Aufbau der UI	94
13.1 Login-Menü	95
13.2 Menüpunkte	95
13.2.1 Übersicht:	95
13.2.2 Kamera:	95
13.2.3 Geräteverwaltung:	96
13.2.4 Benutzerverwaltung:	96
13.2.5 Einstellungen:	97
13.3 Statusbar	98
14 Optimierung und Testing	98
14.1 mouseMoveEvent	98
14.2 ContextSession Cache Refresh	104
15 Schutz vor unauthorisiertem Zugriff durch <i>setuid</i>	107 [yk]
	TODO
IV Anhang	1

A Pläne	2
A.1 Stromlaufplan	2
A.2 Netzwerkplan	12
A.3 Netzwerkplan nach EN81346	16
B Quellcode	20
B.1 cli_get_barcode.py	20
B.2 barcodereader.py	21
B.3 classes.py	29
B.4 keys.py	46
B.5 logger.py	47
B.6 main.py	48
B.7 pydoc.sh	50
B.8 qr_generator.py	50
B.9 slots.py	52
B.10 ui.py	56
B.11 unittests.py	81
B.12 utils.py	83
B.13 main.rs	85
B.14 cargo.toml	85
C Automatisch generierte Dokumentation	86
D Literatur- und Quellenverzeichnis	87
E Eidesstattliche Erklärung	89

Abbildungsverzeichnis

3.1 Iteration 1 Raum	17
3.2 Iteration 2 Raum	19
3.3 Iteration 3 Raum	21
3.4 Iteration 4 Raum	23
7.1 Übersicht über Verhältnisse von Threads	40
9.1 Entity-Relationship-Diagramm	45
9.2 UML-Diagramm	46
10.1 Log-Datei Beispiel	57
10.2 Gegenüberstellung von Polling und Queues	58
11.1 Beispieldatenbank	68
11.2 Prinzip von Hashfunktionen	69
11.3 Passwort-Speicher-Vorgang	70
11.4 Anmelde-Vorgang	73
11.5 Rücksetzen eines Benutzerpassworts	75
11.6 Vergleich von f_1 und f_2	77
11.7 Heatmap aus f_1 und f_2	78
11.8 $k(n)$ aus f_3	80
11.9 Verifizierung zum Rücksetzen eines Admin-Passworts	81
12.1 Struktogramm nach Nassi-Shneiderman zum Responsibility-Baum-Sieb	92
14.1 Vergleich von dict lookup und hasattr (jeweiliger Durchschnitt der Versuchsreihe in Legende)	100
14.2 Visualisierung einer Testreihe	107

Algorithmenverzeichnis

1	Pythons Funktionsmodell	39
2	Schnittstellenhilfe zu cli_get_barcode.py	43
3	Beispiel einer 1-zu-n-Beziehung in SQLAlchemy	47
4	Beispiel einer 1-zu-1-Beziehung in SQLAlchemy	48
5	Objekte in Datenbank ablegen	50
6	Abfrage aus Datenbank anhand eines tatsächlichen Beispiels	51
7	Paralleles Printen mittels Metaprogrammierung	55
8	Methode <i>find_and_mark_barcodes</i> der beiden VideoStream-Klassen	61
9	Definition des Function-Decorators	65
10	Verbesserte Definition des Function-Decorators	67
11	Beispiel von SQL-Injection	68
12	Kompilierung und Debugausgabe der Regular Expression zur Telefonnummernerkennung	87
13	Gegenüberstellung Generator Expression und List Comprehension	92
14	Ursprünglicher Timeout reset	99
15	Alternativer Timeout reset	99
16	Bytecode zu <i>dict-lookup</i> und <i>hasattr</i> erzeugen	101
17	Bytecode zu beiden Varianten	102
18	Analysecode zum Laufzeitvergleich verschiedener Iterationsmöglichkeiten	105
19	Ausschnitt der Konsolenausgabe	106

Abkürzungsverzeichnis

Abk.	Beschreibung
AMQ	Atomic Message Queue
API	Application Programming Interface
CLI	Command Line Interface
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
RCBO	Residual current operated Circuit-Breaker with Overcurrent protection
SCADA	Supervisory Control and Data Acquisition
SHA	Secure Hash Algorithm
SV	Stefan Volz
UI	User Interface
VM	Virtual Machine / Virtuelle Maschine
YK	Yannis Köhler

Begriffsdefinitionen

Begriff	Definition
Duck-Typing	Der Typ einer Instanz wird dadurch beschrieben/festgelegt, welche Member sie besitzt (Von engl.: "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 ² .
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 wettkämpfen, um die Ausgabe als erstes zu beeinflussen ³ . Probleme einer Race-Condition sind beispielsweise, dass sie oftmals verschwindet, wenn das Programm mittels Debugger betrachtet wird.

²[EK17, S. 369]

³[Wik18a, 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 ⁴ .
Singleton	Ein Singleton ist ein Entwurfsmuster, bei dem sichergestellt ist, dass nur eine einzige Instanz einer Klasse existiert (z.B. in Python <code>None</code>).
Thread	(dt.: Faden, Strang) Threads sind im Grunde genommen leichtgewichtige Prozesse (Ein Prozess kann mehrere Threads besitzen). In CPython (erläutert in Abschnitt 7) 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.

⁴[Wik19b, 06.03.19 - 12:07 Uhr]

Teil II

Test- und Integrationsraum

[yk]

3 Raum

3.1 Allgemein

In diesem sogenannten Test- und Integrationsraum sollen neue, zugekaufte Hardwarekomponenten getestet werden können, bevor sie auf größeren Anlagen verbaut werden. Bei diesen Tests wird vor allem auf Ausfallsicherheit und Steadiness geachtet.

Als Beispiel wurden, als wir mit dem Projekt angefangen hatten, mehrere IPC's auf längere Zeit einfach laufen gelassen. Von der Performance waren alle sehr gut, jedoch ist einer nach mehreren Tagen immer abgestürzt. So etwas lässt sich aus den theoretischen Daten des Produktes nicht herauslesen und darf auch auf keinen Fall auf einer schon in Betrieb genommenen Anlage passieren. Da hier in der Regel von großen Solaranlagen die Rede ist könnte ein Ausfall einer solchen in manchen Ländern das ganze Stromnetz ins Wanken oder zum Zusammenbrechen bringen.

Ebenfalls soll dann auf bereits auch auf den Anlagen eingesetzter Hardware neue Softwareversionen der firmeneigenen Überwachungssoftware getestet werden können. So wird eine möglichst Realitätsnahe Umgebung geschaffen um einen späteren Fehlerfall zu 99% ausschließen zu können.

Bei der Konzeptionierung der Testplätze war vor allem eine einheitliche Umgebung und ein modularer Aufbau wichtig. Es sollten für alle Geräte die gleichen Grundbedingungen geschaffen werden, um diese möglichst direkt miteinander vergleichen zu können. Auch der modulare Aufbau stand an hoher Priorität, da Geräte aller Bauformen und Größen getestet werden sollen. Auf einen Potentialausgleich wurde - trotz ausdrücklicher Empfehlung unsererseits - verzichtet, um diese Modularität beizubehalten.

Im Laufe der Planung stellte sich heraus, dass unser Raum für den neuen Server der Firma, einen besonders geeigneten Platz darstellte, also musste für diesen auch noch eine getrennte Spannungsversorgung mit eingeplant werden.

Da die bereits vorhandene Stromversorgung in dem Raum für unsere Zwecke nicht ausreichend und ebenfalls stark veraltet war, musste eine neue Zuleitung aus dem Hauptverteiler, sowie eine neue Unterverteilung eingeplant werden. Für das Netzwerk, dass entstehen soll, wird auch ein Serverrack benötigt. Die Versorgung für den neuen Server sollte ebenfalls über die neue Unterverteilung laufen.

3.2 Ursprüngliche Anforderungen

Die vom Auftraggeber vorgegebenen Anforderungen an den Test- und Integrationsraum. Diese haben sich jedoch, wie wohl bei jedem Projekt, im Verlauf geändert. Die finale Ausführung finden Sie unter 4.1.

- drei bis vier Testplätze, je nach Platzangebot:
 - 230V Netzspannungsversorgung
 - 24V, 12V, 5V DC
 - Netzwerk
 - Modularer Aufbau, um Geräte aller Bauarten dort zu befestigen
- Officeplatz - für z.B. kurze Recherche im Internet o.Ä.
 - 230V Netzspannungsversorgung
 - Netzwerk
- Neue Unterverteilung, im Testraum
 - 400V Zuleitung
 - Netzwerkanschluss
- Serverrack für das Netzwerk - Patchfeld und Switch
 - 230V Netzspannungsversorgung, überwacht

3.3 Planung

Bevor die Planung aufgenommen werden konnte, musste ein geeigneter Raum gefunden werden. Hier entschieden wir uns, in Absprache mit der Firma, für einen leer stehenden Abstellraum in einem Anbau am Entwicklungs- und Verpackungszentrum. Durch die optimale Lage an der Nordseite wurde es im Sommer nicht zu warm und es war, trotz eines Fensters, keine direkte Sonneneinstrahlung vorhanden, welche zu ungleichmäßigen Tests führen könnte.

Trotz alledem wurde eine kleinere Standklimaanlage mit eingeplant, um die Temperatur konstant zu halten. Dies wurde vor allem auch in Anbetracht des erweiterbaren Servers getan, um hier in Zukunft nicht an Temperaturgrenzen zu stoßen.

Um einen besseren Überblick über den verfügbaren Platz zu gewinnen wurde zunächst ein 3D-Modell des Raumes in Autodesk Fusion 360 erstellt. Hierzu wurde der Raum vollständig ausgemessen, inklusive der Türen, des Fensters, den Heizungsleitungen und allen anderen möglichen Hindernissen im Verlaufe der Planung. Hier konnte die ganze Planung, zum Beispiel von den einzelnen Testplätzen, oder der Kleinspannungsversorgung, auch visuell durchgeführt werden.

Mit Hilfe dieses 3D-Modells wurden bereits im Laufe der Planung mehrere Render erstellt. Diese dienten des Öfteren auch als Grundlage bei Argumentationen über die Finanzierung einzelner Komponenten.

3.3.1 Iteration 1

Dies stellt den ersten Entwurf des Test- und Integrationsraumes dar.

- (a) In der Abbildung sind an der rechten Seite (1) bereits drei Testplätze zu sehen. Jeder dieser Testplätze erhält eine kleine Unterverteilung, in der die Kleinspannungsversorgung, also Netzteile und Klemmen, untergebracht sein soll. Als Befestigung für die Testgeräte sind drei Hutschienen, fest an der Wand angeschraubt, vorgesehen.
- (b) Unter den Testplätzen verläuft über die komplette Seite ein Installationskanal (2), in dem die Steckdosen und Netzwerkdosens untergebracht sind.
- (c) In der linken Ecke (3) ist die neue Unterverteilung zu sehen. In ihr ist die Niederspannungsabsicherung untergebracht.
- (d) Rechts daneben (4) wurden als Beispiel mehrere IPCs fest an der Wand angebracht, diese sollen für die Langzeittests der Software dienen und müssen so nicht modular sein.
- (e) Darunter (5) ist ein Arbeitsplatz angedacht, an dem kleinere Hardwarearbeiten durchgeführt werden können.
- (f) Unten in der Mitte (6) ist ein Doppeltisch vorgesehen, an diesem werden zwei Officearbeitsplätze aufgebaut. So kann, während die Tests laufen, auch an anderen Aufgaben gearbeitet werden.
- (g) Unter der Unterverteilung (7) wurde das Serverrack für das rauminterne Netzwerk platziert. In ihm sind die Patchfelder und Switches untergebracht.
- (h) In der linken unteren Ecke (8) ist noch ein Standschrank zu sehen, dieser dient als Abstellplatz, in den nicht benötigte Geräte verräumt werden können.



Abbildung 3.1: Iteration 1 Raum

3.3.2 Iteration 2

In der zweiten Iteration des Raumes sind bereits einige Änderungen zu sehen.

- (a) Die wichtigste Änderung ist hier die Unterverteilung (1), diese sitzt nun neben der Eingangstür. Gewählt wurde dieser Platz, da nun feststand von wo aus die Zuleitung in den Raum kommen würde - über der aktuellen Position der Unterverteilung. So wurde für die Zuleitung ein möglichst kurzer Weg gewählt.
- (b) Links daneben (2) ist nun anstatt des festen Testplatzes ein Regal entstanden. In diesem sollen nun die Langzeittests stattfinden. Dank des Regals ist man deutlich flexibler als bei einer festen Installation der IPCs.
- (c) Wiederum links daneben sind nun noch zwei weitere Testplätze (3) zu sehen. Diese sind mittlerweile modular aufgebaut, das heißt man kann, durch die Aluminiumprofile an der Wand, die Höhe und Anzahl der Hutschienen ganz nach den eigenen Bedürfnissen anpassen.
- (d) An diese Profile lassen sich ebenfalls weitere Kabelkanäle oder auch bereits vorinstallierte und verdrahtete Montageplatten anbringen.
- (e) Zwischen den beiden Testplätzen wurde der Installationskanal (4) bis unter die Decke nach oben gezogen, um auch neben den höher angebrachten Hutschienen noch Steck- und Netzwerkdosens zu haben.
- (f) Beide Testplätze haben zusätzlich jeweils ein Regal bekommen, in diesem lassen sich nicht-hutschienengeeignete Geräte wie zum Beispiel ein Netzwerkswitch, oder Ähnliches unterbringen.
- (g) Das Serverrack (5) hat seinen Platz behalten, da es an möglichen anderen Stellen im Raum zu viel Platz vereinnahmt hätte.



Abbildung 3.2: Iteration 2 Raum

3.3.3 Iteration 3

Die dritte Version des Raumes besitzt auf den ersten Blick keine gravierenden Änderungen, dafür wurden bei dieser Version bereits einige Details eingeplant/beachtet.

- (a) Das Regal für die Langzeittest-IPCs (1) wurde auf die andere Seite, über einen Officearbeitsplatz (2) verlegt, somit entstand auf der anderen Seite Platz für einen weiteren Testplatz.
- (b) Dieser neu entstandene Platz (3) wurde, im Gegensatz zu den übrigen, gespiegelt konzeptioniert, da ansonsten das Regal in der Ecke angebracht wäre, was nicht besonders benutzerfreundlich ist.
- (c) Die ursprünglichen beiden Testplätze (4,5) wanderten somit ebenfalls weiter in die Ecke, wodurch wiederum Platz für noch einen weiteren Testplatz (6) entstand.
- (d) Des Weiteren wurde der dritte Arbeitstisch weggelassen und durch eine verlängerte Ablage (7) ersetzt, da sich herauskristallisierte, dass im Testraum kaum Arbeiten durchgeführt werden müssen.
- (e) Die bereits in der vorherigen Version etablierten, vertikalen Installationskanäle sind nun an allen Testplätzen, jeweils neben den Regalen für die losen Geräte, vorhanden.
- (f) In ihnen sind bereits die ersten Layout-Entwürfe für die Steckdosen und Netzwerkdosen zu sehen, hier wurde auch festgelegt, dass die schaltbaren Steckdosen eine andere Abdeckungsfarbe erhalten.
- (g) Der Abstellschrank (8) ist in dieser Version noch enthalten, jedoch war bereits hier nicht sicher ob er denn benötigt wird, da die Geräte alle im richtigen Lager aufbewahrt werden und somit der zweite Officearbeitsplatz (9) gedreht werden könnte und nicht so weit in den Raum stünde.

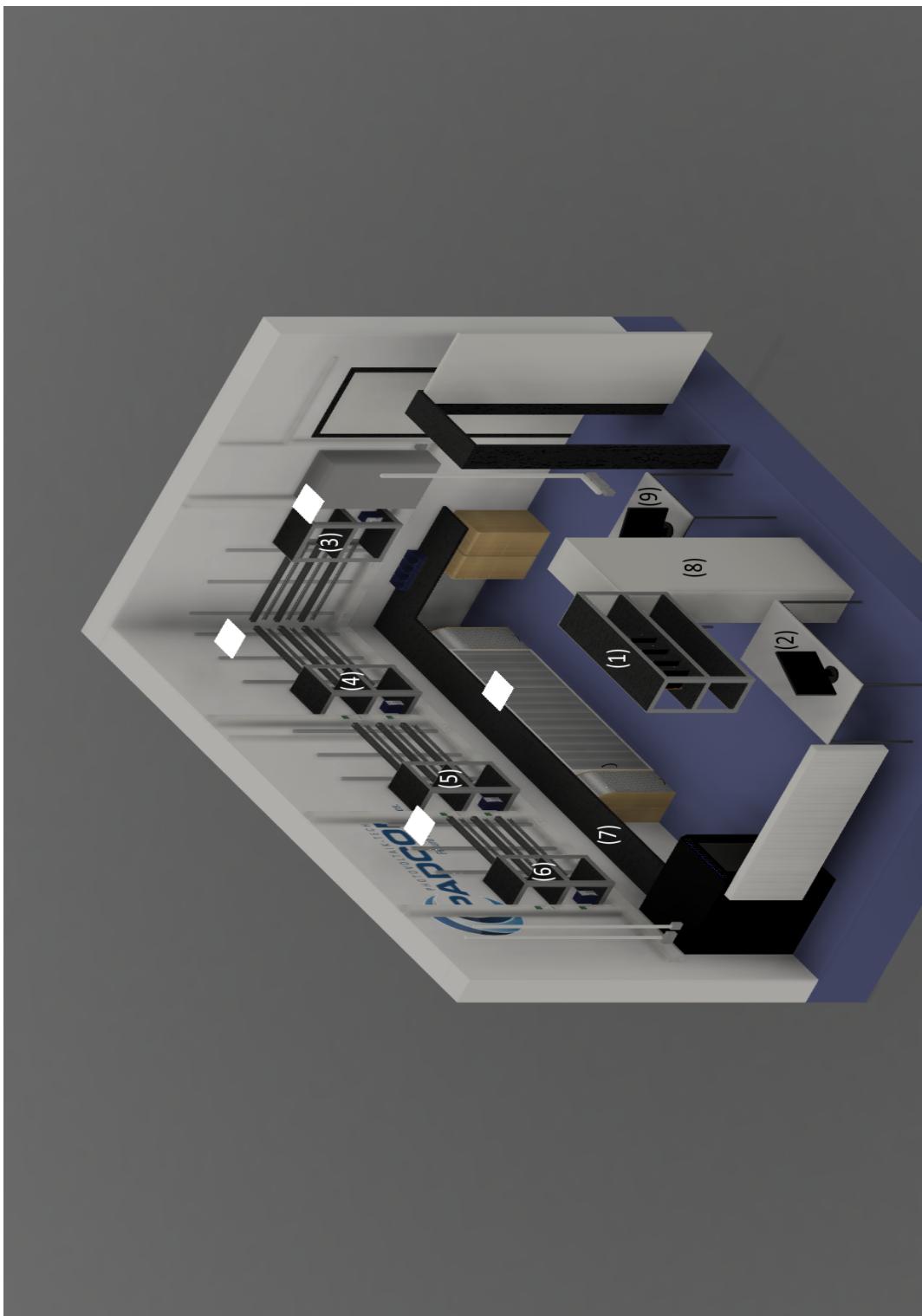


Abbildung 3.3: Iteration 3 Raum

3.3.4 Iteration 4

Dies stellt die finale, vierte Version dar.

- (a) Erstmals taucht hier in der Planung der Serverschrank (1) für die firmeneigenen Server auf. Dieser war vorher leider noch nicht vorgesehen. Da das Testnetzwerk jedoch nicht mit dem firmeninternen Servern in einen Schrank kommen soll, um Sicherheitsrisiken aus dem Weg zu gehen, musste auch das alte Serverack erhalten bleiben. In ihm sind, wie bereits zuvor, alle Netzwerkgeräte des Testnetzwerkes untergebracht.
- (b) Das Serverack (2) wurde jedoch um 90° gedreht, somit ragt es nichtmehr über die Heizung hinaus.
- (c) Testplatz drei und vier (3) wurden zusammengelegt, um auch größere Aufbauten zu ermöglichen.
- (d) Das Regal für die Langzeittests ist weggefallen, diese Tests werden nun an einem normalen Testplatz druchgeführt.
- (e) An dieser Stelle wurde ein neuer, großer Testplatz (4) konzeptioniert. Dieser ist für die firmeneigenen PCBs gedacht. Da diese meist eine dedizierte Spannungsversorgung benötigen wurde hier auf die Kleinspannungsversorgung verzichtet.
- (f) Durch den enormen Platzverbrauch des neuen Servers musste der Officeplatz (5) unter diesen Testplatz wandern. So kann dieser auch gleichzeitig für Hardwarearbeiten mitbenutzt werden, da an ihm einzig die Inventarsoftware benutzt wird.
- (g) Die Unterverteilung (6) hat nun ihre finale Größe, dies hat Testplatz 1 (7) leider etwas Platz weggenommen.
- (h) An den Installationskanälen ist das finale Layout der Steckdosen und Netzwerkdosens zu sehen. Die Kleinspannungsversorgung wird mittels Klemmen auf den Hutschienen angebracht und ist auf dem Bild nicht zu sehen.
- (i) In der linken unteren Ecke ist zusätzlich noch eine Klimaanlage (8) platziert worden, um den, durch den Server entstehenden, höheren Temperaturen entgegen zu wirken.
- (j) An den Testplätzen 1-4 wurde jeweils zwischen Aluprofilen und vertikalem Installationskanal etwas Platz gelassen um hier Regale (9) anzubringen. Da jedoch noch nicht zu 100% fest stand ob diese wirklich montiert werden, wurden sie auf dem Render weggelassen.



Abbildung 3.4: Iteration 4 Raum

4 Elektrische Umsetzung

4.1 Austattung der einzelnen Bereiche

4.1.1 Testplatz 1-4

- Spannungsversorgung 230V, teils schaltbar, überwacht
- Kleinspannungsversorgung 24V, 12V, überwacht
- Netzwerkanbindung

4.1.2 Testplatz PCB

- Spannungsversorgung 230V, teils schaltbar, überwacht
- Netzwerkanbindung

4.1.3 Der Serverschrank

- Spannungsversorgung 230V, überwacht
- Netzwerkanbindung

4.1.4 Das Serverrack

- Spannungsversorgung 230V, überwacht
- Netzwerkanbindung

4.1.5 Die Unterverteilung

- 400V 35A Zuleitung
- Netzwerkanbindung

4.2 Die Unterverteilung

Bei der Unterverteilung entschieden wir uns, da keine Gründe dagegen sprachen, wieder für einen Standardversorger der Firma. Hier für einen Rittal AE 1180.500 mit einer Tiefe von 300mm, 800mm Breite und 1000mm Höhe. Dieser kommt in der Standard-Schalschrankfarbe RAL 7035 und hat die Schutzart IP66 nach IEC 60 529. Dies ist ein geerdeter Schalschrank der Schutzkasse I. Alle Geräte wurden auf einer Montageplatte montiert.

4.2.1 Zuleitung

Als Zuleitung wurde auf eine bereits vorhandene Reserve in der Hauptverteilung zurückgegriffen. Diese war über einen dreipoligen G32A Leitungsschutzschalter abgesichert. Da hier jedoch bei Kurzschläßen keine Selektivität gegeben wäre wurde der Firma empfohlen, diesen durch einfache D02 Schmelzsicherungen zu ersetzen.

Absicherung:

Um die Leistungsaufnahme zu berechnen wurden alle Verbraucher zusammengerechnet. Hiefür wurde jedoch nur die am meisten belastete Phase betrachtet, wobei jedoch auf eine gleichmäßige Verteilung der Last geachtet wurde. Dies lässt sich im Stromlaufplan erkennen. Ebenso ist zu erkennen, dass L1 die am meisten belastete Phase ist.

Folgende Verbraucher gibt es:

- 16A Serversteckdose
- 16A Steckdose Testplatz 1
- 16A Steckdose Testplatz 4
- Netzteil 24V - Testplatz 2: ABB CP-E 24/5.0 = 0,83A
- Netzteil 12V - Testplatz 2: ABB CP-E 12/10.0 = 0,83A

Da die Testplätze jedoch kaum belastet werden, kann man hier von einem Gleichzeitigkeitsfaktor von 0,5 ausgehen. Dies ergibt also

$$16A + (16A + 0,83A \cdot 2 \cdot 0,5) = 32,83A.$$

Somit kann eine 35A Schmelzsicherung eingesetzt werden.

Leitungsberechnung:

Die Leitung wurde auf Grundlage folgender Brechnungen in 10mm² ausgeführt.

$$\begin{aligned} A &= \frac{2 \cdot L \cdot I \cdot \cos(\varphi)}{y \cdot U_a} \\ &= \frac{2 \cdot 45m \cdot 35A \cdot 1}{56 \frac{m}{\Omega \cdot mm^2} \cdot 6,9V} \\ &= 8,15mm^2 \approx 10mm^2 \end{aligned}$$

Legende :

- A = Leitungsquerschnitt
- L = einfache Leitungslänge
- I = Leiterstrom
- $\cos(\varphi)$ = Wirkungsgrad
- y = Leitfähigkeit
- U_a = Spannungsfall

4.2.2 Niederspannung

Als Hauptschalter wurde, wie bei den anderen Geräten im Schaltschrank auch, auf die Firma Siemes zurückgegriffen. Hier wurde ein Siemens 5TL13630 mit einem Schaltvermögen von 63A verbaut. Für die Spannungsversorgung wollten wir Brandschutzschalter verbauen. Diese erhöhen, gerade auch bei Langzeittests, bei denen nicht immer eine Person vor Ort ist, die Sicherheit.

Jeder Testplatz bekommt seine eigene Absicherung. Für diese Stromkreise wurden RC-BO's der Charakteristik B mit einem Auslösestrom 30mA verbaut. Aufgrund des begrenzten Platzangebots wurde hier auf Brandschutzschalter der Firma Siemens zurückgegriffen, da diese Brandschutzschalter und RCBO in einer Einheit mit der Breite von drei Teilungseinheiten

kombinieren. Bei dem geläufigen Zulieferer von Padcon, ABB, ist diese Kombination vier Teilelungseinheiten breit. Dies erscheint im ersten Moment nur als ein kleiner Breitenunterschied, hätte aber im Gesamten die Auswirkung gehabt, dass keine Platzreserven mehr vorhanden geblieben wären. Konkret wurde hier auf die Geräte 5SM6021-1 (Brandschutzschaltermodul) und 5SU1656-6KK16 (RCBO 16A/300mA) / 5SU1356-6KK16 (RCBO 16A/30mA) zurückgegriffen. Auf Wunsch der Firma wurden 300mA RCBO's für die beiden Serversteckdosen vorgesehen. Diese wurden außer Reichweite von Personen, kurz unterhalb der Decke installiert. Trotz bestehender Vorchriften war ein abschließbarer Deckel nicht gewollt, somit wurden die Steckdosen zusätzlich als *ausschließlich für Fachkräfte beschriftet*.

Um die Stromversorgung später auch aus der Ferne zu überwachen, wurden an allen Sicherungen, die direkt aus dem Schaltschrank gehen, Hilfsschalter 5ST3010 verbaut. Diese werden über ein Moxa ioLogik E1210 überwacht und können so über das firmeninterne Netzwerk, mit Hilfe einer Weboberfläche, kontrolliert werden. Gerade für die Serverversorgung ist dies von Vorteil.

Bestimmte Steckdosen an den Testplätzen sollen auch über das Netzwerk geschalten werden können. Diese Steckdosen wurden auf Wunsch der Firma mit blauen Einsätzen versehen. Für die Netzwerkanbindung wurde auch hier wieder auch Moxa ioLogik Geräte gesetzt, da diese in der Firma bereits mehrfach eingesetzt werden. Da diese Komponenten jedoch nicht für größere Lasten geeignet sind, wurden Finder 45.61.9.024.0000 Relais dahinter geschalten. Diese besitzen eine für 24V ausgelegte Steuertspule und haben ein Schaltvermögen von 230V/16A. Mit Hilfe dieser geschalteten Steckdosen kann z.B. ein kurzzeitiger Spannungsaufall auch aus der Ferne simuliert werden.

4.2.3 Kleinspannung

Da viele der zu testenden Geräte mit einer Kleinspannung von 12V oder 24V versorgt werden müssen, wurde an den Testplätzen eine eben solche auch vorgesehen. Es wurden jedoch - nicht wie ursprünglich geplant 24V, 12V und 5V - nur 24V und 12V-Spannungsversorgungen verbaut. Auf die 5V-Versorgung wurde verzichtet, da diese nur sehr selten gebraucht wird und den Mehraufwand nicht gerechtfertigt hätte. Hierfür muss nun ein externes Netzteil verwendet werden.

Für die Netzteile wurde auch hier wieder auf firmenübliche Geräte zurückgegriffen. Es wurden pro Testplatz je ein ABB CP-E 24/5.0 und ein CP-E12/10.0 verbaut. Diese liefern, wie

bereits aus der Bezeichnung herauslesbar, 24V/5A und 12V/10A. Die 24V Geräte besitzen schon standardmäßig einen Hilfskontakt, der Fehlerfälle anzeigt. Dieser wurde auch auf dem ioLogik Gerät mit angeschlossen und lässt sich nun ebenfalls über das Internet überwachen.

Um die Kleinspannung an den Testplätzen auch irgendwie zugänglich zu machen gab es im Verlauf der Planung mehrere Ideen. Die wohl sauberste Lösung wäre ein Einsatz im Installationskanal gewesen, in dem mehrere Buchsen für Laborstecker untergebracht werden.

Diese Lösung wurde jedoch, da immer Kabel mit Laborstecker benötigt werden, als zu umständlich und auch deutlich zu teuer angesehen. So fiel unsere Wahl auf normale Reihenklemmen. Um diese nicht frei an einen Testplatz hängen zu müssen wurden, wie in Iteration 1 () zu sehen, kleine Schaltschränke vorgesehen. Diese wurden jedoch auch als zu umständlich abgetan.

Auf Wunsch der Firma hin wurden nun die Klemmen der Kleinspannungsversorgung jeweils auf die mittlere Hutschiene der Testplätze gesetzt.

4.2.4 Wärmeberechnung

Nach Abschluss der Geräteplanung musste noch eine Wärmeberechnung für den Schaltschrank durchgeführt werden, um zu überprüfen, ob die Passivkühlung durch das Schaltschrankgehäuse aussreicht.

Für die Wärmeberechnung des Schaltschranks werden folgende Angaben benötigt:

- $T_{u\ max}$ = maximale Umgebungstemperatur -> max. 20°C
- $T_{i\ max}$ = maximale Schaltschrankinnentemperatur -> max. 40°C
- Schrankabmessungen (hxwxt) in mm -> 1000mmx800mmx300mm
- Aufstellungsart des Schaltschranks -> Einzelgehäuse für Wandaufbau
- Wärmedurchgangskoeffizient k in Abhängigkeit des Materials aus dem der Schaltschrank besteht -> 5,5 W/(m²K) für Stahlblech lackiert

Aus den Abmessungen und der Aufstellungsart ergibt sich eine effektive Schaltschränkoberfläche (Wärmeabstrahlfläche) nach DIN VDE 0660 Teil 500 von:

$$\begin{aligned}
 A &= 1,4 \cdot B \cdot (H + T) + 1,8 \cdot T \cdot H \\
 &= 1,4 \cdot 0,8m \cdot (1,0m + 0,3m) + 1,8 \cdot 0,3m \cdot 1,0m \\
 &= 1,996m^2
 \end{aligned}$$

Für die Berechnung der Wärmeabfuhr durch die Gehäusewand wird folgende Formel verwendet:

$$Q = U \cdot A \cdot (T_i - T_u)$$

SYMBOL	BESCHREIBUNG	EINHEIT
Q	Wärmeabgabe der Schrankoberfläche	W
U	Wärmedurchgangskoeffizient	$\frac{W}{m \cdot K}$
A	Effektive Schaltschränkoberfläche	m
T_i	Schaltschränkinnentemperatur	$^{\circ}C$
T_u	Umgebungstemperatur	$^{\circ}C$

Werden nun alle Werte in die Rechnung eingesetzt, so ergibt sich folgendes Ergebnis:

$$Q = 5,5 \frac{W}{m \cdot K} \cdot 1,996m^2 \cdot (40^{\circ}C - 20^{\circ}C) = 219,56W$$

Installierte Verlustleistung Q_t :

Anzahl	Gerät	Verlustleistung	Gesamt
4	ABB CP-E 24/5.0	20 W	80 W
4	ABB CP-E 12/10.0	24 W	96 W
1	ABB CP-E 24/0.75	4,45 W	4,45 W
1	SIEMENS 5TL13630	2,2 W je Pol	6,6 W
9	SIEMENS 5SM6021-1	0,6 W	5,4 W
7	SIEMENS 5SU1656-6KK16	3 W	21 W
7	SIEMENS 5SU1356-6KK16	3 W	21 W
6	FINDER 40.61.9.024.0000	1,2 W	7,2 W
1	MOXA ioLogik E1210	2,64 W	2,64 W
1	MOXE ioLogik E1214	4,51 W	4,51 W
		Gesamtverlustleistung:	248,8 W

Tabelle 2: Gesamtverlustleistung Unterverteilung

Vergleicht man nun die installierte Verlustleistung mit der Wärmeabfuhr des Schaltschranks so fällt auf, dass die installierte Verlustleistung höher als die Wärmeabfuhr ist. Dies ließ sich leider nicht vermeiden, da der Schaltschrank schon vor Abschluss der Planung von der Firma bestellt und angeliefert wurde. Es wurde der Firma jedoch empfohlen, eine geeignete Lüftung in den Schrank einzubauen, als Beispiel wäre hier der Filterlüfter SK 3237.100 von Rittal geeignet. Da jedoch aktuell noch tiefere Umgebungstemperaturen herrschten und die Lüftung noch nicht notwendig war, wurde erstmal darauf verzichtet. Ein Test in einer längeranhaltenden Wärmeperiode, bei der sich der Raum auf die maximale Raumtemperatur aufheizt, sowie erste Erfahrungen, wie vieler Geräte denn Gleichzeitig in Betrieb sind, soll zeigen, ob eine Lüftung notwendig wird.

4.2.5 Klemmen

Als Klemmen wurden firmentypisch, PHOENIX Contact Installationsklemmen verwendet. Hier wurden folgende Typen verwendet:

- PT 10, PT10 BU & PT10 PE als Zuleitungsklemmen
- STI 2,5-PE/L/N als Klemmen für die Stromkreise
- PT 1.5/S-QUATTRO RD als Klemmen für die Kleinspannungskreise
-

Da für die zwei Moxa-Geräte jeweils eine Netzwerkanbindung benötigt wird, musste Ethernet in den Schrank gelegt werden. Dies wurde mit Hilfe zweier Patchmodule für Hutschienen (ABB IPM/S1.1) ausgeführt. So konnte, wie zu den anderen Netzwerkdosens auch, Installationskabel bis in den Schaltschrank gelegt werden und im Schaltschrank dann mit Patchkabeln weiter auf die beiden Geräte gegangen werden.

4.2.6 Montage

Da der Schaltschrank neben der Tür angebracht wurde und der optimale Fluchtweg, durch den weit in den Raum ragenden Server, leicht schräg von der Tür weg führt, wurde - um den Fluchtweg feizuhalten - der Türanschlag von rechts nach links gewechselt. Die Unterverteilung wurde jedoch trotzdem 30cm von der Tür entfernt angebracht um genügend Platz für einen uneingeschränkten Durchgang durch die Tür zu lassen. Die Einführung und die Klemmen sind im Schaltschrank nach unten gewandert, da nur ein einziges Kabel - die Zuleitung - von oben kommt.

4.3 Das Netzwerk

In einem Testraum, der nach aktuellem Stand der Technik gebaut wird, darf ein Netzwerk nicht fehlen. So wurde jeder Testplatz mit vier Netzwerkbuchsen ausgestattet, diese werden im dafür vorgesehenen Serverrack so konfiguriert, dass der ganze Raum im Firmennetzwerk eine Insel darstellt und sich die Plätze untereinander auch nur wenn es gewollt ist, erreichen können. Als Installationskabel wurde S-FTP Cat.7 verwendet.

Das dafür vergesehene, etwas kleinere Serverrack enthält am Ende nur zwei 24-Port Patchfelder, einen 48-Port Netzwerkswitch und eine hardwaremäßige Firewall. Diese Patchfelder wurden, zu Übungszwecken, von den Auszubildenden der Firma angeschlossen. Alle anderen Arbeiten am Netzwerk wurden selbstverständlich von uns ausgeführt. Das Netzwerk wurde bereits vor der offiziellen Abnahme vom Systemadmin durchgemessen und auf fehlerfreiheit geprüft. So wurde dies bereits frühzeitig abgenommen.

