Entwicklerdokumentation

Version	Bearbeitungsdatum	Autor
1.0.0	17.11.2019	Till Große, Tobias Köllner, Anna Lopatkina, Kien Dang Tran, Viktor Reusch

1. Einführung und Ziele

1.1. Aufgabenstellung

Für *Miss-Mint Mending Points* soll für die Mitarbeiter eine Software bereitgestellt werden. Mitarbeiter (*Staff*) können über den Admin (*ADMIN*) registriert (*registerUser*) und entfernt (*deleteUser*) werden. Unauthorisierte Personen sollen keinen Zugriff auf betriebsinterne Bereiche bekommen. Der Betrieb stellt folgende Dienstleistungen (*ServiceCategory*) bereit:

- eine Flickschusterei (*KLUDGE*)
- einen Nähservice (*SEWINGSERVICE*)
- einen Schlüsseldienst (*LOCKSMITH*)
- eine Schnellreinigung (*EXPRESS_DRY_CLEANING*)
- eine Elektrowerkstatt (*ELECTRONICWORKSHOP*)
- eine Scherenschleiferei (*SCISSORSGRINDERY*)

Nachdem der Kunde eine Dienstleistung ausgewählt und im Voraus bezahlt hat, wird ein Auftrag (*MissMintOrder*) mit einer Auftragsnummer (*OrderIdentifier*) im System angelegt. Zusätzlich wird ein erwartetes Fertigstellungsdatum (*getExpectedFinished*) berechnet. Der Auftrag kann sich in verschiedenen Zuständen (*OrderState*) befinden.

Für die Bearbeitung der Aufträge wird eine Raumplanung inklusive Zeitplan (*TimeTable*) zur Verfügung gestellt. Dementsprechend werden Räume (*Room*) passend zur Dienstleistung gebucht (*BOOKING*). Für die Dienste wird i. A. Material (*Material*) benötigt. Diese können, falls nötig, nachbestellt werden und die Kosten (*getPrice*) werden automatisch vom System erfasst.

Ab dem Fertigstellungstermin kann die Ware eine Woche lang unter Vorlage des Auftragszettels abgeholt werden. Danach wandert die Ware für drei Monate in eine Aufbewahrungsstelle, wo sie unter Vorlage des Auftragszettels durch Zahlung einer Aufbewahrungsgebühr von 0,50€ pro Woche ausgelöst werden kann. Waren, die auch in diesem Zeitraum nicht abgeholt werden, fallen anschließend an eine karitative Organisation. Umgekehrt wird für jeden ganzen verspäteten Tag Kosten in Höhe von 10% (bis maximal 100%) erstattet.

Jegliches Einkommen bzw. Ausgaben (*FinanceItem*) sind für den Filialleiter sichtbar. Dabei werden sowohl die Gesamtkosten (*calculateTotalPrice*) als auch der Reingewinn erfasst.

1.2. Qualitätsziele

Wartbarkeit

Dieses Merkmal gibt den Grad an Effektivität und Effizienz an, mit dem ein Produkt oder System modifiziert werden kann, um es zu verbessern, zu korrigieren oder an Änderungen in der Umgebung und in den Anforderungen anzupassen.

Nutzerfreundlichkeit

Grad, in dem ein Produkt oder System von bestimmten Benutzern verwendet werden kann, um bestimmte Ziele mit Effektivität, Effizienz und Zufriedenheit in einem bestimmten Verwendungskontext zu erreichen.

Sicherheit

Grad des Schutzes von Informationen und Daten durch ein Produkt oder System, sodass Personen oder andere Produkte oder Systeme über den Datenzugriff verfügen, der für ihre Art und ihre Berechtigungsstufe angemessen ist.

Desgin

Gestaltung der Benutzeroberfläche, um die visuelle Wahrnehmung des Kunden zu beeinflussen, und so die Informationsvermittlung und Nutzerfreundlichkeit zu steigern. Dies beinhaltet Optimierungen für verschiedene Auflösungen und verschiedene Browser. Wichtig ist auch die visuelle Differenzierung von Inhalten, um die Informationsaufnahme des Kunden zu lenken.

1 = Nicht wichtig .. 5 = Sehr wichtig

Qualitätsanspruch	1	2	3	4	5
Wartbarkeit				x	
Benutzerfreundlichkeit					x
Sicherheit				x	
Design		x			

2. Randbedingungen

2.1. Hardware-Vorgaben

Eine Liste von Geräten/Hardware, die benötigt werden um die Software zu benutzen:

- Server
- Computer
- Tastatur
- Maus

2.2. Software-Vorgaben

Die Anwendung setzt folgende Software voraus:

- Java 11 oder neuer
- Mozilla Firefox 70+
- Chromium 78+

2.3. Vorgaben zum Betrieb des Software

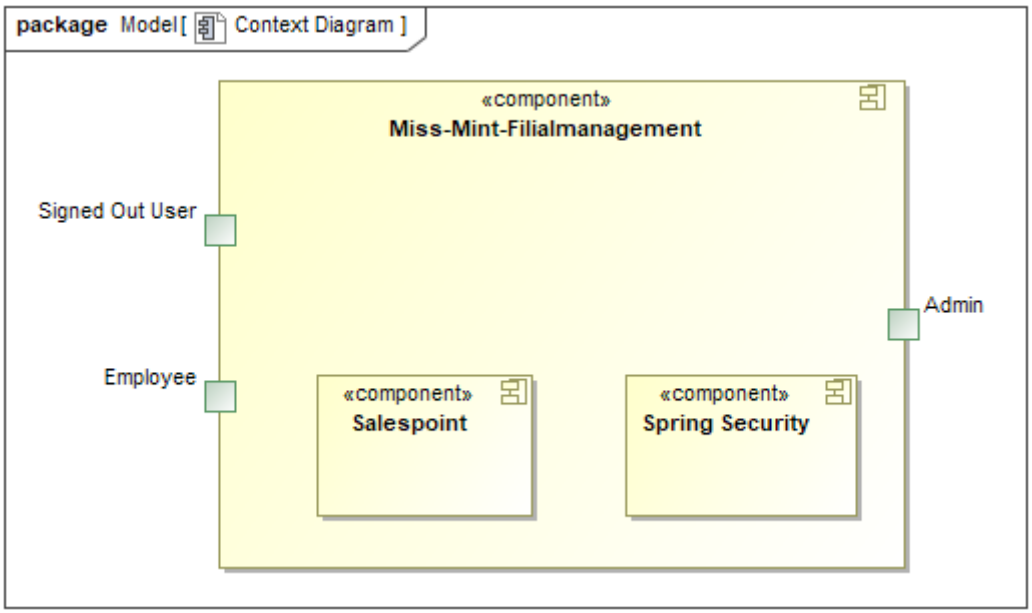
Die Software wird ausschließlich von *Miss Mint Mending Points* Mitarbeiter verwendet. Das System stellt Funktionen für die Annahme, Reparaturplanung, Ausgabe und Lagerung der Ware, sowie die Verwaltung der Ressourcen zur Verfügung. Zudem erhält der Filialleiter Einsicht auf die Finanzen.

Die Software soll rund um die Uhr in Betrieb sein und nur über das interne Firmennetz erreichbar sein.

Die Hauptnutzer der Software sind Mitarbeiter, die typische Website-Navigationsschemata kennen, sowie ein Filialleiter (Admin), der nicht unbedingt über einen technischen Hintergrund verfügt.

Das System muss technisch gewartet werden, da Sicherheitsupdates für Java aber auch die verwendeten Frameworks eingespielt werden müssen. Alle Daten müssen dauerhaft in einer Datenbank gespeichert sein und über die Applikation zugänglich sein (z.B. sollten für einen Filialleiter keine SQL-Kenntnisse erforderlich sein).

