Inhaltsverzeichnis

Projekt Lagersystem Technische Dokumentation

1.Lastenheft

Pro Storage Systems ist ein firmeninternes Lagerhaltungsprogramm. Es vereinfacht das Lagern und vor allem das Wiederfinden und Überwachen von Lagergegenständen die im weiteren als "Items" benannt werden.

Beim Lagersystem kann der Lagerhalter Items unter Angabe eines nicht unbedingt eindeutigen Namen, eines Standorts und einer Anzahl in das System einfügen.
Des weiteren kann er angeben ob dass Objekt automatisch auf die Bestellliste gesetzt wird falls die Anzahl = 0 erreicht.
Der Lagerhalter kann ,unter Angabe eines Namen ,ein neues Lager/ein neues Regal/ein neues Abteil erstellen.
Der Lagerhalter kann die Anzahl eines Items erhöhen oder verringern. Des weiteren kann er die beim Erstellen angegeben Parameter ändern.
Jeder Benutzer kann alle Items einsehen, nach bestimmten Items suchen und diese bearbeiten. Gleiches gilt für die Lager, Regale und Abteile.
Alle Lagerhalter greifen durch das firmeninterne System auf das Lager zu und haben alle gleiche Rechte. Es gibt keine Benutzerprofile.

2.Pflichtenheft

2.1 ITEMVERWALTUNG (A)

2.1.1 Items erstellen

A1 Das System soll es dem Benutzer ermöglichen ein neues Item zu erstellen.
A1.1 Wenn der Benutzer bei der Erstellung eines Items eine ungültige Eingabe macht soll das System eine Fehlermeldung anzeigen

2.1.2 Itemanzahl

A2 Das System soll es dem Benutzer ermöglichen die Anzahl der Items zu erhöhen.
A2.1 Das System soll es dem Benutzer ermöglichen die Anzahl der Items zu verringern.

2.1.3 Beschreibung verwalten

A3 Das System soll es dem Benutzer ermöglichen die Beschreibung des Items zu bearbeiten.

2.1.4 Autobestellung verwalten

A4 Das System soll es dem Benutzer ermöglichen die Autobestellung an- und auszuschalten.

2.1.5 Standort verwalten

A5 Das System soll es dem Benutzer ermöglichen den Standort von einem Item zu ändern.

2.1.6 Item löschen

A6 Das System soll es dem Benutzer ermöglichen ein Item zu löschen.

2.1.7 Items suchen

A7 Das System soll es dem Benutzer ermöglichen ein Item unter Angabe der Item-ID im System zu suchen.

A8 Das System soll es dem Benutzer ermöglichen ein Item unter Angabe des Itemnamen im System zu suchen.

2.1.8 Items einsehen

A9 Das System soll es dem Benutzer ermöglichen alle Items im System einzusehen.

2.2 LAGERVERWALTUNG (B)

2.2.1 Lager verwalten

B1 Das System soll es dem Benutzer ermöglichen ein Lager zu erstellen.

B1.1 Das System soll es dem Benutzer ermöglichen ein Lager zu löschen.

B1.2 Wenn der Benutzer bei der Erstellung eines Lagers eine ungültige Eingabe macht, soll das System eine Fehlermeldung anzeigen.

2.2.2 Lager einsehen

B2 Das System soll es dem Benutzer ermöglichen alle Lager im System einzusehen.

2.2.3 Regal verwalten

B3 Das System soll es dem Benutzer ermöglichen ein Regal zu erstellen.

B3.1 Das System soll es dem Benutzer ermöglichen ein Regal zu löschen.

B3.2 Das System soll es dem Benutzer ermöglichen ein Regal einem Lager zuzuordnen.

B3.3 Wenn der Benutzer bei der Erstellung eines Regals eine ungültige Eingabe macht, soll das System eine Fehlermeldung anzeigen.

2.2.4 Regale einsehen

B4 Das System soll es dem Benutzer ermöglichen alle Regale, zusammen mit den dazugehörigen Lagern im System einzusehen.

2.2.5 Abteil verwalten

B5 Das System soll es dem Benutzer ermöglichen ein Abteil zu erstellen.

B5.1 Das System soll es dem Benutzer ermöglichen ein Abteil zu löschen.

B5.2 Das System soll es dem Benutzer ermöglichen ein Abteil einem Regal zuzuordnen.

