Fachhochschule Nordwestschweiz

Team Hydra

ShopStantly: «Shop Instantly»

Semester Assignment – Enterprise Application Integration

Dokumentation

Fachdozent: Dr. Andreas Martin

Zuletzt bearbeitet
16.12.2018

Erstellt von
Luca Joho,
Kai Krause,
Felix Lehner

Inhaltsverzeichnis

Kurzbeschreibung des Projektes

Für ein traditionelles Detailhandelsunternehmen wird eine neue Plattform erstellt, welche eine vollständig integrierte End-to-End-Lösung sein soll. Die Bestellungen werden dabei über den virtuellen Assistenten «Google Assistant» ausgelöst und sollen so den Bestellvorgang automatisieren und sowohl für den Anbieter wie auch für die Nachfrager vereinfachen.

Aufgrund der Komplexität und des Umfanges der Arbeit haben wir uns vor allem auf die Implementation fokussiert und daher keine zusätzlichen Modelle erstellt. Dies wurde gemäss Arbeitsauftrag für die Dokumentation aber auch nicht gefordert.

Das Projekt wurde im Rahmen einer Semesterarbeit im Modul Enterprise Application Integration erstellt und somit nicht im Auftrag eines realen Unternehmens entwickelt. Sämtliche Annahmen erfolgten durch das Projektteam und wurden nach bestem Wissen und Gewissen an fiktive Anforderungen angepasst.

Ausgangslage

Ein traditioneller Detailhandler hat sich im Rahmen der Digitalen Transformation eine neue Strategie zurechtgelegt. Zudem hat das Unternehmen ein Rebranding durchgeführt und möchte nun unter dem Namen «ShopStantly» – angelehnt an «Shop Instantly», also das «sofortige einkaufen» – durchstarten und Leader im Schweizer Detailhandelsmarkt werden.

Ziel / Nutzen

Das eShop-System soll einfach und schnell Bestellungen von Kunden via virtuellen Assistenten aufnehmen und dann automatisiert ausführen, abbuchen und speichern. Die Ziele sind nachfolgend im Detail aufgeführt.

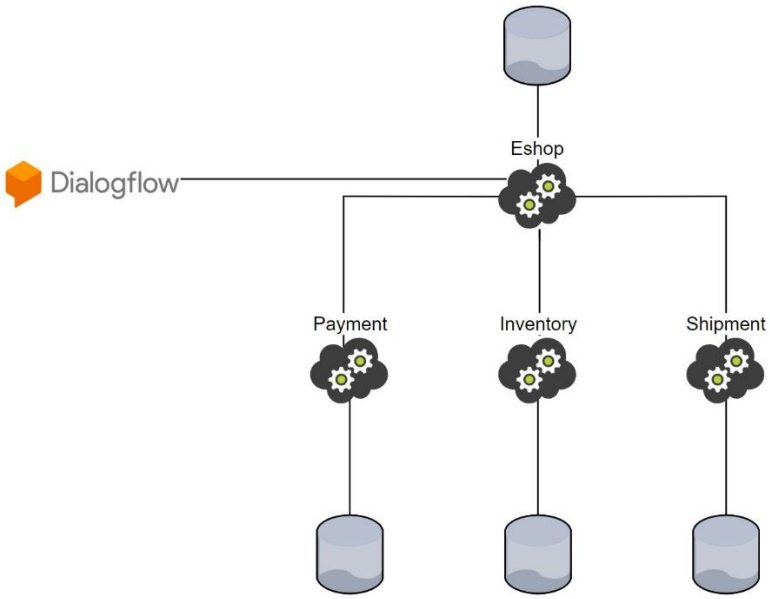
Ziel	Beschreibung
1. E-Shop	• Hauptkomponente • Anbindung Dialogflow und Verhalten des Assistenten auf Eingaben des Users • Modell-Definitionen unter anderem für Produkte, Kunden, Kreditkarten, etc. • Steuerung Einkaufsprozess und Aufruf der benötigten Microservices zum entsprechenden Zeitpunkt
2. Inventory	• Lagerverwaltung unter der Annahme, dass immer genügend Produkte an Lager sind (kein Bestand <= 0) • Erstellung der PDF-Datei (Lieferschein respektive Bestellung) • Erstellung und Senden des Bestätigungsmails
3. Payment	• Bezahlungsmöglichkeiten: Kreditkarte, Sammelpunkte oder Mischung der beiden Zahlungsarten • Kundensammelpunkteverwaltung (Gutschrift, Belastung, Saldo, Umwandlung, etc.) • Abwicklung Zahlungsprozess • Verhalten bei Abbruch inklusive Rückerstattung
4. Shipment	• Abwicklung Versandprozess • Erstellung einer individuellen Sendungsnummer

