Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw School of
Management and Law

Twitter-Review-Prozess

Schritt-für-Schritt-Anleitung

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.iwi.zhaw.ch

Vorliegende Version
1.4.0 (21.11.2018)

Änderungshistorie

Version (Datum)	Wer	Was
1.0.0 (13.9.2018)	scep	Initiale Fassung
1.1.0 (28.9.2018)	scep	Kleinere Korrekturen nach Durchspielen
1.2.0 (5.10.2018)	scep	Link zu Video-Playlist ergänzt
1.3.0 (15.10.2018)	scep	Kapitel 7 ergänzt
1.4.0 (21.11.2018)	scep	Kapitel 9 ergänzt

Inhaltsverzeichnis

Vorbemerkungen

Diese Anleitung ist in erster Linie ein oberflächliches Drehbuch für Björn, damit er den Kleinklassen-Unterricht durchführen kann. Aber allenfalls hilft es auch einzelnen Studierenden, die entweder den Unterricht verpassten oder noch das eine oder andere wiederholen wollen.

Die Ergebnisse verschiedener Schritte sind im Github-Repository commited und mit Tags (pro Kapitel einer) versehen. Jeder einzelne Tag kann separat ausgecheckt werden. Zudem steht eine Video-Playlist zur Verfügung.

1 Basis-Maven-Projekt auschecken und konfigurieren

1. Fork von https://github.com/zhaw-gpi/twitter-review-prozess ins eigene Github-Konto.
2. Clone des geforkten Repository in Netbeans.
3. Nur Björn: Clean&Build > Run > Demo, was das fertige Resultat sein wird von dieser und nächster Woche.
4. Auschecken von Kapitel-1 in einem neuen Branch «KLEINKLASSE-work» (KLEINKLASSE durch vza, vzb, usw. ersetzen).
5. Ansichten «Projects» und «Files» erklären. Bei «Projects» Ansicht der «Java Packages» anpassen.
6. Kurz (!) die vorhandenen Ordner und Dateien erklären in Files-Ansicht (README.md, pom.xml, .gitignore, TwitterReviewApplication.java, application.properties, banner.txt, processes.xml, twitter-review.mv.db und target-Verzeichnis).
7. camunda-license.txt erstellen:

```
---------- BEGIN CAMUNDA BPM LICENSE KEY ----------
fO/l8YyIdjG5vZXXg9ZgaJ3Z0JoaWMDo6iE1Sn0YnsmHGp4D64V9Wa2kHzUkTrRuEuG1NRViLISzy5uiRUhwgcWjAsShtgHLOeSCB2C
5gLKCKQTxQvVOutVyYQM10kqzuzaeUljo6r2wwQREYiKHvn1i4BaiG5FMmbsJrFFNbgo=;zhaw_school_of_management_and_law;201
9-02-09---------- END CAMUNDA BPM LICENSE KEY ----------
```