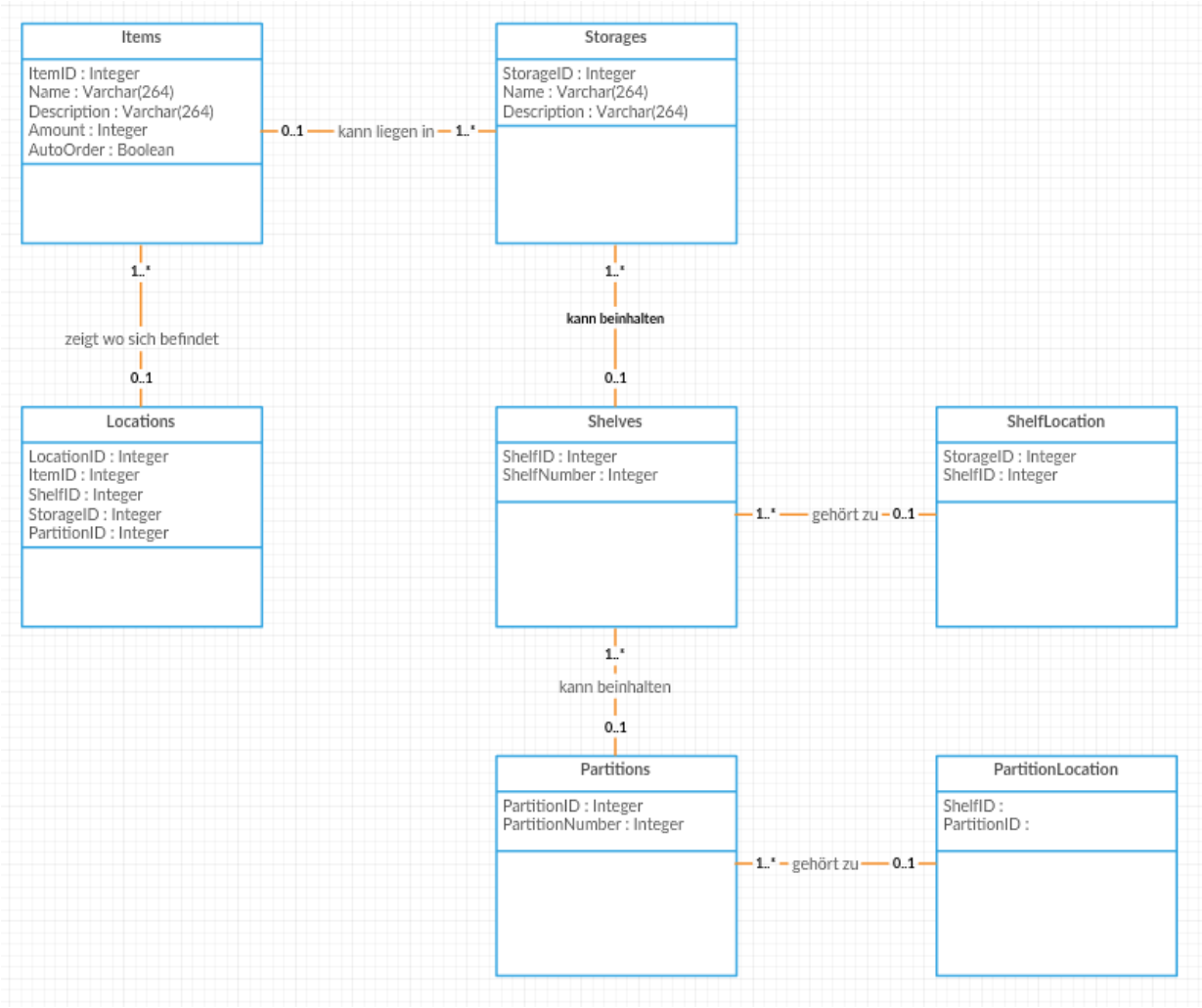
B5.2 Wenn der Benutzer bei der Erstellung eines Abteils eine ungültige Eingabe macht, soll das System eine Fehlermeldung anzeigen.

2.2.6 Abteile einsehen

B6 Das System soll es dem Benutzer ermöglichen alle Abteile, zusammen mit den dazugehörigen Regalen und deren Lagern, einzusehen.

3.Datenmodell

Diese Abbildung zeigt das Datenmodell in Form eines UML-Klassendiagramms. Hier werden zunächst die Datentypen *String* für Zeichenketten, *Boolean* für Ja/Nein, *Integer* für Zahlenwerte, *DateTime* für Zeitstempel und *Longtext* für Texteingaben verwendet. Längenbegrenzungen für die Attribut werte werden erst im Rahmen des Software-technischen Entwurfs festgelegt.



4.Software Architektur

Die Umsetzung des Lagersystems erfolgt in Python mit einer MySQL-Datenbank im Hintergrund um die Daten abzuspeichern.
Für die Kommunikation zwischen Datenbank und Python wird eine eigens geschriebene DB-API benutzt.
Um die Implementierung zu vereinfachen wurde das System in vier Schichten Aufgeteilt:

- SQL
  - Datenbank
- Python
  - Datenbank API
  - Funktionen
  - GUI

5.Implementierung

Im Folgenden sind die Anforderungen des Pflichtenheftes den einzelnen SQL-Befehlen mittels ihrer ID zugeordnet. Einige Anforderungen können auf der Ebene der Datenhaltungsschicht allein durch Fremdschlüssel oder UNIQUE-Bedingungen erfüllt werden.

5.1 SQL:

5.1.1 DATENBANK-SCHEMA:

```
CREATE TABLE Items
(
ItemID          INTEGER         NOT NULL    AUTO_INCREMENT      UNIQUE,
Name            VARCHAR(256)    NOT NULL    UNIQUE,
Description     VARCHAR (256)   NULL,
Amount          INTEGER         NOT NULL,
AutoOrder       BOOLEAN         NOT NULL    DEFAULT false,

PRIMARY KEY (ItemID)

);


CREATE TABLE Storages
(
StorageID       INTEGER         NOT NULL    AUTO_INCREMENT      UNIQUE,
Name            VARCHAR(256)    NOT NULL    UNIQUE,
Description     VARCHAR(256)    NULL,

PRIMARY KEY (StorageID)

);


CREATE TABLE Shelves
(
ShelfID         INTEGER         NOT NULL    AUTO_INCREMENT      UNIQUE,
ShelfNumber     INTEGER         NOT NULL    UNIQUE,

PRIMARY KEY (ShelfID)

);
```



```
CREATE TABLE Partitions
(
PartitionID         INTEGER         NOT NULL    AUTO_INCREMENT      UNIQUE,
PartitionNumber     INTEGER         NOT NULL    UNIQUE,

PRIMARY KEY (PartitionID)

);


CREATE TABLE Locations
(
LocationID          INTEGER         NOT NULL    AUTO_INCREMENT      UNIQUE,
ItemID              INTEGER         NOT NULL,
StorageID           INTEGER         NOT NULL,
ShelfID             INTEGER         NOT NULL,
PartitionID         INTEGER         NULL,

PRIMARY KEY (LocationID),

FOREIGN KEY (ItemID)
 REFERENCES Items(ItemID)
 ON UPDATE CASCADE
 ON DELETE CASCADE,

FOREIGN KEY (StorageID)
 REFERENCES Storages(StorageID)
 ON UPDATE CASCADE
 ON DELETE CASCADE,

FOREIGN KEY (ShelfID)
 REFERENCES Shelves(ShelfID)
 ON UPDATE CASCADE
 ON DELETE CASCADE,

FOREIGN KEY (PartitionID)
 REFERENCES Partitions(PartitionID)
 ON UPDATE CASCADE
 ON DELETE CASCADE

);
```



```
CREATE TABLE ShelfLocations
(
StorageID           INTEGER          NOT NULL,
ShelfID             INTEGER          NOT NULL,

PRIMARY KEY (StorageID),

FOREIGN KEY (ShelfID)
 REFERENCES Shelves(ShelfID)
 ON UPDATE CASCADE
 ON DELETE RESTRICT

);


CREATE TABLE PartitionLocations
(
ShelfID             INTEGER          NOT NULL,
PartitionID         INTEGER          NOT NULL,

PRIMARY KEY (ShelfID),

FOREIGN KEY (PartitionID)
 REFERENCES Partitions(PartitionID)
 ON UPDATE CASCADE
 ON DELETE RESTRICT

);
```

5.2 BEISPIEL DATENBANK INSERT:

Beispiel Inserts befinden sich im Anhang.

5.3 ANFORDERUNGEN:

//In dieser Version noch nicht vorhanden

5.4 PYTHON:

5.4.1 DATENBANK API:

Beschreibung:

Erstellt ein neues Item mit den gegeben Parametern und setzt es in ein Lager-Regal-Abteil.
Bei Erfolg wird ein String zurückgegeben, bei Fehler ein String mit dem Error Text.

Funktion: DBcreateItem(name,description,storageName,shelfName, partitionName,amount,autoOrder)

Parameter:

name:	string
description:	string or ""
storageName:	string
shelfName:	string
partitionName:	string
amount:	integer >=0
autoOrder:	boolean

Return:

if successful = string (string of successful report)
if error = string (string of error report)

Beschreibung:

Erstellt mit den Parametern ein neues Lager
Bei Erfolg wird ein String zurückgegeben, bei Fehler ein String mit dem Error Text.

Funktion: DBcreateStorage(name,description)

Parameter:

name:	string
description:	string or ""

Return:

if successful = string (string of successful report)
if error = string (string of error report)

Beschreibung:

Erstellt mit den Parametern ein neues Regal und setzt es in ein Lager
Bei Erfolg wird ein String zurückgegeben, bei Fehler ein String mit dem Error Text.

Funktion: DBcreateShelf(name,storageName)

Parameter:

name:	string
storageName:	string

Return:

if successful = string (string of successful report)
if error = string (string of error report)

Beschreibung:
Erstellt mit den Parametern ein neues Abteil und setzt es in ein Regal
Bei Erfolg wird ein String zurückgegeben, bei Fehler ein String mit dem Error Text.
Funktion: DBcreatePartition(name,shelfName)
Parameter:
name: string
shelfName: string
Return:
if successful = string (string of successful report)
if error = string (string of error report)

Beschreibung:
Bearbeitet die angegeben Parameter bei dem Item mit dem gegeben namen
Bei Erfolg wird ein String zurückgegeben, bei Fehler ein String mit dem Error Text.
Funktion: DBupdateItem(name,storageName,shelfName, partitionName,autoOrder,amount)
Parameter:
name: string
storageName: string
shelfName: string
partitionName:string
amount: integer >=0
autoOrder: boolean
Return:
if successful = string (string of successful report)
if error = string (string of error report)

Beschreibung:
Wählt ein Item aus Diese werden in einer doppelten Liste ausgegebenWenn item = „all“ ist werden alle items zurückgegeben
Bei Fehler wird ein String mit dem Error Text zurückgegeben.
Funktion: DBselectItem(item)
Parameter:
item: string or „all“
Return:
if successful=
list(list(name,description,amount,storageName,shelfName, partitionName,amount,autoOrder)) //all String
if error = string (string of error report)

Beschreibung:

Wählt alle Lager im System mit allen Ihren Regalen und deren Abteilen aus. Diese werden in einer verketteten Liste ausgegeben. Bei Fehler wird ein String mit dem Error Text zurückgegeben.

Funktion: DBselectAllStorage()

Return:

if successful= ((Lager1(Regal1(Abteil1...Abteilx)))Lager2(Regal1(Abteil1...Abteilx)

if error = string (string of error report)

5.4.2 FUNKTIONEN:

Beschreibung:

Extrahiert die Eingaben aus den Eingabefeldern der GUI und gibt diese an die createItem(...) der DB-API weiter um ein Item zu erstellen.

