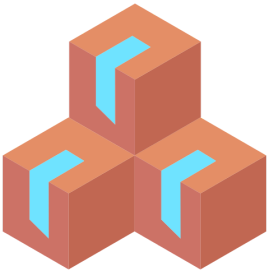
Hochschule **RheinMain**



WAREHOUSE MASTERS

Software zur Logistik-Optimierung
und -Planung

Design-Dokument

14. Juni 2021 — Version 1.0

Marc Bachmann
Florian Bohn
Nina Khalil
Thuy Trang Lena Ngo

Inhaltsverzeichnis

Version und Änderungsgeschichte

Version	Datum	Änderungen
1.0	14.07.2021	erster vollständiger Entwurf

1. Einführung und Ziele

Dieser Abschnitt beschreibt die Aufgabenstellung und Ziele, die das Projekt “Warehouse Masters” verfolgt.

1.1 Aufgabenstellung

Was ist “Warehouse Masters”?

- “Warehouse Masters” ist eine Software, die dem Anwender ermöglicht ein Regal zu erstellen und darin Pakete abzusetzen, zu verschieben, zu löschen und aufeinander zu stapeln
- Ziel: benutzerfreundliches, selbsterklärendes Programm für alle Mitarbeiter des Logistikunternehmens zur Paketeinräumung
- genaueres siehe Anforderungsspezifikationen (Version 1.6) unter Aufgabenstellung

1.2 Qualitätsziele

Qualitätsziel	Motivation, Erläuterung
einladende Experimentier-plattform	Anwender hat die Möglichkeit, beliebig viele Konfigurationen auszuprobieren und die in seinen Augen idealste Konfiguration zu speichern/umzusetzen
selbsterklärende Anwendung	Die Knöpfe müssen möglichst selbsterklärend sein, sodass jeder Anwender auf Anhieb weiß, welche Buttons für welche Funktionen (Paket hinzufügen, Paket löschen usw.) zuständig sind, sodass jeder Mitarbeiter in der Lage ist sich schnell zurecht zu finden. Dies trägt zum effizienten Arbeiten bei.
Oberfläche ist minimalistisch gehalten	Die Benutzungsoberfläche muss übersichtlich sein. Nur das Wichtigste ist auf der Hauptseite zu sehen, damit jeder Anwender erkennt wo er die jeweilige Funktion findet. Für detaillierte Einstellungen wird ein weiteres kleineres Fenster geöffnet, die diese Konfiguration ermöglichst und gleichzeitig die aktuell irrelevantes Einstellungen ausblendet.

User soll auf Fehler hingewiesen werden	Das Regal sowie die Pakete haben bestimmte Eigenschaften, die das Paket Stapeln zu einem bestimmten Grad einschränken, z.B. aufgrund der Höhe oder Farbkompatibilität. Dies muss das Programm kontrollieren und den User auf Fehler hinweisen. Andernfalls kann diese Konfiguration nicht abgespeichert werden.

1.3 Stakeholder

Wer?	**Interesse, Bezug**
Entwickler	• entwickeln der Anwendung • warten der Anwendung
Lagerist	• verwenden die Anwendung um schneller Aufträge auszuführen • weniger körperliche Arbeit aufgebracht
Logistikunternehmen Lagermeister?	• Arbeitgeber vom Lagerist • Verwendung für effizientes Einlagern von Paketen • Zeitersparnis und effizientes Arbeiten für mehr Profit

2. Randbedingungen

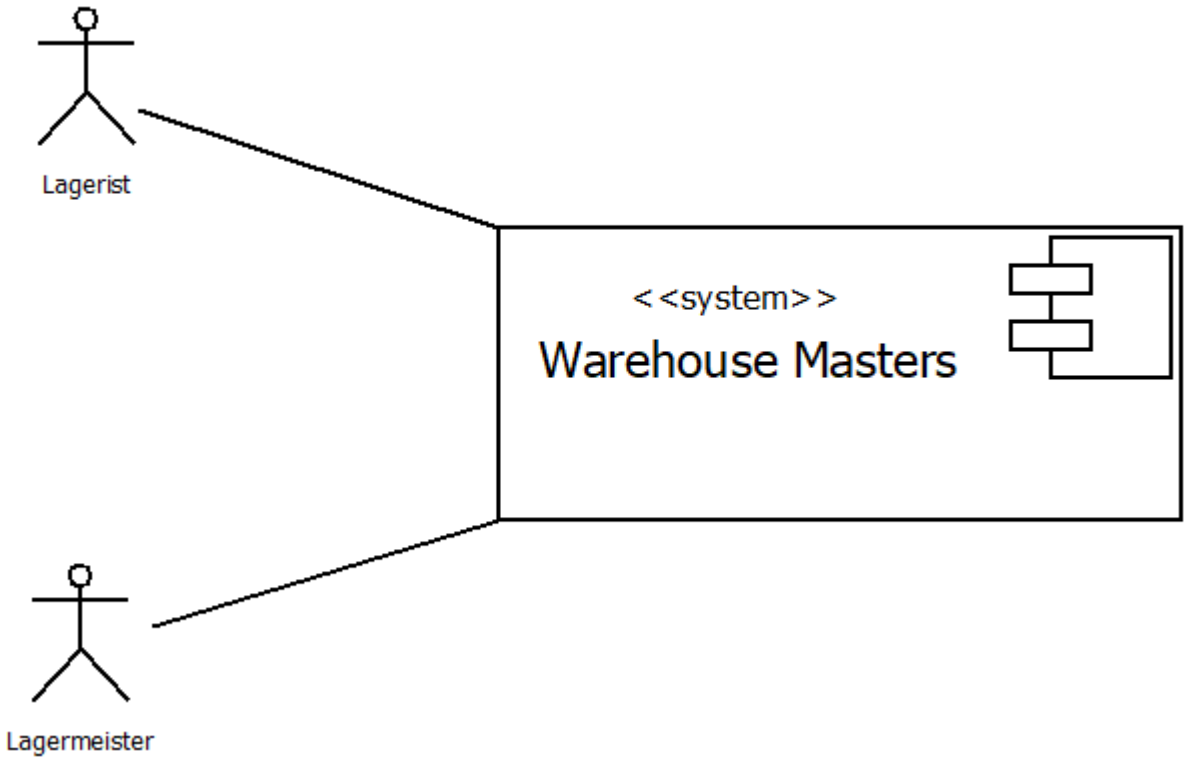
2.1 Technische Randbedingungen

Randbedingung	Erläuterung
JavaFX	Die grafische Oberfläche der Anwendung wird mit JavaFX erstellt
Implementierung in Java	Zum Einsatz kommt die Programmiersprache Java.
Betrieb auf Linux	Es muss sichergestellt werden, dass die Anwendung auf dem Betriebssystem Linux funktioniert.

2.2 Organisatorische Randbedingungen

Randbedingung	Erläuterung
Team	Marc Bachmann, Florian Bohn, Nina Khalil, Lena Ngo
Zeitplan	Beginn der Entwicklung: Ende April 2021 Fertigstellung bis 04. Juli 2021
Vorgehensmodell	für Architektur-Dokumentation: arc42