3. Kontextabgrenzung

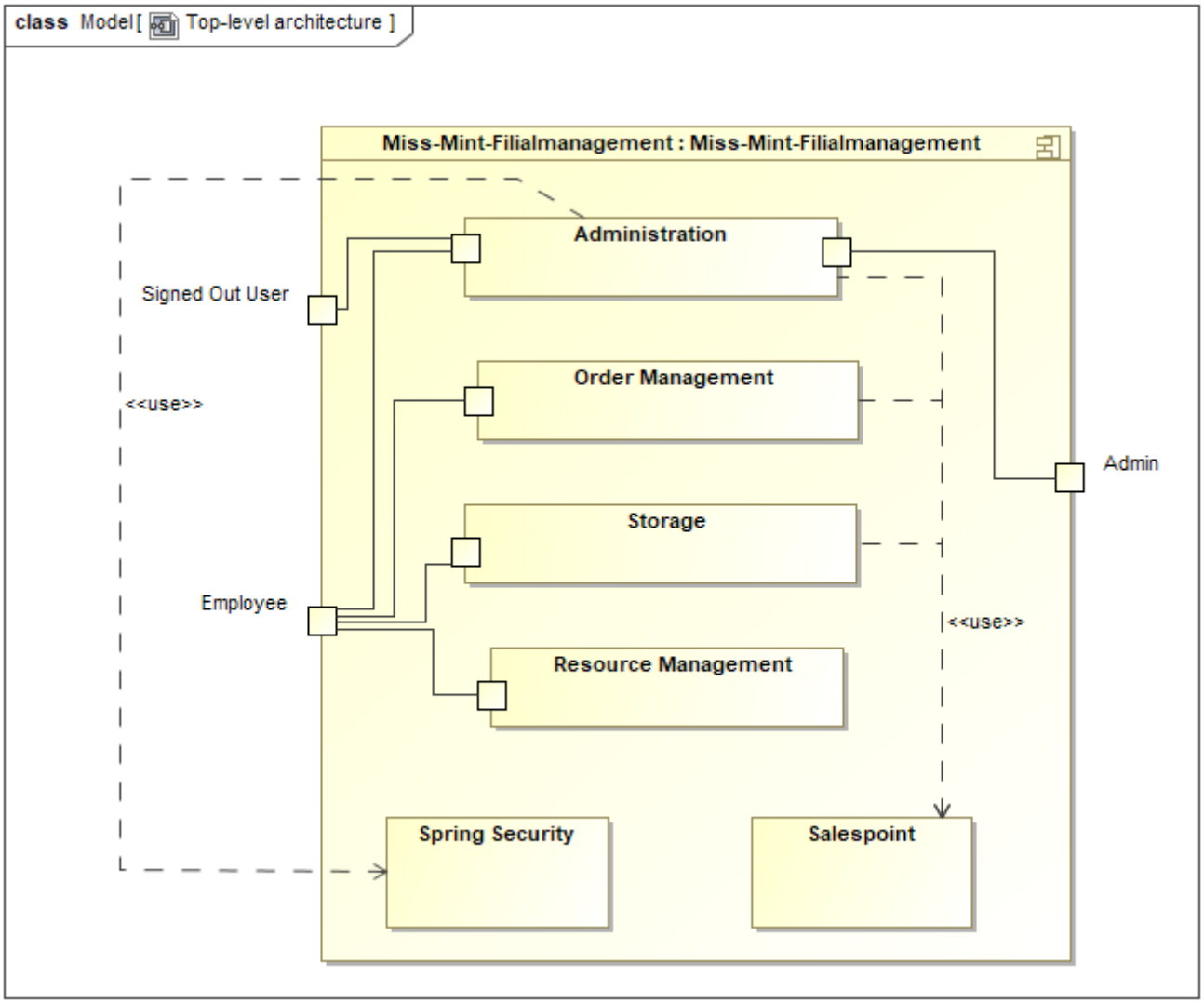


4. Lösungsstrategie

4.1. Erfüllung der Qualitätsziele

Qualitätsziel	Lösungsansatz
Wartbarkeit	• **Modularität**: Zusammensetzen der Anwendung aus möglichst eigenständigen Modulen um den Einfluss den die Änderung eines Moduls auf andere Module hat zu minimieren. • **Wiederverwendbarkeit**: Sicherstellung der Wiederverwendbarkeit von Modulen durch andere Systemkomponenten • **Modifizierbarkeit**: Die Anwendung sollte ohne Verschlechterung der Code-Qualität oder Auftreten von Fehlern veränderbar und/oder erweiterbar sein.
Nutzerfreundlichkeit	• **Erlernbarkeit**: Das System sollte verständlich und einfach bedienbar sein. Das lässt sich z.B. durch eindeutige Beschreibung von Eingaben mit Hilfe von Tooltips und/oder Labels realisieren. • **Handhabung von Fehlern**: Nutzer sollten vor Fehlern geschützt werden. Eingaben dürfen unter keinen Umständen zu ungültigen Systemzuständen führen. • **Ästhetik der Nutzerschnittstelle**: Bereitstellung einer ansprechenden und zufriedenstellenden Interaktion für den Nutzer • **Zugänglichkeit**: Es sollte, z.B. durch die Nutzung passender Schriftgrößen und Kontraste, sichergestellt werden dass Menschen mit einer möglichst großen Bandbreite von Eigenschaften die Möglichkeiten des Systems vollständig nutzen können.
Sicherheit	• **Vertraulichkeit**: Daten dürfen nur von Menschen mit den dafür nötigen Zugriffsrechten eingesehen werden. Dies kann mit Spring Security und dem Thymeleaf (`sec:authorize`-Tag) realisiert werden. • **Integrität**: Nicht-autorisierte Modifikationen sollten verhindert werden. Dafür kann die Spring Security (`@PreAuthorize` - annotation) verwendet werden. • **Verantwortung**: Nachverfolgbarkeit von Aktionen oder Ereignissen zu einer eindeutigen Entität oder Person.

4.2. Softwarearchitektur



4.3. Entwurfsentscheidungen

4.3.1. Verwendete Muster

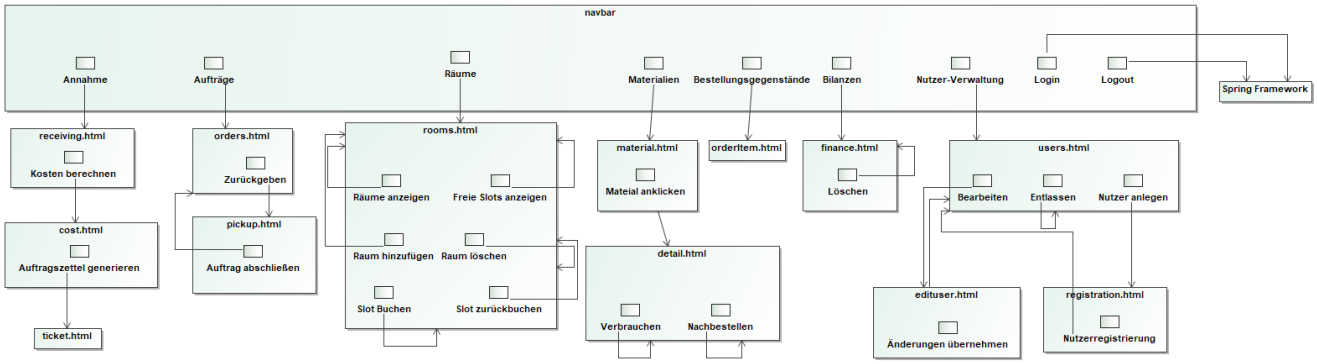
- **Model View Controller** mit Spring MVC
- **Singleton mit Springs** `@Component` etc.
- **Value Object** mit `MonetaryAmount` etc.
- **Data Transfer Object** mit `OrderDTO`, um Daten für Templates zu aggregieren.
- **Dependency Injection** über die Konstruktoren in Spring-Komponenten
- **Repository** mit den Spring-Repositories

4.3.2. Persistenz

Die Anwendung verwendet Hibernate-Annotation-basiertes Mapping, um Java-Klassen Datenbanktabellen zuzuordnen. Als Datenbank wird H2 verwendet. Die Persistenz ist standardmäßig deaktiviert. Um den Persistenzspeicher zu aktivieren, müssen die folgenden zwei Zeilen in der Datei application.properties nicht kommentiert werden:

```
# spring.datasource.url=jdbc:h2:./db/missmint
# spring.jpa.hibernate.ddl-auto=update
```

4.3.3. Benutzeroberfläche



4.3.4. Verwendung externer Frameworks

Externe Klasse	Pfad der externen Klasse	Verwendet von (Klasse der eigenen Anwendung)
Assert	org.springframework.util.Assert	CatalogDataInitializer, InventoryInitializer
BusinessTime	org.salespointframework.time.BusinessTime	ReceivingController, OrderDataInitializer, OrderService
Catalog	org.salespointframework.catalog.Catalog	OrdersController, ReceivingController, OrderDataInitializer, ServiceDataInitializer, InventoryCatalog
Component	org.springframework.stereotype.Component	OrderDataInitializer, ServiceDataInitializer, InventoryInitializer, CatalogDataInitializer
Controller	org.springframework.stereotype.Controller	OrdersController, PickUpController, ReceivingController, UserController, InventoryController, CatalogController
CrudRepository	org.springframework.data.repository.CrudRepository	EntriesRepository, RoomsRepository, StaffRepository
DataInitializer	org.salespointframework.core.DataInitializer	OrderDataInitializer, ServiceDataInitializer, CatalogDataInitializer, InventoryInitializer, TempUserInitializer
Errors	org.springframework.validation.Errors	ReceivingController, UserController
EURO	org.salespointframework.core.Currencies.EURO	CatalogDataInitializer

Externe Klasse	Pfad der externen Klasse	Verwendet von (Klasse der eigenen Anwendung)
GetMapping	org.springframework.web.bind.annotation.GetMapping	PickUpController, ReceivingController, UserController, CatalogController
HttpStatus	org.springframework.http.HttpStatus	PickUpController, ReceivingController
LoggedIn	org.salespointframework.useraccount.web.LoggedIn	ReceivingController
Metric	org.salespointframework.quantity.Metric	UltimateProduct,CatalogDataInitializer
Model	org.springframework.ui.Model	OrdersController, PickUpController, ReceivingController, UserController, CatalogController
Money	org.javamoney.moneta.Money	OrderService, ServiceDataInitializer, UltimateProduct, CatalogDataInitializer
OnDelete	org.hibernate.annotations.OnDelete;	Room, TimeTableEntry
OnDeleteAction	org.hibernate.annotations.OnDeleteAction;	Room, TimeTableEntry
Order	org.salespointframework.order.Order	MissMintOrder, ServiceDataInitializer
Order	org.springframework.core.annotation.Order	OrderDataInitializer
OrderManager	org.salespointframework.order.OrderManager	OrdersController, PickUpController, ReceivingController, OrderDataInitializer, OrderService
Page	org.springframework.data.domain.Page	OrdersController
Pageable	org.springframework.data.domain.Pageable	OrdersController, OrderService
Pair	org.springframework.data.util.Pair	ServiceDataInitializer
PathVariable	org.springframework.web.bind.annotation.PathVariable	PickUpController, UserController
Product	org.salespointframework.catalog.Product	Service, UltimateProduct
ProductIdentifier	org.salespointframework.catalog.ProductIdentifier	OrdersController, ReceivingForm
PostMapping	org.springframework.web.bind.annotation.PostMapping	PickUpController, ReceivingController, UserController
PreAuthorize	org.springframework.security.access.prepost.PreAuthorize	OrdersController, PickUpController, ReceivingController, UserController
Qualifier	org.springframework.beans.factory.annotation.Qualifier	ReceivingController
Quantity	org.salespointframework.quantity.Quantity	MissMintOrder, InventoryInitializer