Funktion:

createItem(nameEntry,descriptionEntry,storageNameEntry,shelfNameEntry, partitionNameEntry,autoOrderTickbox,amountEntry)

Parameter:

nameEnty:	class
descriptionEntry:	class
storageNameEntry:	class
shelfNameEntry:	class
partitionNameEntry:	class
autoOrderTickbox:	class
amountEntry:	class

Return: if successful = string (string of successful report)
if error = string (string of error report)

Beschreibung:

Extrahiert die Eingaben aus den Eingabefeldern der GUI und gibt diese an die createStorage(...) der DB-API weiter um ein Lager zu erstellen.

Funktion: createStorage(nameEntry,descriptionEntry)

Parameter:

nameEntry:	class
descriptionEntry:	class

Return: if successful = string (string of successful report)
if error = string (string of error report)

Beschreibung:

Extrahiert die Eingaben aus den Eingabefeldern der GUI und gibt diese an die createShelf(...) der DB-API weiter um ein Regal zu erstellen.

Funktion: createShelf(nameEntry,storageNameEntry)

Parameter:

nameEntry:	class
storageNameEntry:	class

Return: if successful = string (string of successful report)
if error = string (string of error report)

Beschreibung:
Extrahiert die Eingaben aus den Eingabefeldern der GUI und gibt diese an die createPartiton(...) der DB-API weiter um ein Abteil zu erstellen.
Funktion: createPartiton(nameEntry,shelfNameEntry)
Parameter: nameEntry: class
shelfNameEntry: class

Return: if successful = string (string of successful report)
if error = string (string of error report)

Beschreibung:
Extrahiert die Eingaben aus den Eingabefeldern der GUI und gibt diese an die updateItem(...) der DB-API weiter um die Parameter eines Items zu aktualisieren.
Funktion: updateItem(nameEntry,storageNameEntry,shelfNameEntry, partitionNameEntry,autoOrderTickbox,amountEntry)
Parameter: nameEntry: class
storageNameEntry: class
shelfNameEntry: class
partitionNameEntry: class
amountEntry: class
autoOrderTickbox: class

Return: if successful = string (string of successful report)
if error = string (string of error report)

Beschreibung:
Wählt ein Item aus oder alle wenn „all“ übergeben wird
Funktion: selectItem(item)
Parameter: item: string or „all“

Return:
if successful=
list(list(name,description,amount,storageName,shelfName, partitionName,amount,autoOrder)) //all String
if error = string (string of error report)

Beschreibung:
Schleift die Anfrage durch, alle Lager, ihre Regale und deren Abteile zu returnen . Diese werden in einer verketteten Liste ausgegeben. Bei Fehler wird ein String mit dem Error Text zurückgegeben.
Funktion: selectAllStorage()
Return:

if successful=
((Lager1(Regal1(Abteil1...Abteilx)))Lager2(Regal1(Abteil1...Abteilx)
//all String
if error = string (string of error report)

