PizzaPal

Architekturdokument

David Thomann, Nha-Dan Tran, Aron Schlegel

Sommersemester 2024
16. Juni 2024

Hochschule **RheinMain**

Inhaltsverzeichnis

1 Einführung und Ziele

1.1 Aufgabenstellung

Was ist PizzaPal?

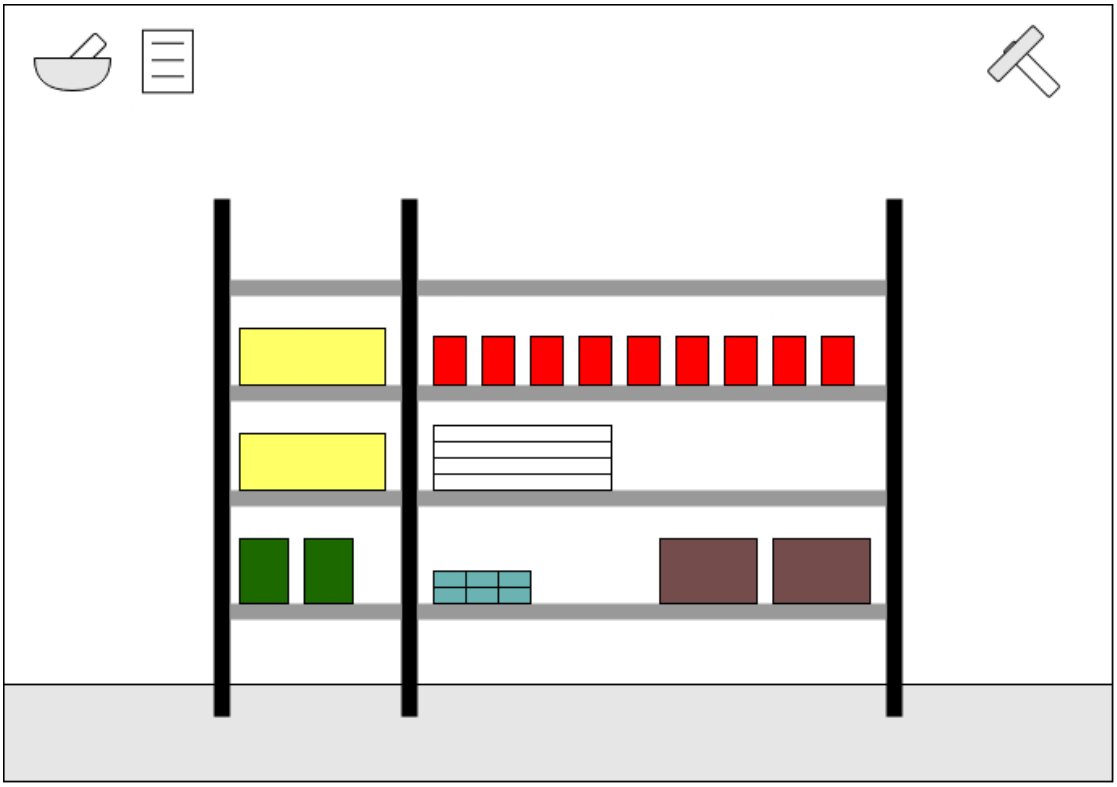
PizzaPal ist eine Software zur Unterstützung der Lagerlogistik von Pizzerien, indem der Benutzer seine Zutaten über eine grafische Oberfläche verwaltet. Das Lager wird als zweidimensionaler Raum dargestellt, in dem ein Regal mit Paketen von Zutaten steht.

Ein Paket enthält Informationen zu Tragkraft, Maßen, Gewicht, Menge der Zutat und eine Liste von Unverträglichkeiten. Die Pakete werden in einer grafischen Oberfläche als Rechtecke in einer Regalstruktur verwaltet. Der Nutzer kann die Regale anpassen und Pakete hinzufügen, entfernen oder umorganisieren. Dabei dürfen keine Bretter überlastet werden und Unverträglichkeiten müssen berücksichtigt werden. Stapel von Paketen müssen ebenfalls verwaltet und verschoben werden können.

Fehler beim Erstellen oder Umlagern von Paketen und bei der Regalkonfiguration sollen dem Nutzer grafisch angezeigt werden. Für spezifizierte Anwendungsfälle siehe Anforderungsdokument (V1.1). In der nachfolgenden Tabelle sind die Anforderung absteigend priorisiert aufgelistet.

Anforderung	**Erklärung**
Paketvorlage anlegen	Eine Vorlage für ein Paket einer bestimmten Zutat, mit dem dazugehörigen Gewicht, der Tragkraft, den Maßen des Pakets und der Menge an Inhalt anlegen
Brett- und Stützenvorlage erzeugen	Vorlagen für Bretter und Stützen anlegen
Bretter und Stützen erzeugen	Bretter- und Stützenobjekte aus Vorlagen erstellen
Regal hinzufügen	In einem leeren Lagerraum ein neues Regal bauen
Pakete erzeugen	Ein Paket aus einem Template erzeugen und platzieren
Pakete anordnen	Ein Paket im Regal per Drag&Drop an einen anderen Ort verschieben
Pakete validieren	Ein Paket wenn es platziert wird auf Verträglichkeiten mit benachbarten Paketen testen
Paketvorlage bearbeiten	Das Gewicht, Tragkraft, Menge oder Maße einer Paketvorlage ändern
Pakete löschen	Ein Paket im Regal löschen
Bestehende Bretter und Stützen anpassen	Bretter und Stützen die bereits existieren anpassen
Bestehende Bretter und Stützen bewegen	Bretter und Stützen im Raum bewegen
Regal bearbeiten	Ein bereits bestehendes Regal umbauen
Inventarliste anzeigen	Zeigt eine nach Zutaten sortierte Liste der in einem Regal vorhandenen Pakete an

Abbildung 1: Beispiel für die GUI



1.2 Qualitätsziele

Die folgende Tabelle beschreibt die zentralen Qualitätsziele von PizzaPal. Die Reihenfolge gibt eine grobe Orientierung bezüglich der Priorisierung vor. Für weitere Details siehe Rahmenbedingungen & Technische Anforderungen im Anforderungsdokument(Version 1.1).

Qualitätsziel	**Motivation und Erläuterung**
Benutzerfreundlichkeit	Das System muss eine intuitive und einfache Bedienung ermöglichen, insbesondere durch die Verwendung von Drag&Drop für die Anordnung der Regale und Pakete.
Zuverlässigkeit	Das System muss sicherstellen, dass keine Regalböden überlastet werden und keine Pakete mit Unverträglichkeiten zusammen gelagert werden.
Leistung	Das Programm soll schnell reagieren, insbesondere beim Laden und Anordnen von Paketen sowie beim Umbauen der Regale.
Skalierbarkeit	Das System muss in der Lage sein, verschiedene Größen und Mengen an Paketen und Regalen zu verwalten, ohne die Leistung zu beeinträchtigen.
Kompatibilität	Das Programm muss mindestens unter der aktuellen Kubuntu 22.04.4 LTS Linux-Version funktionieren und die technischen Anforderungen erfüllen (Java 21, 8 GB RAM, Full-HD Bildschirm, Maus).
Grafische Darstellung	Die grafische Benutzeroberfläche muss klar und übersichtlich sein, Fehler und Unverträglichkeiten müssen deutlich angezeigt werden.