In Zukunft sollen im Serverrack noch weitere Geräte eingebaut werden, die langsame Verbindungen und Paketverluste simulieren zu können. So kann auch dieses Fehlerszenario bereits hausintern getestet werden.

Teil III

TUInventory

[sv]

5 Begriffsdefinitionen

Begriff	Definition
Duck-Typing	Der Typ einer Instanz wird dadurch beschrieben/festgelegt, welche Member sie besitzt (Von engl.: "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 ⁵ .
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 wettkämpfen, um die Ausgabe als erstes zu beeinflussen ⁶ . Probleme einer Race-Condition sind beispielsweise, dass sie oftmals verschwindet, wenn das Programm mittels Debugger betrachtet wird.

⁵[EK17, S. 369]

⁶[Wik18a, 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 ⁷ .
Singleton	Ein Singleton ist ein Entwurfsmuster, bei dem sichergestellt ist, dass nur eine einzige Instanz einer Klasse existiert (z.B. in Python <code>None</code>).
Thread	(dt.: Faden, Strang) Threads sind im Grunde genommen leichtgewichtige Prozesse (Ein Prozess kann mehrere Threads besitzen). In CPython (erläutert in Abschnitt 7) 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.

⁷[Wik19b, 06.03.19 - 12:07 Uhr]

6 Einführung und Problemanalyse

Um ein flüssiges Arbeiten mit dem Testraum zu gewährleisten, ist ein Inventarsystem erforderlich. Dieses System ist grundsätzlich aus drei Teilen aufgebaut:

1. Ein Codierungssystem zum eindeutigen Markieren von Geräten
2. Eine grafische Desktopanwendung, über die Geräte, bzw. eine Datenbank verwaltet werden können
3. Eine Kommandozeilen-Schnittstelle, um spezielle Funktionen extern verfügbar zu machen

6.1 Codierungssystem

An das Codierungssystem werden folgende Anforderungen gestellt:

- Problemlos an einer Vielzahl von Geräten zu befestigen (keine komplexe Befestigung wie z.B. NFC-Tag)
- Nicht-properitäres System, um auch von anderen Geräten/Anwendungen verwertet werden zu können
- Einfache Auswertung und Codierung von selbst definierten Daten

Generell gibt es einige Möglichkeiten zur Codierung, welche mehr oder weniger gut für diesen Use Case geeignet sind:

RFID RFID-Chips sind eher unpraktisch in Sachen nachträglicher Anbringung an beliebigen Geräten.

EAN-Code EAN-Codes sind auf vielen Geräten bereits vorhanden, was sie zu einer attraktiven Wahl macht. Diese bereits vorhandenen Codes sind jedoch nicht für jedes Gerät eindeutig. Tatsächlich sind sie eher hinderlich, da beim Anbringen eines eigenen Codes Verwechslungsgefahr besteht.

QR-Code QR-Codes sind sehr uneingeschränkt in Sachen Anbringung, können beliebige Daten halten und da sie optisch ausgewertet werden können, kann man sie mit jedem modernen Gerät wie einem Smartphone oder einem PC mit Kamera einfach auswerten.

Die Wahl fiel daher auf QR-Codes.

6.2 Desktopanwendung und Kommandozeilen-Schnittstelle

Bei der Umsetzung der Desktopanwendung und der Kommandozeilen-Schnittstelle, die letzen Endes auf demselben Softwarestack aufbauen sollen, stellt sich nach einigen grundlegenden Überlegungen die Frage der gewählten Programmiersprache.

Einige Anforderungen sind:

- Lauffähigkeit auf mehrere Plattformen
- Wartung/Weiterentwicklung durch Padcon möglich

Hierbei gab es grundsätzlich drei Sprachen, die ernsthaft in Betracht gezogen wurden (entweder für das ganze Projekt oder um einzelne Funktionalitäten zu implementieren):

Java Da die Kraftwerksüberwachungssoftware größtenteils in Java geschrieben ist und dementsprechend viele Java-Programmierer bei Padcon zur Verfügung stehen, wäre Java eine attraktive Sprache. Auch die C#-ähnliche Syntax spricht dafür.

C# Da C# bereits aus der Techniker-Ausbildung bekannt ist, wäre es eine denkbare Lösung. Jedoch ist C# immer noch nur bedingt Cross-Plattform-fähig⁸. Eine Weiterentwicklung durch Padcon ist bedingt möglich, so ist die grundlegende Syntax von C# ähnlich der von Java.

Python Python ist eine grundlegend andere Sprache als Java und C#⁹ und attraktiv, da es eine sehr schnelle Entwicklung ermöglicht. Des Weiteren ist es bei praktisch allen Linux-Distributionen vorinstalliert und bietet eine Vielzahl an Möglichkeiten um plattformübergreifende Anwendungen zu entwickeln.

Java wurde schnell verworfen, da es im Zeitraum des Projektes unrealistisch ist, die Anwendung in Java zu entwickeln und dabei idiomatischen, guten Code zu produzieren. C# wurde ebenfalls für ungeeignet befunden, da mit Problemen mit Linux gerechnet wurde. Außerdem liegt auch bei C# ein eher langsamer Entwicklungsprozess vor. Der Vorteil der besseren Performance von C# und Java gegenüber CPython (der Standardimplementierung von Python, andere Implementierungen bieten vergleichbare Performance zu C#), ist hier eher irrelevant, da die Anwendung aufgrund der grafischen Oberfläche ohnehin die meiste Zeit „idle“ ist¹⁰.

⁸Mit dem Mono-Framework besteht grundsätzlich die Möglichkeit, C# Anwendungen für Linux zu entwickeln

⁹Mehr dazu im Kapitel 7

¹⁰Sicher wäre die Implementierung auch mit C++, Rust oder einer Vielzahl anderer Sprachen möglich gewesen, jedoch liegen in den meisten Fällen die gleichen „Probleme“, wie bei C# und Java vor: Es würde sich einfach nicht lohnen eine performantere Sprache auf Kosten der Entwicklungszeit zu nutzen. Sollte zu einem späteren Zeitpunkt eine Erweiterung nötig sein, bei der sich herausstellt, dass die Geschwindigkeit von Python ein Flaschenhals ist,

Python bietet desweiteren exzellente Interoperabilität mit C und kann auf viele große Frameworks aus den Bereichen C/C++ zurückgreifen (z.B. Qt und OpenCV). Der Pflege der Software durch Padcon steht auch bei Python nichts im Wege, da,

1. sich der Code größtenteils wie Pseudocode oder ein englischer Satz liest.
2. die Auszubildenden im Bereich Anwendungsentwicklung in Zukunft auch Python lernen.

Aufgrund dieser Punkte fiel die Wahl der Sprache auf Python.

so lässt sich dieses performancekritische Modul in einer maschinennahen Sprache wie C oder Rust entwickeln und problemlos einbinden (oder man nutzt eine der alternativen Python-Implementierungen, welche in Kapitel 7 erläutert werden).

7 Python 3

Bei Python 3 handelt es sich um eine universell einsetzbare, plattformübergreifende, dynamisch typisierte und multiparadigmatische Hochsprache, welche in verschiedensten Implementierungen als Open Source Projekt entwickelt wird¹¹. 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.”¹²

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 CLI¹³ und Integration in Visual Studio)¹⁴.

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

7.1 Funktionen und Methoden in Python

Eine Besonderheit Pythons ist, dass alles, sei es Klasse, Instanz oder Funktion, ein Objekt ist¹⁵. 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 erläutert¹⁶:

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

¹¹[KB18, S. 65]

¹²[Pyt18b, 18.12.18 - 16:32 Uhr]

¹³Common Language Infrastructure

¹⁴[Rei18, 18.12.18 - 17:08 Uhr]

¹⁵Selbst der Code welcher ausgeführt wird, ist ein, über den `__code__` Member erreichbares Objekt

¹⁶Neben 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 [Het17, 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* [Ros17, 18.12.18 - 17:23 Uhr].

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

Algorithmus 1 Pythons Funktionsmodell

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

    @classmethod
    def my_classmethod(cls):
        pass

    @staticmethod
    def my_static_method():
        pass

is_a_function_now = MyClass.my_method
```

7.2 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 Namensgebungskonventionen¹⁷. PEP 257 hingegen beschäftigt sich mit den sogenannten Docstrings und deren Format¹⁸. 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 C ist sie dennoch zu finden (`{modulname}.html`)). Als weitere Referenz für das genutzte Format galten ein von Google veröffentlichtes Dokument¹⁹, sowie der Talk “Beyond PEP 8 -- Best practices for beautiful

¹⁷[Ros01, 18.12.18 - 16:17 Uhr]

¹⁸[Goo01, 18.12.18 - 16:17 Uhr]

¹⁹[Goo18, 18.12.18 - 17:36 Uhr]

intelligible code“ von Raymond Hettinger, einem der Core Developer von CPython²⁰.

Abschließend ist es jedoch ratsam, bei all diesen Formatforschriften nochmal Bezug auf den Anfang von PEP 8 zu nehmen: “A Foolish Consistency is the Hobgoblin of Little Minds”²¹ - man sollte also wissen, wann es die bessere Entscheidung ist mit den Vorschriften zu brechen.

7.3 Vorwort und Übersicht über die Anwendung

„All computers are now parallel... Parallel programming *is* programming.“

- Michael McCool²²

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:

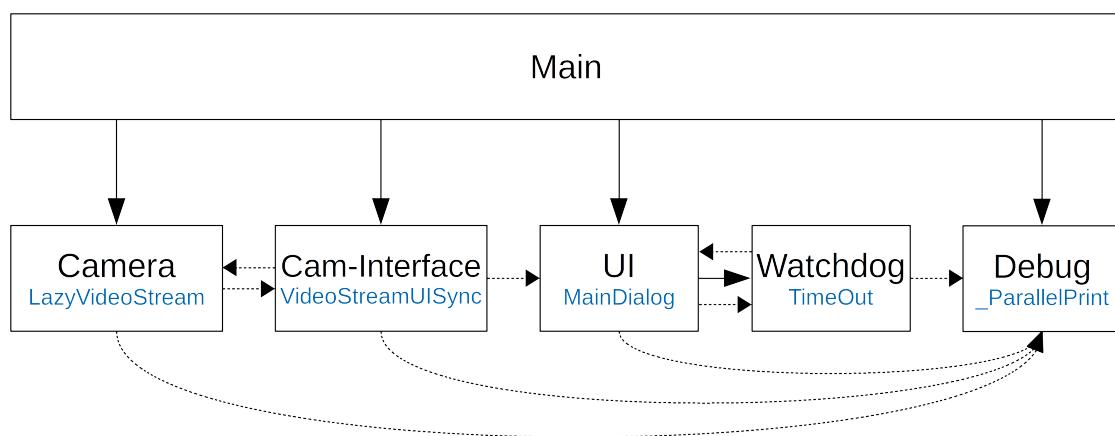


Abbildung 7.1: Übersicht über Verhältnisse von Threads

Des Weiteren 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

²⁰[Het15, 18.12.18 - 17:40 Uhr]

²¹Siehe Fußnote 17 (PEP8)

²²[BO18, S. 1]

Camera class

classes.py

BigInt class

setup_context_session method

ContextSession class

Producer class

Article class

Device class

PhoneNumber class

NoNumberFoundWarning class

Location class

User class

Responsibility class

Timeout class, thread

VideoStreamUISync class, thread

cli

cli_get_barcode.py

keys.py

generate_key method

read_keys method

logger.py

logger RootLogger instance

main.py

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_article** 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
- check_if_file_exists** method
- normalize_filename** method

8 Command Line Interface

Im CLI sollte ursprünglich der volle Umfang des Barcodereaders umgesetzt werden, dies wurde auch so durchgeführt. Jedoch stellte sich heraus, dass lediglich das Extrahieren eines Barcodes aus einem gegebenen Bild nötig war. Mit Blick auf die Größe des CLI wurde daher die restliche Funktionalität entfernt.

Das CLI setzt intern auf Pythons eingebautes Modul *argparse*, bzw. dessen *ArgumentParser*. Das Interface zeigt sich über Kommandozeilenparameter *-h* dabei wie folgt:

Algorithmus 2 Schnittstellenhilfe zu *cli_get_barcode.py*

```
/$ python cli_get_barcode.py -h
usage: cli_get_barcode.py [-h] [-i PATH] [-d] [-tb]

Extract all barcodes from an image

optional arguments:
  -h, --help            show this help message and exit
  -i PATH, --image PATH
                        Path to image
  -d, --detailed        Return all extracted information
  -tb, --enable_tracebacks
                        Enable tracebacks rather than custom messages
```

Über *-h* kann also wie oben dargestellt eine Schnittstellenbeschreibung angezeigt werden. *-i* ist der Parameter, über welchen der Pfad zu einem Bild übergeben wird. Aus diesem Bild wird der Barcode ausgelesen und zurückgegeben. In der Regel wird lediglich der Inhalt eines Codes benötigt, jedoch kann es mitunter nützlich sein, mehr Details (wie z.B. um welche Art Code es sich handelt) zu erhalten. Daher existiert das Flag *-d*, welches es ermöglicht, alle intern verfügbaren Informationen über das Bild, bzw. die darin enthaltenen Codes wiederzugeben. Hierüber ist es außerdem möglich mehrere Codes aus einem Bild zu extrahieren.

Der letzte Parameter ist *-tb*. *cli_get_barcode* ist so aufgebaut, dass alle möglichen Fehler abgefangen werden und zu Fehlern, die dem Nutzer (der potentiell mit einer Python-Fehlermeldung nichts anfangen kann) mehr sagen, gewandelt werden. Jedoch ist in manchen Fällen ein sog. *Traceback* nützlich. Daher ist es über den Parameter *-tb* möglich, die benutzerdefinierten Fehler abzustellen und stattdessen einen Python Traceback zu erhalten.

Das CLI läuft schlussendlich auf einem Server und kann über eine RESTful API angesprochen werden. Um hier ein einfaches Setup zu ermöglichen, wird es mittels pyinstaller als *onefile* bzw. *onedir* Lösung verpackt. Hierbei wird zum Beispiel für Windows eine exe erzeugt, die eine volle Python-Installation und alle benötigten Packages beinhaltet. Die *onefile* Lösung stellt sich jedoch als eher ungeeignet heraus, da hier zum Programmstart ein Archiv temporär entpackt wird, was einen enormen Overhead schafft, der bei so kurzer Programmlaufzeit nicht gerechtfertigt ist. Daher wurde das CLI dem Auftraggeber als Linux-*onedir*-Variante übergeben.

9 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)*²³ 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.

9.1 Das Datenbankmodell

Das mit SQLAlchemy umgesetzte Datenbankmodell wurde zunächst wie folgt geplant:

²³[Bay18, 18.12.18 - 17:26 Uhr]

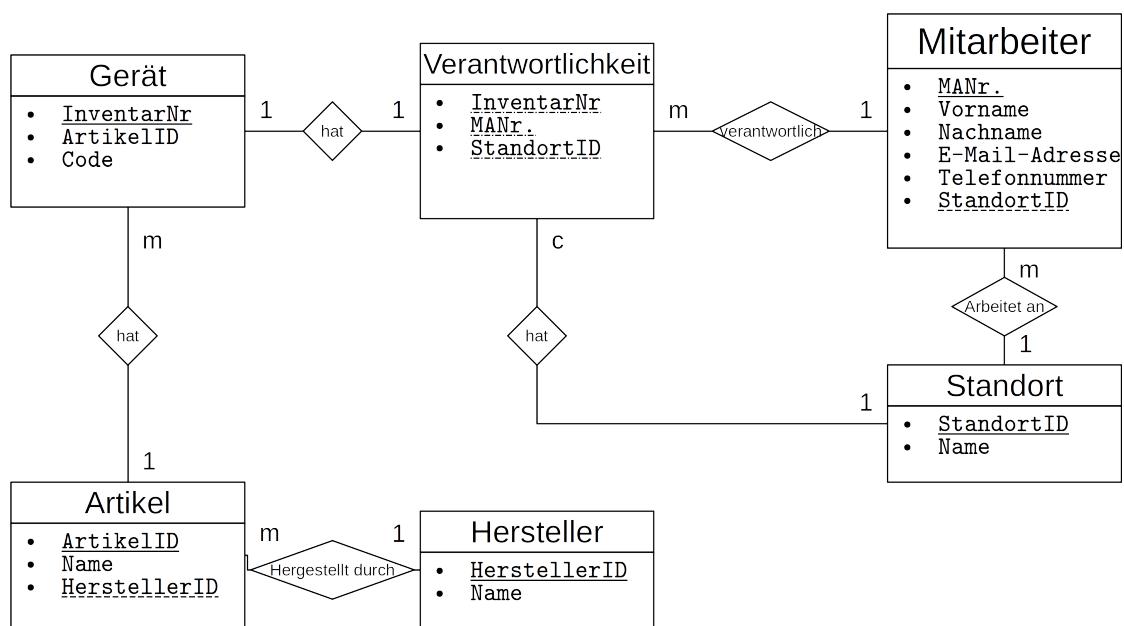


Abbildung 9.1: Entity-Relationship-Diagramm

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).

9.2 Implementierung des Datenbankmodells

Auf Basis dieses ERM wurden Klassen wie folgt erstellt:

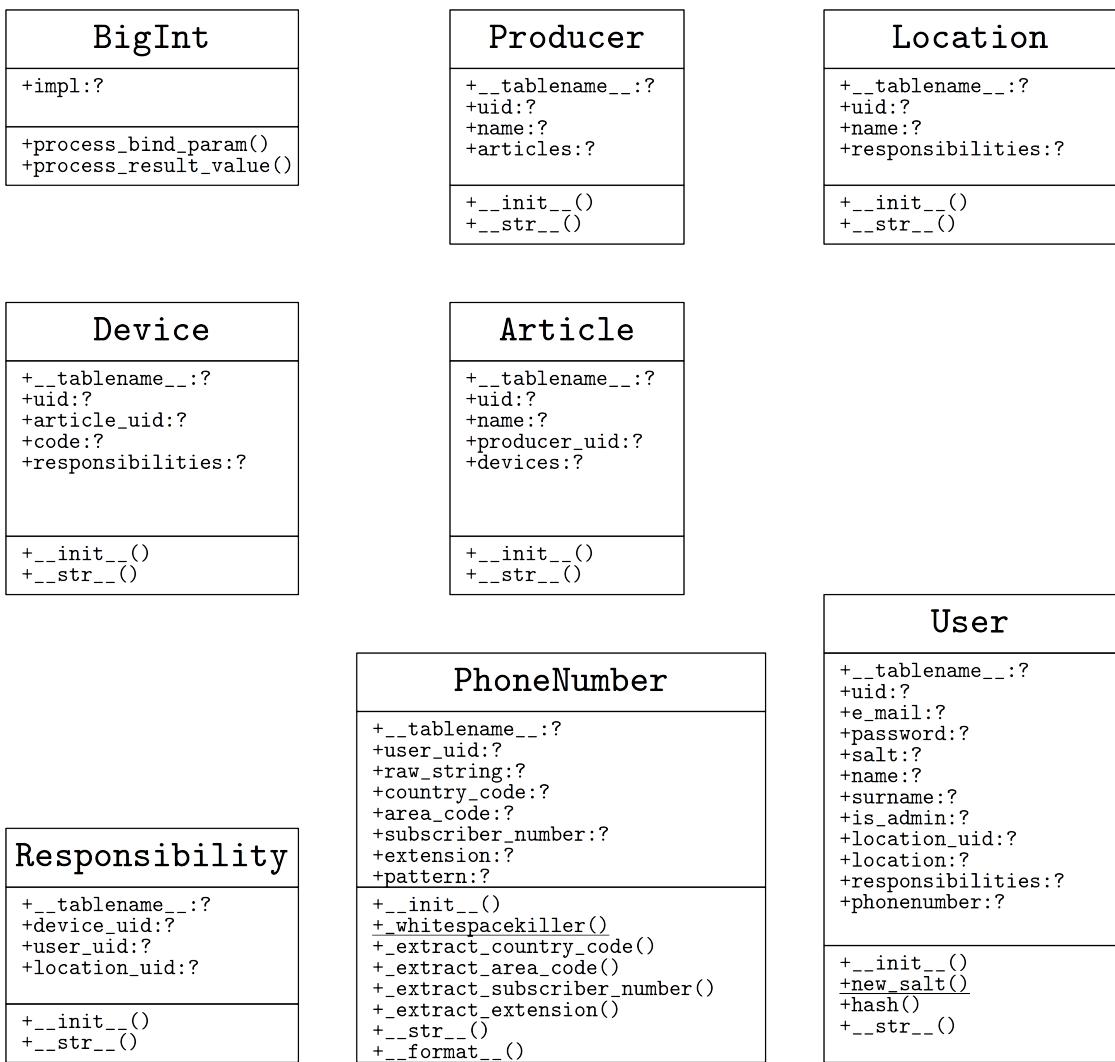


Abbildung 9.2: UML-Diagramm

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.

9.3 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.

9.4 Beziehungen herstellen in SQLAlchemy

9.4.1 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 3 Beispiel einer 1-zu-n-Beziehung in SQLAlchemy

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

class Child(Base):
    __tablename__ = "children"
    parent_uid = 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.

9.4.2 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 4 Beispiel einer 1-zu-1-Beziehung in SQLAlchemy

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

class Child(Base):
    __tablename__ = "children"
    parent_uid = 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.

9.5 Arbeiten mit SQLAlchemy

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

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."²⁴

9.6 *ContextSessions*

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.

Der Aufbau der Fabrikmethode ist in soweit non-trivial, dass in ihr eine Klasse definiert wird, in der wiederum weitere Klassen definiert werden, welche mitunter voneinander erben. Dieses vielleicht überkompliziert anmutende Konstrukt ist eigentlich nicht nötig, stellt aber eine sehr elegante Lösung dar und ist tatsächlich sehr einfach erklärt. Die erste innere Klasse wurde bereits erläutert und ist die, welche von der Funktion schließlich zurückgegeben wird. In dieser wird über den *sessionmaker* von SQLAlchemy eine Klasse *_Session* erzeugt - auf Instanzen dieser können bereits Datenbankoperationen durchgeführt werden, i.d.R. wird das bei SQLAlchemy auch so gemacht. Wir hingegen erben nun von dieser Klasse. Im Konstruktor der abgeleiteten Klasse delegieren wir die tatsächliche Initialisierung zum Konstruktor der Grundklasse und hinterlegen eine leere Liste, welche uns später als Cache dient. Des Weiteren redefinieren wir die Methode *add* der Grundklasse²⁵, in ihr wird nun jede übergebene Instanz zum vorher angelegten Cache hinzugefügt und anschließend der Funktionsaufruf wieder an die Basisklasse delegiert. Dieses Cache ermöglicht es der Session sobald ihr Kontext verlassen

²⁴[SQL18, 22.12.18 - 16:41 Uhr]

²⁵Tatsächlich werden dem gleichen Prinzip folgend, weitere Methoden redefiniert

wird, alle hinzugefügten Instanzen zu „refreshen“²⁶. Hierbei wird die Instanz mit der Datenbank synchronisiert - die meisten Objekte erhalten hier ihre UID von der Datenbank. Später wurde die innerste Klassendefinition dahingegeng geändert, dass der *sessionmaker* entfällt. Stattdessen wird von der *Session* Grundklasse geerbt und die Konfiguration selbst durchgeführt.

9.6.1 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 werden²⁷.

Algorithmus 5 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.

9.6.2 Abfragen

Abfragen können auf zwei verschiedenen Wegen erfolgen:

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

²⁶Dieser refresh wird mittels *map* durchgeführt. Eigentlich ist *map* eine Funktion, um eine Funktion auf alle Elemente eines iterierbaren Objektes anzuwenden und aus den Ergebnissen eine Liste zu erstellen. Hier wird die Ergebnisliste jedoch verworfen, *map* wird lediglich für die Seiteneffekte genutzt.

²⁷das „...“ im Code ist tatsächlich valider Code und wird in Python *Ellipsis* genannt. Es gibt Stellen an denen Ellipsen funktionalen Nutzen haben, hier dienen sie lediglich als Platzhalter.

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 6 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

    Args:
        e_mail (str): e_mail of the user that wants to log in
        password (str): user provided password to check against
    """
    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 \
                           user {e_mail}")
                return None
        except (AttributeError, ValueError) as e: #user not found \
            exception
            logger.info(f"Attempted login from unknown user {e_mail}")
            raise ValueError(f"Attempted login from unknown user {\
                             e_mail}")

```

Die wichtigen Punkte sind hier die Zeilen 14 und 22. In Zeile 14 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 22. 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 22 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, so dass 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.

9.6.3 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` bzw. `commit` aufgerufen wird.

10 Frameworkfreie Implementierung

Die Anwendung umfasst auch viele Funktionen und Klassen, die nicht allzu stark an SQLAlchemy oder PyQt gebunden sind - diese werden hier behandelt.

10.1 Utilities

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

10.1.1 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. Führt man 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).

10.1.2 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 gar nicht 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 ist²⁸), indem die Referenz von *_ParallelPrint* nicht mehr auf der Klasse, sondern auf einer Instanz dieser

²⁸ Ein Nutzer kann eine mit „_“-benannte Klasse/Methode/Instanz etc. zwar dennoch verwenden, sollte dabei allerdings die internen Mechanismen im Hinterkopf behalten und ist sogenannt im Fehlerfall auf sich alleine gestellt.

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 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.

Zu einem späteren Zeitpunkt kam ein weiterer Lösungsansatz auf:

Python erlaubt es beim Erzeugen einer Klasse einen Schlüsselwortparameter `metaclass` zu übergeben. Diese Metaklasse ist, wenn nicht anders definiert, stets `type`. Nun könnte man hier angreifen, eine eigene Metaklasse schreiben, bei der sich jede Klasse, die sie als Metaklasse nutzt, registriert und dadurch eine Mehrfachinstanziierung verhindern. Konkret sieht das dann wie folgt aus:

Algorithmus 7 Paralleles Printen mittels Metaprogrammierung

```
from collections import Counter
from queue import Queue
from threading import Thread

class Singleton(type):
    """Prevent a class from being instantiated more than once"""
    counter = Counter()
    def __call__(self):
        cls = super().__call__()
        self.counter.update((cls.__class__,))
        if self.counter[cls.__class__] > 2:
            raise ResourceWarning(
                f"{cls.__class__.__name__} should only be instantiated\\
                 once!")
        return cls

class _ParallelPrint(Thread, metaclass=Singleton):
    """Provides a threadsafe print, instantiation below"""
    print_ = Queue()

    def __new__(cls):
        instance = super().__new__(cls)
        setattr(cls, "__call__", cls.print_.put)
        return instance

    def __init__(self):
        super().__init__(name=f"{self.__class__.__name__}Thread")
        self.daemon = True

    @classmethod
    def run(cls):
        while True:
            val = cls.print_.get()
            print(val)
            cls.print_.task_done()

printer = _ParallelPrint()
printer.start()
printer("test")
```

Singleton ist eine Metaklasse, erbt daher von der Basismetaklasse *type*. In *Singleton* ist eine *Counter*-Instanz hinterlegt, die bei jeder Instanziierung einer Klasse, die *Singleton* als

Metaklasse nutzt, mitzählt, wie oft diese Klasse bereits instanziert wurde. In dieser konkreten Anwendung ist der Counter eigentlich überflüssig, er könnte genauso ein Dictionary mit boolschen Werten als *values*, oder einfach eine Liste der Klassen sein - dennoch ist er im Code, da die Counter-Klasse klar macht was sein Zweck ist und außerdem lässt sich die Klasse so leicht auf eventuelle Mehrfach-Instanziierungen erweitern. Bei `_ParallelPrint` entfallen in dieser Variante jegliche mehrfachinstanziierungsverhindernde Maßnahmen. Eine weitere Änderung ist, dass die magic-method `__new__` hinzugefügt wurde. `__new__` ist in Python der tatsächliche Konstruktor - auch wenn man bei `__init__` in der Regel vom Konstruktor spricht. In ihm wird der Klasse bei Instanziierung die magic-method `__call__` angehängt, welche dann die `put` Methode der Queue `print_` der Klasse ist - dadurch entfällt die externe Deklaration von `parallel_print`. Alternativ hätte hier eine explizite Deklaration von `__call__` mit Durchreichung an `put` erfolgen können.

10.1.3 Dateitools - Pfadvalidierung und Normalisierung von Dateinamen

10.2 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 10.1: 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.

10.3 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 vorbereitet²⁹. 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*³⁰, 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).

²⁹Eine genaue Erklärung einiger Prozesse des parallelen Programmierens erfolgt im Abschnitt 11.3

³⁰Eine Erklärung dieser Konstrukte von Raymond Hettinger, der sie entwickelt/implementiert hat, gab es u.a. auf der PyCon Russia 2016 [Het16, 20.12.18 - 23:03 Uhr]

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].

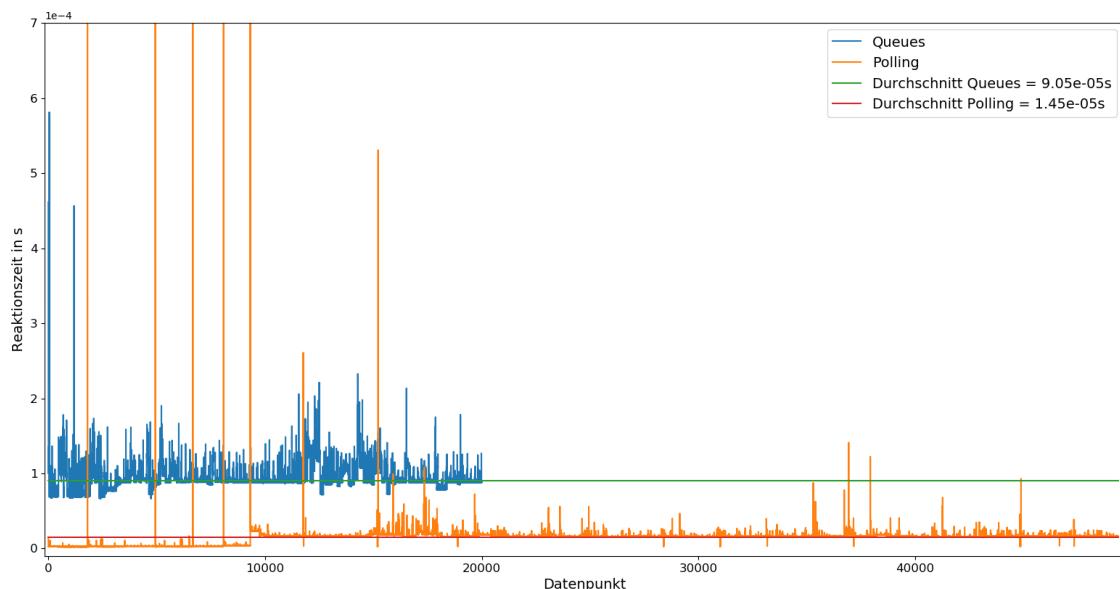


Abbildung 10.2: Gegenüberstellung von Polling und Queue

Wie in Abbildung 10.2 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.

Weiterhin stellt sich die Frage, wieso auch die Antworten über eine Queue geregelt sind, anstatt auf eine gelockte Property zu setzen: Die Request-Response-Variante hat den Vorteil, dass es nicht möglich ist, veraltete Daten abzufragen. Jeder, der ein Frame von der Kamera möchte, ist gezwungen vorher eine Abfrage zu starten - daher ist dieses grundsätzlich aktuell.

Dieser Mechanismus hätte, nach folgendem Schema auch in einer Property gekapselt werden können:

- bei *get-Zugriff* der Property wird intern eine Abfrage in der Request-Queue abgelegt
- blockierende Methode *get* der Response-Queue wird aufgerufen
- Daten werden zurückgegeben

Hierauf wurde jedoch verzichtet, da man bei manuellem Zugriff auf die Queues die Laufzeit der Bildabfrage mit einkalkulieren kann. Zum Beispiel weiß man an Punkt A, dass man bald ein Frame benötigt, also fragt man es an. Danach erledigt man einige andere Arbeiten, bis man das Frame an Punkt B tatsächlich entgegennimmt und verarbeitet.

10.4 Auslesen von Barcodes und grundlegende Aufbereitung von Frames

Die Funktionalität des Barcode-Auslesens und eine erste Bearbeitung der Frames findet in der Methode `find_and_mark_barcodes` der `VideoStream`-Klassen statt. Der Methode wird ein Frame übergeben, auf dem dann operiert wird.

Grundsätzlich gibt es zwei gängige Formate, in denen Bilder im Speicher dargestellt werden:

$$A_1 = \begin{pmatrix} \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \cdots & r_{mn} \end{bmatrix} \\ \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{m1} & g_{m2} & \cdots & g_{mn} \end{bmatrix} \\ \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix} \end{pmatrix}$$

$$A_2 = \begin{pmatrix} \begin{pmatrix} r_{11} \\ g_{11} \\ b_{11} \end{pmatrix} & \begin{pmatrix} r_{12} \\ g_{12} \\ b_{12} \end{pmatrix} & \cdots & \begin{pmatrix} r_{1n} \\ g_{1n} \\ b_{1n} \end{pmatrix} \\ \begin{pmatrix} r_{21} \\ g_{21} \\ b_{21} \end{pmatrix} & \begin{pmatrix} r_{22} \\ g_{22} \\ b_{22} \end{pmatrix} & \cdots & \begin{pmatrix} r_{2n} \\ g_{2n} \\ b_{2n} \end{pmatrix} \\ \vdots & \vdots & \ddots & \vdots \\ \begin{pmatrix} r_{m1} \\ g_{m1} \\ b_{m1} \end{pmatrix} & \begin{pmatrix} r_{m2} \\ g_{m2} \\ b_{m2} \end{pmatrix} & \cdots & \begin{pmatrix} r_{mn} \\ g_{mn} \\ b_{mn} \end{pmatrix} \end{pmatrix}$$

A_1 ist dabei ein Vektor(zwecks Übersichtlichkeit - im Code ist es ein beliebiger Container) mit je einer Matrix pro Farbkanal. A_2 ist hingegen eine Matrix, welche dann einen Vektor je Farbkanal enthält.

Da einige Zeilen der Auswertung nontrivial sind, wird die Funktion hier im Detail durchgesprochen.

Algorithmus 8 Methode *find_and_mark_barcodes* der beiden VideoStream-Klassen

```
def find_and_mark_barcodes(self, frame):
    """Find barcodes in given frame
    Args:
        frame: Frame that barcodes are to be detected in
    Returns:
        Tuple of frame where barcodes are marked, list of all \
            found codes in frame
    """
    barcodes = pyzbar.decode(frame)
    found_codes = []
    for barcode in barcodes:
        barcode_information = (barcode.type, barcode.data.decode("\
            utf-8"))
        if barcode_information not in found_codes:
            found_codes.append(barcode_information)
        poly = barcode.polygon
        poly = np.asarray([(point.x, point.y) for point in poly])
        poly = poly.reshape((-1, 1, 2))
        cv2.polylines(frame, [poly], True, (0, 255, 0), 2)
        cv2.rectangle(frame, *self.rect_transformation(*barcode.\
            rect), (255, 0, 0), 2)
        x, y = barcode.rect[:2]
        cv2.putText(
            frame,
            "{}({})".format(*barcode_information),
            (x, y - 10),
            cv2.FONT_HERSHEY_PLAIN,
            1,
            (0, 0, 255),
            1)
    return frame, found_codes
```

Der Ablauf ist wie folgt:

1. Über pyzbar, eine Python Bibliothek, die es ermöglicht Barcodes aus Bildern auszulesen, werden alle Barcodes aus dem Frame ausgelesen.
2. Es wird eine leere Liste *found_codes* angelegt und über alle Barcodes iteriert
 - (a) Es wird ein Tupel angelegt, dass den Barcode Typ (z.B. EAN-14 oder QR-Code) und die enthaltenen Daten mit UTF-8 codiert enthält.
 - (b) Nun wird überprüft, ob dieses Tupel bereits in *found_codes* hinterlegt ist, wenn nicht wird es hinzugefügt. Grundsätzlich gäbe es hier zwei Alternativlösungen. Man könnte entweder nicht überprüfen, ob das Tupel bereits enthalten ist, und *found_codes*

am Ende in ein *set* konvertieren, oder eine *set-comprehension* einsetzen um *found_codes* zu erzeugen. Sets sind dabei Python Container, die hashbare Instanzen in nicht geordneter Reihenfolge speichern. Sie sind soz. ein Dictionary, dass nur aus Schlüsseln besteht.