5.3 GUI:

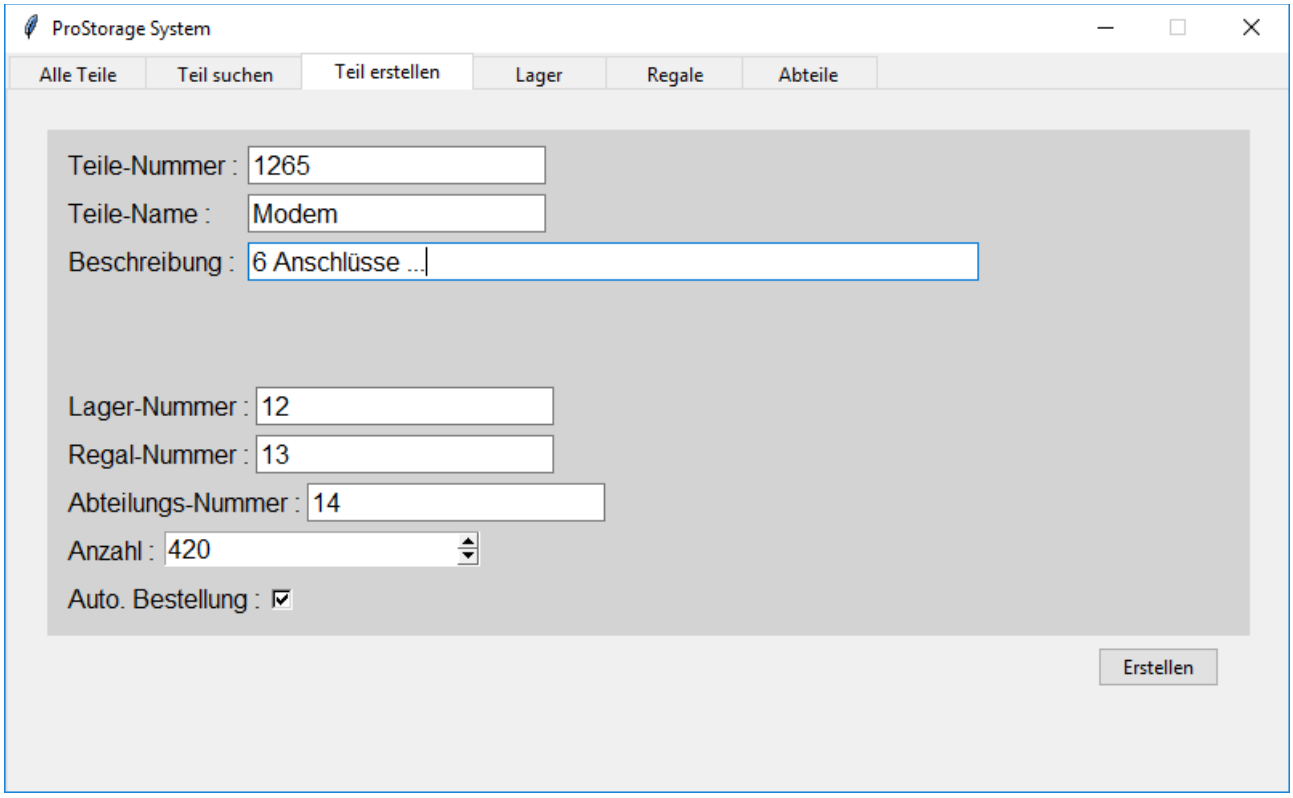


Abbildung 1: 5.3.1 Item erstellen --Anforderung A1 auf Seite 2

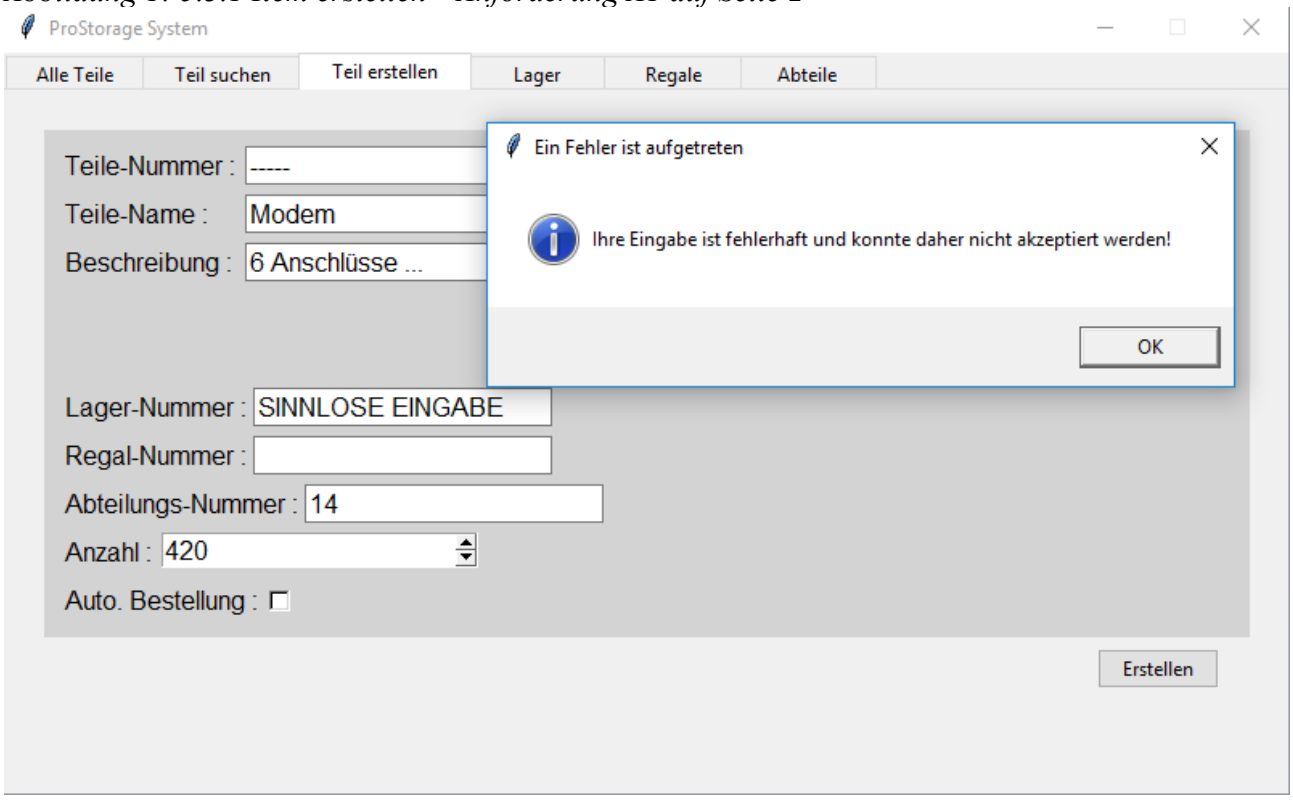


Abbildung 2: 5.3.2 Fehlermeldung bei ungültiger Eingabe--Anforderung A1.1 auf Seite 2

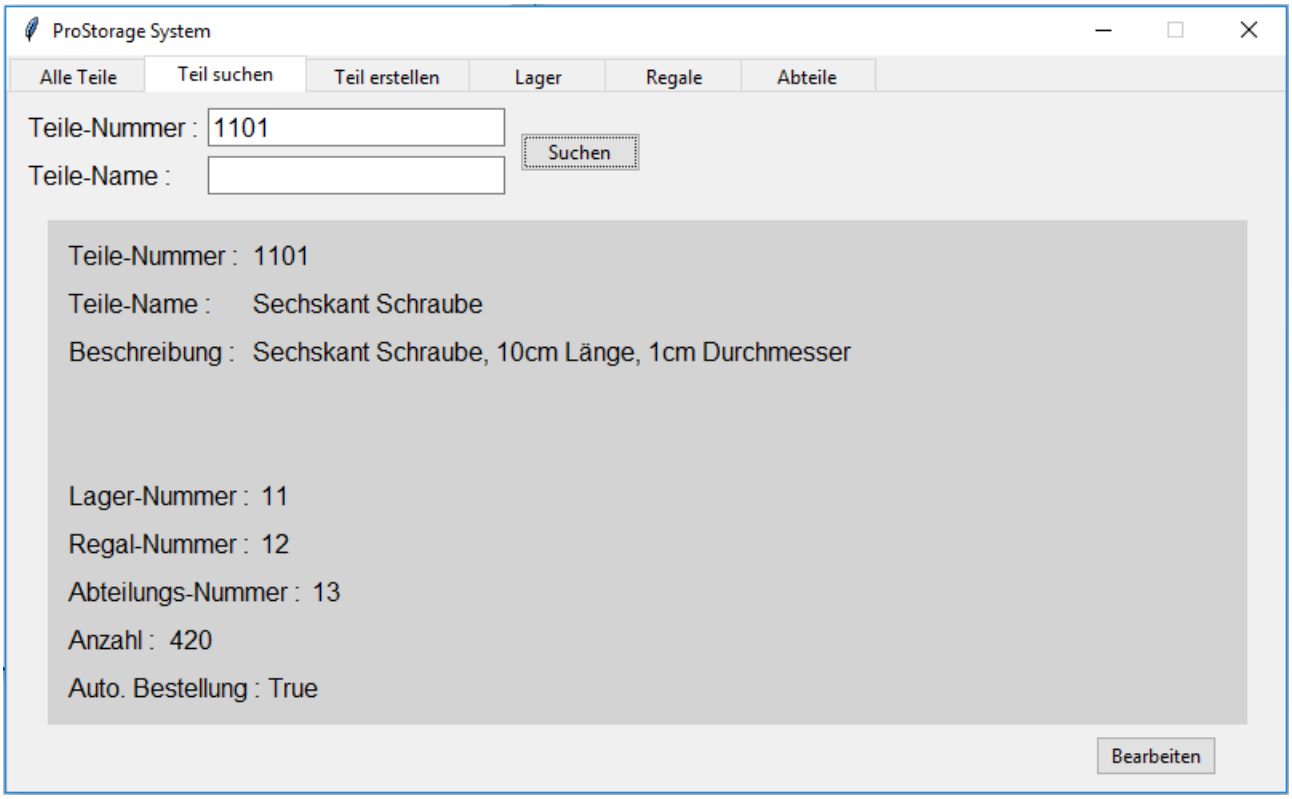


Abbildung 3: 5.3.3 Item suchen--Anforderung A7 und A8 auf Seite 3

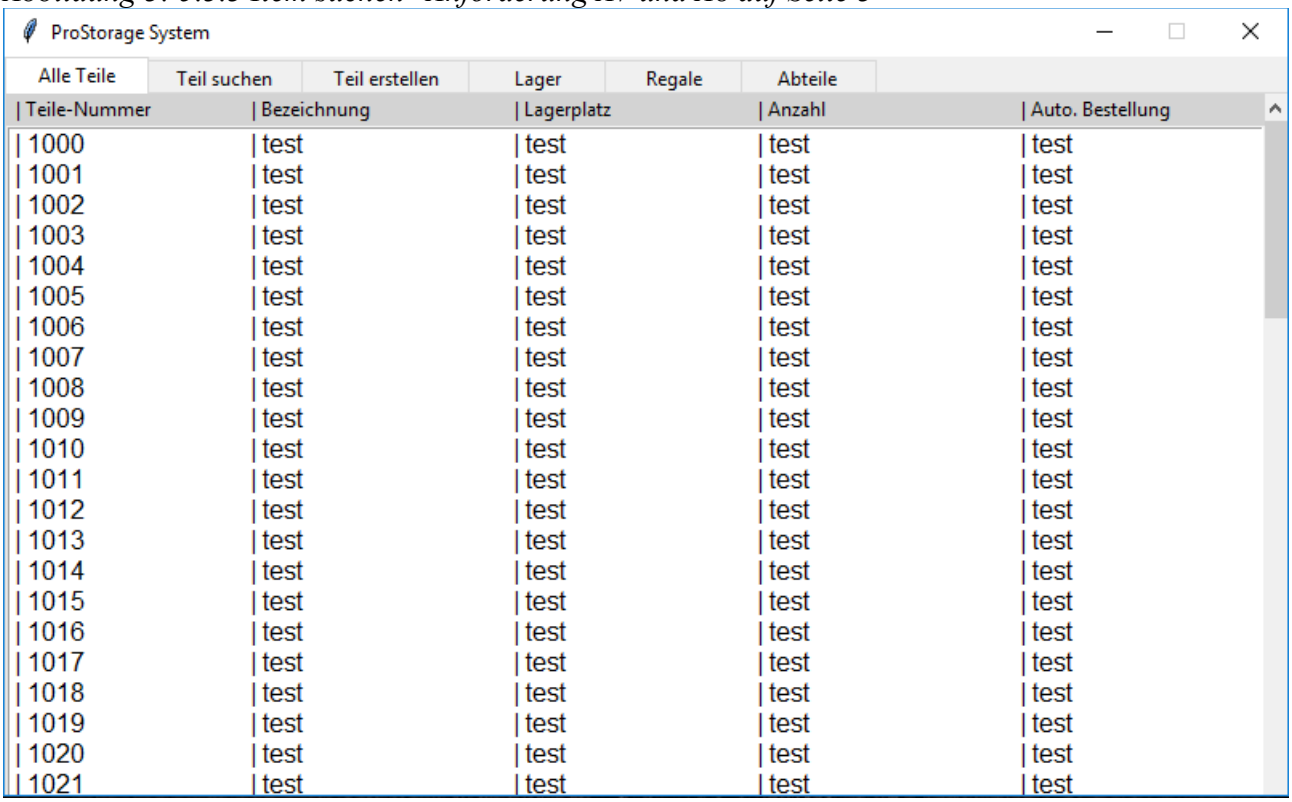


Abbildung 4: 5.3.4 Items einsehen --Anforderung A9 auf Seite 3

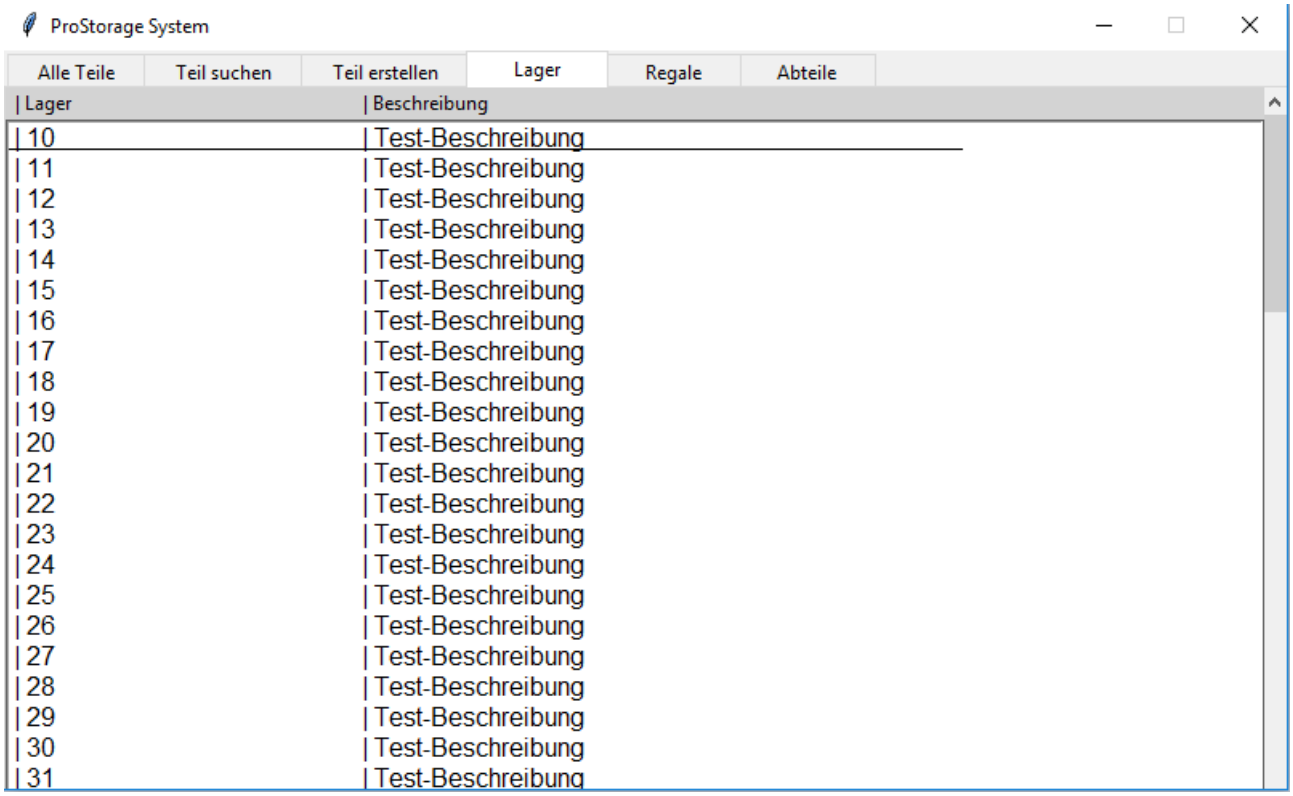


Abbildung 5: 5.3.5 Alle Lager einsehen--Anforderung B2 auf Seite 3

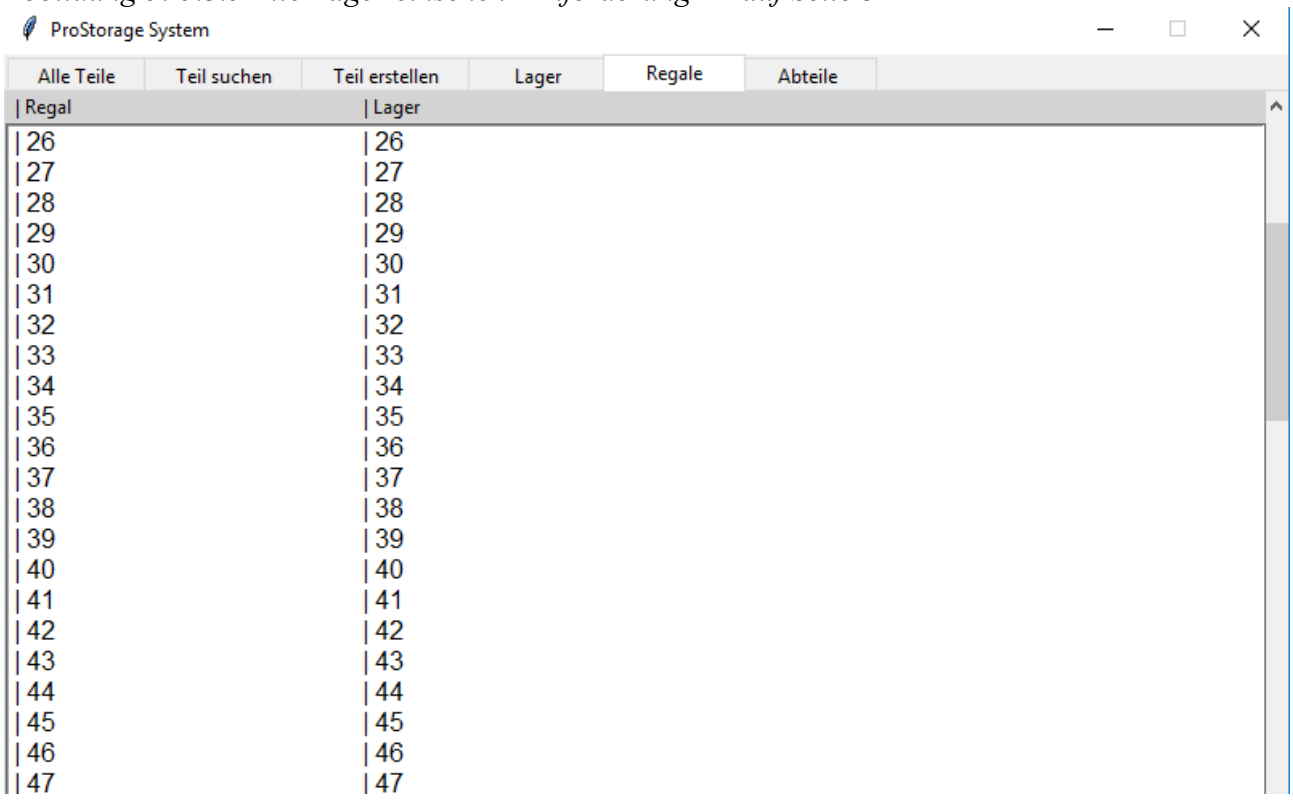


