Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw School of
Management and Law

eUmzug Kanton Bern

Aufgabe 10: Einbinden Zahlungsanbieter

Modul Geschäftsprozessintegration im BSc Wirtschaftsinformatik

Autor
Björn Scheppler
scep@zhaw.ch

Herausgeber
Zürcher Hochschule für Angewandte Wissenschaften
School of Management and Law
Institut für Wirtschaftsinformatik (IWI)
Stadthausstrasse 14
CH-8041 Winterthur
www.zhaw.ch/iwi

Vorliegende Version
1.0.0 (5.12.2018)

Änderungshistorie

Version (Datum)	Wer	Was
1.0.0 (5.12.2018)	scep	Initiale Fassung

Hinweis: Aus Gründen der Lesbarkeit wurde im Text stets die männliche Form gewählt.

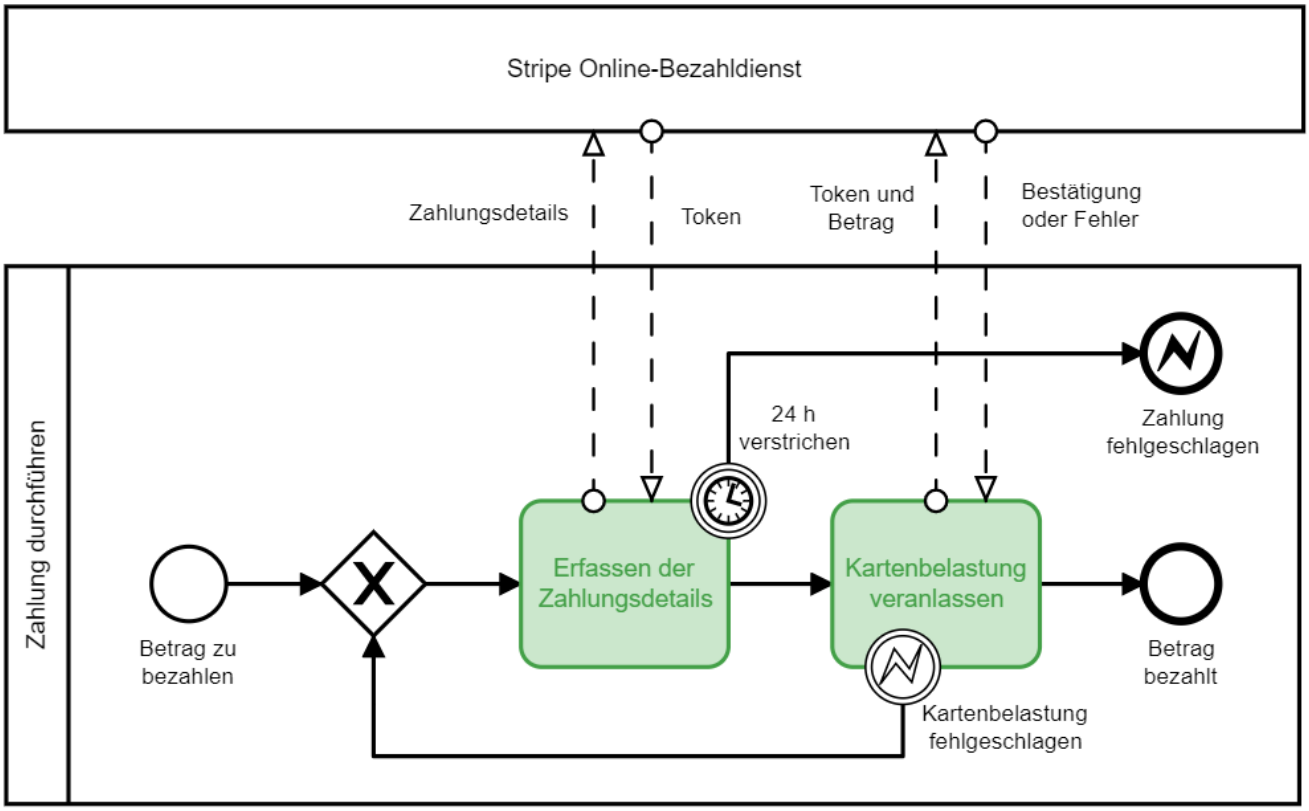
1 Zielsetzung

1. Am Beispiel des Online-Zahlungsanbieters Stripe und der Umzugsplattform habt ihr **gelernt**, …
   a. … wie **IT-Systeme** in Prozessapplikationen **eingebunden** (= integriert) werden,
   b. … über programmiersprachen-spezifische **API** von Drittanbietern.
2. In der Umzugsplattform habt ihr den Prozess **Zahlung durchführen** implementiert.

2 Vorgeschlagenes Vorgehen

2.1 Zusammenhänge verstehen und Vorbereitungsarbeiten

1. Besucht die **Kleinklasse** in der Semesterwoche 12. Dort erläutert Björn nebst der **Aufgabenstellung**, wie **Online-Zahlungen** allgemein funktionieren (anhand von hier und hier), wie **Stripe** im Speziellen funktioniert (anhand von hier) und wie dies im Kontext der **eUmzugPlattform** geschieht (anhand vom BPMN-Modell unten und einer Live-Demo).



2. Eine Person in der Gruppe erstellt ein **Stripe-Konto** und beschafft sich die für die Integration erforderlichen **API Keys**:
