Technology
Arts Sciences
TH Köln

WAWI Grobentwurf 2022

Softwarepraktikum

TH Köln - Informatik Labor

Version 8.3, 2022-03-23 08:27:53 UTC

Inhaltsverzeichnis

1. Speicherstruktur der Daten

Die konkreten Speicherstrukturen der Daten eines Systems werden typischerweise erst im Feinentwurf festgelegt.

Im Software-Praktikum legen wir diese jedoch bereits im Grobentwurf fest, weil diese Strukturen für alle Teilnehmer vorgegeben sind und somit alle Teilnehmer auf den gleichen Datenstrukturen Ihren eigenen Feinentwurf durchführen können. (Achtung: Nicht im Systementwurfs-Praktikum nachmachen!)

Die Abbildung zeigt die konkrete Umsetzung der Datenspezifikation aus der Systemspezifikation in der WAWI-Datenbank.

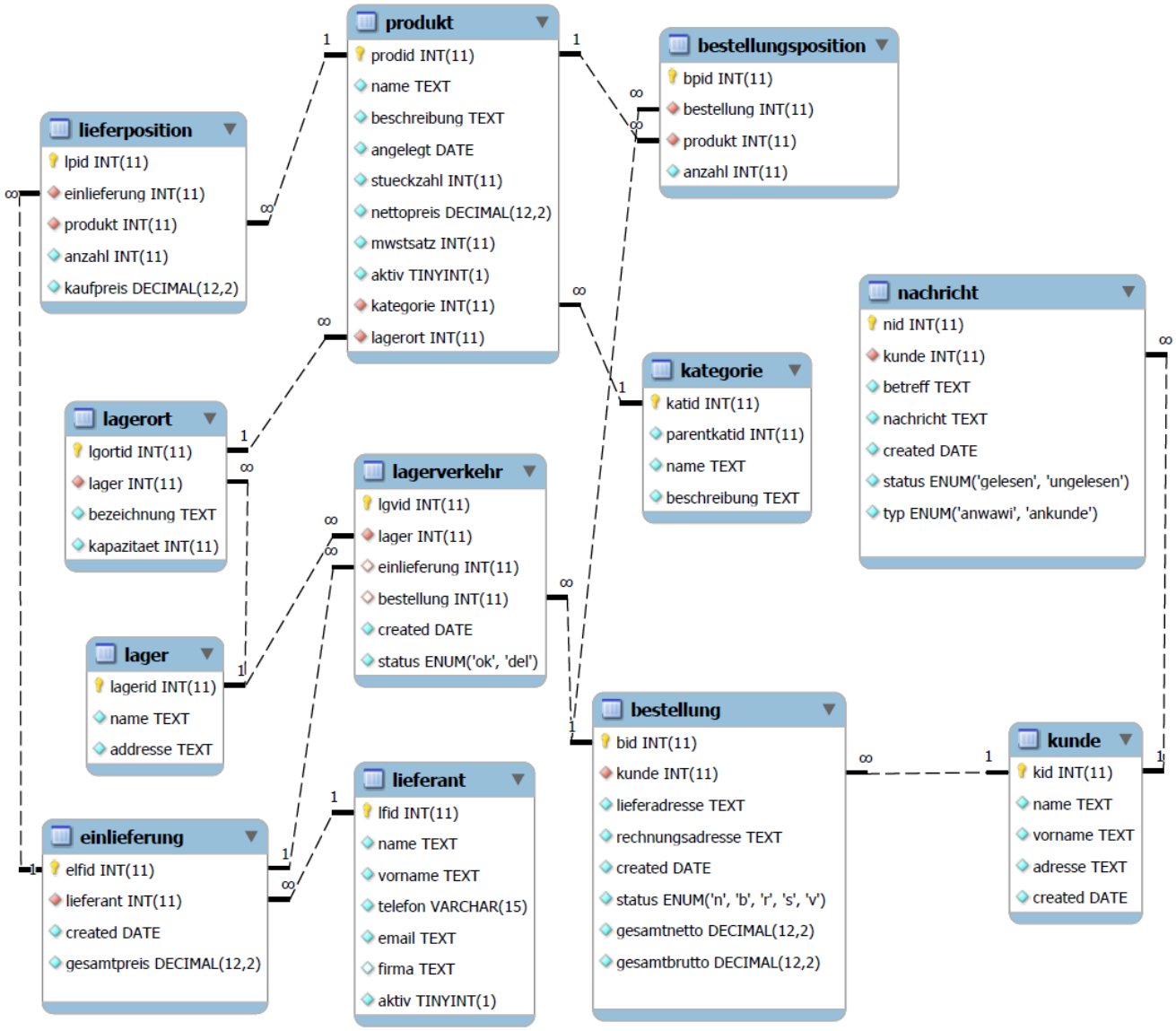

Abbildung 1. Umsetzung der Datenspezifikation in der wawidb in MySQL

2. Systemarchitektur

In der Abbildung sind alle Komponenten des WAWI-Systems zusammen mit ihren angebotenen und zu verwendenden Schnittstelle zu erkennen. Eine Komponente im Softwarepraktikum entspricht genau einem Java-Modul. Jede Komponente wird in genau einem Maven-Projekt entwickelt. Der Name dieses Maven-Projekts in Gitlab entspricht dem Namen der Komponente (jedoch nur Kleinbuchstaben).

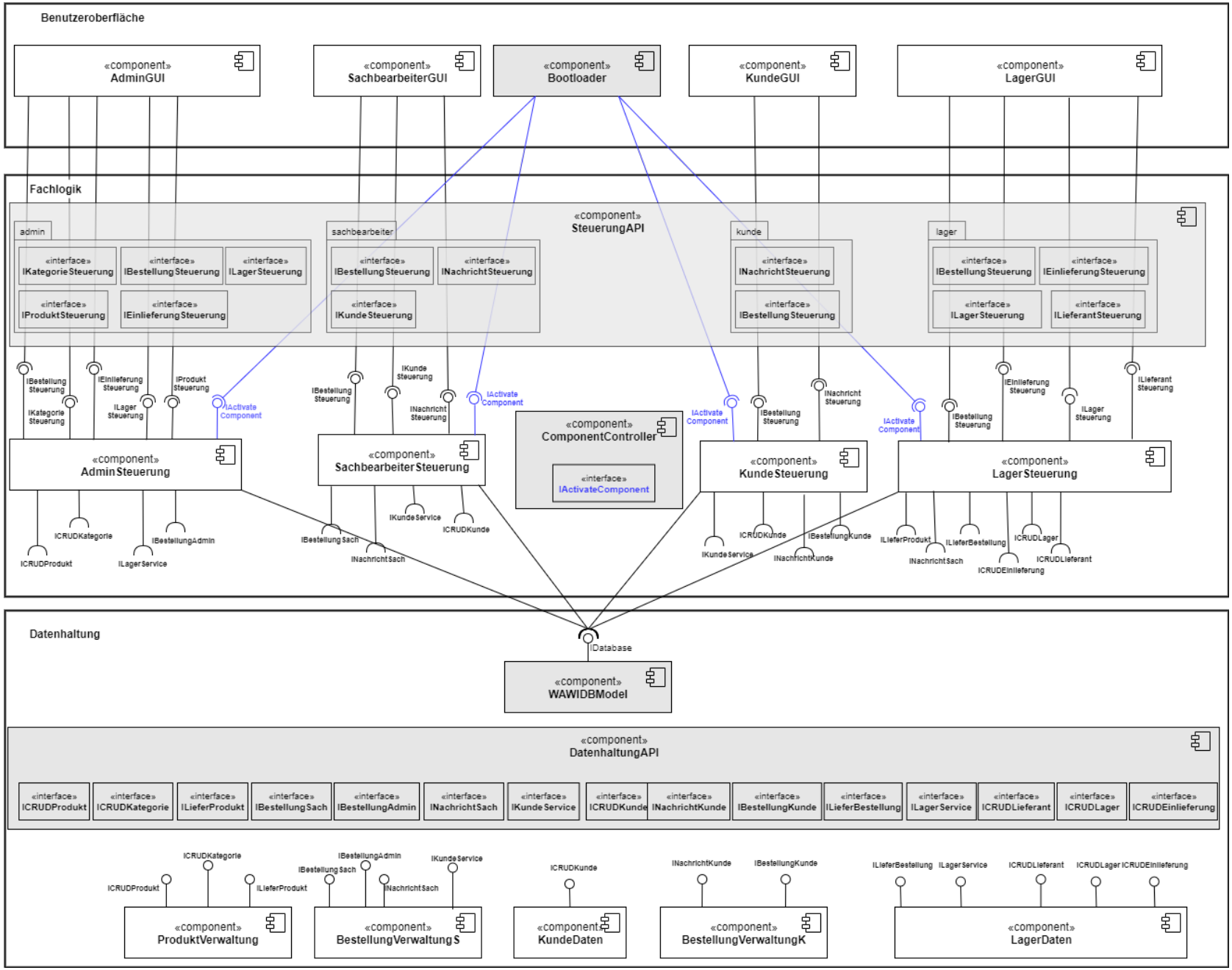

Abbildung 2. WAWI Komponenten-Diagramm mit allen Schnittstellen

3. Komponentenspezifikationen

3.1. Bootloader

3.1.1. Funktionen

Die Bootloader-Komponente stellt das zentrale Startprogramm des WAWI-Systems bereit. Es handelt sich dabei um eine Oberfläche, von der aus die speziellen Bedienoberflächen der Komponenten `AdminGUI`, `SachbearbeiterGUI`, `LagerhalterGUI` und `KundeGUI` gestartet werden können. Der Start der speziellen Bedienoberflächen dieser Komponenten geschieht durch die Verwendung der jeweiligen Implementierung der IActivateComponent- Schnittstelle der zugehörigen Komponenten auf der Fachlogik-Schicht.

