

Blumenladen

SWP WS2024/25
Gruppe 07

Oleh Khmelyk
Andrejs Borodajevs
Vadym Zaitsev
Valeriia Sarkisian
Davyd Okaianchenko
Bechir Sayadi
Paul-Valentin Vela



Einführung

Derzeit ist die manuelle Verwaltung von Beständen, Bestellungen und Abrechnungen im Blumenladen zeitaufwändig, fehleranfällig und unübersichtlich.

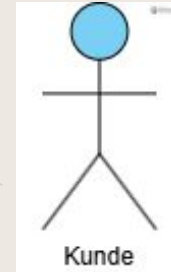
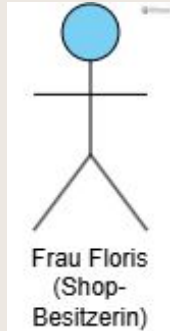
Wichtige Prozesse wie die Buchung von Hochzeiten, die Pflege von Serviceverträgen und die Dokumentation von Verlusten sind ineffizient organisiert.

Es fehlt an einer zentralen, digitalen Lösung, die den Überblick über Angebote, Verkäufe und Finanzen gewährleistet und Routineaufgaben automatisiert.

Lösung: Wir haben eine Software entwickelt, die all diese Probleme löst und die Verwaltung des Ladens, sämtlicher Transaktionen sowie die Kommunikation mit Kunden und vieles mehr erheblich erleichtert.

Akteure

Personen, die direkt bzw. indirekt mit dem Software interagieren



Gebiete

Logische Teile des Programms




Vertrieb

Lager

Finanzen

Services

Kalender



MVC Pattern

Request

Controller

Model

`http://localhost:8080/sell`

SalesController

SalesService

View

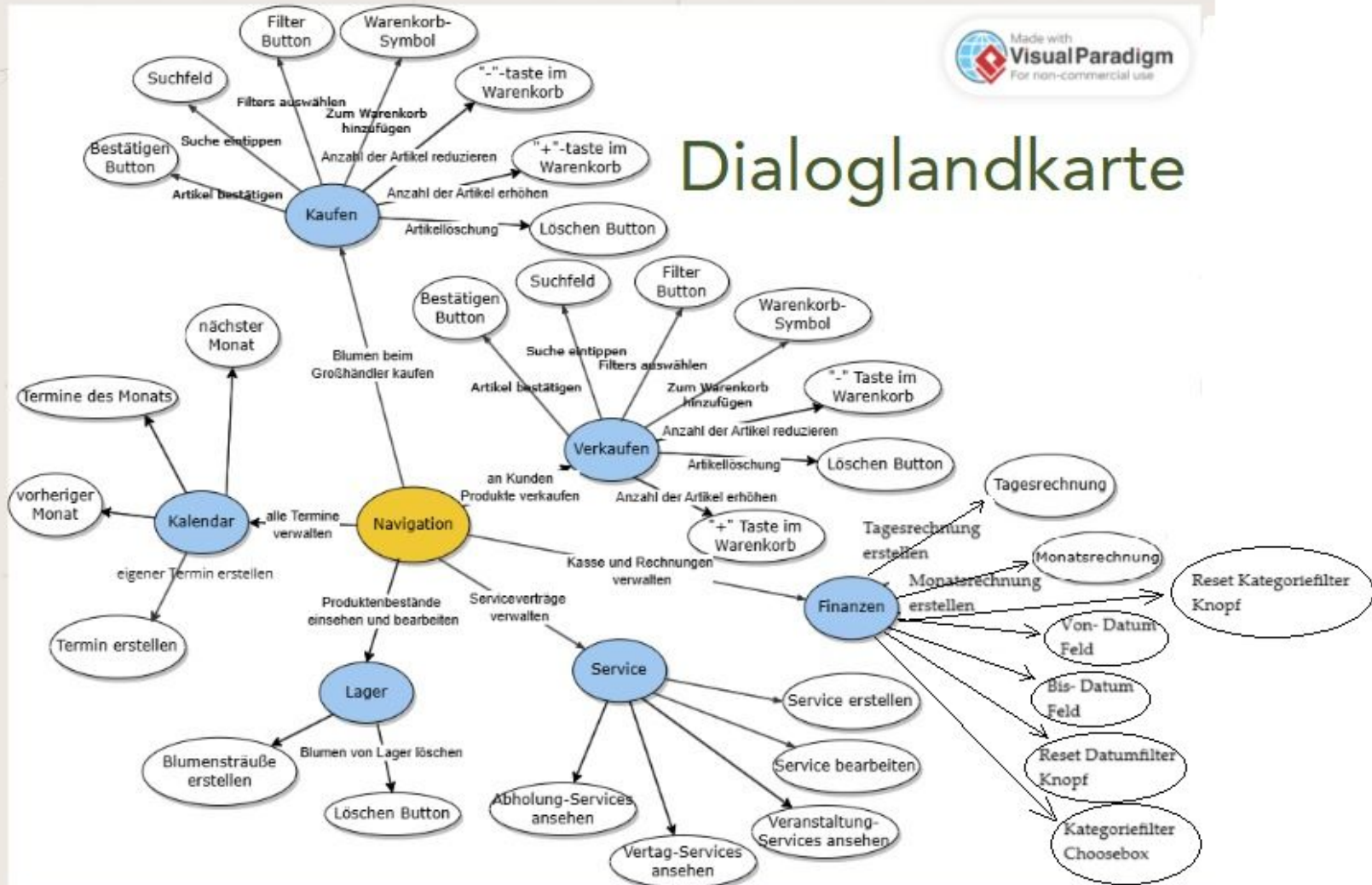
Verkäufe

Icon	Name	Farbe	Anzahl	Preis	
🌸	Rose	Red	21	EUR 20	🛒
🌸	Sunflower	Yellow	10	EUR 5	🛒
🌸	Lily	White	10	EUR 18	🛒
🌸	Lily2	White	30	EUR 19	🛒
🌸	Lily3	Purple	73	EUR 20	🛒
🌸	Rose and Lily Bouquet	Bouquet	5	EUR 101	🛒
🌸	Perfect Spring Bouquet	Bouquet	2	EUR 173	🛒

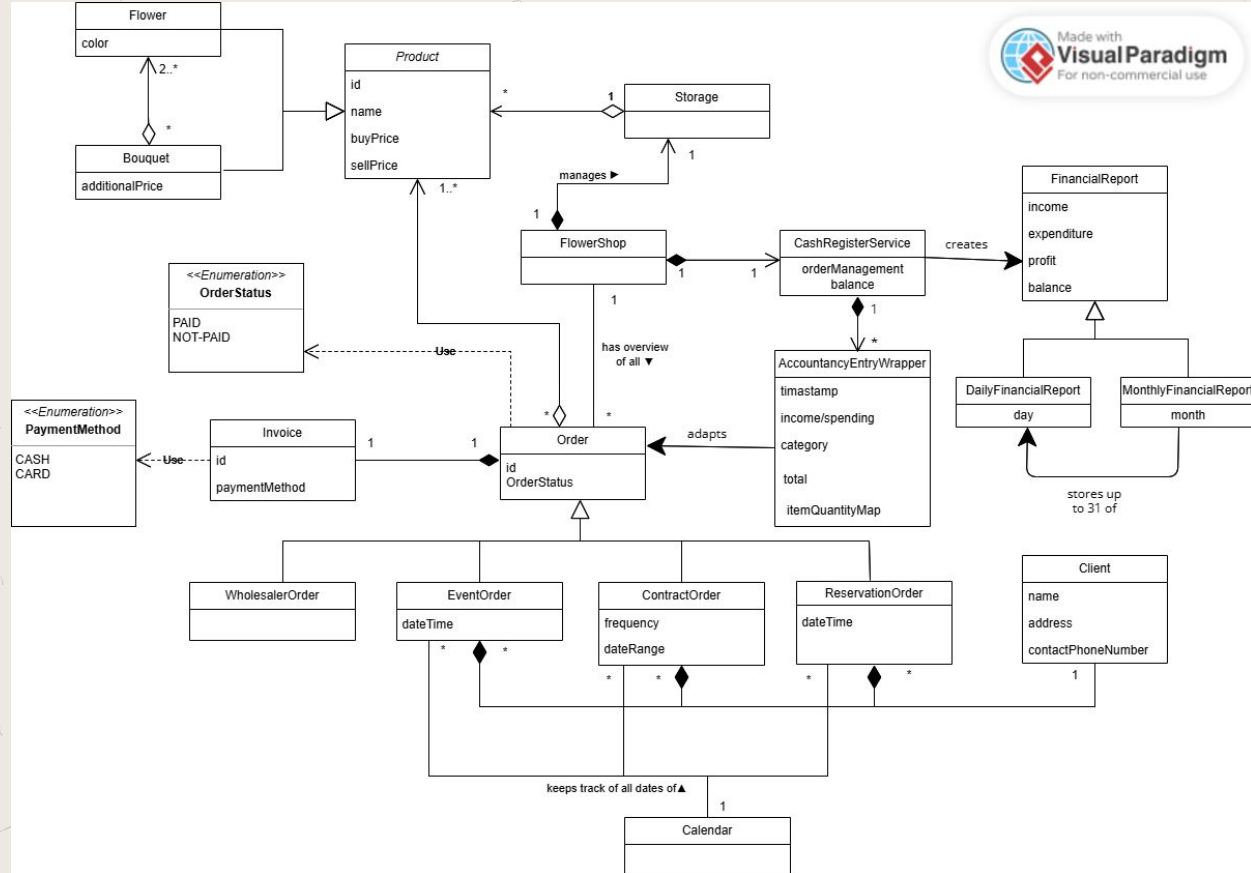
Verkaufen
Kaufen
Lager
Dienstleistungen
Finanzen
Kalender

Modify order
🛒 Your basket

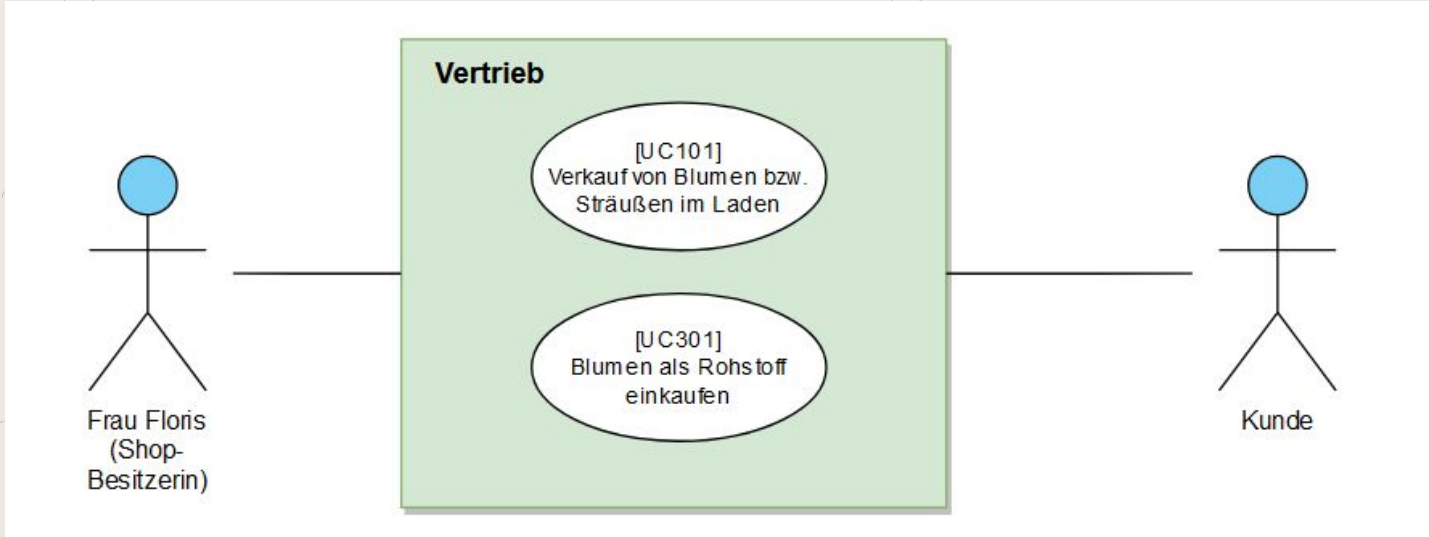
Dialoglandkarte



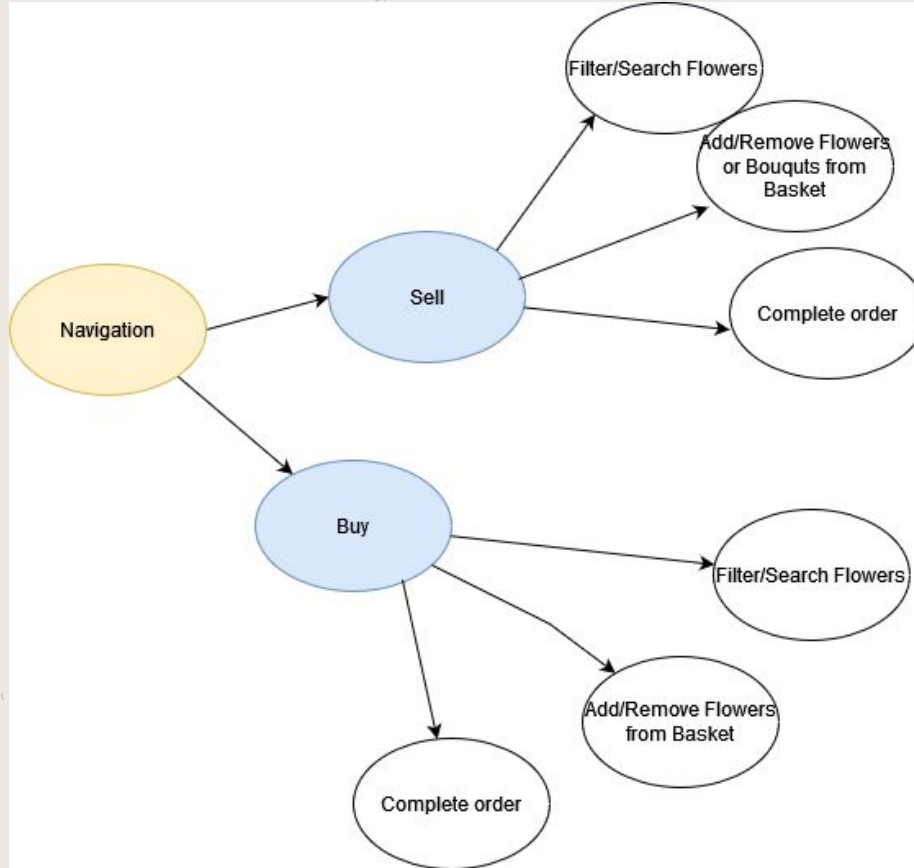
Domain Model



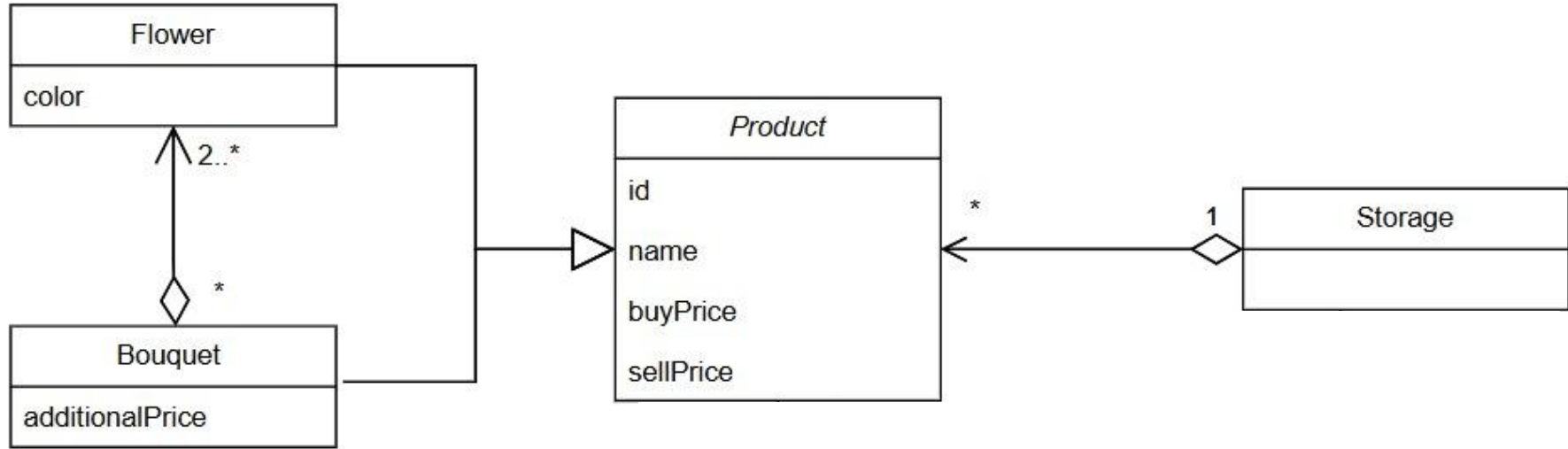
Vertrieb



Vertrieb

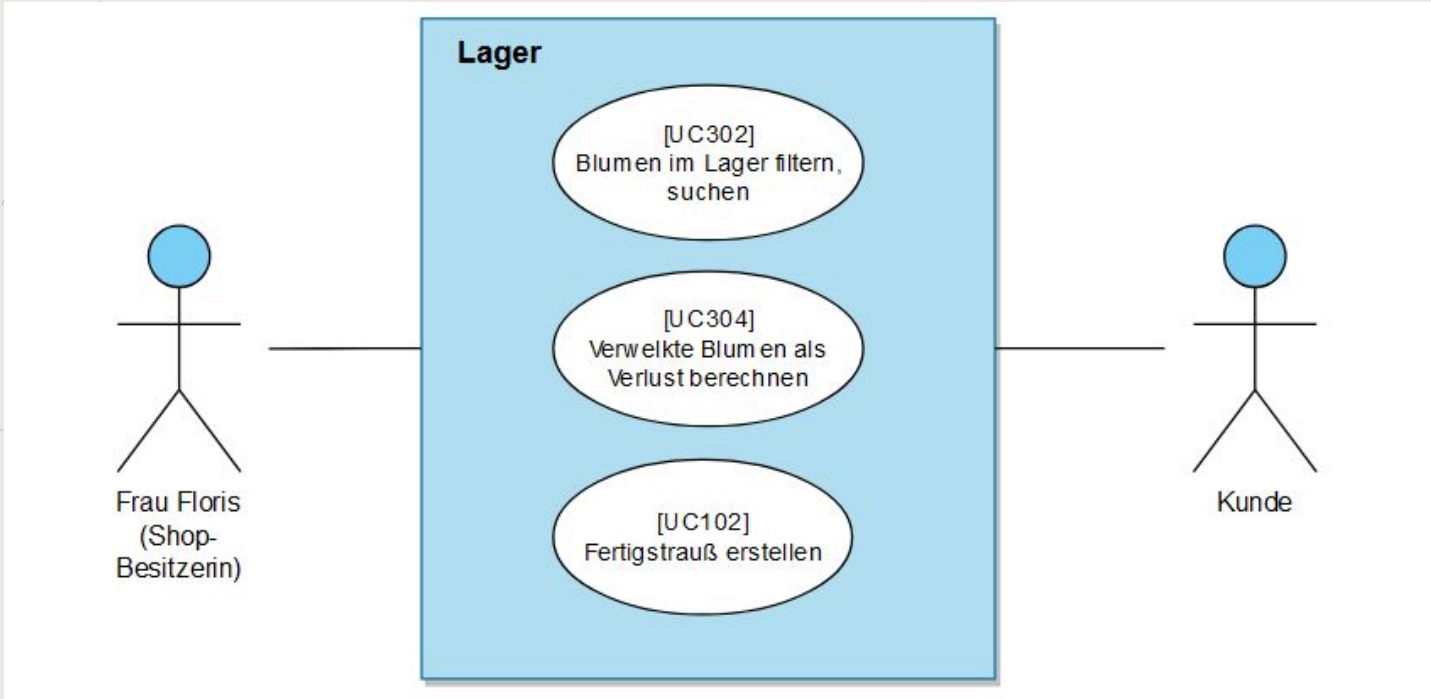


Vertrieb

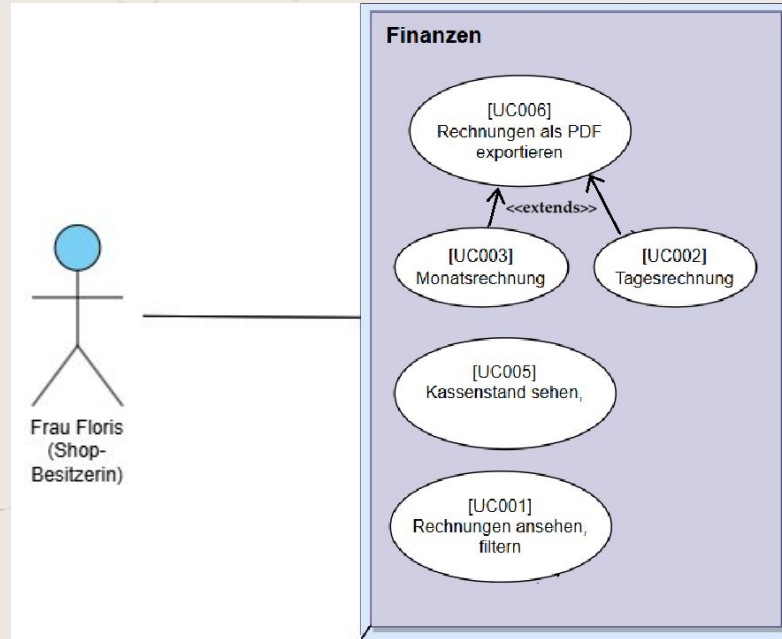


Domain model

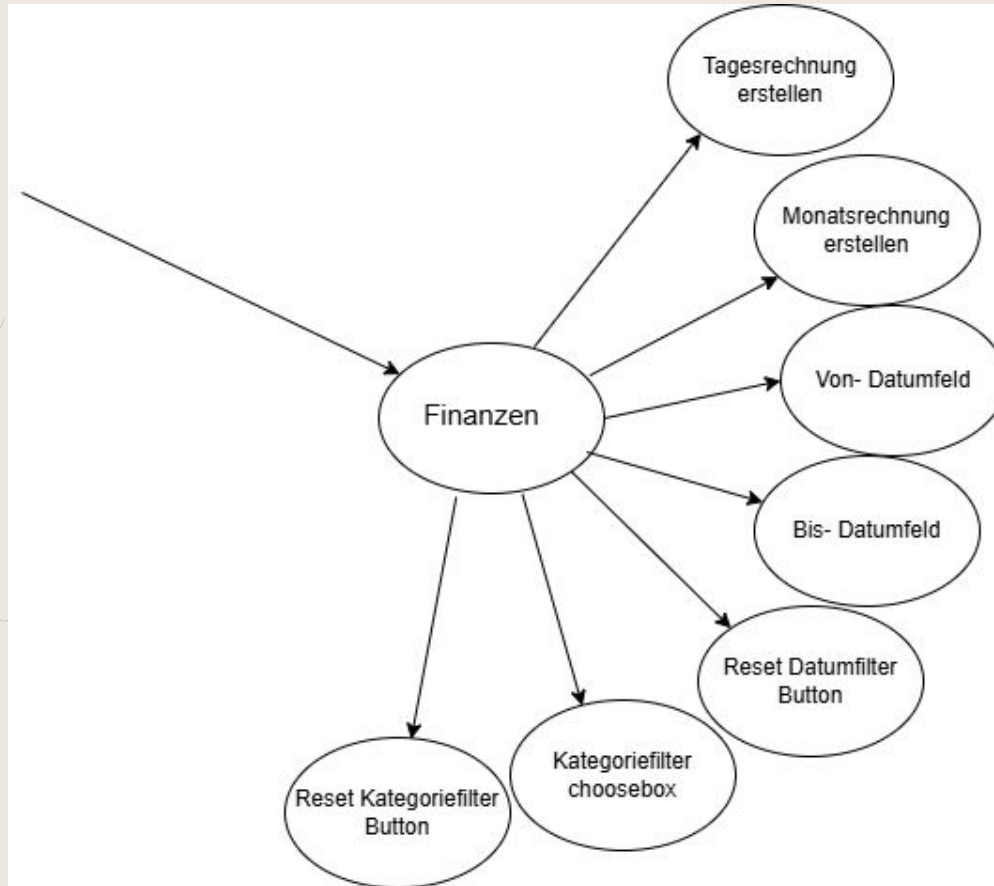
Lager



Finanzen

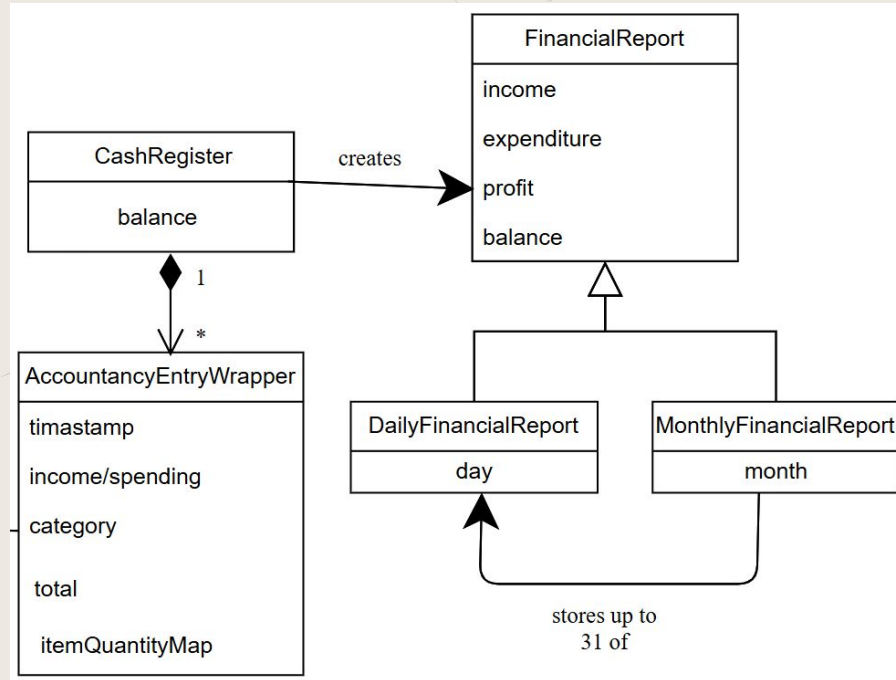


Finanzen

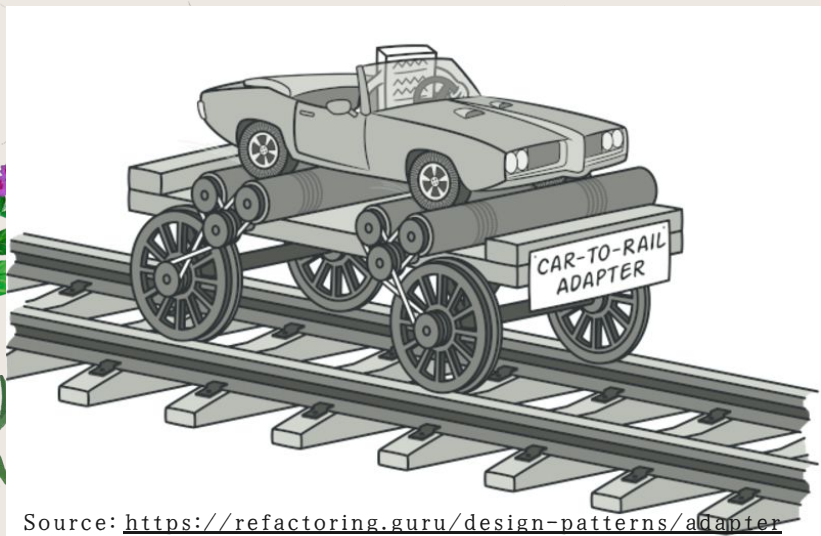


Finanzen

Domain Model



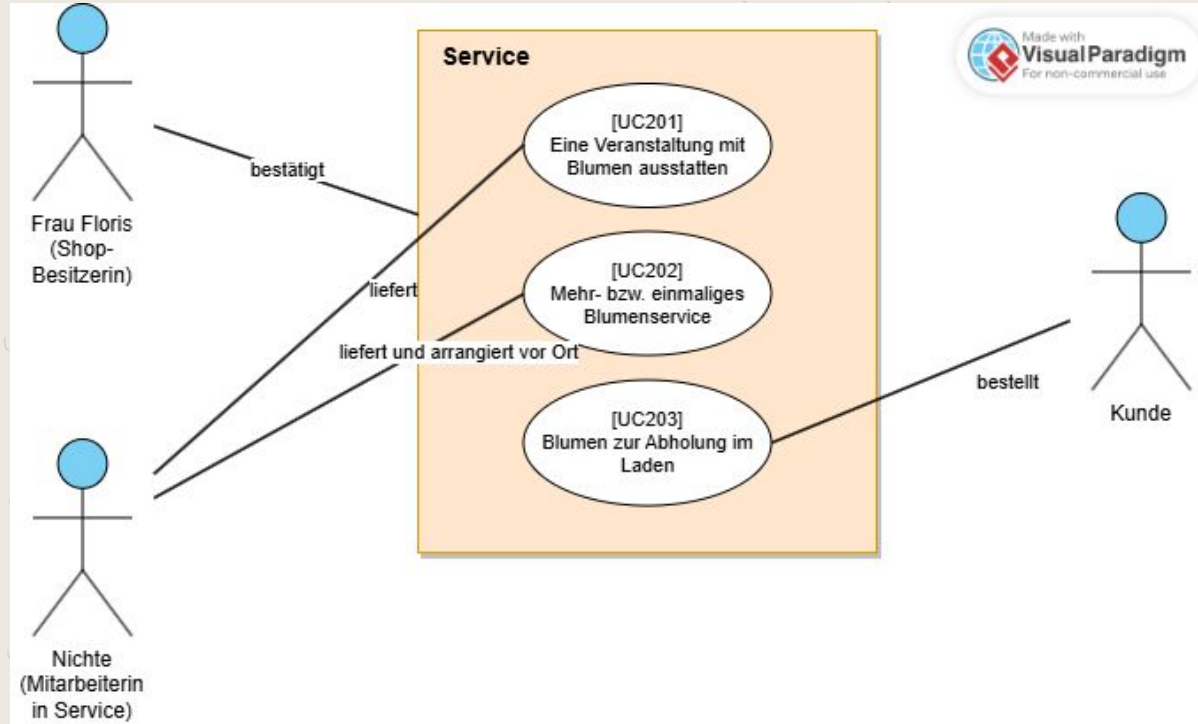
Adapter Pattern



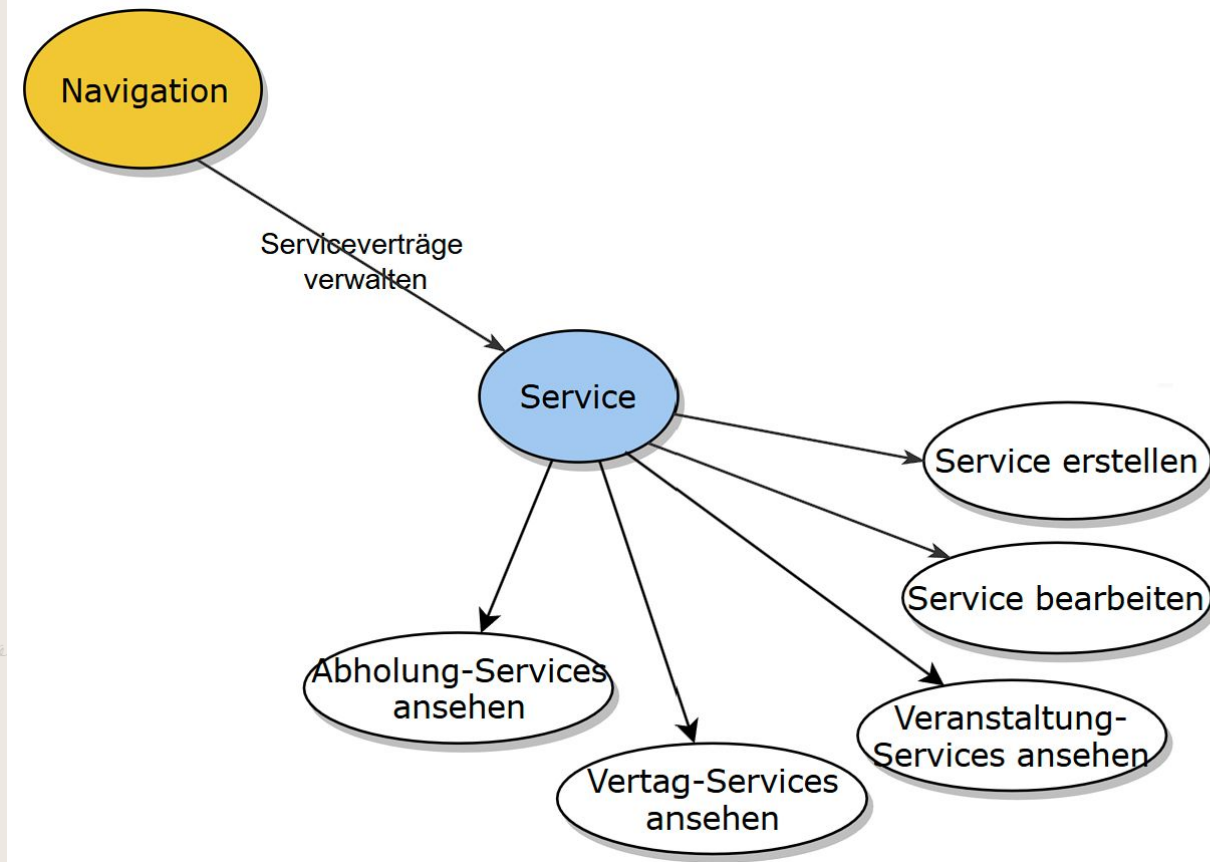
Source: <https://refactoring.guru/design-patterns/adapter>

```
@EventListener  Vadym Zaitsev
public void onOrderPaid(OrderEvents.OrderPaid event){
    AbstractOrder order = (AbstractOrder) event.getOrder();
    //convert order to AccountancyEntry
    AccountancyEntryWrapper convertedOrder = new AccountancyEntryWrapper(order);
    this.add(convertedOrder);
}
```


Service

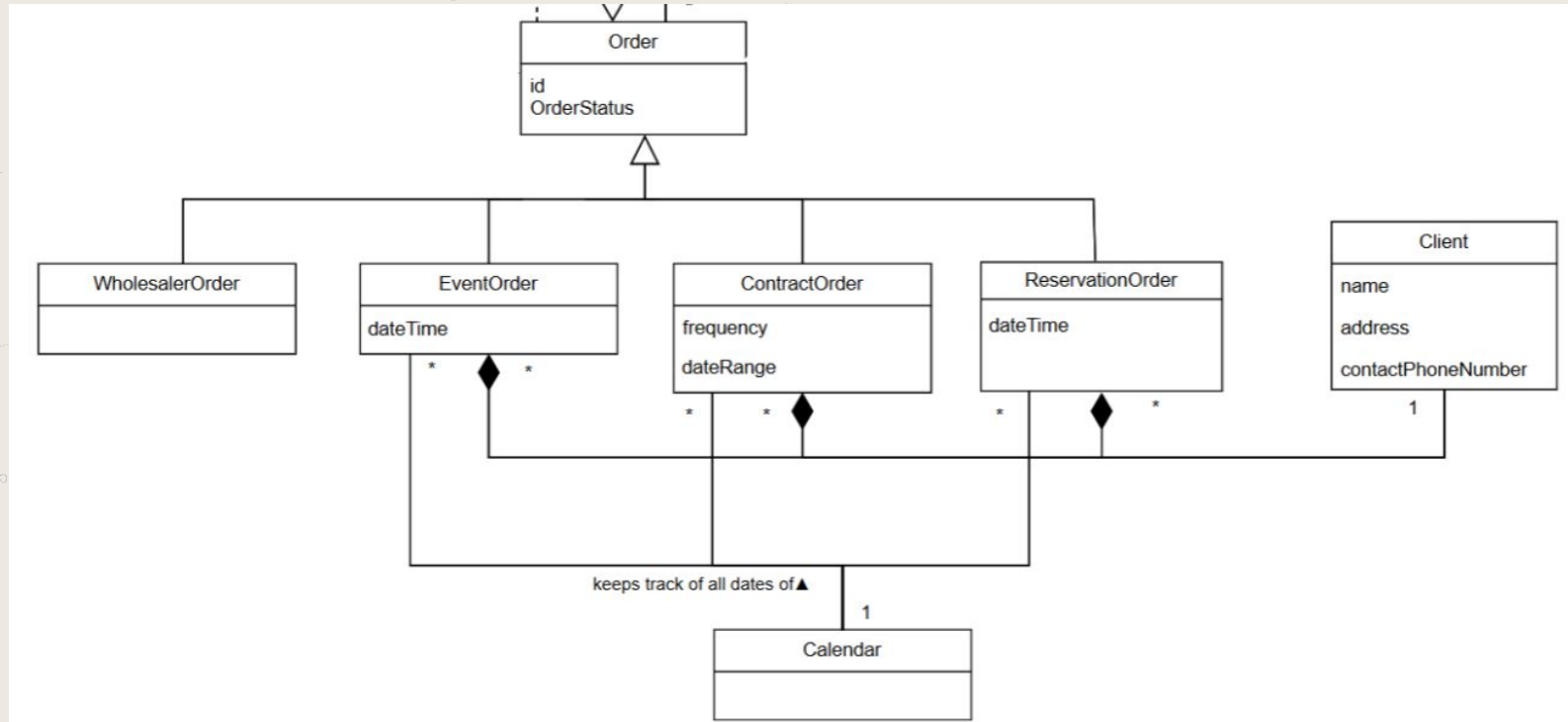


Service

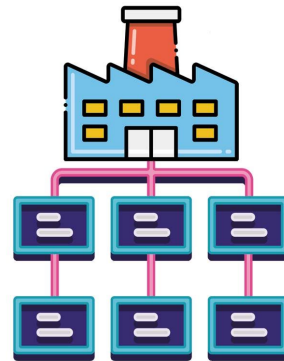


Service

Domain Model



Factory Pattern

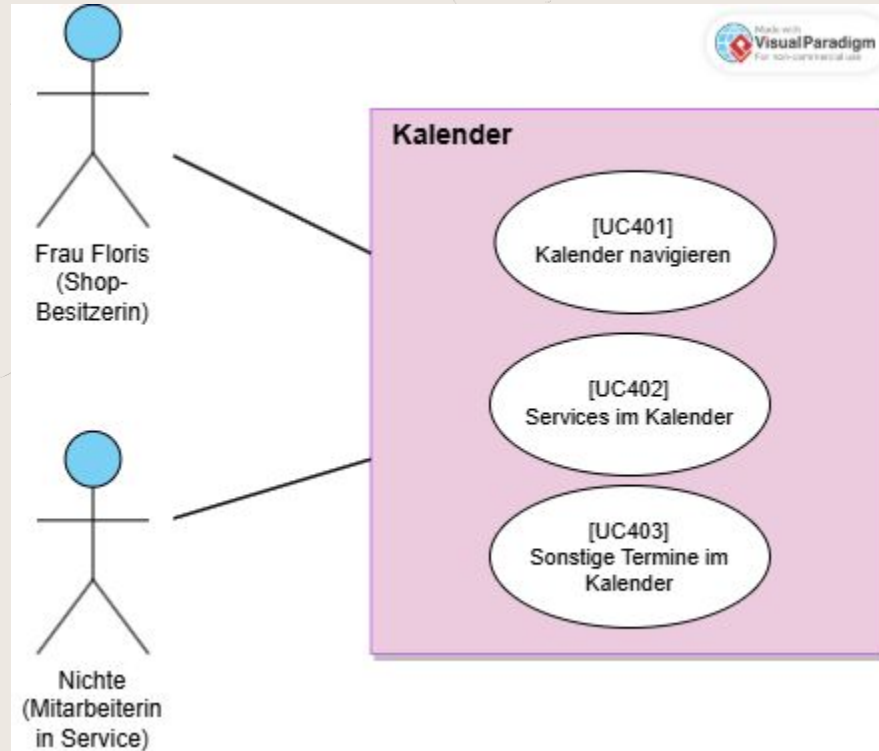


Source: <https://rock-the-prototype.com/programmieren-lernen/factory-pattern/>

```
public class OrderFactory {  
    private final UserAccountManagement userAccountManagement; 2 usages  
  
    public OrderFactory(UserAccountManagement userAccountManagement) { 1 DeeWan17  
        this.userAccountManagement = userAccountManagement;  
    }  
  
    public SimpleOrder createSimpleOrder() { return new SimpleOrder(getDefaultUserAccount()); }  
  
    public EventOrder createEventOrder(LocalDate eventDate, String deliveryAddress, Client client, String notes) {  
        return new EventOrder(getDefaultUserAccount(), eventDate, deliveryAddress, client, notes);  
    }  
}
```

```
@PostMapping("/reservations/create") 1 ekso  
public String createReservationOrder(@RequestParam String clientName,  
                                     @RequestParam("reservationDateTime") LocalDateTime reservationDateTime,  
                                     @RequestParam("phone") String phone,  
                                     @RequestParam Map<String, String> products,  
                                     @RequestParam("notes") String notes) {  
    ReservationOrder reservationOrder = orderFactory.createReservationOrder(reservationDateTime,  
                                     getOrCreateClient(clientName, phone), notes);  
    reservationOrderService.save(reservationOrder, products);  
    return "redirect:/services";  
}
```

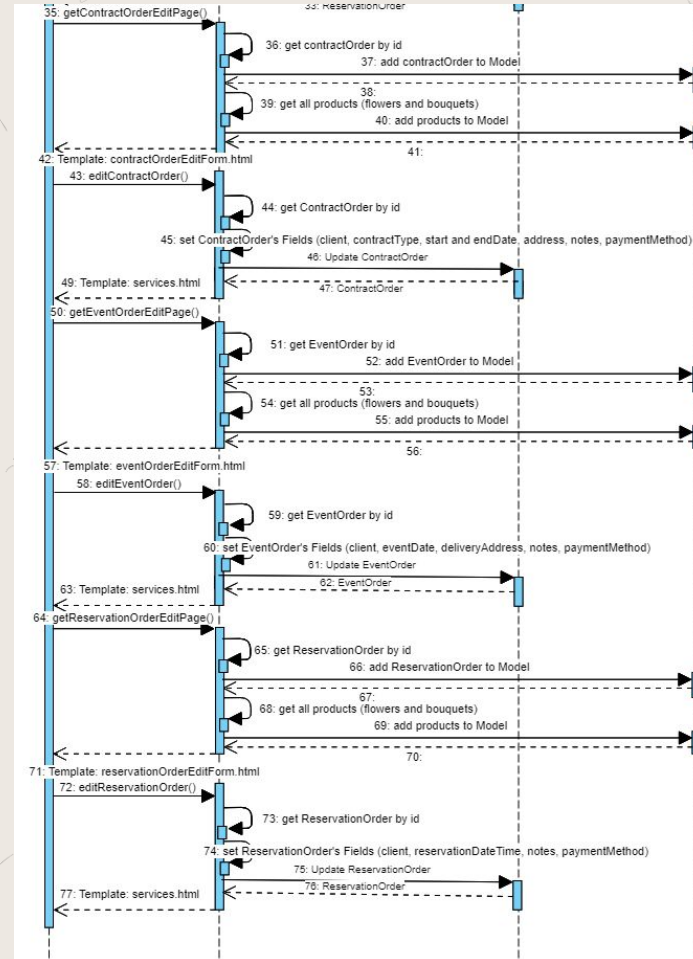
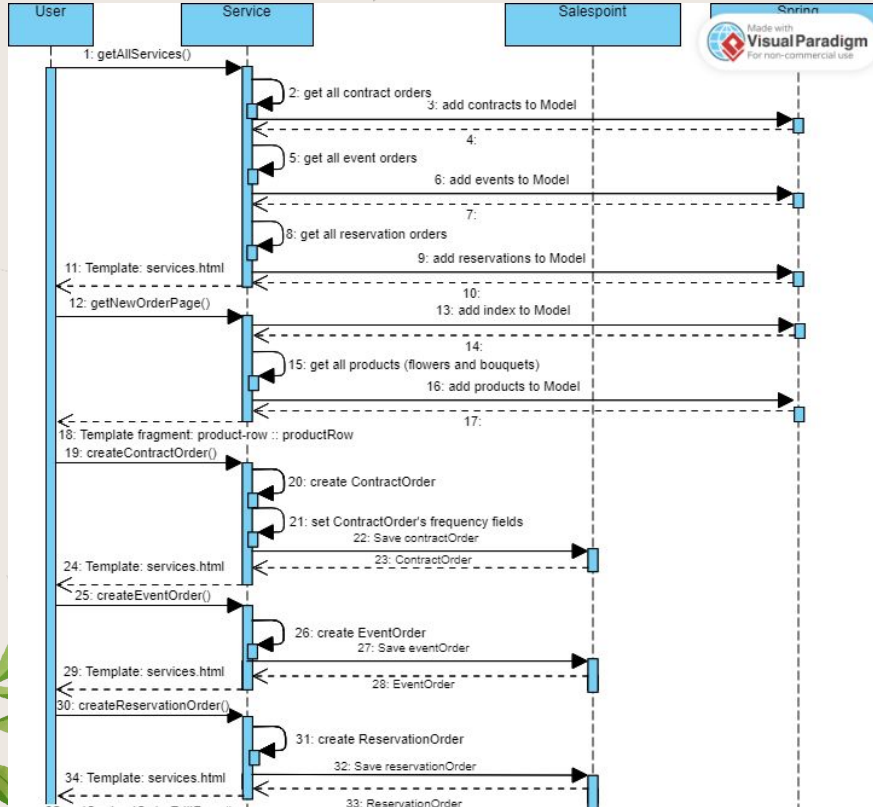
Kalender

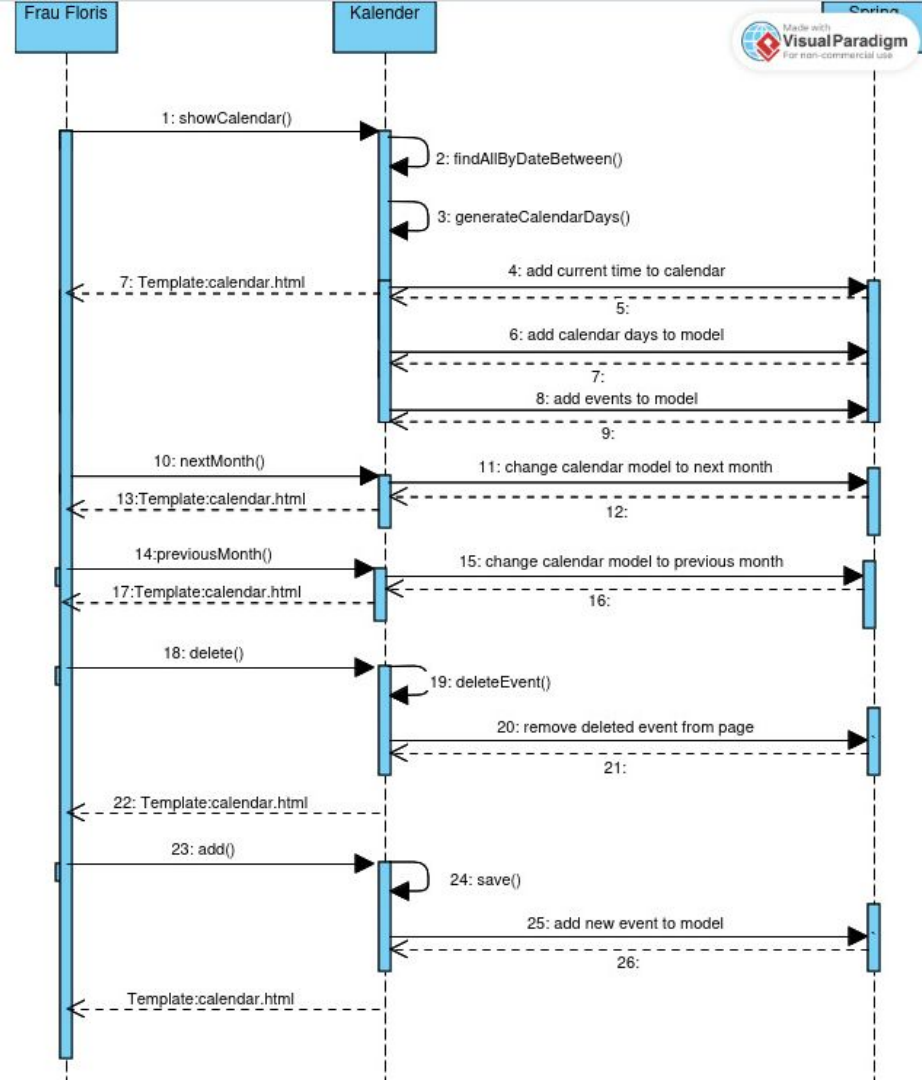


Sequenzdiagramme



Service





Kalender