1.3 Stakeholder

Stakeholder	**Interesse, Bezug**
Pizzeriapersonal	Die Hauptnutzer der Software, die die Zutaten im Lager verwalten müssen, für das Anlegen, Bearbeiten und Organisieren der Lagerregale und Pakete verantwortlich sind oder die gelegentlich das Lager verwalten oder Inventuren durchführen müssen.
Software-Entwickler	Personen, die das System entwickeln und warten.
Geschäftsleitung der Pizzeria	Interessenten an einer effizienten und fehlerfreien Lagerverwaltung zur Kostenoptimierung und Verbesserung der Betriebsabläufe.
IT-Support	Zuständig für die technische Unterstützung und Wartung der Software.

2 Randbedingungen

2.1 Technische Randbedingung

Randbedingungen	Erläuterungen bzw. Hintergrund
Hardwareaustattung	Zur Interaktion mit dem Programm ist eine Tastatur und eine Maus erforderlich.
Betrieb auf Linux Desktop Betriebssystemen	Die Software richtet sich speziell an kleine Pizzerien, die oft weniger IT-Ressourcen haben und daher von einem stabilen, sicheren und kostenfreien Betriebssystem profitieren.
Implementierung in Java	Es wird mindestens Java 21 benötigt.
GUI Implementierung mit JavaFX	JavaFX ermöglicht die Entwicklung plattformunabhängiger Anwendungen, die auf verschiedenen Betriebssystemen ohne größere Anpassungen laufen können. Dies ist besonders nützlich für zukünftige Erweiterungen auf andere Betriebssysteme.

2.2 Organisatorische Randbedingungen

Randbedingungen	Erläuterungen bzw. Hintergrund
Team	David Thomann, Aron Schlegel, Nha-Dan Tran
Vorgehensmodell	Iterativ. Die Dokumentation erfolgt unter Einsatz des arc42 Templates.
Entwicklungswerkzeuge	Draw.io und PlantUML für Diagramme. Latex für die Erzeugung des Anforderungs- und Architekturdokuments. Visual Studio Code zur Erstellung der Quelltexte. Die Software muss aber auch allein mit Gradle, also ohne IDE erzeugbar sein.
Konfigurations- und Versionsverwaltung	Rhodecode
Testwerkzeuge und -prozesse	(geplant) JUnit im Annotationsstil für Integrationtstests.
Veröffentlichung	Geplant als Open Source auf GitHub zur Verfügung zur stellen. Lizenz: GNU General Public License version 3.0 (GPLv3).

3 Kontextabgrenzung

Dieser Abschnitt beschreibt das Umfeld von PizzaPal. Für wen ist es da, und mit welchen Fremdsystemen interagiert es?

3.1 Fachlicher Kontext

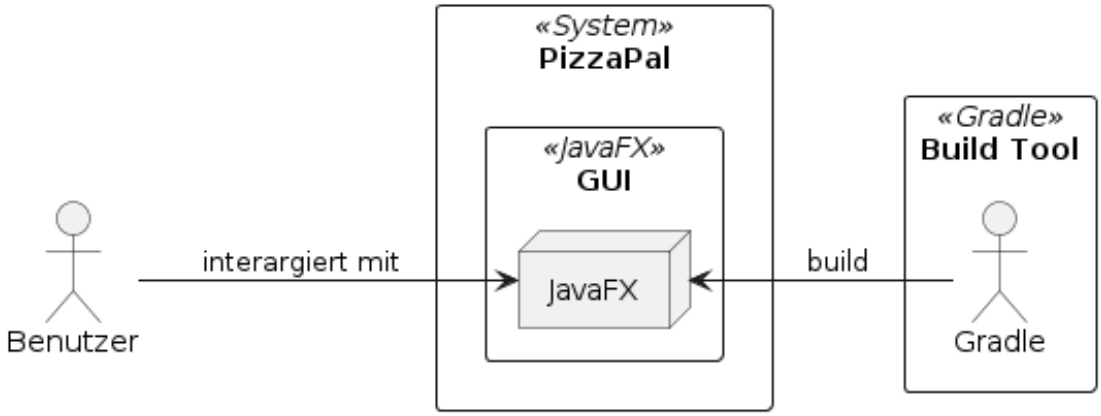


Abbildung 2: Fachlicher Kontext

Nachbar	**Beschreibung**
Benutzer	Gibt die zu sortierenden Zutaten über die GUI in das System ein. Das System überprüft die Eingabedaten auf Validität und zeigt die Ergebnisse auf der GUI entsprechend an.
Build Tool (Gradle)	Wird verwendet um die JavaFX-Anwendung zu bauen.

3.2 Technischer Kontext

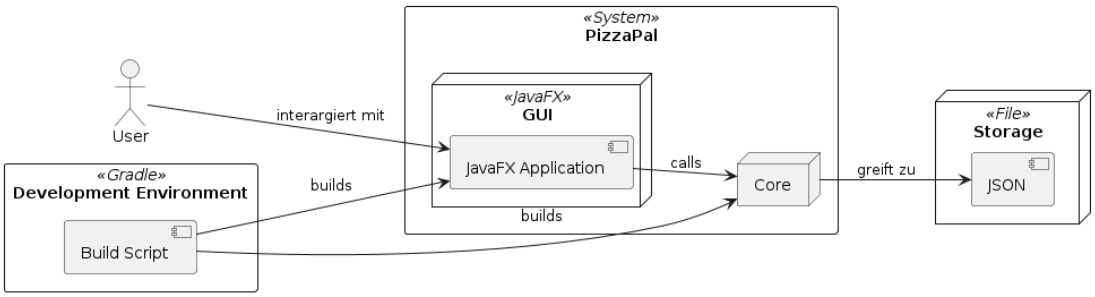


Abbildung 3: Technischer Kontext

Nachbar	**Beschreibung**
Benutzer	Ein Akteur, der mit dem System interargiert.
Developer Environment(Gradle)	Entwicklungsrechner, der das Build-Script für das Gradle Plugin stellt. Wird verwendet um die Anwendung zu bauen.
JSON File	Eine JSON-Datei, die als Speichermedium für Daten dient.
Core	Enthält die Geschäftslogik des Systems.

4 Bausteinsicht

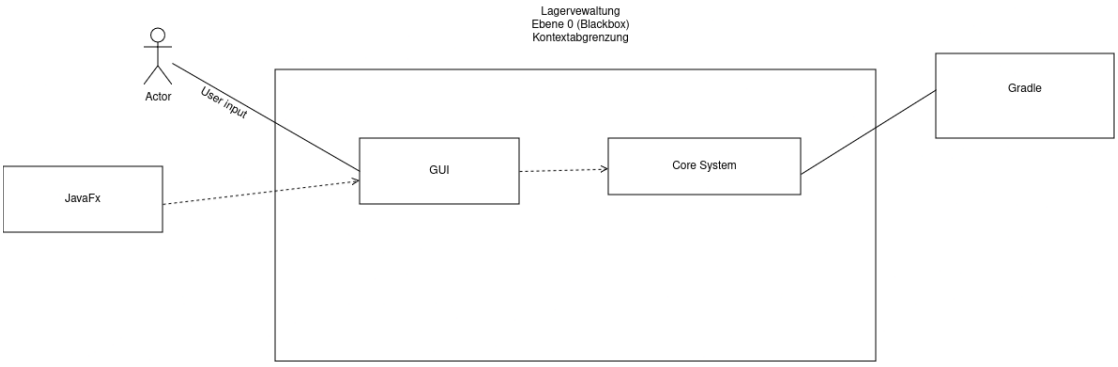


Abbildung 4: Systemübersicht

1. Begründung: Das gesamte Kernsystem mitsamt Datenhaltung, Validation und Datenmanipulation sollte dem Nutzer nicht direkt, sondern über eine grafische Benutzer-Oberfläche angeboten werden.
2. Enthaltene Bausteine: Die Präsentation der Daten in der Blackbox GUI, sowie das Kernsystem (Model, Service und Valiation) in der Blackbox Core System
3. Wichtige Schnittstellen: Die Präsentationsschicht wird mithilfe von JavaFX umgesetzt. Als Build-Tool wird Gradle genutzt