Externe Klasse	Pfad der externen Klasse	Verwendet von (Klasse der eigenen Anwendung)
RequestMapping	org.springframework.web.bind.annotation.RequestMapping	OrdersController
ResponseStatusException	org.springframework.web.server.ResponseStatusException	PickUpController, ReceivingController
Service	org.springframework.stereotype.Service	OrderService, UserManagement
Sort	org.springframework.data.domain.Sort	InventoryCatalog
UniqueInventory	org.salespointframework.inventory.UniqueInventory	InventoryInitializer
UniqueInventoryItem	org.salespointframework.inventory.UniqueInventoryItem	InventoryInitializer, InventoryController, CatalogController
UserAccount	org.salespointframework.useraccount.UserAccount	ReceivingController, MissMintOrder, OrderDataInitializer, User, UserManagement
UserAccountManager	org.salespointframework.useraccount.UserAccountManager	OrderDataInitializer, UserManagement
Value	org.springframework.beans.factory.annotation.Value	OrderService, ServiceDataInitializer

5. Bausteinsicht

5.1. Aufträge

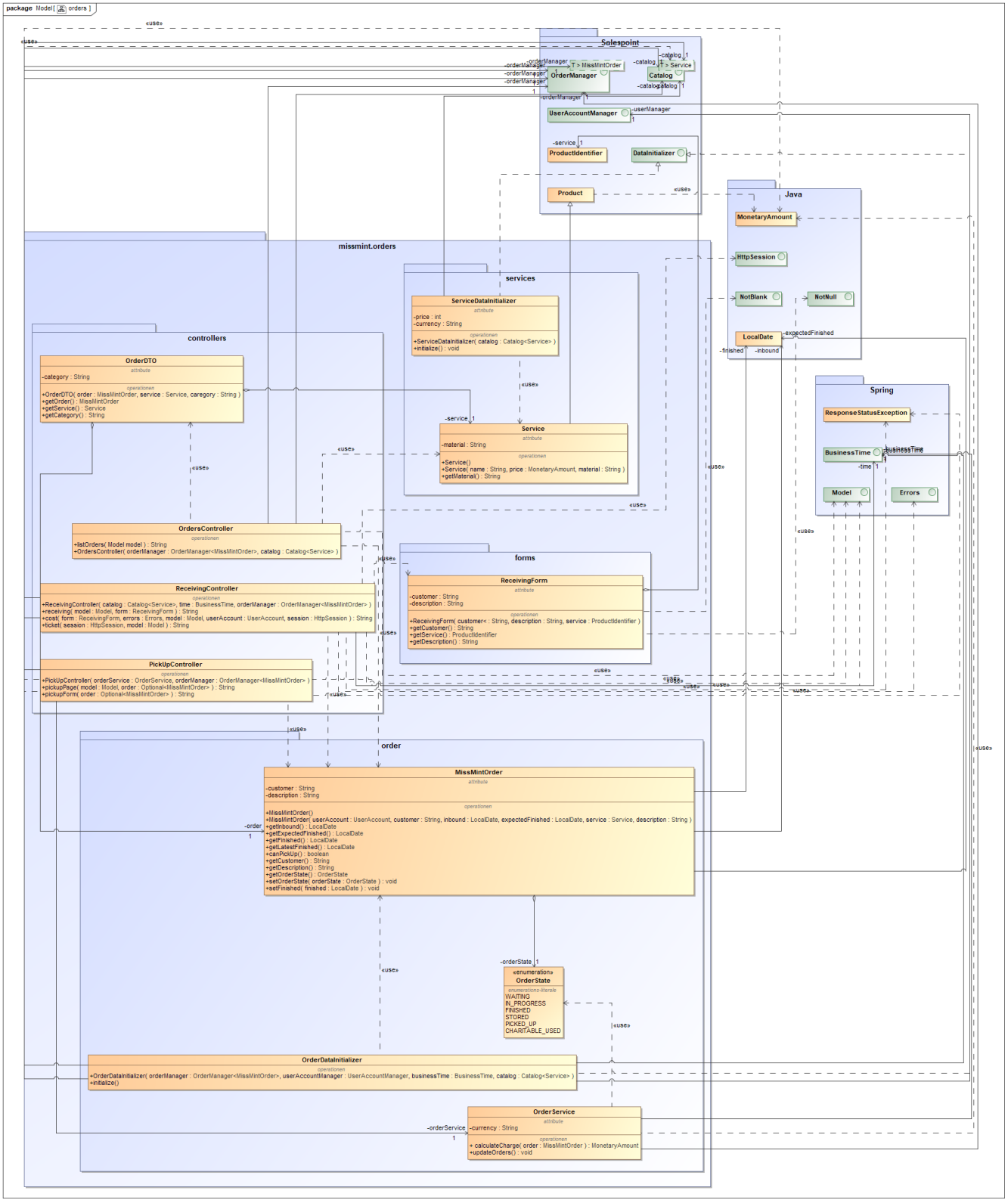
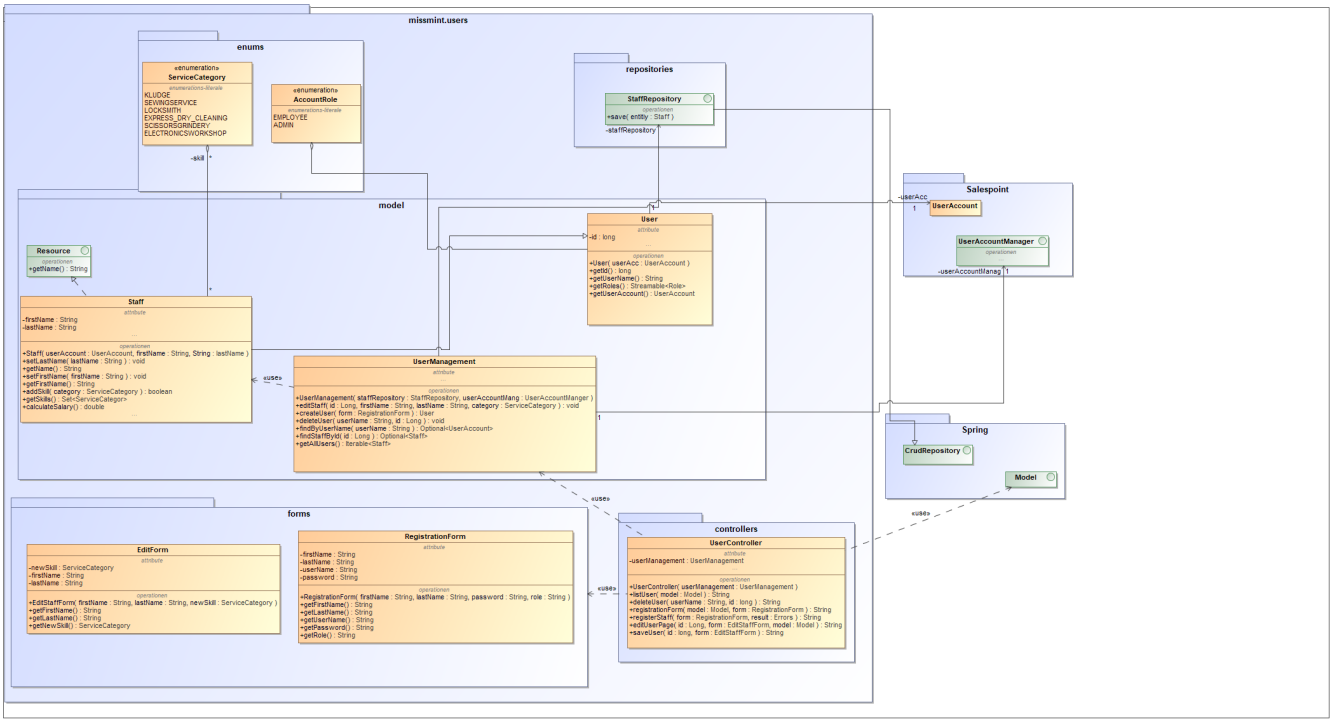


Figure 1. Klassendiagramm für das orders Paket

Klasse/Enumeration	Beschreibung
OrdersController	Ein Spring MVC Controller, der Anfragen für die Anzeige der aktuellen Aufträge `MissMintOrder` beantwortet.
OrderDTO	Diese Klasse dient nur dem Datentransfer zum `orders.html`-Template.
PickUpController	Ein Spring MVC Controller, der Anfragen für die Rückgabe von Gegenständen an den Kunden handhabt. Dazu berechnet die Klasse auch anfallende Kosten.