3. Kontextabgrenzung



4. Lösungsstrategie

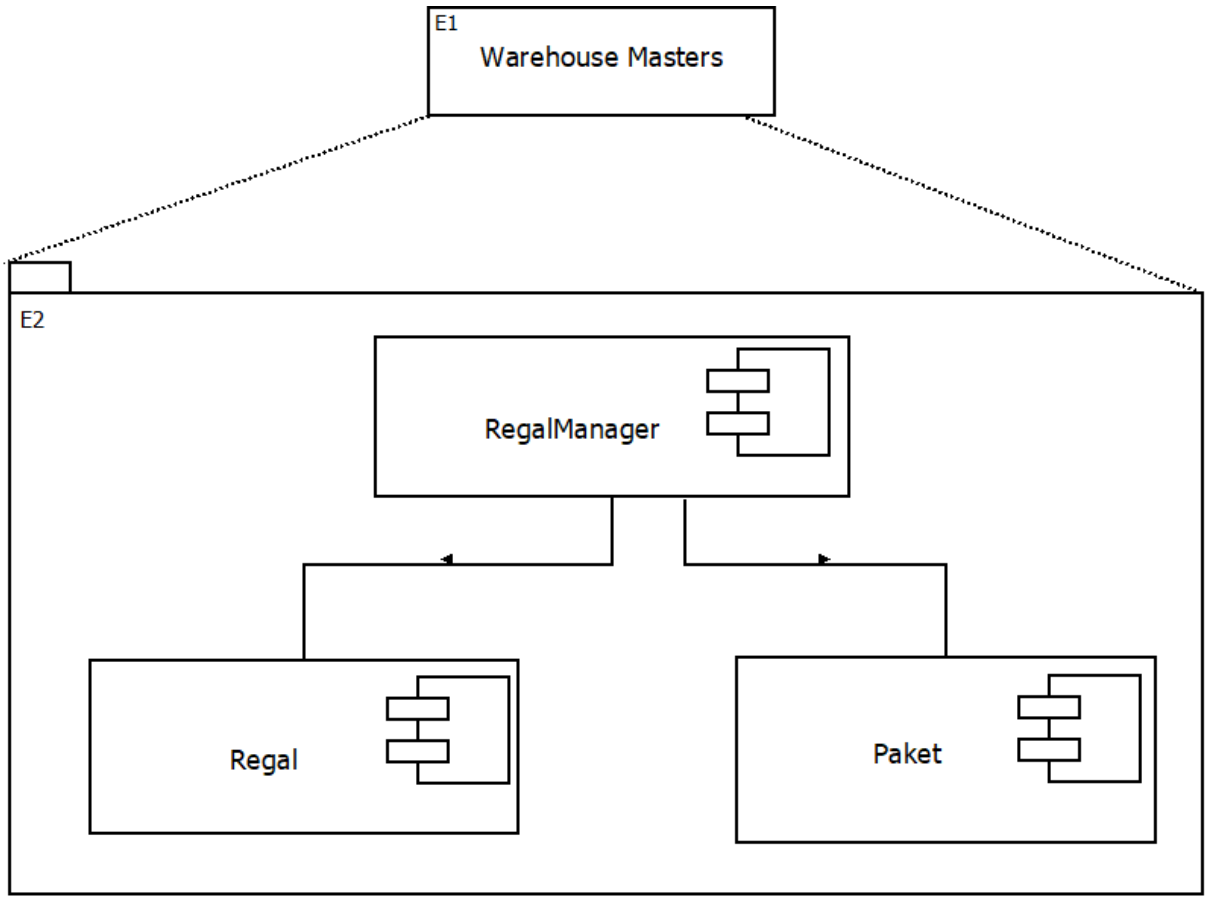
Das Projekt “Warehouse Masters” ist eine Stand-Alone Java-Applikation. Da es ein kleines Projekt ist, reicht dies vollkommen für die Umsetzung der Aufgaben aus.

Für die Verknüpfung zwischen dem Regal, also das Gerüst bestehend aus Stützen und den einzelnen Böden, und den Paketen ist ein sogenannter Regal-Manager (ShelfManager) zuständig. Dieser hat Zugriff auf das Regal und die Pakete, die erstellt wurden als auch die Zuständigkeit für die Überprüfungen bezüglich Überstand, Überlast und Kompatibilität und das Speichern/Laden von Lagerkonfigurationen.

Das Speichern einer Lagerkonfiguration sowie einer Paketvorlage erfolgt mit Hilfe von JSON-Dateien. Dieses Vorgehen erfolgt durch einen JSON-Parser, der die erstellten JSON-Dateien lokal auf dem Rechner speichert. Eine JSON-Datei hat gegenüber einer einfachen Textdatei den Vorteil, dass die Struktur eines Objektes richtig geschachtelt und so übersichtlich abgespeichert werden können, während eine Textdatei schnell unübersichtlich werden kann.

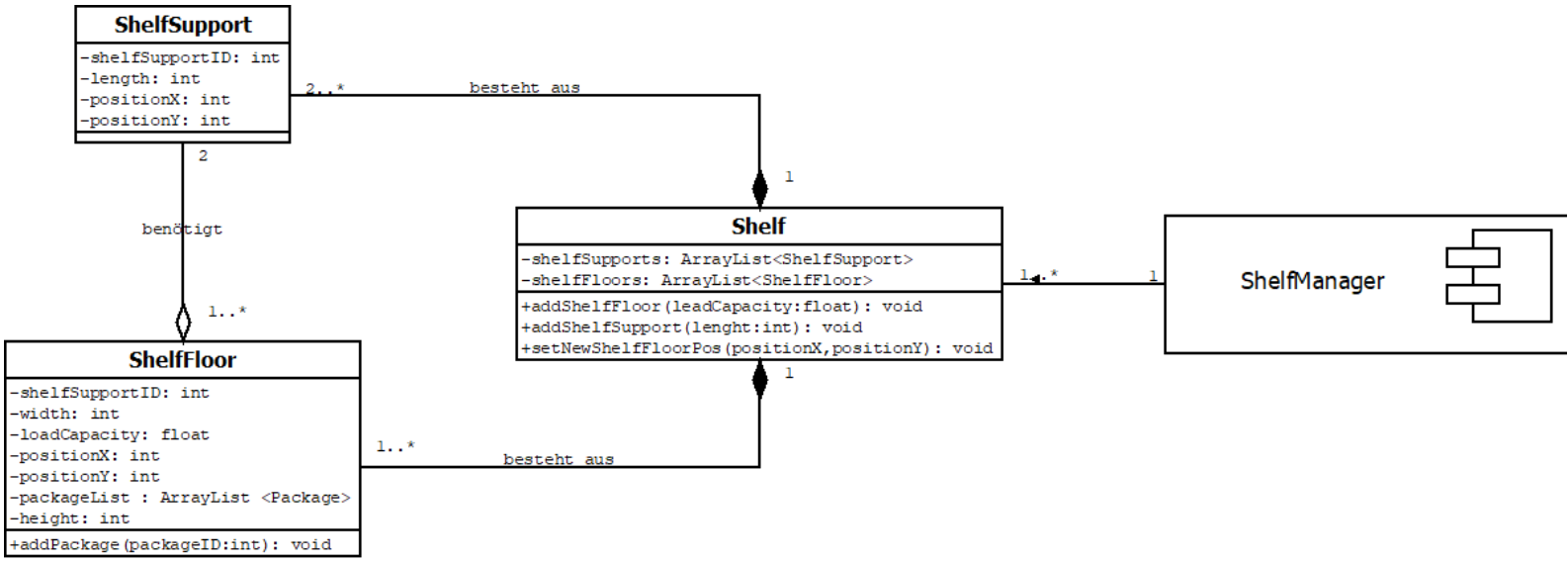
5. Bausteinsicht

Dieser Abschnitt beschreibt die die einzelnen Bausteine von Warehouse Masters. Die Anwendung wird in mehrere Subsysteme zerlegt. In den unten aufgeführten 3 Ebenen lässt sich erkennen wie die Subsysteme zerlegt werden. Unsere Anwendung, die den Namen Warehouse Masters trägt lässt sich unterteilen in ShelfManager, Shelf und Package.