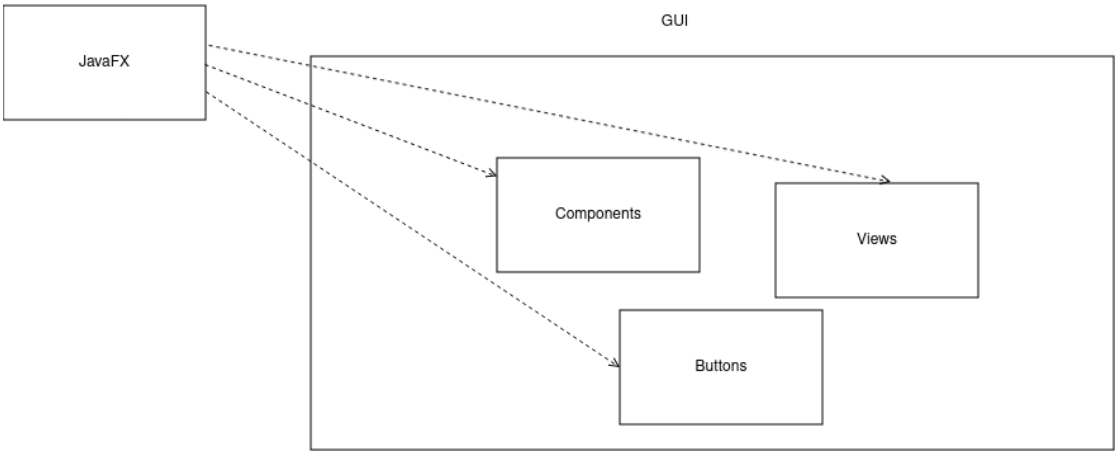


Abbildung 5: GUI

1. Begründung: Die GUI wurde unterteilt in Layout und Interaktionselemente

2. Enthaltene Bausteine: Die Blackbox Views enthält Layouts von JavaFX-Elementen. In diesem Fall die Hauptansicht des Programms. Die Blackbox Buttons enthält diejenigen Interaktionselemente die diese Hauptansicht funktional beeinflussen. Die Components-Blackbox beinhaltet JavaFX-Elemente, die das Modell darstellen.

3. Wichtige Schnittstellen: Alle Blackboxen erben von JavaFX-Klassen

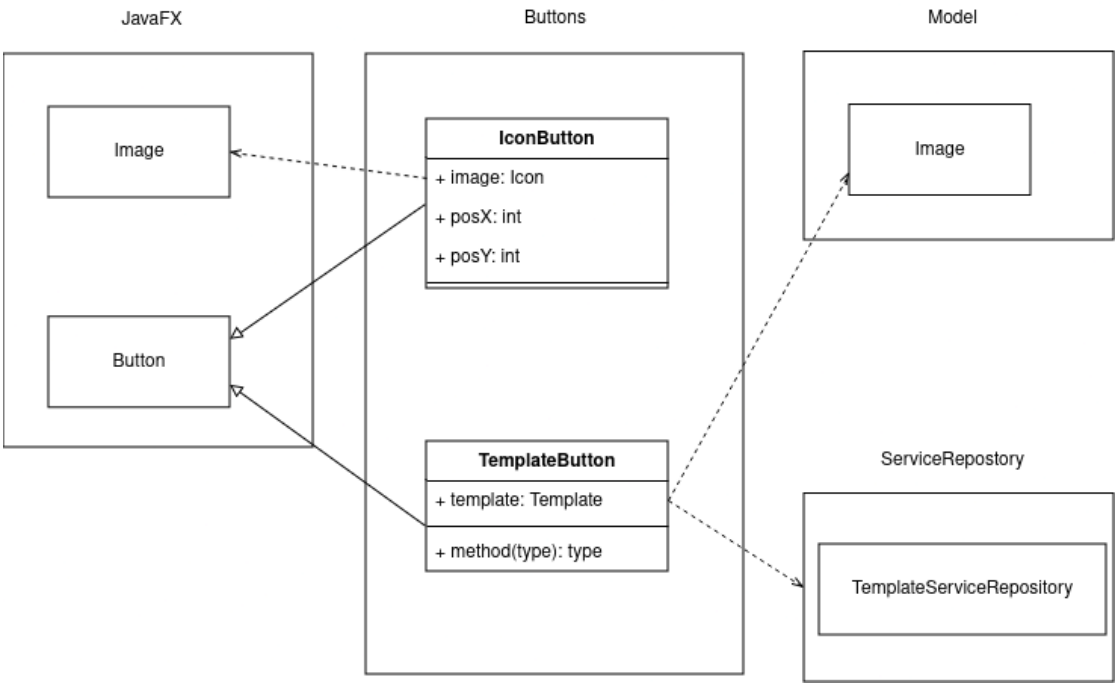


Abbildung 6: Buttons

1. Begründung: Die Buttons wurden in Buttons IconButtons, zum Umschalten der Funktionalität und TemplateButtons zum Erzeugen neuer Objekte unterteilt.

2. Wichtige Schnittstellen: Alle Buttons erben von der JavaFX-Klasse Button

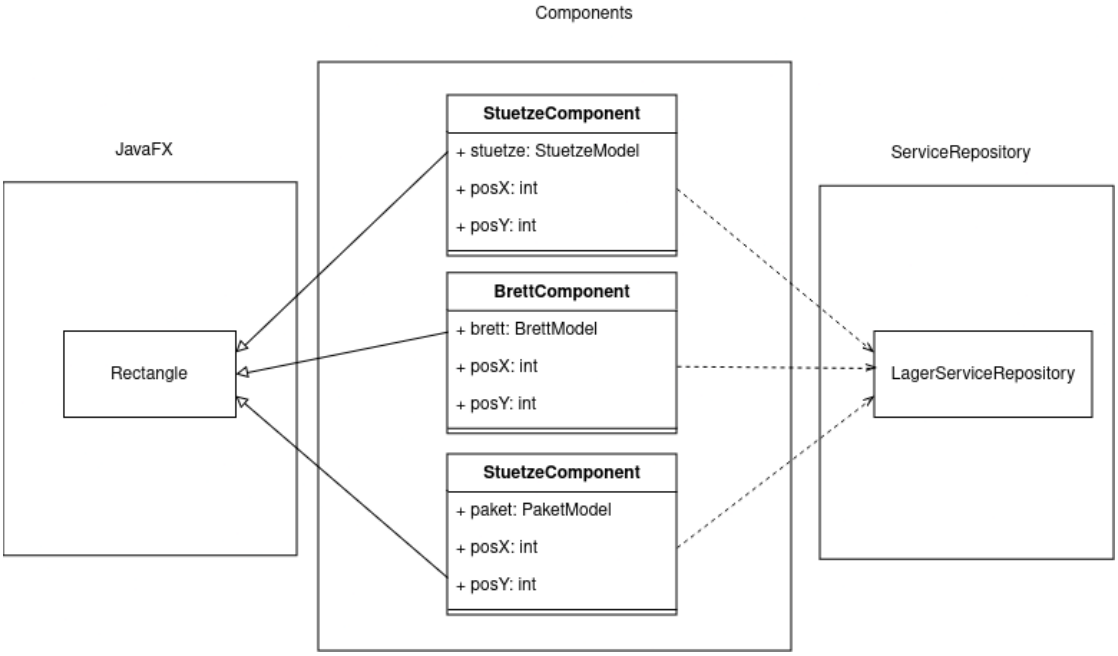


Abbildung 7: Components

1. Begründung: Die Components stellen die dem Modell zugrundeliegenden Objekte dar. Sie geben Aktionen des Nutzers an die Service Schicht weiter und hören auf ihr Modell