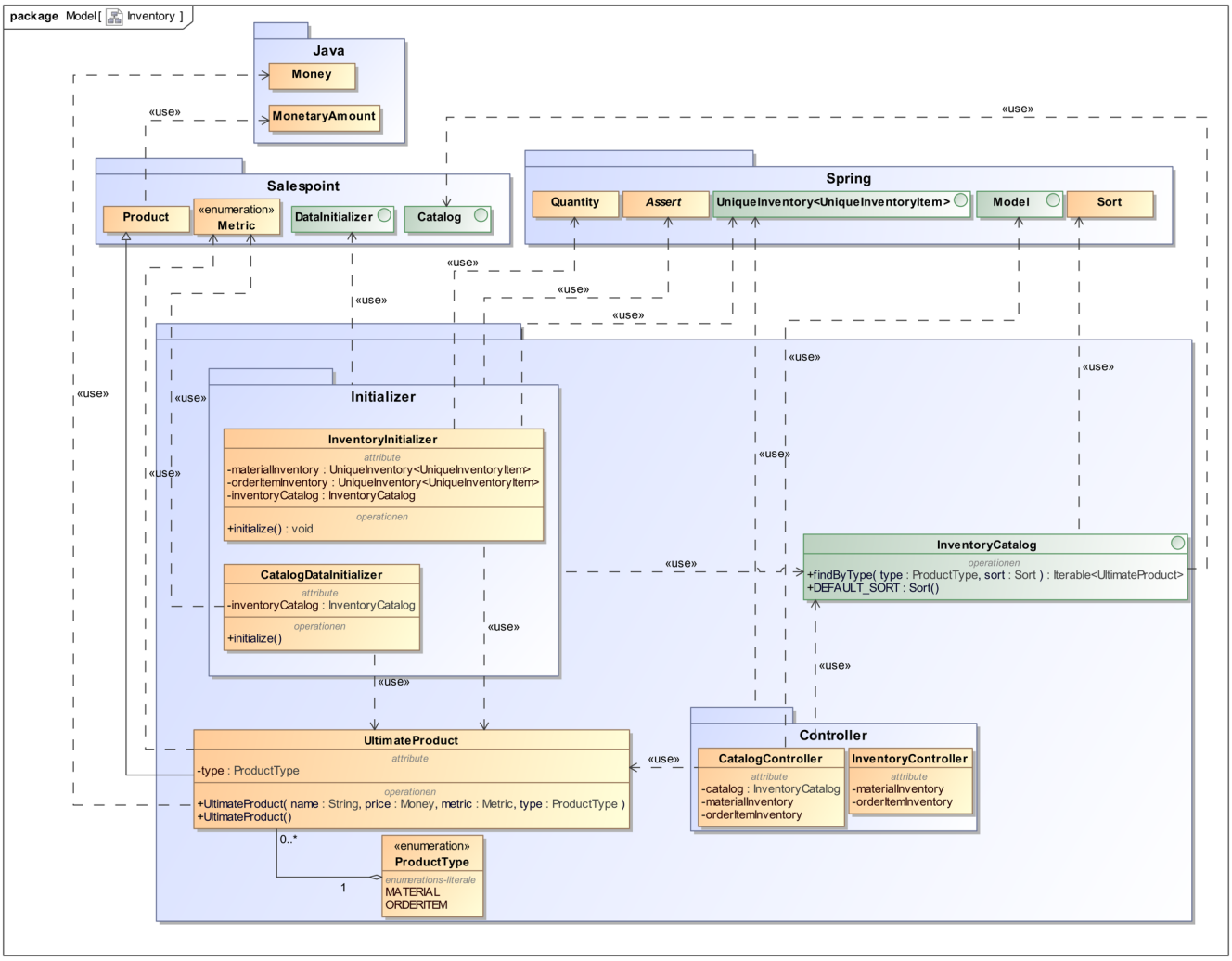
Klasse/Enumeration	Beschreibung
ReceivingController	Ein Spring MVC Controller, der Anfragen für die Aufnahme von Aufträgen `MissMintOrder` in das System beantwortet. Dazu berechnet die Klasse auch anfallende Kosten.
ReceivingForm	Eine Klasse, um die Mitarbeitereingaben für die Annahme zu validieren.
MissMintOrder	Diese Klasse ist eine Erweiterung der Salespoint `Order`. Sie enthält Kunden-, Dienstleistungen- und Zeit-Informationen. Zudem hat der Auftrag einen Zustand.
OrderDataInitializer	Eine Implementation des `DataInitializer`s, um einige vordefinierte Aufträge zum Testen anzulegen.
OrderService	Ein Dienst, der Hilfsfunktionen für die Auftragsverwaltung bereitstellt.
OrderState	Eine Enumeration für die Zustände der Aufträge.
Service	Erweiterung des Salespoint `Product`s, um es von anderen Produkten in der späteren Software abzugrenzen. Im Prototypen speichert es auch noch das Material.
ServiceDataInitializer	Eine Implementation des `DataInitializer`s, um die einzelnen Dienstleistungen anzulegen.

5.2. Mitarbeiter



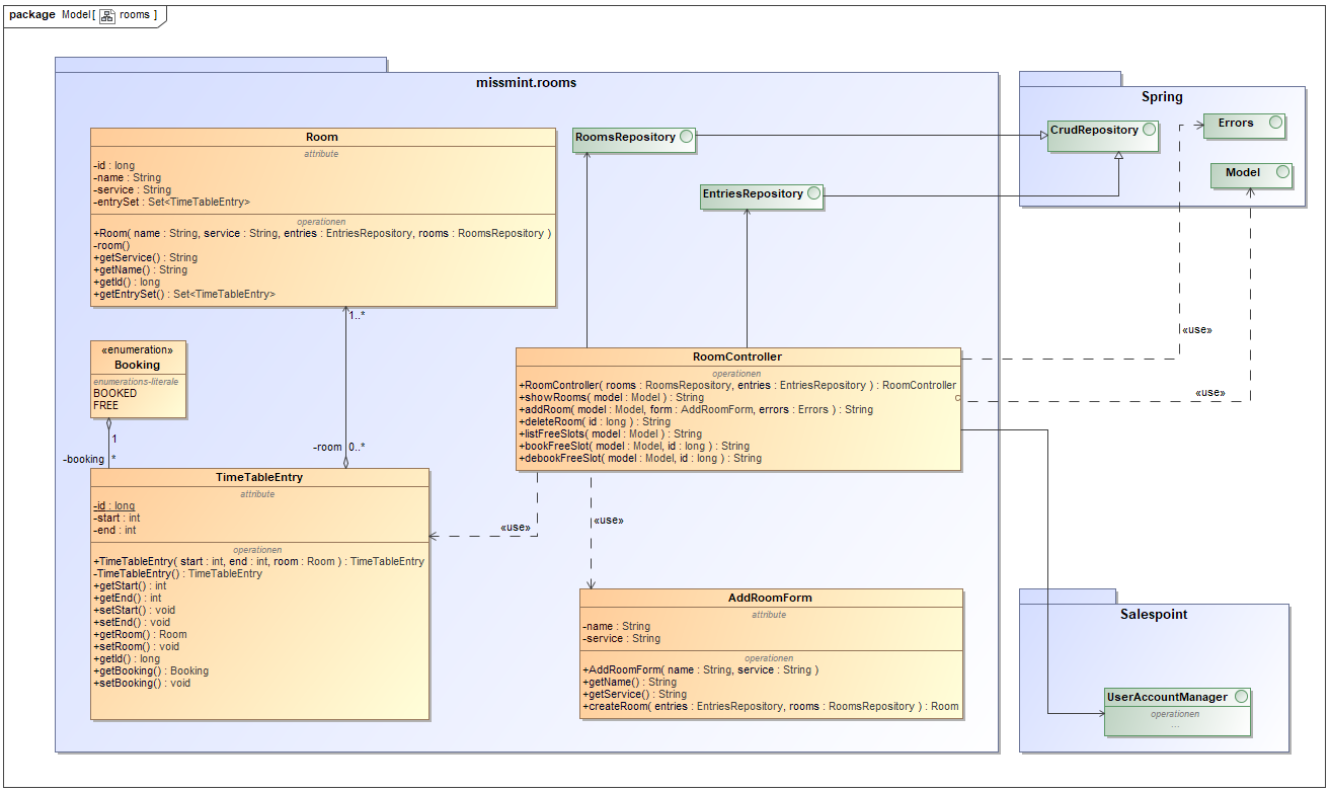
Klasse/Enumeration/Schnittstelle	Beschreibung
UserConstroller	Ein Spring MVC Controller, der Anfragen bzgl. Mitarbeiterübersicht und zudem Erstellung, Bearbeitung und Löschen von Mitarbeitern verarbeitet.
User	Jeder User hat eine eindeutige `Id`, einen `UserAccount` von *SalesPoint* und eine `UserAccountRole`.
Staff	Diese Klasse erweitert die `User` Klasse und implementiert das `Resource Interface`. Der Mitarbeiter hat Vor- und Nachname. Er kann Fertigkeiten in einer oder mehreren Dienstleistungen haben.
UserManagement	Ein Dienst, der zuständig ist, die Mitarbeiter im System zu verwalten.
StaffRepository	Die Schnittstelle erweitert die `CrudRepository` von Spring. Sie ist verantwortlich für die Persistenz der Mitarbeiterdaten.
AccountRole	Eine Enumeration um Mitarbeiter verschiedene Nutzerrechte zu geben.
ServiceCategory	Die Enumeration stellt die verschiedenen Dienstleistungen von *Miss Mint*-Betrieben dar.
RegistrationForm	Ein Formular für die Registrierung von neuen Mitarbeitern.
EditStaffForm	Ein Formular um Mitarbeiterdaten zu bearbeiten.