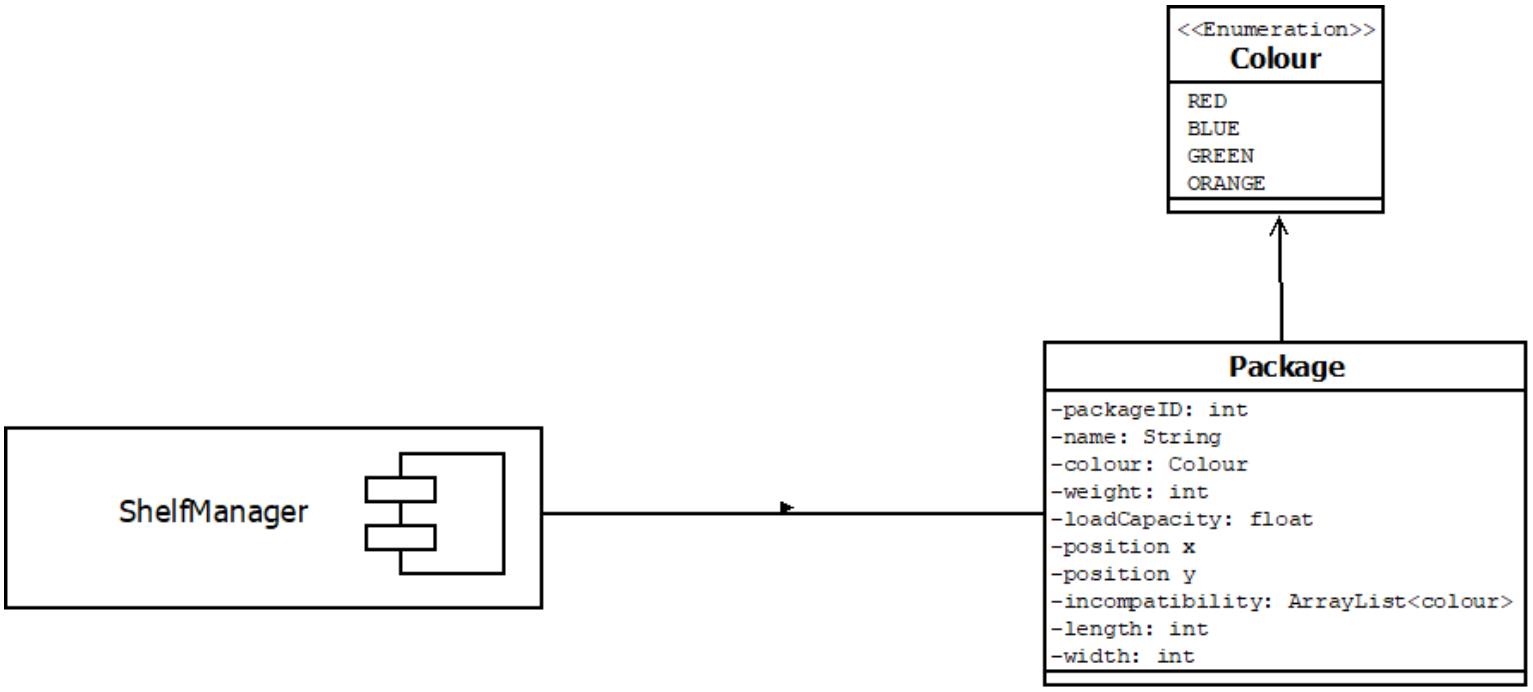
5.1 Subsystem E3: Shelf

Das erste Subsystem ist das “Shelf” mit den Klassen ShelfSupport, ShelfFloor und Shelf. Dieses Subsystem dient dazu, das Regal logisch darzustellen. Das benachbarte Subsystem ist der ShelfManager



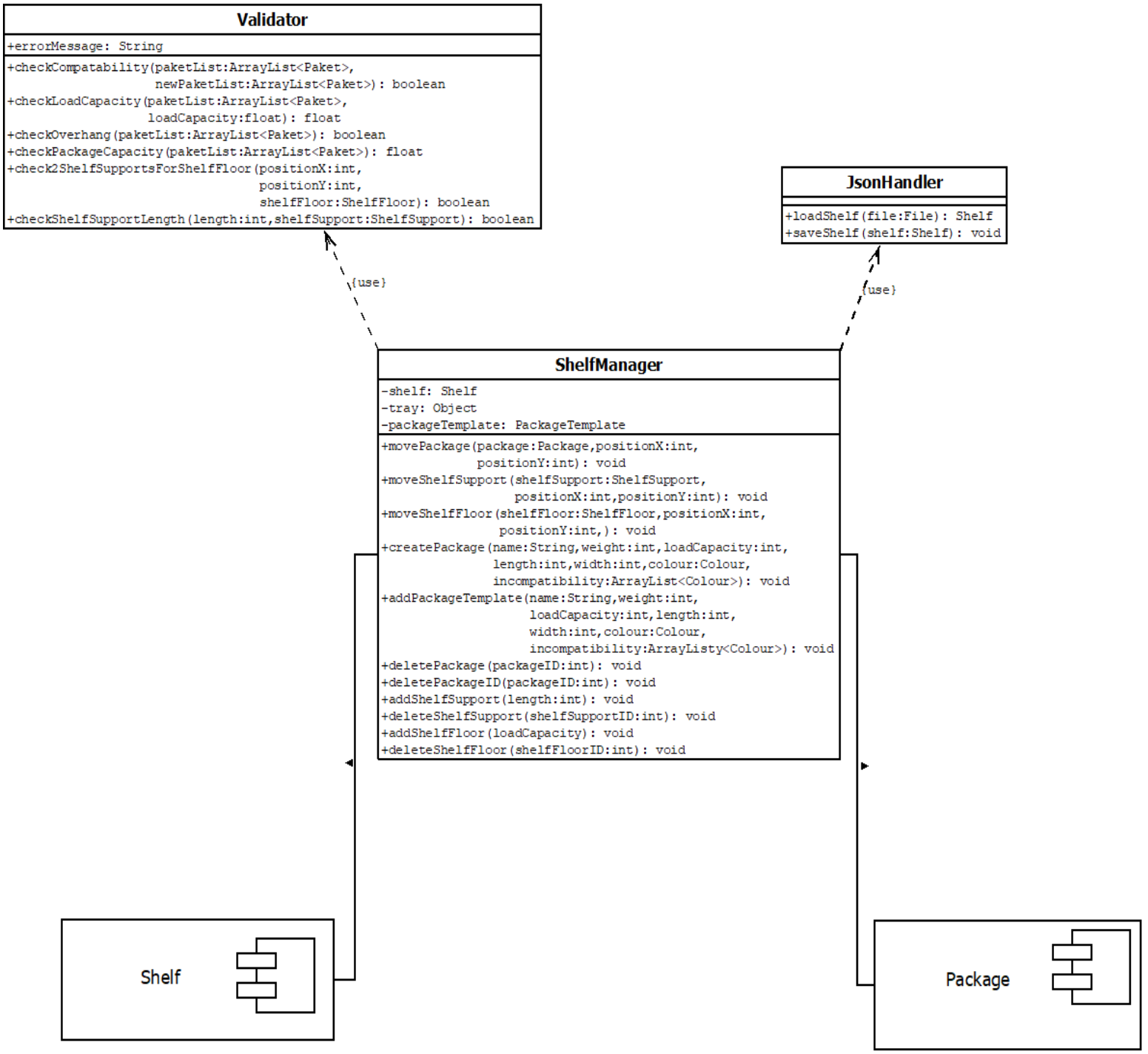
5.2 Subsystem E3: Package

Das zweite Subsystem ist das System “Package” mit den Klassen “Package” und einer Enumeration “Colour”. Das benachbarte Subystem ist ebenfalls der ShelfManager. Das “Package” Subsystem repräsentiert ein Paket der Anwendung.