2. Wichtige Schnittstellen: Alle Komponenten erben von der JavaFX-Klasse Rectangle und kommunizieren mit dem

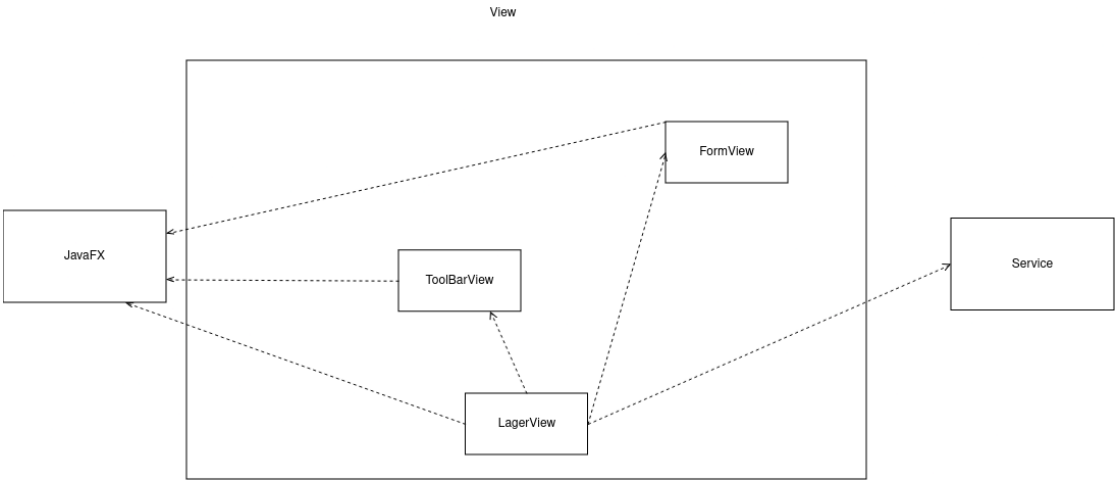


Abbildung 8: Views

1. Begründung: Die Views sind alle Layouts, die auf die Scene/Stage gelegt werden. Die betten Components und Buttons ein. Views werden geschachtelt aufgebaut

2. Wichtige Schnittstellen: JavaFX-Regions Package, Repositories zum Anzeigen des aktuellen Lagerzustands

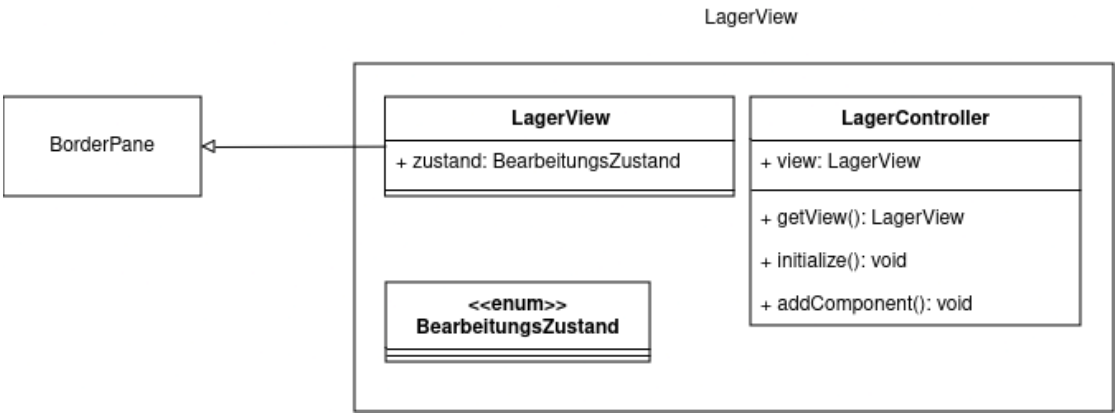


Abbildung 9: LagerView

1. Begründung: Da kein Menü geplant ist, ist der LagerView zentral. Er setzt sich aus den Komponenten und der ToolBar zusammen

2. Wichtige Schnittstellen: LagerService um über neue Entities informiert zu werden

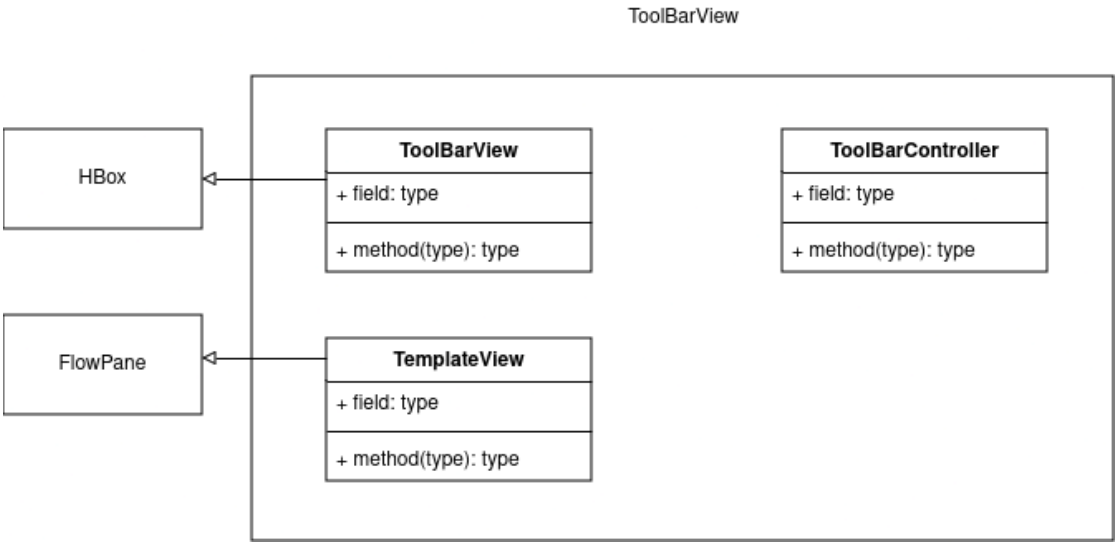


Abbildung 10: ToolBarView

1. Begründung: Die ToolBar bietet alle Buttons an. Sie schaltet zwischen den Modi des Systems um.
2. Wichtige Schnittstellen: LagerView

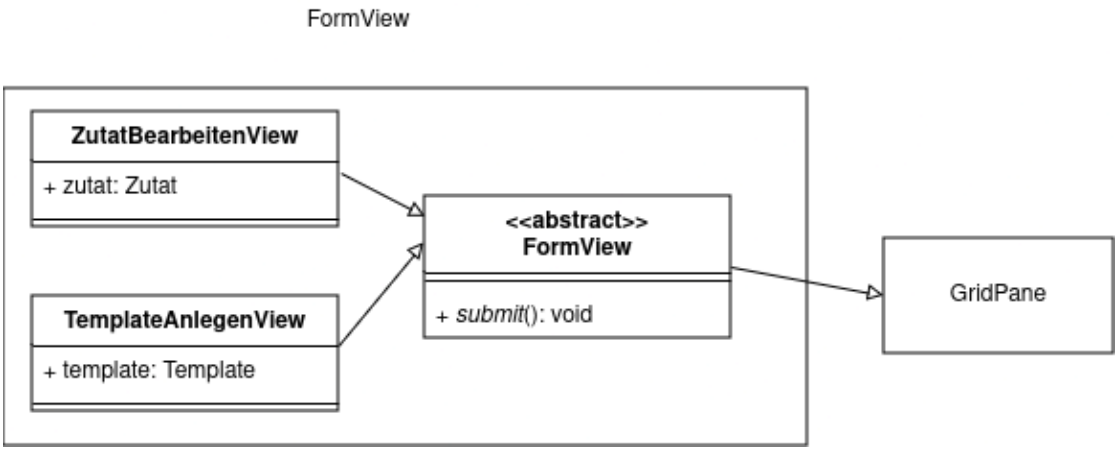


Abbildung 11: FormView

1. Begründung: Die FormView ist gedacht zum Anlegen neuer Zutaten und Templates. Sie spricht direkt mit den jeweiligen Service-Klassen um ihr eigenes Modell anlegen zu lassen

2. Wichtige Schnittstellen: ZutatService und TemplateService

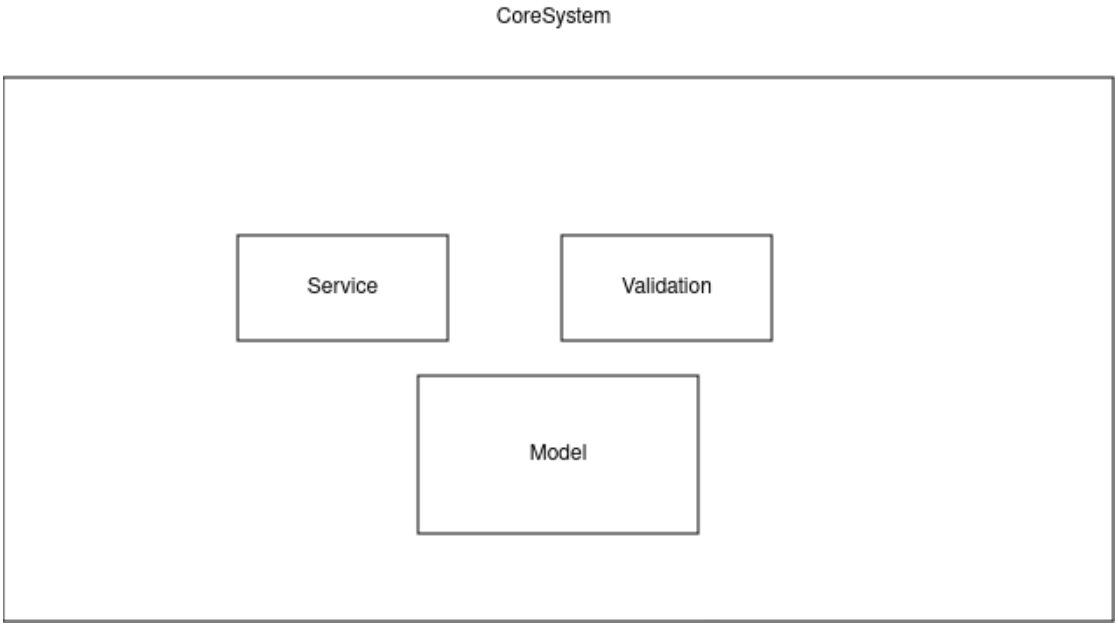


Abbildung 12: Core

1. Begründung: Das Core-System umfasst die Modellwelt und die sie ändernde (ServiceRepository) und überprüfende Schicht (Validator)
2. Wichtige Schnittstellen: Service-Klassen als Schnittstelle zur GUI, Modell-Struktur als Grundlage der Datenhaltung

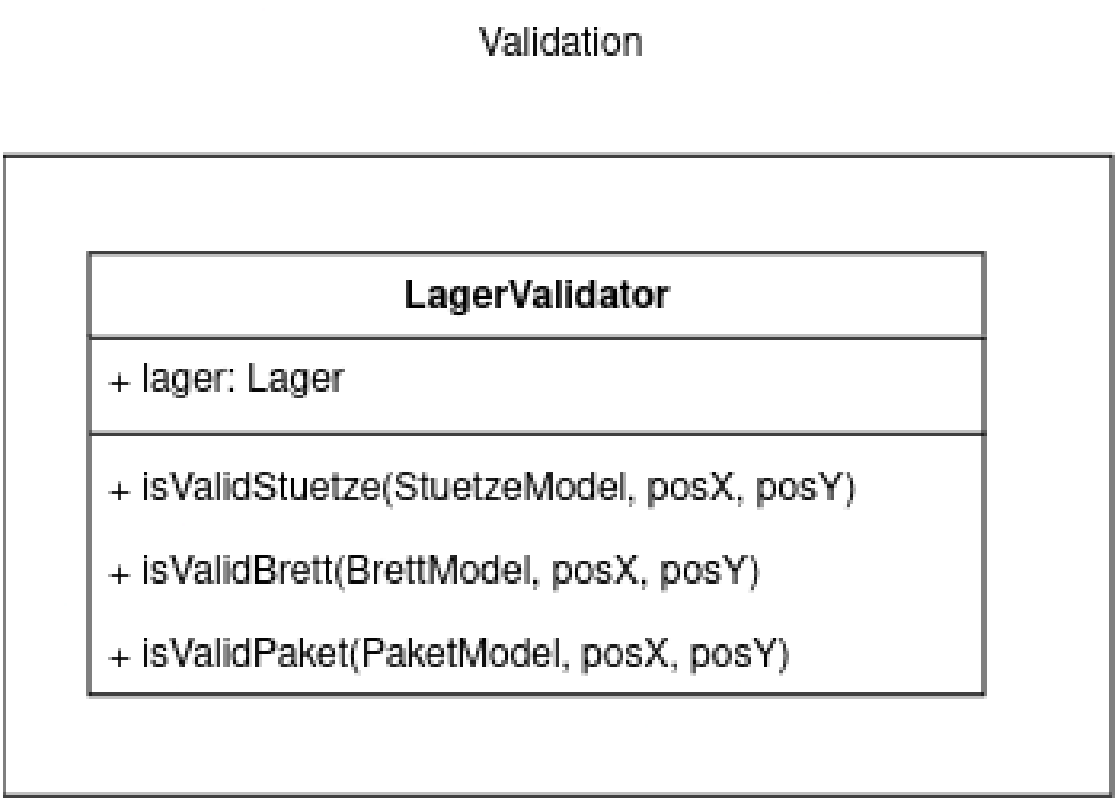


Abbildung 13: Validation

1. Begründung: Validierung sollte optional von der Modellschicht sein, damit man sie freier nutzen kann. Validierung ist in dieser Anwendung zwingend erforderlich um valide Aktionen des Nutzers zu bestimmen
2. Wichtige Schnittstellen: ServiceRepository um auf aktuellen Zustand des Lagers zugreifen zu können

