HTBL Imst

AUFBAULEHRGANG FÜR INFORMATIK

Übungszettel – Makroarchitektur

Name: Michael Bogensberger

Datum: 18.05.2022

1 Inhalt

2 Wieso Ports und Adapter

Zunächst wollen wir die klassische Schichtenarchitektur betrachten. Sie besteht zumeist aus drei Schichten:

- Der Datenhaltungsschicht, die für die Speicherung der Anwendungsdaten zuständig ist,
- der Logikschicht, die das Herzstück der Anwendung darstellt,
- und zuletzt der Darstellungsschicht, zuständig dafür die Daten anzuzeigen und mit dem Benutzer zu interagieren.

Um die drei Teile der Anwendung nun koordiniert miteinander kommunizieren zu lassen, legt man meist fest, dass eine Schicht nur mit der unter ihr liegenden Schichten kommunizieren darf. In der Praxis werden jedoch häufig Geschäftslogik und Benutzeroberflächen-Code vermischt. Daraus resultieren folgende Hauptproblemfelder:

- Die Anwendung kann nicht ohne größeren Aufwand automatisiert getestet werden.
- Es ist knifflig die Anwendung ganz oder in Teilen wiederzuverwenden beziehungsweise zu ersetzen.
- Die Entwicklung der einzelnen Anwendungskomponenten lässt sich nur schwer unabhängig voneinander vorantreiben.

Ein Beispiel aus der Praxis: Eine Anwendung des Kunden nutzte einen proprietären SQL-Datenbank-Server für die Datenhaltung. Für bestimmte Anwendungsfälle wurde die Funktionalität als sogenannte „Stored Procedures“, also als in der Datenbank hinterlegte Funktionen, realisiert. Wann immer ein Benutzer gewisse Anwendungsfälle durchführen wollte, rief er von der Darstellungsschicht über die Persistenzschicht eine entsprechende Prozedur auf. Die Anwendung war somit direkt abhängig von der gewählten Persistenzlösung. Details aus der eigentlich untersten Schicht übertrugen sich bis fast hinauf in die oberste.

Durch die Vermischung von Anwendungslogik und Datenhaltung ließ sich die Persistenzschicht nicht ohne Weiteres auszutauschen. Zur Lösung des Problems wurde ein Service entwickelt, der die Interaktion mit der Datenbank übernahm.

Wäre die Anwendung ursprünglich nach dem „Ports and Adapters“ Muster entwickelt worden, hätte man nicht in eine komplett neue zusätzliche Anwendung investieren müssen. Ein einzelner neuer Adapter wäre ausreichend gewesen.

2.1 Definition

Anstatt der üblichen Schichten wird die Anwendung in die Namen gebenden Ports und Adapter eingeteilt. Adapter sind Komponenten nach dem klassischen Adapter-Pattern der Gang of Four. Das Wort „Port“ wurde gewählt, um an die Ports eines Computers zu erinnern. An einen solchen Port kann ein beliebiges Gerät angeschlossen werden. Dazu muss es lediglich das Protokoll des Anschlusses verstehen. Für jedes Gerät gibt es einen Adapter, der zwischen der API und den Signalen übersetzt, die das Gerät benötigt. Ein passendes Beispiel hierfür sind die USB-Anschlüsse an einem Rechner. Von Abschussrampen die Schaumstoffpfeile verschießen, bis zu Tastaturen und Mäusen kann man dank einheitlicher Schnittstelle alles anschließen und betreiben.

Dieses Bildnis aus Anschlüssen und Adaptern lässt sich leicht auf Teile von Anwendungen übertragen: Die Benutzeroberfläche (GUI - Graphical User Interface) ist ein Beispiel für einen Adapter, der die Kommunikation zwischen Nutzer und Anwendung ermöglicht. Eine Datenbank ist ein Adapter, der die Datenhaltung verwaltet.

Bei Ports werden primäre und sekundäre Ports unterschieden. Primäre Ports sind solche, die die Anwendung anbietet und von außen aufgerufen werden. Die eigentliche Anwendungslogik ist z.B. ein primärer Port.

Sekundäre Ports werden von der Anwendung selbst aufgerufen. Der Port für die Datenhaltung ist ein solcher sekundärer Port.

Hilfreiche Guides:

- Ports-and-Adapters-software-engineering-de
- Ports-and-Adapters-software-architecture

3 Svelte-Frontend

Ich habe mich beim Svelte Frontend für das Carbon Design System von IBM entschieden, da es das beste Komplett-UI-System für Svelte ist.

3.1 Homepage

Startet man die App so gelangt man auf die Index-Seite. Hier kann man nun weiter navigieren. Zurzeit sind die „Orders“ und „Packings“ Seiten komplett implementiert. Die „Costumers“ Seite ist noch in Bearbeitung. Für das Routing habe ich den „svelte-spa-router“ verwendet.

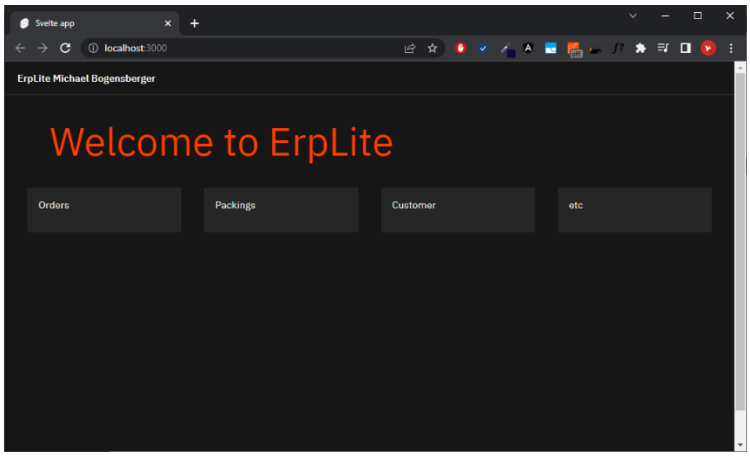

Abbildung 1 Index Page – ERP-Lite

3.2 Routing

Fürs Routing muss man in der “App.svelte” folgendes Object mit den Routen erstellen. Nun müssen nur noch die Dateien unter „routes“ erstellt werden und das Routing ist schon aufgesetzt.

```
let routes = {
  "/": Home,
  "/order": Orders,
  "/customer" : Customer,
  "/packings": Packings,
  "*": NotFound,
};
```

Abbildung 2 Svelte-Routing

3.3 Orders

Auf der „Orders“ Page kann man Bestellungen aufnehmen, diese in einer Tabelle ansehen und diese dann auf „verified“ setzen. Wenn man auf „place Order“ klickt öffnet sich ein Modal mit dem man eine Bestellung setzen kann. Ist die Bestellung gesetzt wird die Tabelle aktualisiert.

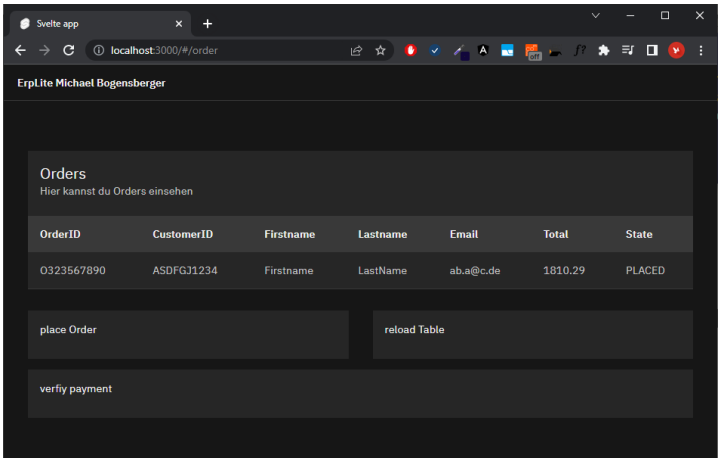

Abbildung 4 Order-Page

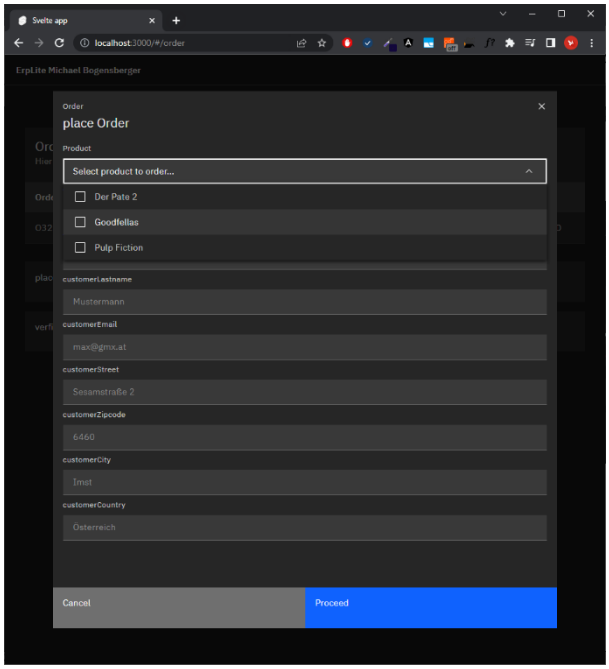

Abbildung 3 place Order Modal

Will man nun eine Bestellung verifizieren klickt man auf „verify payment“ und es öffnet sich ein Modal indem man die OrderID der Bestellung eingibt und die Bestellung ist verifiziert.

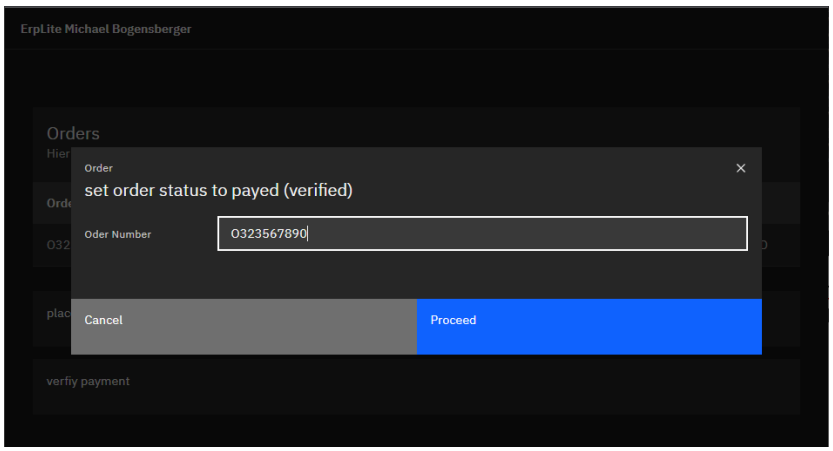

Abbildung 5 verify order Modal

3.4 Packings

Auf der „Packings“ Seite sieht man alle verifizierten Bestellungen bzw. ist dort die Paketliste mit den jeweils Bestellten Produkten zu sehen.

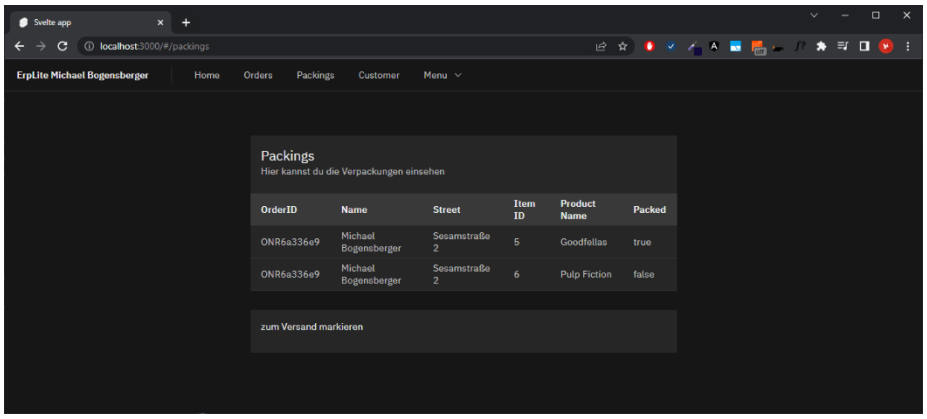

Abbildung 6 packings-Page

Will man ein Item in der Bestellung zum Versand markieren kann man auf „zum Versand markieren“ klicken und es öffnet sich wieder ein Modal. Hier gibt man nun die Item ID an. Nun kann man auf der „Orders“ Page sehen, dass das Produkt für den Versand vorbereitet wird. Zudem ist nun „Packed“ auf true gestellt.

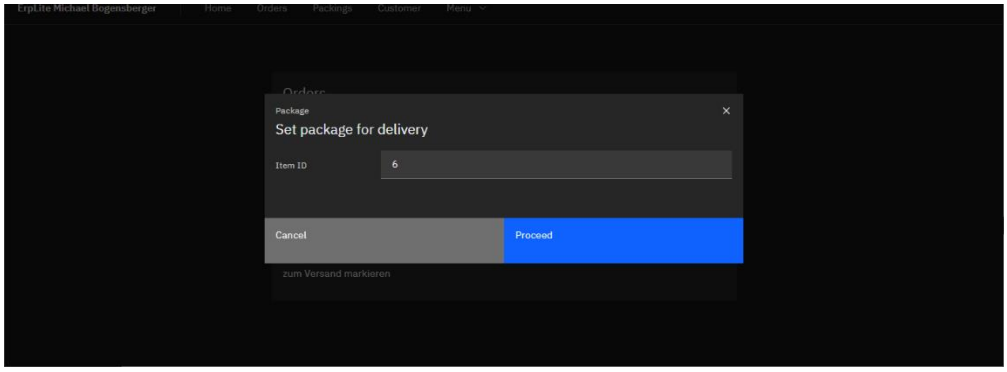

Abbildung 7 packing to delifery Modal

3.5 Customer

Auf der „Customer“ Page kann man Kunden verwalten und erstellen. Das Modal zur Erstellung des Kunden sieht folgendermaßen aus:

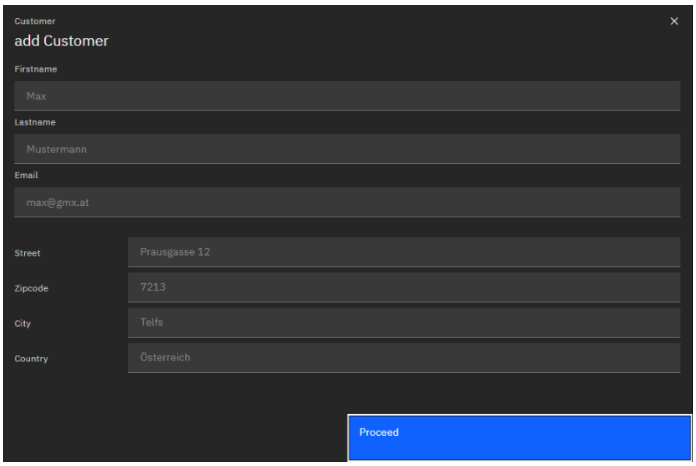

Abbildung 8 add customer Modal

4 Backend – setPackedForPacking

In folgendem Abschnitt ist die Funktionalität im „PackingRestController“ zu sehen. Genauer gesagt sieht man hier jenes Mapping das für das zum Versand markieren zuständig ist.

```
@PostMapping("/setPackedForPacking/{packingItemId}")
public void setPackingItemPackedForPacking(@PathVariable Long packingItemId) {
    Logger.getLogger(this.getClass().getName()).log(Level.INFO, "Handling packing for item# " + packingItemId);

    Optional<PackingItem> optionalPackingItem = this.packingItemRepository.findById(packingItemId);
    if (optionalPackingItem.isPresent()) {
        PackingItem packingItem = optionalPackingItem.get();
        packingItem.setPacked(true);
        packingItemRepository.save(packingItem);

        Long packingId = packingItem.getPacking().getId();

        Optional<Packing> packing = this.packingRepository.findById(packingId);

        boolean allpaked = true;
        for (PackingItem item : packing.get().getPackingItemList()) {
            if (!item.isPacked()) allpaked = false;
        }
        if (allpaked) {
            Logger.getLogger(this.getClass().getName()).log(Level.INFO, "All items for order# " +
            packing.get().getOrderId() + "packed. Publishing event ...");
            this.stockMessagePublisher.publishOrderPackedSpringEventForOrderId(packing.get().getOrderId());
        }
    }
}
```

Abbildung 9 setPackedForPacking Mapping

Hier wird zunächst geloggt, dass ein Item gepackt werden soll. Nun wird das Optionale PackingItem mit dem Repository und der ID gesucht. Ist es gefunden bzw. vorhanden gehen wir in eine IF Anweisung. Hier holen wir uns das packingItem und setzen den „packed“ Status auf true. Nun speichern wir das Item. Als nächstes holen wir uns die packingID vom PackingItem und danach das packing durch die ID.

Wie wir sehen können, ist nun „allpaked“ bereits auf true. Nun gehen wir mit einer for schleife alle items durch und wenn eines davon nicht gepackt ist, setzen wir „allpaked“ auf false. Wenn „allpaked“ nun true ist gehen wir erneut in eine IF Anweisung hinein. Hier loggen wir nun das alle Items bereit sind. Nun setzen wir das „packing“ auf „bereit für den Versand“.

5 Backend – packings

In folgendem Bild ist das „packings“ Mapping zu sehen. Hier suchen wir uns einfach durch das Repository alle Packings raus und geben dieses zurück.

```
@GetMapping("/packings")
public ResponseEntity<List<Packing>> getAllPackings() {
    List<Packing> packingLists = this.packingRepository.findAll();
    return ResponseEntity.ok(packingLists);
}
```

Abbildung 10 packings Mapping

6 C4-Diagramme

In folgendem Abschnitt sind zum Projekt passende C4-Diagramme zu finden.

6.1 System Context

In folgendem C4-Diagramm ist der System Context zu sehen. Sprich, hier sieht man das System von ganz außen.

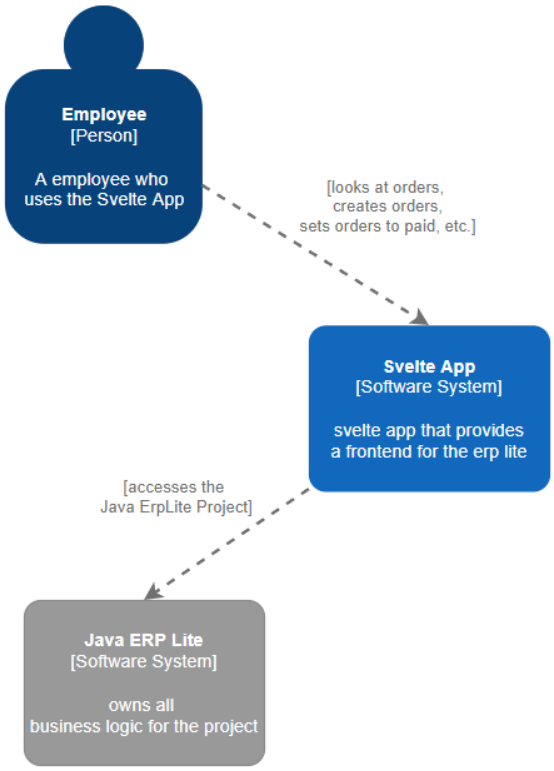

Abbildung 11 C4-System-Context

6.2 Container View

Hier ist die Container View zu sehen. Hier ist das System weiter aufgeteilt zu sehen. Zum einen gibt es immer noch die Single-Page-Application. Jedoch ist das Java Projekt weiter aufgeteilt. Hier zum Beispiel in API Application und der Datenbank.

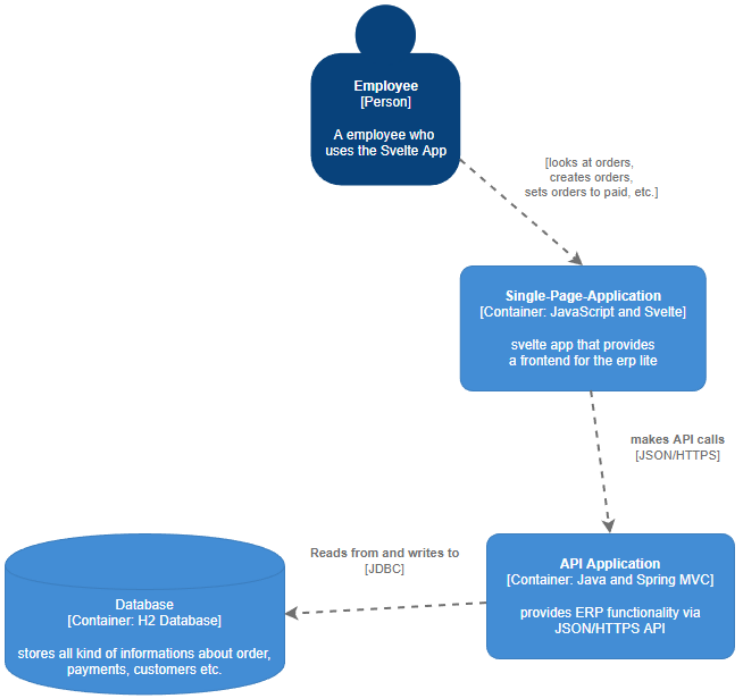

Abbildung 12 C4-Container-View

6.3 Component View

In folgendem C4-Diagramm ist die Component View zu sehen.

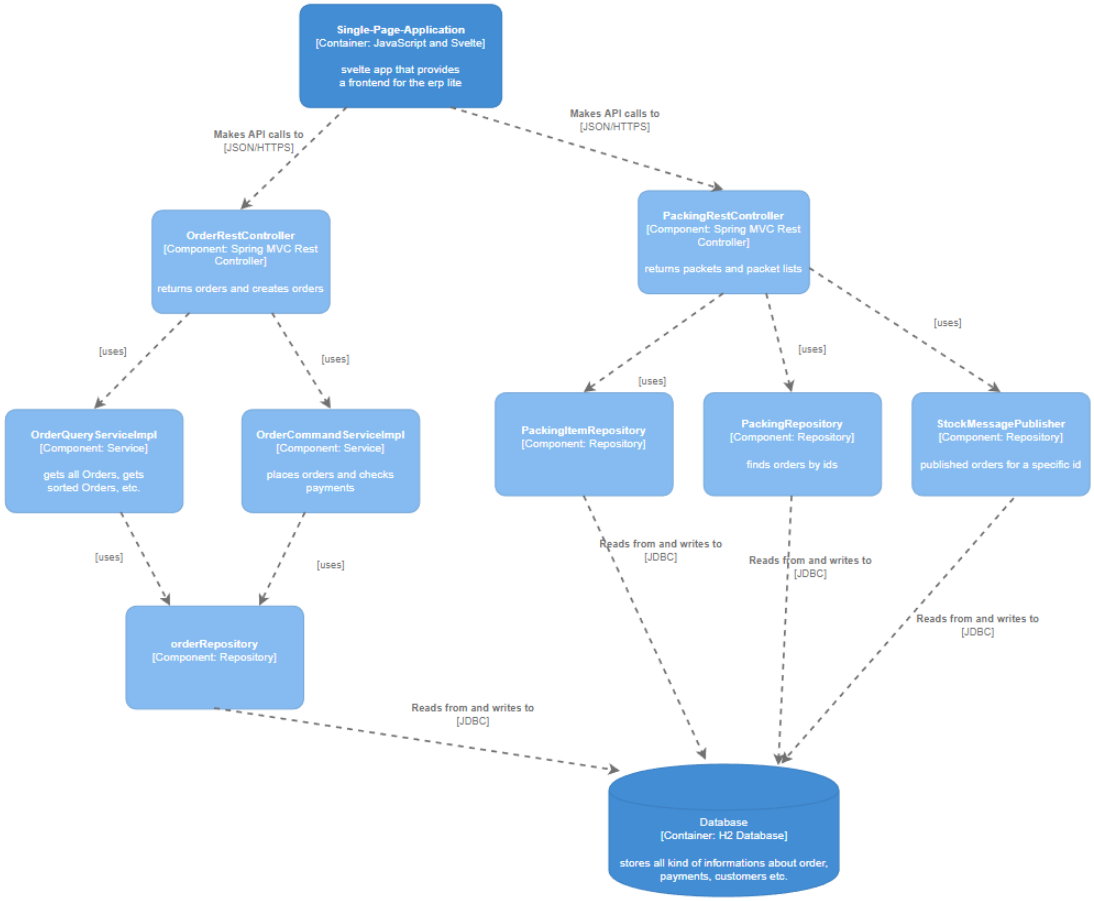

Abbildung 13 C4-Component-View

7 Code

Der Code zum Projekt ist unter folgendem Link auf GitHub zu finden: GitHub Repo

8 Abbildungsverzeichnis