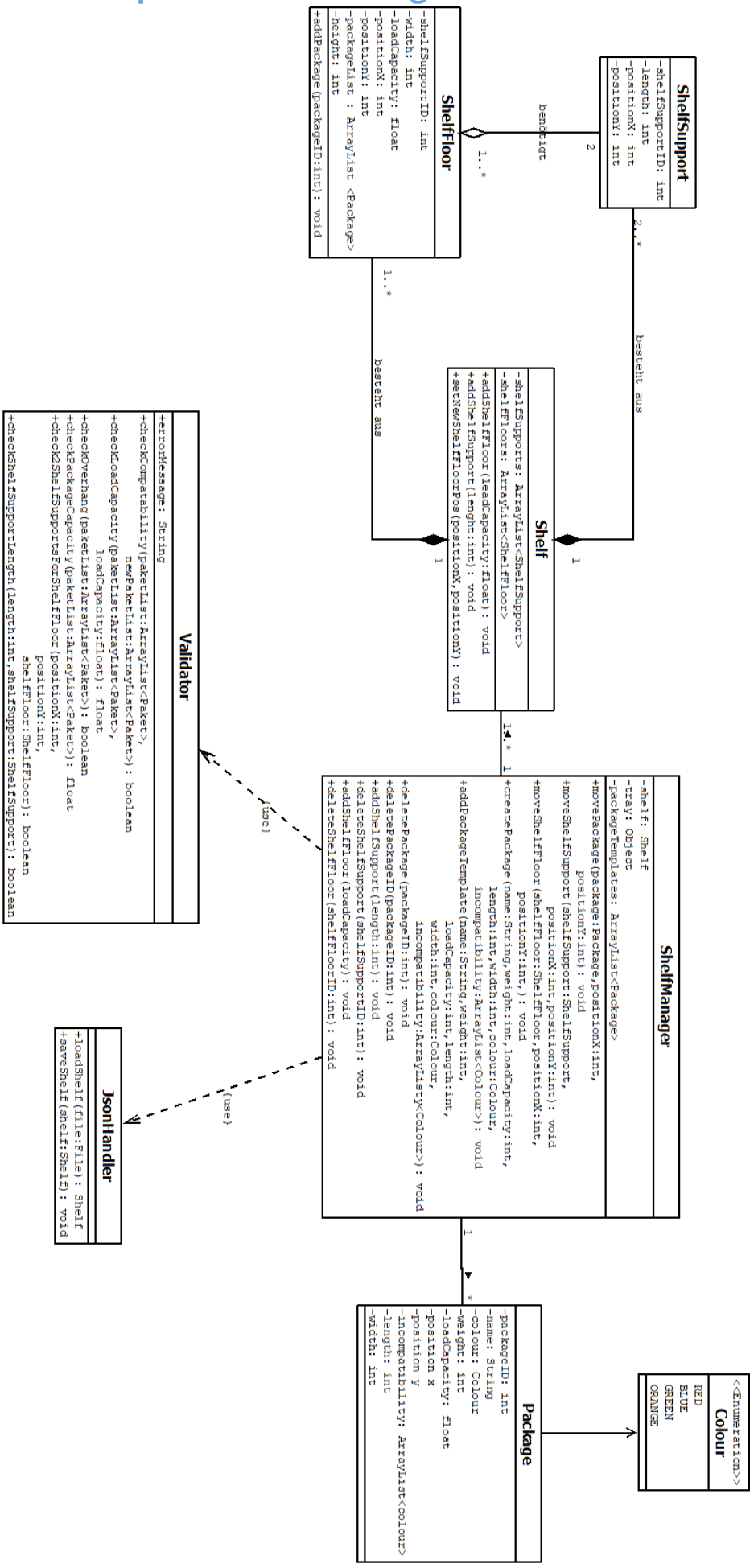


5.3 Subsystem E3: ShelfManager

Das dritte und letzte Subsystem ist der “ShelfManager” mit der Klasse “ShelfManager”. Außerdem greift er noch auf zwei weitere statische Klassen zu: dem “Validator” zum Überprüfen, ob ein Paket korrekt abgelegt wurde, und dem JsonHandler zum Speichern der Lagerkonfiguration. Der ShelfManager umfasst zwei benachbarte Subsystem: “Shelf” und “Package”. Er ist ebenfalls für die Steuerung und Koordinierung der ganzen Logik zuständig. In ihm fließt alles zusammen.



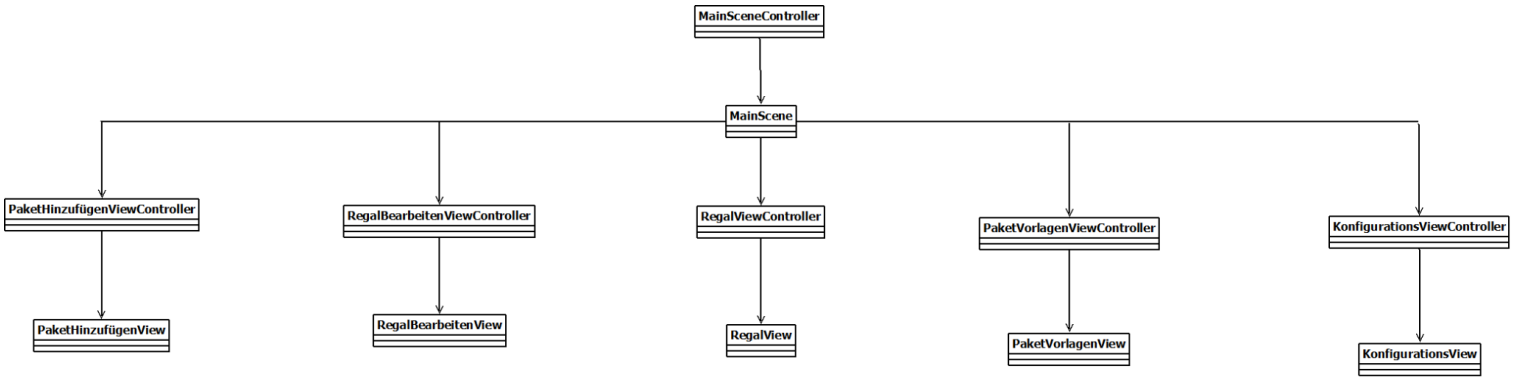
5.4 Die Komplette Business-Logik



5.6 GUI

Da man “Warehouse Masters” nicht mit einer Konsole bedienen möchte, da dieses frei von jeglichen Komfort wäre, wurde zusätzlich eine GUI entwickelt, mit der sich das Programm bedienen lässt.

Im folgenden ist der Aufbau der GUI dargestellt:



5.7 GUI Controller

5.7.1 MainController

Der Main Controller stellt die Verbindung zwischen Businesslogik und GUI dar. Er beinhaltet eine Instanz des ShelfManagers, welcher alle funktionalen Methoden und Objekte verwaltet. Die Methode changeView verwaltet alle Ansichten und öffnet bei bedarf ein neues Fenster.

5.7.2 TrayViewController

Dieser Controller gehört zur Ablage, in welcher alle neuen Pakete abgelegt werden bevor sie ins Regal gesetzt werden. Der Change-Listener der trayList reagiert auf Veränderungen in der Ablage, um diese auch in der Ansicht zu aktualisieren.
Der moveObject Event-Handler reagiert auf Benutzereingaben wie Anklicken und Verschieben eines Paketes.

5.7.3 AddPackageController

Der AddPackageController gehört zum “Paket hinzufügen”-Fenster und verknüpft die Schaltflächen “Fertig” und “neue Vorlage” mit den Funktionen der Businesslogik.

5.7.4 PackageTemplateViewController

Dieser Contoller gehört zur Paketvorlagen-Ansicht. Die beiden “Hinzufügen”- und “Löschen”-Knöpfe sind mit den jeweiligen Event-Handler-Methoden verknüpft.
Der Change-Listener des packageTemplate reagiert auf Änderungen in der Vorlagenliste, um diese darzustellen.

5.7.5 ShelfViewController

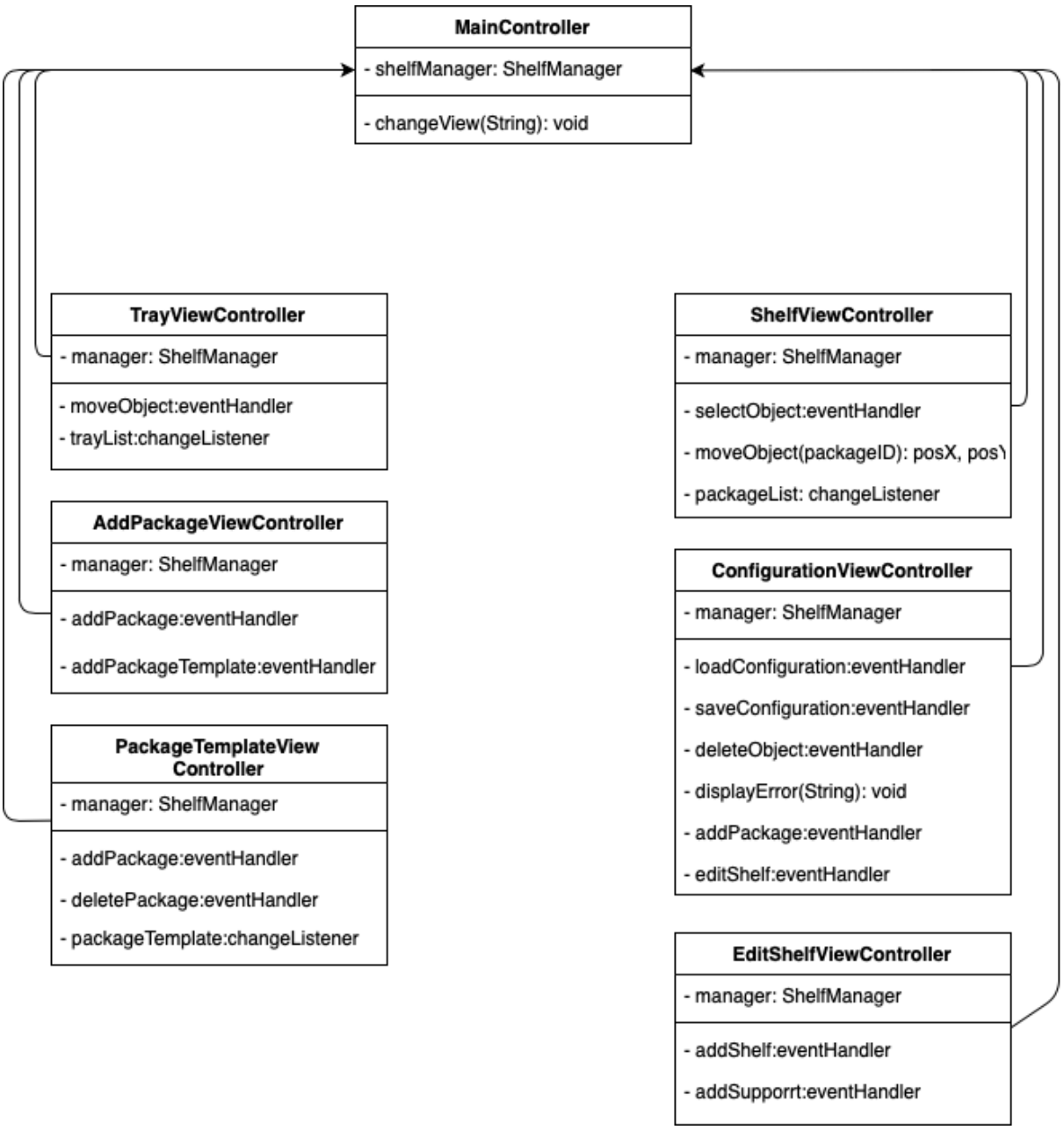
Dieser Controller gehört zur Hauptansicht des Regals. Der Change-Listener der trayList reagiert auf Veränderungen im Regalbestand um diese auch in der Ansicht zu aktualisieren.
Der moveObject Event-Handler reagiert auf Benutzereingaben zum Verschieben eines Paketes.