- (c) ZBar gibt *Barcode* Objekte zurück, die nebst dem tatsächlichen Payload des Codes, auch Metainformationen wie z.B. Geometrien um den Code enthalten. Hier setzen wir den *polygon* Member ein, um jene Punkte (Pixel) zu erhalten, die ein Polygon um den Code im Bild bilden.
- (d) Wir bilden uns nun eine Liste, die Tupel mit allen Koordinatenpaaren dieser Punkte enthält und setzen diese als Parameter für die Funktion *np.asarray* ein. Dabei ist *np* die Bibliothek *numpy* - eine hochoptimierte Bibliothek, die vor allem auf große Vektor- und Matrixoperationen - also lineare Algebra - ausgerichtet ist. *asarray* bildet nun aus der Liste aus Tupeln eine neue Struktur - das Array (tatsächlich ist es ein *ndarray* - ein n-dimensionales Array also). Diese Arrays, wie auch praktisch alle Funktionen auf ihnen, sind durch NumPy direkt in C³¹ implementiert.
- (e) Nun folgt ein Aufruf der *reshape* Methode. Diese ermöglicht es *arrays* effizient umzuformen (z.B. würde ein *a.reshape(2 * len(a))* aus einem Array wie unserem, ein 1-dimensionales Array bilden). Über die -1 im übergebenen Tupel, teilen wir NumPy mit, dass es uns egal ist, was es mit dieser Dimension macht, uns interessieren lediglich die anderen beiden, welche wir dann auf 1 bzw. 2 setzen.
- (f) Die folgende Zeile zeichnet nun, dem Array *poly* entsprechend ein Polygon auf unser Frame. True definiert das Polygon hierbei als geschlossenes Polygon, das Tupel (0,255,0), ein RGB Triplet, das unsere Farbe festlegt und die 2 ist die Dicke der zu zeichnenden Linien.
- (g) Analog dazu wird noch ein Rechteck um den Code gezeichnet. Ein C-Programmierer würde die dabei vorkommenden Asteriske eventuell als Pointer deuten, tatsächlich handelt es sich jedoch um doppeltes Tupel-Unpacking. Zuerst rufen wir die statische Methode *rect_transformation* unserer Klasse auf. Diese führt eine Koordinatentransformation des *rec*³²-Members des Barcode Obektes durch. Hierbei werden die Koordinaten der Form „Ursprung und Größe“ in zwei absolute Eckpunktskoordinaten gewandelt. Das dabei zurückgegebene Tupel wird nun direkt wieder unpacked und kann gezeichnet werden.

³¹Mitunter werden in numpy auch noch andere Sprachen wie z.B. Fortran eingesetzt

³²Von Englisch „rectangle“ zu deutsch „Rechteck“

- (h) Die anschließende Zeile ist eine sog. *slice* Operation, bei der wir die ersten beiden Elemente des *rect* Members nehmen, unpacken und in Variablen ablegen.
 - (i) Mithilfe dieser Variablen und einem Offset wird nun ein Text auf das Frame gezeichnet. Dieser Text enthält den Typ des Codes, sowie seine Daten. Dies ist eine der wenigen, wenn nicht sogar die einzige Stelle, im gesamten Programm, an der zur String-Interpolation kein f-String, sondern die alte *str.format* Methode eingesetzt wird. Diese kann nämlich über Unpacking komfortabel Tuple verarbeiten - mit einem f-String hätte jedes Element einzeln angesprochen werden müssen.
3. Sind alle Barcodes verarbeitet, werden das bearbeitete Frame und alle gefundenen Codes zurückgegeben.

10.5 Weitere Kapselung: VideostreamUISync

Zwischen *VideoStream* und UI sitzt ein weiterer Abstraktionslayer. Dieser kümmert sich darum, die Bilddaten auszuwerten, aufzubereiten und in der UI darzustellen. Konkret handelt es sich hierbei um einen weiteren Daemon-Thread, der bei seiner Instanziierung den *LazyVideoStream* von dem er seine Frames bekommt, sowie ein *canvas* übergeben bekommt. Das *canvas* ist ein beliebiges Qt Widget, das die Methode *setPixmap* implementiert. Tatsächlich verwendet Qt nämlich nicht einfach Matrizen zur Bilddarstellung, sondern implementiert hierzu eine eigene Klasse *QPixmap*. Der Ablauf einer Schleifeniteration sieht wie folgt aus:

1. Anfrage eines Frames bei VideoStream
2. Abrufen der Antwort - Aufteilung in Bilddaten und Barcodes
3. Aufruf der Methode *_matrice_to_QPixmap*
 - (a) Farbkorrektur über Wandlung von BGR zu RGB mittels *cvtColor* Methode von OpenCV - dies ist grundsätzlich eine sehr simple Aufgabe, es werden je nach Aufbau der Matrix lediglich zwei Arrays getauscht. Der Einsatz der Methode macht jedoch klar, was hier passiert.
 - (b) Ermitteln der Dimensionen der Matrix
 - (c) Konstruktion eines *QImage*; ein für Qt nötiges Zwischenkonstrukt. Hier werden alle Dimensionen und der gewünschte Farbraum (RGB mit 3x8bit) festgelegt.
 - (d) Konstruktion und Rückgabe einer *QPixmap*
4. Schreiben des Frames in die UI

5. Sofern codes gefunden wurden, wird die Methode *update* des *Counters barcodes* aufgerufen, falls ein Barcode als sicher erkannt eingestuft wird, wird ein Qt Signal emmitiert, welches die UI entgegennimmt und verarbeitet.
6. Kurze Verzögerung, die OpenCV benötigt, um korrekt zu funktionieren.

In Schritt 5 wird ein Counter angesprochen. Counter sind eine Datenstruktur des Python Standardmoduls *collections*, das spezielle, high-performance Containertypen zur Verfügung stellt³³. Konkret ist ein Counter eine *HashMap*, also ein Python dictionary, das hashbare Objekte und ihre Häufigkeit in sich oder einer anderen Datenstruktur, wie einer Liste speichert. Dieser Counter ermöglicht es, Fehler der Barcodeerkennung (diese sind extrem selten, kommen jedoch vor) auszugleichen. Ein Barcode wird erst als korrekt erkannt anerkannt, wenn er in einer bestimmten Anzahl an Frames erkannt wurde. Der Thread implementiert außerdem Methoden, um den Counter zurückzusetzen und das häufigste Element abzufragen.

Nach dem Verknüpfen mit der UI zeichnet sich ab, dass diese Lösung nicht funktioniert. Das Emittieren eines Qt Signals scheint, entgegen den Erwartungen, eine sehr teure (im Bezug auf Laufzeit) Operation zu sein - so läuft das Bild perfekt flüssig, bis man ein Signal einfügt, dann ruckelt es merklich; nämlich immer dann, wenn ein Signal emittiert wird. Außerdem stürzt die Anwendung ohne jegliche Fehlermeldung ab, wenn zu viele Signale in kurzer Zeit emittiert werden. Die Tatsache, dass hier keine Python Exception geworfen wird, aber auch der Hauptthread nicht korrekt „durchläuft“, sondern einfach schließt, lässt auf einen Fehler in der C/C++ Ebene von Qt schließen. Ein grundsätzlicher Workaround wäre, einen *sleep* von 1-2s nachdem ein Signal emittiert wurde, einzubauen oder die Sensitivität herunter zu stellen (also höherer Wert). Da in der Anwendung i.d.R. nicht nötig ist, innerhalb weniger Sekunden viele Codes einzuscanen, wäre das auch erstmal akzeptabel und funktionell - jedoch nicht sonderlich elegant, da das Bild in dieser Zeit immer stillstehen würde.

Dank des modularen Designs bieten sich einige bessere Möglichkeiten, um dieses Problem zu beheben:

- Weiterer Thread, sodass ein Thread nur Bilddaten verarbeitet und ein weiterer nur Barcodes
- Die Qt *QApplication* auf eine nicht-blockierende Art starten und eine eigene Eventloop für die UI implementieren, um eine performantere Kommunikation zwischen UI und Sync selbst zu implementieren.

³³[Pyt19a, 25.02.19 - 19:11 Uhr]

Umgesetzt wird die erste Variante. Dies ist mit nur geringfügigen Änderungen (weiteres Attribut der *VideostreamUISync* Klasse, zwei Verzweigungen und weitere Instanziierung der Synchronisation) möglich und erzielt das gewünschte Ergebnis.

10.6 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 9 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
```

Dekoratoren sind Konstrukte der Metaprogrammierung, die aus dem Programmierparadigma der funktionalen Programmierung stammen. Ihre Daten sind andere Funktionen, Klassen etc.³⁴.

Ein Function-Decorator ist so aufgebaut, dass er mit der @-Syntax, die ihm nachgestellte Funktion³⁵ als ersten positionellen Parameter übergeben bekommt und diese mit seiner Rück-

³⁴[Ste17, 16.01.19 - 17:00 Uhr] und [Gos19, 16.01.19 - 17:04 Uhr]

³⁵Hier sei gesagt, dass Function-Decorator auch auf anderen aufrufbaren Objekten arbeiten können und selbst durch verschiedene aufrufbare Objekte (*callable* im Python Jargon) dargestellt werden können (z.B. eine Klasse die *__call__* implementiert). Außerdem müssen Dekoratoren nicht zwangsläufig Funktionen zurückgeben - sie unterliegen keinerlei Einschränkungen im Bezug auf Parameter und Rückgabewerten. Jedoch sind sie in der Form

gabe 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, fällt das *decorated*-Flag auf seinen Standardwert *False* zurück, 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 implizit die dekorierte Funktion übergeben bekommt. Des Weiteren 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 weiterverarbeitet werden, bevor sie an den ursprünglichen Funktionsaufruf zurückgegeben wird.

In einer weiteren Verbesserungsiteration konnte die Funktion immens vereinfacht werden:

Funktion-dekoriert-Funktion-und-gibt-Funktion-zurück am häufigsten anzutreffen.

Algorithmus 10 Verbesserte Definition des Function-Decorators

```
def synchronized(function):
    """Function-decorator to automatically add the instance a function\
    returns to DB"""
    def synchronized_function(*args, **kwargs):
        instance = function(*args, **kwargs)
        save_to_db(instance)
        return instance
    return synchronized_function
```

Der Verzicht auf den rekursiven, partiell evaluierten Aufruf des Dekorators, welcher durch die Definition einer Funktion im Funktionskörper des Dekorators ersetzt wird, macht den Code wesentlich klarer. Die grundsätzliche Funktionsweise bleibt abseits dessen jedoch gleich.

11 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 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 gefüllt wird. Dies könnte, wenn nicht richtig abgefangen, zur SQL-Injection genutzt werden.

11.1 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:

Name	Typ	Schema
- Tabellen (1)		
- users		CREATE TABLE users(uid int AUTO_INCREMENT, name varchar(255), PRIMARY KEY (uid))
uid	int AUTO_INCREMENT	`uid` int AUTO_INCREMENT
name	varchar (255)	`name` varchar (255)
- Indizes (0)		
- Ansichten (0)		
- Trigger (0)		

Abbildung 11.1: Beispieldatenbank

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

Algorithmus 11 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.

11.2 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 umgehen³⁶. 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, eine Menge an Eingangsdaten auf eine begrenzte Menge an Ausgangsdaten abbildet und 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. Sogenannte Hash-Kollisionen, also die Tatsache, dass mehrere Werte der Eingabemenge auf die gleichen Elemente der Ausgabemenge gemappt werden, bewirken außerdem, dass die Funktion selbst wenn man eine Umkehrfunktion ermitteln kann, diese nicht eindeutig ist.

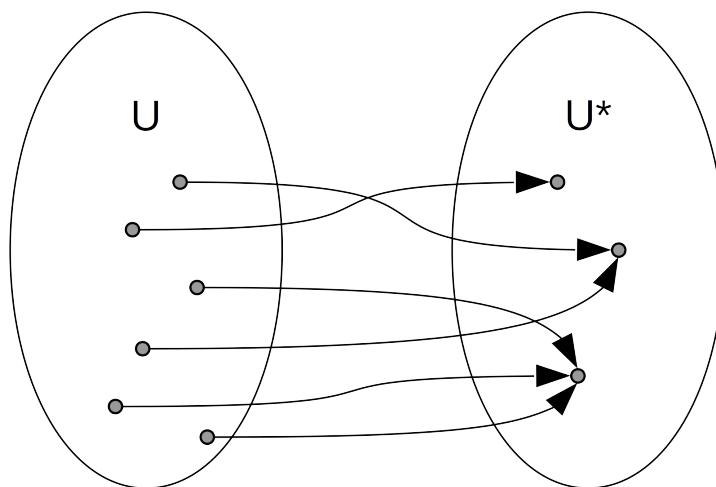


Abbildung 11.2: Prinzip von Hashfunktionen

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.

11.2.1 Benutzer anlegen

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

³⁶Auf Linux lässt sich das Ganze wie in Abschnitt 15 dargestellt sichern.

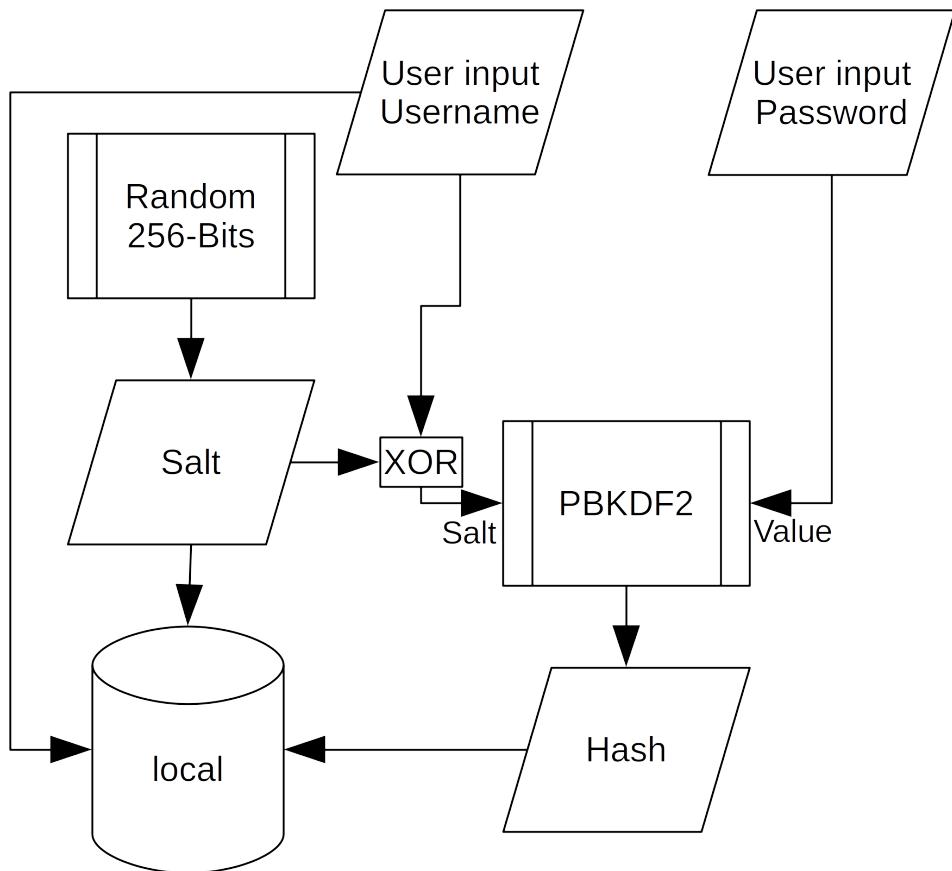


Abbildung 11.3: Passwort-Speicher-Vorgang

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 bitweises 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 beträgt die Wahrscheinlichkeit $\frac{5}{2^{256}} \approx 4,32 \cdot 10^{-75}\%$) 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 Username und Zufallszahl hat man somit ein für jeden Nutzer garantierter einzigartiges Salt. Ein Angreifer kann dadurch eine Bruteforce Attacke nur für einen einzigen Nutzer auf einmal ausführen.

Geplant war außerdem, dass der Eingangsbereich des *salt*s von 2^{256} Werten erhöht wird. Dies ist theoretisch der Fall, jedoch ist die E-Mail-Adresse in der Regel nicht groß genug, um dieses Phänomen ausnutzen zu können.

Betrachten wir zum Beispiel den *bytestring* `e_mail = b"karl@googlemail.com"` und den zugehörigen Integerwert `x = int.from_bytes(e_mail, "big")` erhalten wir nach der Formel $f(x) = \lfloor \log_2 x \rfloor + 1$ einen Wert von 151 Bit. Da der Ausgangsbereich eines XOR immer der, des größeren Operanden ist, wächst er daher nicht an.

Im Zuge des Refactoring, wird hier nachgebessert. So wird der PBKDF2-HMAC Algorithmus mit dem aktuelleren Argon2 ersetzt. Außerdem lassen sich bessere Ergebnisse erzielen, indem man Salt und E-Mail-Adresse nicht mit einem XOR verknüpft. Stattdessen werden beide Werte als Strings aneinandergehängt³⁷. Dies hat den Vorteil, dass längere Salts entstehen. Die Chance einer Salt-Kollision sinkt damit je nach Passwortlänge weiter.

³⁷Es wurde zunächst eine kompliziertere Variante mit einer Reißverschlussverknüpfung beider Sequenzen und Zufallszahlen implementiert, jedoch wurde diese schnell wieder verworfen da der Code unverhältnismäßig komplex war.

Die Parametrierung des Argon2 erfolgt mithilfe der empfohlenen Parameter im zum Algorithmus gehörenden PDF³⁸, bzw. der Parametrierungshilfe. Demnach wurden die Parameter wie folgt festgelegt:

- Als Typ y wird Argon2i gewählt.
- Als maximale Speichermenge m in Kibibyte werden 1024KiB festgelegt.
- Als maximale Zeit x in Sekunden werden den vorgeschlagenen Parametern der zugehörigen Wikipedia-Seite³⁹ für Server-Authentifizierung folgend, 0.5s gewählt.
- Als Salt-Länge sind laut PDF 128 Bit ausreichend - da das Grund-Salt tendentiell durch die E-Mail Adresse noch einmal verlängert wird, wählen wir daher 128 Bit als Grundlänge.
- Als Schlüssellänge werden ebenfalls 128 Bit als ausreichend angegeben - wir wählen 256 Bit.
- Die Auswahl des Parallelisierungsgrades bzw. der maximalen Anzahl Threads h und der Anzahl an Iterationen t ist ohne Informationen über das endgültige Zielsystem schwer festzulegen. Da davon ausgegangen wird, dass das Zielsystem etwas langsamer als das Testsystem ist, wurde $h = 4$ gewählt und x auf 0.4s nachkorrigiert.
- Mit den gegebenen Parametern ergibt sich $t \approx 512$

11.2.2 Benutzer anmelden

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

³⁸[UBDK17, 09.03.19 - 00:57 Uhr]

³⁹[Wik19a, 09.03.19 - 00:57 Uhr]

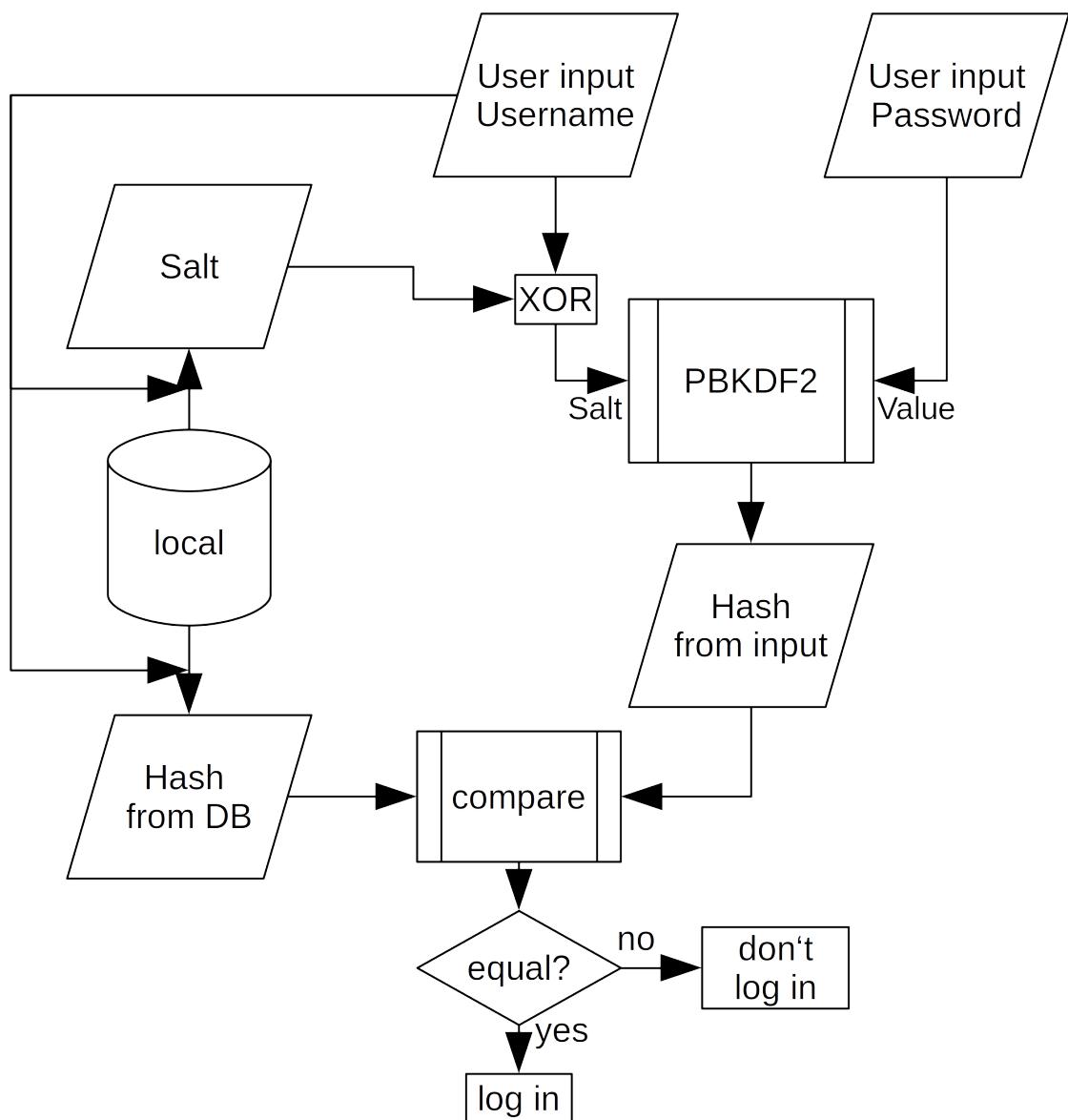


Abbildung 11.4: Anmelde-Vorgang

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.

11.2.3 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:

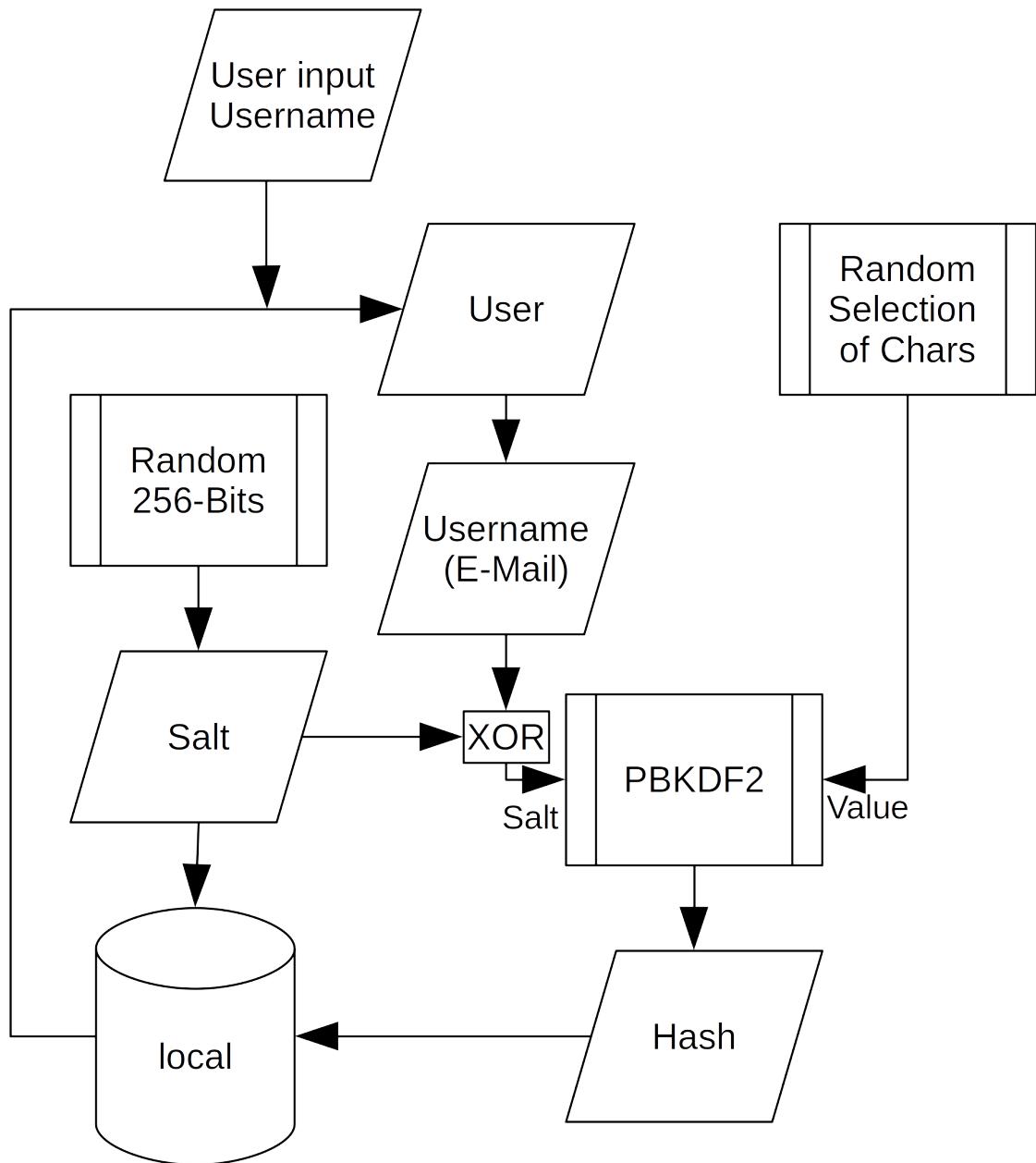


Abbildung 11.5: Rücksetzen eines Benutzerpassworts

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 un-

terscheiden 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.

11.2.4 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

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

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

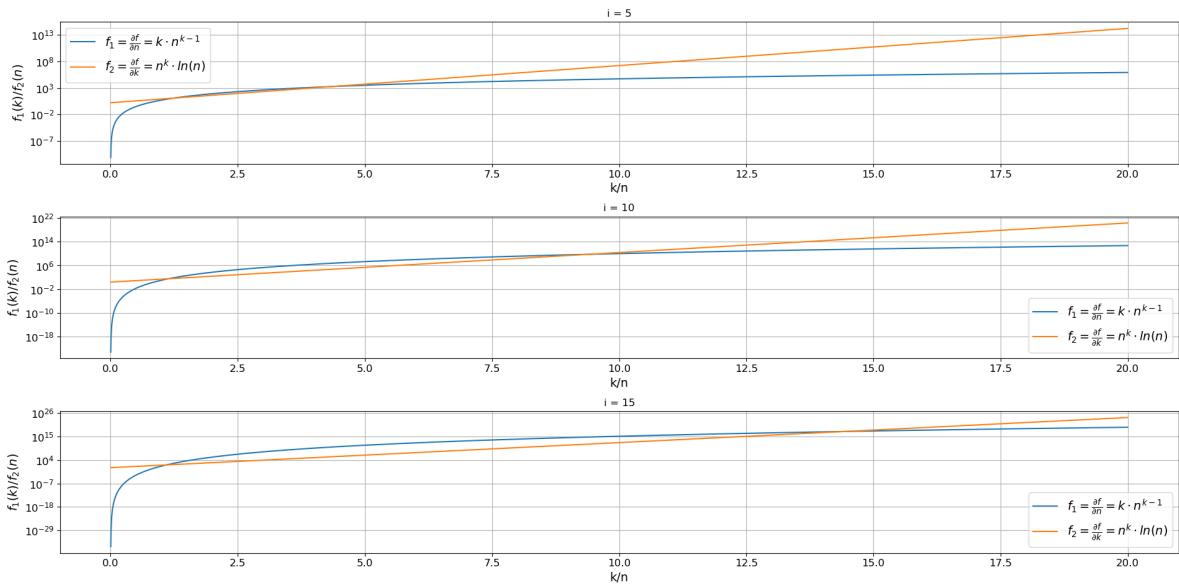


Abbildung 11.6: Vergleich von f_1 und f_2

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:

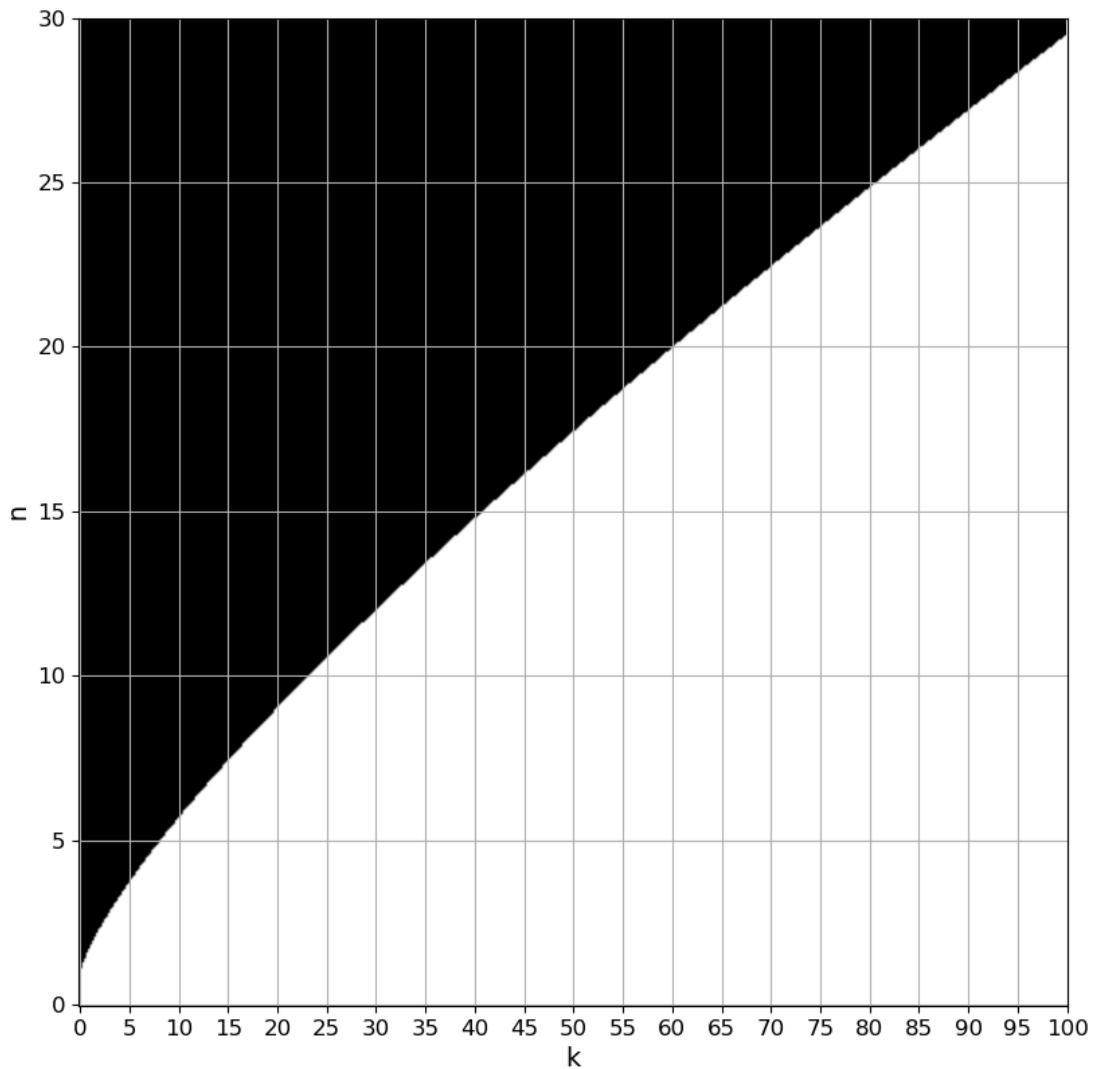


Abbildung 11.7: Heatmap aus f_1 und f_2

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

$$k \cdot n^{k-1} - n^k \cdot \ln(n) = 0$$

mit Subtraktion von $n^k \cdot \ln(n)$ folgt

$$n^k \cdot \ln(n) = k \cdot n^{k-1}$$

Division durch n^{k-1} liefert

$$\begin{aligned} \frac{n^k}{n^{k-1}} \cdot \ln(n) &= k \\ n^{k-(k-1)} \cdot \ln(n) &= k \\ k &= n \cdot \ln(n) \end{aligned}$$

Was letztendlich unser Ergebnis darstellt

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

und ein einfacheres Überprüfen der Koordinaten zulässt.

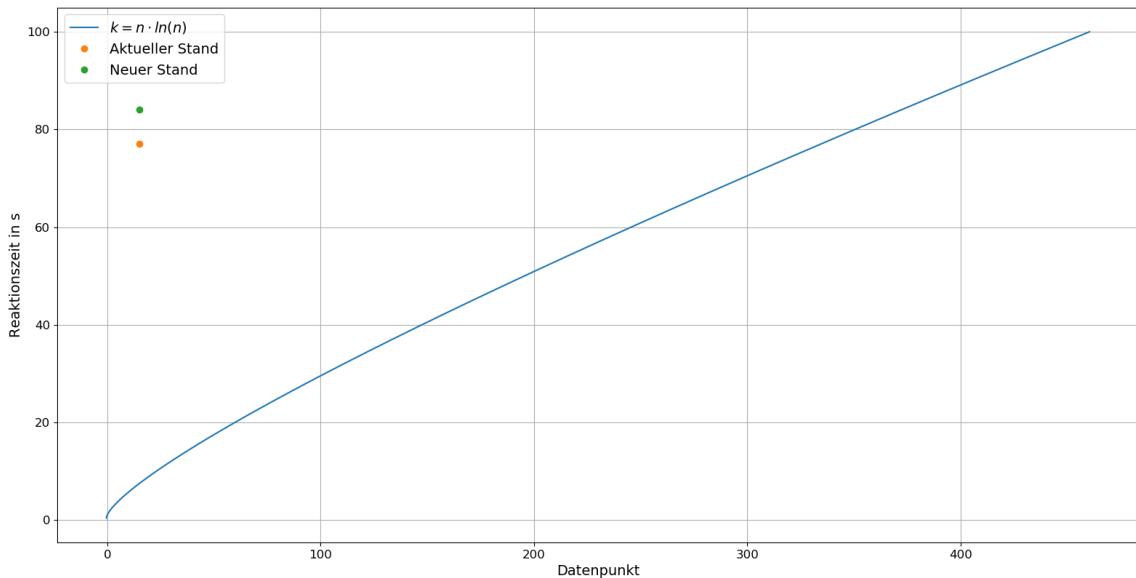


Abbildung 11.8: $k(n)$ aus f_3

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.

11.2.5 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.

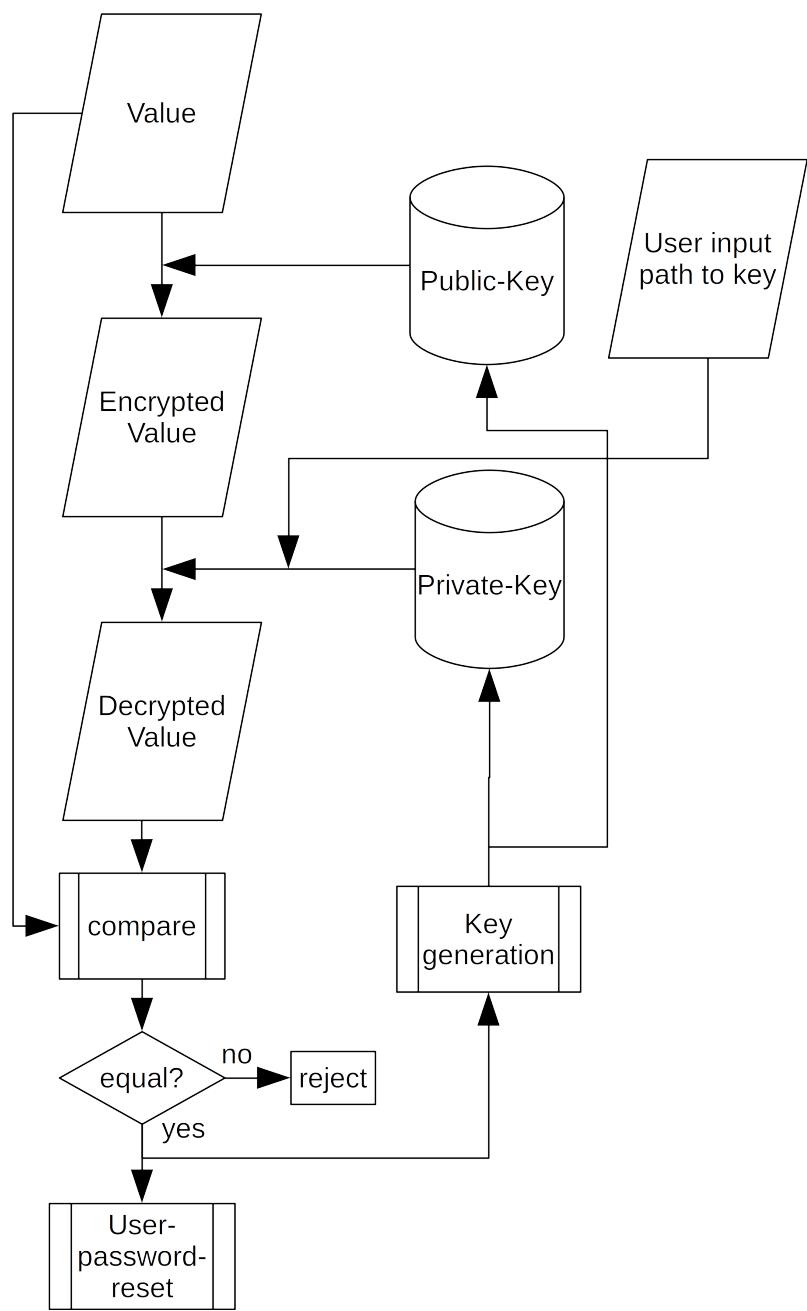


Abbildung 11.9: Verifizierung zum Rücksetzen eines Admin-Passworts

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.

11.3 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 prüft, ob die eingesetzte 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 beliebig viele positionelle bzw. Schlüsselwort-Argumente. 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. Des Weiteren ist hier der Vergleich auf *None* zu beachten. *None* ist in Python ein Singleton, das stets

mit *is* *anstatt* *mit ==* verglichen werden sollte. In einer späteren Revision wurde die Klasse dahingehend abgeändert, dass sie auch Keyword-Arguments unterstützt.

Die Funktion, die ausgeführt wird, ist eine sog. *Lambda Funktion/Lambda Expression*. Lambda Expressions sind anonyme Funktionen, die vorwiegend eingesetzt werden, wenn der Umfang einer Funktion zu gering ist um eine Funktionsdefinition mit *def* zu rechtfertigen oder die Funktionsdefinition *inline* geschehen soll. Die eingesetzte Funktion ist eine Funktion, welche ein *Signal* übergeben bekommt und dieses mit dem Parameter *True* emittiert. Dieses Signal ist ein Qt-Signal, welches soz. von einem *Slot* der UI abgefangen wird. Ein direkter Aufruf einer Methode der UI ist nicht möglich, da Qt nicht *threadsafe* aufgebaut ist. Das Prinzip der Slots und Signale wird im Abschnitt über PyQt näher erläutert.

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* kommen⁴⁰. 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 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* oder *pipes*, 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

⁴⁰[EK17, S. 565]

können. Da dies jedoch weder weiteren Nutzen noch bessere Performance bringt, wurde darauf verzichtet.

11.4 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)⁴¹. 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 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.

⁴¹[Wik18b, 22.12.18 - 18:02 Uhr]

Eine Regex ist über die runden Klammern in Gruppen unterteilt, die jeweils eine Aufgabe übernehmen. 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 das 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 ODER zwischen Gruppen oder Ausdrücken dar und erlaubt damit Alternativen

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

Gruppe_7

- \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_7

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

/Gruppe_6 bisher lässt sich feststellen, ob eine Vorwahl gegeben wurde und ob eine Rufnummer vorhanden ist - die umschließende Gruppe 6 matched alle Gruppe 7 Iterationen

Gruppe_8

- 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_8

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

/Pattern

Mittels `re.compile`, mit gesetztem `re.DEBUG` Flag, lässt sich außerdem folgende, die Regex beschreibende Ausgabe erzeugen:

Algorithmus 12 Kompilierung und Debugausgabe der Regular Expression zur Telefonnummernerkennung