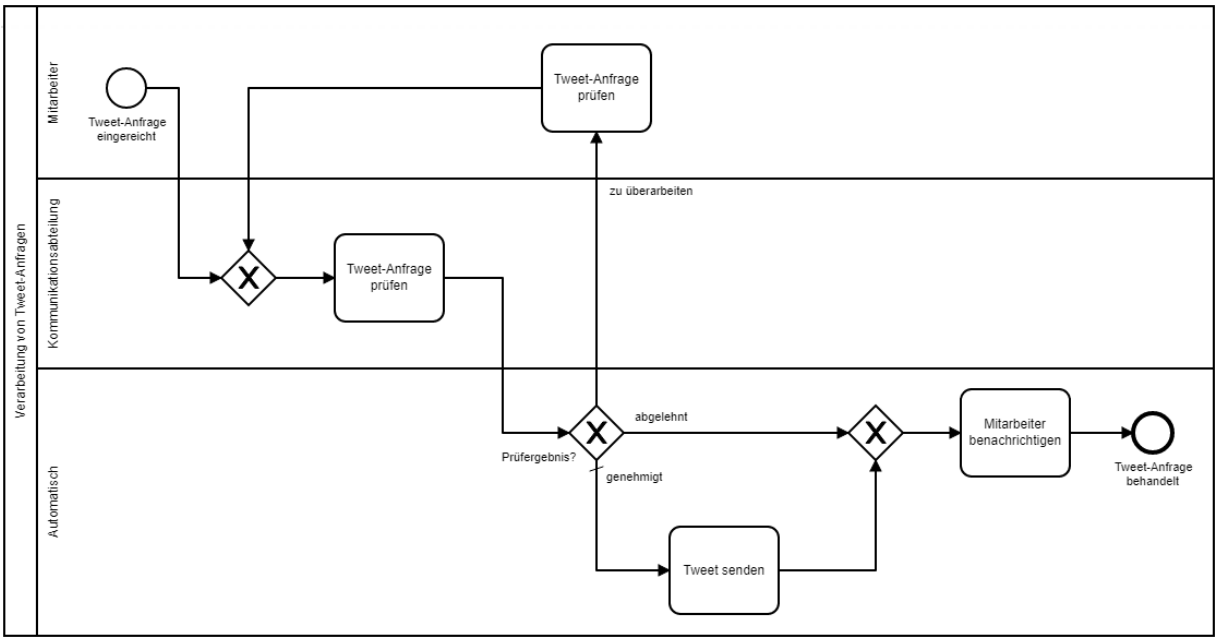
8. Nur Björn: Datenbank wieder löschen
9. settings.xml erstellen:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
 <servers>
  <server>
    <id>camunda-bpm-ee</id>
    <username>trial_scep_zhaw_ch</username>
    <password>476bb474-33a8-4cd5-8</password>
  </server>
 </servers>
</settings>
```

10. Clean and Build > Run
11. Grob erklären, was in der Console ausgegeben wird
12. Anmelden auf localhost:8080 > Tasklist > Start process > Keine Prozesse vorhanden
13. Stop

2 Fachliches Prozessmodell erstellen

1. Camunda Modeler: neues BPMN-Diagramm
2. Speichern unter src\main\resources\twitter-review.bpmn
3. Von https://raw.githubusercontent.com/zhaw-gpi/twitter-review-prozess/1005994e2a3f6e9e35655782bc53e72344f882a3/src/main/resources/twitter-review.bpmn den Text in die XML-Ansicht einfügen, damit das untenstehende Modell erstellt wird.



4. Executable-Eigenschaft erklären
5. Auf die Expresssions eingehen und Default Flow löschen
6. Speichern und erneut Run
7. Erklären, was in Console anders ist (bpmn gefunden, aber Fehler «Exclusive Gateway without condition")
8. Auf Process Id und Name mit CamelCase-Schreibweise eingehen, damit man Fehler besser versteht.
9. Default Flow herstellen
10. Speichern und Run > Anmelden > Tasklist > Start Process > Jetzt ist er da
11. Neue Instanz starten > Start drücken => Fehler, weil checkResult nicht bekannt => neue String-Variable mit Inhalt irgendwas (z.B. a) > Start > Erklären, was passiert ist anhand Cockpit-History-Ansicht
12. Nochmals eine Instanz starten, dieses Mal aber checkResult auf rejected setzen > Cockpit
13. Und nochmals, nun checkResult auf revisable setzen > Cockpit: Keine Reaktion, Tasklist: keine Reaktion, Windows-Task Manager: Netbeans (Java) verbraucht alle Rechner-Ressourcen. Grund?
14. Stop und erneut Run behebt das Problem, da die neue Prozessinstanz noch nicht persistiert wurde (kein Haltepunkt erreicht). Für Fortgeschrittene: Asynchronous after setzen und dann dasselbe nochmals versuchen.
15. Stop

3 User Tasks und Generated Forms

Kurz erklären, was generierte Forms sind und Ausblick auf Embedded Forms

3.1 Startereignis «Tweet-Anfrage eingereicht»

1. Initiator auf anfrageStellenderBenutzer setzen
2. Folgende form-Fields einfügen:
    a. email, String, E-Mail
    b. tweetContent, String, Zu veröffentlichender Text

3.2 User Task «Tweet-Anfrage prüfen»

1. Zu User Task umwandeln
2. Candidate Group auf kommunikationsabteilung setzen
3. Folgende form-Fields einfügen:
    a. Dito oben
    b. checkResult, enum, Prüfergebnis, 3 Werte
        i. accepted, Genehmigt
        ii. revisable, Zu überarbeiten
        iii. rejected, Abgelehnt
    c. checkResultComment, string, Hinweise für Mitarbeiter zum Prüfergebnis

3.3 Formular für «Tweet-Anfrage überarbeiten»

1. Um nicht alles neu zu erstellen, bisherigen Task «Tweet-Anfrage überarbeiten» löschen und durch Kopie von User Task «Tweet-Anfrage prüfen» ersetzen.
2. Canditate Group löschen und dafür Assignee auf ${anfrageStellenderBenutzer} setzen
3. Form field “email” und «checkResult» löschen.
4. Bei «checkResultComment» property «disabled» hinzufügen

3.4 Deploy und Testen

1. Speichern und erneut Run
2. In Tasklist die verschiedenen Varianten durchspielen, jeweils auch den Diagram-Tab anzeigen
3. In Cockpit Runtime/History zeigen, was passiert (ist).
4. Stop

3.5 Commit und Push

1. Zwischenstand in Github speichern:
    a. Commit «Bis und mit User Tasks»
    b. Push von KLEINKLASSE-work

4 Script Task erstellen

1. Es ist etwas dämlich, dass die E-Mail-Adresse des Mitarbeiters und nicht dessen Name oder Kürzel angezeigt wird.
2. In der Realität würde man das aus einem Active Directory oder ähnlichem auslesen.
3. Wir machen es etwas einfacher, indem wir das Kürzel aus den ersten vier Zeichen der Mail-Adresse extrahieren (just for fun, respektive um Script Tasks einführen zu können).
4. Neuer Script Task „Mitarbeiter-Kürzel extrahieren“ nach Startereignis einfügen vom Format JavaScript mit Code

```
emailAdress = execution.getVariable("email");
var alias = emailAdress.substring(0,4);
execution.setVariable("alias",alias);
```

5. User Task “Tweet-Anfrage prüfen”: Form Field email durch alias (Kürzel) ersetzen
6. Speichern und Run > Testen, ob es klappt
7. Stop

5 Service Tasks erstellen

Wie erwähnt, mocken wir vorläufig das Senden von Tweets und das Benachrichtigen des Mitarbeiters lediglich. Später im Semester wird's dann real.

5.1 Tweet senden

1. Zum Service Task umwandeln
2. Implementation «Delegate Expression»: #{sendTweetAdapter}, Asynchronous Before
3. Speichern
4. Neues Package "delegates"
5. Neue Klasse "SendTweetDelegate"
6. License Header löschen
7. Implements JavaDelegate
8. Falls Zeit bleibt: Auf Google suchen nach folgendem String, um zu zeigen, wie der Code dahinter aussieht:

```
site:github.com DelegateExecution.java
```

9. «Fehler» beheben durch «Implement all abstract methods»
10. «throw new…» löschen
11. Zunächst JavaDoc für Klasse ergänzen: «Implementation des Service Task "Tweet senden"»
12. @Named("sendTweetAdapter")
13. Dann JavaDoc für Methode erstellen:

```
* Mockt das Senden eines Tweeets
    *
    * 1. Die Prozessvariable tweetContent wird ausgelesen
    * 2. Dieser Text wird in der Console ausgegeben
    *
    * @param de          Objekt, welches die Verknüpfung zur Process Engine und zur aktuellen Execution enthält
```

14. Dies nun implementieren: Erste Zeile zunächst ohne Casting und JavaDoc anzeigen => JavaDoc herunterladen

```
String tweetContent = (String) de.getVariable("tweetContent");
System.out.println(tweetContent);
```

15. Speichern
16. Breakpoint setzen bei erster Code-Zeile
17. Debug
18. Neue Instanz starten, Formular ausfüllen, Genehmigen => Breakpoint wird ausgelöst
19. Zeigen, was der Inhalt der Variable de ist (z.B. activityName, processInstanceId, variableStore>variables>tweetContent (key/value)>textValue
20. F8 > Inhalt von tweetContent zeigen
21. F8 > Console zeigen
22. Cockpit > Runtime-Ansicht zeigen
23. F8 > Geht uns zu weit > Play-Symbol drücken
24. Ausgabe von nur tweetContent etwas irreführend => neu System.out.println("!!!!!!!!!!!!!!!! Folgender Tweet wird veröffentlicht: " + tweetContent);
25. «Apply Code Changes»-Symbol drücken -> eine Bestätigung sollte erscheinen
26. Breakpoint entfernen

27. Wie bei 18 und dann Console-Output anschauen
28. Stop

5.2 Mitarbeiter benachrichtigen

Gleiches Vorgehen mit folgenden Anpassungen:

1. Send Task statt Service Task (technisch identisch)
2. notifyEmployeeAdapter, auch Asynchronous Before
3. NotifyEmployeeDelegate
4. JavaDoc Klasse: Implementation des Send Task "Mitarbeiter benachrichtigen"
5. JavaDoc Methode:

```
Mockt das Senden einer Benachrichtigung per Mail
 *
 * 1. Die benötigten Prozessvariablen auslesen
 * 2. Die E-Mail-Nachricht zusammenstellen
 * 3. Die E-Mail in der Konsole ausgeben
```

6. Den eigentlichen Code aus Zeitgründen von Github (Raw-Ansicht) kopieren und nur erläutern
7. Speichern
8. Run
9. Zwei Instanzen, einmal genehmigen, einmal ablehnen
10. Cockpit und Console zeigen

5.3 Commit und Push

1. Zwischenstand in Github speichern:
    a. Commit «Bis und mit Service Tasks»
    b. Push von KLEINKLASSE-work

6 Zeitüberschreitung abfangen

Wenn der Mitarbeiter oder die Kommunikationsabteilung nicht innerhalb von 3 Tagen reagiert, soll die Instanz beendet werden. Wie geht das?

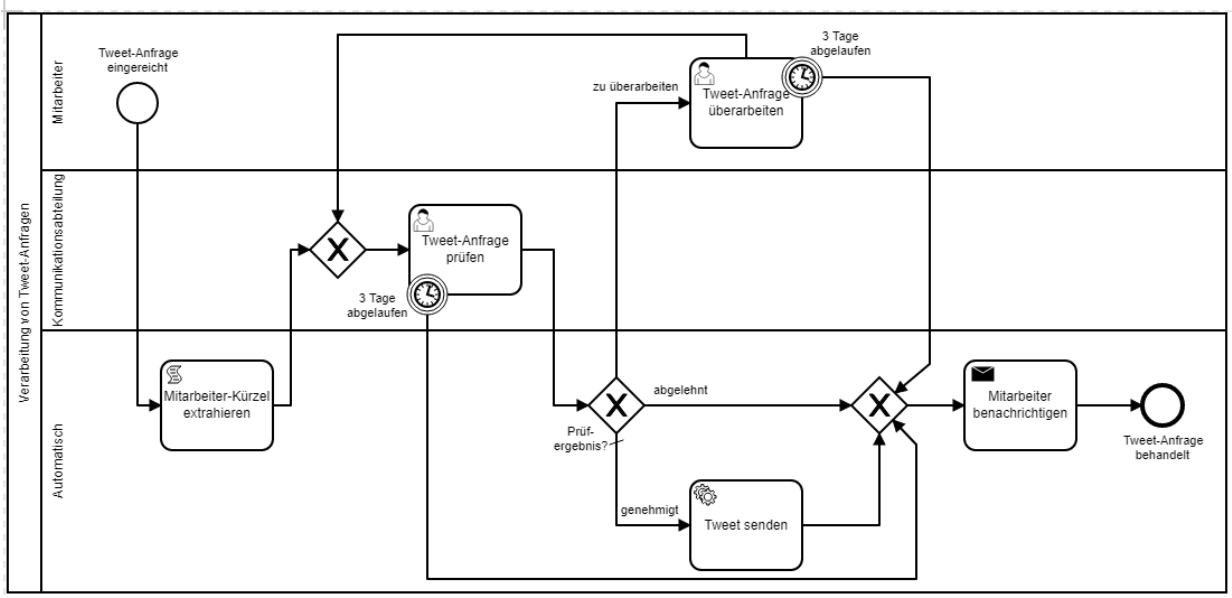
1. Da gibt es elegante Ansätze, wir wählen nun einen Quick&Dirty-Ansatz für dieses Demo-Beispiel:
2. Timer Boundary Event an «Tweet-Anfrage prüfen”:
    a. Bezeichnung «3 Tage abgelaufen»
    b. Verbindung zu XOR-Join vor «Mitarbeiter benachrichtigen»
    c. Definition Type: Duration
    d. Definition: PT10S (eigentlich wären es PT3T)
    e. End Listener JavaScript mit Code:

```
execution.setVariable("mailMainPart","Dieser konnte nicht veröffentlicht werden, weil er nicht durch die Kommunikationsabteilung innert 3 Tagen  geprüft wurde.");
```

3. Boundary Event für «Tweet-Anfrage überarbeiten» erstellen. Alle Eigenschaften identisch ausser Listener Code:

```
execution.setVariable("mailMainPart","Dieser konnte nicht veröffentlicht werden, weil Du nicht innert 3 Tagen diesen überarbeitet hast.");
```

4. Das Diagramm sieht dann wie folgt aus:



5. NotifyEmployeeDelegate anpassen:

```
Object mailMainPart = de.getVariable("mailMainPart");
if(mailMainPart instanceof String){
    mailHauptteil = (String) mailMainPart;
} else if(checkResult.equals("rejected")){
```

6. Run und neue Instanzen erstellen, aber ohne etwas zu tun. Nach 10 bis 60 Sekunden sollte das Mail in der Konsole ausgegeben werden.

6.1 Commit und Push

2. Zwischenstand in Github speichern:
    a. Commit «Bis und mit Zeitüberschreitung abfangen»
    b. Push von KLEINKLASSE-work

7 Testing

Zu diesem Kapitel wird kein Video erstellt. Peter wird dies in der Grossklasse vermitteln und sofern die Zeit nicht ausreicht, Björn in der Kleinklasse abschliessen.

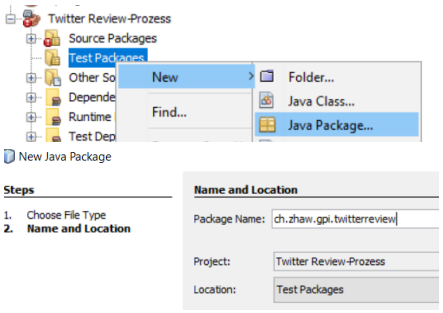
7.1 Vorbereitungsarbeiten

Der **Code** zu diesem Kapitel ist im Tag «kapitel-7-1» enthalten.

1. pom.xml um **Dependencies** fürs Testen ergänzen:

```
<!-- Importiert Komponenent fürs Testing. Die dritte Dependency ist notwendig,
    weil die standardmässig geladene 3.9.1-Version einen Bug verursacht. Details siehe
    https://github.com/camunda/camunda-bpm-assert/issues/90 -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.camunda.bpm.extension</groupId>
      <artifactId>camunda-bpm-assert</artifactId>
      <version>2.0-SNAPSHOT</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.assertj</groupId>
      <artifactId>assertj-core</artifactId>
      <version>1.7.1</version>
      <scope>test</scope>
    </dependency>
```

2. Im pom.xml die Eigenschaft **skipTests** auf false setzen.
3. In Test Packages ein neues Package anlegen *ch.zhaw.gpi.twitterreview*:



4. Innerhalb dieses Packages eine **TestConfiguration**-Klasse erstellen, um eine neue In-Memory-ProcessEngine zu konfigurieren. Code siehe hier.
5. Innerhalb dieses Packages eine **TwitterReviewProcessTest**-Klasse erstellen mit folgenden Elemente (Code siehe hier):
    a. **Annotations** (@RunWith und @ContextConfiguration)
    b. Verwendete **Konstanten** definieren (nicht zwingend, aber sauberer)
    c. **Autowiring** der benötigten ProcessEngine-Services
    d. Anlegen der drei durchzuführenden **Unit Tests mit Zeilenkommentaren**, damit man weiss, was man im folgenden Kapitel konkret testen will. Typischerweise wären die Assertions in Testfällen beschrieben, welche man als Basis für die Unit Tests nimmt.

7.2 Eigentliche Tests

Der **Code** zu diesem Kapitel ist im Tag «kapitel-7-2» enthalten.

1. Die **Zeilenkommentare abarbeiten** mithilfe der BPM-Assert-Dokumentation und der JobExecutor-Dokumentation. Für den fertigen Code siehe hier.
2. Nun etwas rumspielen mit verschiedenen Varianten, um die Tests auszulösen:
    a. **Clean & Build**: Da skipTests = false, werden auch beim Build-Prozess Tests durchgeführt, um keinen fehlerhaften Code in ein produktives System zu deployen.
    b. Rechtsklick auf **TwitterReviewProcessTest**-Klasse**:**
        i. Auf **Test File** klicken, um nur die Tests auszuführen ohne Build-Prozess
        ii. **Breakpoint** an gewünschter Stelle setzen und dann auf **Debug Test File** klicken
3. Bewusst eine **Assertion unsinnig setzen**, um zu sehen, wie es aussieht, wenn ein Test fehlschlägt (z.B. in Zeile 115 isActive statt isEnded).

8 Spring Framework für Twitter und Mail-Versand

8.1 Zielsetzung

8.1.1 Lernziele

- Passend zur Grossklasse von SW 7 das Spring Framework besser verstehen
- Suchstrategie für das Lösen von konkreten Herausforderungen kennen
- Integration von Umsystemen über APIs an zwei Beispielen sehen

8.1.2 Ziele für den Twitter-Review-Prozess

- Echter Tweet posten statt nur Ausgabe in Konsole
- Echtes Mail senden statt nur Ausgabe in Konsole

8.2 Tweets auf Twitter posten

8.2.1 Wie finden wir heraus, wie wir dies machen

1. Sinnvolle Google-Suche: *spring twitter getting started*
2. projects.spring.io erläutern (Sample Projects auf Github, Getting Started Guides, Reference, API)
3. Baeldung erläutern
4. Unser Kontext: Spring Boot(!)

8.2.2 Library im Projekt verfügbar machen

1. pom.xml erweitern um

```
<!-- Import die Spring Social Twitter-Komponenten -->
    <dependency>
      <groupId>org.springframework.social</groupId>
      <artifactId>spring-social-twitter</artifactId>
      <version>1.1.2.RELEASE</version>
    </dependency>
```

2. Leider auch folgende Dependency erforderlich:

```
<!-- Überschreibt die Version von Spring-Web, welche bereits im Spring Boot Starter
        enthalten ist. Grund ist ein im aktuellen Release nicht gelöster Bug,
        beschrieben z.B. in https://stackoverflow.com/questions/50853639/twitter-template-fails-during-sending-direct-message-but-get-rate-limit-request -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>5.0.4.RELEASE</version>
    </dependency>
```

3. Clean & Build
4. PS: Tests deaktivieren für schnelleren Build: <skipTests>true</skipTests>

8.2.3 Zugangsdaten für Twitter

1. Bereits ein Konto erstellt für uns. Bei Interesse für ein eigenes: https://spring.io/guides/gs/register-twitter-app/
2. Dürfen auf keinen Fall auf github veröffentlicht werden => zwei Varianten:
    a. Umgebungsvariablen (werden wir bei Mail senden so machen)
    b. Eigenes properties-File statt application.properties und .gitignore (unsere Wahl)
3. Neu: twitter.properties
4. Git > ignore
5. Inhalt:

```
# Twitter Zugangsdaten
twitter.consumerKey=ljqUPQQ3YkbzVTxokXDPlrQT8
twitter.consumerSecret=v4M3Ln5Q2rgRjmjmwiUZDg0gvpLpqidLnuWPGfPuUgfcVVw1w2
twitter.accessToken=1025006426685616129-zro50TMbzcGKRAjam8rG9EHn7vg2wP
twitter.accessTokenSecret=M4E5EVhuDDnZLDOB8bEidG7wteOu4DuJzcp48ibbyvUa4
```

6. Dieses File wird ignoriert von Spring, da nur application.properties standardmässig geladen wird.
7. Abhilfe:
    a. Neues Package *configuration* mit einer neuen Klasse *ApplicationConfiguration*
    b. *@Configuration* und *@PropertySource("classpath:twitter.properties")* als Annotationen für die Klasse
    c. JavaDoc hinzufügen:

```
/**
 * Verschiedene Einstellungen für die Spring-Applikation
 *
 * @Configuration stellt sicher, dass Spring diese Klasse als Einstellungs-
 * Klasse erkennt und damit beim Starten berücksichtigt
 *
 * @PropertySource("classpath:twitter.properties") stellt sicher, dass die Werte
 * aus dieser Datei über @Value-Annotationen ausgelesen werden können
 *
 * @author scep
 */
```

8.2.4 Service-Klasse für Kommunikation mit Twitter

1. JavaDelegate *SendTweetDelegate* ist nur das Bindeglied zwischen Process Engine und einem neu zu erstellenden Service *TwitterService*. Gründe:
    a. Falls mehr als ein Delegate mit Twitter kommunizieren will, wäre sonst Code mehrfach vorhanden.
    b. Die Verbindung zu Twitter würde bei jedem Aufruf neu hergestellt (Singleton).
    c. Die Delegate-Klasse würde überfrachtet/schwer wartbar.
2. Neues Package *services* und Klasse *TwitterService*
3. Annotation *@Service*
4. JavaDoc:

```
/**
 * Service-Klasse für die Kommunikation mit der Twitter API
 *
 * @Service stellt sicher, dass diese Klasse beim Starten der Applikation als
```

```
 * Bean generiert wird (wie bei @Component und @Bean), aber als Service gekennzeichnet ist
 *
 * @author scep
 */
```

5. Klassen-Variablen deklarieren:

```
// Variable für ein Twitter-Objekt, welches die Verbindung zur Twitter API beinhaltet
    private Twitter twitter;

    // Variable für ein TimelineOperations-Objekt (Sub-API für Timeline lesen und Tweets posten)
    private TimelineOperations timelineOperations;
```

6. Authentifizierungs-Angaben auslesen und in Variablen speichern

```
// Werte aus twitter.properties für die Authentifizierung bei der Twitter API
    @Value("${twitter.consumerKey}")
    private String consumerKey;
    @Value("${twitter.consumerSecret}")
    private String consumerSecret;
    @Value("${twitter.accessToken}")
    private String accessToken;
    @Value("${twitter.accessTokenSecret}")
    private String accessTokenSecret;
```

7. Neue Methode mit folgendem Code:

```
/**
     * Verbindung zur Twitter API aufbauen beim Initialisieren dieser Klasse
     *
     * @PostConstruct (und die init-Bezeichnung) stellen sicher,
     * dass die Methode erst ausgeführt wird, wenn die Klasse bereits konstruiert
     * wurde und daher z.B. @Value bereits ausgeführt wurde
     */
    @PostConstruct
    private void init() {
        // Versuchen, die Verbindung mit Twitter aufzubauen
        try {
            // Anmelden bei Twitter API über die Zugangsdaten
            twitter = new TwitterTemplate(consumerKey, consumerSecret, accessToken, accessTokenSecret);

            // TimelineOperations-Objekt der entsprechenden Variable zuweisen
            timelineOperations = twitter.timelineOperations();

            // In Konsole erfolgreiche Anmeldung ausgeben
            System.out.println("Anmeldung bei Twitter erfolgreich. Angemeldeter Benutzer: "
                    + twitter.userOperations().getScreenName());
        } catch (ApiException e) {
            // Falls fehlgeschlagen, dann Fehlermeldung in Konsole ausgeben
            System.err.println("Anmeldung bei Twitter fehlgeschlagen. Meldung: " + e.getLocalizedMessage());
        }
    }
```

8. Neue Methode mit folgendem Code:

```
/**
     * Postet einen neuen Tweet auf Twitter (= updateStatus)
     *
     * @param statusText Der zu postende Text
```

```
     * @throws java.lang.Exception
     */
    public void updateStatus(String statusText) throws Exception {
        // Versucht, den Tweet zu posten oder "behandelt" Fehler durch Ausgabe von Meldungen in der Konsole
        try {
            // Tweet posten
            timelineOperations.updateStatus(statusText);

            // Als Bestätigung in der Output-Konsole den Text des letzten Tweets zurückgeben
            System.out.println("---------- Letzter gepoteter Tweet: " +
                    timelineOperations.getUserTimeline(1).get(0).getText());
        } catch (ApiException e) {
            // Fehler ausgeben in Konsole (vereinfacht das Testen)
            System.err.println("Tweet posten fehlgeschlagen: " + e.getLocalizedMessage());

            // Fehler an aufrufende Methode zurück geben
            throw new Exception("Tweet posten fehlgeschlagen", e);
        }
    }
```

8.2.5 SendTweetDelegate überarbeiten

1. Ziel: Statt Ausgabe in Konsole nun Aufruf der Methode updateStatus des Twitter-Service
2. TwitterService mit @Autowired verdrahten
3. Execute-Methode überarbeiten:

```
/**
     * Postet einen Tweet mit dem gewünschten Text
     *
     * @param de          Objekt, welches die Verknüpfung zur Process Engine und zur aktuellen Execution enthält
     * @throws Exception
     */
    @Override
    public void execute(DelegateExecution de) throws Exception {
        // Prozessvariable tweetContent wird ausgelesen
        String tweetContent = (String) de.getVariable("tweetContent");

        // Dieser Text wird dem Twitter Service an die Methode updateStatus übergeben
        twitterService.updateStatus(tweetContent);
    }
```

8.2.6 Testen

1. Clean & Build
2. Run
3. Prozessinstanz erstellen. tweetContent muss einzigartig sein.
4. Konsole beachten und https://twitter.com/GPI45985629
5. Nochmals gleichen Tweet versuchen zu posten
6. Konsole > Camunda Cockpit

8.3 Mail versenden

8.3.1 Wie finden wir heraus, wie wir dies machen

Analog: *spring email getting started*

8.3.2 Library im Projekt verfügbar machen

Ist bereits vorhanden im pom.xml:

```
<!-- Importiert die Mail-Unterstützung für Spring Boot-Projekte. Sofern
    keine Mails gesendet werden sollen, ist diese Abhängigkeit überflüssig -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-mail</artifactId>
    </dependency>
```

8.3.3 Zugangsdaten für SMTP-Server

1. Spezielles Konto von ZHAW kann genutzt werden, aber jedes SMTP-Server-Konto würde klappen
2. Konfiguration: Gleiche Möglichkeiten wie bei Twitter, dieses Mal wählen wir Variante A (Umgebungsvariablen):
    a. In Netbeans Rechtsklick auf Projekt > Properties > Actions > neue Einträge bei Run/Debug:

```
Env.mailHost=smtps.zhaw.ch
Env.mailUser=exmailagent.gpi@zhaw.ch
Env.mailPass=bqAl1KumJLkqCHpgt6sA
Env.mailAddress=exmailagent.gpi@zhaw.ch
```

    b. In application.properties diese Variablen auslesen als auch weitere Eigenschaften für Spring Mail setzen:

```
# Mail-Konfiguration für Versand per SMTP
# SMTP-Server, Username, Password und Sender-Adresse werden aus lokalen Umgebungsvariablen ausgelesen.
# Bitte beachten, dass die Zugangsdaten damit zwar sicher nie auf Github landen, aber dass jeder, welcher Zugriff
# auf den Notebook hat, auch das Passwort auslesen kann
mail.senderaddress=${mailAddress}
spring.mail.host=${mailHost}
spring.mail.port=587
spring.mail.username=${mailUser}
spring.mail.password=${mailPass}
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

8.3.4 Service-Klasse für Kommunikation mit SMTP-Server

1. Neue Klasse *EmailService*
2. @Service-Annotation
3. JavaDoc:

```
/**
 * Service-Klasse für die Kommunikation mit einem SMTP Server
 *
 * @Service stellt sicher, dass diese Klasse beim Starten der Applikation als
 * Bean generiert wird (wie bei @Component und @Bean), aber als Service gekennzeichnet ist
 *
 * @author scep
 */
```

4. JavaMailSender verdrahten mit @Autowired (wurde automatisch beim Starten der Applikation von Spring generiert und mit den Zugangsdaten in application.properties initialisiert)
5. Sender-Adresse aus application.properties auslesen mit *@Value("${mail.senderaddress}")*
6. Mail-Senden-Methode erstellen mit folgendem Code:

```
/**
    * Sendet eine einfache Text-Mail
    *
    * @param to Empfänger
    * @param subject Betreff
    * @param body Mail-Text
    * @throws java.lang.Exception
    */
   public void sendSimpleMail(String to, String subject, String body) throws Exception {
         // Instanziert eine neue SimpleMail-Nachricht
         SimpleMailMessage simpleMailMessage = new SimpleMailMessage();

         // Legt Empfänger, Betreff und Mail-Text fest
         simpleMailMessage.setTo(to);
         simpleMailMessage.setSubject(subject);
         simpleMailMessage.setText(body);
         simpleMailMessage.setFrom(senderAddress);
         simpleMailMessage.setReplyTo(senderAddress);

         // Versucht, die Mail abzusenden
         try {
            // Mail versenden
            javaMailSender.send(simpleMailMessage);

            // In der Konsole mitteilen, dass die Mail versandt wurde für einfacheres Debugging
            System.out.println("Mail erfolgreich versandt");
         } catch (MailException me) {
            // Fehlermeldung ausgeben in Konsole
            System.err.println(me.getLocalizedMessage());

            // Fehler weitergeben an aufrufende Methode
            throw new Exception("Mail senden fehlgeschlagen", me);
         }
   }
```

8.3.5 NotifyEmployeeDelegate überarbeiten

1. Verdrahten des Mail-Services mit @Autowired
2. Statt «Mail in Konsole ausgeben» nun:

```
// Mail über Mailservice versenden
emailService.sendSimpleMail(email, "Neuigkeiten zu Ihrer Tweet-Anfrage", mailBody);
```

8.3.6 Testen

Analog 8.2.6, aber nun entweder Ablehnung, damit nicht noch gleichzeitig ein Tweet gesendet wird

9 REST-Service erstellen und konsumieren

9.1 Zielsetzung und Überblick

9.1.1 Lernziele

- Passend zur Grossklasse von SW 10 und 11 vertiefen, wie man einen einfachen REST-Service baut und konsumiert
- Damit auch das Rüstzeug haben, um selbständig die Aufgabe 9 in der Modulabschlussarbeit lösen zu können

9.1.2 Ziele und Überblick für den Twitter-Review-Prozess

1. Eine eigenständige Spring-Boot-Applikation, welche ein Benutzerverzeichnis mockt, auf welches man per REST zugreifen kann. Dieses enthält der Einfachheit halber lediglich:
    a. Eine Entität User mit den Attributen userId, userName, firstName, officialName, eMail.
    b. Ein zugehöriges Repository mit einer Methode findByUserName.
    c. Ein REST-Controller, welcher diese Methode nach aussen anbietet.
    d. Testdaten mit einem Benutzer userName=a
2. Vorname und E-Mail-Adresse sollen neu unter Angabe des userName des angemeldeten Benutzers über den REST-Service ausgelesen werden statt wie bisher gar nicht vorhanden zu sein (Vorname), respektive vom Benutzer angegeben werden zu müssen (eMail). Dies bedingt:
    a. Neuer Service Task «Benutzer-Informationen auslesen»
    b. JavaDelegate-Klasse GetUserInformationDelegate
    c. UserService zur Kommunikation mit dem REST-Service
    d. User-Resource-Klasse, um die Antwort des REST-Services deserialisieren zu können.
    e. Angabe der REST-Service-URL in application.properties.