5.7.6 ConfigurationViewController

Der ConfigurationViewController verknüpft alle Knöpfe der Fußzeile mit den jeweiligen Funktionen.

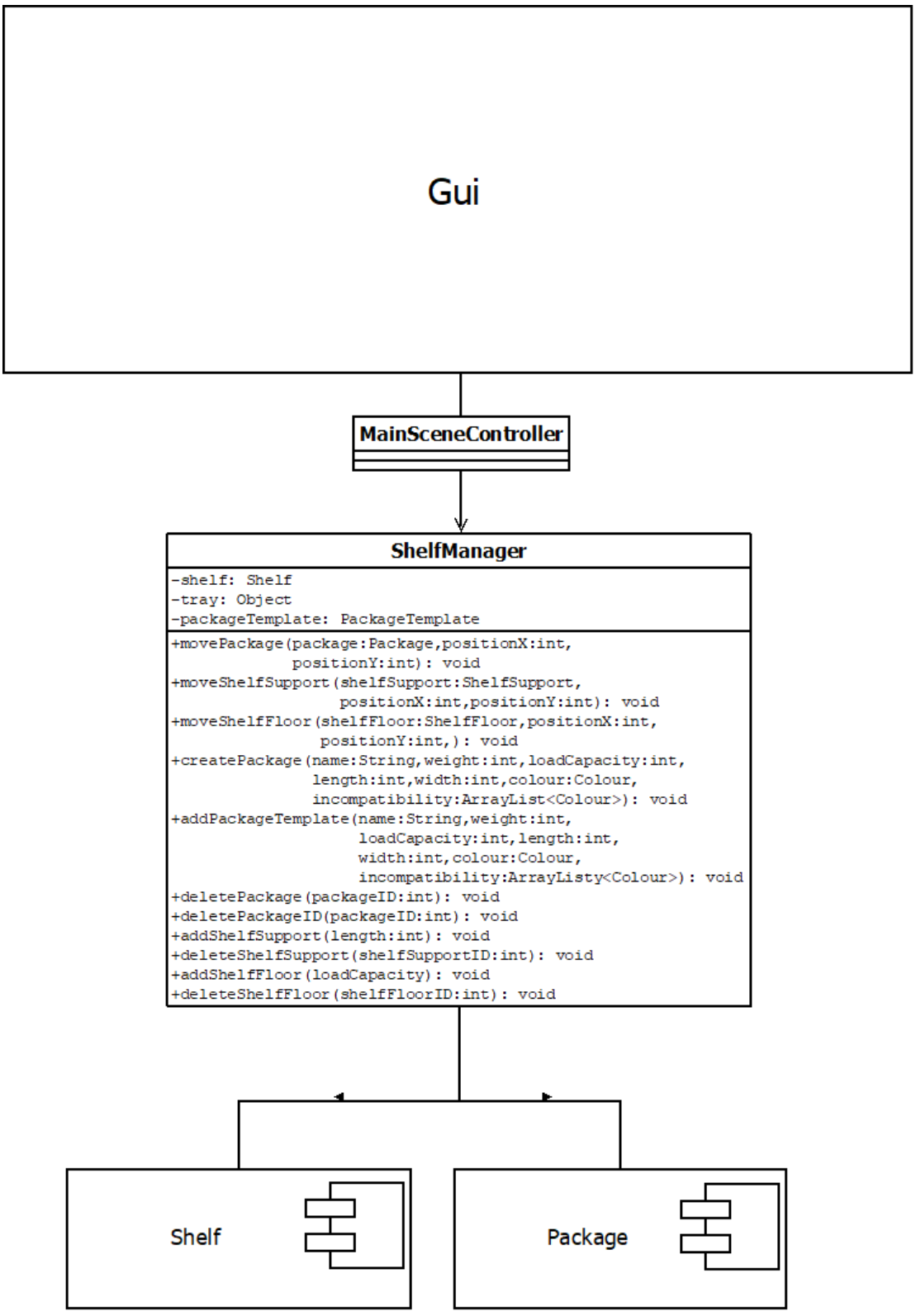
5.7.7 EditShelfViewController

Dieser Controller reagiert auf die Benutzereingabe der Knöpfe zum Hinzufügen eines Regalbodens und zum Hinzufügen einer Stütze.


MainController
- shelfManager: ShelfManager
- changeView(String): void
TrayViewController
- manager: ShelfManager
- moveObject:eventHandler
- trayList:changeListener
AddPackageViewController
- manager: ShelfManager
- addPackage:eventHandler
- addPackageTemplate:eventHandler
PackageTemplateView Controller
- manager: ShelfManager
- addPackage:eventHandler
- deletePackage:eventHandler
- packageTemplate:changeListener
ShelfViewController
- manager: ShelfManager
- selectObject:eventHandler
- moveObject(packageID): posX, pos\
- packageList: changeListener
ConfigurationViewController
- manager: ShelfManager
- loadConfiguration:eventHandler
- saveConfiguration:eventHandler
- deleteObject:eventHandler
- displayError(String): void
- addPackage:eventHandler
- editShelf:eventHandler
EditShelfViewController
- manager: ShelfManager
- addShelf:eventHandler
- addSupporrt:eventHandler

5.8 Business-GUI Verknüpfung

Im folgenden lässt sich erkennen wie die Businesslogik und die grafische Benutzeroberfläche miteinander verknüpft sind. Die zwei Bindeglieder, die zusammenhängen, und Business sowie GUI verknüpfen sind der “ShelfManager” (auf der Businesslogik-Seite) und der “MainSceneController” (auf der GUI-Seite). Die GUI und Logik werden nach dem MVC-Prinzip behandelt. Das heißt, sie sind losgelöst voneinander. Die GUI stellt nur das darüber liegende Interaktionskonzept da.