5.3. Inventar



Klasse/Enumeration/Schnittstelle	Beschreibung
CatalogController	Ein Spring MVC Controller, der die zu den Bestellungen gehörenden Gegenstände anzeigt.
CatalogDataInitializer	Implementation des `DataInitializer`s für den Produktkatalog
InventoryCatalog	Erweiterung des Salespoint`Catalog`s der Sortierung nach `ProductType` ermöglicht.
InventoryController	Ein Spring MVC Controller, der Anfragen bezüglich des Materialinventars annimmt.
InventoryInitializer	Implementation des `DataInitializer`s für die Inventare der jeweiligen Produkte.
UltimateProduct	Erweiterung des Salespoint `Product`s , mit zusätzlichen Eigenschaften um Filtern nach eigenen Attributen zu ermöglichen.

5.4. Räume



Klasse/Enumeration/Schnittstelle	Beschreibung
Room	Klasse die die Räume repräsentiert. Räume stehen in enger Verbindung mit den Timeslots, wobei jeder Raum seine eigenen TimeSlots besitzt.
Booking	Enumeration für die verschiedenen Buchungsfälle der Timeslots.
TimeTableEntry	Diese Klasse repräsentiert Timeslots, die durch Buchung von Räumen in einer bestimmten Zeit zu vollwertigen Einträgen werden können.
RoomController	Ein Spring MVC Controller, der die Verwaltung von Räumen und das buchen von TimeSlots ermöglicht.
AddRoomForm	Diese Klasse prüft die Nutzereingabe auf fehler und erstellt ggf. einen neuen Raum.
RoomsRepository	Interface welches das Crud Repository von Spring erbt. Es speichert Raumeinträge.
EntriesRepository	Interface welches das Crud Repository von Spring erbt. Es speichert Einträge in TimeSlots.

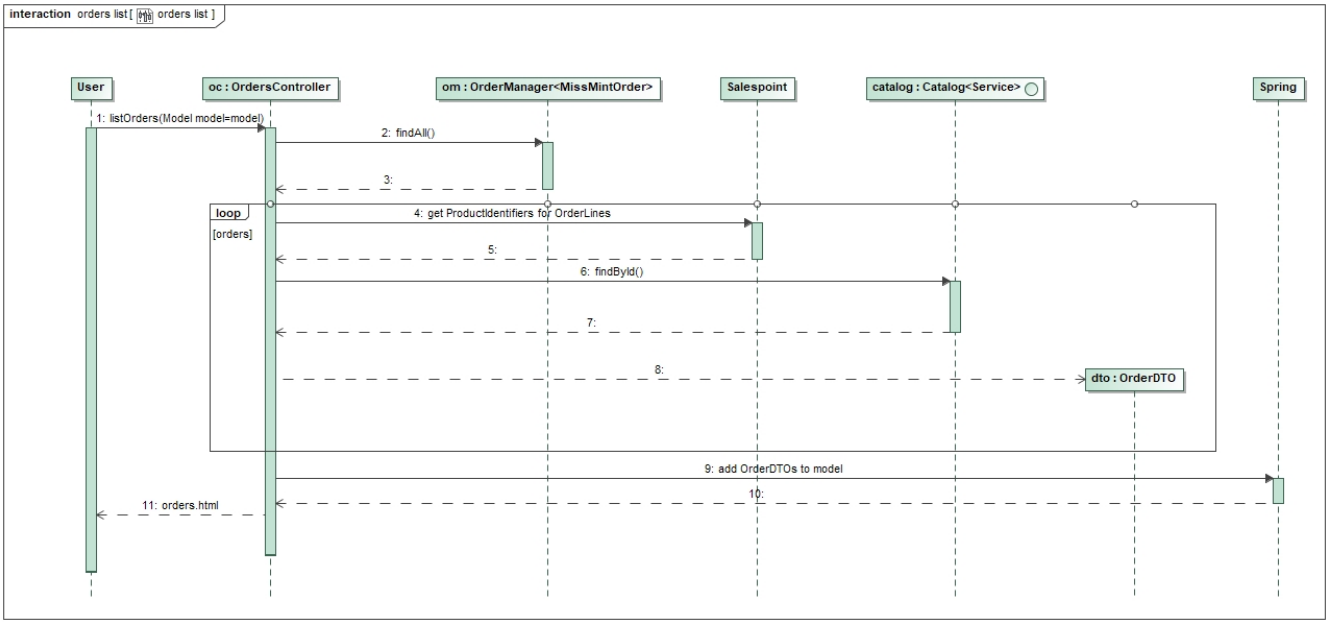
5.5. Rückverfolgbarkeit zwischen Analyse- und Entwurfsmodell

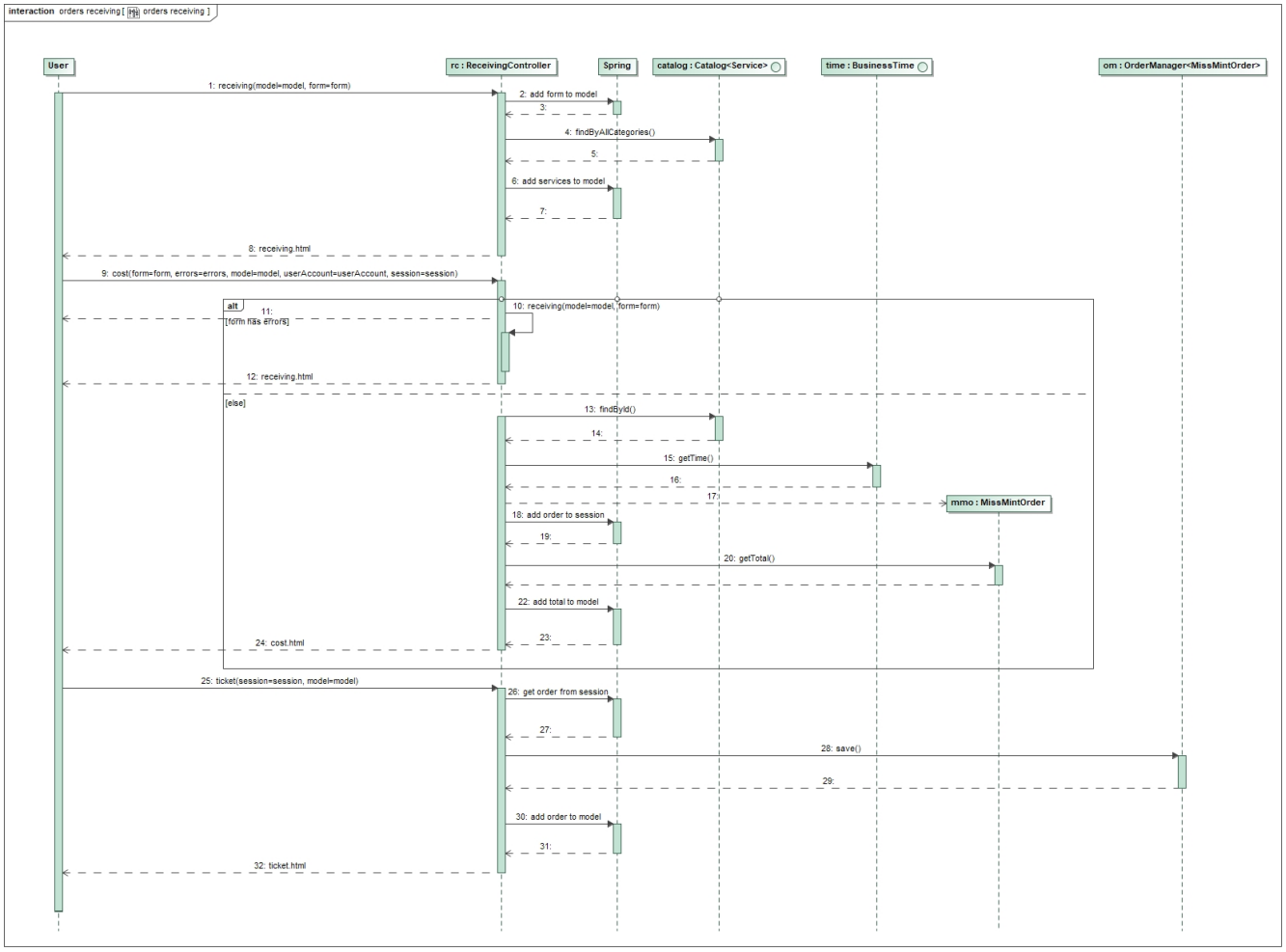
Klasse/Enumeration (Analysemodell)	Klasse/Enumeration (Entwurfsmodell)
Order	Order
Service	Service
State	OrderState

