Softwaretechnik II – Praktikum

Subsystem 4 – Zubereitung

Eine Dokumentation von:

J. Faßbender

J. Gobelet

L. Gobelet

E. Gödel

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Meilenstein 1 – Datenzugriffsschicht

1.1 Teilaufgabe 1: Ausschnitt aus Logischem DM mit Entities und Value Objects

1.1.1 Klassendiagramm

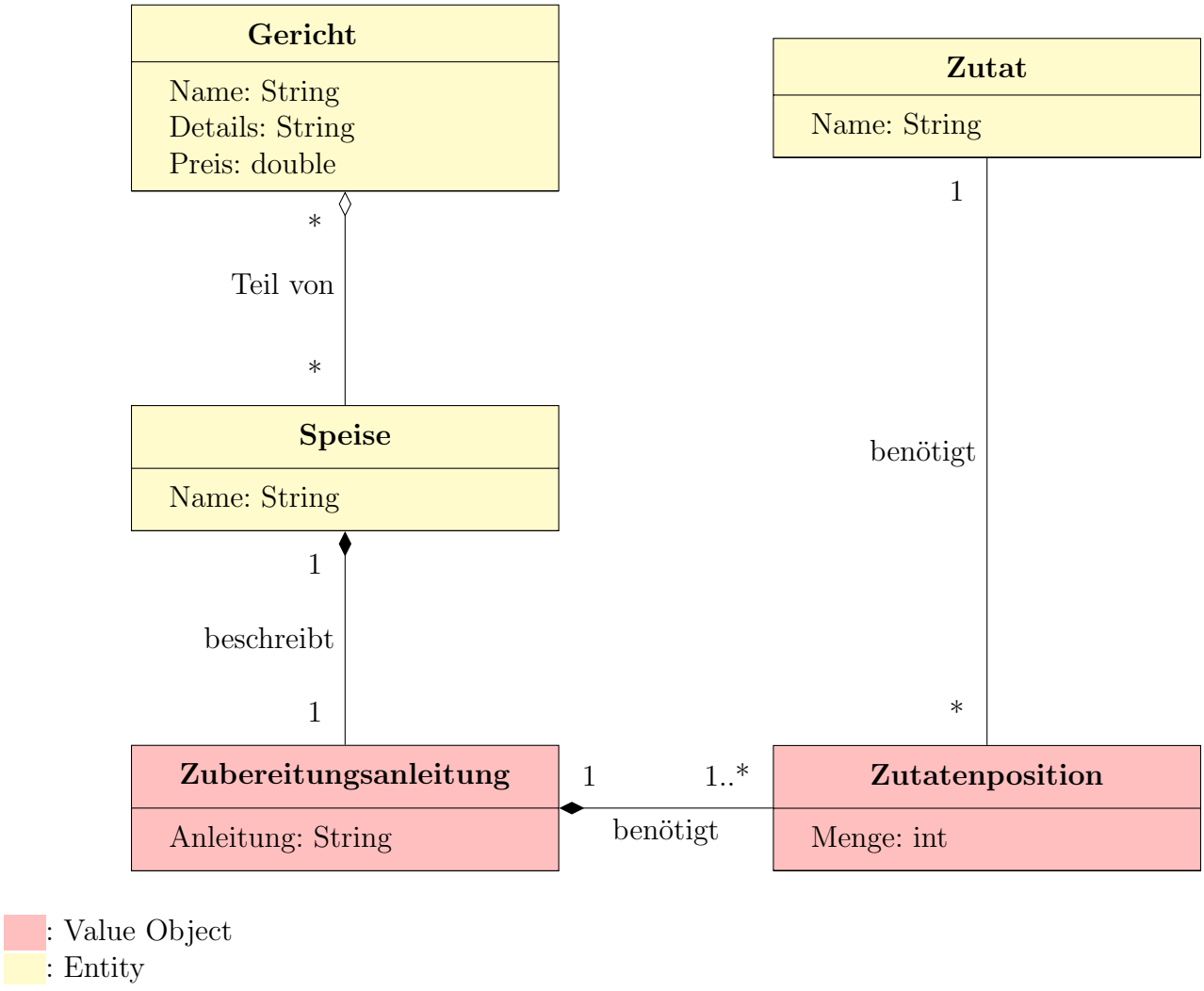

Abbildung 1: Klassendiagramm

1.1.2 Fachliches Glossar

Geschäftsobjekt	Attribut	Erklärung
Gericht		Vom Restaurant angebotenes Mahl.
	Name	Gerichtsbezeichnung.
	Details	Wird dem Gast angezeigt. Enthält nähere Angaben zu den Zutaten.
	Preis	Geldbetrag der für das Gericht zu bezahlen ist.
Speise		Teil eines Gerichts. Beispielsweise wäre eine Salatbeilage als Speise zu verstehen.
	Name	Bezeichnung der Speise.
Zubereitungsanleitung		Leitfaden zur Zubereitung einer Speise.
	Anleitung	Erklärender Text, der beschreibt, wie eine Speise zuzubereiten ist.
Zutat		Benötigt für die Zubereitung einer Speise.
	Name	Bezeichnung der Zutat.
Zutatenposition		Zuordnung zwischen Zutat und Zubereitungsanleitung. Gibt die Menge einer Zutat an, die für die Zubereitung notwendig ist.
	Menge	Die benötigte Menge.

1.1.3 Erweiterungen der Aufgabenstellung

Da es in unserem Logischen Datenmodell keine 1:1-Beziehung gab, haben wir eine zusätzliche redundante Entität eingebaut.

Hierbei handelt es sich um die Entität Speise. Diese Entität hätte genauso gut einfach Teil der Zubereitungsanleitung sein können und ist nur in unser Modell aufgenommen worden, damit wir die für die Aufgabenstellung benötigte 1:1-Beziehung in unserem Diagramm haben.

1.1.4 Erläuterungen

Wir haben Zubereitungsanleitung als Value Object und nicht als Entity deklariert, da hier unserer Meinung nach Sharing nicht sinnvoll ist und ein Zubereitungsanleitungsobjekt deshalb persistent als Teil der zugeordneten Speise in der Datenbank gespeichert werden sollte.

Gleiches gilt für die Zutatenposition.

1.2 Teilaufgabe 2: Entities und Value Objects mit JPA-Annotierung