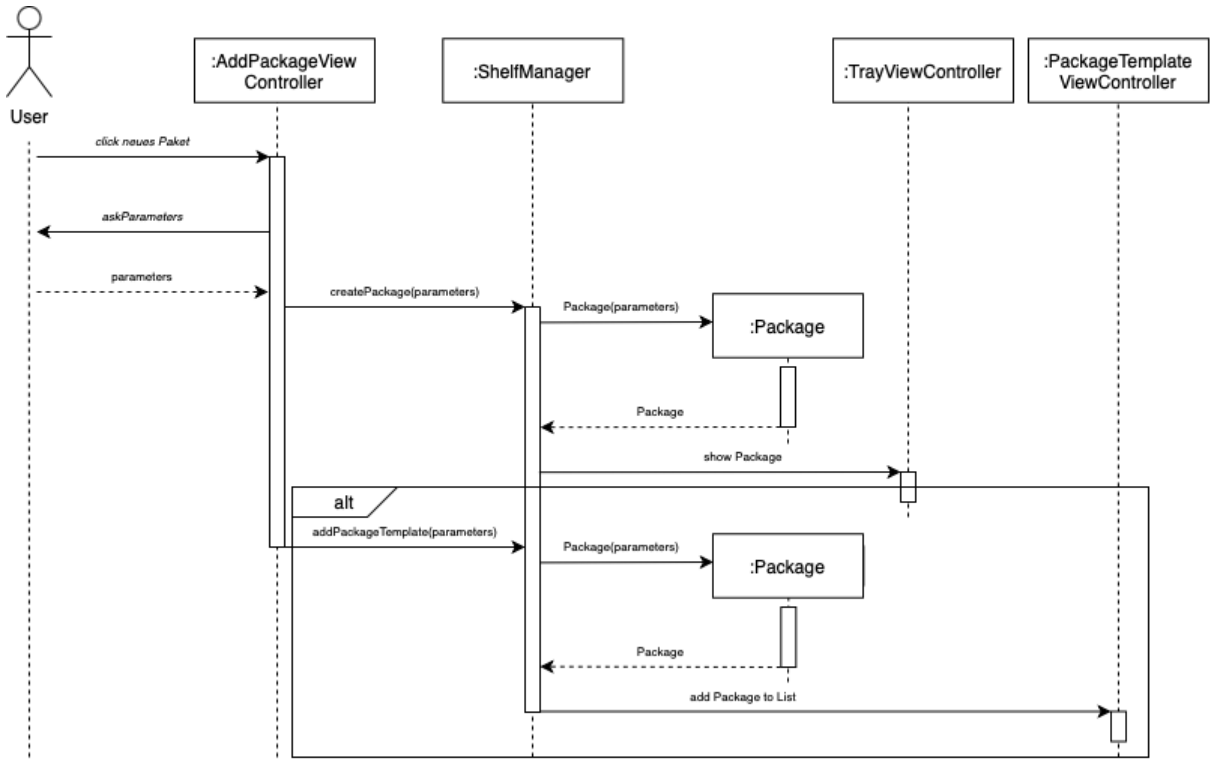
6. Laufzeitsicht

6.1 Paket hinzufügen

Diese Funktion wird vom User in der ControlView durch Klicken auf “Paket hinzufügen” angestoßen. In einem Pop-Up Fenster können alle erforderlichen Informationen für das neue Paket eingegeben werden. Durch Klicken auf “Fertig” wird das neue Paket erstellt und erscheint auf der Ablage.

Alternativ kann die Konfiguration durch Klicken auf “neue Vorlage” als neue Paketvorlage gespeichert werden. In diesem Fall erscheint das neue Paket in der PackageTemplateView.

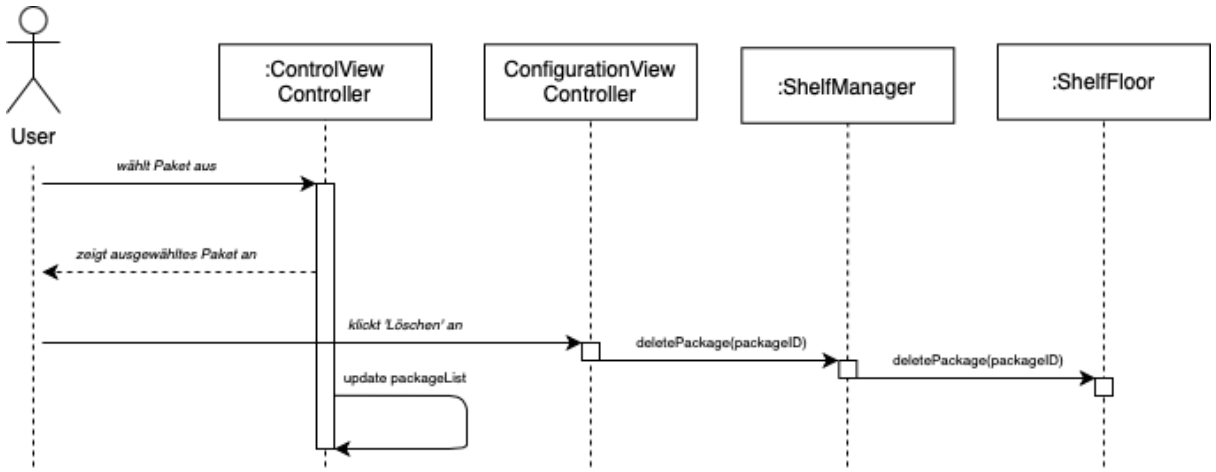
- Der User möchte durch Klick auf die entsprechende Schaltfläche ein neues Paket hinzufügen
- In einem Pop-Up Fenster gibt der User alle Parameter an
- Der Regal Manager erstellt mit den erhaltenen Parametern ein neues Paket
- Das neue Paket wird im Tray (Ablage) gespeichert
- Der ChangeListener der Ablage erkennt das neue Paket und zeigt es an



6.2 Paket löschen

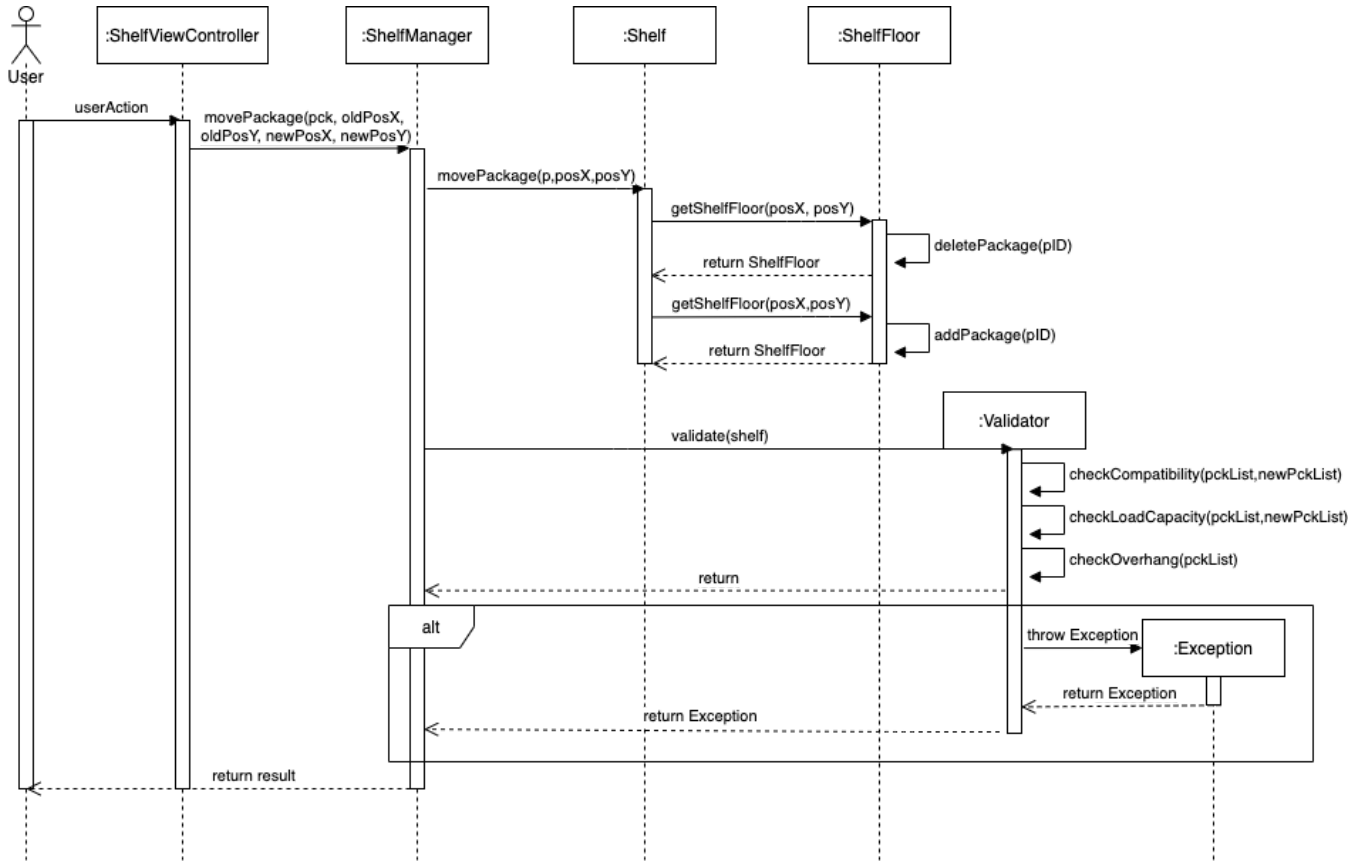
Um ein Paket zu löschen wird dieses zuerst in der ShelfView ausgewählt und anschließend über den Button “Löschen” entfernt.

- User wählt das zu löschende Paket im Hauptfenster durch Anklicken aus
- User drückt den “Löschen”-Knopf
- Der ShelfViewController übergibt dem ShelfManager das zu löschende Paket
- Die ShelfView aktualisiert die Ansicht



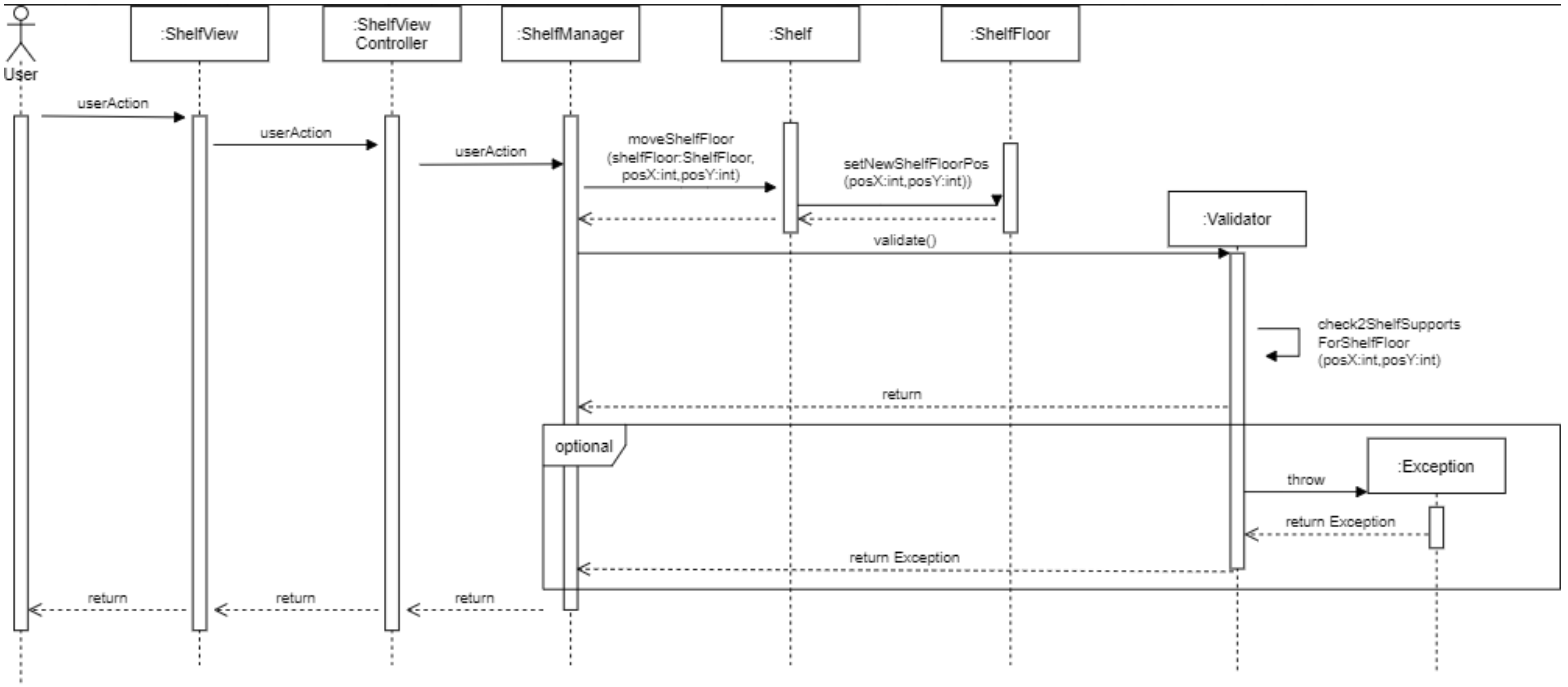
6.3 Paket verschieben

Wenn ein Paket verschoben wird, wird die Methode “movePackage” im ShelfManager aufgerufen. Das Paket wird vorerst verschoben und vom alten Regalboden entfernt. Anschließend wird überprüft, ob das Paket oder der verschobene Paketstapel an der neuen Position die Bedingungen erfüllt. Falls dies nicht der Fall ist, wird eine Exception geworfen und der Fehler dem User grafisch als auch in Form einer Nachricht mitgeteilt. Der User erhält auch eine Bestätigung, falls die Ausführung erfolgreich war. Ansonsten wird der User aufgefordert den Fehler zu beheben.



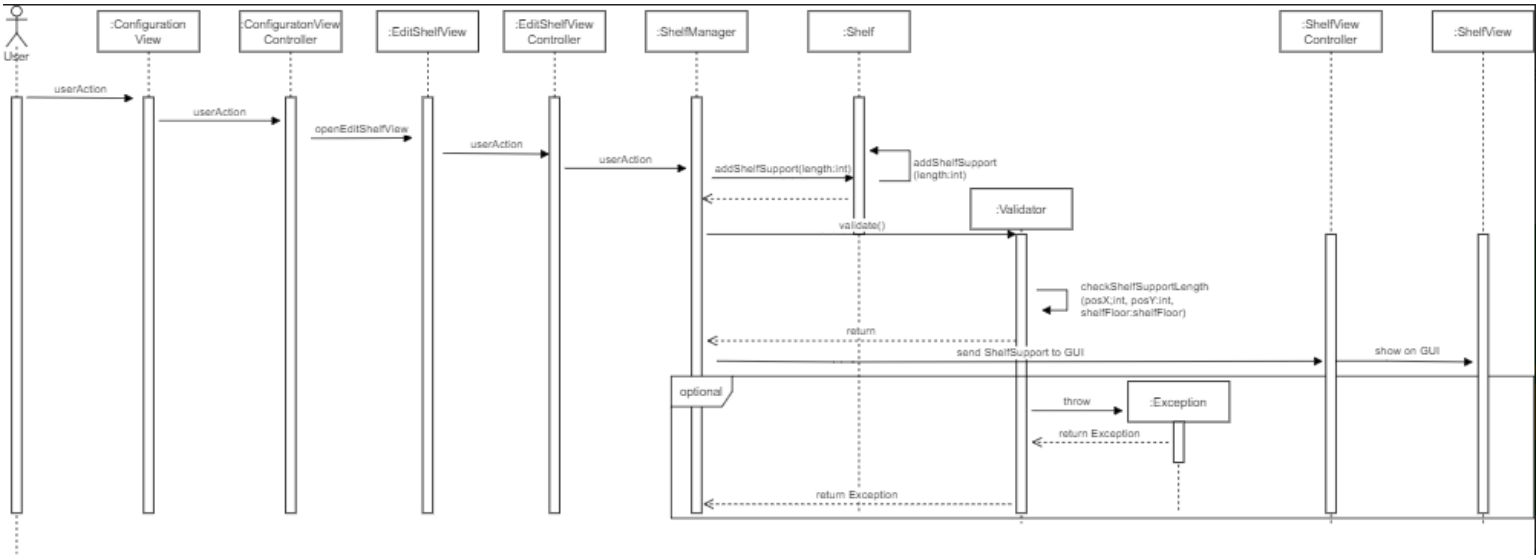
6.4 Regalboden verschieben/Anbringen

Um ein Regalboden an zwei Regalstützen anzubringen, wählt der User in der ShelfView den richtigen Regalboden aus und verschiebt ihn. Daraufhin greift der ShelfViewController auf den ShelfManager der Businesslogik zu. Von dort aus ruft der ShelfManager die Methode “moveShelfFloor”. Diese greift wiederum auf das Regal (Shelf) zu und ruft dort die Methode “setNewShelfFloorPos” auf. Diese ändert dann letztendlich den Positionswert im ShelfFloor. Die Informationen gehen zurück an den ShelfManager und werden dann vom Validator überprüft. Der Validator ruft die Methode “check2ShelfsupportForShelfFloor” auf um zu überprüfen, ob der Regalboden an der richtigen Stelle platziert wurde. Ist dies der Fall, werden die neuen Informationen an die GUI gegeben. Ist dies nicht der der Fall, wird eine Exception geworfen.