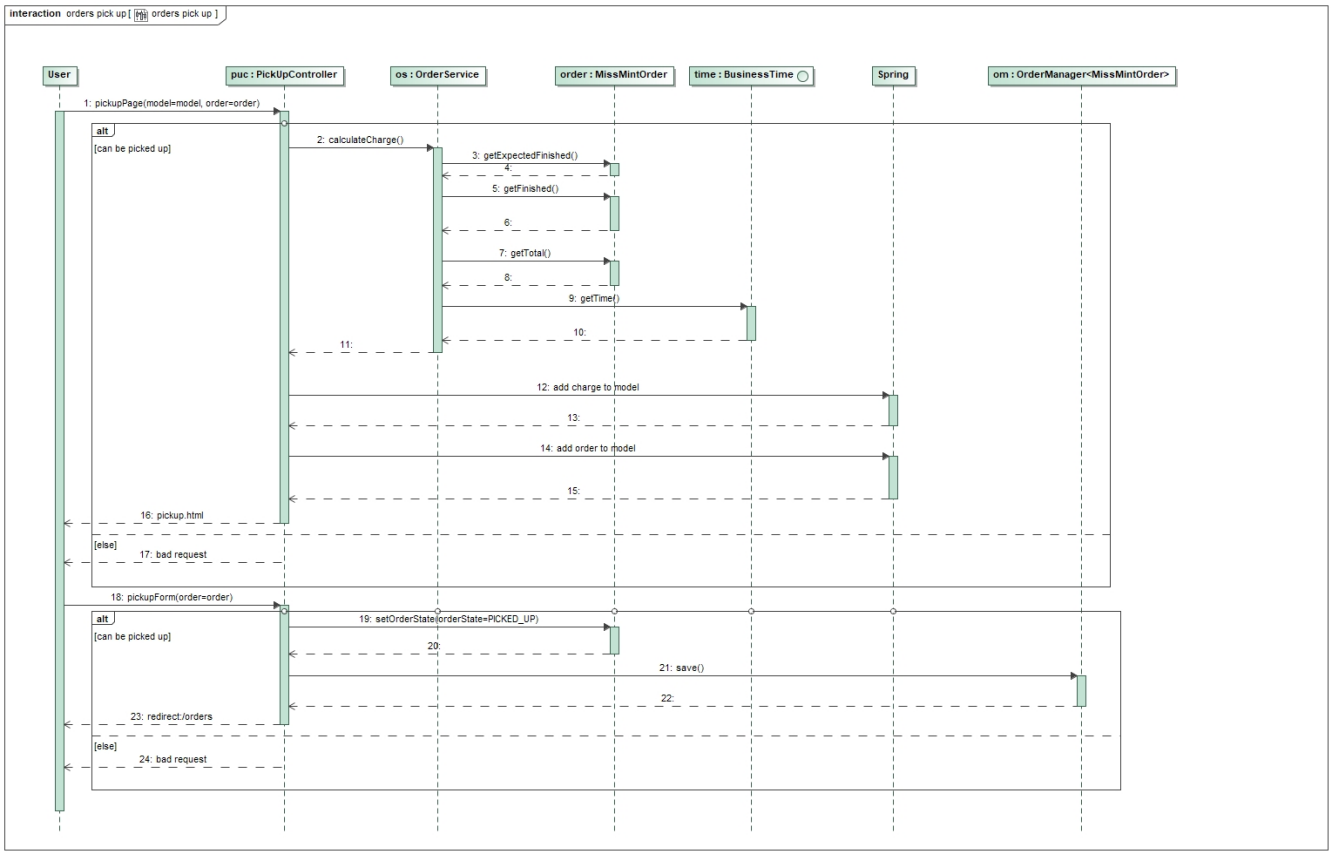
Klasse/Enumeration (Analysemodell)	Klasse/Enumeration (Entwurfsmodell)
Order Management	OrderManager
Service Category	String
OrderTicket	-
User	User
Staff	Staff
-	UserManagement
Storage	Inventory
Material	Inventory
ResourceManagement	-
Resource	UltimateProduct
Room	Room
TimeTable	-
-	Booking
TimeTableEntry	TimeTableEntry

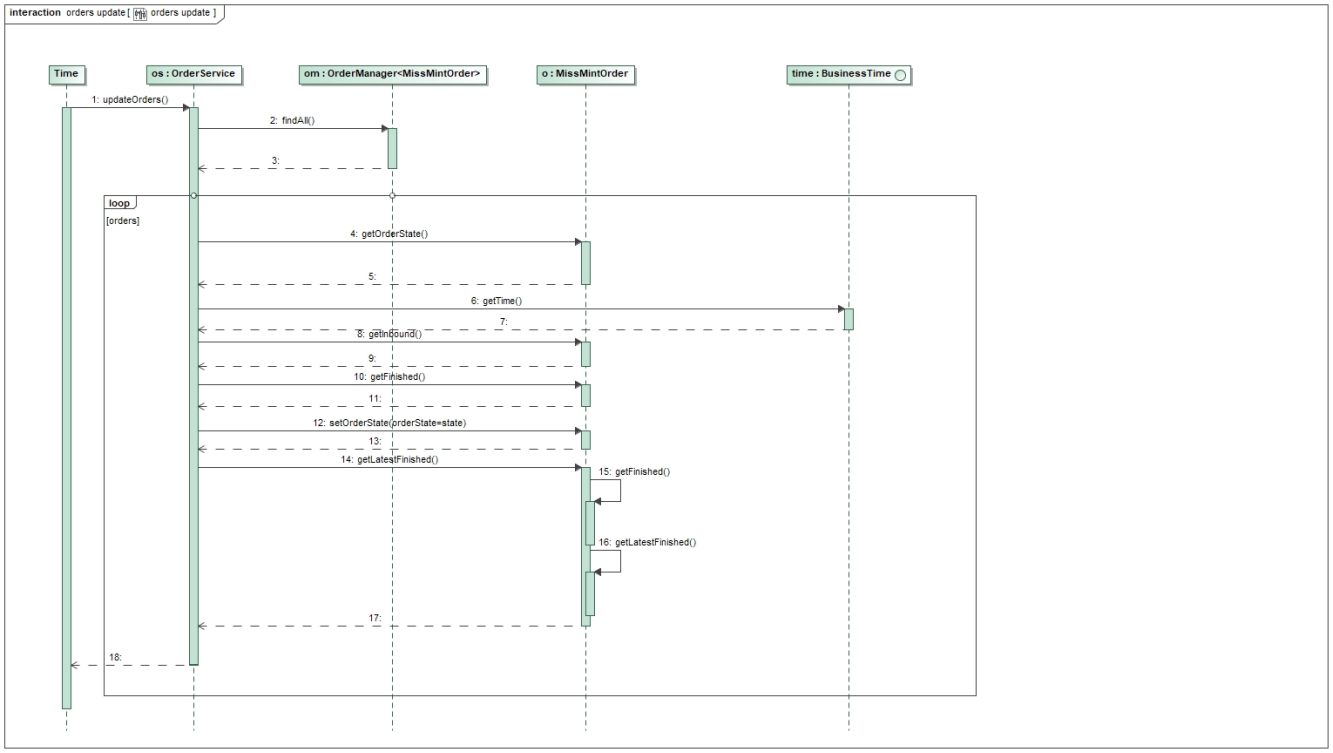
6. Laufzeitsicht

6.1. Aufträge

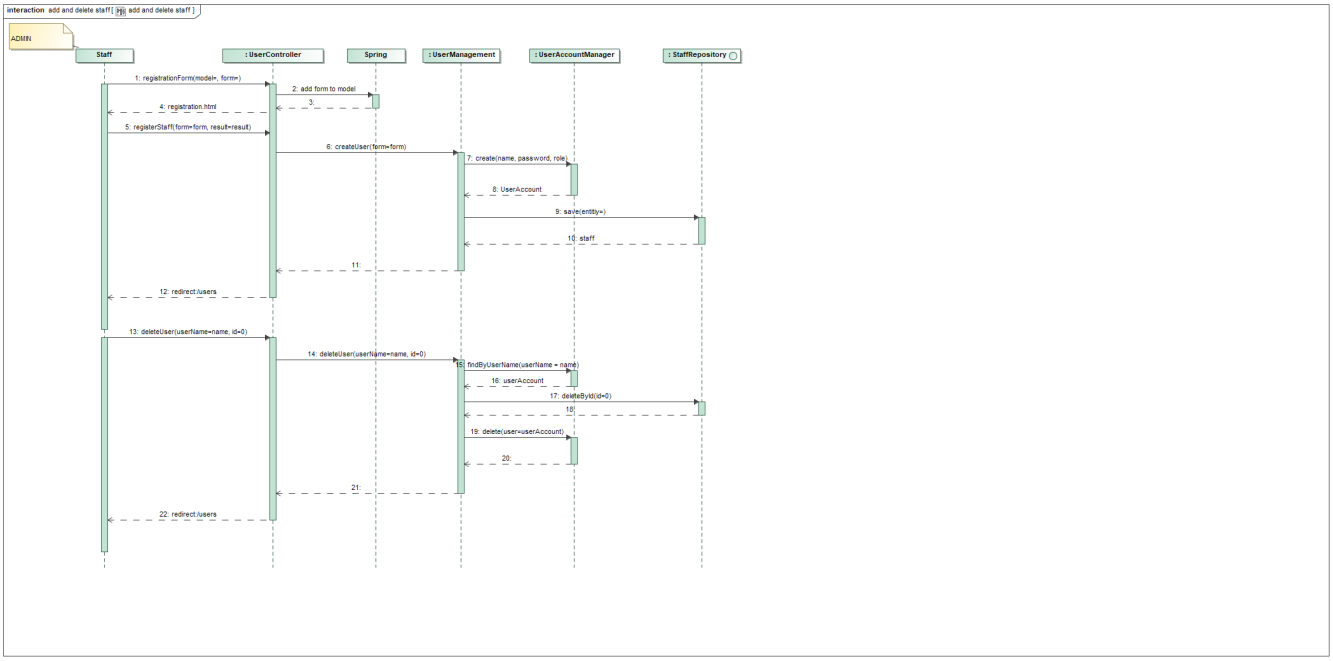



interaction orders receiving [orders receiving]
User
rc : ReceivingController
Spring
catalog : Catalog<Service>
time : BusinessTime
om : OrderManager<MissMintOrder>
1: receiving(model=model, form=form)
2: add form to model
3:
4: findByAllCategories()
5:
6: add services to model
7:
8: receiving.html
9: cost(form=form, errors=errors, model=model, userAccount=userAccount, session=session)
alt
[form has errors]
11:
10: receiving(model=model, form=form)
12: receiving.html
[else]
13: findById()
14:
15: getTime()
16:
17:
mmo : MissMintOrder
18: add order to session
19:
20: getTotal()
22: add total to model
23:
24: cost.html
25: ticket(session=session, model=model)
26: get order from session
27:
28: save()
29:
30: add order to model
31:
32: ticket.html