Diese Komponente wird von den Organisatoren des Software-Praktikums implementiert.

Schnittstellen

Diese Komponente bietet keine implementierten Schnittstellen für andere Komponenten an.

Besonderheiten

Dieser Komponente müssen zur Ausführungszeit alle anderen Komponenten und alle verwendeten Bibliotheken zur Verfügung stehen.

3.2. ComponentController

3.2.1. Funktionen

Die ComponentController-Komponente stellt die Interface-Klasse `IActivateComponent` im Paket `de.thkoeln.swp.wawi.componentcontroller.services` zur Verfügung. Diese muss von den Komponenten der Fachlogik-Schicht implementiert werden.

Diese Komponente wird von den Organisatoren des Software-Praktikums implementiert.

3.2.2. Schnittstellen

Diese Komponente bietet keine implementierten Schnittstellen für andere Komponenten an.

3.3. WAWIDBModel

3.3.1. Funktionen

Die `WAWIDBModel`-Komponente besteht aus Entitätsklassen, welche die Tabellen oder Entitäten der Datenbank des Warenwirtschaftssystems repräsentieren wie sie in der Systemspezifikation spezifiziert sind. Es existieren zwei identische Instanzen der Datenbank des Warenwirtschaftssystems: `wawidb_dev` (Development) und `wawidb_prod` (Produktiv). Entsprechend dazu enthält die `WAWIDBModel`-Komponente auch zwei Persistence-Units namens `WAWIDBDEVPU` und

`WAWIDBPRODPU`, mithilfe derer auf die entsprechende Instanz der Datenbank zugegriffen werden kann. Während der gesamten Entwicklungsphase muss die `wawidb_dev`-Datenbank verwendet werden. Erst für die Integration des Gesamtsystems (d.h. zum MS5) soll auf die `wawidb_prod` -Datenbank zugegriffen werden.

Die `WAWIDBModel`-Komponente wurde bereits implementiert und ist im Git-Repository verfügbar, darf aber nicht verändert werden!

Die Klassen der Entitäten wird von dieser Komponente im Paket `de.thkoeln.swp.wawi.wawidbmodel.entities` bereit gestellt.

3.3.2. Schnittstellen

IDatabase

Die `WAWIDBModel`-Komponente bietet die `IDatabase`-Schnittstelle an. Diese wird von allen Daten- und Steuerungskomponenten verwendet, um über einen systemweit einzigen `EntityManager` einen konsistenten Zugriff auf die Datenbank zu ermöglichen.

Die Schnittstelle ist in nachfolgender Abbildung spezifiziert.

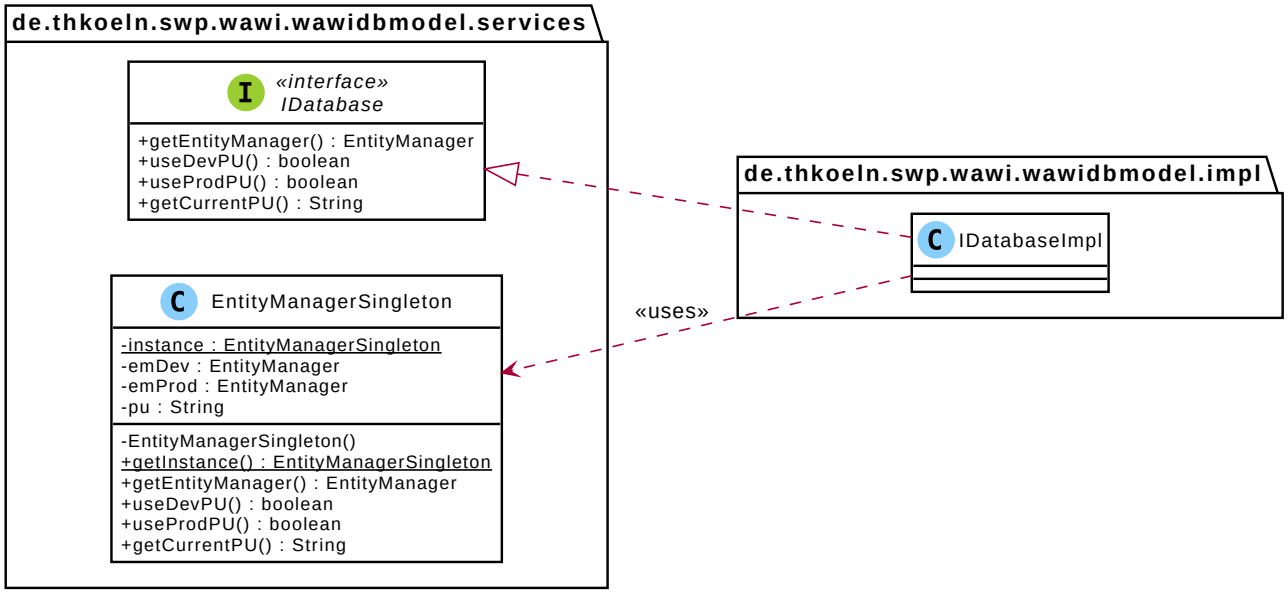

Abbildung 3. `IDatabase`-Schnittstelle der `WAWIDBModel`-Komponente

`useDevPU`

: wählt die Developer DB zur Benutzung durch die Persistence Unit aus.

`useProdPU`

: wählt die Produktiv DB zur Benutzung durch die Persistence Unit aus.

`getCurrentPU`

: liefert eine Information über die aktuell ausgewählte Persistence Unit (PU). Der Rückgabewert ist vom Typ `String` und hat entweder den Wert *WAWIDEVPU* oder *WAWIPRODPU*.

Diese Schnittstelle ist bereits implementiert und muss von den Teilnehmern des SWP verwendet werden.

3.3.3. Verhalten

Diese Komponente besitzt kein zu spezifizierendes Verhalten.

3.4. DatenhaltungAPI

3.4.1. Funktionen

Diese Komponente wurde von den Organisatoren des Software-Praktikums bereits implementiert.

Diese Komponente enthält alle Interface-Klassen der Schicht Datenhaltung. Diese Interface-Klassen werden von anderen Komponenten der Schicht Datenhaltung implementiert. Diese Implementierungen werden von den Komponenten der Schicht Fachlogik für Zugriffe auf die gespeicherten Daten verwendet.

3.4.2. Schnittstellen

Diese Komponente bietet keine implementierten Schnittstellen für andere Komponenten an.

3.5. SteuerungAPI

3.5.1. Funktionen

Diese Komponente wurde von den Organisatoren des Software-Praktikums bereits implementiert.

Diese Komponente enthält alle Interface-Klassen der Schicht Steuerung. Diese Interface-Klassen werden von anderen Komponenten der Schicht Steuerung implementiert. Diese Implementierungen werden von den Komponenten der Schicht GUI für Zugriffe auf die Anwendungsfälle benötigt. Zusätzlich bietet diese Komponente die benötigten Grenzklassen an.

3.5.2. Schnittstellen

Diese Komponente bietet keine implementierten Schnittstellen für andere Komponenten an.

3.6. AdminGUI

3.6.1. Funktionen

Die `AdminGUI`-Komponente stellt die Benutzeroberfläche für alle Anwendungsfälle des Admins zur Verfügung.

Sie soll mittels JavaFX realisiert werden.

Die `AdminApp`-Klasse und `AdminController`-Klasse sowie eine `admin.fxml`-Datei werden wie folgt vorgegeben:

- im `src/main/java`-Ordner:
  - `de.thkoeln.swp.wawi.admingui.application.AdminApp`

    - `de.thkoeln.swp.wawi.admingui.control.AdminController`

- im `src/main/resources`-Ordner:
    - `admin.fxml`

Der weitere Feinentwurf dieser Komponente ist nicht vorgegeben und soll von dem `Admin`-Team eigenständig erstellt werden.

3.6.2. Schnittstellen

Diese Komponente bietet keine implementierten Schnittstellen für andere Komponenten an.

3.7. AdminSteuerung

3.7.1. Funktionen

Diese Komponente realisiert die Anwendungsfälle: /LF100/, /LF105/, /LF110/, /LF120/, /LF130/ und /LF140/.