Tabelle 1: Ziele

Übersicht der Implementation

Aufbau der Applikation

Auf der folgenden Grafik ist der Aufbau der Applikation übersichtlich dargestellt. Das System ist in die vier Microservices E-Shop, Payment, Inventory und Shipment aufgeteilt, wobei die Klasse OrderController.java des Services E-Shop das Kontroll- und Ausführelement der verschiedenen Funktionen ist.



E-Shop

In der Klasse OrderController.java werden die durch den Google Assistenten aufgenommenen Befehle verarbeitet und entsprechend gehandelt. Der untenstehende Code-Ausschnitt zeigt zum Beispiel was passiert, wenn der Kunde ein Produkt in den Warenkorb legen möchte.

```
switch (jsonRequest.getResult().getAction().toString()) {
    case "item.add":
        Products products = preferencesRepository.findByName(jsonRequest.getResult().getParameters().getProduct()).getProducts();
        if (products != null) {
            if (carts.getProductsList().stream().noneMatch(dbProduct -> dbProduct.getName().equals(products.getName()))) {
                carts.getProductsList().add(products);
                cartsRepository.save(carts);
                response.put("speech", "Alright, I've added " + products.getName() + " to your cart. Anything else?");
            } else {
                response.put("speech", "Could not add " + products.getName() + " to cart. It has already been added.");
            }
        } else {
            response.put("speech", "Sorry, we do not have the product " + jsonRequest.getResult().getParameters().getProduct());
        }
        break;
```

Mittels Buffern können die Daten zwischen den Microservices ausgetauscht werden. Im untenstehenden Code-Ausschnitt wird der Payment-Service aufgerufen und allfällige Fehler beim Bezahlvorgang entsprechend aufgefangen, respektive definiert, was der Google Assistent in einem solchen Fall dem User ausgeben soll.

```
PaymentBuffer paymentBuffer = (PaymentBuffer) callService(createPaymentBuffer(order),  port: 8081,  path: "payment");
if (paymentBuffer.hasFailed) {
    System.err.println("Payment was not successful");
    response.put("speech", "Payment could not be processed. Try again later.");
    return response;
} else {
    callService(createInventoryBuffer(order),  port: 8082,  path: "inventory");
    callService(createShipmentBuffer(order),  port: 8083,  path: "shipment");

    clearCart(carts);
    response.put("displayText", "Payed with Credit Card: Fr. " + paymentBuffer.payedPrice +
            "\nDeducted Loyalty Points " + paymentBuffer.deductedLoyaltyPoints + "\nReceived Loyalty Points: " + paymentBuffer.addedLoyaltyPoints);
    Data responseData = new Data("Track your parcel", "Click me", "https://www.post.ch/static/Post/IT/FDS/post.jpg", "Post",
            "http://www.post.ch/swisspost-tracking?formattedParcelCodes=" + trackingID, "Thank you for your order! " +
            "Your packing slip will be sent to your e-mail address shortly.\n\nYou can already track your parcel online.", true);
    response.putPOJO("data", responseData);
}
```

In obenstehendem Code-Ausschnitt ist auch gut ersichtlich, dass der E-Shop-Service lediglich die Inputs der anderen Services verarbeitet und sozusagen mit dem Anwender kommuniziert. Die Berechnungen, zum Beispiel der Saldo der Sammelpunkte, erfolgen allerdings im jeweiligen Microservice.