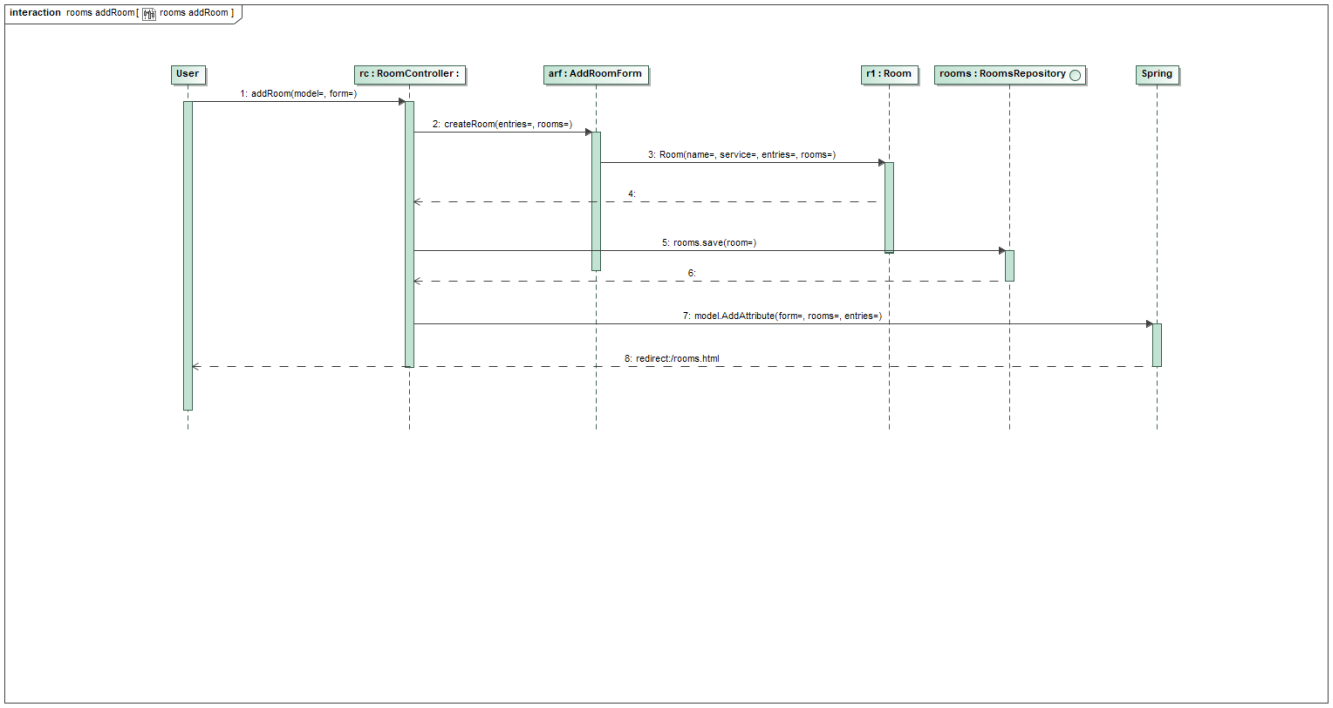

interaction orders pick up [orders pick up]
User
puc : PickUpController
os : OrderService
order : MissMintOrder
time : BusinessTime
Spring
om : OrderManager<MissMintOrder>
1: pickupPage(model=model, order=order)
alt
[can be picked up]
2: calculateCharge()
3: getExpectedFinished()
4:
5: getFinished()
6:
7: getTotal()
8:
9: getTime()
10:
11:
12: add charge to model
13:
14: add order to model
15:
16: pickup.html
[else]
17: bad request
18: pickupForm(order=order)
alt
[can be picked up]
19: setOrderState(orderState=PICKED_UP)
20:
21: save()
22:
23: redirect:/orders
[else]
24: bad request