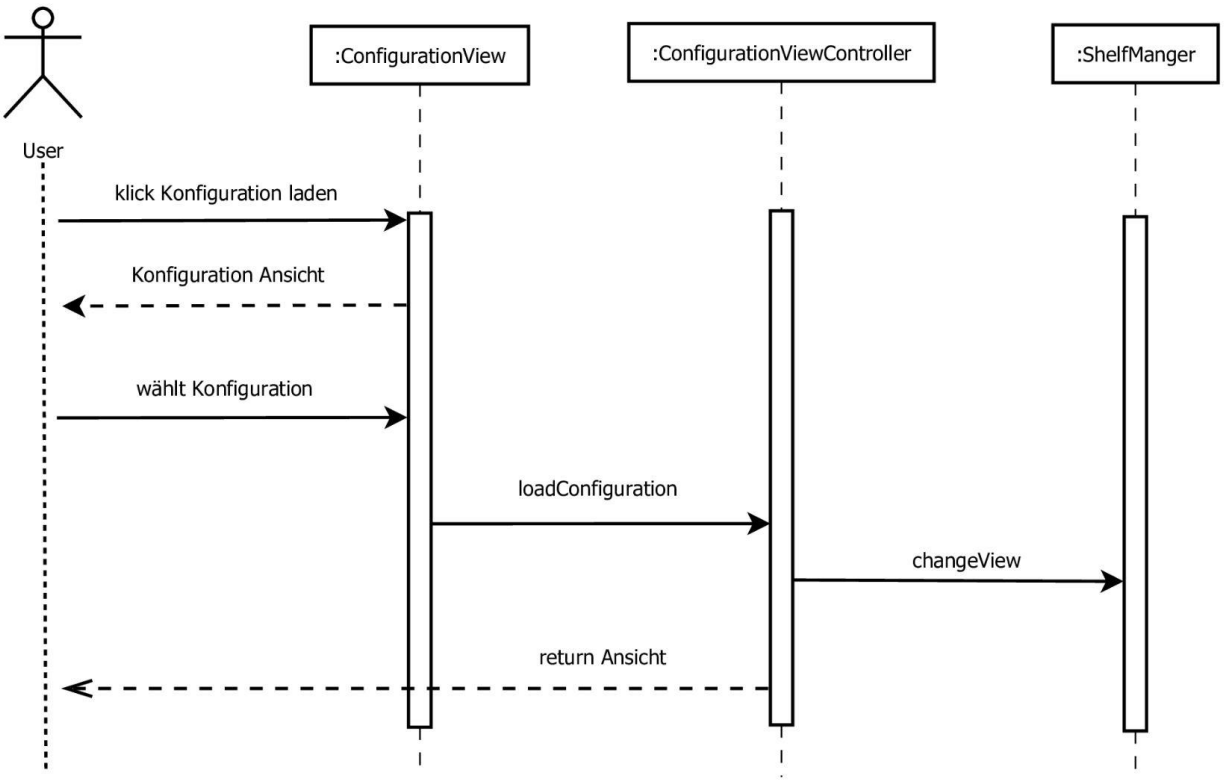
6.5 Regalstützen hinzufügen

Um Regalstützen hinzuzufügen betätigt der User in der ConfigurationsView die Schaltfläche “Regel bearbeiten”. Daraufhin vom ConfigurationsViewController die EditShelfView aufgerufen. Diese ermöglicht den User eine neue Regalstütze zu erstellen. Dafür muss er die Länge der Regalstütze eingeben. Diese Informationen gelangen letztendlich an den ShelfManager der Businesslogik. Dieser ruft die Methode “addShelfSupport” mit dem gesetzen Längenparameter auf. Diese Methode ruft dann wiederum im Shelf die Methode “addShelfSupport” auf, die dann letztendlich ein neues ShelfSupport-Objekt erstellt. Vor der Erstellung überprüft der Validator noch mit der Methode “checkShelfSupportLength”, ob die Länge der Regalstütze passend ist. Ist dies der Fall, wird die Information an die GUI gegeben. Falls nicht wird eine Exception geworfen.



6.6 Lagerkonfiguration laden

Um eine Lagerkonfiguration zu laden klickt der User auf “Konfiguration laden”, wodurch er die ConfigurationView betätigt. Der User kann aus dieser Ansicht eine Konfiguration wählen. Nachdem die Konfiguration ausgewählt ist, wird die Methode “loadConfiguration” im ConfigurationViewController aufgerufen und es wird an das ShelfManger weitergeleitet.
Die ausgewählte Konfiguration wird vom ConfigurationViewController zurückgegeben und dem User angezeigt.



7. Konzepte

Abhängigkeiten

Das Projekt besteht aus drei Hauptkomponenten. Das Regal selbst und die Pakete, die jeweils nur die Objekte darstellen und den Regal-Manager, der die Verknüpfung bildet. Aus dem Grund sind diese drei Klassen stark voneinander abhängig.

Im Gegensatz dazu steht das Speichersystem unabhängig von den Klassen. Die Speichersystem-Klasse bietet lediglich die Möglichkeit an, ist aber nicht essentiell. Sie beinhaltet nur eine Reihe von Methoden, die zum Speichern und Laden verantwortlich sind. Beim Laden muss jedoch eine JSON-Datei vorhanden sein. Falls dies nicht der Fall ist, muss eine Fehlerbehandlung stattfinden.

In diesem Projekt wird eine GUI angeboten, die nach dem MVC-Prinzip aufgebaut ist. Es herrscht also eine Abhängigkeit zwischen der View und dem jeweiligen Controller. Die Controller wiederum haben Zugriff auf die Business-Schicht. Der Zugriff ist also nur auf die direkt darunter liegende Schicht möglich.

Validierung

Damit überprüft wird, ob ein Paket korrekt abgelegt wurde, kommt ein sogenannter Validator zum Einsatz. Diese Klasse beinhaltet Methoden, die der Regal-Manager (ShelfManager) nutzt um eventuelle Fehler zu erkennen und diese anschließend dem User auf der GUI anzuzeigen.

Fehlerbehandlung

Bei der Validierung werden bei Problemen Exceptions geworfen, für jeden Fehler gibt es eine eigenen Exception, die jeweils dem User weitergeleitet werden, damit der User weiß, um welches Problem es sich handelt und ihn beheben kann.

Testbarkeit

Zum Testen des Programms werden kleine Unit Tests geschrieben, die bestimmte Fälle überprüfen, ob das Programm die User Actions korrekt ausführt.

Persistenz

Das Speichern erfolgt durch einen JSON-Parser, der das Paket als auch das gesamte Regal als JSON-Objekt in eine JSON-Datei schreibt und lokal auf dem Rechner speichert. Mit Hilfe eines FileChoosers kann der User eine geeignete Regalkonfiguration wiederherstellen.

8. Entwurfsentscheidungen

Die Entwurfsentscheidungen, sollen die gesamten Programmentscheidungen beschreiben, die im Architektur geplant wurden. Dazu gehören auch alle verworfene Entscheidungen, die nicht im End-Architektur eingeflossen sind. Damit wird veranschaulicht, welche Alternativen und Entwürfe es gab und warum die verworfen wurden. Somit werden alle Entscheidungen dokumentiert und dessen Verwerfung Grund ebenfalls.
Die Dokumentierung verhindert wieder auftretende Entscheidungen und Fehlern, die im laufe der Entwicklung verworfen wurden. Die Arbeitszeit wird dadurch verkürzt.

Entscheidungen:

- Entwicklung vom RegalManger (ShelfManger), da ein Regal klasse alleine keine Kontrollmöglichkeiten anbietet.

9. Qualitätsszenarien

Bewertung Szenario

1. Ein Person, der UML Grundkenntnisse hat, kann das Lagerprogramm, anhand dessen Diagramme und Handouts, in 15 Minuten verstehen.
2. Ein Entwickler, der erfahren in Java ist, sucht eine Implementation im entwurf und findet es Problemlos im Quellcode.
3. Ein Entwickler, kann das Programm ganz einfach erweitern, soweit er sich an die Architektur und Quellcode hält

10. Glossar

Titel	Beschreibung/Erläuterung
Businesslogik	= Geschäftslogik In der Schichtenarchitektur ist dies die Schicht, die sich zwischen Datenbank und Benutzerschnittstelle befindet, um eine Abgrenzung zu schaffen.
Controller	verwaltet die Präsentation und das Modell, indem sie durch Benutzerinteraktionen informiert wird und dies an die Businesslogik weiterleiten kann
Model	enthält Daten, die von der View dargestellt werden
MVC	= Model View Controller, ist ein Muster zur Unterteilung einer Anwendung in die drei Komponenten Datenmodell, Präsentation und Programmsteuerung, die flexibel zu verändern, zu erweitern und wiederzuverwenden ist
Stakeholder	Person, die aufgrund ihrer Interessenlage Einfluss hat, wie ein bestimmtes Unternehmen handelt (z. B. Aktionär, Mitarbeiter, Kunde, Lieferant)
Subsystem	Bereich innerhalb eines Systems, welcher eigene Merkmale von einem Systems aufweist
View	ist die Präsentation, die für die Darstellung der Daten zuständig ist