Die einzelnen Buffer, hier der PaymentBuffer werden ebenfalls im E-Shop erstellt, um sie danach mittels der Methode «callService» für den Aufruf des entsprechenden Microservices verwenden zu können.

```
/**
 * Creates a PaymentBuffer
 *
 * @param order Order that will be processed
 * @return PaymentBuffer
 */
private PaymentBuffer createPaymentBuffer(Orders order) {
    // PaymentService: Pay for Order
    PaymentBuffer paymentBuffer = new PaymentBuffer();
    paymentBuffer.price = order.getPrice();
    paymentBuffer.paymentType = order.getPaymentType();
    paymentBuffer.orderID = order.getID();
    paymentBuffer.customerID = order.getCustomers().getID();
    return paymentBuffer;
}
```

```
/**
 * Calls the microservice
 *
 * @param buffer Buffer that needs to be sent
 * @param port   Port of the receiving application
 * @param path   Name of the receiving application
 * @return boolean
 */
private Object callService(Object buffer, int port, String path) {
    final String uri = "https://shopstantly-" + path + ".herokuapp.com/" + path; // @heroku
    // final String uri = "http://localhost:" + port + "/" + path; // @localhost
    RestTemplate restTemplate = new RestTemplate();
    System.out.println("Called Service: " + uri);

    Object result = null;
    if ("payment".equals(path)) {
        result = restTemplate.postForObject(uri, buffer, PaymentBuffer.class);
    } else if ("inventory".equals(path)) {
        result = restTemplate.postForObject(uri, buffer, InventoryBuffer.class);
    } else if ("shipment".equals(path)) {
        result = restTemplate.postForObject(uri, buffer, ShipmentBuffer.class);
        trackingID = ((ShipmentBuffer) result).trackingID;
    }

    System.out.println(result);
    return result;
}
```

Inventory

Im zweiten Microservice, dem «Inventory», sind insbesondere die beiden sehr wichtigen Klassen «MailHelper» und «PdfHelper» erwähnenswert. Bevor die Bestätigungsemail erstellt wird, werden die Properties mit den Zugangsdaten zum Mail-Server von Gmail initialisiert.

Nach der Initialisierung werden die benötigten Daten, also der Text sowie der Betreff im Mail eingefügt. Die Daten stammen dabei aus der Klasse «InventoryController.java» von wo aus auch die Methode zum Versand der E-Mails aufgerufen wird.

```
/**
 * Send an email with the passed content to the recipient
 * by using the predefined hydragroup@gmail.com address as the sender.
 *
 * @param to      email address from the recipient
 * @param subject subject line of the email
 * @param text    content/body of the mail
 * @param attachment
 * @return true/false if the send was successful
 */
public static boolean SendMail(String to, String subject, String text, ByteArrayOutputStream attachment, String attachmentName) {
    Properties properties = new Properties();
    properties.put("mail.smtp.host", MAIL_HOST);
    properties.put("mail.smtp.socketFactory.port", MAIL_PORT);
    properties.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
    properties.put("mail.smtp.auth", "true");
    properties.put("mail.smtp.port", MAIL_PORT);

    Session session = Session.getDefaultInstance(properties,
            new javax.mail.Authenticator() {
                protected PasswordAuthentication getPasswordAuthentication() {
                    return new PasswordAuthentication(USERNAME, PASSWORD);
                }
            });
```

Der Lieferschein, respektive die Bestellbestätigung, wird per E-Mail als PDF-Datei an den Kunden gesendet. Für die Erstellung der PDF-Datei ist die Klasse «PdfHelper.java» zuständig. Dazu werden die Daten als OutputStream in einem Byte Array ausgegeben.

```
public ByteArrayOutputStream createPdf(InventoryBuffer inventoryBuffer) {
    try {
        Document document = new Document();
        PdfWriter.getInstance(document, baos);
        document.open();
        addMetaData(document);
        addContent(document, inventoryBuffer);
        document.close();
        System.out.println("PackingSlip created.");
    } catch (Exception e) {
        e.printStackTrace();
    }
    return baos;
}
```

Payment

Wenn der Kunde die Bezahlung mit Sammelpunkten tätigen möchte, dazu aber über einen zu kleinen Saldo verfügt, wird der Restbetrag von der Kreditkarte abgebucht. Im untenstehenden Code-Ausschnitt wird dieser Spezialfall im letzten «else»-Teil behandelt.

```
/**
 * Business Process of Payment
 *
 * @param req Receives filled PaymentBuffer
 * @return PaymentBuffer
 */
@PostMapping("/payment")
public PaymentBuffer startPaymentProcess(@Valid @RequestBody PaymentBuffer req) {
    Customers customers = retrieveCustomer(req.customerID);
    int availableLoyaltyPoints = customers.getLoyaltyPoints();
    int priceInLoyaltyPoints = (int) ((req.price * LOYALTYPOINTSMULTIPLIER));

    if (req.paymentType == PaymentType.CREDITCARD) {
        customers.setLoyaltyPoints(availableLoyaltyPoints + priceInLoyaltyPoints);
        req.addedLoyaltyPoints = priceInLoyaltyPoints;
        req.payedPrice = req.price;
    } else if (req.paymentType == PaymentType.LOYALTYPOINTS) {
        if (availableLoyaltyPoints >= priceInLoyaltyPoints) {
            customers.setLoyaltyPoints(availableLoyaltyPoints - priceInLoyaltyPoints);
            req.deductedLoyaltyPoints = priceInLoyaltyPoints;
        } else {
            // Difference between availableLoyaltyPoints and priceInLoyaltyPoints will be payed with CreditCard
            customers.setLoyaltyPoints(0);
            req.payedPrice = req.price - (availableLoyaltyPoints / LOYALTYPOINTSMULTIPLIER);
            req.deductedLoyaltyPoints = availableLoyaltyPoints;
        }
    }
```