Abbildung 6: 5.3.6 Alle Regale einsehen--Anforderung B4 auf Seite 3

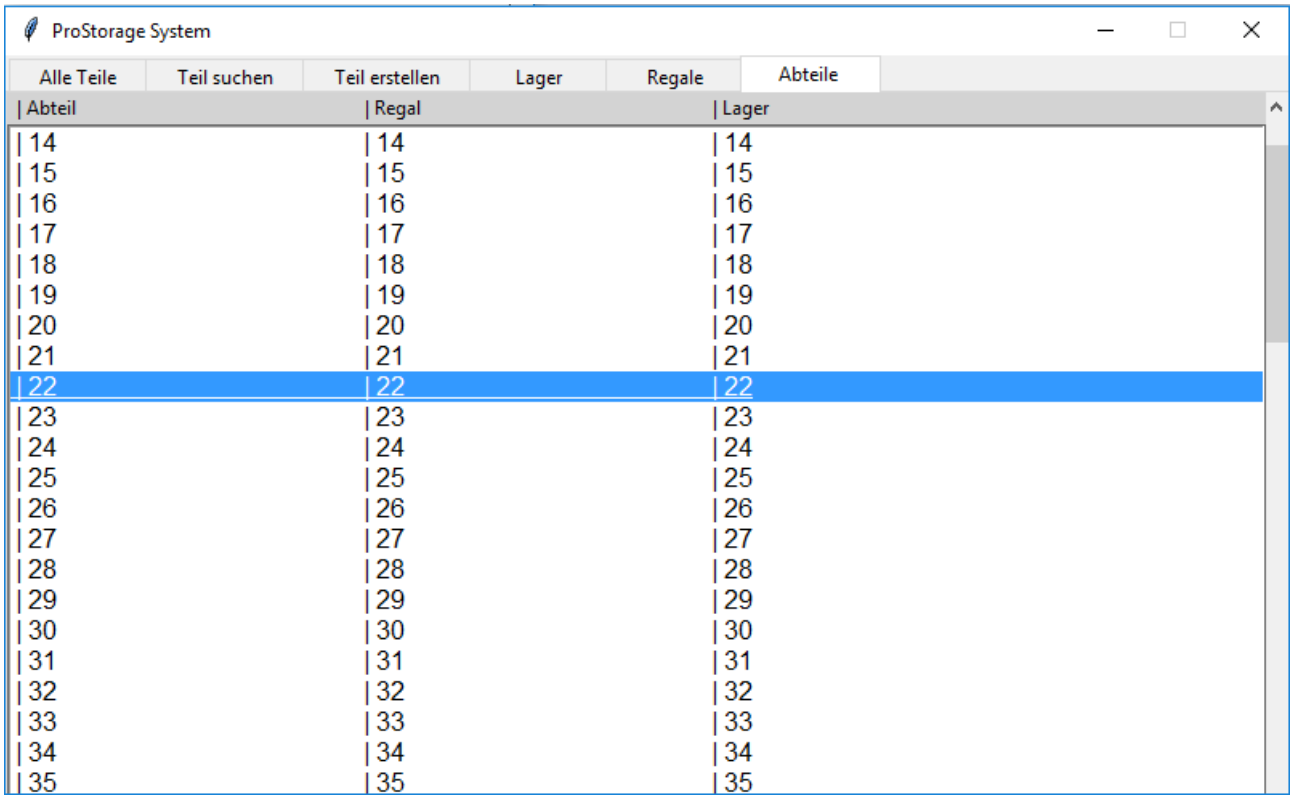


Abbildung 7: 5.3.7 Alle Abteile einsehen--Anforderung B6 auf Seite 3