   a. Auf stripe.com neues Konto erstellen zwingend mit der **students.zhaw.ch-Mail-Adresse** und einem **Passwort**, das ihr **in keinem anderen Dienst** verwendet, da ihr es sowohl euren Gruppenkollegen als auch Björn aushändigen werdet.
   b. Man wird auf https://dashboard.stripe.com weitergeleitet
   c. **NICHT** auf "Activate your account" klicken

   d. In der **API Keys**-Seite die beiden Schlüssel (publishable und secret) herauskopieren
   e. Etwas später kommt noch ein Mail. Den **Bestätigungslink** in der erhaltenen Mail ausführen.

3. Macht euch mit dem **Stripe Zahlungs-Modul** vertraut:
   a. Lest das **QuickStart-Tutorial** für (Kredit)kartenzahlungen hier und die Checkout-Dokumentation im Speziellen hier.
   b. Auch wenn ihr es vermutlich jetzt nicht bereits testet, ist doch wichtig, dass ihr den **Mechanismus versteht**, da in den folgenden Kapiteln lediglich ein paar Tipps/Anforderungen im Kontext von unserer Prozessapplikation gegeben werden, auf alles andere müsst ihr mit dem Quickstart-Tutorial, den von dort verlinkten Seiten oder der vollständigen API-Referenz kommen. Letzterer hier ohne Verlinkung, da je nach PDF-Viewer sonst der Inhalt heruntergeladen statt angezeigt wird: https://stripe.com/docs/api?lang=java. Fürs Verständnis: Wir nutzen die Java API, daneben gibt es zahlreiche weitere programmiersprachen-spezifische APIs. All diese APIs sind lediglich Wrapper für die REST API, welche in der Referenz gefunden wird, wenn man auf curl umstellt.
   c. Im Tutorial werden verschiedene Varianten in Step 1 aufgeführt: **Checkout** ist die für uns relevante.
   d. Bei Checkout gibt es zwei Varianten, **Custom** ist die für uns relevante, da wir das token in eine Prozessvariable mit der Bezeichnung **stripeToken** schreiben wollen, dies aber bei Simple nicht möglich ist.

2.2 Client-seitige Arbeiten

1. Lest genau die Element Documentation bei der Aktivität **ErfassenDerZahlungsdetails**. Da diese nicht länger als 4000 Zeichen sein darf 😊, musste ich einen Teil hier auslagern, daher alles zu JavaScript hier aufgeführt:
   a. Einbinden der **Javascript-Bibliothek von Stripe**, welche die ganze client-seitige Checkout-Funktionalität enthält -> ist allerdings nicht erforderlich, da bereits in BE Services Plattform integriert. Bei Interesse siehe hier: https://github.com/zhaw-gpi/be-services-plattform/blob/master/src/main/resources/META-INF/resources/app/tasklist/scripts/config.js
   b. Die folgenden **Prozessvariablen** wie üblich laden und AngularJS-Scope-Variablen zuweisen: feeMap, feesTotal, emailAddress und chargingErrorMessage
   c. Für Checkout-Formular einen String zusammensetzen, welcher die **Gebührenbeschreibungen** aus feeMap enthält. Beispiel: Die feeMap enthalte zwei Einträge "Wegzugsgebühr Lohnsdorf" und "Zuzugsgebühr Kallnach". Dann soll die Gebührenbeschreibung "Wegzugsgebühr Lohnsdorf & Zuzugsgebühr Kallnach" sein.
   d. Stripe-**Checkout-Formular konfigurieren**: Öffentlicher Schlüssel, Bild [fakultativ], Sprache, Name, Währung, Text auf Bezahl-Schaltfläche
   e. **Callback-Funktion** bei erfolgreichem Erhalten eines Tokens, welche die Erfolgreich-Meldung anzeigt, alle anderen Meldungen sowie die Schaltfläche "Zahlungsdetails erfassen" ausblendet und das Token in einer Scope-Variable speichert. Hinweis bezüglich dem Ein-/Ausblenden: Ihr könnt entweder mit ng-show/ng-hide arbeiten wie gewohnt, aber müsst damit leben, dass es eine gewisse Zeit braucht, nachdem das Stripe-Formular geschlossen wurde, bis diese Aktionen durchgeführt werden. Alternativ arbeitet ihr ganz klassisch mit document.getElementById('ID').style.display = "none", um etwas auszublenden und document.getElementById('ID').style.display = "block", um etwas einzublenden
   f. **EventListener-Funktion**, welche auf das Klicken auf die Zahlungsschaltfläche reagiert

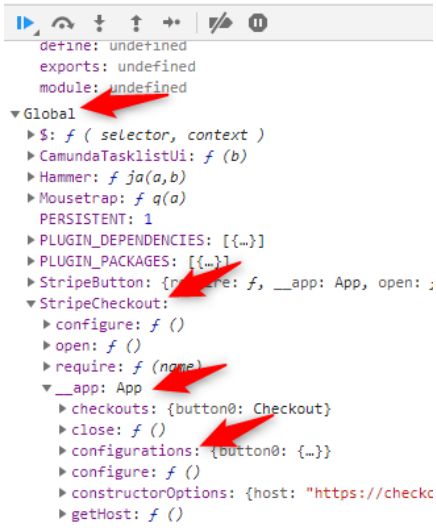
und dabei das Stripe Checkout-Formular öffnet

g. Funktion, welche **vor dem Submit des Formulars** den von Stripe erhaltenen Token, respektive dessen Id in einer neuen Prozessvariable stripeToken speichert und damit an die Process Engine übergibt. Wenn dieses Token hingegen nicht existiert, weil der Benutzer auf Abschliessen geklickt hat, ohne die Zahlungsdetails zu erfassen, dann soll hingegen die entsprechende Fehlermeldung dem Benutzer angezeigt werden und das Formular nicht submitted werden. Dies gelingt, wenn man der Funktion in camForm.on('submit' noch einen Parameter, wie z.B. e für event mitgibt und dann e.submitPrevented = true; setzt.

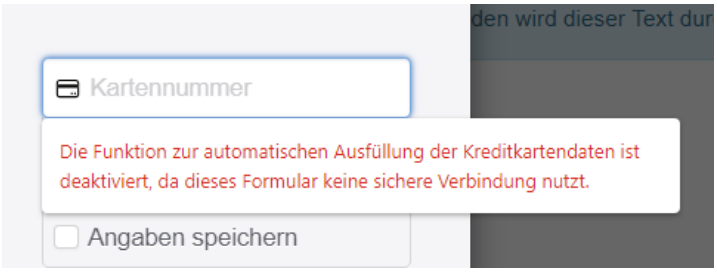
2. Konfiguriert den **UserTask** und erstellt eine **HTML-Datei** entsprechend der Vorgaben aus der Element Documentation.

3. Testet mit dem **soapUI**-Testdatensatz bei **BeforeAlleAngabenPruefen** mit an geeigneten Stellen gesetzten **debugger**; und **Testkreditkartenangaben** hier, ob sich das Formular verhält, wie es sollte und ob ihr beim Absenden eine Prozessvariable stripeToken habt, die ein von Stripe erhaltene Token erhält.

   a. Falls ihr sehen wollt, welche Eigenschaften das StripeCheckout-Objekt hat, dann sucht hier:



   b. Die untenstehende Fehlermeldung könnt ihr ignorieren:



   c. Prüft beim Testen z.B. auch solche Kartennummern, welche eine CVC-Verletzung auslösen oder gebt ein Ablaufdatum in der Vergangenheit ein, usw.

2.3 Server-seitige Arbeiten

Lest genau die **Element Documentation** bei der Aktivität **KartenbelastungVeranlassen** und setzt die **server-seitigen Anforderungen** um, wie dort gefordert.

2.4 Umfassendes Testen und Abgabe

1. Testet mit verschiedenen **Testkreditkartenangaben** hier, ob sich alle Komponenten (Client, Delegate, Service, Prozessfluss) so verhalten, wie gemäss BPMN-Modell und Element Documentations gefordert. Ein paar Hinweise hierzu:
    a. Es gibt hier **beliebig viele Testkreditkarten** mit allen möglichen Varianten, natürlich müsst ihr die nicht alle durchtesten, zumal sie auch nicht alle für unseren Kontext sinnvoll sind, aber prüft mindestens auch eine, welche eine StripeException auslöst.
    b. Beachtet auch, dass nicht alle unter «Testing for specific reasons and errors» in einer Exception münden, sondern teilweise lediglich gewisse **Attribute im Charge-Objekt** anders setzen (z.B. den Risk Level). Dies könnt ihr prüfen, wenn ihr im Debug-Modus die Variable Charge genauer untersucht.
    c. Andere dieser Testkreditkarten wiederum wirken sich bereits **formular-seitig** aus, so dass man also gar kein Token erhält.
    d. Nutzt hier wiederum den soapUI-Testdatensatz **BeforeAlleAngabenPruefen**. Macht die Tests vorzugsweise im Debug-Modus, um allfällige Fehler einfacher zu entdecken.
    e. Seid nicht erstaunt, dass es **einige Sekunden dauern** kann, bis ihr eine Antwort vom Stripe-Server erhaltet, denn das Ganze wird ja synchron aufgerufen.
    f. Prüft dabei auf der **Stripe-Events-Seite** hier, ob Buchungen gelingen. Stellt sicher, dass mindestens eine Buchung erfolgreich war. Schaut euch auch die **Payments-Seite** hier an für Details zu den Zahlungen. Und schaut euch unter **Balance-Transactions** hier an, was euch tatsächlich nach Abzug der Gebühren an Einnahmen bleibt.
2. Gebt die Aufgabe gemäss den Vorgaben im nächsten Kapitel gleichzeitig mit Aufgabe 9 ab.

3 Abgabe

1. Die inhaltliche Abgabe ist ein Commit und Push in mindestens den zwei Github-Repositories VeKa und Umzugsplattform der Gruppe als auch eventuell des GWR, falls hier noch fortgeschrittene Anforderungen erstellt wurden.
2. Die formale Abgabe erfolgt auf Moodle hier durch EINE Person im Team.
3. Dies erfolgt spätestens bis zum im Betreff der verlinkten Aufgabe aufgeführten Zeit.
4. Die Abgabe besteht darin, die Ids der zu korrigierenden Commits und der Zugangsdaten zu Stripe als Text einzufügen wie folgt und auf «Abgabe einreichen» zu klicken.
    a. Variante ohne fortgeschrittene Anforderungen

Umzugsplattform: [Commit ID]
Veka: [Commit ID]
Benutzername Stripe: [Studentenkuerzel]@students.zhaw.ch
Passwort Stripe: [Passwort]

    b. Variante mit fortgeschrittenen Anforderungen in der Maximalvariante, dass in allen Repositories solche vorhanden sind und dass die fortgeschrittenen Anforderungen teilweiese in einem separaten Branch seien.

Umzugsplattform: [Commit ID] (Fortgeschrittene Anforderungen in Branch [Branch-Bezeichnung])
Veka: [Commit ID] (Fortgeschrittene Anforderungen)
GWR: [Commit ID] (Fortgeschrittene Anforderungen in Branch [Branch-Bezeichnung])
Benutzername Stripe: [Studentenkuerzel]@students.zhaw.ch
Passwort Stripe: [Passwort]

4 Bewertungskriterien

4.1 Kriterien

Maximal können **10 Punkte** erreicht werden, die sich auf die folgenden **Kriterien** aufteilen:

1. **Benutzerkonto** angelegt und dessen Zugangsdaten bekanntgegeben sowie korrekt angepasstes **BPMN-Modell** (User und Service Task korrekt konfiguriert) (**1 Punkt**)
2. Implementation des Formulars **ErfassenDerZahlungsdetailsForm** inklusive JavaScript entsprechend der Anforderungen aus der Element Documentation (**4.5 Punkte**)
3. Implementation der JavaDelegate-Klasse **CreateChargeDelegate** entsprechend der Anforderungen aus der Element Documentation (**1.5 Punkte**)
4. Implementation der Klasse **StripeClientService** entsprechend der Anforderungen aus der Element Documentation inklusive angepasstem pom.xml (**2 Punkte**)
5. Mindestens einen erfolgreichen und einen fehlgeschlagenen Zahlungseintrag in der **Stripe Events**-Seite (**0.5 Punkte**)
6. **Eleganz** und **Robustheit** des Codes: Erläuterungen bei Bedarf in Aufgabe 6 (**0.5 Punkte**)

4.2 Aufwandreduktion 4er-Gruppen

Keine Aufwandreduktion

4.3 0 Punkte

Keine Abgabe innerhalb der Abgabefrist auf Moodle und Github.

4.4 Punkteabzug

Für die folgenden formalen Fehler und fehlenden/falschen Dokumentationen gibt es die Abzüge auf die erreichte Punktzahl gemäss Kapitel 4.1. Diese Abzüge werden aufsummiert, wobei eine Gruppe nicht weniger als 0 Punkte erreichen kann.

1. **Keine fristgerechte Abgabe auf Moodle**, aber auf Github eine Lösung gepusht: 4 Punkte
2. **Auf Moodle formal eine falsche Abgabe** (z.B. keine vorhandene Commit-ID, etwas anderes als eine Commit-ID, usw.): 2 Punkte
3. Java-Klassen in falsch(en) (bezeichneten) **Packages**: 1 Punkt
4. Java-Klassen, Methoden und Variablen **falsch benannt** (Klassen anders als vorgegeben [soweit vorgegeben] sowie Gross-/Kleinschreibung): maximal 2 Punkte
5. Klassen und Methoden ohne aussagekräftige Java Documentation (**JavaDoc**). Enthält eine Klasse nur eine Methode, so reicht eine Java Documentation der Klasse: maximal 4 Punkte
6. Jedes Java-Statement (mit Semikolon beendet) sowie if-then-else- oder for-Blöcke enthlaten einen oder mehrere zutreffende, selbst formulierte, deutschsprachige **Zeilenkommentare**. Davon ausgenommen sind Standard-Getter- & Setter-Methoden sowie wenn direkt untereinander mehrfach dasselbe gemacht wird (z.B. Auslesen von vier Prozessvariablen in lokale Variablen). Verletzung dieser Regel: maximal 4 Punkte.