1.2.1 Annotationen der Entities und Value Objects

Gericht

```
@Entity
public class Gericht {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private int id;
  private String name;
  private String details;
  private double preis;

  // Ein Gericht besteht aus mehreren Speisen und eine Speise kann
   mehreren Gerichten zugeordnet sein.
  @ManyToMany
  @JoinTable(name = "gericht_speise",
    joinColumns = @JoinColumn(name = "gericht_id"),
    inverseJoinColumns = @JoinColumn(name = "speise_id")
  )
  private Set<Speise> speisen = new HashSet<Speise>();
```

Speise

```
@Entity
public class Speise {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private int id;
  private String name;

  // bidirektionale Beziehung: Gericht kennt zugehoerige Speisen und
    die Speisen kennen zugehoerige Gerichte
  @ManyToMany(mappedBy = "speisen")
  private Set<Gericht> gerichte = new HashSet<Gericht>();
```

Zubereitungsanleitung

```
@Embeddable
public class Zubereitungsanleitung {
  private String anleitung;

  // Die Anleitung enthaelt mehrere Zutatenangaben als Value-Objects
  @ElementCollection (targetClass = Zutatenmenge.class, fetch =
```

```
    FetchType.EAGER)
  @CollectionTable(name = "ZUTATENANGABE")
  private Set<Zutatenmenge> angaben = new HashSet<Zutatenmenge>();
```

Zutat

```
@Entity
public class Zutat {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private int id;
  private String name;
```

Zutatenposition

```
@Embeddable
public class Zutatenmenge {
  private int menge;

  @ManyToOne
  private Zutat zutat;
```

1.2.2 H2-Console

SELECT * FROM GERICHT;

ID	DETAILS	NAME	PREIS
10	Voll das Oma-Essen!	Kartoffelbrei mit Möhren	7.5
11	Jede Erbse macht einen Knall!	Kartoffelbrei mit Erbsen	8.5

(2 rows, 9 ms)

Abbildung 2: Gerichtstabelle

SELECT * FROM SPEISE;

ID	ANLEITUNG	NAME
7	Möhren und Pfeffer umrühren!	Möhrengemüse
8	Erbsen, Salz und Pfeffer verbrennen lassen!	Erbsengemüse
9	Kartoffeln, Salz und Butter vermatschen!	Kartoffelbrei

(3 rows, 3 ms)

Abbildung 3: Speisentabelle

SELECT * FROM ZUTAT;

ID	NAME
1	Erbse
2	Butter
3	Salz
4	Möhre
5	Pfeffer
6	Kartoffel

(6 rows, 1 ms)

Abbildung 4: Zutatentabelle

SELECT * FROM ZUTATENMENGE;

SPEISE_ID	MENGE	ZUTAT_ID
7	1	5
7	3	4
8	100	1
8	2	3
8	5	5
9	6	6
9	5	3
9	2	2

(8 rows, 8 ms)

Abbildung 5: Zutatenpositionstabelle

SELECT * FROM GERICHT_SPEISE;

GERICHT_ID	SPEISE_ID
10	7
10	9
11	8
11	9

(4 rows, 1 ms)

Abbildung 6: Zuordnungstabelle Gericht - Speise

1.3 Teilaufgabe 3: Factories und Repositories

Factory für Erstellung von Gerichten

```
@Component
public class GerichtFactory {

  // Erstelle ein Gericht, das nur aus einer Speise besteht.
  public static Gericht createGerichtWithSpeise(String name, String
   details, double preis, Speise speise) {
    Gericht gericht = new Gericht(name,details, preis);
```

```
    gericht.addSpeise(speise);
    // Rueckreferenz setzen
    speise.addGericht(gericht);
    return gericht;
  }

  // Erstelle ein Gericht, das aus mehreren Speisen besteht.
  public static Gericht createGerichtWithSpeisen(String name, String
    details, double preis, Collection<Speise> speisen) {
    Gericht gericht = new Gericht(name,details, preis);
    gericht.addSpeisen(speisen);
    for(Speise s : speisen) {
      // Rueckreferenz setzen
      s.addGericht(gericht);
    }
    return gericht;
  }
}
```

Hier sieht man gut, warum Factories notwendig sind. Bei der Erstellung von Gerichten muss zugleich die Rückreferenz von Speise auf Gericht gesetzt werden.

Factory für Erstellung von Gerichten

```
public interface SpeiseRepository extends CrudRepository<Speise,
   Integer> {
  // Die Abfrage ist in JPQL geschrieben - Eine objektorientierte
   Abfragesprache, welche SQL aehnlich ist
  // Findet alle Speisen, die eine bestimmte Zutat enthalten
  @Query("select s from Speise s join s.anleitung a join a.angaben
   ang where ang.zutat = :zutat")
  List<Speise> findByContainsZutat(@Param("zutat")Zutat zutat);
}
```

Ausgabe in der Konsole

```
    // gib alle Speisen aus, die Salz enthalten
    System.out.println("\nSalzige Speisen: ");
    speiseRepository.findByContainsZutat(zutaten.get("Salz")).
   forEach(s -> System.out.println(s.getName()));
```

Folgendes wird dann in der Konsole ausgegeben:

```
Salzige Speisen:
Erbsengemüse
Kartoffelbrei
```

Abbildung 7: Ausgabe in der Konsole

2 Meilenstein 2 – Komponentenschnitt

2.1 Teilaufgabe 1: Vorbereitung des Komponentenschnitts

2.1.1 Liste der Geschäftsobjekte

- Arbeitsplatz
- Bestellung
- Gericht
- Sitzplatz
- Speisekarte
- Zubereitungsanleitung
- Zutat
- Zutatenposition

2.1.2 Liste der Use Cases

- Am Arbeitsplatz an-/abmelden
- Gericht bestellen
- Gericht zubereiten

2.1.3 Liste der Umsysteme

Umsystem	Was geschieht zwischen Umsystem und unserem Subsystem?	Schnittstelle angeboten oder aufgerufen
Rezeptverwaltung	Rezeptverwaltung verwaltet die Geschäftsobjekte Gericht, Zubereitungsanleitung und Speisekarte. Der Gast fragt über das ihm zur Verfügung gestellte Frontend die Speisekarte und die Gerichte ab, während der Koch an seinem Terminal die Zubereitungsanleitung und die hiermit verbundenen Zutatenpositionen, angezeigt bekommt.	Aufruf einer Schnittstelle zur Rezeptverwaltung
Lagerverwaltung	Abfrage zum Zutatenbestand	Aufruf einer Schnittstelle zur Lagerverwaltung
Lagerverwaltung	Angabe zur Zutantenentnahme (kann auch über die gleiche Schnittstelle, die im obrigen Tabelleneintrag spezifiziert ist, realisiert werden)	Aufruf einer Schnittstelle zur Lagerverwaltung
Buchhaltung	Abfrage der Bestellungen	Schnittstelle wird Buchhaltung zur verfügung gestellt

Erläuterung

Wir legen redundant zur Lagerverwaltung unsere eingene Verwaltung mit Angaben zum Zutatenbestand an, um auch bei Nichterreichbarkeit der Lagerverwaltung funktionsfähig zu bleiben, da unser Subsystem essentiell für den Umsatz verantwortlich ist und ein Ausfall, das heißt in diesem Fall der Zustand, dass eine Zutat nicht mehr in benötigter Menge im Lager zur Verfügung steht, nicht auf Grund technischer Probleme eintreten sollte.

Allerdings stellen wir keinen Anspruch auf absolute Richtigkeit unserer Zutantenbestandsverwaltung, da wir nur die Ereignisse unseres Subsystems, das heißt in diesem Fall die Entnahme einer Zutat zur Zubereitung, protokollieren und die restlichen Angaben aus der Lagerverwaltung stammen.

Ist diese nun nicht erreichbar, verwendet unsere Zutatenbestandsverwaltung mitunter veraltete Daten, was wir nicht mit einbeziehen.

Der Lagerverwaltung wird die Entnahme von unserem Subsystem aus mitgeteilt.

Für den kompletten Synchronisationsprozess zwischen den beiden Systemen stellt uns die Lagerverwaltung zwei Schnittstellen (oder eine, die beide Aufgaben - Entnahme mitteilen und Zutatenbestand abfragen - zusammenfasst) zur Verfügung.

Zusätzlich haben wir eine Schnittstelle für die Buchhaltung angelegt. Diese ist zwar kein explizites Subsystem, wird aber, unserer Meinung nach, im Betriebsumfeld höchstwahrscheinlich als eigenes Subsystem existieren und unsere Schnittstelle zu den Bestellungen (im Endeffekt der Unternehmens umsatz aus dem Hauptgeschäft) nutzen wollen.

2.2 Teilaufgabe 2: Ermittlung der verschiedenen Komponenten-Typen

2.2.1 Schritt 1: Geschäftsobjekte in zusammenhängende Gruppen einteilen

Datenkomponente	Zugeordnete Geschäftsobjekte	Erklärung
Bestelldaten	Bestellung	Die einzigen Daten die in diesem Subsystem tasächlich generiert werden. Da die Bestellungen sehr wichtig für das Hauptgeschäft der Firma ist, es das einzige Datenobjekt mit Implementierung eines Create-Interfaces (Factory) ist und auch sonst nicht in unsere sonstigen Datenkomponenten passt, wird die Bestellung, unserer Meinung nach, in einer eigenen Komponente implementiert.
Standortdaten	Arbeitsplatz, Sitzplatz	Diese Daten ändern sich äußerst selten (und auch nicht in unserem Subsystem) und umfassen im Vergleich zu anderen Komponenten wenig Datensätze und können deshalb, unserer Meinung nach, gut zusammengefasst werden.
Gerichtsdaten	Gericht, Speisekarte, Zubereitungsanleitung, Zutat, Zutatenposition	Stammdaten die für unseren Prozess der Zubereitung essentiell sind. Diese Daten stammen nicht aus unserem Subsystem, sondern sind über Schnittstellen abrufbar, sowohl von der Lagerverwaltung (Zutat), als auch von der Rezepteverwaltung (Gericht, Speisekarte, Zubereitungsanleitung, Zutatenposition). Unsere Datenkomponente greift über Adapterkomponenten auf diese Schnittstellen zu.

2.2.2 Schritt 2: Use Cases auf Daten/Logik analysieren

Daten-/Logikkomponente	Zugeordnete(r) Use Case(s)	Erklärung
Bestellabwicklung (Logik)	Am Arbeitsplatz an-/abmelden, Gericht bestellen, Gericht zubereiten	Unser „Backend“, was ab der Bestellungsaufgabe den Zubereitungsprozess steuert. Die Komponente umfasst die Vergabewarteschlange mit den besetzten und freien Arbeitsplätzen und übernimmt die Zuweisung, sobald eine Bestellung von einem Clienten eingeht. Sobald ein Gericht fertig zubereitet ist und der Koch dies seinem Terminal mitteilt, übernimmt diese Komponente auch die Anzeige der Ordernummer (im Gast-UI). Da dies alles vom Umfang her eher kleinere Aufgaben sind, haben wir uns dazu entschieden, diese Aufgaben in einer Komponente zusammenzufassen.

2.2.3 Schritt 3: Use Cases auf Nutzer-Interaktion analysieren

Dialogkomponente	Zugeordnete(r) Use Case(s)	Eigene Fassadenkomponente sinnvoll?	Erklärung
Zubereitungs-UI	Gericht zubereiten	Ja	Fassadenkomponente zur Orchestrierung der Gerichtszubereitung.
An-/Abmeldungs-UI	Am Arbeitsplatz an-/abmelden	Ja	Fassadenkomponente für den Zugriff auf Datenkomponente „Standortdaten“ (Read- und Update-operationen auf den Arbeitsplatz) und um das „Strict Layering“ einzuhalten.
Gast-UI	Gericht bestellen	Ja	Fassadenkomponente zur Orchestrierung des Bestellvorgangs.

2.2.4 Schritt 4: Angebot von externen Schnittstellen

Umsystem/Schnittstelle	Eigene Fassadenkomponente sinnvoll?	Erklärung
Buchhaltung	Ja	Da die Buchhaltung lesenden Zugriff auf usere Bestellungen haben soll, ist es notwendig eine spezialisierte Komponente hierfür anzulegen und nicht, wie intern in unserem Subsystem, den Zugriff über die Bestelldatenkomponente zu regeln.
Lagerverwaltung	Nein	Zugriff erfolgt nur aus der Gerichtsdatenkomponente über die Adapterkomponente der Lagerverwaltung, weshalb, unserer Meinung nach, keine Fassadenkomponente notwendig ist.
Rezeptverwaltung	Nein	Zugriff erfolgt nur aus der Gerichtsdatenkomponente über die Adapterkomponente der Rezepteverwaltung, weshalb, unserer Meinung nach, keine Fassadenkomponente notwendig ist.

2.2.5 Schritt 5: Aufruf von externen Schnittstellen/Umsystemen

Umsystem/Schnittstelle	Adapterkomponente sinnvoll?	Erklärung
Buchhaltung	Nein	Bereits spezialisierte Fassadenkomponente vorhanden.
Lagerverwaltung	Ja	Adapterkomponente für unsere Gerichtsdatenkomponente, die die Lese- und Schreibvorgänge zur Verfügung stellt und gleichzeitig bei Ausfällen als „Anti-Corruption-Layer“ fungiert.
Rezeptverwaltung	Ja	Adapterkomponente für unsere Gerichtsdatenkomponente, die die Lesevorgänge zur Verfügung stellt und gleichzeitig bei Ausfällen als „Anti-Corruption-Layer“ fungiert.

2.3 Teilaufgabe 3: Komponentendiagramm

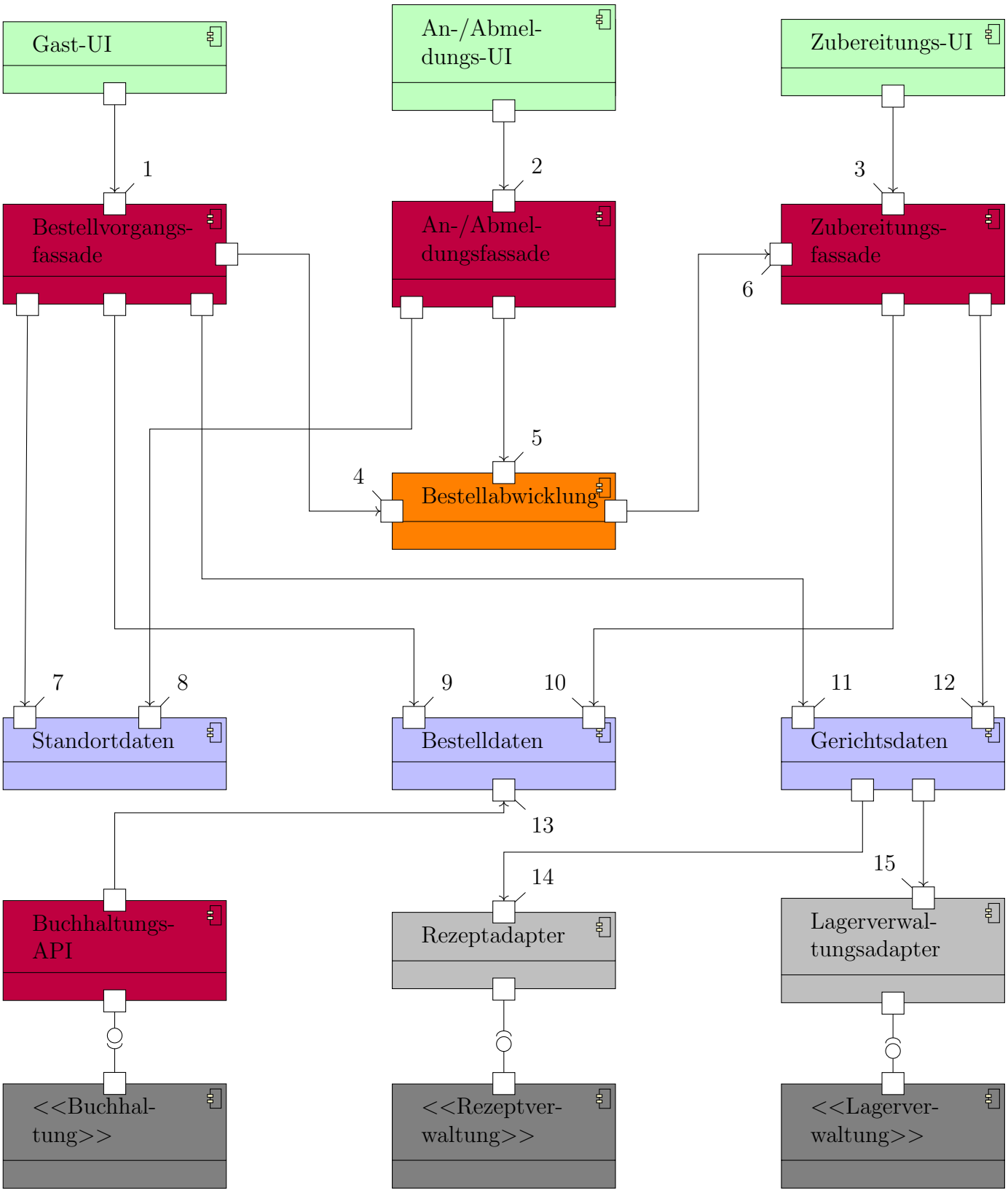

Abbildung 8: Komponentendiagramm

: Dialogkomponente

: Fassadenkomponente

: Datenkomponente

: Logikkomponente

: Adapterkomponente

: Umsystem

Portbeschriftungen

1.	**Gast-UI**	→	**Bestellvorgangsfassade**	:	Bestellvorgang beginnen.
2.	**An-/Abmeldungs-UI**	→	**An-/Abmeldungsfassade**	:	Am Arbeitsplatz an-/abmelden.
3.	**Zubereitungs-UI**	→	**Zubereitungsfassade**	:	Zubereitung beginnen.
4.	**Bestellvorgansfassade**	→	**Bestellabwicklung**	:	Bestellung aufgeben.
5.	**An-/Abmeldefassade**	→	**Bestellabwicklung**	:	In Vergabewarteschlange aufnehmen.
6.	**Bestellabwicklung**	→	**Zubereitungsfassade**	:	Bestellung zubereiten.
7.	**Bestellvorgangsfassade**	→	**Standortdaten**	:	Sitzplatz abfragen.
8.	**An-/Abmeldungsfassade**	→	**Standortdaten**	:	Arbeitsplatz abfragen.
9.	**Bestellvorgangsfassade**	→	**Bestelldaten**	:	Bestellung speichern.
10.	**Zubereitungsfassade**	→	**Bestelldaten**	:	Zuzubereitende Bestellung abfragen.
11.	**Bestellvorgangsfassade**	→	**Gerichtsdaten**	:	Speisekarte und Gerichte abfragen.
12.	**Zubereitungsfassade**	→	**Gerichtsdaten**	:	Zubereitungsanleitung abfragen.
13	**Buchhaltungs-API**	→	**Bestelldaten**	:	Bestellungen abfragen.
14.	**Gerichtsdaten**	→	**Rezeptadapter**	:	Daten aus Subsystem Rezeptverwaltung holen.
15.	**Gerichtsdaten**	→	**Lagerverwaltungsadapter**	:	Daten aus Subsystem Lagerverwaltung holen und synchronisieren.

3 Meilenstein 3 – Spezifikation, Implementierung und Demo eines REST-API

3.1 Teilaufgabe 1: Festlegen von Aggregates

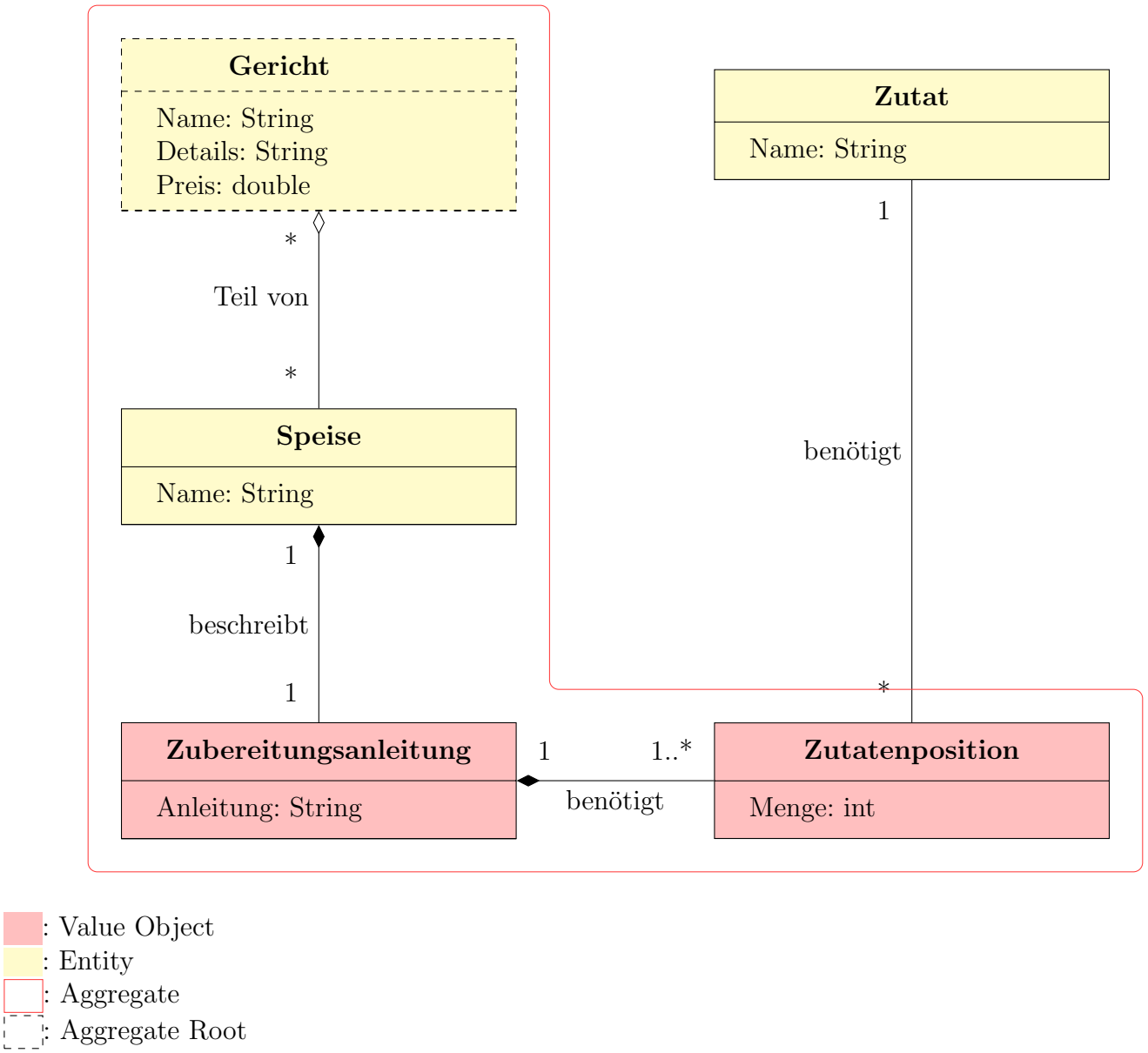

Abbildung 9: Aggregates

Wir sind der Meinung, dass sich die Datenobjekte Gericht, Speise, Zubereitungsanleitung und Zutatenposition als ein Aggregate mit Gericht als Aggregate Root zusammenfassen lassen, da keine Referenzen auf innere Entities existieren und ein fachlicher Zusammenhang besteht, da ein Gericht aus Speisen besteht, Speisen eine Zubereitungsanleitung haben und diese wiederum Zutatenpositionen, ergibt sich hier ein enges fachliches Geflecht.

Eine mögliche Invariante wäre, wenn *Gericht.name* eine Kombination von den zugehörigen Speisen wäre. Als Beispiel hierfür: *Gericht.name*: „Schnitzel mit Pommes und Salat“. Daraus lassen sich die Speisen Schnitzel, Pommes und Salat ableiten.

3.2 Teilaufgabe 2: Design des REST-API

Für unser REST-API verwenden wir folgenden Ausschnitt aus unserem Klassendiagramm aus Meilenstein 1:

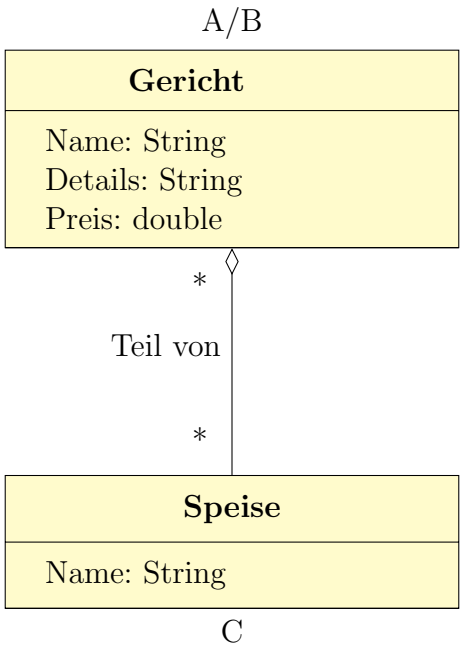

Abbildung 10: Ausschnitt Klassendiagramm für REST-API

Szen.-Nr.	URI	HTTP Verb	Request-Body	Ressource und Aktion
A1, A5, BC1	/gerichte/{gericht_id}	PUT, DELETE	**Nur bei Put:** { name=..., details=..., preis=...}	Neues Gericht anlegen, Gericht überschreiben, Gericht löschen.
A2, A4	/gerichte?filter[preis]>{wert}	GET		Alle Gerichte a ausgeben, für die $a.preis > wert$ gilt.
A3	/gerichte/{gericht_id}/preis	PUT	{wert}	Preis von Gericht a mit $a.gericht\_id = \{gericht\_id\}$ auf $\{wert\}$ setzen.
A6	/gerichte/{gericht_id}/id	GET		Id von Gericht abfragen. 404 wird geworfen, falls Gericht nicht vorhanden.
BC2	/speisen/{speise_id}	PUT	{name=...}	Speise anlegen, überschreiben.
BC3, BC6	/gerichte/{gericht_id}/speisen	PUT, DELETE	{speise_id=...}	Speise einem Gericht hinzufügen oder löschen.
BC4, BC7	/gerichte	GET		Alle Gerichte ausgeben.
BC5	/speisen	GET		Alle Speisen ausgeben.

Bemerkung

Bei Szenario BC3 und BC6 haben wir uns entschieden, dass die Beziehung zwsichen einer Instanz von Gericht und einer Instanz von Speise über /gerichte/{gericht_id}/speisen hinzugefügt oder gelöscht weren kann. Man hätte dies auch über /speisen/{speisen_id}/gerichte tun können, was wir jedoch für unübersichtlicher und nicht so naheliegend wie unsere Variante gehalten haben.

3.3 Teilaufgabe 3: Implementierung in Spring Data JPA / Web MVC