```
>>> import re
>>> pattern = re.compile(r"(((\+\d{1,3})|(0))\?|([1-9]+\?)?((\d+\?)|+)\?
(-\d+)?", re.DEBUG)
MAX_REPEAT 0 1
SUBPATTERN 1 0 0
SUBPATTERN 2 0 0
BRANCH
    SUBPATTERN 3 0 0
        LITERAL 43
        MAX_REPEAT 1 3
        IN
            CATEGORY CATEGORY_DIGIT
OR
    SUBPATTERN 4 0 0
        LITERAL 48
MAX_REPEAT 0 1
        LITERAL 32
SUBPATTERN 5 0 0
    MAX_REPEAT 1 MAXREPEAT
    IN
        RANGE (49, 57)
        LITERAL 32
SUBPATTERN 6 0 0
    MAX_REPEAT 1 MAXREPEAT
SUBPATTERN 7 0 0
    MAX_REPEAT 1 MAXREPEAT
    IN
        CATEGORY CATEGORY_DIGIT
    MAX_REPEAT 0 1
        LITERAL 32
MAX_REPEAT 0 1
SUBPATTERN 8 0 0
    LITERAL 45
    MAX_REPEAT 1 MAXREPEAT
    IN
        CATEGORY CATEGORY_DIGIT
```

Auf diese Ausgabe sei hier nicht näher eingegangen. Im Code selbst findet sich die Kompilierung nicht wieder. Sie ist nicht nötig, da die Funktionen von *re* so implementiert sind, dass sie die übergebenen patterns intern kompilieren und cachen. Wenn nicht sehr viele verschiedene patterns genutzt werden, kann daher auf dieses Cache zurückgegriffen werden. Dies hat den Vorteil, dass die Kompilierung nur stattfindet, wenn sie tatsächlich nötig ist und außerdem pro Session maximal einmal.

Dieses Pattern ist als Klassenvariable der Klasse TelephoneNumber hinterlegt. Wird nun TelephoneNumber instanziert, so wird über `re.search`⁴² 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 8 für Vorwahl, Rufnummer und Durchwahl ausgewertet.

Nach dieser grundsätzlich funktionellen Implementierung, wird die komplette Regex noch mit Hinblick auf bessere Les- und Wartbarkeit abgeändert. Die resultierende Regex sieht wie folgt aus:

```
r"(?P<prelude>(?P<country_code>(?:\+\d{1,3})|(?:0)) *(?P<area_code>(?:[1-9])+) (?P<subscriber_number>(?:d+ ?)+)(?P<extension>[-+]\d+)?"
```

Die zu Grunde liegende Funktion bleibt die Gleiche, jedoch gibt es einige entscheidende Vorteile:

1. Untergruppen, die selbst nicht angesprochen werden müssen, werden mit `(?...)` unterdrückt (macht sich im Kompilat durch `Nones` an Subpatterns bemerkbar)
2. Einzelbestandteile/Gruppen werden mit `(?P<name>...)` benannt. Sollte zu einem späteren Zeitpunkt eine Änderung, die Gruppen hinzufügt oder entfernt erfolgen, so werden die einzelnen Teile weiterhin korrekt erkannt (z.B. war die Durchwahl in der ursprünglichen Regex unter Gruppe 8 zu finden). Python erlaubt die Abfrage von benannten Gruppen über dieselbe Schnittstelle (`group`-Methode einer Match-Instanz) - für benannte Gruppen wird einfach die Gruppenbezeichnung übergeben.

⁴²Alternativ könnte man hier `re.match` nutzen. Der Unterschied ist, dass `re.search` den gesamten String durchsucht, während `re.match` lediglich den Anfang betrachtet.[Pyt19b, 01.03.19 - 19:32 Uhr]

Die Gruppen sind also:

prelude umfasst der Rufnummer vorgestellte Segmente

country_code umfasst Länderkennzahl

area_code umfasst Vorwahl

subscriber_number umfasst Rufnummer

extension umfasst Durchwahl

Des Weiteren 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 emittiert, 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 wird dennoch fehlerfrei erkannt.

12 PyQt5

[yk]

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 fähig ist und dabei auf allen Plattformen konsistent gut aussieht.

12.1 Signale und Slots

12.1.1 Signal

Im Gegensatz zu Kommandozeilen-Interfaces sind UI-Anwendungen immer ereignisgesteuert. Das heißt, der User drückt zum Beispiel einen Button, oder macht eine Eingabe in einem Textfeld und daraus entsteht dann ein bestimmtes Ereignis, z.B. „Button gedrückt“ oder „Text geändert“. Diese Ereignisse senden in PyQt sogenannte Signale, welche dann mit Slots verbunden werden müssen, um bei einem bestimmten Ereignis, eine bestimmte Funktion auszuführen.

Signale können auch mit mehreren Slots und ebenso können Slots mit mehreren Signalen verbunden werden. Dies kann von Vorteil sein, wenn zum Beispiel mehrere Buttons die selbe Funktion auslösen sollen. Signale können ebenfalls mit anderen Signalen verbunden werden, dies war in unserem Fall jedoch nicht nötig und wird hier nicht weiter behandelt.

12.1.2 Slot

Ein Slot wird aufgerufen, wenn das an ihn verbundene Signal ausgelöst wird. Slots stellen normale Funktionen dar mit dem einzigen Unterschied, dass Signale an sie gebunden werden können.

12.1.3 Signale und Slots

Im folgenden finden findet sich ein Beispiel, wie ein Signal mit einem Slot verbunden wird.

Solche sogenannten Events enthalten folgende Informationen: Event-Quelle, Event-Objekt und Event-Ziel

Dies sieht dann wie folgt aus:

```
widget.signal.connect(slot)
```

ein Beispiel dafür wäre:

```
self.button.clicked.connect(button_click)
```

self.button ist die Event-Quelle

.clicked ist das Event-Objekt um das es hier geht

.connect sagt aus, dass beide Teile miteinander verbunden werden sollen (Man kann Signale auch wieder disconnecten)

button_click ist dann der Slot der bei diesem Event aufgerufen wird.

12.2 Darstellen der Verantwortlichkeiten als Baumstruktur

[sv]

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. Des Weiteren 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 TUIventory 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 *Responsibilities* 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* eingesortiert 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:

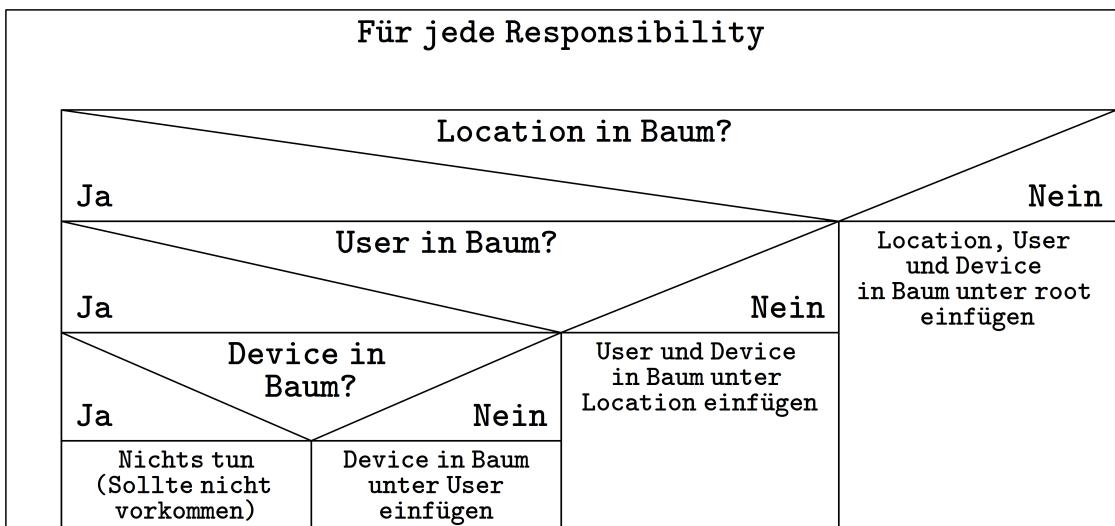


Abbildung 12.1: Struktogramm nach Nassi-Shneiderman zum Responsibility-Baum-Sieb

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 13 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 über 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.

Im Zuge des Refactoring wird auch hier nachgebessert. Der Filtervorgang wird von „Pure Python“ zu SQL verlegt, wodurch wir uns höhere Performanz versprechen. Wie bei allen anderen Abfragen, werden auch hier die Abfragen nicht über SQL direkt, sondern über die Objektorientierte Query-API von SQLAlchemy durchgeführt. Da der Baum vom Nutzer direkt einsehbar ist und die Abfrage-Texte vom Nutzer beeinflusst werden können, wäre an dieser Stelle die Gefahr für SQL Injection mittels Interpolation der Namen in die Abfragebefehle groß.

Dem geänderten Arbeitsprinzip nach, werden zunächst alle *Locations* aus der Datenbank abgefragt; hier findet auch direkt eine alphabetische Sortierung statt. Nun wird für jede Location überprüft, ob es Responsibilities in der Datenbank gibt, über einen *join* (ORM-Methode für

einen *Inner Join*) werden die dazugehörigen Nutzer abgefragt. Nachdem auch die Nutzer alphabetisch sortiert sind, wird über diese iteriert und über eine weitere Abfrage ermittelt, welche Geräte der aktuellen Location und dem aktuellen User zuzuordnen sind. Diese geänderte Version hat außerdem den Vorteil, dass weniger Interaktion mit PyQt nötig ist. Es werden lediglich die entsprechenden *Items* zu jeder Location, jedem User und jedem Gerät jeweils exakt 1 mal erstellt und an der korrekten Stelle in den Baum eingepflegt. Abfragen aus dem Baum entfallen vollständig.

12.3 Fazit

[sv]

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 eventuell geändert werden sollen.

13 Aufbau der UI

Da TUIventory keine Consumer-Software darstellt, die sich auch bei den Anwendern gut verkaufen muss, wurde beim Aufbau der UI auf eine schlichte und übersichtliche Struktur gesetzt.

Die Software wird, außer beim initialen Geräteanlegen, selten länger als 5-10min am Stück verwendet. Gerade auch aus diesem Grund, sollte die Software möglichst selbsterklärend aufgebaut sein. Um hier trotzdem niemand auf sich sitzen zu lassen werden alle Mitarbeiter, die diese Software später benutzen sollen, in der Bedienung unterwiesen. Zusätzlich wird auch eine kurze Bedienungsanleitung als PDF-Datei beigelegt. Alle diese Aufwendungen sollten jedoch kaum nötig sein, da das Programm ausschließlich von IT-Personal benutzt wird.

Die UI enthält ein Navigationsmenü auf der linken Seite, einen Home-Button/Logo an der oberen Seite und ein Login-Menü in der rechten oberen Ecke. Ebenfalls ist über der unteren Leiste eine Statusbar enthalten. Dies entspricht dem Standard-Layout, das User gewohnt sind.

Das Navigationsmenü lässt einen durch die unter 13.2 beschriebenen Menüpunkte switchen, beim Klicken auf den Home-Button wechselt die UI zum Übersichtsfenster.

13.1 Login-Menü

Beim klicken auf den Login-Button öffnet sich ein neues Fenster, in dem man seine Anmelde-daten, also Email und Passwort, eingeben kann. Wurden diese Angaben alle richtig gemacht, so schließt sich das Fenster wieder und im Hauptfenster sieht man oben rechts nun den Namen des eingeloggten Users, sowie einen Button zum ausloggen.

Da es durchaus vorkommen kann, das selbst der Admin sein Passwort verliert, wurde die Möglichkeit eingebaut, auch dieses Zurückzusetzen. Wie dies funktioniert ist unter 11.2.5 zu finden. Aufrufen lässt sich diese Funktion im Login-Fenster mit Hilfe des Buttons „Passwort vergessen“.

13.2 Menüpunkte

13.2.1 Übersicht:

Hier befindet sich eine Liste aller vorhandener Geräte, geordnet nach Orten, an denen sich die Geräte befinden. Diese Orte stellen Drop-Down-Menüs dar, welche sich aufklappen lassen, dort sind nun alle ansässigen Personen mit ihren jeweiligen Geräten zu finden. Sollte sich ein

QR-Code auf dem Gerät abgelöst haben, so kann hier mit Hilfe des Buttons „QR-Code generieren“ ein neuer QR-Code für jedes beliebige Gerät generiert werden. Hierfür muss vorher das entsprechende Gerät in der Übersicht ausgewählt werden. Der generierte Code wird, unter dem in den Einstellungen (13.2.5) festgelegten Pfad, abgelegt. Dies geschieht als SVG-Datei, kann jedoch bei Bedarf auch als zum Beispiel einer PNG- oder JPEG-Datei geschehen.

13.2.2 Kamera:

Unter diesem Unterpunkt kann der vorher angebrachte QR-Code des Geräts eingescannt werden. Unterhalb des Kamerabilds werden nun alle wichtigen Informationen angezeigt. Hier lässt sich ebenso der Ort ändern oder man kann es einer anderen Person zuweisen.

Sollte ein Gerät fest installiert sein, das heißt man kann es nicht in die Kamera halten, so kann der Code auch mit einem Handy abfotografiert werden und dieses Bild abgescannt werden.

13.2.3 Geräteverwaltung:

Hier sind drei Unterpunkte zu finden:

- Gerät:

Auf dieser Seite lassen sich neue Geräte inklusive Verantwortlichen und Ort anlegen. Beim Speichern wird automatisch ein QR-Code für das neue Gerät erzeugt und unter dem vorher ausgewählten Pfad abgelegt. Hierfür muss jedoch erst ein Artikel angelegt werden.

- Artikel:

Hier können Artikel erstellt werden. Es sollte ein eindeutig erkennbarer Name gewählt werden. Hierfür muss jedoch erst ein Hersteller angelegt werden.

- Hersteller:

Hier können Hersteller angelegt werden.

Ein einfacher „Speichern“-Button auf allen drei Seiten überprüft die Eingaben auf Vollständigkeit. Sollte dies gegeben sein, so wird das jeweilige Objekt erstellt und in die Datenbank abgespeichert.

13.2.4 Benutzerverwaltung:

Je nach Login-Status des Users gibt es hier unterschiedliche Funktionen:

- Kein User eingeloggt

Neue Benutzer können hier angelegt werden, solange man nicht eingeloggt ist. Die eingegebene Emailadresse wird später beim Einloggen als Benutzername verwendet.

- Nutzer eingeloggt

Sobald man eingeloggt ist, kann man hier seine angegebenen Daten auch wieder ändern. Hierfür sind die Felder dann bereits mit den entsprechenden Angaben vorausgefüllt.

- Admin eingeloggt

Sollten Sie als Admin eingeloggt sein, so lassen sich auch die Daten anderen Nutzer ändern oder diese ebenfalls zum Admin befördern. Die zu ändernde Person lässt sich über ein Drop-Down-Menü auswählen. Als Admin lässt sich hier ebenfalls ein neues Passwort für andere Nutzer generieren, sollte diese ihres vergessen haben.

Hierzu sei gesagt, dass nur Daten angegeben werden müssen, die innerhalb einer Firma sowieso öffentlich zugänglich sind, wie Name, Email und Telefonnummer. Beim Ändern werden ebenfalls nur diese Daten angezeigt, Passwörter bleiben verschlüsselt und werden nicht sichtbar.

13.2.5 Einstellungen:

Auf dieser Seite können die Standardeinstellungen geändert werden.

- Kamera spiegeln:

Beim Code einscannen ist es für die meisten User einfacher, wenn das Kamerabild gespiegelt wird, da sich so das Bild in die selbe Richtung wie auch das zu einscannende Bauteil bewegt. Sollte dies nicht gewünscht sein, so kann dies hier abgestellt werden.

- Time Out Delay:

Hier lässt sich einstellen, nach wie viel Minuten ohne aktive Nutzerinteraktion der User automatisch ausgeloggt wird. Aus Sicherheitsgründen kann hier minimal ein Wert von 2 Minuten eingestellt werden. Wird ein User nach Ablauf dieser Zeit ausgeloggt, so öffnet sich ein neues Fenster mit dem Grund für das Ausloggen, sodass der User weiß, warum er ausgeloggt wurde.

- QR-Code Ablage:

Die generierten QR-Codes werden unter diesem Pfad angelegt. Der Pfad kann auch händisch geändert werden, beim abspeichern der Einstellungen wird dieser, je nach Betriebssystem, auf die richtige Schreibweise überprüft.

13.3 Statusbar

Da es bei einer Software mit Benutzeroberfläche dem Benutzer immer wieder, unterschiedliche Informationen übermittelt werden müssen entschieden wir uns eine Statusbar einzufügen. Hier werden Nachrichten angezeigt, wie zum Beispiel beim Erkennen eines Barcodes in der Kamera „Barcode erkannt“. Die meisten Meldungen sind jedoch sozusagen Fehlermeldungen, z.B. wenn ein User beim Einloggen ein falsches Passwort eingibt, oder beim Benutzer anlegen nicht alle Felder ausgefüllt werden. Um bereits auf den ersten Blick klar zu machen, ob diese Meldung jetzt gut oder schlecht ist, wurden die unterschiedlichen Meldungen in unterschiedlichen Farben ausgeführt. Fehlermeldungen werden immer in rot angezeigt, einfache Information in grün.

Werden gerade keine Meldungen in der Statusbar angezeigt, so fällt ihr Vorhandensein nicht auf, da der Hintergrund transparent gehalten wurde. Durch die farbige Schrift fallen Meldungen trotzdem sofort auf.

14 Optimierung und Testing

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

14.1 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 optimieren⁴³, 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.

⁴³Außer durch ein anderes Event, wie z.B. Überprüfung, ob der Nutzer noch aktiv die UI bedient und Elemente anklickt

Algorithmus 14 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\x7f4898283798>,
 [...]
 'ui': <__main__.MainDialog object at 0x7f48a17f3948>,
 'videoFeed': <PyQt5.QtWidgets.QLabel object at 0x7f4898283948>}
"""
```

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 15 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 ist⁴⁴, was einen Geschwindigkeitsschub in Aussicht stellt. Dies wird jedoch dadurch relativiert, dass auch Dictionaries in C implementiert sind⁴⁵.

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

⁴⁴<https://github.com/python/cpython/blob/master/Python/bltinmodule.c>

⁴⁵<https://github.com/python/cpython/blob/master/Objects/dictobject.c>

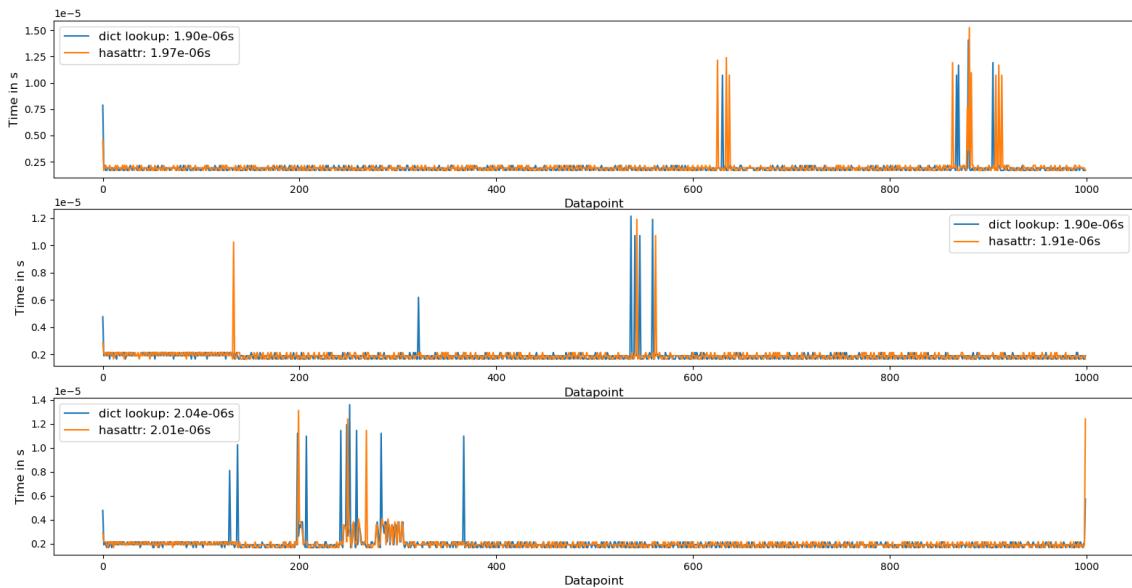


Abbildung 14.1: Vergleich von dict lookup und hasattr (jeweiliger Durchschnitt der Versuchsreihe in Legende)

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 16 Bytecode zu *dict-lookup* und *hasattr* erzeugen

```
1 import dis
2
3
4 class B():
5     def __init__(self):
6         self.status = False
7
8     def reset(self):
9         self.status = True
10
11
12 class A():
13     def __init__(self):
14         self.timeout = B()
15         for i in range(100):
16             setattr(self, f"{i}", i)
17
18     def dict_lookup_(self):
19         if "timeout" in self.__dict__:
20             self.timeout.reset()
21
22     def hasattr_(self):
23         if hasattr(self, "timeout"):
24             self.timeout.reset()
25
26
27 a = A()
28
29
30 def hasattr_():
31     if hasattr(a, "timeout"):
32         a.timeout.reset()
33
34
35 def dict_lookup_():
36     if "timeout" in a.__dict__:
37         a.timeout.reset()
38
39
40 print(f"{'hasattr':->50}")
41 print(dis.dis(hasattr_))
42 print("\n")
43 print(f"{'dict-lookup':->50}")
44 print(dis.dis(dict_lookup_))
```

Algorithmus 17 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)⁴⁶:

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

⁴⁶[Pyt18a, 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 werden⁴⁷, 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 letzten 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*'

⁴⁷Einige 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 an⁴⁸, 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.

14.2 ContextSession Cache Refresh

In der Magic Method `__exit__` der ContextSession Klasse, ist es erforderlich, auf jedes Objekt einer Liste eine Funktion anzuwenden. Klassischerweise würde man hier auf eine `for` Schleife zurückgreifen, doch Python bietet noch andere Lösungsmöglichkeiten. Grundsätzlich sagt man in Python, dass List Comprehensions oftmals schneller als for-Schleifen sind, da sie auf optimierte Bytecode Operationen zurückgreifen können. Jedoch bezieht sich das stets auf Anwendungen, in denen eine neue Liste erzeugt wird. Eine Anwendung wie unsere, bei der lediglich die Seiteneffekte der Schleife gewünscht sind, und keine neue Liste benötigt wird, wie eine List Comprehension sie erzeugt, sind jedoch nichts, was man in Python häufig antrifft, bzw. was i.d.R. mit einer Comprehension gelöst wird. Des Weiteren gibt es noch die Funktion `map`, welche ebenfalls auf jedes Element eines iterierbaren Objektes eine Funktion anwendet. Auch hier wurde ein Vergleich über eine einfache Grundfunktion nach dem bereits bekannten Verfahren durchgeführt. Die interessanten Funktionen sind *f_1*, *f_2* und *f_3*.

⁴⁸<https://github.com/python/cpython/blob/master/Python/ceval.c>, 24.12.18 - 21:14 Uhr

Algorithmus 18 Analysecode zum Laufzeitvergleich verschiedener Iterationsmöglichkeiten

```
from statistics import median
from string import ascii_letters
import tempfile
from time import time

import matplotlib.pyplot as plt

with tempfile.TemporaryFile("w+") as out:
    def printer(a):
        print(a, file=out)

    def f_1(a):
        for b in a:
            printer(b)

    def f_2(a):
        [printer(b) for b in a]

    def f_3(a):
        map(printer, a)

    ...
```

Im Beispiel wird hier für jedes Element einer Liste (die Liste beinhaltet alle ASCII Buchstaben) eine Funktion aufgerufen, die dieses Element printed. Dabei erfolgt das Printen jedoch nicht auf den Standardausgabestream *stdout*, den *print* regulär nutzt, sondern in eine temporäre Datei. Führt man diesen Code aus, ergibt sich folgende Ausgabe:

Algorithmus 19 Ausschnitt der Konsolenausgabe

```
Dataset #0
For loop           : 2.9039e-04s
List Comprehension: 2.9087e-04s
Map                : 1.4305e-06s
Fastest was: Map

Dataset #1
For loop           : 2.8825e-04s
List Comprehension: 2.8872e-04s
Map                : 1.1921e-06s
Fastest was: Map

Dataset #2
For loop           : 2.8992e-04s
List Comprehension: 2.8872e-04s
Map                : 1.1921e-06s
Fastest was: Map

Dataset #3
For loop           : 2.8801e-04s
List Comprehension: 2.8777e-04s
Map                : 1.1921e-06s
Fastest was: Map

Dataset #4
For loop           : 2.8849e-04s
List Comprehension: 2.8801e-04s
Map                : 1.1921e-06s
Fastest was: Map
```

...

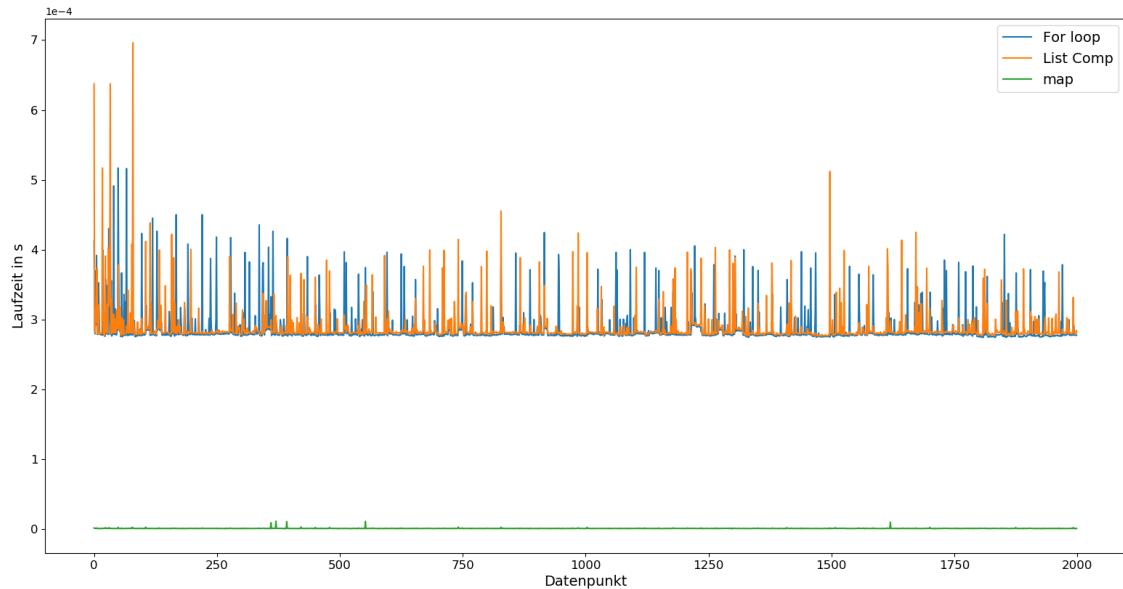


Abbildung 14.2: Visualisierung einer Testreihe

Die Konsolenausgabe stellt dabei eine bestimmte Anzahl an Datensätzen dar, in denen jeweils 2000 mal der Funktionsaufruf durchgeführt und der jeweilige Durchschnitt der verschiedenen Methoden bestimmt wird. Die letzte Zeile jedes Datensatzes gibt die jeweils schnellste Methode an.

Wie zu sehen ist, besteht beim Verzicht auf den Rückgabewert effektiv kein Unterschied zwischen List Comprehensions und for-Schleifen. Außerdem sieht man, dass *map* für solche Anwendungen wesentlich effizienter arbeitet, so ist es um einige Größenordnungen schneller als die beiden anderen Lösungen. Dementsprechend wird die ursprüngliche Implementierung durch eine auf *map* basierende ersetzt. Die optimierte Version läuft damit ≈ 250 mal schneller.

15 Schutz vor unauthorized Zugriff durch *setuid*

[yk]
TODO

Setuid, ausgeschrieben *Set User ID*, stellt ein erweitertes Dateirecht auf Linux dar.

Um ein verändern des Quellcodes zu verhindern wird dieser, für den Nutzer schreibgeschützt, abgelegt. Durch das Abspeichern von Daten, z.B. Geräte, Nutzerkonten, etc., sind jedoch bestimmte Schreibrechte notwendig. Um dies zu realisieren kann auf Linux *setuid* verwendet werden. Hierzu ist ein Launcher für die Software notwendig. In ihm werden dem ausführenden Nutzer nun bestimmte Schreibrechte gewährt. Diese sind jedoch nur durch das

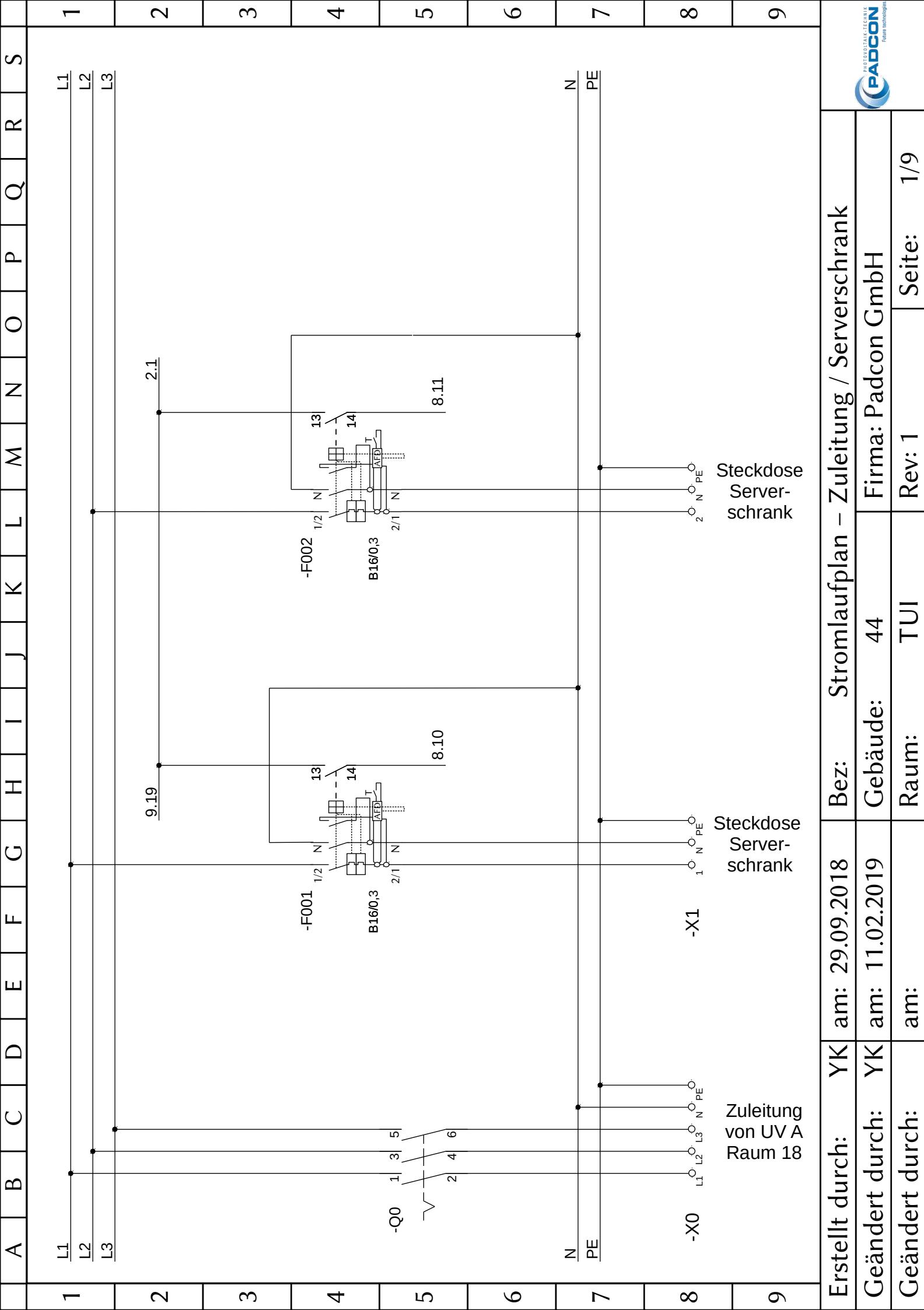
ausgeführte Programm (hier TUIventory) möglich und können somit keine Änderungen am Quellcode erzeugen.