Im Payment-Service wird selbstverständlich auch der Fall einer nicht erfolgreichen Bezahlung behandelt. Dabei wird der alte Saldo der Sammelpunkte wiederhergestellt und der Kunde wird über die nicht erfolgreiche Bezahlung informiert.

```
/**
 * Reverts the complete Payment including LoyaltyPoints
 *
 * @param req Receives filled PaymentBuffer
 * @param customers Customers
 * @param priceInLoyaltyPoints Price of order in loyaltyPoints
 * @param availableLoyaltyPoints available loyaltyPoints of customer
 */
private void revertPayment(PaymentBuffer req, Customers customers, int priceInLoyaltyPoints, int availableLoyaltyPoints) {
    if (req.paymentType == PaymentType.CREDITCARD) {
        customers.setLoyaltyPoints(customers.getLoyaltyPoints() - priceInLoyaltyPoints);
        System.err.println("*** Payment failed! *** \nLoyaltyPoints removed: " + priceInLoyaltyPoints);
    } else if (req.paymentType == PaymentType.LOYALTYPOINTS) {
        if (customers.getLoyaltyPoints() == 0) {
            customers.setLoyaltyPoints(availableLoyaltyPoints);
            System.err.println("*** Payment failed! *** \nLoyaltyPoints added: " + availableLoyaltyPoints);
        } else {
            customers.setLoyaltyPoints(availableLoyaltyPoints + priceInLoyaltyPoints);
            System.err.print("*** Payment failed! *** \nLoyaltyPoints loyaltyPoints: " + priceInLoyaltyPoints + "\n\n");
        }
    }
    customersRepository.save(customers);
}
```

Shipment

Im kleinsten der vier Microservices wird eine individuelle Trackingnummer für das Paket erstellt, damit der Kunde den Lieferstatus seiner Bestellung verfolgen kann. Die Ausgabe an den Kunden inklusive Direktlink zur Schweizerischen Post erfolgt dann wiederum über die Hauptklasse «OrderController.java» des Microservices «E-Shop».

```
/**
 * Business Process of Shipment
 *
 * @param req complete ShipmentBuffer
 * @return ShipmentBuffer
 */
@PostMapping("/shipment")
public ShipmentBuffer createShipment(@Valid @RequestBody ShipmentBuffer req) {
    Shipments shipment = new Shipments();
    shipment.setOrderID(req.orderID);
    shipment.setTrackingId(generateString());
    new Thread(() -> shipmentRepository.save(shipment)).start();

    req.trackingID = shipment.getTrackingId();
    return req;
}
```

Tools

IntelliJ IDEA Ultimate Edition

Für die Programmierung setzten wir auf die integrierte Entwicklungsumgebung IntelliJ IDEA von JetBrains. Da die Projektgruppe bereits frühere Implementationen mit IntelliJ IDEA umgesetzt und sehr gute Erfahrungen gemacht hat, kam für uns keine andere Entwicklungsumgebung in Frage.

Bitbucket & Sourcetree

Für die Versionsverwaltung via Git haben wir als Filehosting-Dienst «Bitbucket» und für die Verwaltung der Commits «Sourcetree» eingesetzt. Mit Bitbucket und Sourcetree hat das Projektteam bereits in früheren gemeinsamen Projekten gute Erfahrungen gemacht, weshalb wir auch für dieses Projekt wiederum auf die bewährte Kombination setzten.

Heroku

Gemäss den Vorgaben sollte das Projekt wenn möglich auf einer Cloud-Plattform laufen. Wir haben uns dazu für den Cloud platform as a service (PaaS)-Anbieter Heroku entschieden, welchen wir auch beim Praxisprojekt bereits benutzt haben. Leider fallen die Microservices jeweils nach 30 Minuten in einen Ruhezustand und müssen danach wieder einzeln neu gestartet werden. Im Ruhezustand können die Microservices nicht aufgerufen werden, das Ausführen des eShop-Systems mit dem Google Assistant ist dann nicht möglich.

Postman

Bevor die Applikation dank der Anbindung an Dialogflow mit dem Google Assistant aufgerufen und ausgeführt werden konnte, verwendeten wir Postman, um die einzelnen Requests zu testen. Da Postman auch schon beim Praxisprojekt eine grosse Hilfe war und die Software auch in den Vorlesungen vorgestellt wurde, haben wir uns wiederum für diese API Entwicklungsumgebung entschieden.

Dialogflow

Das von Google aufgekaufte und weiterentwickelte Tool ist gemäss der eigenen Webseite das meistverwendete Werkzeug, um Aktionen für Google Assistant Geräte zu erstellen. Obwohl uns die Anbindung viel Zeit kostete, sind wir mit dem Resultat sehr zufrieden, können wir doch nun das eShop-System mittels Spracheingaben auf dem Handy steuern und ausführen.

Hibernate

Dank dem Framework Hibernate von JBoss können gewöhnliche Objekte mit Attributen und Methoden in einer relationalen Datenbank gespeichert werden. Zudem ermöglicht Hibernate das Erstellen von Objekten aus den entsprechenden Datensätzen.

Javax.mail

Da die Bestellbestätigung respektive der Lieferschein per E-Mail versendet werden soll, ist das Package javax.mail im Inventory-Microservice importiert und verwendet worden. Die Verwendung dieses Packages erleichtert die Erstellung von E-Mail-Nachrichten mit Anhängen (in unserem Fall der Lieferschein im PDF-Format), die über die gesicherte SSL-Verbindung versendet werden sollen.

iText PDF

Zur Erzeugung der PDF-Dateien haben wir uns für die freie Programmbibliothek iText entschieden, welche das Erstellen unseres Lieferscheines in übersichtlichem und gut verständlichem Code erledigt. Die iText-Bibliothek ermöglicht es zudem, auch Metadaten mitzugeben, wie zum Beispiel den Titel und den Autoren der PDF-Datei.

Swagger 2

Das in der Vorlesung besprochene Framework Swagger haben wir ebenfalls in dieser Arbeit eingesetzt, um für jeden Service eine interaktive API-Dokumentation zu erstellen. Diese können unter den folgenden Links abgerufen werden:

E-Shop: https://shopstantly-eshop.herokuapp.com/swagger-ui.html#/
Inventory: https://shopstantly-inventory.herokuapp.com/swagger-ui.html#/
Payment: https://shopstantly-payment.herokuapp.com/swagger-ui.html#/
Shipment: https://shopstantly-shipment.herokuapp.com/swagger-ui.html#/

Fazit

Die Bearbeitung der Semesterarbeit und somit das Erstellen des «ShopStantly»-E-Shops hat uns stark gefordert und hatte dementsprechend einen sehr hohen Zeitaufwand zur Folge. Trotz den vielen und sehr ausführlichen Erklärungen während den Vorlesungen mussten viele Stunden für die Recherche und Planung eingesetzt werden.

Auch die Umsetzung war sehr zeitaufwändig, wobei vor allem das Zusammenspiel zwischen den einzelnen Microservices und das Deployment auf Heroku zum Knackpunkt wurden. Aber auch bei der Anbindung an den Google Assistenten via Dialogflow stiessen wir zum Teil an unsere Grenzen.

Nach dem Trial-and-Error-Prinzip wurden diverse Versuche gestartet, bis schlussendlich die Applikation auf den Smartphones via Google Assistant getestet werden konnte. Umso grösser war dann die Freude beim Ausprobieren und Testen, wobei nichts unversucht blieb. Dabei merkten wir jedoch schnell, dass es wohl noch einiges mehr braucht, bis ein solcher Onlineshop komplett über einen virtuellen Assistenten kommunizieren kann, wobei vor allem unvorhersehbare User-Anfragen das System empfindlich treffen und zum Abbruch der Transaktion führen können.

Schlussendlich hat uns dieses Projekt viel gelehrt und spannende Einblicke in die Praxis des «Enterprise Application Integration»-Alltags gegeben.