5 Feedback

Spätestens am 25.12.2018 10:00 ist die **Musterlösung** unterhalb der Aufgabenstellung auf Moodle freigeschaltet. Spätestens am 6.1.2019 20:00 sieht diejenige Person, welche die Aufgabe eingereicht hat, die erreichte Punktzahl pro Kriterium und insgesamt inklusive allfälliger Kommentare, wenn sie erneut auf die **Moodle-Aufgabe** klickt.

6 Fortgeschrittene Anforderungen (optional)

6.1 Mögliche zu implementierende Anforderungen

6.1.1 Buchhaltungsabteilung informieren (vermutlich eher einfach bis vermutlich eher komplex)

Ausgangslage: Im realen Umzugsprozess muss die Buchhaltungsabteilung des Kantons regelmässig die eingenommenen Gebühren an die betroffenen Gemeinden weiterleiten. Aktuell müsste die Buchhaltungsabteilung hier aktiv daran denken, dass dies nicht vergessen geht sowie manuell ins Dashboard von Stripe einloggen und zum Beispiel auf dieser Seite einen Export der Zahlungen machen. Dies ist umständlich und unzuverlässig.

Anforderung: Langfristig würde man das Übertragen der Einnahmen zu den Gemeinden sicher automatisieren, aber als Zwischenschritt entschied der Kanton, dass immer am ersten Tag jeden Monats automatisch eine Liste aller Zahlungseingänge per Mail bei der Buchhaltungsabteilung eintreffen soll, wobei jeder Listeneintrag die Angaben zur Gebührenhöhe pro Gemeinde enthält, also z.B. «15.11.2018, Lohnsdorf, CHF 10.00».

Details und Lösungsvorschlag:

1. Um den Prozess monatlich auszulösen, erstellt ihr am einfachsten ein neues BPMN-Modell mit einem Prozess, der als Startereignis ein entsprechend konfiguriertes Zeitereignis verwendet.
2. Wie Mails versendet werden, kennt ihr aus dem Twitter-Review-Prozess. Als Mail-Adresse zum Testen, wählt ihr eure eigene als Absender und Empfänger.
3. Den StripeClientService müsst ihr so erweitern, dass er nebst Gebührenhöhe insgesamt und dem Token auch eure feeMap entgegennimmt und aus dieser eine sinnvolle description an die Charge-Parameter übergibt oder sogar noch sinnvoller die metadata-Eigenschaft setzt.
4. Um die Liste der Zahlungseingänge zu erhalten, nutzt die list-Ressource von charges. Erstellt dabei auch einen Filter, indem ihr created so setzt, dass nur die Zahlungsvorgänge des letzten Monats berücksichtigt werden. Fürs Testen müsst ihr hier natürlich andere Filter setzen, damit ihr überhaupt etwas zurückerhaltet.
5. Wenn ihr die Liste der Zahlungseingänge habt, dann sollen nur diejenigen in das Mail mit aufgenommen werden, die erfolgreich waren, also den Status succeeded haben.
6. Speichert einen Screenshot eines dieser Mails im Ordner src\test\resources\mailScreenshot.jpg oder png als Beweis, dass es bei euch funktioniert.

6.2 Abgabe

Die Regeln in Kapitel 5.2 der Hauptaufgabestellung einhalten. Die Abgabe erfolgt am 24.12.2018 gleichzeitig mit den Basis-Anforderungen. Details siehe Kapitel 3.