Teil IV

Anhang

A Pläne

A.1 Stromlaufplan



Stromlaufplan – Zuleitung / Serverschrank

Bez: 29.09.2018 am: 29.09.2018

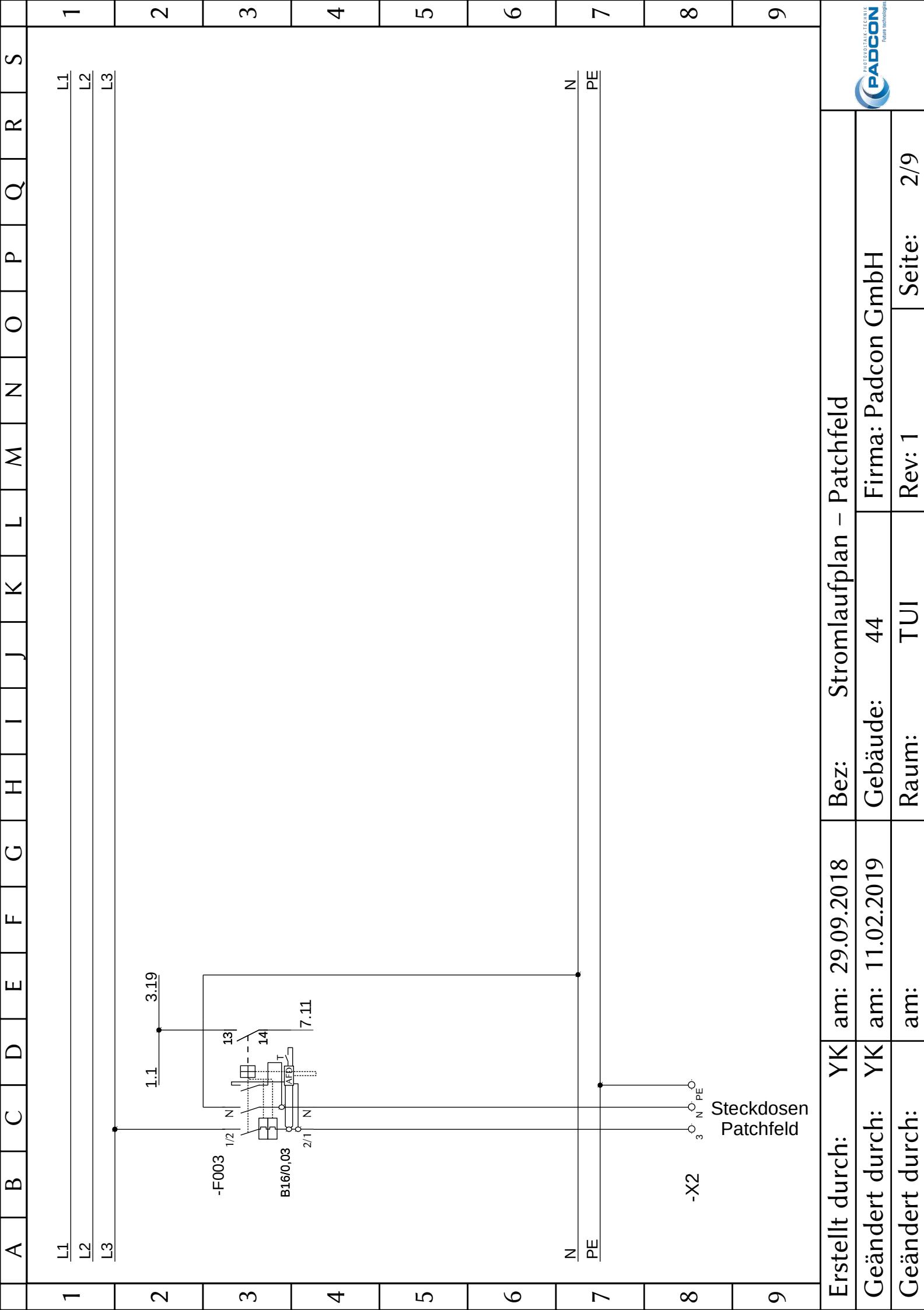
Gebäude: 44 am: 11.02.2019

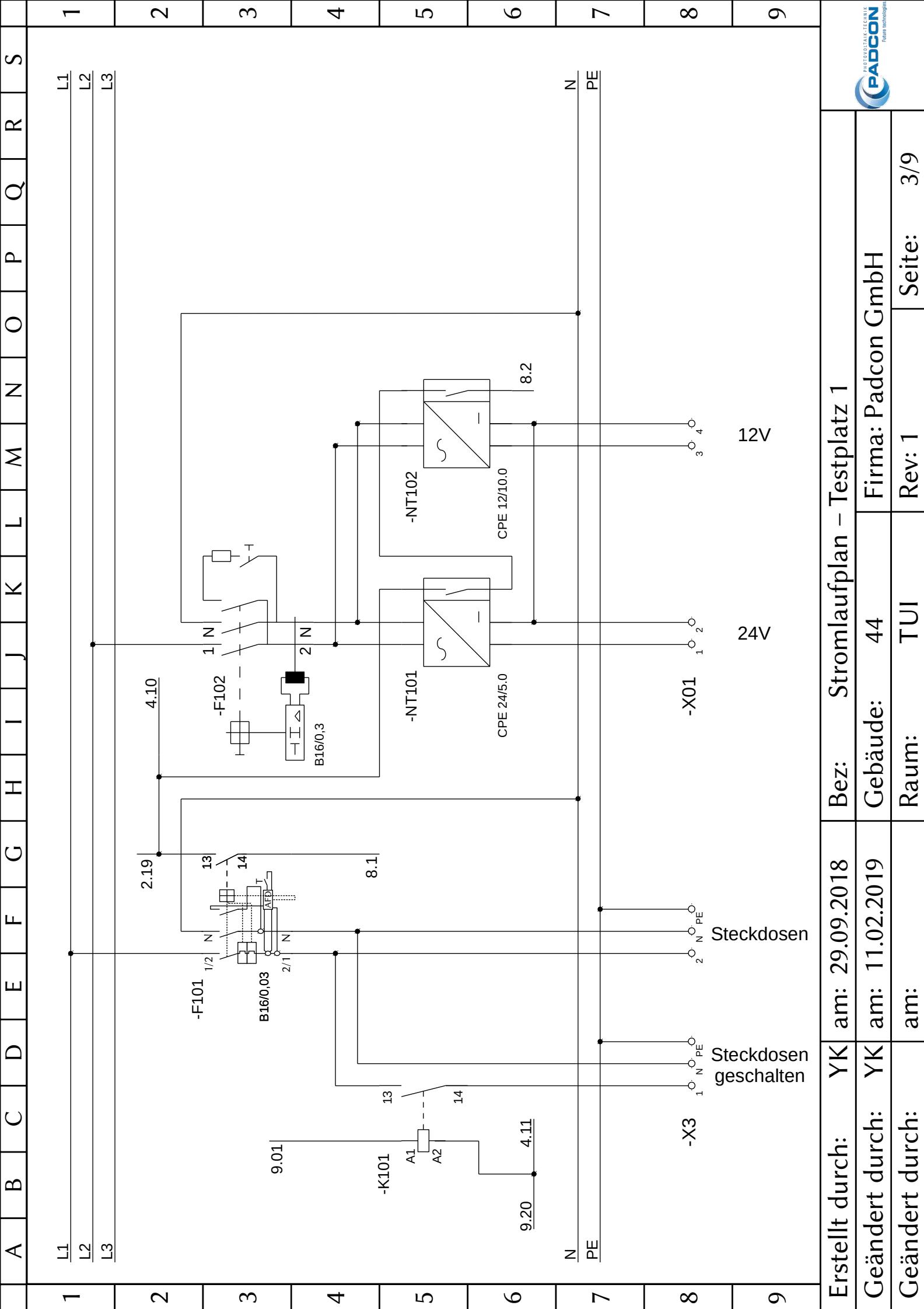
Raum: TUI am:

Erstellt durch: YK am: 29.09.2018

Geändert durch: YK am: 11.02.2019

Geändert durch: am:

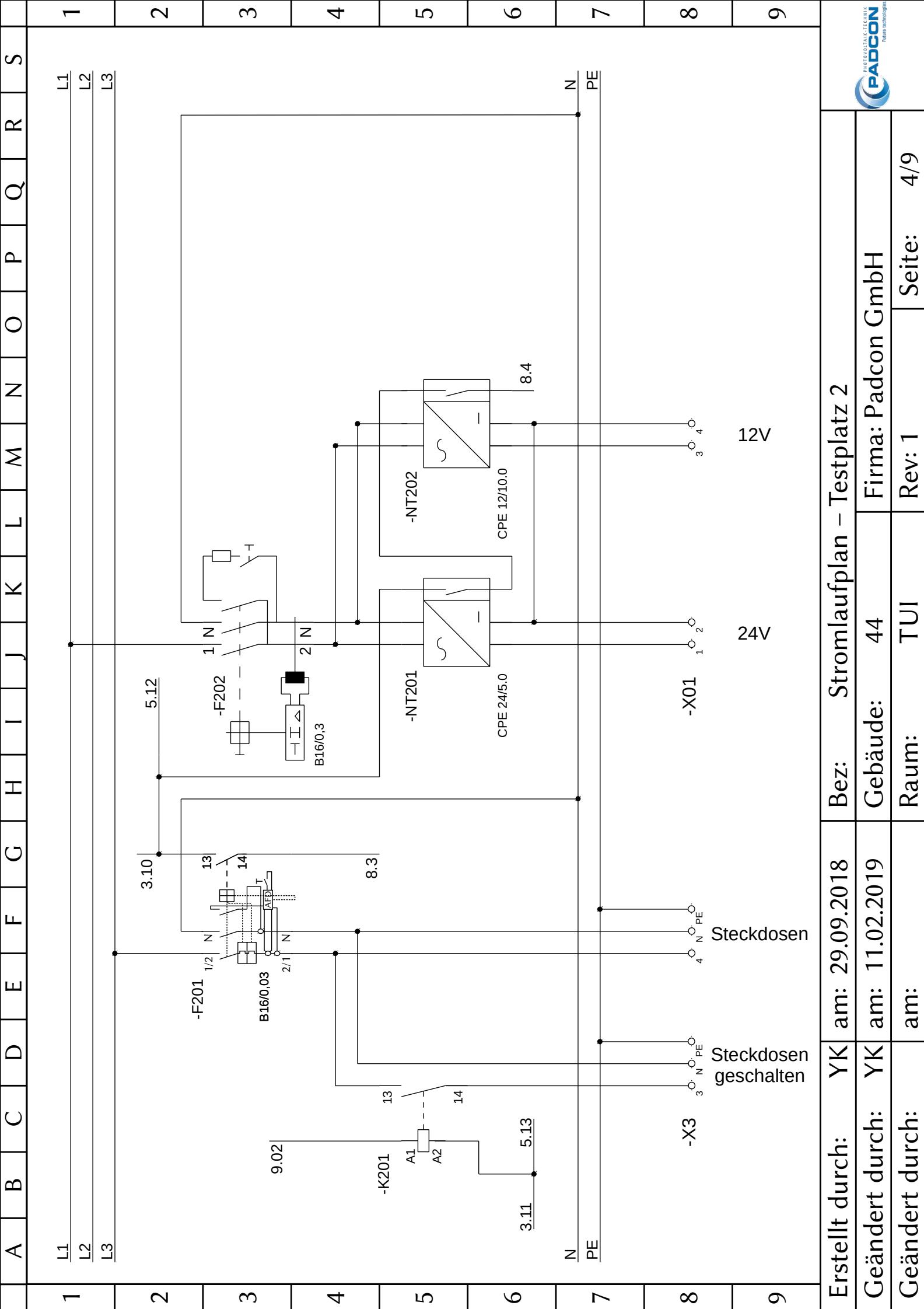




Erstellt durch: YK am: 29.09.2018 Bez: Stromlaufplan – Testplatz 1

Geändert durch: YK am: 11.02.2019 Gebäude: 44 Raum: TUI Rev: 1

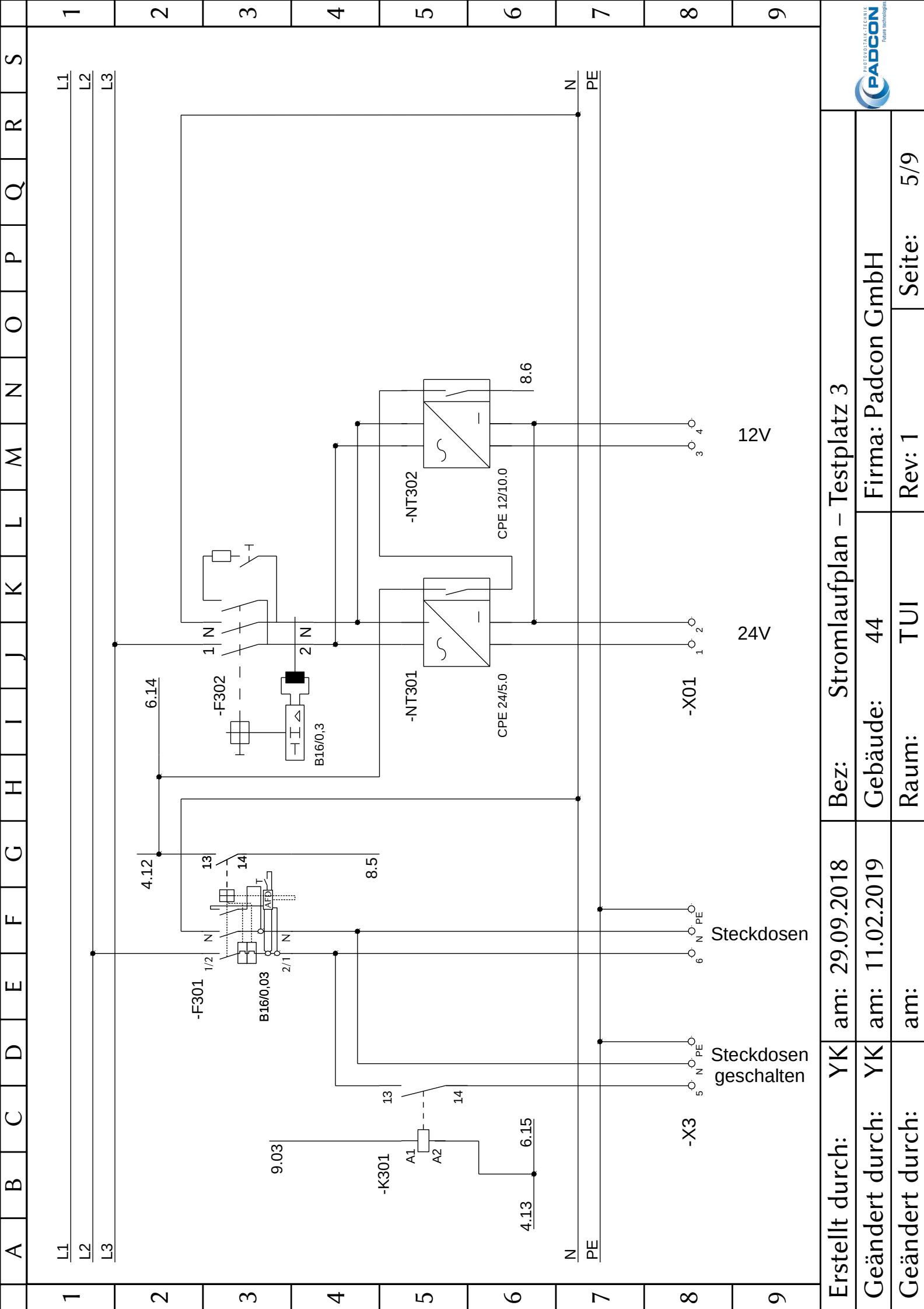
Firma: Padcon GmbH Seite: 3/9



Erstellt durch: YK am: 29.09.2018 Bez: Stromlaufplan – Testplatz 2

Geändert durch: YK am: 11.02.2019 Gebäude: 44 Firma: Padcon GmbH

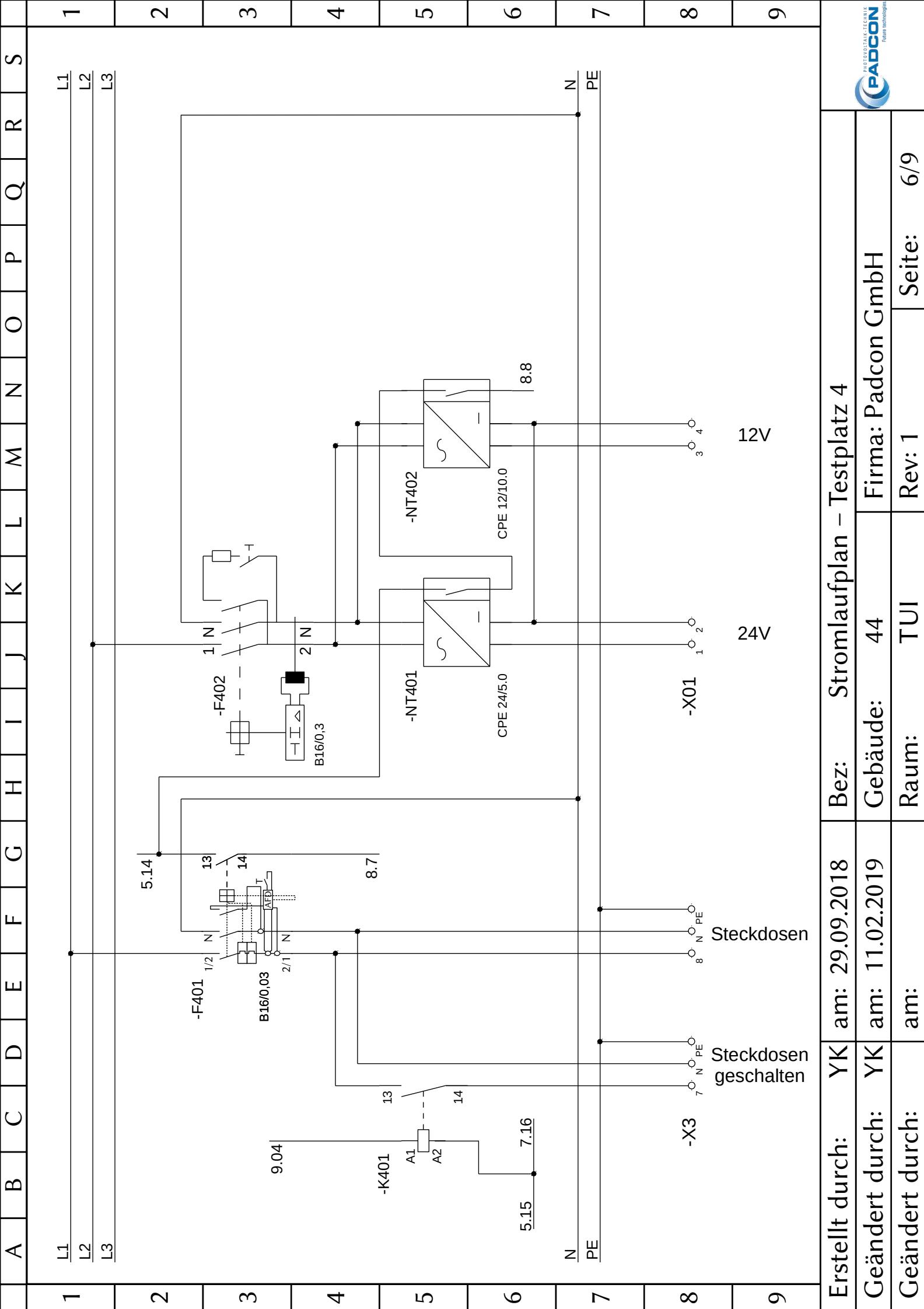
Geändert durch: am: Raum: TUI Rev: 1 Seite: 4/9



Erstellt durch: YK am: 29.09.2018 Bez: Stromlaufplan – Testplatz 3

Geändert durch: YK am: 11.02.2019 Gebäude: 44 Firma: Padcon GmbH

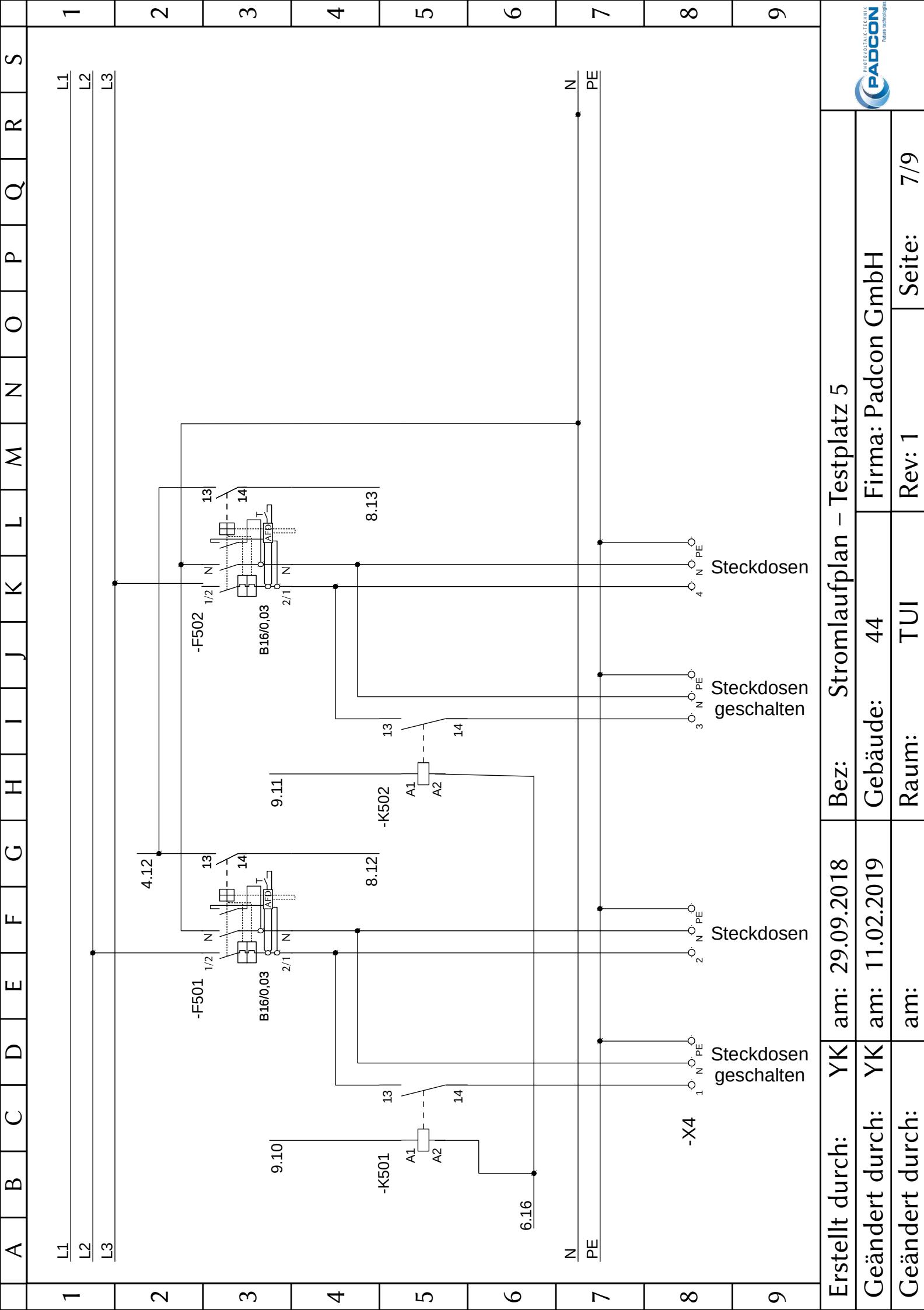
Geändert durch: am: Raum: TUI Rev: 1 Seite: 5/1

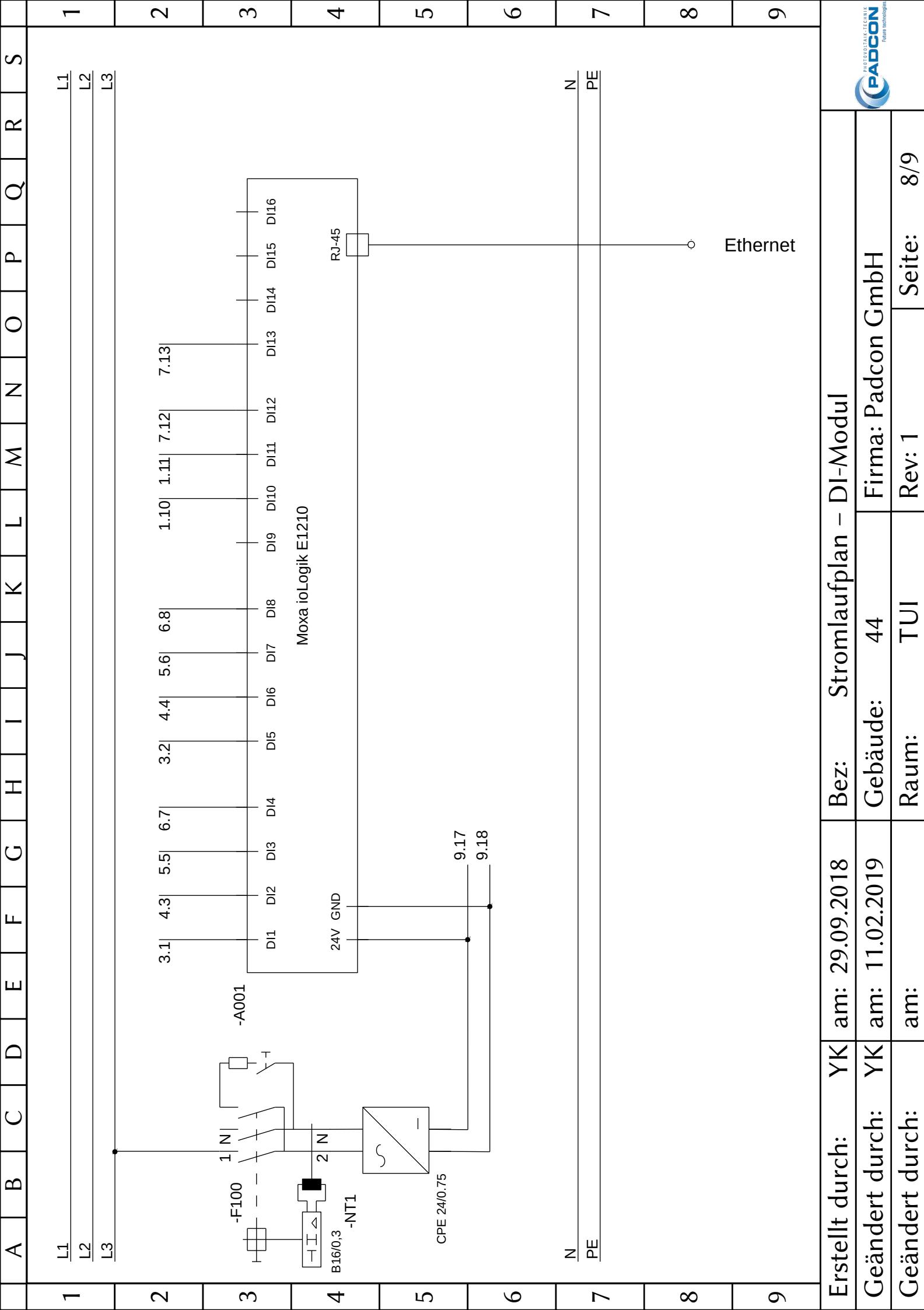


Erstellt durch: YK am: 29.09.2018 Bez: Stromlaufplan – Testplatz 4

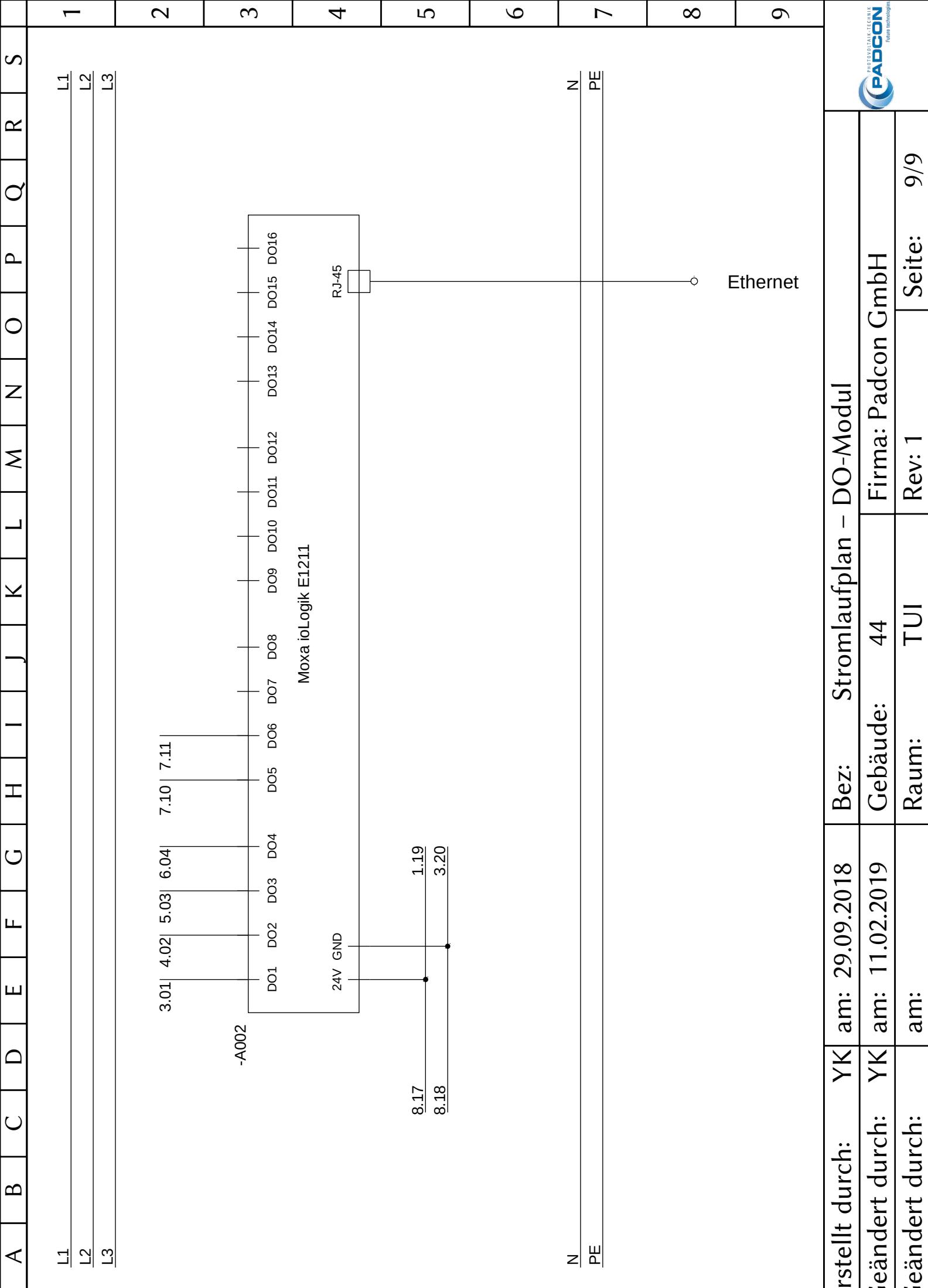
Geändert durch: YK am: 11.02.2019 Gebäude: 44 Firma: Padcon GmbH

Geändert durch: am: Raum: TUI Rev: 1 Seite: 6/1 Seite: 6/9





Erstellt durch:	YK	am: 29.09.2018	Bez:	Stromlaufplan – DI-Modul
Geändert durch:	YK	am: 11.02.2019	Gebäude:	44 Firma: Padcon GmbH
Geändert durch:		am:	Raum:	TU Rev: 1 Seite: 8/1

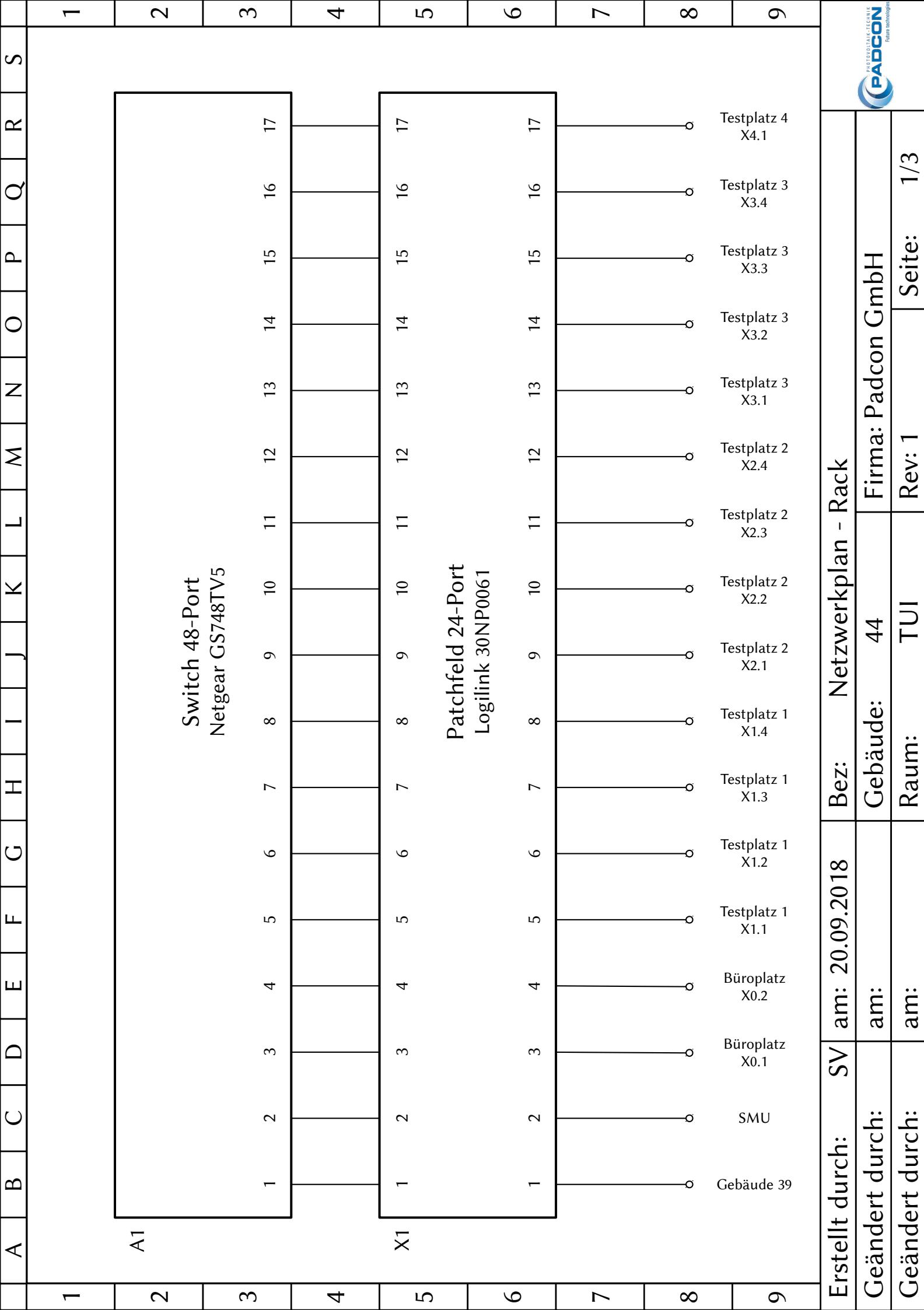


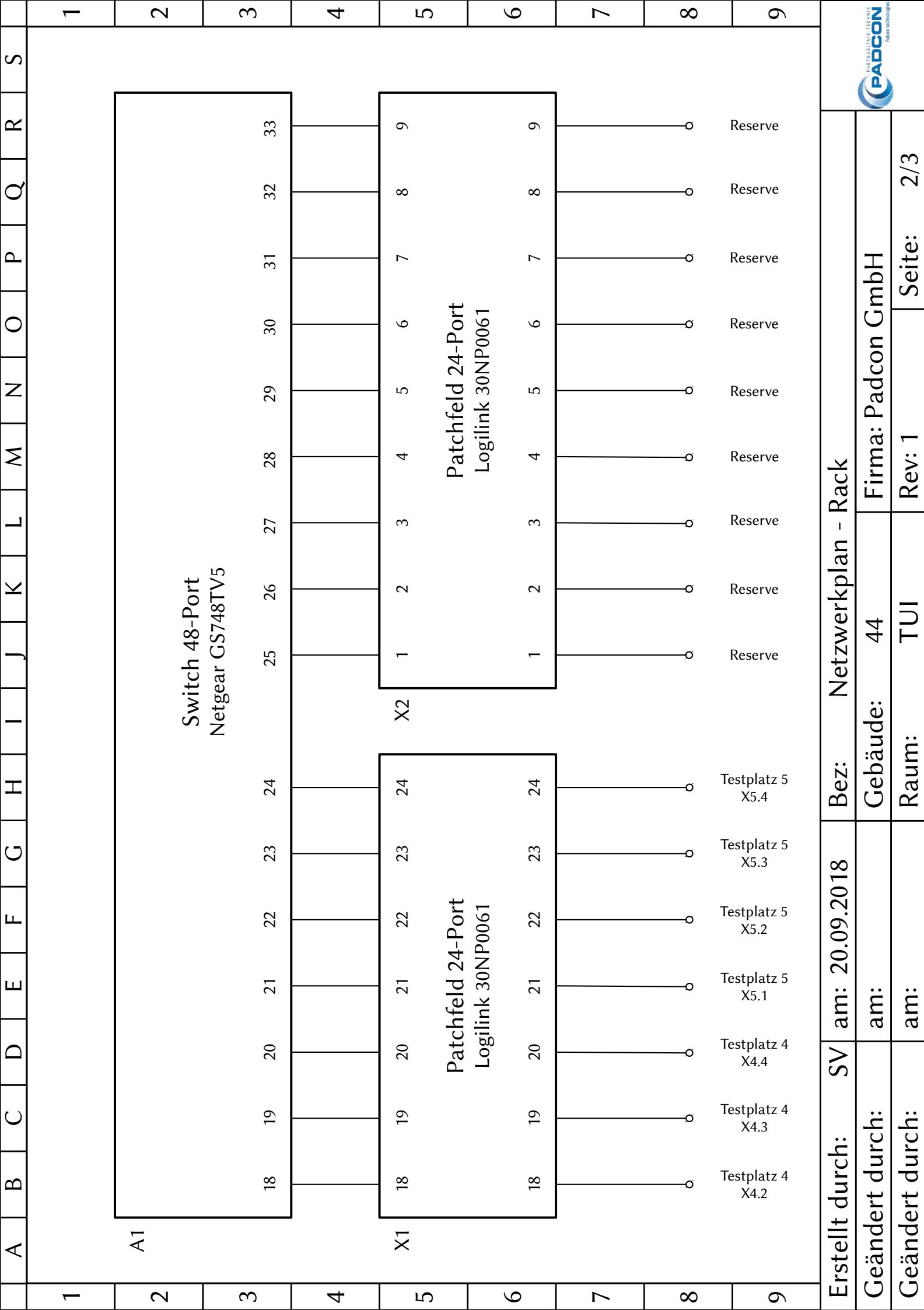


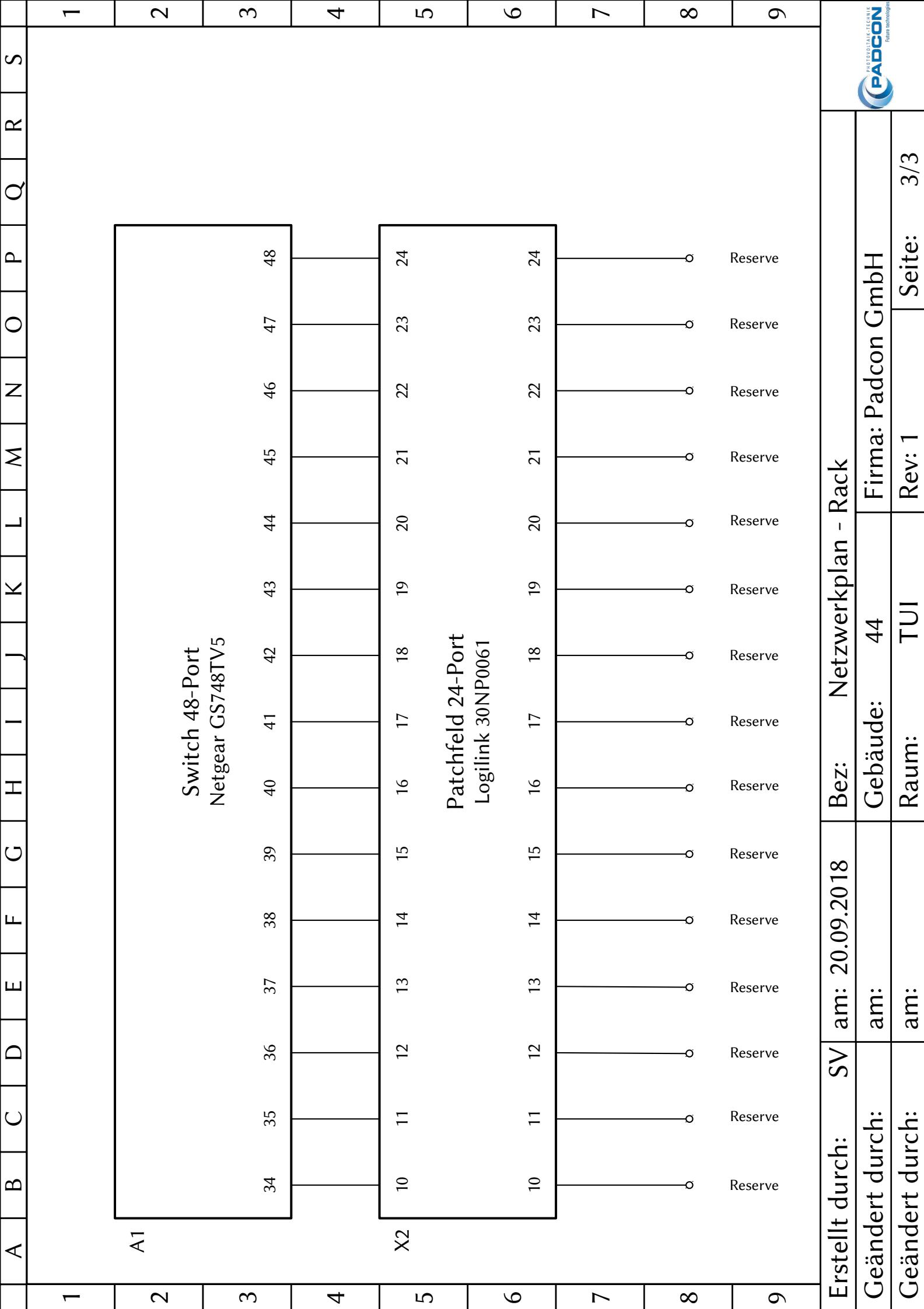
PADCON
PHOTOVOLTAIK-TECHNIK
Future technologies

Erstellt durch:	YK	am: 29.09.2018	Bez:	Stromlaufplan – DO-Modul	
Geändert durch:	YK	am: 11.02.2019	Gebäude:	44	
Geändert durch:	am:		Raum:	TUI	Rev: 1

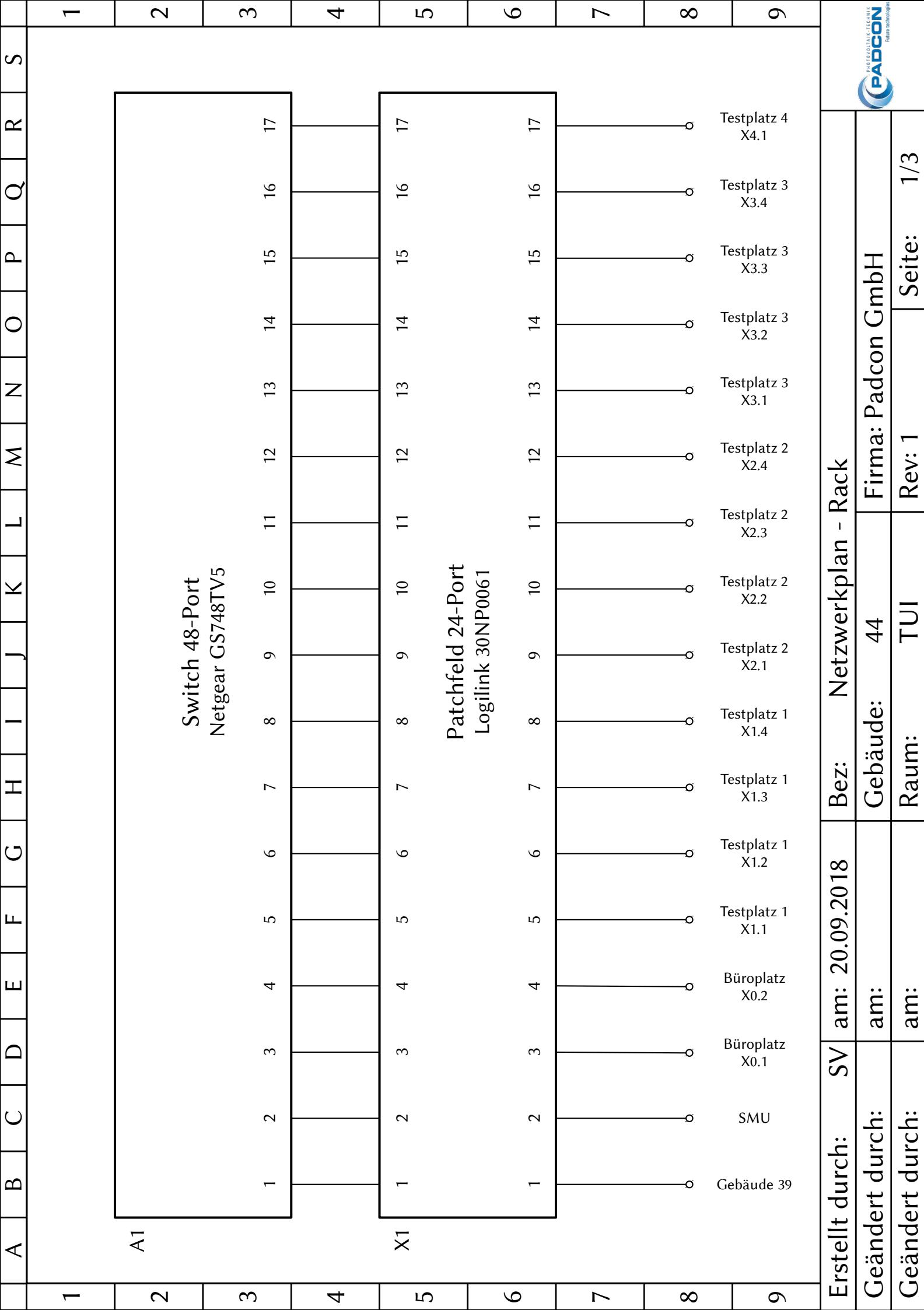
A.2 Netzwerkplan

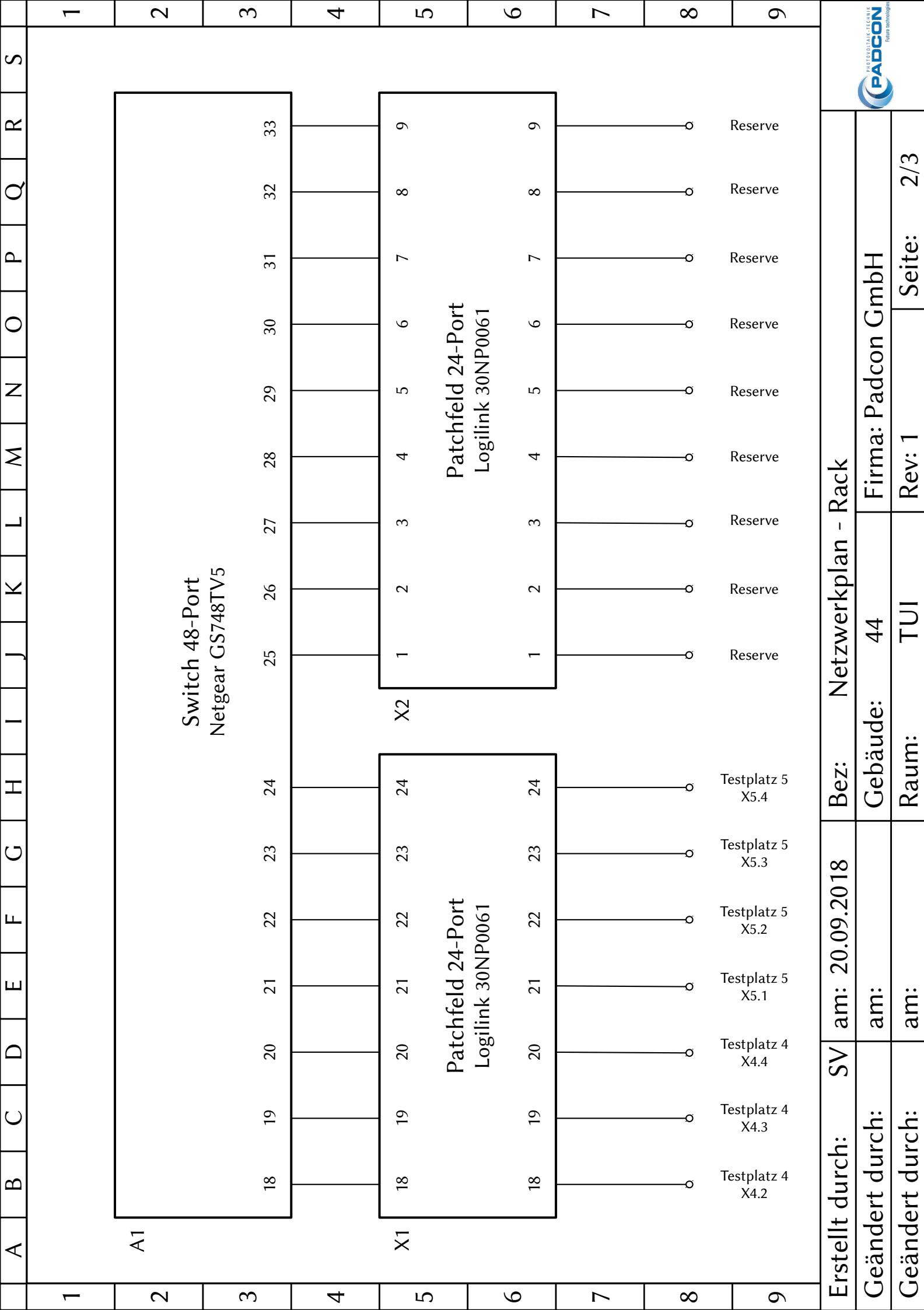


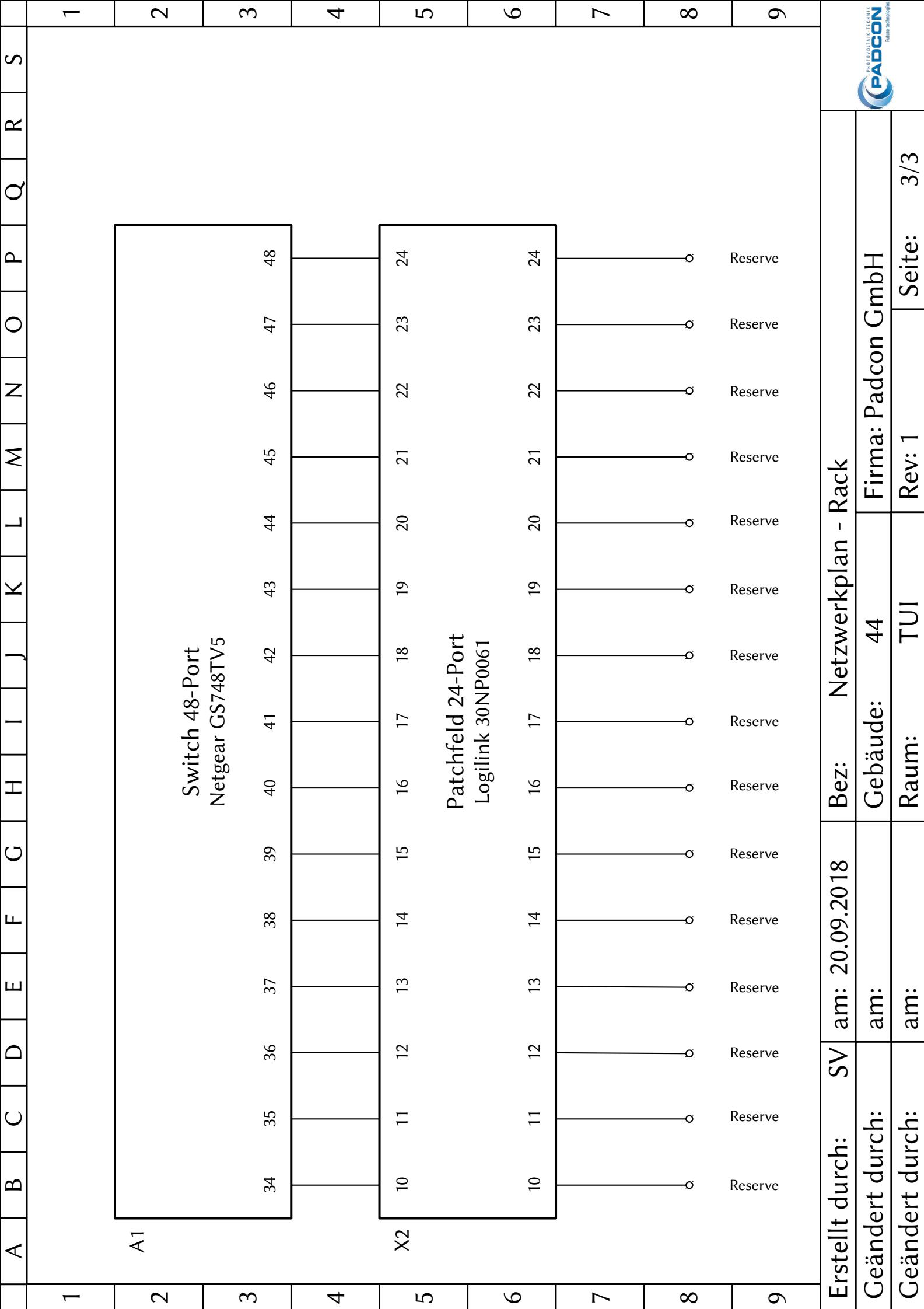




A.3 Netzwerkplan nach EN81346







B Quellcode

B.1 cli_get_barcode.py

```
1 """Simple barcodereader command line interface"""
2
3 from argparse import ArgumentParser
4 from sys import exit, stderr, stdout
5
6 parser = ArgumentParser(description="Extract all barcodes from an \
    image")
7 parser.add_argument("-i", "--image", dest="path", type=str, help="Path \
    to image")
8 parser.add_argument("-d", "--detailed", dest="detailed", action="\
    store_true", help="Return all extracted information")
9 parser.add_argument("-tb", "--enable_tracebacks", dest="\
    enable_tracebacks", action="store_true", help="Enable tracebacks \
    rather than custom messages")
10
11 args = parser.parse_args()
12 path = args.path
13 detailed = args.detailed
14 enable_tracebacks = args.enable_tracebacks
15
16 try:
17     from cv2 import imread
18 except ImportError as e:
19     if enable_tracebacks:
20         raise e
21     exit("Failed to import OpenCV (Module cv2)")
22 try:
23     from pyzbar import pyzbar
24 except ImportError as e:
25     if enable_tracebacks:
26         raise e
27     exit("Failed to import pyzbar")
28
29
30 if not path:
31     stderr.write("No path provided\n")
```

```

32     exit(1)
33
34 try:
35     image = imread(path)
36     barcodes = pyzbar.decode(image)
37     if not barcodes:
38         stderr.write("Couldn't find any barcodes\n")
39         exit(1)
40 except TypeError as e:
41     if enable_tracebacks:
42         raise e
43     stderr.write(f"There's no valid image at {path}\n")
44     exit(1)
45
46
47 if detailed:
48     stdout.write(str(barcodes))
49 else:
50     stdout.write(barcodes[0].data.decode("utf-8"))
51 exit(0)

```

B.2 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 platform, 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 \
18         marking them in the image
19     Be sure you can't use the LazyVideoStream instead!
20     """
21
22     Args:
23         target_resolution: Tuple of (width, height) to set the \
24             final resolution of the frames
25         camera_id: The id of the camera that's to be used (if your\
26             system only has one it's zero)
27
28     super().__init__(name=f"{self.__class__.__name__}Thread_{\
29         camera_id}")
30     self.camera = Camera(camera_id)
31     self.camera_id = camera_id
32     self.barcodes = []
33     self._mirror = False
34     self._abort = False
35     self.frame_lock = threading.Lock()
36     self.gp_lock = threading.Lock()
37     self._target_resolution = target_resolution
38     self._frame = np.zeros((1, 1))
39
40     def _set_frame(self, frame):
41         with self.frame_lock:
42             self._frame = frame
43
44     def _get_frame(self):
45         with self.frame_lock:
46             frame = self._frame
47             if self.mirror:
48                 frame = cv2.flip(frame, 1)
49             if self.target_resolution:
50                 frame = cv2.resize(frame, (self.target_resolution[0], self\
51                     .target_resolution[1]))
52         return frame
53
54     def _set_mirror(self, mirror):
55         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=\
78                                   _set_target_resolution)
79
80     @staticmethod
81     def rect_transformation(x, y, width, height):
82         """Transform rectangle of type "origin + size" to "two-point"
83         Args:
84             x (int): x coordinate of origin
85             y (int): y coordinate of origin
86             width (int): width of rectangle
87             height (int): height of rectangle
88         Returns:
89             Tuple of tuple of int with x-y-coordinate pairs for both \
90             points
91         """

```

```

90         return ((x, y), (x + width, y + height))
91
92     def find_and_mark_barcodes(self, frame):
93         """Find barcodes in given frame
94
95         Args:
96             frame: Frame that barcodes are to be detected in
97
98         Returns:
99             Tuple of frame where barcodes are marked, list of all \
100                found codes in frame
101
102        """
103
104        barcodes = pyzbar.decode(frame)
105        found_codes = []
106        for barcode in barcodes:
107            barcode_information = (barcode.type, barcode.data.decode("\
108                utf-8"))
109            if barcode_information not in found_codes:
110                found_codes.append(barcode_information)
111                poly = barcode.polygon
112                poly = np.asarray([(point.x, point.y) for point in poly])
113                poly = poly.reshape((-1, 1, 2))
114                cv2.polyline(frame, [poly], True, (0, 255, 0), 2)
115                cv2.rectangle(frame, *self.rect_transformation(*barcode.\
116                    rect), (255, 0, 0), 2)
117                x, y = barcode.rect[:2]
118                cv2.putText(
119                    frame,
120                    "{}({})".format(*barcode_information),
121                    (x, y - 10),
122                    cv2.FONT_HERSHEY_PLAIN,
123                    1,
124                    (0, 0, 255),
125                    1)
126
127        return frame, found_codes
128
129    def run(self):
130        with self.camera as camera:
131            while not self.abort:
132                frame = camera.read()[1]
133                marked_frame, found_codes = self.\
134                    find_and_mark_barcodes(frame)

```

```

126         self.frame = marked_frame
127     if found_codes:
128         self.barcodes.append(found_codes)
129
130
131 class LazyVideoStream(threading.Thread):
132     """Class for reading barcodes of all kinds from a video feed and \
133         marking them in the image
134     This Lazy implementation only processes frames on demand
135
136     Args:
137         target_resolution: Tuple of (width, height) to set the \
138             final resolution of the frames
139         camera_id: The id of the camera that's to be used (if your\
140             system only has one it's zero)
141
142     Attributes:
143         camera: Instance of Camera with for given camera_id
144         camera_id: Given camera_id
145         _mirror: Set to True to mirror the frame
146         gp_lock: general purpose lock for property access
147         _target_resolution: Tuple of (width, height) to set the \
148             final resolution of the frames
149         request_queue: Queue that's used to request a frame, \
150             request by putting True
151         frame_queue: Queue that answer frames get pushed into
152
153     Properties:
154         mirror: gp_lock locked _mirror
155         target_resolution: gp_lock locked _target_resolution
156
157     """
158     def __init__(self, target_resolution=None, camera_id=0):
159         super().__init__(name=f"{self.__class__.__name__}Thread_{id(\\"
160             self)})")
161         self.daemon = True
162         self.camera = Camera(camera_id)
163         self.camera_id = camera_id
164         self._mirror = False
165         self.gp_lock = threading.Lock()
166         self._target_resolution = target_resolution # (width, height)

```

```

160         self.request_queue = queue.Queue()
161         self.frame_queue = queue.Queue()
162         with self.camera:
163             pass
164
165     def _set_mirror(self, mirror):
166         with self.gp_lock:
167             self._mirror = mirror
168
169     def _get_mirror(self):
170         with self.gp_lock:
171             return self._mirror
172
173     def _set_target_resolution(self, resolution):
174         with self.gp_lock:
175             self._target_resolution = resolution
176
177     def _get_target_resolution(self):
178         with self.gp_lock:
179             return self._target_resolution
180
181     mirror = property(fget=_get_mirror, fset=_set_mirror)
182     target_resolution = property(fget=_get_target_resolution, fset=\
183                                 _set_target_resolution)
184
185     @staticmethod
186     def rect_transformation(x, y, width, height):
187         """Transform rectangle of type "origin + size" to "two-point"
188
189         Args:
190             x (int): x coordinate of origin
191             y (int): y coordinate of origin
192             width (int): width of rectangle
193             height (int): height of rectangle
194
195         Returns:
196             Tuple of tuple of int with x-y-coordinate pairs for \
197             both points
198
199         """
200         return ((x, y), (x + width, y + height))

```

```

198
199     def find_and_mark_barcodes(self, frame):
200         """Find barcodes in given frame
201         Args:
202             frame: Frame that barcodes are to be detected in
203         Returns:
204             Tuple of frame where barcodes are marked, list of all \
205                 found codes in frame
206         """
207
208         barcodes = pyzbar.decode(frame)
209         found_codes = []
210         for barcode in barcodes:
211             barcode_information = (barcode.type, barcode.data.decode("\
212                 utf-8"))
213             if barcode_information not in found_codes:
214                 found_codes.append(barcode_information)
215             poly = barcode.polygon
216             poly = np.asarray([(point.x, point.y) for point in poly])
217             poly = poly.reshape((-1, 1, 2))
218             cv2.polyline(frame, [poly], True, (0, 255, 0), 2)
219             # cv2.rectangle(frame, *self.rect_transformation(*barcode.\
220                 rect), (255, 0, 0), 2)
221             x, y = barcode.rect[:2]
222             """
223             cv2.putText(
224                 frame,
225                 "{}({})".format(*barcode_information),
226                 (x, y - 10),
227                 cv2.FONT_HERSHEY_PLAIN,
228                 1,
229                 (0, 0, 255),
230                 1)
231             """
232
233         return frame, found_codes
234
235     def request(self):
236         """Convenience function to abstract the request_queue from the \
237             user"""
238         self.request_queue.put(True)
239
240     def run(self):

```

```

234     """Start connection to camera and answer requests"""
235     with self.camera as camera:
236         while True:
237             self.request_queue.get()
238             frame = camera.read()[1]
239             marked_frame, found_codes = self.\
240                 find_and_mark_barcodes(frame)
241             if self.target_resolution:
242                 marked_frame = cv2.resize(marked_frame, (self.\
243                     target_resolution[0], self.target_resolution\
244                     [1]))
245             if self.mirror:
246                 marked_frame = cv2.flip(marked_frame, 1)
247                 self.frame_queue.put((marked_frame, found_codes))
248             self.request_queue.task_done()
249
250     class Camera():
251         def __init__(self, camera_id=0):
252             self.camera_id = camera_id
253         def __enter__(self):
254             if "win32" in platform:
255                 self.camera = cv2.VideoCapture(cv2.CAP_DSHOW + self.\
256                     camera_id)
257             else:
258                 self.camera = cv2.VideoCapture(self.camera_id)
259
260             if not self.camera.isOpened():
261                 raise IOError(f"Failed to open camera {self.camera_id}")
262             while not self.camera.read()[0]:
263                 pass
264             return self.camera
265         def __exit__(self, exc_type, exc_value, traceback):
266             self.camera.release()
267             while self.camera.isOpened():
268                 pass
269             return True
270
271 if __name__ == "__main__":
272     lazy_feed = LazyVideoStream()

```

```

270     lazy_feed.start()
271     window = "window"
272     cv2.namedWindow(window)
273
274     while True:
275         lazy_feed.request_queue.put(True)
276         frame, codes = lazy_feed.frame_queue.get()
277         lazy_feed.frame_queue.task_done()
278         cv2.imshow(window, frame)
279         cv2.waitKey(1)
280         if codes:
281             parallel_print(codes)

```

B.3 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 from passlib.hash import argon2
14 import sqlalchemy
15 from sqlalchemy import Boolean, Column, Float, Integer, LargeBinary, \
16     String
16 from sqlalchemy.ext.declarative import declarative_base
17 from PyQt5.QtGui import QImage, QPixmap
18
19 from logger import logger
20 from utils import absolute_path
21
22 orm = sqlalchemy.orm
23
24 Base = declarative_base()

```

```

25 if __name__ == "__main__":
26     print("")
27     if not input("Warning! Do you really want to delete the database \
28         and write some example data to it? [y/N]: ") in ("y", "Y"):
29         exit()
30     print("")
31     logger.info("Database cleared! This was authorized via a prompt")
32     with open(absolute_path("test.db"), "w") as f:
33         f.flush()
34 engine = sqlalchemy.create_engine(f"sqlite:///{absolute_path('test.db')\
35         }", echo=False)
36
37 class IntegrityError(IOError):
38     """Error to raise if an operation compromises database integrity\
39     """
40     def __init__(self, message):
41         self.message = message
42     def __str__(self):
43         return f'Couldn\'t create instance with given Parameters:{\
44             self.message}'
45 class BigInt(sqlalchemy.types.TypeDecorator):
46     """SQLAlchemy datatype for dealing with ints that potentially \
47         overflow a basic SQL Integer"""
48     impl = sqlalchemy.types.String
49     def process_bind_param(self, value, dialect):
50         """Gets called when writing to db"""
51         return str(value)
52     def process_result_value(self, value, dialect):
53         """Gets called when reading from db"""
54         return int(value)
55
56
57 def setup_context_session(engine):
58     """Factory for contextmanagers for Session objects
59     Initialize a ContextSession class

```

```

60
61     Args:
62         engine (sqlalchemy.engine.base.Engine): Engine that's \
63             bound to the sessionmaker
64
65     Example:
66         engine = sqlalchemy.create_engine('sqlite:///memory:')
67         CSession = setup_context_session(engine)
68         with CSession() as session:
69             session.add(user1)
70             session.add(user2)
71
72     class ContextSession():
73         _engine = engine
74
75         class _StateKeepingSession(orm.session.Session):
76             def __init__(self):
77                 super().__init__(bind=engine)
78                 self.instances = []
79
80             def add(self, instance, *args, **kwargs):
81                 self.instances.append(instance)
82                 super().add(instance, *args, **kwargs)
83
84             def add_all(self, instances, *args, **kwargs):
85                 self.instances += instances
86                 super().add_all(instances, *args, **kwargs)
87
88             def __enter__(self):
89                 self.session = self._StateKeepingSession()
90                 return self.session
91
92             def __exit__(self, exc_type, exc_value, traceback):
93                 if exc_value or exc_type or traceback:
94                     self.session.rollback()
95                     return False # propagate exceptions upwards
96                 else:
97                     self.session.commit()
98                     # [self.session.refresh(instance) for instance in self\
99                     .session.instances] if I didn't mess up the map \

```

```

    below, this can be deleted
98     for x in map(self.session.refresh, self.session.\
99         instances):
100         pass # refresh all instances via side effect of \
101             map
100         self.session.expunge_all()
101         self.session.close()
102         return True
103     return ContextSession
104
105
106 class Producer(Base):
107     """Represents a producer of articles"""
108     __tablename__ = "producers"
109     uid = Column(Integer, primary_key=True)
110     name = Column(String, unique=True)
111     articles = orm.relationship("Article", backref="producer")
112     def __init__(self, name, uid=None):
113         self.uid = uid
114         self.name = name.title()
115
116     def __str__(self):
117         return f"{self.uid} {self.name}"
118
119
120 class Article(Base):
121     """Represents an article"""
122     __tablename__ = "articles"
123     uid = Column(Integer, primary_key=True)
124     name = Column(String, unique=True)
125     producer_uid = Column(Integer, sqlalchemy.ForeignKey("producers.\
126         uid"))
126     #last_price = Column(Float(asdecimal=True))
127     devices = orm.relationship(
128         "Device",
129         backref=orm.backref("article", lazy="immediate"),
130         lazy="immediate")
131     def __init__(self, name, producer=None, uid=None):
132         self.uid = uid
133         self.name = name

```

```

134         self.producer = producer
135
136     def __str__(self):
137         return f"{self.uid}\u222a{self.name}"
138
139 class Device(Base):
140     """Represents a device"""
141     __tablename__ = "devices"
142     uid = Column(Integer, primary_key=True)
143     article_uid = Column(Integer, sqlalchemy.ForeignKey("articles.uid"\ \
144         ))
144     code = Column(String)
145     responsibility = orm.relationship(
146         "Responsibility",
147         backref=orm.backref("device", lazy="immediate", uselist=False)\ \
148             ,
149             lazy="immediate",
150             uselist=False)
150     def __init__(self, code=None, uid=None):
151         self.uid = uid
152         self.code = code
153
154     def __str__(self):
155         return f"{self.article.name}\u222a{self.uid}\u222aID\u222a{self.uid}"
156
157
158 class PhoneNumber(Base):
159     """Get a Phone Number from raw input string
160     PhoneNumber can also be stored in database table
161
162     Args:
163         raw_string (str): String that holds the number
164
165     To do:
166         - config to set locale (subscriber number prefix and \
167             country code)
168             probably not possible due to precision limitations of \
169                 system-locale
168     """
169     __tablename__ = "phone_numbers"

```

```

170     uid = Column(Integer, primary_key=True)
171     user_uid = Column(Integer, sqlalchemy.ForeignKey("users.uid"))
172     raw_string = Column(String)
173     country_code = Column(String)
174     area_code = Column(String)
175     subscriber_number = Column(String)
176     extension = Column(String)
177     pattern = r"(?P<prelude>(?P<country_code>(?:\+\d{1,3})|(?:0))\u00a5*(?P<
178         <area_code>(?:[1-9])+)?)?(?P<subscriber_number>(?:\d+\u00a5?)+)(?P<\u00a5
179         extension>[-\+]\d+)?"
180     def __init__(self, raw_string):
181         self.raw_string = raw_string
182         if not self.raw_string:
183             self.country_code = ""
184             self.area_code = ""
185             self.subscriber_number = ""
186             self.extension = ""
187             self.match = None
188             return
189         self.match = re.search(self.pattern, self.raw_string)
190         if not self.match:
191             raise self.NoNumberFoundWarning(raw_string)
192         self.country_code = self._extract_country_code()
193         self.area_code = self._extract_area_code()
194         self.subscriber_number = self._extract_subscriber_number()
195         self.extension = self._extract_extension()
196     @staticmethod
197     def _whitespacekiller(string):
198         """Remove all non-number characters from a string"""
199         return re.sub(r"\D", "", string)
200     def _extract_country_code(self):
201         country_code = self.match.group("country_code")
202         if country_code and "+" in country_code:
203             return self._whitespacekiller(country_code)
204         else:
205             return "049"
206     def _extract_area_code(self):

```

```

208     area_code = self.match.group("area_code")
209     area_code = "9321" if not area_code else area_code
210     return self._whitespacekiller(area_code)
211
212 def _extract_subscriber_number(self):
213     subscriber_number = self.match.group("subscriber_number")
214     return self._whitespacekiller(subscriber_number)
215
216 def _extract_extension(self):
217     extension = self.match.group("extension")
218     extension = "" if not extension else extension
219     return self._whitespacekiller(extension)
220
221 def __str__(self):
222     """Build string of the telephone number based on DIN 5008"""
223     extension = f"-{self.extension}" if self.extension else ""
224     return f"+{self.country_code}{self.area_code}{self.\
225             subscriber_number}{extension}"
226
227 def __format__(self, format_spec):
228     return f"{str(self) : {format_spec}}"
229
230 class NoNumberFoundWarning(Warning):
231     def __init__(self, raw_string):
232         self.raw_string = raw_string
233     def __str__(self):
234         return f'No telephone number was found in: {self.\
235             raw_string}',
236
237 class Location(Base):
238     """Represents a physical location where a User (or Responsibility) \
239         may be located"""
240     __tablename__ = "locations"
241     uid = Column(Integer, primary_key=True)
242     name = Column(String, unique=True)
243     #devices = orm.relationship("Device", backref=orm.backref("\
244         location", uselist=False))
245     responsibilities = orm.relationship("Responsibility", backref="\
246         location", lazy="immediate")

```

```

243     def __init__(self, name="", uid=None):
244         self.name = name.title()
245         self.uid = uid
246
247     def __str__():
248         return self.name
249
250
251 class User(Base):
252     """Represents a user of the application"""
253     __tablename__ = "users"
254     uid = Column(Integer, primary_key=True)
255     e_mail = Column(String, unique=True)
256     password = Column(LargeBinary)
257     salt = Column(BigInt)
258     name = Column(String)
259     surname = Column(String)
260     is_admin = Column(Boolean)
261     location_uid = Column(Integer, sqlalchemy.ForeignKey("locations.\
262                 uid"))
262     location = orm.relationship(
263         "Location",
264         backref=orm.backref("users", lazy="immediate"),
265         uselist=False,
266         lazy="immediate")
267     responsibilities = orm.relationship("Responsibility", backref="\\
268                 user")
269     phonenumbers = orm.relationship(
270         "PhoneNumber",
271         backref=orm.backref("user", cascade="all,delete-orphan", \
272                         single_parent=True),
273         uselist=False,
274         lazy="immediate")
275
276     def __init__(self, e_mail, password, name="", surname="", \
277                  phonenumbers="", uid=None, salt=None):
278         self.uid = uid
279         self.e_mail = e_mail.lower()
280         self.salt = salt if salt else self.new_salt()
281         self.name = name.title()

```

```

279         self.surname = surname.title()
280         if isinstance(phonenumbers, PhoneNumber):
281             self.phonenumber = phonenumbers
282         else:
283             self.phonenumber = PhoneNumber(phonenumbers)
284         self.hash(password)
285         self.is_admin = False
286
287     @staticmethod
288     def new_salt():
289         return randbits(128)
290
291     def hash(self, password):
292         """Hash a string with argon2
293         The salt is concatenated with the e-mail to get the final salt
294         """
295         salt = f"{self.salt}{self.e_mail}".encode()
296         hash = argon2(using(
297             salt=salt,
298             rounds=512,
299             memory_cost=1024,
300             max_threads=8,
301             digest_size=256).hash(password))
302         self.password = re.match(r"\$argon2i\$v=\d+\$m=\d+,t=\d+,p=\d\"
303             +\$(?P<hash>.+)", hash).\
304             group("hash").encode()
305
306     def __str__(self):
307         return f"{self.name} {self.surname}".title()
308
309 class Responsibility(Base):
310     """Represents a responsibility a User has for a Device"""
311     __tablename__ = "responsibilities"
312     device_uid = Column(Integer, sqlalchemy.ForeignKey("devices.uid"), \
313         primary_key=True)
314     user_uid = Column(Integer, sqlalchemy.ForeignKey("users.uid"), \
315         primary_key=True)
316     location_uid = Column(Integer, sqlalchemy.ForeignKey("locations.\"
317         uid"), primary_key=True)

```

```

315
316     def __init__(self, device=None, user=None, location=None):
317         self.device = device
318         self.user = user
319         self.location = location
320
321     def __str__(self):
322         return f"{self.user} is responsible for device {self.device} \
323             at {self.location}"
324
325 Base.metadata.create_all(bind=engine) # Database initialized
326 logger.info("Database initialized, tables verified")
327
328
329 class Timeout(Thread):
330     """Timer that runs in background and executes a function if it's \
331         not refreshed
332     Important: This is different from the threading.Timer class in \
333         that it can provide
334     arguments to a function as well as allows resetting the timer, \
335         rather than canceling
336     completely. To cancel a Timeout set function to None, the thread \
337         will then close itself
338     down on the next lifecycle check.
339
340     Args:
341         function: function that is executed once time runs out
342         timeout: time in seconds after which the timeout executes \
343             the function
344         args: arguments for function
345         kwargs: keyword arguments for function
346
347     Attributes:
348         timeout: time in seconds after which the timer times out
349         function: function to be executed once time runs out
350         args: arguments for function
351         kwargs: keyword arguments for function
352         lock: mutex for various attributes
353         timed_out: boolean presenting whether the timer timed out

```

```

349             last_interaction_timestamp: unix timestamp of last refresh\
350                 /initialization
350
351         Properties:
352             timer: Shows the time remaining until timeout
353         """
354
355     def __init__(self, timeout, function, *args, **kwargs):
356         super().__init__(name=f"{self.__class__._name_}Thread_{\
357                         function.__name__}")
357         self.daemon = True
358         self.timeout = timeout
359         self.function = function
360         self.args = args
361         self.kwargs = kwargs
362         self.lock = Lock()
363         self.timed_out = False
364         self.reset()
365
366     def _refresh_timer(self):
367         with self.lock:
368             return self.timeout - (time() - self.\
369                             last_interaction_timestamp)
369
370     timer = property(fget=_refresh_timer)
371
372     def reset(self):
373         """Reset the internal timer"""
374         with self.lock:
375             if not self.timed_out:
376                 self.is_reset = True
377                 self.last_interaction_timestamp = time()
378
379     def run(self):
380         """Start the timer"""
381         if self.function is None:
382             logger.debug("Timeout thread closed because function was \
383                           None")
383             return
384         with self.lock:

```

```

385         if self.is_reset:
386             self.is_reset = False
387         else:
388             self.timed_out = True
389             self.function(*self.args, **self.kwargs)
390             return
391         difference_to_timeout = self.timeout - (time() - self.\
392                                         last_interaction_timestamp)
393         sleep(difference_to_timeout)
394         self.run()
395
396 class VideoStreamUISync(Thread):
397     """Class to tie a LazyVideoStream to some canvas in Qt
398     Args:
399         canvas: canvas has to be able to take pixmaps/implement \
400                 setPixmap
400         videotostream: Instance of LazyVideoStream that supplies the\
401                         frames
401         signal: qt-signal that's emitted if a barcode has been \
402                         recognized
402         job: set to "both", "codes" or "frames" to set whether it \
403                         should
403         only process image data, barcodes and signals or both
404
405     Attributes:
406         barcodes: Counter that holds all found barcodes USE \
407                     barcode_lock WHEN ACCESSING!
407         barcode_lock: Lock for barcodes
408         sensibility: How often a barcode has to be recognized to \
409                         count it as valid
409     """
410     def __init__(self, canvas, videotostream, signal, job="both"):
411         super().__init__(name=f"{self.__class__._name_}Thread_{id(\\"
412                                         self)}")
412         self.canvas = canvas
413         self.videostream = videotostream
414         self.daemon = True
415         self.barcodes = Counter()
416         self.barcode_lock = Lock()

```

```

417         self.sensibility = 10
418         self.signal = signal
419         self.job = job
420
421     @staticmethod
422     def _matrice_to_QPixmap(frame):
423         """Convert cv2/numpy matrice to a Qt QPixmap"""
424         frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
425         height, width, channel = frame.shape
426         image = QImage(frame.data, width, height, 3 * width, QImage.\
427                         Format_RGB888)
428         return QPixmap(image)
429
430     def get_most_common(self):
431         """Get barcode with highest occurence"""
432         with self.barcode_lock:
433             return self.barcodes.most_common(1)[0]
434
435     def reset_counter(self):
436         with self.barcode_lock:
437             self.barcodes = Counter()
438
439     def update_counter(self, found_codes):
440         with self.barcode_lock:
441             self.barcodes.update(found_codes)
442
443     def run(self):
444         """Start synchronization"""
445         while True:
446             self.videostream.request_queue.put(True)
447             frame, found_codes = self.videostream.frame_queue.get()
448             if self.job == "frames" or self.job == "both":
449                 pixmap = self._matrice_to_QPixmap(frame)
450                 self.canvas.setPixmap(pixmap)
451             elif self.job == "codes" or self.job == "both":
452                 if found_codes:
453                     self.update_counter(found_codes)
454                     most_common = self.get_most_common()
455                     if most_common[1] > self.sensibility:
456                         self.signal.emit(most_common[0][1])

```

```

456                     self.reset_counter()
457                     sleep(10)
458             cv2.waitKey(1)
459
460
461 if __name__ == "__main__":
462     """Tests"""
463     """Timer Test
464     import threading
465     timeout = Timeout(timeout=10, function=print, args=["timed out"])
466     timeout.start()
467     reset_thread = threading.Thread(target=lambda: timeout.reset() if \
468         input() else None) # function for testing - reset on input
469     reset_thread.start()
470     t = 0
471     delta_t = 1
472     t1 = time()
473     while not timeout.timed_out:
474         print(f"Not timed out yet - {t:.2f} seconds passed")
475         print(timeout.timer)
476         t += delta_t
477         sleep(delta_t)
478         print(time() - t1)
479         timeout.join()
480         reset_thread.join()
481     """
482
483     import random
484     random.seed(0)
485
486     location1 = Location("Testplatz\u20ac1")
487     location2 = Location("Testplatz\u20ac2")
488     location3 = Location("Testplatz\u20ac3")
489     location4 = Location("Testplatz\u20ac4")
490     location5 = Location("Testplatz\u20ac5")
491     location6 = Location("B\u0107ro")
492     location7 = Location("Lager")
493     locations = (
494         location1,
495         location2,

```

```

495         location3 ,
496         location4 ,
497         location5 ,
498         location6 ,
499         location7)
500
501
502     user0 = User(e_mail="admin", password="123", name="Admin", surname="\n",
503                 " , phonenumber="888")
504     user0.is_admin = True
505     user1 = User(e_mail="Karl@googlemail.com", password="123", name="\n",
506                 " Karl", surname="KĂśnig", phonenumber="71825")
507     user2 = User(e_mail="sv@gmail.com", password="password", name="\n",
508                 " Stefan", surname="VĂślz", phonenumber="09729\u20221411")
509     user2.is_admin = True
510     user3 = User(e_mail="yk@gmail.com", password="asdf", name="Yannis"\n,
511                 " , surname="Kohler", phonenumber="09729\u20221552")
512     user3.is_admin = True
513     user4 = User(e_mail="MaxMustermann@t-online.de", password="123456"\n,
514                 " , name="Max", surname="Mustermann", phonenumber="12345")
515     user5 = User(e_mail="JohnDoe@web.com", password="john", name="John"\n,
516                 " , surname="Doe", phonenumber="+01\u20221123-5")
517
518     users = (
519         user0 ,
520         user1 ,
521         user2 ,
522         user3 ,
523         user4 ,
524         user5)
525
526     for user in users:
527         user.location = random.choice(locations)
528
529
530     producer1 = Producer("ABB")
531     producer2 = Producer("iEi")
532     producer3 = Producer("Moxa")
533     producer4 = Producer("Wago")
534     producer5 = Producer("Phoenix\u2022Contact")
535     producers = (

```

```

529         producer1 ,
530         producer2 ,
531         producer3 ,
532         producer4 ,
533         producer5)
534
535     article1 = Article("CP-E\u24/20.0")
536     article2 = Article("EtherDevice\u24Switch\u316")
537     article3 = Article("ioLogik\u24E1241")
538     article4 = Article("Managed\u24Switch\u852-104")
539     article5 = Article("MINI-PS-100-240AC/5DC/3")
540     article6 = Article("UIBX-250-BW")
541     """
542     article7 = Article("PID Killer")
543     article8 = Article("IPC AMOS-3005-1Q12A2")
544     article9 = Article("IPC JBC323U591-3160-B")
545     article10 = Article("IPC HM-1000")
546     article11 = Article("MSwitch JRL116M-2F-M")
547     article12 = Article("Switch SPIDER 8TX")
548     article13 = Article("Switch SPIDER 5TX")"""
549     article1.producer = producer1
550     article2.producer = producer3
551     article3.producer = producer3
552     article4.producer = producer4
553     article5.producer = producer5
554     article6.producer = producer2
555     """
556     article7.producer = producer6
557     article8.producer = producer7
558     article9.producer = producer8
559     article10.producer = producer8
560     article11.producer = producer8
561     article12.producer = producer9
562     article13.producer = producer9"""
563     articles = (
564         article1 ,
565         article2 ,
566         article3 ,
567         article4 ,
568         article5 ,

```

```

569         article6)
570
571     devices = [Device(str(i)) for i in range(35)]
572     resps = [Responsibility() for device in devices]
573     for device, resp in zip(devices, resps):
574         device.article = random.choice(articles)
575         while resp.user == user0 or not resp.user:
576             resp.user = random.choice(users)
577             resp.location = random.choice(locations)
578             resp.device = device
579
580     Session = orm.sessionmaker(bind=engine)
581     session = Session()
582
583     session.add_all(locations)
584     session.add_all(users)
585     session.add_all(producers)
586     session.add_all(articles)
587     session.add_all(devices)
588     session.add_all(resps)
589
590     print("-"*30)
591     print(f"{article1.producer.name} produces the following articles")
592     for art in producer1.articles:
593         print(f'{art.name}'*5)
594         print(f'{art.name}'*10) instances of these articles are:")
595         for device in art.devices:
596             print(f"device.code:{device.code}>20} stored at {device.responsibility}\n
597             .location.name:{device.location.name}<20}")
598     print("-"*30)
599
600     del user1
601     del user2
602     del location1
603     del location2
604
605     try:
606         print(user1.name)
607     except Exception:
608         print("There's no user1, this is wanted and good")

```

```

608     print("-"*30)
609
610     users = session.query(User).all()
611     locations = session.query()
612
613     for user in users:
614         print(f"{user.uid:>5} {user.e_mail:>40} {user.phonenumber}\n"
615             f":<20> {user.location.name}")
616     print("-"*30)
617     print("Known responsibilities for these users are:")
618     for user in users:
619         for resp in user.responsibilities:
620             print(f" {str(resp.user):^30} {resp.device.code:^8} {resp.\n"
621                 f"location.name:^15} {resp.device.article.name:^30}")
622     for device in devices:
623         print(f" id={device.uid} name={device.article.name}")
624
625     session.commit()
626     session.close()

```

B.4 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
13 from utils import absolute_path
14
15 PUBLIC_KEY_PATH = absolute_path("pub.key")
16
17 def generate_key(path_public, path_private):

```

```

18     """Generate new RSA key-pair and save it to the given paths
19
20     Args:
21         path_public: Path where the public-key should be stored
22         path_private: Path where the private-key should be stored
23     """
24
25     key = RSA.generate(4096)
26     with open(path_public, "wb") as f:
27         f.write(key.publickey().exportKey())
28     with open(path_private, "wb") as f:
29         f.write(key.exportKey())
30
31 def read_keys(path_public, path_private):
32     """Read a key-pair from the given paths and build ciphers from it
33
34     Args:
35         path_public: Path where the public-key is stored
36         path_private: Path where the private-key is stored
37
38     Returns:
39         (PKCS1_OAEP-cipher from public-key, PKCS1_OAEP-decipher \
40             from private-key)
41     """
42     with open(path_public, "rb") as f:
43         publickey = RSA.importKey(f.read())
44     with open(path_private, "rb") as f:
45         privatekey = RSA.importKey(f.read())
46     cipher = PKCS1_OAEP.new(publickey)
47     decipher = PKCS1_OAEP.new(privatekey)
48     return cipher, decipher

```

B.5 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:>20}. {funcName:30}}{ \
10         message} ', \
11         style="{}",
12         datefmt="%Y-%m-%dT%H:%m:%S")
13 handler.setFormatter(formatter)
14 logger = logging.getLogger()
15 logger.addHandler(handler)
16 logger.setLevel(logging.INFO)
17
18 del handler
19 del formatter

```

B.6 main.py

```

1 """main module that ties everything together"""
2 __version__ = "0.3.0"
3
4 import logging
5 import pathlib
6 import sys
7 from time import sleep
8
9 from PyQt5.QtWidgets import QApplication
10
11 from barcodereader import LazyVideoStream, VideoStream
12 from classes import VideoStreamUISync, Timeout
13 from config import config, SettingsManger
14 import keys
15 from logger import logger
16 import ui
17 from utils import absolute_path
18
19 if "win32" in sys.platform:
20     import ctypes
21     try:
22         app_id = f"Padcon.TUIventory.DesktopApp.{__version__}"

```

```

23         ctypes.windll.shell32.SetCurrentProcessExplicitAppUserModelID(\ 
24             app_id)
25     except AttributeError as e:
26         logger.debug(str(e))
27
28 class Dummy():
29     pass
30
31
32 def main():
33     public_key_path = keys.PUBLIC_KEY_PATH
34     if not public_key_path.exists():
35         private_key_path = absolute_path("priv.key")
36         keys.generate_key(public_key_path, private_key_path)
37     app = QApplication(sys.argv)
38     dialog_main = ui.MainDialog()
39     dialog_main.showMaximized()
40
41     try:
42         videostream = LazyVideoStream()
43         videostream.mirror = config["mirror"]
44         videostream.start()
45         logger.info(f"Camera_{videostream.camera_id} successfully \
46             opened")
47         video_ui_sync_1 = VideoStreamUISync(
48             dialog_main.ui.videoFeed, videostream,
49             dialog_main.code_recognized, "frames")
50         video_ui_sync_2 = VideoStreamUISync(
51             dialog_main.ui.videoFeed, videostream,
52             dialog_main.code_recognized, "codes")
53         video_ui_sync_1.start()
54         video_ui_sync_2.start()
55         logger.info("Connected Camera to UI")
56     except IOError as e:
57         logger.error(str(e))
58         videostream = Dummy()
59         video_ui_sync_1 = Dummy()
60         video_ui_sync_2 = Dummy()

```

```

61     settings_manager = SettingsManger(video_ui_sync_1, video_ui_sync_2 \
62                                         , videostream)
63     settings_manager.start()
64     dialog_main.settings_event = settings_manager.event
65     return app.exec_()
66
67 if __name__ == "__main__":
68     # Profiling via terminal
69     # import cProfile
70     # cProfile.run("main()", sort="time")
71
72     sys.exit(main())

```

B.7 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

```

B.8 qr_generator.py

```

1 """Handles QR-Code generation"""
2
3 from logger import logger
4 import re
5
6 import qrcode
7 import qrcode.image.svg as svg
8
9 def generate_qr(device, path):

```

```

10     """Generate the QR-Code for a given device and store it locally as \
11         svg"""
12     qr = qrcode.QRCode(
13         version=None,
14         error_correction=qrcode.constants.ERROR_CORRECT_H,
15         box_size=20,
16         border=6)
17     code = f"id={device.uid}\nname={device.article.name}"
18     qr.add_data(code)
19     qr.make(fit=True) # autosize according to data
20
21     filetype = re.match(r".*\.(?P<filetype>.*$)", str(path), re.\
22         IGNORECASE)\ \
23         .group("filetype").lower() # could probably also use .suffix \
24         of Path object
25     if filetype == "svg":
26         img = qr.make_image(image_factory=svg.SvgPathFillImage)
27     else:
28         raise NotImplementedError("See qrcode-docs at https://pypi.org/ \
29             /project/qrcode/ to see how to install additional formats" \
30             , filetype)
31     img.save(str(path))
32     logger.info(f"Successfully created and saved qr-code for {device.\
33         uid}")
34
35 if __name__=="__main__":
36     import pathlib
37     class TestArticle():
38         def __init__(self, name):
39             self.name = name
40     class TestDevice():
41         def __init__(self, uid, code, article):
42             self.uid = uid
43             self.code = code
44             self.article = article
45     article = TestArticle("Article")
46     device = TestDevice(10, "", article)
47     generate_qr(device, pathlib.Path("test.svg"))

```

B.9 slots.py

```
1 from functools import wraps
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):
16     """Function-decorator to automatically add the instance a function\
           returns to DB"""
17     @wraps(function)
18     def synchronized_function(*args, **kwargs):
19         instance = function(*args, **kwargs)
20         try:
21             save_to_db(instance)
22         except sqlalchemy.exc.IntegrityError: # uid already in db
23             raise classes.IntegrityError(str(args) + str(kwargs))
24         return instance
25     return synchronized_function
26
27
28 def save_to_db(instance):
29     """Save instance to it's corresponding table"""
30     with CSession() as session:
31         session.add(instance)
32
33
34 @synchronized
35 def create_user(*args, **kwargs):
36     """Create a new user"""
37     new_user = classes.User(*args, **kwargs)
38     return new_user
```

```

39
40
41     @synchronized
42     def create_admin(*args, **kwargs):
43         """Create a new admin"""
44         new_admin = classes.User(*args, **kwargs)
45         new_admin.is_admin = True
46         return new_admin
47
48
49     def login(e_mail, password):
50         """Log user into application
51         Checks if there's a user of given name in the database,
52         if the given password is correct and returns the user if both is \
53             the case
54         Args:
55             e_mail (str): e-mail of the user that wants to log in
56             password (str): user provided password to check against
57         """
58         e_mail = e_mail.lower()
59         with CSession() as session:
60             try:
61                 user = session.query(classes.User).filter_by(e_mail=e_mail\
62                     ).first()
63                 user_at_gate = classes.User(e_mail, password, salt=user.\
64                     salt)
65                 if compare_digest(user_at_gate.password, user.password):
66                     session.expunge(user)
67                     logger.info(f"Successfully logged in as {user.uid}")
68                     return user
69                 else:
70                     logger.info(f"Attempted login with wrong password for \
71                         user {e_mail}")
72                     return None
73             except (AttributeError, ValueError) as e: #user not found \
74                 exception
75                 logger.info(f"Attempted login from unknown user {e_mail}")
76                 raise ValueError(f"Attempted login from unknown user {\
77                     e_mail}")

```

```

73
74 def logout():
75     """Log user out of application"""
76     pass
77
78
79 @synchronized
80 def create_device(article):
81     new_device = classes.Device()
82     new_device.article = article
83     return new_device
84
85
86 @synchronized
87 def create_article(*args, **kwargs):
88     return classes.Article(*args, **kwargs)
89
90
91 @synchronized
92 def create_location(*args, **kwargs):
93     return classes.Location(*args, **kwargs)
94
95
96 @synchronized
97 def create_producer(*args, **kwargs):
98     return classes.Producer(*args, **kwargs)
99
100
101 def generate_password(len_=15):
102     """Generate an human readable password of given length"""
103     alphabet = ascii_letters
104     without = list("0OIl")
105     pw_chars = alphabet + digits + "!_,.-_+-*()[]{}$%=?~Ž#’~ÃÂ§Ã&"
106     pw_chars = "".join((letter for letter in pw_chars if letter not in \
107                         without))
108     pw = "".join((choice(pw_chars) for i in range(len_)))
109     return pw
110
111 def reset_password(user):

```

```

112     user.salt = user.new_salt()
113     password = generate_password()
114     user.hash(password)
115     return password
116
117
118 def reset_admin_password(user, path_public, path_private):
119     try:
120         cipher, decipher = keys.read_keys(path_public, path_private)
121     except ValueError:
122         raise ValueError(f"Invalid RSA private-key at {path_private}!")
123     text = b"True"
124     ciphertext = cipher.encrypt(text)
125     if compare_digest(decipher.decrypt(ciphertext), text):
126         keys.generate_key(path_public, path_private)
127         new_password = reset_password(user)
128         return new_password
129     else:
130         return None
131
132
133 if __name__ == "__main__":
134     from datetime import datetime
135
136     with CSession() as session:
137         loc = session.query(classes.Location).first()
138         session.add(loc)
139
140     print(list(map(lambda x: str(x), loc.users)))
141
142     user = create_user(str(datetime.now()), "password", "name", \
143                         "surname", "94123-12315-123")
144
145     with CSession() as session:
146         session.add(user)
147         session.add(loc)
148         user.location = loc
149
150     print(user.uid)

```

```

150     print(user.location.uid)
151
152     with CSession() as session:
153         session.add(user)
154         session.add(loc)
155         user.name = "updated"
156
157     admin = create_admin(str(datetime.now()), "password", "name", "\\\n"
158                           "surname", "94123\u202212315-123")
159     print(admin.is_admin)
160
161     with CSession() as session:
162         article = session.query(classes.Article).first()
163         session.add(article)
164         session.add_all(article.devices)
165
166     list(map(print, article.devices))

```

B.10 ui.py

```

1 """PyQt5 UI classes and linking to slots"""
2
3 import os
4 import pathlib
5 import re
6 import sys
7
8 from PyQt5 import uic, QtCore, QtGui, QtWidgets
9 from PyQt5.QtGui import QColor, QIcon, QPainter, QPen
10 from PyQt5.QtCore import QSize, Qt, pyqtSignal
11 from PyQt5.QtWidgets import QFileDialog, QPushButton, QStyle
12
13 import classes
14 from config import config
15 import keys
16 from logger import logger
17 import slots
18 import utils
19 from qr_generator import generate_qr
20

```

```

21 CSession = classes.setup_context_session(classes.engine)
22
23
24 class MainDialog(QtWidgets.QMainWindow):
25     code_recognized = pyqtSignal(str)
26
27     def __init__(self, parent=None):
28         path = utils.absolute_path("mainScaling.ui")
29         super().__init__(parent)
30         self.ui = uic.loadUi(path, self)
31         self.logged_in_user = None
32         self.savepath = None
33         self.set_tree()
34         self.set_combobox_location_u()
35         self.set_combobox_location_u_admin()
36         self.set_combobox_location_d()
37         self.set_combobox_producer_a()
38         self.set_combobox_producer_d()
39         self.set_combobox_device_user()
40         self.set_combobox_device_location()
41         self.set_combobox_article_d()
42         self.set_combobox_user_d()
43         self.set_combobox_user_admin()
44         self.setMouseTracking(True)
45
46         self.ui.rb_mirror_yes.setChecked(config["mirror"])
47         self.ui.rb_mirror_no.setChecked(not config["mirror"])
48
49         self.t_path_device.setText(config["qr_path"])
50
51         icon = QtGui.QIcon()
52         icon_path = utils.absolute_path("pictures/TUI_Logo.png")
53         icon.addFile(f"{icon_path}", QtCore.QSize(256, 256))
54         self.setWindowIcon(icon)
55
56         self.ui.b_user_login.clicked.connect(self.b_user_login_click)
57         self.ui.b_create_new_qrcode.clicked.connect(self.\
58             b_create_new_qrcode_click)
59         self.ui.b_user_logout.clicked.connect(self.b_user_logout_click\
59             )

```

```

59         self.ui.b_home_1.clicked.connect(self.b_home_1_click)
60         self.ui.b_user_change.clicked.connect(self.b_user_change_click \
61             )
61         self.ui.b_user_change_admin.clicked.connect(self.\ \
62             b_user_change_admin_click)
62         self.ui.b_user_reset_admin.clicked.connect(self.reset_password \
63             )
63         self.ui.b_save_device.clicked.connect(self.b_save_device_click \
64             )
64         self.ui.b_change_device.clicked.connect(self.\ \
65             b_change_device_click)
65         self.ui.b_delete_device.clicked.connect(self.\ \
66             b_delete_device_click)
66         self.ui.b_create_device.clicked.connect(self.\ \
67             b_create_device_click)
67         self.ui.b_create_article.clicked.connect(self.\ \
68             b_create_article_click)
68         self.ui.b_create_producer.clicked.connect(self.\ \
69             b_create_producer_click)
69         self.ui.b_qr_path.clicked.connect(self.b_qr_path_click)
70         self.ui.b_save_settings.clicked.connect(self.mirror_setting)
71         self.ui.b_tab_1.clicked.connect(self.b_tab_1_click)
72         self.ui.b_tab_2.clicked.connect(self.b_tab_2_click)
73         self.ui.b_tab_3.clicked.connect(self.b_tab_3_click)
74         self.ui.b_tab_4.clicked.connect(self.b_tab_4_click)
75         self.ui.b_tab_5.clicked.connect(self.b_tab_5_click)
76         self.ui.in_phone.textChanged.connect(self.set_phonenumber)
77         self.ui.in_phone_admin.textChanged.connect(self.\ \
78             set_phonenumber)
78         self.ui.cb_producer_d.currentIndexChanged.connect(self.\ \
79             reload_combobox_article_d)
79         self.ui.cb_user_admin.currentIndexChanged.connect(self.\ \
80             reload_user_change)

81         foldericon_path = utils.absolute_path("pictures/folder.png")
82         self.ui.b_save_device.setIcon(QtGui.QIcon(f"{foldericon_path}\ \
83             )))
83         self.ui.b_save_device.setIconSize(QtCore.QSize(20,20))
84         self.ui.b_qr_path.setIcon(QtGui.QIcon(f"{foldericon_path}")))
85         self.ui.b_qr_path.setIconSize(QtCore.QSize(20,20))

```

```

86
87     self.setAutoFillBackground(True) # background / white
88     p = self.palette()
89     p.setColor(self.backgroundRole(), Qt.white)
90     self.setPalette(p)
91
92     palette1 = self.line_1.palette()
93     role1 = self.line_1.backgroundRole()
94     palette1.setColor(role1, QColor('blue'))
95     self.ui.line_1.setPalette(palette1)
96     self.ui.line_1.setAutoFillBackground(True)
97
98     palette2 = self.line_2.palette()
99     role2 = self.line_2.backgroundRole()
100    palette2.setColor(role2, QColor('blue'))
101    self.ui.line_2.setPalette(palette2)
102    self.ui.line_2.setAutoFillBackground(True)
103
104    palette3= self.line_3.palette()
105    role3 = self.line_3.backgroundRole()
106    palette3.setColor(role3, QColor('blue'))
107    self.ui.line_3.setPalette(palette3)
108    self.ui.line_3.setAutoFillBackground(True)
109
110    palette4 = self.line_4.palette()
111    role4 = self.line_4.backgroundRole()
112    palette4.setColor(role4, QColor('blue'))
113    self.ui.line_4.setPalette(palette4)
114    self.ui.line_4.setAutoFillBackground(True)
115
116    palette5 = self.line_5.palette()
117    role5 = self.line_5.backgroundRole()
118    palette5.setColor(role5, QColor('blue'))
119    self.ui.line_5.setPalette(palette5)
120    self.ui.line_5.setAutoFillBackground(True)
121
122    palette7 = self.bottom_frame.palette()
123    role7 = self.bottom_frame.backgroundRole()
124    palette7.setColor(role7, QColor('blue'))
125    self.ui.bottom_frame.setPalette(palette7)

```

```

126         self.ui.bottom_frame.setAutoFillBackground(True)
127         self.ui.bottom_frame.setStyleSheet("color:\u00e4white")
128
129         self.ui.stackedWidget.setCurrentIndex(0)
130         self.ui.line_1.show()
131         self.ui.line_2.hide()
132         self.ui.line_3.hide()
133         self.ui.line_4.hide()
134         self.ui.line_5.hide()
135         self.update_user_dependant()
136         self.code_recognized.connect(self.recognized_barcode)
137
138         self.ui.t_setting_timeout.setText(str(config["timeout"]))
139         self.ui.t_setting_qr_path.setText(config["qr_path"])
140
141
142     def status_bar_text(self, text, time, color):
143         """Status bar"""
144         self.ui.label_status.setStyleSheet(f"color:\u00e4{color}")
145         self.ui.label_status.setText(text)
146         self.timeout = classes.Timeout(time, MainDialog.\ \
147             status_bar_clear, self)
148         self.timeout.start()
149
150     def status_bar_clear(self):
151         self.ui.label_status.setStyleSheet("color:\u00e4black")
152         self.ui.label_status.setText("")
153
154     def mirror_setting(self):
155         """Save settings to config.ini"""
156         if self.ui.rb_mirror_yes.isChecked():
157             mirror = True
158         else:
159             mirror = False
160
161         timeout = float(self.ui.t_setting_timeout.text())
162         if timeout < 1.9:
163             timeout = config["timeout"]
164             self.status_bar_text("Bitte\u00e4geben\u00e4Sie\u00e4einen\u00e4Wert\u00e4von\u00e4\\"
165                 mindestens\u00e4zwei\u00e4Minuten\u00e4ein", 4, "red")

```

```

164         return
165
166     qr_path = self.ui.t_setting_qr_path.text()
167
168     settings = [mirror, timeout, qr_path]
169     conf = [config["mirror"], config["timeout"], config["qr_path"]\]
170         ]
171
172     if not all(True if x==y else False for (x,y) in zip(settings, \
173             conf)):          #check if changes are made
174         config["mirror"] = mirror
175         config["timeout"] = timeout
176         config["qr_path"] = qr_path
177         config.flush()
178
179         self.status_bar_text("Ihre Einstellungen wurden \
180             erfolgreich gespeichert", 3, "green")
181
182     self.t_path_device.setText(config["qr_path"])
183
184     self.settings_event.set()
185
186 else:
187     self.status_bar_text("Es wurden keine Änderungen \
188             vorgenommen", 3, "green")
189
190 def b_qr_path_click(self):
191     """Show selected path in textbox"""
192     qr_path = QtWidgets.QFileDialog.getExistingDirectory(self, \
193             "Bitte wählen Sie ein Verzeichnis")
194
195     if qr_path:
196         self.t_setting_qr_path.setText(qr_path)
197
198 def b_home_1_click(self):
199     self.ui.stackWidget.setCurrentIndex(0)
200     self.ui.line_1.show()
201     self.ui.line_2.hide()
202     self.ui.line_3.hide()
203     self.ui.line_4.hide()
204     self.ui.line_5.hide()
205
206 def b_tab_1_click(self):
207     self.ui.stackWidget.setCurrentIndex(0)
208     self.ui.line_1.show()

```

```

199         self.ui.line_2.hide()
200         self.ui.line_3.hide()
201         self.ui.line_4.hide()
202         self.ui.line_5.hide()
203
204     def b_tab_2_click(self):
205         self.ui.stackedWidget.setCurrentIndex(1)
206         self.ui.line_1.hide()
207         self.ui.line_2.show()
208         self.ui.line_3.hide()
209         self.ui.line_4.hide()
210         self.ui.line_5.hide()
211
212     def b_tab_3_click(self):
213         self.ui.stackedWidget.setCurrentIndex(2)
214         self.ui.line_1.hide()
215         self.ui.line_2.hide()
216         self.ui.line_3.show()
217         self.ui.line_4.hide()
218         self.ui.line_5.hide()
219
220     def b_tab_4_click(self):
221         self.ui.line_1.hide()
222         self.ui.line_2.hide()
223         self.ui.line_3.hide()
224         self.ui.line_4.show()
225         self.ui.line_5.hide()
226         self.ui.stackedWidget.setCurrentIndex(4)
227         if self.logged_in_user:                      # check if \
228             logged in user is admin
229             if self.logged_in_user.is_admin:
230                 self.ui.stackedWidget.setCurrentIndex(3)
231
232     def b_tab_5_click(self):
233         self.ui.stackedWidget.setCurrentIndex(5)
234         self.ui.line_1.hide()
235         self.ui.line_2.hide()
236         self.ui.line_3.hide()
237         self.ui.line_4.hide()
238         self.ui.line_5.show()

```

```

238
239     def set_tree(self):
240         """Fill main screen treeWidget"""
241         self.treeWidget.clear()
242         with CSession() as session:
243             locations = session.query(classes.Location).all()
244             locations.sort(key=lambda loc: loc.name)
245             for location in locations:
246                 users = session.query(classes.User).join(classes.\
247                     Responsibility).\
248                     filter_by(location=location).all()
249                     users.sort(key=lambda user: str(user))
250                     if users:
251                         location_item = QtWidgets.QTreeWidgetItem([str(\
252                             location.name)])
253                         self.treeWidget.addTopLevelItem(location_item)
254                     else:
255                         continue
256                     for user in users:
257                         devices = session.query(classes.Device).join(\
258                             classes.Responsibility).\
259                             filter_by(user=user, location=location).all()
260                         user_item = QtWidgets.QTreeWidgetItem([str(user)])
261                         devices.sort(key=lambda device: str(device))
262                         location_item.addChild(user_item)
263                         for device in devices:
264                             device_item = QtWidgets.QTreeWidgetItem([str(\
265                                 device)])
266                             user_item.addChild(device_item)

267     def set_combobox_location_u(self):
268         """Fill Location ComboBox for User creation"""
269         self.cb_location_u.clear()
270         with CSession() as session:
271             locations = session.query(classes.Location).all()
272             locations.sort(key=lambda location: str(location))
273             for location in locations:
274                 self.cb_location_u.addItem(location.name)

275     def set_combobox_location_u_admin(self):

```

```

274     """Fill Location ComboBox for User creation"""
275     self.cb_location_u_admin.clear()
276     with CSession() as session:
277         locations = session.query(classes.Location).all()
278         locations.sort(key=lambda location: str(location))
279         for location in locations:
280             self.cb_location_u_admin.addItem(location.name)
281
282     def set_combobox_location_d(self):
283         """Fill Location ComboBox for Device creation"""
284         self.cb_location_d.clear()
285         with CSession() as session:
286             locations = session.query(classes.Location).all()
287             locations.sort(key=lambda location: str(location))
288             for location in locations:
289                 self.cb_location_d.addItem(location.name)
290
291     def set_combobox_producer_a(self):
292         """Fill Producer ComboBox for Article creation"""
293         self.cb_producer_a.clear()
294         with CSession() as session:
295             producers = session.query(classes.Producer).all()
296             producers.sort(key=lambda producer: str(producer.name))
297             for producer in producers:
298                 self.cb_producer_a.addItem(producer.name)
299
300     def set_combobox_producer_d(self):
301         """Fill Producer ComboBox for Device creation"""
302         self.cb_producer_d.clear()
303         self.cb_producer_d.addItem("")
304         with CSession() as session:
305             producers = session.query(classes.Producer).all()
306             producers.sort(key=lambda producer: str(producer.name))
307             for producer in producers:
308                 self.cb_producer_d.addItem(producer.name)
309
310     def set_combobox_article_d(self):
311         """Fill Article ComboBox for Device creation"""
312         self.cb_article_d.clear()
313         with CSession() as session:

```

```

314         articles = session.query(classes.Article).all()
315         for article in articles:
316             self.cb_article_d.addItem(article.name)
317
318     def reload_combobox_article_d(self):
319         """Change article depending on selected Producer"""
320         self.cb_article_d.clear()
321         with CSession() as session:
322             articles = session.query(classes.Article).all()
323             producers = session.query(classes.Producer).all()
324             selected_producer = self.cb_producer_d.currentText()
325             if selected_producer:
326                 for producer in producers:
327                     if producer.name == selected_producer:
328                         self.producerXY = producer.uid
329                         print (self.producerXY)
330                     for article in articles:
331                         if article.producer_uid == self.producerXY:
332                             self.cb_article_d.addItem(article.name)
333             else:
334                 for article in articles:
335                     self.cb_article_d.addItem(article.name)
336
337     def set_combobox_device_user(self):
338         """Fill User ComboBox for QR-Code readings"""
339         self.cb_device_user.clear()
340         self.cb_device_user.addItem("")
341         with CSession() as session:
342             users = session.query(classes.User).all()
343             for user in users:
344                 self.cb_device_user.addItem(f"{user.uid} {str(user)}")
345
346     def set_combobox_device_location(self):
347         """Fill Location ComboBox for QR-Code readings"""
348         self.cb_device_location.clear()
349         self.cb_device_location.addItem("")
350         with CSession() as session:
351             locations = session.query(classes.Location).all()
352             locations.sort(key=lambda location: str(location))
353             for location in locations:

```

```

354             self.cb_device_location.addItem(location.name)
355
356     def set_combobox_user_d(self):
357         """Fill User ComboBox for Device creation"""
358         self.cb_user_d.clear()
359         self.cb_user_d.addItem("")
360         with CSession() as session:
361             users = session.query(classes.User).all()
362             for user in users:
363                 self.cb_user_d.addItem(f"{user.uid}\u2022{str(user)}")
364
365     def set_combobox_user_admin(self):
366         """Fill User ComboBox for User change"""
367         self.cb_user_admin.clear()
368         self.cb_user_admin.addItem("")
369         with CSession() as session:
370             users = session.query(classes.User).all()
371             for user in users:
372                 self.cb_user_admin.addItem(f"{user.uid}\u2022{str(user)}")
373
374     def reload_user_change(self):
375         """Fill TextBoxes for User change"""
376         with CSession() as session:
377             users = session.query(classes.User).all()
378             selected_user = self.cb_user_admin.currentText()
379             if selected_user == "":
380                 self.in_name_admin.setText("")
381                 self.in_surname_admin.setText("")
382                 self.in_email_admin.setText("")
383                 self.in_phone_admin.setText("")
384             return
385             user_uid = int(selected_user.split("\u2022")[0])
386             if user_uid == self.logged_in_user.uid:
387                 self.checkBox.setEnabled(False)
388             else:
389                 self.checkBox.setEnabled(True)
390
391             for user in users:
392                 if user.uid == user_uid:

```

```

393                     self.in_name_admin.setText(user.name) # fill \
394                         textEdits for UserChange
395                     self.in_surname_admin.setText(user.surname)
396                     self.in_email_admin.setText(user.e_mail)
397                     self.in_phone_admin.setText(str(user.phonenumber))
398                     self.cb_location_u_admin.setCurrentIndex(user.\
399                         location_uid)
400
401                     if user.is_admin:
402                         self.checkBox.setCheckState(2)
403                     else:
404                         self.checkBox.setCheckState(0)
405
406             def set_phonenumber(self, str_):
407                 """Set reference PhoneNumber display on User creation tab
408                 to PhoneNumber represantation of str_
409                 Args:
410                     str_(str): str_ to try and interpret as PhoneNumber
411                     """
412                     if str_:
413                         try:
414                             text = str(classes.PhoneNumber(str_))
415                         except classes.PhoneNumber.NoNumberFoundWarning:
416                             text = "Es\u00d7konnte\u00d7keine\u00d7Nummer\u00d7erkannt\u00d7werden.\u00d7\
417                                 Empfohlene\u00d7Syntax\u00d7ist:\u00d7Vorwahl\u00d7Benutzernummer-\u00d7
418                                 Durchwahl"
419                     else:
420                         text = "Bitte\u00d7geben\u00d7Sie\u00d7Ihre\u00d7Telefonnummer\u00d7ein."
421                     self.out_phone.setText(text)
422                     self.out_phone_admin.setText(text)
423
424             def b_user_login_click(self):
425                 """Login button click"""
426                 LoginDialog(self).exec()
427                 self.update_user_dependant()
428                 if self.logged_in_user:
429                     # logger.info(f"Logged in as {self.logged_in_user}") # \
430                         already logged in login
431                     self.timeout = classes.Timeout(
432                         config["timeout"]*60,

```

```

428         lambda signal: signal.emit(True),
429         self.ui.b_user_logout.clicked)
430     self.timeout.start()
431
432 def b_user_logout_click(self, timed_out=False):
433     """Logout button click - also handles timeouts via timed_out \
434     flag
435     Args:
436         timed_out(bool): Set to True to handle logout as one \
437             rooted in timeout
438     """
439     if timed_out:
440         self.timed_out()
441     self.logged_in_user = None # may want to use slots.logout if \
442         that does something eventually
443     self.update_user_dependant()
444     self.timeout.function = None
445     del self.timeout
446     self.in_name.setText("")
447     self.in_surname.setText("")
448     self.in_email.setText("")
449     self.in_phone.setText("")
450
451     def timed_out(self):
452         logger.info(f"User {self.logged_in_user.uid} logged out due to \
453             inactivity")
454
455         self.status_bar_text("Sie wurden wegen Inaktivität automatisch \
456             ausgeloggt!", 5, "red")
457
458         self.in_name.setText("")
459         self.in_surname.setText("")
460         self.in_email.setText("")
461         self.in_phone.setText("")
462
463         messagebox = QtWidgets.QMessageBox()
464         messagebox.setIcon(QtWidgets.QMessageBox.Information)
465         messagebox.setWindowTitle("Automatisch ausgeloggt")
466         messagebox.setText("Sie wurden wegen Inaktivität automatisch \
467             ausgeloggt!")

```

```

462     messagebox.setStandardButtons(QtWidgets.QMessageBox.Ok)
463     messagebox.exec_()
464
465     def update_user_dependant(self):
466         if self.logged_in_user:
467             self.ui.log_in_out.setCurrentIndex(1)
468             self.label.setText(str(self.logged_in_user))
469             self.status_bar_text(f"Sie sind jetzt als {self.\
470                 logged_in_user} angemeldet", 2, "green")
471
472             self.in_name.setText(self.logged_in_user.name)
473             self.in_surname.setText(self.logged_in_user.surname)
474             self.in_email.setText(self.logged_in_user.e_mail)
475             with CSession() as session:
476                 users = session.query(classes.User).all()
477                 userX = self.logged_in_user.name
478                 for user in users:
479                     if user.name == userX:
480                         self.in_phone.setText(str(user.phonenumber))
481
482             if self.logged_in_user.is_admin:
483                 if self.ui.stackWidget.currentIndex() == 4:
484                     self.ui.stackWidget.setCurrentIndex(3)
485
486             else:
487                 self.ui.log_in_out.setCurrentIndex(0)
488                 self.label.setText("")
489                 if self.ui.stackWidget.currentIndex() == 3:
490                     self.ui.stackWidget.setCurrentIndex(4)
491
492     def mousePressEvent(self, QMouseEvent=None):
493         """Event that's called on mouse click"""
494         # print("Maus geklickt")
495         self.mouseMoveEvent()
496
497     def mouseMoveEvent(self, QMouseEvent=None):
498         """Event that's called on mouse move"""
499
500         if QMouseEvent is not None:
501             print("Maus bewegt")"""

```

```

501         if hasattr(self, "timeout"):
502             self.timeout.reset()
503
504     def b_user_change_click(self):
505         name = self.in_name.text()
506         surname = self.in_surname.text()
507         e_mail = self.in_email.text()
508         phonenumerber = self.in_phone.text()
509         location = self.cb_location_u.currentText()
510         password_1 = self.in_password1.text()
511         password_2 = self.in_password2.text()
512         if password_1 == "" and self.logged_in_user:
513             password_1 = False # leave password as it
514             password_2 = False
515
516         locals_ = {key: value for (key, value) in locals().items() if \
517                     key != "self"}
518         if "" in locals_.values():
519             self.not_all_fields_filled_notice()
520             return
521
522         if password_1 != password_2:
523             self.status_bar_text("Die Passwörter stimmen nicht überein", 5, "red")
524             return
525         else:
526             password = password_1
527
528         if self.logged_in_user:
529             user = self.logged_in_user
530         else:
531             user = classes.User(e_mail, password)
532
533         with CSession() as session:
534             session.add(user)
535             location = session.query(classes.Location).filter_by(name=location).first()
536             user.name = name
537             user.surname = surname
538             user.e_mail = e_mail

```

```

538     user.phonenumber = classes.PhoneNumber(phonenumber)
539     user.location = location
540     if self.logged_in_user and password:
541         user.hash(password)
542
543     if self.logged_in_user:
544         self.logged_in_user = user
545         logger.info(f"User {user} changed through UI")
546         self.status_bar_text(f"Benutzer {user} wurde erfolgreich \\\
547             geÄndert", 5, "green")
548     else:
549         self.status_bar_text(f"Benutzer {user} wurde erfolgreich \\\
550             angelegt", 5, "green")
551         logger.info(f"User {user} created through UI")
552
553     def b_user_change_admin_click(self):
554         user_to_change = self.cb_user_admin.currentText()
555         name = self.in_name_admin.text()
556         surname = self.in_surname_admin.text()
557         e_mail = self.in_email_admin.text()
558         phononenumber = self.in_phone_admin.text()
559         location = self.cb_location_u_admin.currentText()
560         locals_ = {key: value for (key, value) in locals().items() if \
561             key != "self"}
562         if "" in locals_.values():
563             self.not_all_fields_filled_notice()
564             return
565
566         user_to_change_uid = user_to_change.split(" ")[0]
567         with CSession() as session:
568             user = session.query(classes.User).filter_by(uid=\
569                 user_to_change_uid).first()
570             location = session.query(classes.Location).filter_by(name=\
571                 location).first()
572             user.name = name
573             user.surname = surname
574             user.e_mail = e_mail
575             user.phonenumber = classes.PhoneNumber(phonenumber)
576             user.location = location
577             user.is_admin = self.checkBox.isChecked()

```

```

573         session.add_all((user, location))
574
575     if self.logged_in_user.uid == user.uid:
576         self.logged_in_user = user
577     logger.info(f"User {user} changed through UI by admin {self.\n        logged_in_user.uid}")
578     self.status_bar_text(f"Benutzer {user} wurde erfolgreich \\\n        geÄndert", 5, "green")
579
580 def reset_password(self):
581     """Set new password and salt for user, push it to db and show \
582         it in UI"""
583     user_to_change = self.cb_user_admin.currentText()
584     password = self.in_password_admin.text()
585     if password == "":
586         password = False # autogenerate password
587     locals_ = {key: value for (key, value) in locals().items() if \
588                 key != "self"}
589     if "" in locals_.values():
590         self.not_all_fields_filled_notice()
591     return
592
593     user_to_change_uid = user_to_change.split(" ")[0]
594     with CSsession() as session:
595         user = session.query(classes.User).filter_by(uid=\
596             user_to_change_uid).first()
597         session.add(user)
598         if password:
599             user.hash(password)
600         else:
601             password = slots.reset_password(user)
602
603         self.in_password_admin.setText("")
604         messagebox = QtWidgets.QMessageBox()
605         messagebox.setIcon(QtWidgets.QMessageBox.Information)
606         messagebox.setWindowTitle(f"Neues Passwort fÄr User {user.uid}\n        ")
607         messagebox.setText(f"Das neue Passwort fÄr {user} ist \\\n            password")
608         messagebox.setStandardButtons(QtWidgets.QMessageBox.Ok)

```

```

606         messagebox.exec_()
607
608     def b_save_device_click(self):
609         """Dialog to choose path for qr-codes"""
610         qr_path = QtWidgets.QFileDialog.getExistingDirectory(self, "\\\n        Bitte wÄhlen Sie ein Verzeichnis")
611         if qr_path:
612             self.t_path_device.setText(qr_path)
613
614     def b_change_device_click(self):
615         """Change Responsibility of a Device"""
616         new_location = self.cb_device_location.currentText() # gives \
617             location.name
618         new_user = self.cb_device_user.currentText()      # gives 'user.\
619             uid user.surname user.name'
620         user_uid = int(new_user.split(" ")[0])
621         device = int(self.t_code_device.text().split(" ")[-1])
622
623         locals_ = {key: value for (key, value) in locals().items() if \
624             key != "self"}
625         if "" in locals_.values():
626             self.not_all_fields_filled_notice()
627             return
628
629         with CSession() as session:
630             resp = session.query(classes.Responsibility).join(classes.\
631                 Device).filter_by(uid=device).first()
632             location = session.query(classes.Location).filter_by(name=\
633                 new_location).first()
634             user = session.query(classes.User).filter_by(uid=user_uid)\.
635                 .first()
636             if self.logged_in_user:
637                 if user.uid != self.logged_in_user.uid and not self.\
638                     logged_in_user.is_admin:
639                     self.status_bar_text("Um GerÄte einem anderen \
640                         Nutzer zuzuweisen ist ein Administrator nÃstig\\
641                         !", 5, "red")
642                     return
643             else:

```

```

635         self.status_bar_text("Um GerÄtete\u00e4inem\u00e4nderen\u00e4 Nutzer\u00e4\
636             zuzuweisen\u00e4 ist\u00e4 ein\u00e4 Administrator\u00e4 nÄstig!", 5, "red\
637             ")
638             return
639             resp.location = location
640             resp.user = user
641             logger.info(f"Modified\u00e4 Responsibility\u00e4 for\u00e4 Device\u00e4{resp.\\
642                 device.uid}")
643             self.status_bar_text(f"Verantwortlichkeit\u00e4 fÄzr\u00e4 GerÄt\u00e4{\
644                 resp.device.uid}\u00e4 wurde\u00e4 bearbeitet", 5, "green")
645
646     def b_delete_device_click(self):
647         """Delete device from database"""
648         device = int(self.t_code_device.text().split("\u00a0")[-1])
649         if device == "":
650             self.not_all_fields_filled_notice()
651             return
652         if not self.logged_in_user:
653             self.status_bar_text("Um GerÄtete\u00e4 zu\u00e4 lÄsschen\u00e4 mÄsssen\u00e4 Sie\u00e4\
654                 angemeldet\u00e4 sein", 5, "red")
655             return
656         with CSession() as session:
657             resp = session.query(classes.Responsibility).join(classes.\\
658                 Device).filter_by(uid=device).first()
659             session.delete(resp.device)
660             session.delete(resp)
661             logger.info(f"Deleted\u00e4 Device\u00e4{resp.device.uid}")
662             self.status_bar_text(f"GerÄt\u00e4{resp.device.uid}\u00e4 wurde\u00e4 aus\u00e4\
663                 der\u00e4 Datenbank\u00e4 entfernt", 5, "green")
664             self.t_code_device.setText("")
665             self.t_code_user.setText("")
666             self.t_code_location.setText("")
667             self.set_tree()

668     def b_create_device_click(self): # todo: handle if logged in user \
669         is no admin and can't create devices for others
670         article = self.cb_article_d.currentText()
671         location = self.cb_location_d.currentText()
672         user = self.cb_user_d.currentText()
673         path = self.t_path_device.text()

```

```

667     locals_ = {key: value for (key, value) in locals().items() if \
668         key != "self"}
669     if "" in locals_.values():
670         self.not_all_fields_filled_notice()
671         return
672
672     path = pathlib.Path(path)
673     user_uid = int(user.split(" ")[0])
674     with CSession() as session:
675         article = session.query(classes.Article).filter_by(name=\
676             article).first()
676         user = session.query(classes.User).filter_by(uid=user_uid) \
677             .first()
677         location = session.query(classes.Location).filter_by(name=\
678             location).first()
678         if self.logged_in_user:
679             if user.uid != self.logged_in_user.uid and not self. \
680                 logged_in_user.is_admin:
680                 self.status_bar_text("Um GerÄte fÄr einen \u
681                     anderen Nutzer zu erzeugen ist ein \u
682                     Administrator nÄstig!", 5, "red")
683             else:
683                 self.status_bar_text("Um GerÄte zu erzeugen mÄssen \u
684                     Sie eingeloggt sein", 5, "red")
685             return
685         device = classes.Device()
686         session.add(device)
687         device.article = article
688         resp = classes.Responsibility(device, user, location)
689         session.add(resp)
690         path = pathlib.Path(self.t_path_device.text())
691         if not re.match(r"\.(?P<filetype>.*$)", str(path), re. \
692             IGNORECASE):
692             path /= utils.normalize_filename(f"{device.uid}_{device. \
693                 article.name}.svg")
693         if not utils.check_if_file_exists(path):
694             self.status_bar_text(f"{path} ist kein gÄltiger Pfad/eine \u
695                     bereits vorhandene Datei", 5, "red")
695         return

```

```

696     try:
697         generate_qr(device, path)
698     except NotImplementedError as e:
699         logger.error(str(e))
700         self.status_bar_text(f"Um {e[1]} Dateien zu speichern sind \
701             weitere Pakete nötig. Das Standartformat ist svg", \
702             10, "red")
703     else:
704         self.status_bar_text(f"Gerät erfolgreich angelegt. Der QR \
705             -Code wurde unter {path} gespeichert", 10, "green")
706
707
708     def b_create_new_qrcode_click(self):
709         """Creates a new QR-Code for a device"""
710         device = self.ui.treeWidget.currentItem().text(0)
711         try:
712             device_uid = re.match(r"(.* mit ID) (?P<ID>\d+)$", \
713             device).group("ID")
714         except AttributeError:
715             self.status_bar_text("Es konnte leider kein Gerät erkannt \
716             werden", 8, "red")
717         return
718
719         with CSession() as session:
720             device = session.query(classes.Device).filter_by(uid=\
721                 device_uid).first()
722             filename = utils.normalize_filename(f"{device.uid}_{device \
723                 .article.name}.svg")
724             path = pathlib.Path(config["qr_path"])
725             path /= filename
726             if not utils.check_if_file_exists(path):
727                 self.status_bar_text(f"{path} ist kein gültiger Pfad / \
728                     eine bereits vorhandene Datei", 5, "red")
729             return
730         try:
731             generate_qr(device, path)
732         except NotImplementedError as e:

```

```

728         logger.error(str(e))
729         self.status_bar_text(f"Um {e[1]} Dateien zu speichern\"
730             sind weitere Pakete n\u00e4stig. Das Standartformat ist \
731            (svg", 10, "red")
732     else:
733         self.status_bar_text(f"Der QR-Code wurde unter {config\"
734             ['qr_path']} gespeichert", 8, "green")
735
736
737     def b_create_article_click(self):
738         name = self.t_name_a.text()
739         producer_name = self.cb_producer_a.currentText()
740         locals_ = {key: value for (key, value) in locals().items() if \
741             key != "self"}
742         if "" in locals_.values():
743             self.not_all_fields_filled_notice()
744             return
745         with CSession() as session:
746             producer = session.query(classes.Producer)\.
747                 filter_by(name=producer_name).first()
748         try:
749             slots.create_article(name=name, producer=producer)
750         except classes.IntegrityError as e:
751             logger.info(str(e))
752             self.status_bar_text(f'Artikel mit Name "{name}" existiert \
753                 bereits', 5, "red")
754         else:
755             self.status_bar_text(f'Artikel "{name}" wurde angelegt', \
756                 5, "green")
757             self.t_name_a.setText("")
758             self.set_combobox_article_d()
759
760     def b_create_producer_click(self):
761         name = self.t_name_p.text()
762         try:
763             slots.create_producer(name=name)
764         except classes.IntegrityError as e:
765             logger.info(str(e))
766             self.status_bar_text(f'Produzent mit Namen "{name}" \
767                 existiert bereits', 5, "red")

```

```

761     else:
762         self.status_bar_text(f'Produzent "{name}" wurde angelegt', \
763                             5, "green")
764         self.t_name_p.setText("")
765         self.set_combobox_producer_d()
766         self.set_combobox_producer_a()
767
767 def not_all_fields_filled_notice(self):
768     """Show message that user hasn't filled all necessary fields
769     This could also be reworked to use signals and slots
770     """
771     self.status_bar_text("Bitte füllen Sie alle Felder aus", 5, "\\
772                             red")
773
773 def recognized_barcode(self, str_):
774     """Slot that's called if the camera recognized a barcode"""
775     logger.info(f"Recognized barcode: {str_}")
776     try:
777         match = re.search(r"id=(?P<id>\d+).*", str_)
778         # Could also match on r"id=(?P<id>\d+).*name=(?P<name>.*)"\
779         # to only recognize codes that contain name and uid
780         uid = match.group("id")
781     except AttributeError:
782         logger.info("Tried scanning external code/code with wrong\
783                     data")
784     return
785     with CSession() as session:
786         resp = session.query(classes.Responsibility).join(classes.\
787                         Device).filter_by(uid=uid).first()
788         if resp is None:
789             logger.info(f"Scanned device is not in Database {uid}"\
790                         )
791         return
792         self.t_code_device.setText(str(resp.device))
793         self.t_code_user.setText(str(resp.user))
794         self.t_code_location.setText(str(resp.location))
795         self.status_bar_text("Barcode erkannt", 2, "green")
796     logger.info("Successfully processed barcode")
797
798

```

```

795 class LoginDialog(QtWidgets.QDialog):
796
797     def __init__(self, parent=None):
798         path = utils.absolute_path("login.ui")
799         super().__init__(parent)
800         self.parent = parent
801         self.ui = uic.loadUi(path, self)
802         self.b_login.clicked.connect(self.b_login_click)
803         self.b_password_lost.clicked.connect(self.\
804             b_password_lost_click)
805
806     def b_login_click(self):
807         username = self.t_username.text()
808         password = self.t_password.text()
809         try:
810             self.parent.logged_in_user = slots.login(username, \
811                 password)
812         except ValueError:
813             self.parent.status_bar_text(f'Benutzer "{username}" ist "\\
814                 nicht bekannt', 5, "red")
815         self.close()
816
817
818 class ResetDialog(QtWidgets.QDialog):           #Dialog to select a \
819     file path for password reset
820     filepath = ""
821
822     def __init__(self, parent=None):
823         path = utils.absolute_path("reset.ui")
824         super().__init__(parent)
825         self.parent = parent
826         self.ui = uic.loadUi(path, self)
827         self.b_browse.clicked.connect(self.b_browse_click)
828         self.b_password_reset.clicked.connect(self.\
829             b_password_reset_click)
830         self.b_password_close.clicked.connect(self.\
831             b_password_close_click)
832         self.t_path.setText(self.filepath)

```

```

829         self.ui.b_browse.setIcon(QtGui.QIcon("pictures/folder.png"))
830         self.ui.b_browse.setIconSize(QtCore.QSize(20,20))
831
832     def b_browse_click(self):
833         self.filepath = QtWidgets.QFileDialog.getOpenFileName(self, '\
834             SingleFile')
835         self.t_path.setText(self.filepath[0])
836
837     def b_password_reset_click(self):
838         user = self.t_user.text()
839         path = self.t_path.text()
840         if "" in (user, path):
841             self.close()
842             self.parent.parent.not_all_fields_filled_notice()
843             ResetDialog(self.parent).exec()
844             return
845
846         private_path = pathlib.Path(path)
847         public_path = keys.PUBLIC_KEY_PATH
848         with CSession() as session:
849             user = session.query(classes.User).filter_by(e_mail=user).\
850                 first()
851             session.add(user)
852             if not user:
853                 self.parent.parent.status_bar_text("Unbekannter\ \
854                     Benutzer!", 5, "red")
855             return
856         try:
857             password = slots.reset_admin_password(user, \
858                 public_path, private_path)
859         except ValueError as e:
860             logger.info(str(e))
861             self.parent.parent.status_bar_text("Ungültiger\ \
862                 Schlüssel!", 5, "red")
863             return
864         except FileNotFoundError as e:
865             logger.info(str(e))
866             self.parent.parent.status_bar_text(f"Keine gültige\ \
867                 Datei unter {private_path}!", 5, "red")
868             return

```

```

863
864     messagebox = QtWidgets.QMessageBox()
865     messagebox.setIcon(QtWidgets.QMessageBox.Information)
866     messagebox.setWindowTitle(f"Neues Passwort fÃ¼r User {user.uid}\n")
867     messagebox.setText(f"Das neue Passwort fÃ¼r {user} ist\n"
868     messagebox.setStandardButtons(QtWidgets.QMessageBox.Ok)
869     messagebox.exec_()
870
871     self.close()
872     self.parent.close()
873
874     def b_password_close_click(self):
875         self.close()
876
877
878 if __name__ == "__main__":
879     app = QtWidgets.QApplication(sys.argv)
880     dialog_main = MainDialog()
881     dialog_login = LoginDialog()
882     dialog_save = ResetDialog()
883
884     dialog_main.show() # show dialog_main as modeless dialog => return\
885     control back immediately
886     sys.exit(app.exec_())

```

B.11 unittests.py

```

1 import os
2 import random
3 import unittest
4
5
6 import classes
7 import utils
8
9
10 class TestUtils(unittest.TestCase):

```

```

11     def test_normalize_filename(self):
12         self.assertEqual(utils.normalize_filename(
13             "123abd.edg/(&(&%Â§!14\uuuug\n1gj2141.pdf"),
14             "123abd.edg4---gj2141.pdf"))
15
16     def test_check_if_file_exists(self):
17         self.assertTrue(utils.check_if_file_exists("abc"))
18         with open("abc", "w") as f:
19             self.assertFalse(utils.check_if_file_exists("abc"))
20         os.remove("abc")
21
22
23 class TestTelephoneNumber(unittest.TestCase):
24     @classmethod
25     def phone_number(cls, string):
26         try:
27             pn = classes.PhoneNumber(string)
28         except Exception as e:
29             raise e
30         else:
31             return str(pn)
32
33     def test_basic_function(self):
34         self.assertEqual(self.phone_number("+049\u9723\u1234-123"), "\u00b7
35                                         +049\u9723\u1234-123")
36
37     def test_non_conforming_input(self):
38         self.assertEqual(self.phone_number("09723\u1234"), "+049\u9723\u1234")
39         self.assertEqual(self.phone_number("09723\u1234\u56789-10"), "\u00b7
40                                         +049\u9723\u123456789-10")
41         self.assertEqual(self.phone_number("09723\u1234\u56789+10"), "\u00b7
42                                         +049\u9723\u123456789-10")
43
44     def test_text_input(self):
45         self.assertEqual(self.phone_number("My\u00a0telephonenumber\u00a0is:\u00a0\
46                                         09723\u1234.\u00a0Yes."), "+049\u9723\u1234")
47         self.assertEqual(self.phone_number("My\u00a0number\u00a0is\u00a00\u9723\u1234+1\u00a0\
48                                         and\u00a0yours\u00a0is\u00a009723\u1234"), "+049\u9723\u1234-1")

```

```

45
46 if __name__ == "__main__":
47     try:
48         unittest.main()
49     except SystemExit:
50         pass

```

B.12 utils.py

```

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

```

```

31         print(val)
32         cls.print_.task_done()
33
34     @classmethod
35     def __call__(cls):
36         raise ResourceWarning(f"{cls.__class__} should only be \
37             instantiated once!")
38
39 _ParallelPrint = _ParallelPrint()
40 _ParallelPrint.start()
41 parallel_print = _ParallelPrint.print_.put
42
43
44 def check_if_file_exists(path):
45     """Check whether a file already exists, create directories in path\
46         that don't exist yet"""
47     try:
48         with open(path, "x"):
49             pass
50     except FileExistsError as e:
51         return False
52     except FileNotFoundError:
53         path.parents[0].mkdir(parents=True)
54         return True
55     else:
56         os.remove(path)
57     return True
58
59 def normalize_filename(string):
60     """Remove all non ASCII letters and digits from a string and
61     replace whitespaces with underscores keeping dots untouched"""
62     string = umlaut_converter(string)
63     segs = string.split(".")
64     for i, seg in enumerate(segs):
65         segs[i] = re.sub(r"\s", "_", segs[i])
66         segs[i] = re.sub(r"\W", "", seg, flags=re.ASCII)
67     return ".".join(segs)
68

```

```

69
70 def umlaut_converter(string):
71     """Convert all umlauts to their e-equivalent"""
72     return string.replace("Ã¤", "ae").replace("Ãš", "oe").replace("Ãž" \
        , "ue")

```

B.13 main.rs

```

1 use std::process::Command;
2
3 fn main() {
4
5     // execute from "launcher/" or change path accordingly
6     let output = if cfg!(target_os = "windows") {
7         Command::new("cmd")
8             .args(&["/C"]) // "/Q" hides terminal but also prevents Qt\
9                 -GUI from showing
10            .arg("python..../main.py")
11            .output()
12            .expect("Failed to execute process");
13    } else {
14        Command::new("sh")
15            .arg("-c")
16            .arg("python..../main.py")
17            .output()
18            .expect("Failed to execute process");
19    };
20    println!("{:?}", output);
21 }

```

B.14 cargo.toml

```

1 [package]
2 name = "launcher"
3 version = "0.1.0"
4 authors = ["Stefan Volz <volzstefan97@googlemail.com>"]
5 edition = "2018"
6
7 [dependencies]

```

C Automatisch generierte Dokumentation

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

D Literatur- und Quellenverzeichnis

Literatur

- [Bay18] BAYER, Michael: SQLAlchemy. In: *The Architecture of Open Source Applications 2* (2018). <http://aosabook.org/en/sqlalchemy.html>
- [BO18] BLANDY, Jim ; ORENDORFF, Jason ; BLEIEL, Jeff (Hrsg.): *Programming Rust: Fast, Safe Systems Development.* 2. O'Reilly Media, 2018 <http://shop.oreilly.com/product/0636920040385.do>. – ISBN 978-1-491-92728-1
- [EK17] ERNEST, Johannes ; KAISER, Peter ; SCHEIBE, Anne (Hrsg.): *Python 3 - Das umfassende Handbuch.* 5. Rheinwerk Verlag, 2017 https://www.rheinwerk-verlag.de/python-3_4467/. – ISBN 978-3-8362-5864-7
- [Goo01] GOODGER, David: *Docstring Conventions*, 2001. <https://www.python.org/dev/peps/pep-0257/>
- [Goo18] GOOGLE: *Google Python Style Guide*, 2018. <http://google.github.io/styleguide/pyguide.html>
- [Gos19] GOSH, Sumit: Demystifying @decorators in Python. In: *Musings of Sumit Gosh* (2019). <https://sumit-ghosh.com/articles/demystifying-decorators-python/>
- [Het15] HETTINGER, Raymond: *Beyond PEP 8 – Best practices for beautiful intelligible code*, 2015. <https://youtu.be/wf-BqAjZb8M>
- [Het16] HETTINGER, Raymond: *Thinking about Concurrency*, 2016. <https://youtu.be/Bv25Dwe84g0>
- [Het17] HETTINGER, Raymond: Descriptor HowTo Guide. In: *Python Documentation* (2017). <https://docs.python.org/3.3/howto/descriptor.html>
- [KB18] KRYPCZYK, Veikko ; BOCHKOR, Olena ; POLL, Almut (Hrsg.) ; BECHTOLD, Simone (Hrsg.): *Handbuch fuer Softwareentwickler.* 1. Rheinwerk Verlag, 2018 https://www.rheinwerk-verlag.de/handbuch-fur-softwareentwickler_4329/. – ISBN 978-3-8362-4476-3
- [Pyt18a] PYTHON SOFTWARE FOUNDATION: *dis - Disassembler for Python bytecode*, 2018. <https://docs.python.org/3/library/dis.html>

- [Pyt18b] PYTHON SOFTWARE FOUNDATION: www.python.org, 2018.
<https://www.python.org/>
- [Pyt19a] PYTHON SOFTWARE FOUNDATION: *collections - Container datatypes*, 2019.
<https://docs.python.org/3/library/collections.html>
- [Pyt19b] PYTHON SOFTWARE FOUNDATION: *re - Regular expression operations*, 2019.
<https://docs.python.org/3/library/re.html>
- [Rei18] REITZ, Kenneth: Picking a Python Interpreter (3 vs 2). In: *The Hitchhiker's Guide to Python* (2018). <https://docs.python-guide.org/starting/which-python/>
- [Ros01] ROSSUM, Guido van: *Style Guide for Python Code*, 2001.
<https://www.python.org/dev/peps/pep-0008/>
- [Ros17] ROSSUM, Guido van: Whats New In Python 3.0. In: *Python Documentation* (2017).
<https://python.readthedocs.io/en/stable/whatsnew/3.0.html>
- [SQL18] SQLALCHEMY: *Session Basics*, 2018. https://docs.sqlalchemy.org/en/latest/orm/session_basics.html
- [Ste17] STEVENS, H. C.: The decorators they won't tell you about. (2017). <https://github.com/hchasesteven/hchasesteven.github.io/blob/master/notebooks/the-decorators-they-wont-tell-you-about.ipynb>
- [UBDK17] UNIVERSITY OF LUXEMBOURG ; BIRYUKOV, Alex ; DINU, Daniel ; KHOVRATOVICH, Dmitry: *Argon2: the memory-hard function for password hashing and other applications*, März 2017. <https://github.com/P-H-C/phc-winner-argon2/blob/master/argon2-specs.pdf>
- [Wik18a] WIKIPEDIA: *Race Condition*, 2018. https://de.wikipedia.org/wiki/Race_Condition
- [Wik18b] WIKIPEDIA: *Rufnummer*, 2018. <https://de.wikipedia.org/wiki/Rufnummer>
- [Wik19a] WIKIPEDIA: *Argon2*, 2019. <https://de.wikipedia.org/wiki/Argon2>
- [Wik19b] WIKIPEDIA: *Salt (Kryptologie)*, 2019. [https://de.wikipedia.org/wiki/Salt_\(Kryptologie\)](https://de.wikipedia.org/wiki/Salt_(Kryptologie))

E Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und dabei keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Zitate wurden gemäß dem „Leitfaden für die Durchführung und Gestaltung der Technikerarbeit“ der Franz-Oberhür-Schule Würzburg als solche gekennzeichnet.

Unterschrift : Stefan Volz

Datum

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und dabei keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Zitate wurden gemäß dem „Leitfaden für die Durchführung und Gestaltung der Technikerarbeit“ der Franz-Oberhür-Schule Würzburg als solche gekennzeichnet.

Unterschrift : Yannis Köhler

Datum