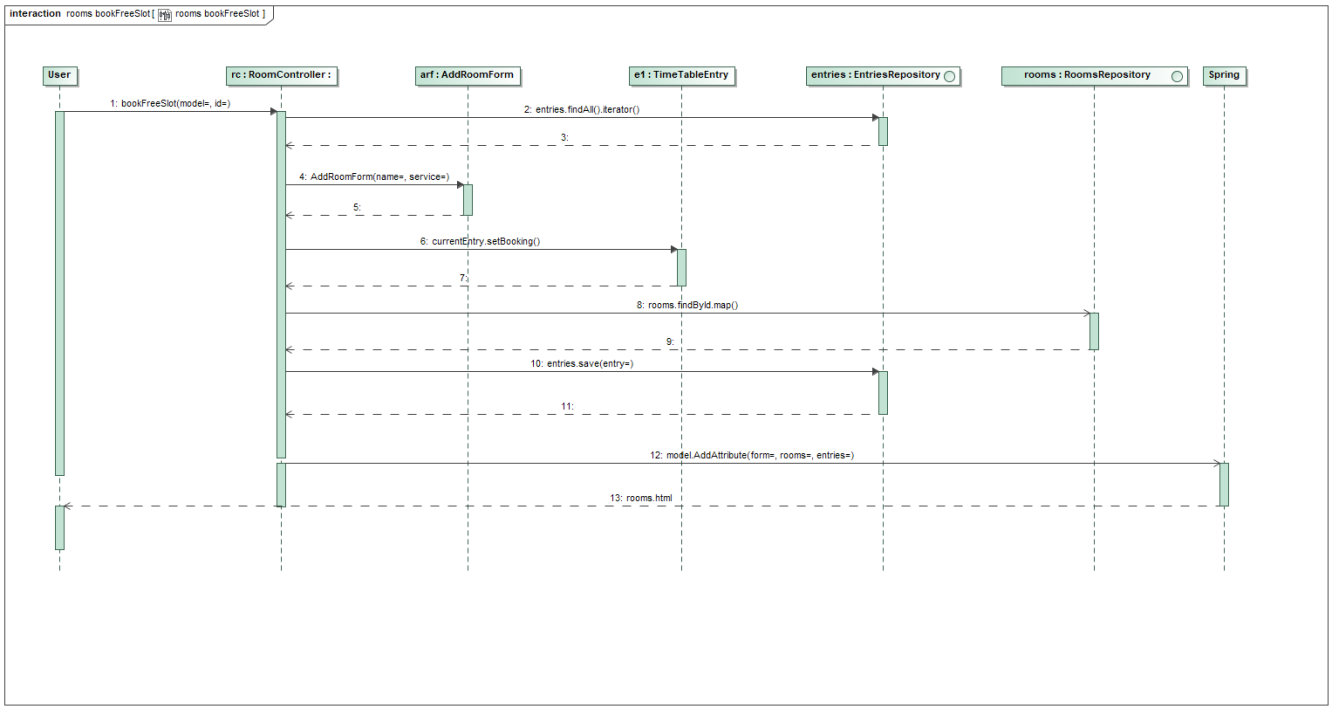


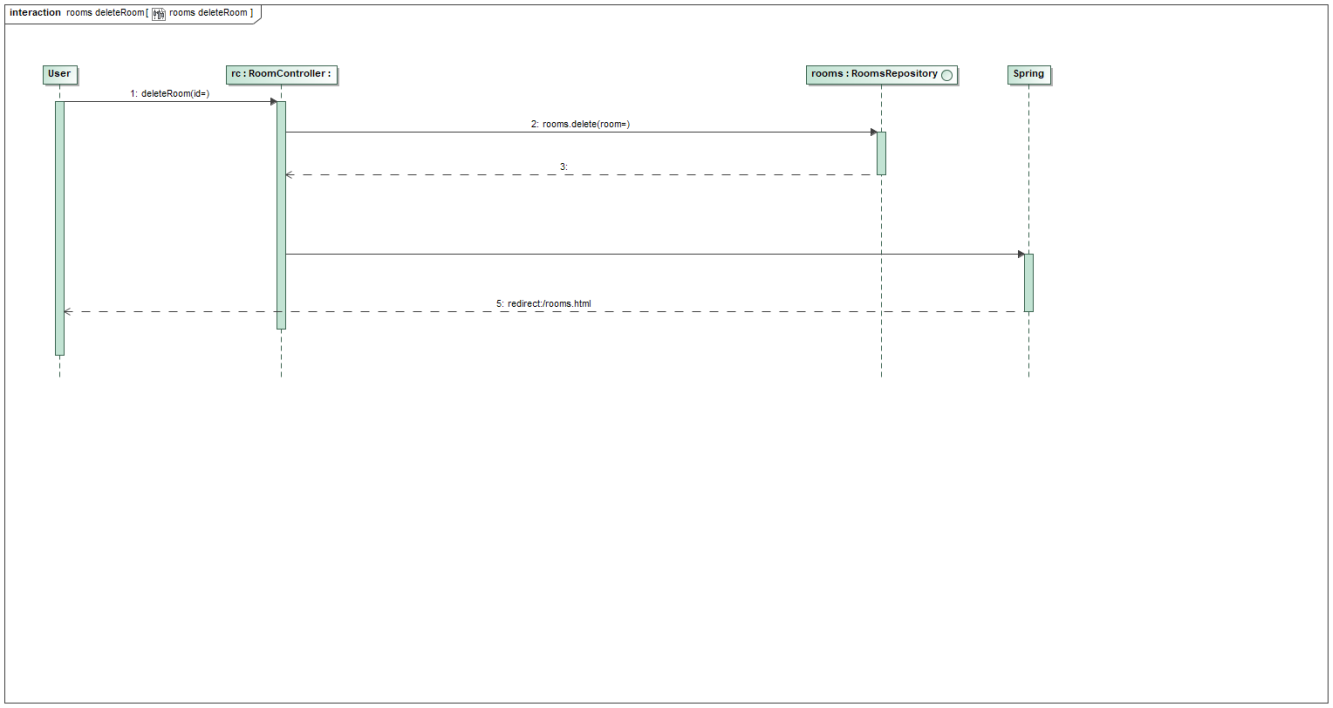
6.2. Mitarbeiter

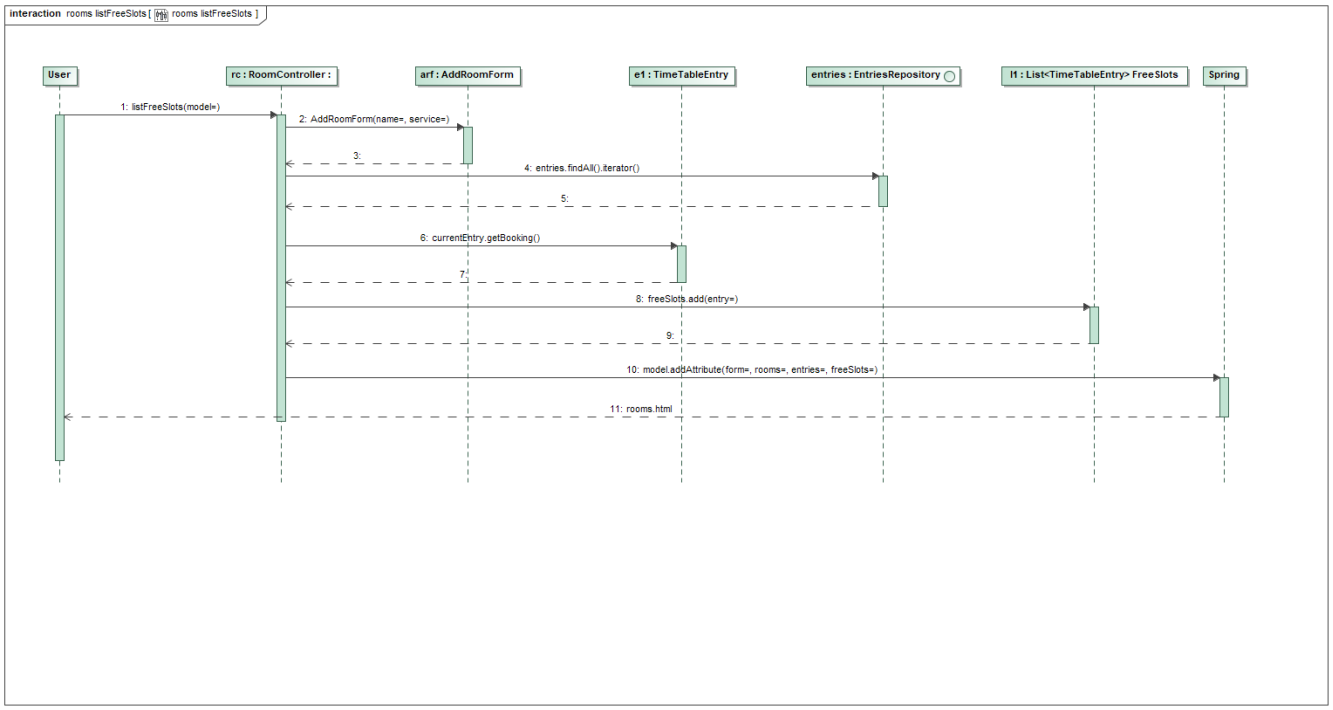


6.3. Räume

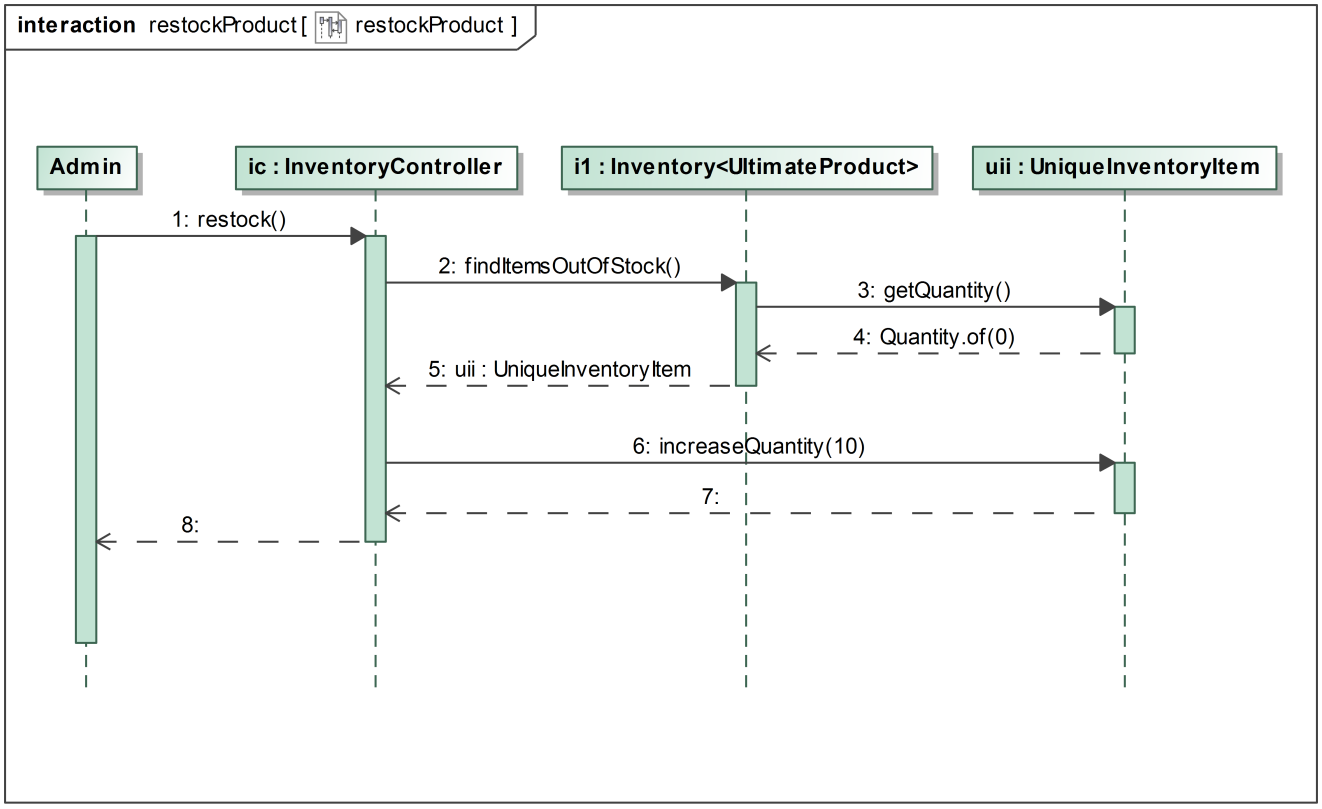

interaction rooms addRoom [rooms addRoom]
User
rc : RoomController :
arf : AddRoomForm
r1 : Room
rooms : RoomsRepository
Spring
1: addRoom(model=, form=)
2: createRoom(entries=, rooms=)
3: Room(name=, service=, entries=, rooms=)
4:
5: rooms.save(room=)
6:
7: model.AddAttribute(form=, rooms=, entries=)
8: redirect:/rooms.html


interaction rooms bookFreeSlot [rooms bookFreeSlot]
User
rc : RoomController :
arf : AddRoomForm
e1 : TimeTableEntry
entries : EntriesRepository
rooms : RoomsRepository
Spring
1: bookFreeSlot(model=, id=)
2: entries.findAll().iterator()
3:
4: AddRoomForm(name=, service=)
5:
6: currentEntry.setBooking()
7:
8: rooms.findById.map()
9:
10: entries.save(entry=)
11:
12: model.AddAttribute(form=, rooms=, entries=)
13: rooms.html


interaction rooms deleteRoom [rooms deleteRoom]
User
rc : RoomController :
rooms : RoomsRepository
Spring
1: deleteRoom(id=)
2: rooms.delete(room=)
3:
5: redirect:/rooms.html


interaction rooms listFreeSlots [rooms listFreeSlots]
User
rc : RoomController :
arf : AddRoomForm
e1 : TimeTableEntry
entries : EntriesRepository
l1 : List<TimeTableEntry> FreeSlots
Spring
1: listFreeSlots(model=)
2: AddRoomForm(name=, service=)
3:
4: entries.findAll().iterator()
5:
6: currentEntry.getBooking()
7:
8: freeSlots.add(entry=)
9:
10: model.addAttribute(form=, rooms=, entries=, freeSlots=)
11: rooms.html

6.4. Inventar


interaction restockProduct [restockProduct]
Admin
ic : InventoryController
i1 : Inventory<UltimateProduct>
uii : UniqueInventoryItem
1: restock()
2: findItemsOutOfStock()
3: getQuantity()
4: Quantity.of(0)
5: uii : UniqueInventoryItem
6: increaseQuantity(10)
7:
8: