EAI: Semester Assignment

E-Shop Backend with Rest

Ein Studentenprojekt der Fachhochschule Nordwestschweiz

Alle Bilder von den Folien oder Screenshots

GitHub:

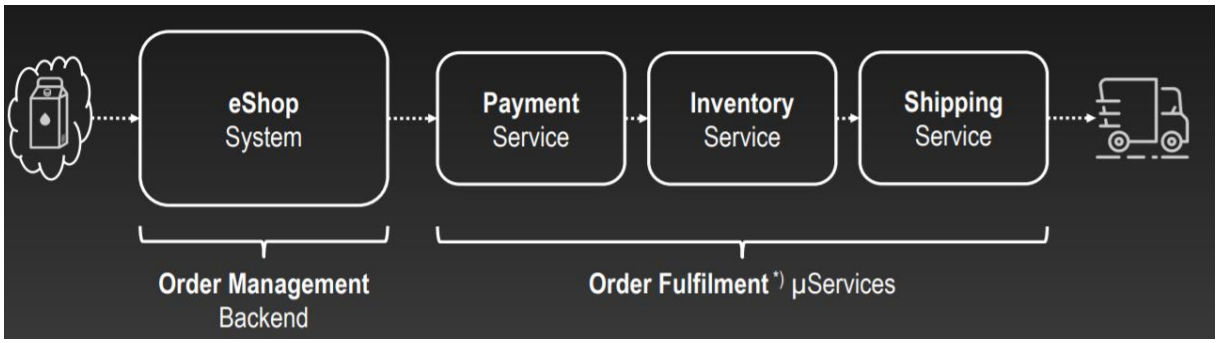
https://github.com/dubidamdam/EAI_Shop

Projekthandbuch

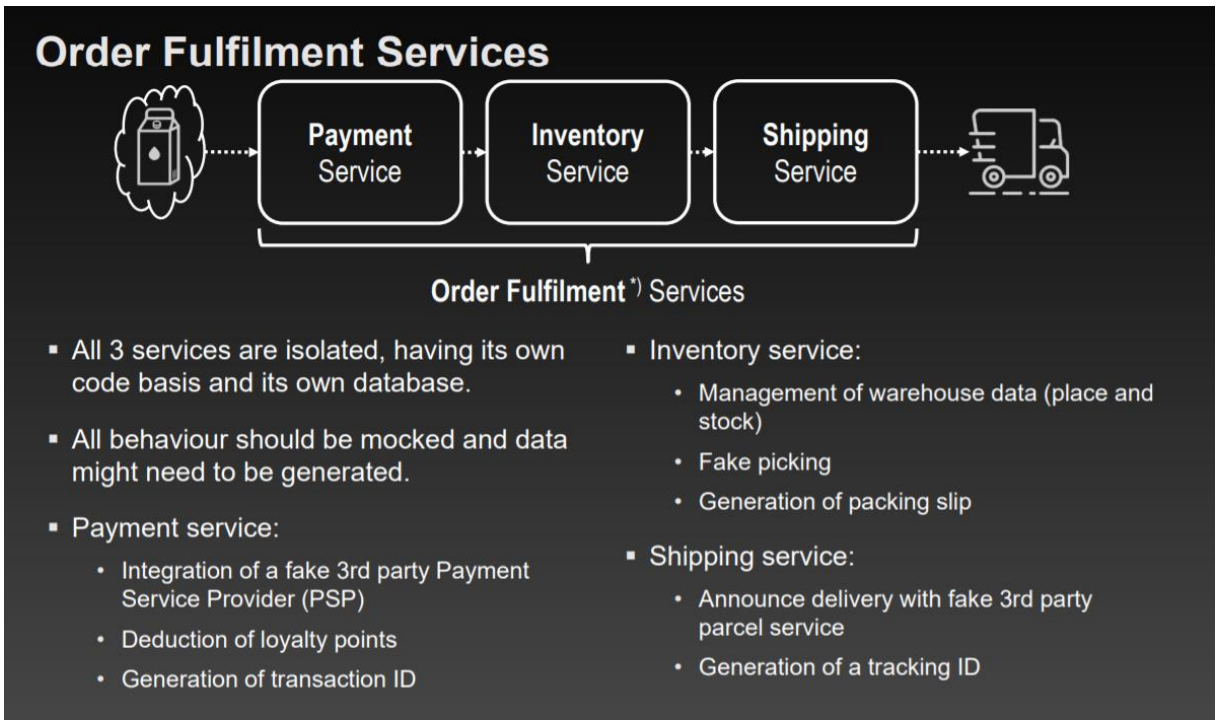
Projektmitglieder	Herr Fabian Dubach
Dozierende	Herr Andreas Martin
Version	final
Studiengang	Fachhochschule Nordwestschweiz Wirtschaftsinformatik, 5. Semester
Datum	16.12.2018

Inhalt

1. Kurzbeschreibung des Projekts



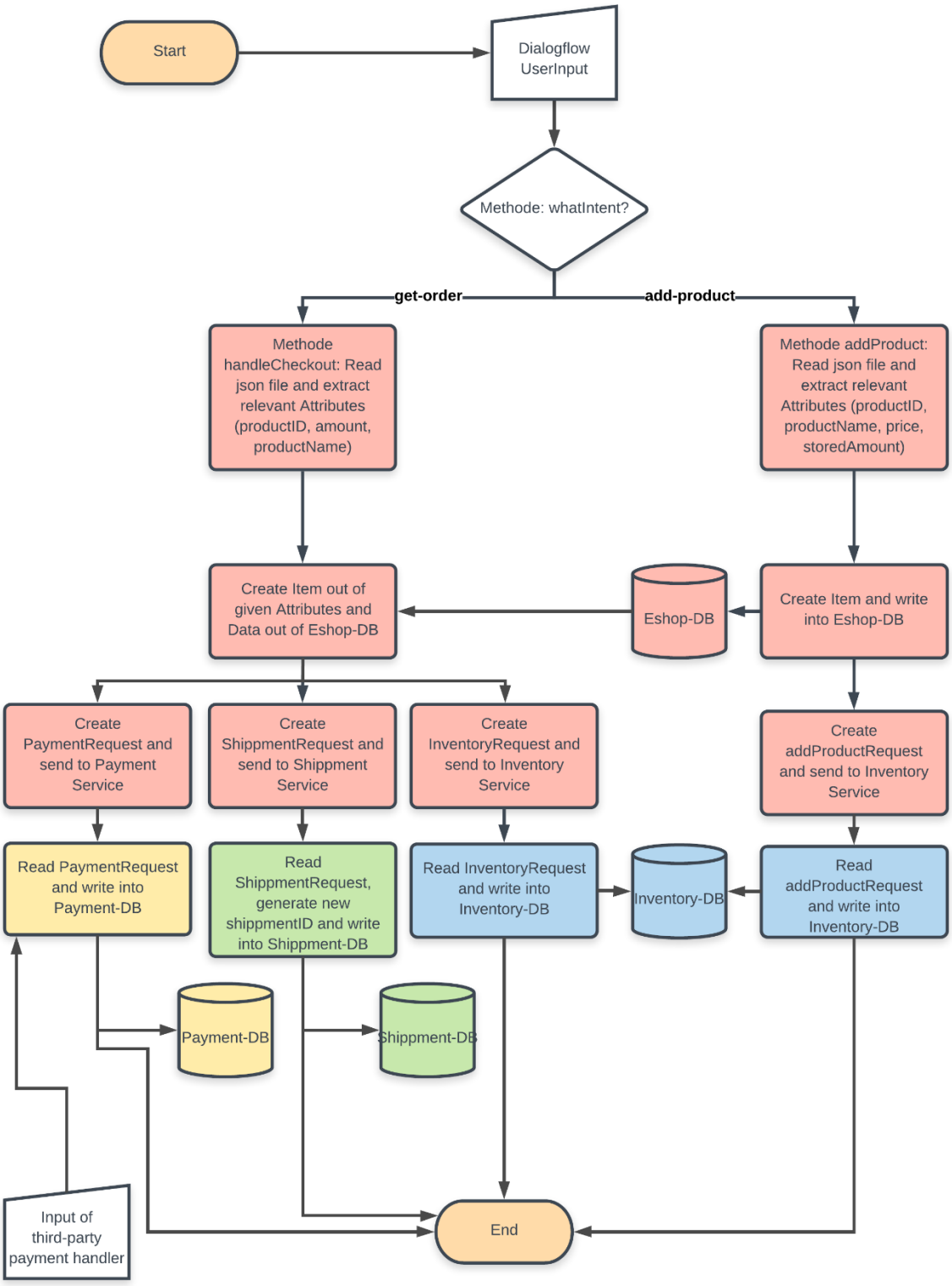
End-to-end E-Shop welcher aus unabhängigen Services besteht. Alles Services haben eine eigene h2-DB. Sie kommunizieren via Rest. Der E-Shop fungiert als Kontroller für das ganze Geschehen.



2. Funktionales

2.1 Ablauf

Ursprünglich habe ich noch weitere Methoden (wie bspw. deleteProduct) eingeplant, diese sind jedoch weggefallen, da der Scope des Projekts gekürzt wurde (da ich das Projekt alleine durchgeführt habe. Folgendes Diagramm soll mögliche Abläufe der Applikation vereinfacht und bildlich darstellen.



2.2 Repository Interfaces

Die Repository Interfaces erweitern JPA Repository und dienen der Speicherung und dem Zugriff auf die Datenbank.

2.3 Eshop

2.3.1 Methode whatIntent

Die Methode whatInten nimmt ein json von Dialogflow entgegen und schaut was der übergebene „Intent“ (Absicht) ist. Je nach dem wird die entsprechende interne Methode aufgerufen.

```
//Methode die den Intent prüft.
@PostMapping("/eshop")
public boolean whatIntent(@Valid @RequestBody Jsonconv json) {
    String intent = json.getQueryResult().getIntent().getDisplayName();
    if (intent.equals("add-product")) {
        this.addAProduct(json);
    } else if (intent.equals("get-order")) {
        this.handleCheckout(json);
    }
    return true;
}
```

2.3.2 Methode AddProdukt

Die AddProdukt Methode stellt sicher das der Shopbetreiber neue Produkte erstellen kann. Die Funktion nimmt ein Produkt entgegen. Das Produkt wird dann in der Eshop Datenbank mit der EshopRepository Klasse eingetragen. Ausserdem wird ein RestPost an Inventory gesendet und dort dasselbe Produkt eingetragen. Die Produkte stehen mit dem Attribut „ProduktID“ (nicht zu verwechseln mit ID) im Bezug zu einander. Sozusagen ein Foreignkey, welcher jeweils per Rest übergeben wird.

Die Methode nimmt ein json entgegen welches mit Hilfe von Java Klassen welche mit jsonschema2pojo erstellt wurden, die benötigten Parameter ausliest und ins Produkt schreibt.

```
private boolean addAProduct(Jsonconv json) {
    Long productID1 = json.getQueryResult().getParameters().getProductID();
    Long price1 = json.getQueryResult().getParameters().getPrice();
    String productName1 = json.getQueryResult().getParameters().getProductName();
    Long storedAmount1 = json.getQueryResult().getParameters().getStoredAmount();

    Integer storedAmount2 = storedAmount1.intValue();
    Integer price2 = price1.intValue();

    Product addProduct = new Product();
    addProduct.setProductID(productID1);
    addProduct.setPrice(price2);
    addProduct.setProductName(productName1);
    addProduct.setStoredAmount(storedAmount2);

    //System.out.println(p1.getProductName().toString());
    productRepository.save(addProduct);

    //generiert AddProductRequest
    AddProductRequest addProductRequest = new AddProductRequest();
    addProductRequest.myList.add(addProduct);
    System.out.println(restTemplate.postForObject( url: "http://localhost:8083/addProduct", addProductRequest, Boolean.class));
    return true;
```

2.3.3 Methode handleCheckout

Diese Methode bearbeitet eine Bestellung und sendet alle nötigen Informationen an die anderen Services. Sie nimmt das json und liest wiederum von dort die benötigten Parameter ein.

```
//generiert PaymentRequest und übernimmt Preis und ID von orderItem
PaymentRequest paymentRequest = new PaymentRequest();
Integer stckprice = productRepository.findByProductID(orderItem.getProductID()).getPrice();
Long productID = productRepository.findByProductID(orderItem.getProductID()).getProductID();
Integer amount = orderItem.getAmount();
//preisberechnung aus amount*stückpreis
Integer price = stckprice * amount;
paymentRequest.price = price;
paymentRequest.productID = productID;

//generiert ShipmentRequest
ShipmentRequest shipmentRequest = new ShipmentRequest();
shipmentRequest.productID = productID;

//generiert inventoryRequest
// Fügt Item der AL in InventoryRequest und somit der DB hinzu
InventoryRequest inventoryRequest = new InventoryRequest();
inventoryRequest.myList.add(orderItem);

//Übergibt die Requests per Rest/Post
System.out.println(restTemplate.postForObject( url: "http://localhost:8082/payment", paymentRequest, Boolean.class));
System.out.println(restTemplate.postForObject( url: "http://localhost:8083/inventory", inventoryRequest, Boolean.class));
System.out.println(restTemplate.postForObject( url: "http://localhost:8081/shipment", shipmentRequest, String.class));
```

2.4 Inventory

Im Inventar wird von der Checkout Methode ein InventoryRequest entgegengenommen. Dieses beinhaltet eine Liste mit Items, die Bestellt wurden. Diese werden dann mit der Methode takeFromInventory in der Datenbank angepasst.

Die addNewProduct Methode fügt der Datenbank allfällige neue Produkte hinzu.

2.5 Shipment

Das Shipment ist relativ simpel gestaltet. Es generiert mit dieser Line eine neue UUID:

```
shipment.setTrackingId(UUID.randomUUID().toString());
```

Diese wird dann auch zurückgegeben und zusammen mit der productID in der Shipment DB abgelegt.

2.6 Payment

Das Payment erhält über einen PaymentRequest einen Gesamtbetrag, welcher bereits im Eshop, in der Checkout Methode errechnet wurde. Dieser trägt es dann in die Payment DB ein. Ausserdem wäre eine Methode getPaymentStatus bereit welche noch mit der Bezahl-Logik abgefüllt werden müsste. Für unser Projekt gibt diese einfach immer true zurück.

3. Tools

3.1 IntelliJ

Wieso: Weil Lieblingsumgebung.

Wofür: Ganze Java/Spring Implementation

3.2 H2 DB

Wieso: Weil so empfohlen.

Wofür: Jeweils unabhängige Datenbanken für die vier Services.

3.3 Dialogflow

Wofür: Post Inputs via Sprache/Texteingabe

Wieso: Weil optionale Anforderung und weil cool.

3.4 Ngrok

https://ngrok.com/

Wieso: Die Verbindung zu Dialogflow ist nur mühsam konfigurierbar, sofern man lokal arbeitet.

Wofür: HTTP/S Verbindung auf localhost von Dialogflow

3.5 Jsonschema2pojo

http://www.jsonschema2pojo.org/

Wieso: Ich hatte mit den übergebenen JSONs zu kämpfen.

Wofür: Um JSON in Java umzuwandeln.

Special thanks: Felix Lehner, welcher mir dieses Tool empfohlen hat.

4. Schwierigkeiten

Abgesehen davon, dass mein komplettes Projektteam (namentlich Team Rocket) zwei Wochen vor Abgabe abgesprungen ist, habe ich eigentlich vor allem kleinere Hindernisse gehabt.

Vor allem der Start ins Projekt verlief eher harzig, da es ohne Erfahrung im Bereich EAI, doch sehr schwierig ist, irgendwo mit dem Projekt anzufangen.

Danach hatte ich eigentlich nur noch Probleme mit der „Skalierung“ des Projekts. Da ich jeweils die Services immer funktional ein bisschen verbessert habe (Step-by-Step), hatte ich zwischendurch viel Aufwand in Form von Clean-Up des Codes. In einem weiteren Projekt würde ich zuerst die Services genauer „designen“ und danach programmieren. Hier habe ich halt mehr oder weniger einfach mal „drauflos programmiert“.

5. Fazit

Schade, dass das Projekt von der Studiengangleitung aus nicht mehr gewichtet wird. Das Projekt war zwar als EAI-Noob extrem aufwändig, aber auch extrem lehrreich. Teilweise hätte ich gerne noch weitere Funktionen eingebaut (gerade im Bereich Assistant/Dialogflow), aber musste Aufwand und Ertrag auch etwas in Relation setzen, weil wir momentan ja noch weitere Abgaben (ToBIT, Web-Eng), haben oder hatten.
Trotzdem finde ich das Module wie EAI unbedingt solche Projekte beinhalten müssen, da diese viel mehr Inhalt bieten als trockene Vorlesungen und Prüfungen wo auf Papier programmiert wird. Die Projekte müssen jedoch auch entsprechend gewichtet werden.
Ausserdem wäre auch die Implementation von weiteren Methoden, wie löschen und ändern von Produkten, Erstellung von Nutzer, Erstellung von Shopping-Carts etc. noch sehr interessant gewesen.