Zur Realisierung der Anwendungsfälle darf nicht direkt auf die Datenbank zugegriffen werden. Es dürfen nur die Schnittstellen verwendet werden, für die in Abbildung 2 der Zugriff definiert wurde.

3.7.2. Schnittstellen

IActivateComponent

Die `AdminSteuerung`-Komponente implementiert die `IActivateComponent`-Schnittstelle (und exportiert diese Implementierung), die von der `Bootloader`-Komponente verwendet wird, um die Komponente zu aktivieren und zu deaktivieren. Die Implementierung muss dafür im Paket `de.thkoeln.swp.wawi.adminsteuerung.impl` mit dem Namen `IActivateComponentImpl` liegen, da diese sonst vom Bootloader nicht gefunden werden kann.

Die Schnittstellen-Klasse `IActivateComponent` wird von der Komponente `ComponentController` bereitgestellt.

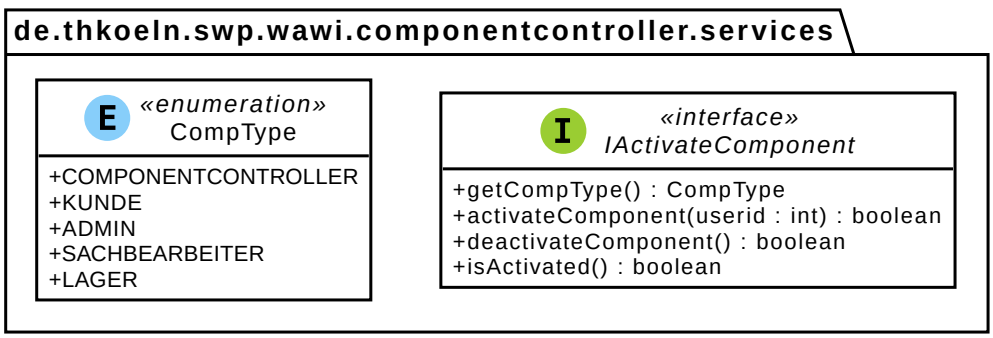

Abbildung 4. `IActivateComponent`-Schnittstelle zur Verwendung in der `AdminSteuerung`-Komponente

Der Parameter der `activateComponent`-Funktion der `IActivateComponent`-Schnittstelle enthält die jeweilige ID des Admins, für den die Komponente aktiviert werden soll. Falls eine falsche ID

übergeben wird, sollte die Komponente nicht aktiviert werden und `false` zurückliefern. Die Komponente darf nur für den Default-Admin (userid = 14) aktiviert werden.

Die Komponente kann die beiden Zustände *Aktiviert* und *Deaktiviert* annehmen. Die Methoden der Schnittstelle `IActivateComponent` können zu Zustandsübergängen führen:

- Wird die Komponente gestartet, so befindet sie sich zunächst im Zustand *Deaktiviert*.
- Wird im Zustand *Deaktiviert* `activateComponent()` mit zulässiger ID aufgerufen, gibt die Methode ein `true` zurück und geht in den Zustand *Aktiviert*.
- Wird `activateComponent()` mit einer nicht zulässigen ID aufgerunfen, wird nur `false` zurück gegeben.
- Wird im Zustand *Aktiviert* `activateComponent()` aufgerufen, wird ein `false` zurück gegeben, die Komponenten bleibt im Zustand *Aktiviert*.
- Wird im Zustand *Aktiviert* `deactivateComponent()` aufgerufen, gibt die Methode ein `true` zurück und geht in den Zustand *Deaktiviert*.
- Wird im Zustand *Deaktiviert* `deactivateComponent()` aufgerufen, wird ein `false` zurück gegeben, die Komponenten bleibt im Zustand *Deaktiviert*.
- Wird im Zustand *Deaktiviert* die Methode `getCompType()` aufgerufen, liefert sie den `CompType ADMIN` als Ergebnis.
- Wird im Zustand *Aktiviert* die Methode `getCompType()` aufgerufen, liefert sie den `CompType ADMIN` als Ergebnis.
- Wird im Zustand *Deaktiviert* die Methode `isActivated()` aufgerufen, wird ein `false` zurück gegeben.
- Wird im Zustand *Aktiviert* die Methode `isActivated()` aufgerufen, wird ein `true` zurück gegeben.

Schnittstellen

Die Schnittstellen-Klassen `IKategorieSteuerung`, `IBestellungSteuerung`, `IEinlieferungSteuerung`, `ILagerSteuerung` und `IProduktSteuerung` werden von der Komponente `SteuerungAPI` bereitgestellt.

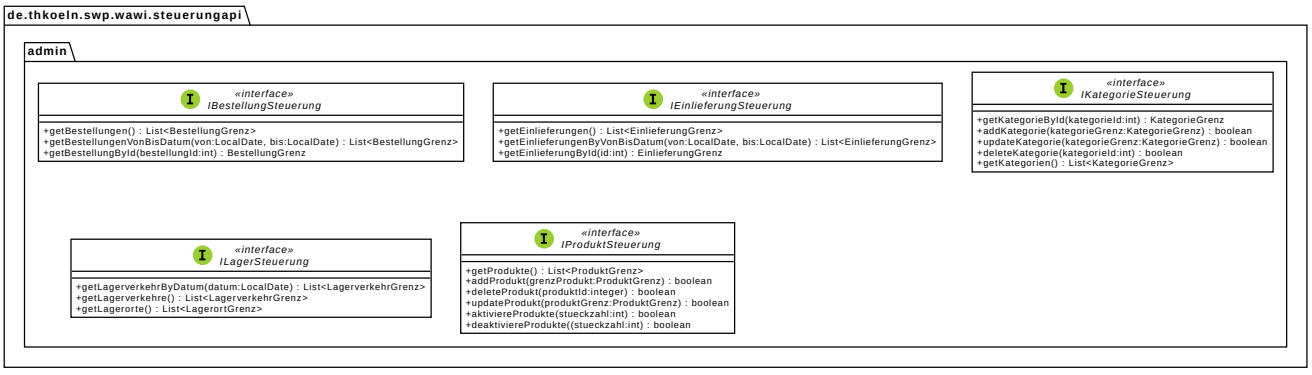

Abbildung 5. Schnittstellen zur Verwendung in der `AdminSteuerung`-Komponente

3.8. ProduktVerwaltung

3.8.1. Funktionen

Diese Komponente realisiert einige Schnittstellen für den Zugriff auf die Entitätsklassen.

3.8.2. Schnittstellen

ICRUDProdukt

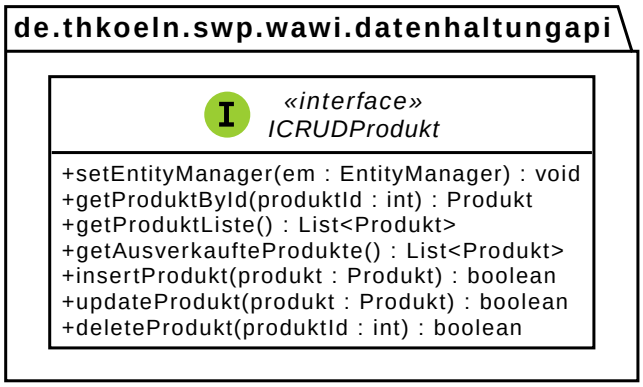

Abbildung 6. `ICRUDProdukt`-Schnittstelle zur Verwendung in der `ProduktVerwaltung`-Komponente

Grobe Spezifikation der Methoden:

`getProduktById`
: liefert das Produkt mit der angegebenen Id.

`getProduktListe`
: liefert alle Produkte als Liste.

`getAusverkaufteProdukte`
: lieferte alle Produkte deren Stückzahl = 0 ist.

`insertProdukt`
: fügt eines neues Produkt ein.

`updateProdukt`
: modifiziert ein existierendes Produkt.

`deleteProdukt`
: löscht ein existierendes Produkt.

Die detaillierte Spezifikation des erwarteten Verhaltens der Methoden dieser Schnittstelle sind in der Testspezifikation gegeben.

ICRUDKategorie

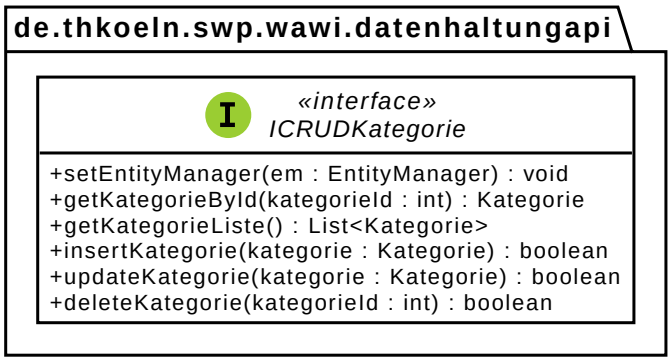

Abbildung 7. `ICRUDKategorie`-Schnittstelle zur Verwendung in der `ProduktVerwaltung`-Komponente

Grobe Spezifikation der Methoden:

`getKategorieById`

liefert die Kategorie mit der angegebenen Id.