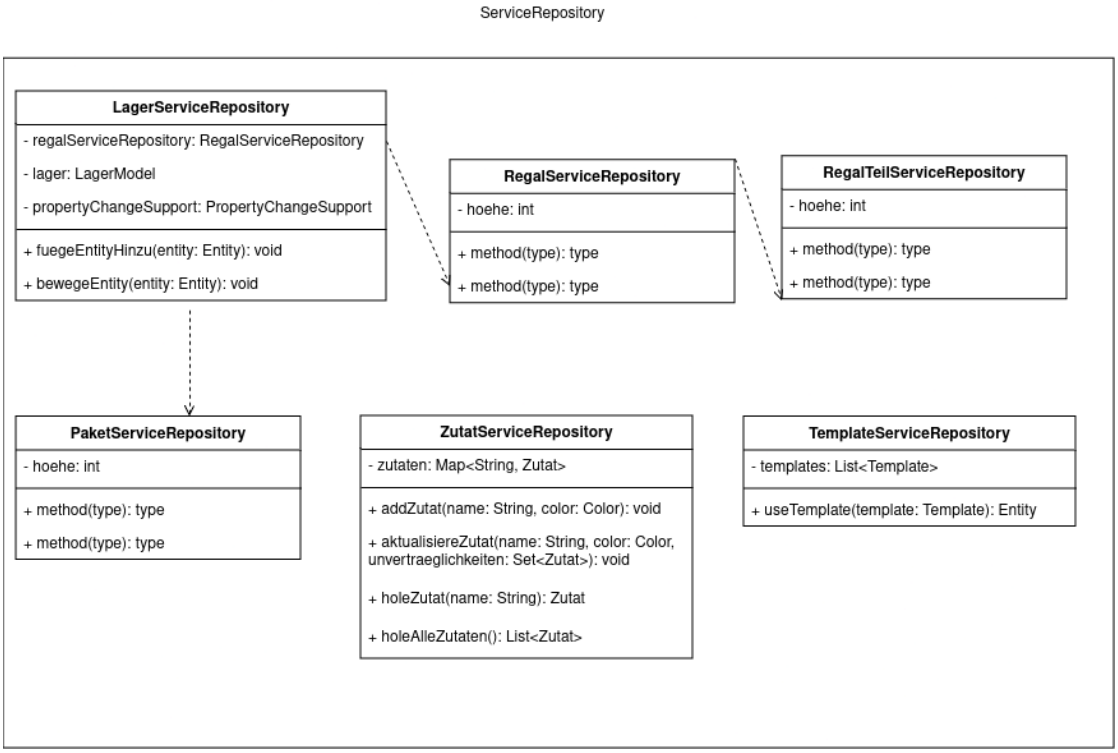


Abbildung 14: ServiceRepository

1. Begründung: Das ServiceRepository ist die Haltung der Daten und ihrer Manipulation zuständig. Ein separates Repository ist aufgrund der gerinen Dateimenge nicht notwendig, die Objekte werden in Dateien gespeichert.

2. Wichtige Schnittstellen: GUI, um Aktionen des Nutzers zu empfangen

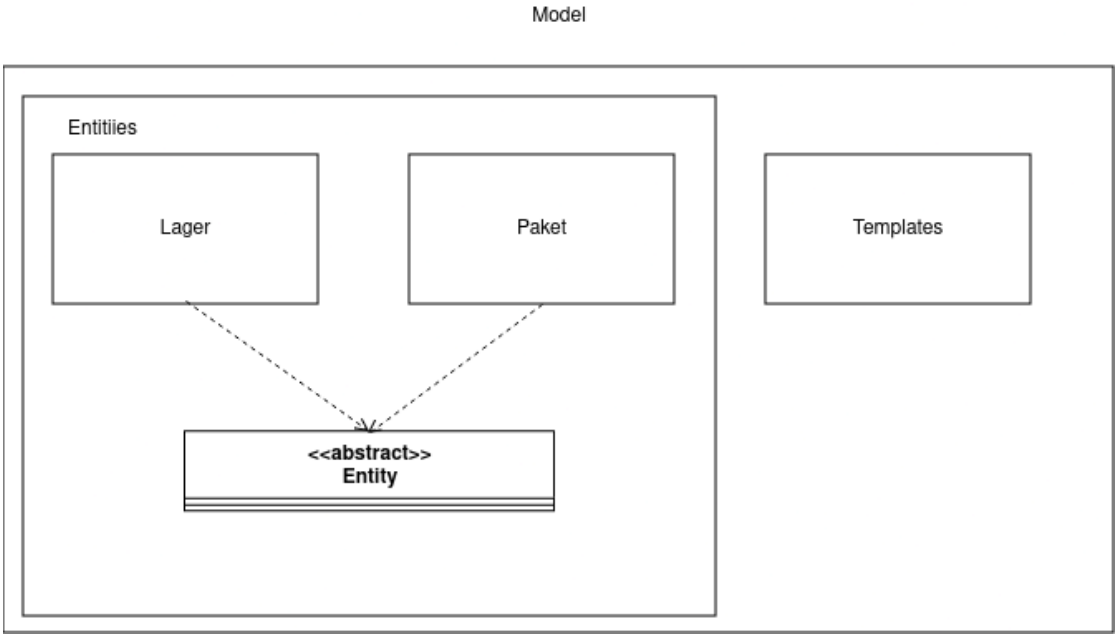


Abbildung 15: Model

1. Begründung: Das Modell ist in zwei Teile aufgeteilt. Templates sind wiederverwendbar, aber nicht notwendig um das Modell zu nutzen. Entities können standardisiert erstellt werden und zusammen verwaltet werden.

2. Wichtige Schnittstellen: Entity-Package ist zentral

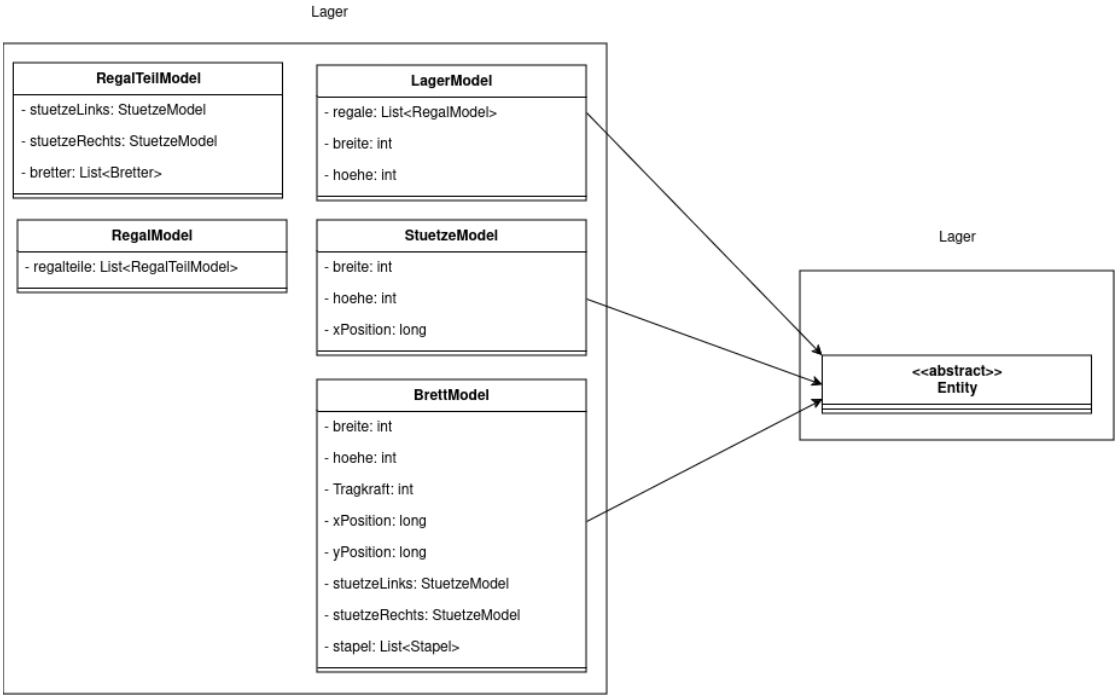


Abbildung 16: Lager

1. Begründung: Die Teile des Lagers, die in der GUI auftauchen erben von Entity. Separate Klassen, die nur im internen Aufbau des Lagers wichtig sind, erben nicht davon und können dementsprechend nicht über Templates erstellt werden

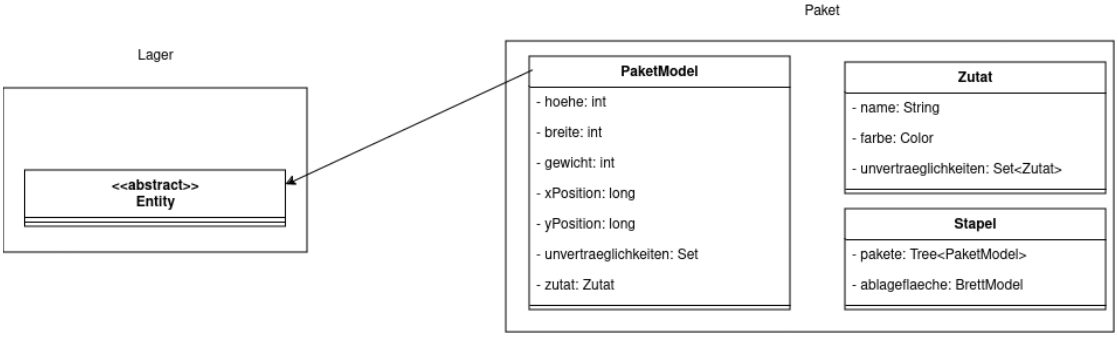


Abbildung 17: Paket

1. Begründung: Das PaketModel ist wieder ein Entity, die Zutaten werden separat verwaltet.

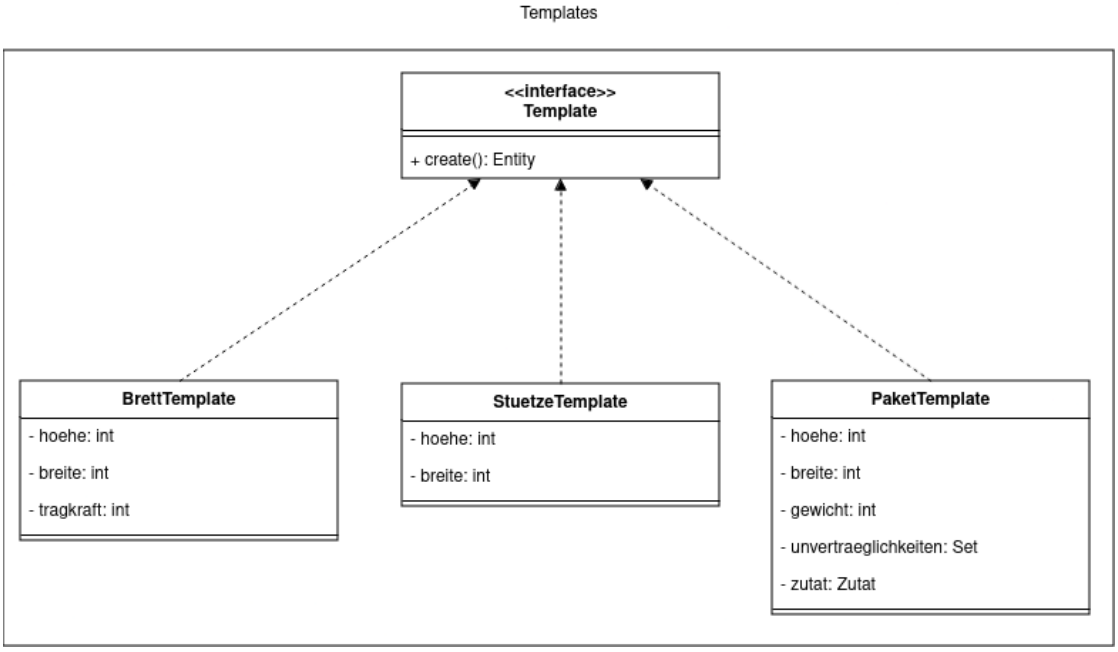


Abbildung 18: Template

1. Begründung: Templates werden separat von den Entities gehalten um über die GUI einfach dargestellt zu werden.

2. Wichtige Schnittstellen: TemplateButtons

5 Laufzeitsicht

5.1 Templates anlegen

Author: Nha-Dan Tran
Um Templates für verschiedene Objekte zu generieren, verwendet die JavaFX-Anwendung ein Factory Pattern. Der Ablauf beginnt, wenn der Benutzer auf einen speziellen Button klickt. Das Ereignis wird von der JavaFX-Anwendung verarbeitet, indem sie einen ButtonHandler aufruft. Der ButtonHandler ist für die Handhabung von Benutzeraktionen zuständig und initiiert die Erstellung von Templates durch die Factory.

Die Factory wird angewiesen, ein Brett-Template zu erstellen. Dazu ruft der ButtonHandler die entsprechende Methode in der Factory auf (createTemplate("Brett")). Die Factory instantiiert daraufhin ein Brett-Template und gibt diese Instanz zurück. Dieselbe Prozedur wird für Paket- und Stütze-Templates wiederholt. Der ButtonHandler koordiniert die Erstellung der Templates für jedes spezifizierte Objekt durch Aufrufen der entsprechenden Methoden in der Factory.

Das Factory Pattern ermöglicht es der Anwendung, flexibel und dynamisch Templates zu erstellen, ohne an die konkreten Klassen der Templates gebunden zu sein. Dies fördert die Wartbarkeit und Erweiterbarkeit des Systems, da neue Templates einfach hinzugefügt werden können, ohne die bestehende Logik zu ändern.

Dieser Prozess gewährleistet, dass die JavaFX-Anwendung effizient und klar strukturiert bleibt, indem sie die Erstellung von Templates auf eine abstrahierte Factory-Schnittstelle delegiert, die für die Auswahl und Erzeugung der richtigen Templates verantwortlich ist.

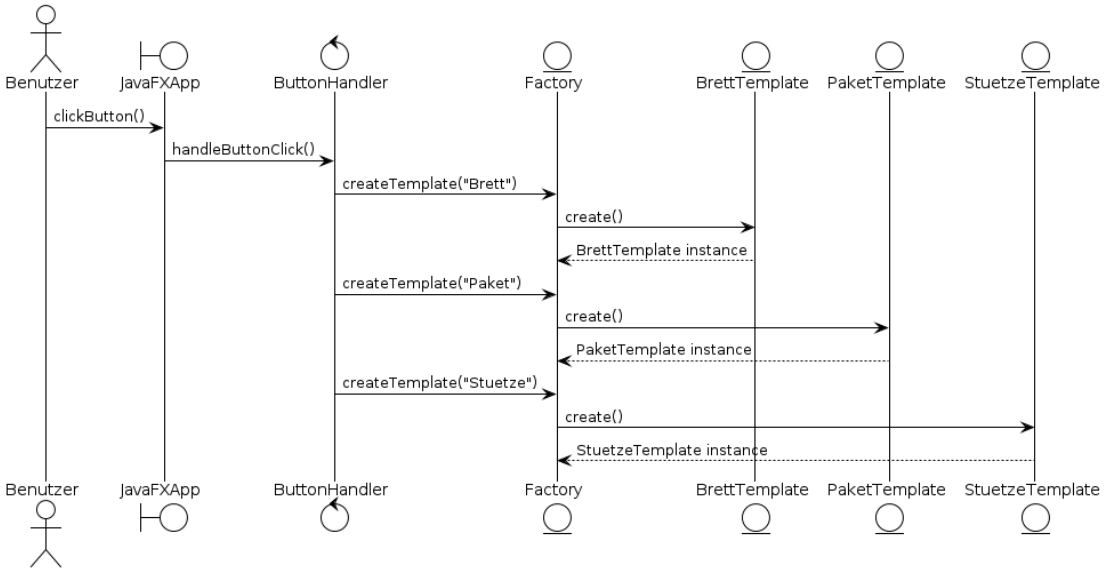


Abbildung 19: Fachlicher Kontext

5.2 Pakete verschieben

Author: Aron-Merlin Schlegel
Das Sequenzdiagramm zeigt den detaillierten Ablauf, der ausgelöst wird, wenn ein Benutzer ein Paket innerhalb der GUI verschiebt und loslässt. Diese Aktion wird durch den Benutzer initiiert und durchläuft mehrere Schritte, bis die neue Position des Pakets validiert und gegebenenfalls die GUI aktualisiert wird.

Der Prozess beginnt damit, dass der Benutzer das Paket mit der Maus auswählt und verschiebt (‘onMousePressed‘ und ‘onMouseDragged‘). Diese Ereignisse werden von der ‘PaketComponent‘ erfasst und an den ‘RegalTeilController‘ weitergeleitet. Der ‘RegalTeilController‘ speichert die Anfangsposition des Pakets, um bei Bedarf später darauf zurückgreifen zu können. Während der Drag-Phase wird das visuelle Feedback durch Translation des Pakets angepasst.

Beim Loslassen der Maus (‘onMouseReleased‘) initiiert der ‘RegalTeilController‘ die Validierung der neuen Position des Pakets. Diese Validierung wird durch den ‘ValidationService‘ durchgeführt, der die Methode ‘checkPlacement‘ aufruft. Hierbei wird geprüft, ob die neue Position des Pakets mit anderen Objekten kollidiert oder toleriert werden kann. Falls eine Kollision erkannt wird, wird die Position des Pakets zurückgesetzt und der ‘ErrorHandler‘ wird aufgerufen, um eine entsprechende Fehlermeldung in der ‘FehlermeldungView‘ anzuzeigen.

Wenn keine Kollision oder Intoleranz festgestellt wird, aktualisiert der ‘RegalTeilController‘ die Position im ‘PaketModel‘ mit den Methoden ‘setXPos‘ und ‘setYPos‘. Das ‘PaketModel‘ informiert dann alle registrierten Observer, wie die ‘RegalTeilView‘, über die neuen Positionen des Pakets, wodurch die Anzeige automatisch aktualisiert wird.

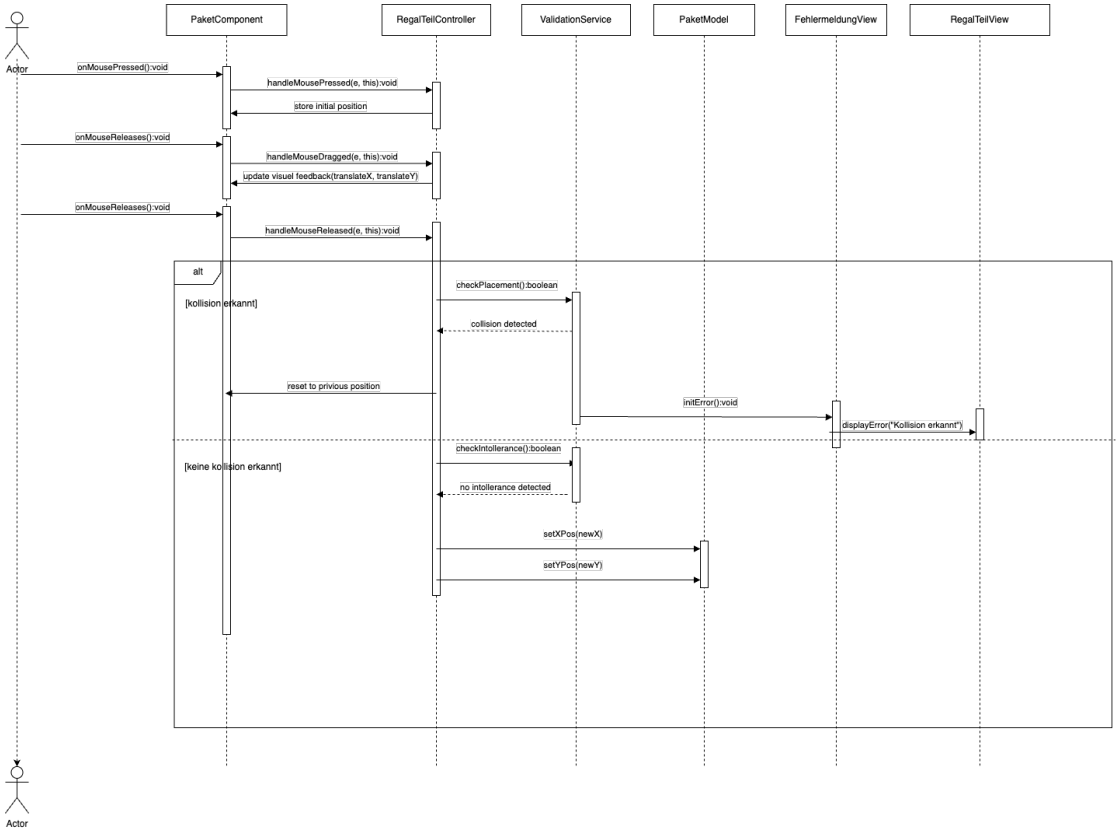


Abbildung 20: Fachlicher Kontext

5.3 Sequenz 1: Stütze erstellen

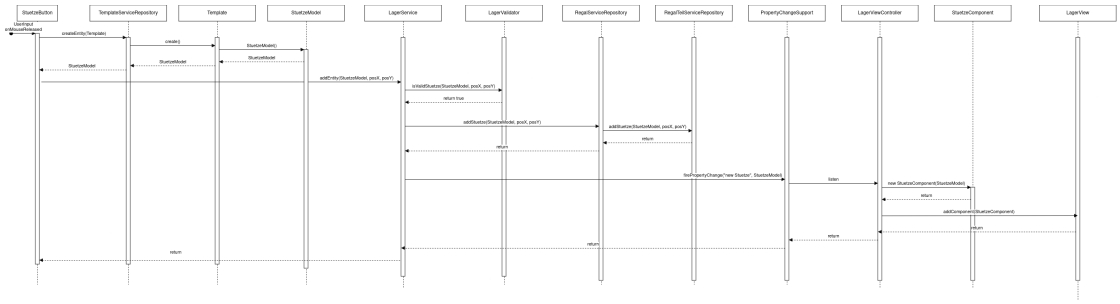


Abbildung 21: Sequenzdiagramm: Stütze erstellen

Durch das MausReleased Event ruft der TemplateButton zuerst den TemplateService auf, wodurch ein StuetzeModel erzeugt wird. Der Button reicht dieses Model, zusam-

men mit den Koordinaten des Mausevents dann an den LagerService weiter, um die Platzierung anzustoßen. Der LagerService lasst den Validator prüfen, ob die Stütze an dieser Position platziert werden kann - in diesem Fall ist das möglich. Danach baut der LagerService die Stütze ins Modell ein (mit den jeweiligen Subservices) und feuert eine Änderungen an den PropertyChangeSupport. Dieser informiert seine Listener, in diesem Fall der LagerViewController, welcher eine neue StuetzeComponent erzeugt und in den Graphen des LagerView einhängt.