`getKategorieListe`

liefert eine Liste aller Kategorien.

`insertKategorie`

fügt eine neue Kategorie ein.

`updateKategorie`

modifiziert eine existierende Kategorie.

`deleteKategorie`

löscht eine existierende Kategorie.

Die detaillierte Spezifikation des erwarteten Verhaltens der Methoden dieser Schnittstelle sind in der Testspezifikation gegeben.

ILieferProdukt

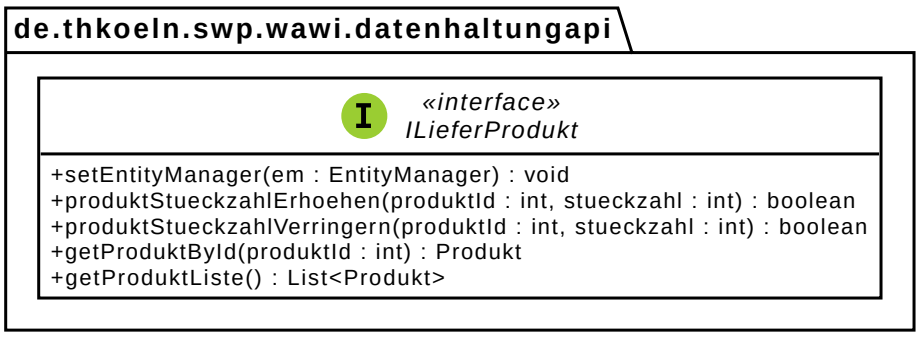

Abbildung 8. `ILieferProdukt`-Schnittstelle zur Verwendung in der `ProduktVerwaltung`-Komponente

Grobe Spezifikation der Methoden:

`produktStueckzahlErhoehen`

die Stückzahl eines über die ID bestimmten Produkts wird um die übergebene Stückzahl erhöht.War die Operation erfolgreich gibt die Methode ein `true` zurück, sonst `false`.

`produktStueckzahlVerringern`

die Stückzahl eines über die ID bestimmten Produkts wird um die übergebene Stückzahl verringert.War die Operation erfolgreich gibt die Methode ein `true` zurück, sonst `false`.

`getProduktById`

liefert das Produkt mit der angegebenen Id.

`getProduktListe`

liefert alle Produkte als Liste.

Die detaillierte Spezifikation des erwarteten Verhaltens der Methoden dieser Schnittstelle sind in der Testspezifikation gegeben.

3.9. SachbearbeiterGUI

3.9.1. Funktionen

Die `SachbearbeiterGUI`-Komponente stellt die Benutzeroberfläche für alle Anwendungsfälle des Sachbearbeiters zur Verfügung.

Sie soll mittels JavaFX realisiert werden.

Die `SachbearbeiterApp`-Klasse und `SachbearbeiterController`-Klasse sowie eine `sachbearbeiter.fxml`-Datei werden wie folgt vorgegeben:

- im `src/main/java`-Ordner:
  - `de.thkoeln.swp.wawi.sachbearbeitergui.application.SachbearbeiterApp`
  - `de.thkoeln.swp.wawi.sachbearbeitergui.control.SachbearbeiterController`
- im `src/main/resources`-Ordner:
  - `sachbearbeiter.fxml`

Der weitere Feinentwurf dieser Komponente ist nicht vorgegeben und soll von dem `Sachbearbeiter`-Team eigenständig erstellt werden.

3.9.2. Schnittstellen

Diese Komponente bietet keine implementierten Schnittstellen für andere Komponenten an.

3.10. SachbearbeiterSteuerung

3.10.1. Funktionen

Diese Komponente realisiert die Anwendungsfälle: /LF50/, /LF61/, /LF62/, /LF63/, /LF64/, /LF65/, /LF70/, /LF80/, /LF90/.

Zur Realisierung der Anwendungsfälle darf nicht direkt auf die Datenbank zugegriffen werden. Es dürfen nur die Schnittstellen verwendet werden, für die in Abbildung 2 der Zugriff definiert

wurde.

3.10.2. Schnittstellen

IActivateComponent

Die SachbearbeiterSteuerung-Komponente implementiert die IActivateComponent-Schnittstelle (und exportiert diese Implementierung), die von der Bootloader-Komponente verwendet wird, um die Komponente zu aktivieren und zu deaktivieren. Die Implementierung muss dafür im Paket de.thkoeln.swp.wawi.sachbearbeitersteuerung.impl mit dem Namen IActivateComponentImpl liegen, da diese sonst vom Bootloader nicht gefunden werden kann.

Die Schnittstellen-Klasse IActivateComponent wird von der Komponente ComponentController bereitgestellt.

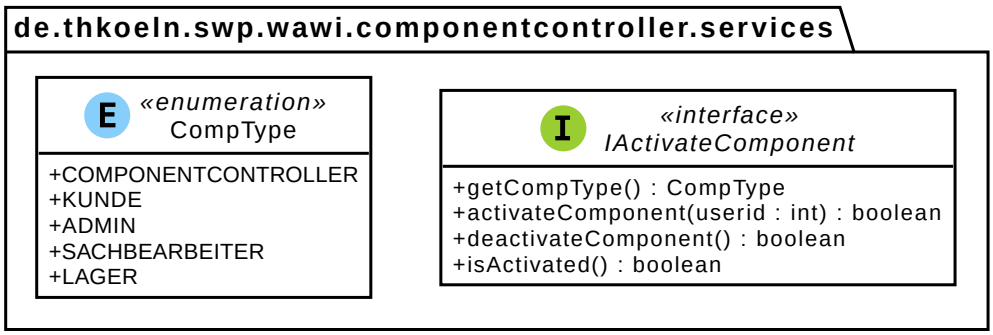

Abbildung 9. IActivateComponent-Schnittstelle zur Verwendung in der SachbearbeiterSteuerung-Komponente

Der Parameter der activateComponent-Funktion der IActivateComponent-Schnittstelle enthält die ID des Sachbearbeiters, für den die Komponente aktiviert werden soll. Falls eine falsche ID übergeben wird, sollte die Komponente nicht aktiviert werden und false zurückliefern. Die Komponente darf nur für den Default-Sachbearbeiter (userid = 20) aktiviert werden.

Die Komponente kann die beiden Zustände *Aktiviert* und *Deaktiviert* annehmen. Die Methoden der Schnittstelle IActivateComponent können zu Zustandsübergängen führen:

- Wird die Komponente gestartet, so befindet sie sich zunächst im Zustand *Deaktiviert*.
- Wird im Zustand *Deaktiviert* activateComponent() mit zulässiger ID aufgerufen, gibt die Methode ein true zurück und geht in den Zustand *Aktiviert*.
- Wird activateComponent() mit einer nicht zulässigen ID aufgerunfen, wird nur false zurück gegeben.
- Wird im Zustand *Aktiviert* activateComponent() aufgerufen, wird ein false zurück gegeben, die Komponenten bleibt im Zustand *Aktiviert*.
- Wird im Zustand *Aktiviert* deactivateComponent() aufgerufen, gibt die Methode ein true zurück und geht in den Zustand *Deaktiviert*.
- Wird im Zustand *Deaktiviert* deactivateComponent() aufgerufen, wird ein false zurück gegeben, die Komponenten bleibt im Zustand *Deaktiviert*.
- Wird im Zustand *Deaktiviert* die Methode getCompType() aufgerufen, liefert sie den CompType SACHBEARBEITER als Ergebnis.

- Wird im Zustand *Aktiviert* die Methode `getCompType()` aufgerufen, liefert sie den `CompType` `SACHBEARBEITER` als Ergebnis.
- Wird im Zustand *Deaktiviert* die Methode `isActivated()` aufgerufen, wird ein `false` zurück gegeben.
- Wird im Zustand *Aktiviert* die Methode `isActivated()` aufgerufen, wird ein `true` zurück gegeben.

Schnittstellen

Die Schnittstellen-Klassen `IBestellungSteuerung`, `IKundeSteuerung` und `INachrichtSteuerung` werden von der Komponente `SteuerungAPI` bereitgestellt.

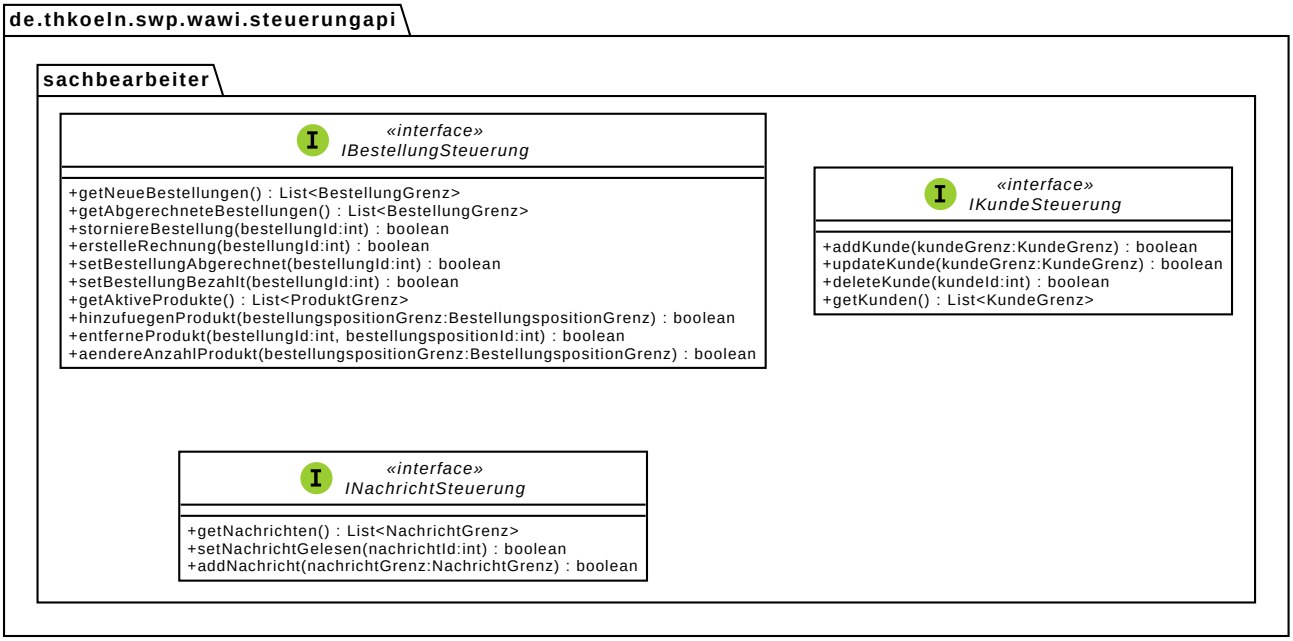

Abbildung 10. Schnittstellen zur Verwendung in der `SachbearbeiterSteuerung`-Komponente

3.11. BestellungVerwaltungS

3.11.1. Funktionen

Diese Komponente realisiert einige Schnittstellen für den Zugriff auf die Entitätsklassen.

3.11.2. Schnittstellen

IBestellungSach

Abbildung 11. `IBestellungSach`-Schnittstelle zur Verwendung in der `BestellungVerwaltungS`-Komponente

Grobe Spezifikation der Methoden:

`getAlleNeuenBestellungen`

liefert alle neuen Bestellungen.

`getAlleAbgerechnetenBestellungen`

liefert alle Bestellungen mit dem Status *abgerechnet*.

`getProduktById`

liefert das Produkt mit der angegebenen Id.

`setBestellungRechnungGesendet`

setzt den Status einer Bestellung auf *verschickt*.

`setBestellungRechnungBezahlt`

setzt den Status einer Bestellung auf *bezahlt*.

`setBestellungRechnungStorniert`

setzt den Status einer Bestellung auf *storniert*.

`updateBestellung`

modifiziert eine existierende Bestellung.

`deleteBestellung`

löscht eine existierende Bestellung.

`getBestellungById`

liefert die Bestellung mit der angegebenen Id.

`insertBestellposition`

fügt eine neue Bestellposition ein.

`updateBestellposition`

modifiziert eine existierende Bestellposition.

`deleteBestellposition`

löscht eine existierende Bestellposition.

`getBestellpositionById`

liefert die Bestellposition mit der angegebenen Id.

Die detaillierte Spezifikation des erwarteten Verhaltens der Methoden dieser Schnittstelle sind in der Testspezifikation gegeben.

IBestellungAdmin

Abbildung 12. `IBestellungAdmin`-Schnittstelle zur Verwendung in der `BestellungVerwaltungS`-Komponente

Grobe Spezifikation der Methoden:

`getAlleBestellungen`

liefert alle Bestellungen.

`getBestellungenByDatum`

liefert alle Bestellungen aus einem gegebenen Intervall.

`getBestellungById`

liefert die Bestellung mit der angegebenen Id.

Die detaillierte Spezifikation des erwarteten Verhaltens der Methoden dieser Schnittstelle sind in der Testspezifikation gegeben.

INachrichtSach

Abbildung 13. `INachrichtSach`-Schnittstelle zur Verwendung in der `BestellungVerwaltungS`-Komponente

Grobe Spezifikation der Methoden:

`sendNachrichtAnKunde`
: legt eine neue Nachricht an mit Id des Kunden.

`getAlleNachrichten`
: liefert alle Nachrichten.

`setNachrichtGelesen`
: setzt den Status einer Nachricht auf *gelesen*.

`rechnungAnKundeSenden`
: sendet eine Nachricht mit Inhalt Rechnung an einen Kunden.

Die detaillierte Spezifikation des erwarteten Verhaltens der Methoden dieser Schnittstelle sind in der Testspezifikation gegeben.

IKundeService

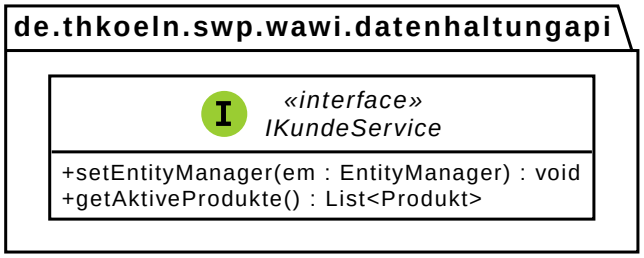

Abbildung 14. `IKundeService`-Schnittstelle zur Verwendung in der `BestellungVerwaltungS`-Komponente

Grobe Spezifikation der Methoden:

`getAktiveProdukte`
: liefert alle Produkte mit dem Status *aktiv* .

Die detaillierte Spezifikation des erwarteten Verhaltens der Methoden dieser Schnittstelle sind in der Testspezifikation gegeben.

3.12. KundeGUI

3.12.1. Funktionen

Die `KundeGUI`-Komponente stellt die Benutzeroberfläche für alle Anwendungsfälle des Kundeen zur Verfügung.

Sie soll mittels JavaFX realisiert werden.

Die `KundeApp`-Klasse und `KundeController`-Klasse sowie eine `kunde.fxml`-Datei werden wie folgt vorgegeben:

- im `src/main/java`-Ordner:

- de.thkoeln.swp.wawi.kundegui.application.KundeApp
- de.thkoeln.swp.wawi.kundegui.control.KundeController

- im src/main/resources-Ordner:
  - kunde.fxml

Der weitere Feinentwurf dieser Komponente ist nicht vorgegeben und soll von dem Kunde-Team eigenständig erstellt werden.

3.12.2. Schnittstellen

Diese Komponente bietet keine implementierten Schnittstellen für andere Komponenten an.

3.13. KundeSteuerung

3.13.1. Funktionen

Diese Komponente realisiert die Anwendungsfälle: /LF150/, /LF160/ und /LF170/.

Zur Realisierung der Anwendungsfälle darf nicht direkt auf die Datenbank zugegriffen werden. Es dürfen nur die Schnittstellen verwendet werden, für die in Abbildung 2 der Zugriff definiert wurde.

3.13.2. Schnittstellen

IActivateComponent

Die KundeSteuerung-Komponente implementiert die IActivateComponent - Schnittstelle (und exportiert diese Implementierung), die von der Bootloader-Komponente verwendet wird, um die Komponente zu aktivieren und zu deaktivieren. Die Implementierung muss dafür im Paket de.thkoeln.swp.wawi.kundesteuerung.impl mit dem Namen IActivateComponentImpl liegen, da diese sonst vom Bootloader nicht gefunden werden kann.

Die Schnittstellen-Klasse IActivateComponent wird von der Komponente ComponentController bereitgestellt.

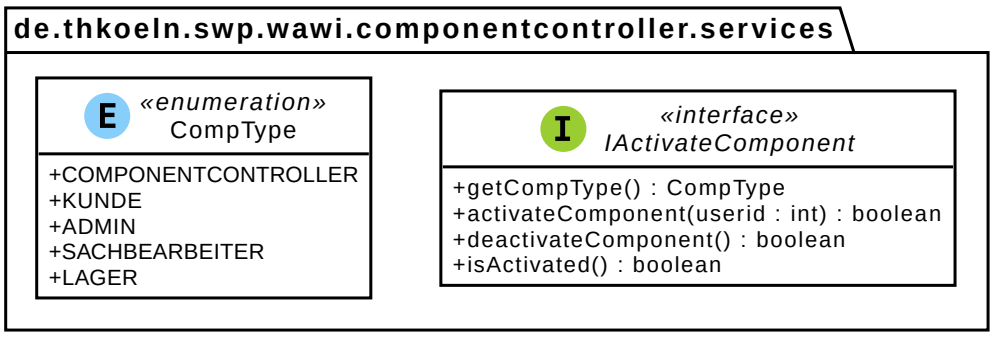

Abbildung 15. IActivateComponent-Schnittstelle zur Verwendung in der KundeSteuerung-Komponente

Der Parameter der activateComponent-Funktion der IActivateComponent-Schnittstelle enthält die jeweilige ID des Kunden, für den die Komponente aktiviert werden soll. Falls eine falsche ID

übergeben wird, sollte die Komponente nicht aktiviert werden und `false` zurückliefern. Zur Aktivierung der Komponente für einen Beispiel-Kunden mit `userid = 14` und `name = Max Mustermann` wird die `activateComponent`-Methode mit `14` aufgerufen. Die Komponente sollte nur dann aktiviert werden, wenn dieser Kunde in der Datenbank bereits existiert.

Die Komponente kann die beiden Zustände *Aktiviert* und *Deaktiviert* annehmen. Die Methoden der Schnittstelle `IActivateComponent` können zu Zustandsübergängen führen:

- Wird die Komponente gestartet, so befindet sie sich zunächst im Zustand *Deaktiviert*.
- Wird im Zustand *Deaktiviert* `activateComponent()` mit zulässiger ID aufgerufen, gibt die Methode ein `true` zurück und geht in den Zustand *Aktiviert*.
- Wird `activateComponent()` mit einer nicht zulässigen ID aufgerunfen, wird nur `false` zurück gegeben.
- Wird im Zustand *Aktiviert* `activateComponent()` aufgerufen, wird ein `false` zurück gegeben, die Komponenten bleibt im Zustand *Aktiviert*.
- Wird im Zustand *Aktiviert* `deactivateComponent()` aufgerufen, gibt die Methode ein `true` zurück und geht in den Zustand *Deaktiviert*.
- Wird im Zustand *Deaktiviert* `deactivateComponent()` aufgerufen, wird ein `false` zurück gegeben, die Komponenten bleibt im Zustand *Deaktiviert*.
- Wird im Zustand *Deaktiviert* die Methode `getCompType()` aufgerufen, liefert sie den `CompType KUNDE` als Ergebnis.
- Wird im Zustand *Aktiviert* die Methode `getCompType()` aufgerufen, liefert sie den `CompType KUNDE` als Ergebnis.
- Wird im Zustand *Deaktiviert* die Methode `isActivated()` aufgerufen, wird ein `false` zurück gegeben.
- Wird im Zustand *Aktiviert* die Methode `isActivated()` aufgerufen, wird ein `true` zurück gegeben.

Schnittstellen

Die Schnittstellen-Klassen `INachrichtSteuerung` und `IBestellungSteuerung` werden von der Komponente `SteuerungAPI` bereitgestellt.

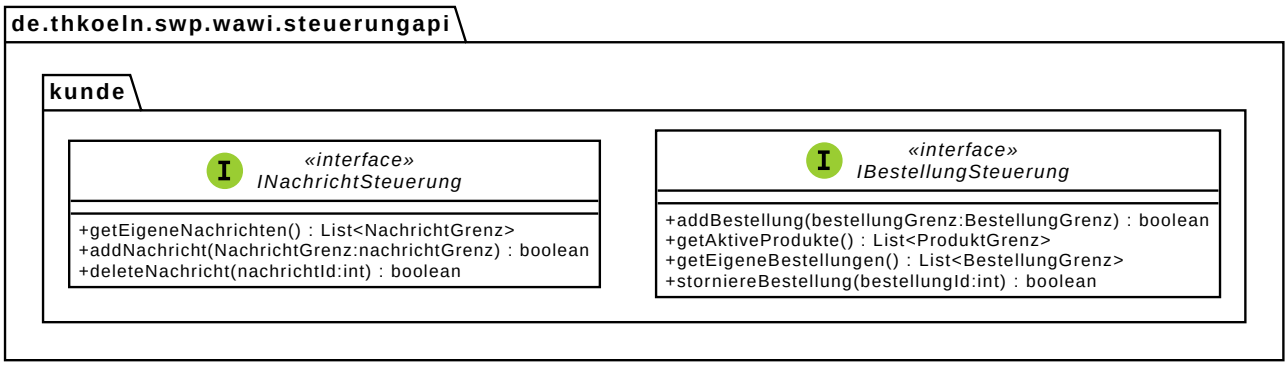

Abbildung 16. Schnittstellen zur Verwendung in der `KundeSteuerung`-Komponente

3.14. KundeDaten

3.14.1. Funktionen

Diese Komponente realisiert einige Schnittstellen für den Zugriff auf die Entitätsklassen.

3.14.2. Schnittstellen

ICRUDKunde

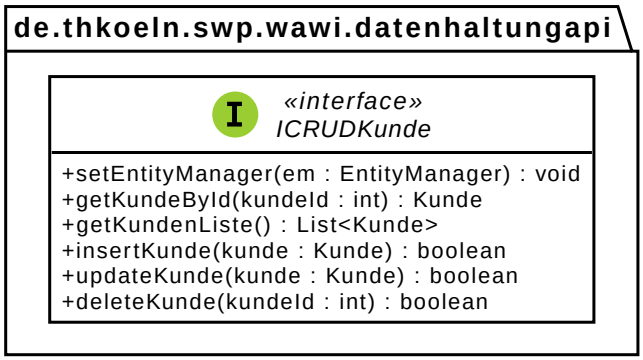

Abbildung 17. `ICRUDKunde`-Schnittstelle zur Verwendung in der `KundeDaten`-Komponente

Grobe Spezifikation der Methoden:

`getKundeById`
: liefert den Kunden mit der angegebenen Id.

`getKundenListe`
: liefert alle Kunden als Liste.

`insertKunde`
: fügt einen neuen Kunden ein.

`updateKunde`
: modifiziert einen existierenden Kunden.

`deleteKunde`
: löscht einen existierenden Kunden.

Die detaillierte Spezifikation des erwarteten Verhaltens der Methoden dieser Schnittstelle sind in der Testspezifikation gegeben.

3.15. BestellungVerwaltungK

3.15.1. Funktionen

Diese Komponente realisiert einige Schnittstellen für den Zugriff auf die Entitätsklassen.

3.15.2. Schnittstellen

IBestellungKunde

Abbildung 18. `IBestellungKunde`-Schnittstelle zur Verwendung in der `BestellungVerwaltungK`-Komponente

Grobe Spezifikation der Methoden:

`bestellungAnlegen`
: fügt eine neue Bestellung mit Status *neu* ein.

`getAlleBestellungenDesKunden`
: liefert alle Bestellungen des Kunden als Liste.

`setBestellungStorniert`
: setzt den Status einer Bestellung auf *storniert*.

`getProduktById`
: liefert das Produkt mit der angegebenen Id.

Die detaillierte Spezifikation des erwarteten Verhaltens der Methoden dieser Schnittstelle sind in der Testspezifikation gegeben.

INachrichtKunde

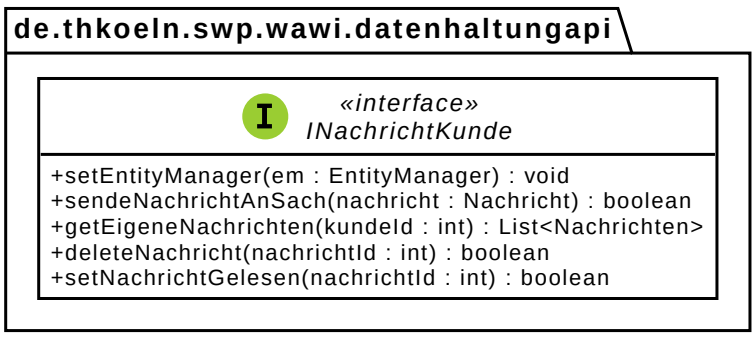

Abbildung 19. `INachrichtKunde`-Schnittstelle zur Verwendung in der `BestellungVerwaltungK`-Komponente

Grobe Spezifikation der Methoden:

`sendeNachrichtAnSach`
: erzeugt eine Nachricht an den Sachbearbeiter in der DB aus der übergebenen Nachricht (typ = *anwawi*, status = *ungelesen*)

`getEigeneNachrichten`

liefert alle Nachrichten des Kunden als Liste.

`deleteNachricht`

löscht eine existierende Nachricht.

`setNachrichtGelesen`

setzt den Status einer Nachricht auf *gelesen*.

Die detaillierte Spezifikation des erwarteten Verhaltens der Methoden dieser Schnittstelle sind in der Testspezifikation gegeben.

3.16. LagerGUI

3.16.1. Funktionen

Die `LagerGUI`-Komponente stellt die Benutzeroberfläche für alle Anwendungsfälle des Lagerhalters zur Verfügung.

Sie soll mittels JavaFX realisiert werden.

Die `LagerApp`-Klasse und `LagerController`-Klasse sowie eine `lager.fxml`-Datei werden wie folgt vorgegeben:

- im `src/main/java`-Ordner:
  - `de.thkoeln.swp.wawi.lagergui.application.LagerApp`
  - `de.thkoeln.swp.wawi.lagergui.control.LagerController`
- im `src/main/resources`-Ordner:
  - `lager.fxml`

Der weitere Feinentwurf dieser Komponente ist nicht vorgegeben und soll von dem `Lager`-Team eigenständig erstellt werden.

3.16.2. Schnittstellen

Diese Komponente bietet keine implementierten Schnittstellen für andere Komponenten an.

3.17. LagerSteuerung

3.17.1. Funktionen

Diese Komponente realisiert die Anwendungsfälle: /LF10/, /LF15/, /LF21/, /LF22/, /LF23/, /LF34/, /LF25/, /LF27/, /LF30/, /LF40/ und /LF45/.

Zur Realisierung der Anwendungsfälle darf nicht direkt auf die Datenbank zugegriffen werden. Es dürfen nur die Schnittstellen verwendet werden, für die in Abbildung 2 der Zugriff definiert wurde.

3.17.2. Schnittstellen

IActivateComponent

Die `LagerSteuerung`-Komponente implementiert die `IActivateComponent` - Schnittstelle (und exportiert diese Implementierung), die von der `Bootloader`-Komponente verwendet wird, um die Komponente zu aktivieren und zu deaktivieren. Die Implementierung muss dafür im Paket `de.thkoeln.swp.wawi.lagersteuerung.impl` mit dem Namen `IActivateComponentImpl` liegen, da diese sonst vom Bootloader nicht gefunden werden kann.

Die Schnittstellen-Klasse `IActivateComponent` wird von der Komponente `ComponentController` bereitgestellt.

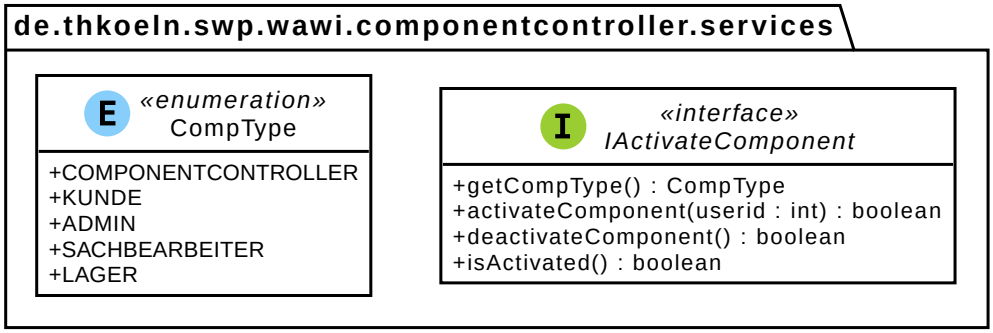

Abbildung 20. `IActivateComponent`-Schnittstelle zur Verwendung in der `LagerSteuerung`-Komponente

Der Parameter der `activateComponent`-Funktion der `IActivateComponent`-Schnittstelle enthält die ID des Lagerhalters, für den die Komponente aktiviert werden soll. Falls eine falsche ID übergeben wird, sollte die Komponente nicht aktiviert werden und `false` zurückliefern. Die Komponente darf nur für den Default-Lagerhalter (`userid = 10`) aktiviert werden.

Die Komponente kann die beiden Zustände *Aktiviert* und *Deaktiviert* annehmen. Die Methoden der Schnittstelle `IActivateComponent` können zu Zustandsübergängen führen:

- Wird die Komponente gestartet, so befindet sie sich zunächst im Zustand *Deaktiviert*.
- Wird im Zustand *Deaktiviert* `activateComponent()` mit zulässiger ID aufgerufen, gibt die Methode ein `true` zurück und geht in den Zustand *Aktiviert*.
- Wird `activateComponent()` mit einer nicht zulässigen ID aufgerunfen, wird nur `false` zurück gegeben.
- Wird im Zustand *Aktiviert* `activateComponent()` aufgerufen, wird ein `false` zurück gegeben, die Komponenten bleibt im Zustand *Aktiviert*.
- Wird im Zustand *Aktiviert* `deactivateComponent()` aufgerufen, gibt die Methode ein `true` zurück und geht in den Zustand *Deaktiviert*.
- Wird im Zustand *Deaktiviert* `deactivateComponent()` aufgerufen, wird ein `false` zurück gegeben, die Komponenten bleibt im Zustand *Deaktiviert*.
- Wird im Zustand *Deaktiviert* die Methode `getCompType()` aufgerufen, liefert sie den `CompType LAGER` als Ergebnis.
- Wird im Zustand *Aktiviert* die Methode `getCompType()` aufgerufen, liefert sie den `CompType LAGER` als Ergebnis.

- Wird im Zustand *Deaktiviert* die Methode `isActivated()` aufgerufen, wird ein `false` zurück gegeben.
- Wird im Zustand *Aktiviert* die Methode `isActivated()` aufgerufen, wird ein `true` zurück gegeben.

Schnittstellen

Die Schnittstellen-Klassen `IBestellungSteuerung`, `IEinlieferungSteuerung`, `ILagerSteuerung` und `ILieferantSteuerung` werden von der Komponente `SteuerungAPI` bereitgestellt.

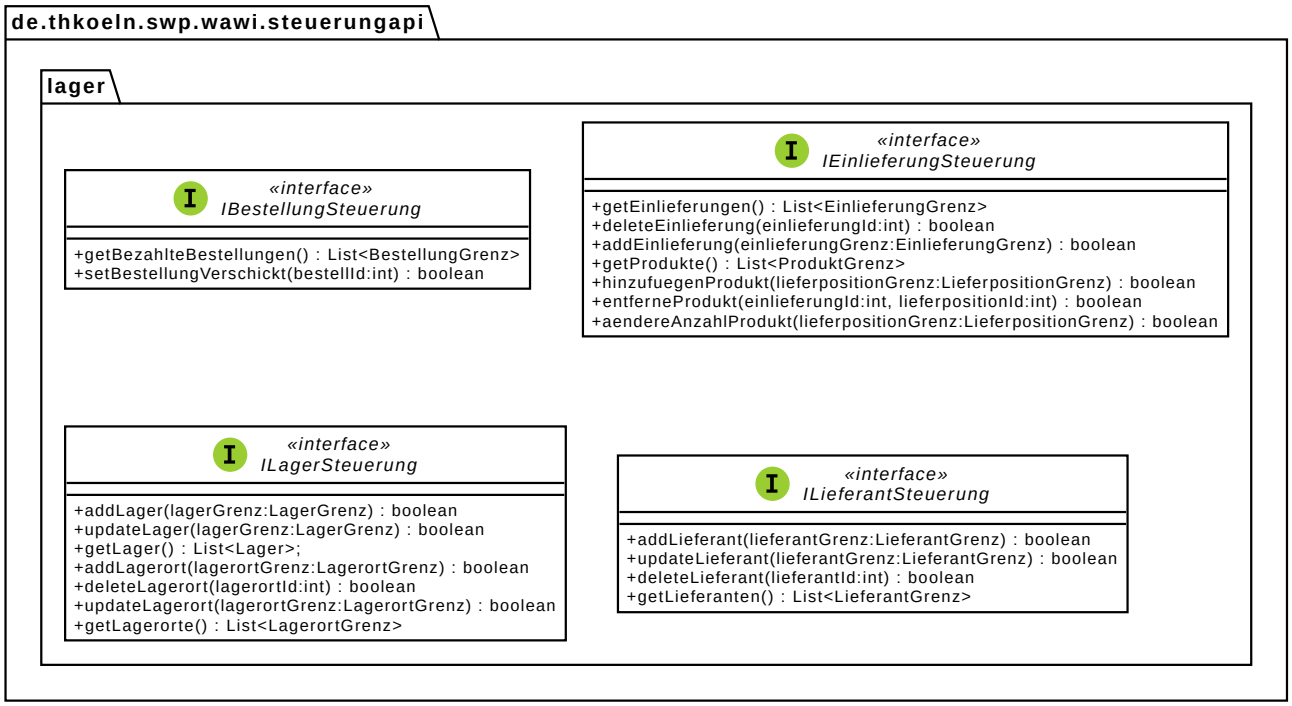

Abbildung 21. Schnittstellen zur Verwendung in der `KundeSteuerung`-Komponente

3.18. LagerDaten

3.18.1. Funktionen

Diese Komponente realisiert einige Schnittstellen für den Zugriff auf die Entitätsklassen.

3.18.2. Schnittstellen

ICRUDLieferant

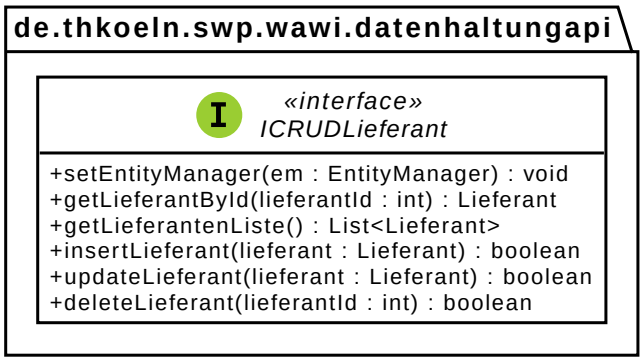

Abbildung 22. `ICRUDLieferant`-Schnittstelle zur Verwendung in der `LagerDaten`-Komponente

Grobe Spezifikation der Methoden:

`getLieferantById`

liefert den Lieferanten mit der angegebenen Id.

`getLieferantenListe`

liefert alle Lieferanten als Liste.

`insertLieferant`

fügt einen neuen Lieferanten ein.

`updateLieferant`

modifiziert einen existierenden Lieferanten.

`deleteLieferant`

löscht einen existierenden Lieferanten.

Die detaillierte Spezifikation des erwarteten Verhaltens der Methoden dieser Schnittstelle sind in der Testspezifikation gegeben.

ILagerService

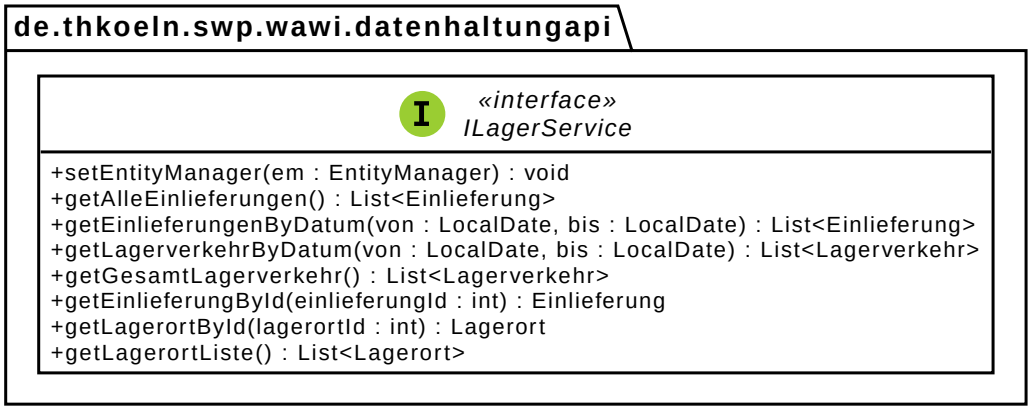

Abbildung 23. `ILagerService`-Schnittstelle zur Verwendungin der `LagerDaten`-Komponente

Grobe Spezifikation der Methoden:

`getAlleEinlieferungen`

liefert alle Einlieferungen als Liste.

`getEinlieferungenByDatum`

liefert alle Einlieferungen aus einem gegebenen Intervall.

`getLagerverkehrBydatum`

liefert alle Lagerverkehre aus einem gegebenen Intervall..

`getGesamtLagerverkehr`

liefert alle Lagerverkehre als Liste.

`getEinlieferungById`

liefert die Einlieferung mit der angegebenen Id.

`getLagerortById`

liefert den Lagerort mit der angegebenen Id.

`getLagerortListe`

liefert alle Lagerorte als Liste.

Die detaillierte Spezifikation des erwarteten Verhaltens der Methoden dieser Schnittstelle sind in der Testspezifikation gegeben.

ICRUDEinlieferung

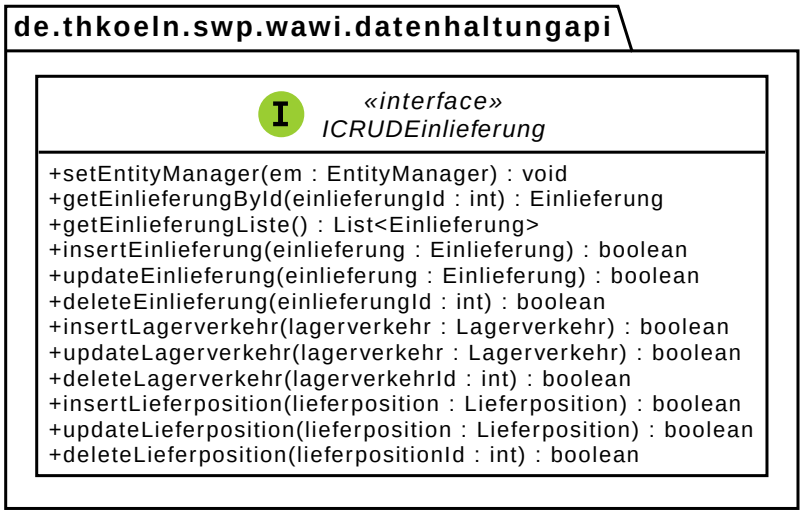

Abbildung 24. `ICRUDEinlieferung`-Schnittstelle zur Verwendung in der `LagerDaten`-Komponente

Grobe Spezifikation der Methoden:

`getEinlieferungById`

liefert die Einlieferung mit der angegebenen Id.

`getEinlieferungListe`

liefert alle Einlieferung als Liste.

`insertEinlieferung`

fügt eine neue Einlieferung ein.

`updateEinlieferung`

modifiziert eine existierende Einlieferung.

`deleteEinlieferung`

löscht eine existierende Einlieferung.

`insertLagerverkehr`

fügt einen neuen Lagerverkehr ein.

`updateLagerverkehr`

modifiziert einen existierenden Lagerverkehr.

`deleteLagerverkehr`

löscht einen existierenden Lagerverkehr.

`insertLieferposition`

fügt eine neue Lieferposition ein.

`updateLieferposition`

modifiziert eine existierende Lieferposition.

`deleteLieferposition`

löscht eine existierende Lieferposition.

Die detaillierte Spezifikation des erwarteten Verhaltens der Methoden dieser Schnittstelle sind in der Testspezifikation gegeben.

ICRUDLager

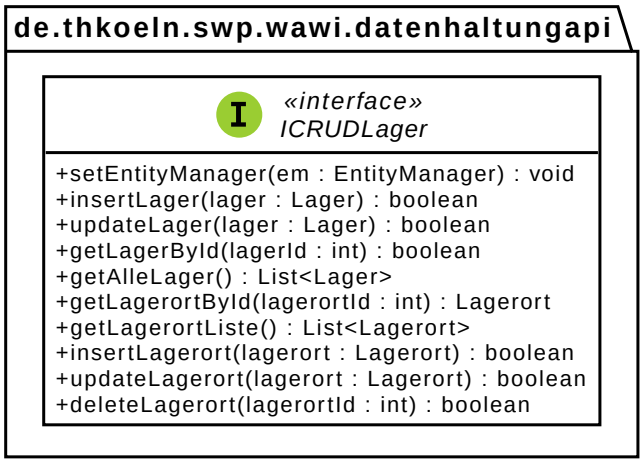

Abbildung 25. `ICRUDLager`-Schnittstelle zur Verwendung in der `LagerDaten`-Komponente

Grobe Spezifikation der Methoden:

`insertLager`

fügt ein neues Lager ein.

`updateLager`

modifiziert ein existierendes Lager.

`getLagerById`

liefert das Lager mit der angegebenen Id.

`getAlleLager`

liefert alle Lager als Liste.

`getLagerortById`

liefert den Lagerort mit der angegebenen Id.

`insertLagerort`

fügt einen neuen Lagerort ein.

`updateLagerort`

modifiziert einen existierenden Lagerort.

`deleetLagerort`

löscht einen existierenden Lagerort.

Die detaillierte Spezifikation des erwarteten Verhaltens der Methoden dieser Schnittstelle sind in der Testspezifikation gegeben.

ILieferBestellung

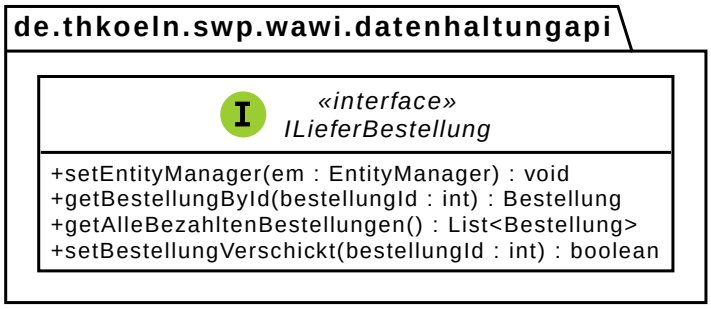

Abbildung 26. `ILieferBestellung`-Schnittstelle zur Verwendung in der `LagerDaten`-Komponente

Grobe Spezifikation der Methoden:

`getBestellungById`

liefert die Bestellung mit der angegebenen Id.

`getAlleBezahltenBestellungen`

liefert alle Bestellungen mit Status *bezahlt* als Liste.

`setBestellungVerschickt`

setzt den Status einer Bestellung auf *verschickt*.

Die detaillierte Spezifikation des erwarteten Verhaltens der Methoden dieser Schnittstelle sind in der Testspezifikation gegeben.