9.2 REST-Service (User-Service) bauen

9.2.1 Neue SpringBoot-Applikation inkl. Konfiguration

Alternativ zum gezeigten Vorgehen kann auch eine Kopie der VeKa-Application als Basis genommen werden.

1. Neues Maven-Projekt «userservice»
2. Pom-File anpassen:
    a. Name & artifactId wie gehabt
    b. Parent festlegen wie gehabt (spring-boot-starter-parent 2.0.6)
    c. Dependencies festlegen:
        i. spring-boot-starter-web
        ii. spring-boot-starter-data-jpa
        iii. h2
    d. Maven-Build-Plugin wie gehabt: spring-boot-maven-plugin
3. Neue Main-Klasse UserServiceApplication wie gehabt mit @SpringBootApplication und SpringApplication.run

4. application.properties erstellen mit folgenden Angaben:
    a. spring.datasource.url=jdbc:h2:./userDb;TRACE_LEVEL_FILE=0;TRACE_LEVEL_SYSTEM_OUT=1
    b. username und password wie gehabt
    c. spring.h2.console.enabled=true und spring.h2.console.path=/console
    d. spring.jpa.hibernate.ddl-auto=update
    e. server.port = 8070
5. Clean & Build & Run & Test

9.2.2 Persistenz-Layer bauen

1. UserEntity mit name User, einer userId (Identity-generated) sowie vier normalen Attributen userName, firstName, officialName und eMail inkl. Getter und Setter.
2. Ein UserRepository mit einer findByUserName-Methodendeklaration.
3. Eine src\test\sql\initialData.sql mit folgendem Inhalt, wobei die Email durch die eigene ersetzt wird:

```
INSERT INTO USER (USER_ID, USER_NAME, FIRST_NAME, OFFICIAL_NAME, E_MAIL) VALUES (1, 'a', 'Max', 'Muster', 'max.muster@a.ch');
```

4. Clean & Build & Run & Anmelden an Konsole & SQL ausführen & Server stoppen

9.2.3 Controller bauen

Ziel: Wir wollen eine Optional<UserEntity> zurück geben (noch ohne REST) und dann in TempCommandLineRunner testen.

1. Neue UserController-Klasse mit @Component
2. Eine einzige Methode Optional<UserEntity> getUser(String userName) wie gehabt.
3. Ein TempCommandLineRunner wie gehabt, welcher die Email-Adresse zum Benutzer «a» in der Konsole ausgibt.
4. Clean & Build & Run

9.2.4 REST-Controller bauen

Ziel: Die bestehende Controller-Klasse wird so erweitert, dass statt TempCommandLineRunner innerhalb des UserService, dieser nun von aussen per REST zugreifbar wird.

1. Von @Component zu @RestController
2. Von Optional<UserEntity> zu ResponseEntity<UserEntity>
3. Der Teil aus dem TempCommandLineRunner, welcher auf Presence prüft übernehmen
4. TempCommandLineRunner löschen
5. Wenn er present ist, dann:

```
return new ResponseEntity(user.get(), HttpStatus.OK);
```

6. Ansonsten

```
return new ResponseEntity(HttpStatus.NOT_FOUND);
```

7. Nun fehlt noch das Mapping dieser Methode zu einer REST-Ressource sowie des URL-Templates zum Methodenparameter:

```
@RequestMapping(method = RequestMethod.GET, value = "/userapi/v1/users/{userName}")
@PathVariable
```

8. C'est tout. Clean & Build & Run & localhost:8070/userapi/v1/users/a und b aufrufen

9. Just for fun: Neues SoapUI Rest-Projekt über URL http://localhost:8070/userapi/v1/users/, neuer Template-Parameter für userName mit Value a und b

9.2.5 Ergebnis

Falls es nicht funktioniert: vergleiche Code mit https://github.com/zhaw-gpi/rest-service-template

9.3 REST-Service konsumieren aus Twitter-Review-Prozess

9.3.1 BPMN-Modell anpassen

1. «Mitarbeiter-Kürzel extrahieren»-Task entfernen
2. Neuer Service Task «Benutzer-Informationen auslesen» mit Delegate Expression getUserInformationAdapter

9.3.2 Neue JavaDelegate-Klasse

1. Neue JavaDelegate-Klasse GetUserInformationDelegate mit @Named und execute wie üblich
2. Antragsteller auslesen mit

```
String anfrageStellenderBenutzer = (String) execution.getVariable("anfrageStellenderBenutzer");
```

3. Nun würden wir gerne den REST-Service aufrufen mit dem userName (= anfrageStellenderBenutzer).
4. Das machen wir aber nicht hier, sondern in einer eigenen Klasse (Schichtentrennung).

9.3.3 Neue UserService-Klasse

1. Neue Klasse in package services: UserService mit @Component wie üblich
2. Sie heisst UserService, weil sie gegenüber den anderen Klassen (z.B. JavaDelegate) als Service-Klasse auftritt. Selbst nutzt sie aber auch einen UserService, aber per REST, ist also demgegenüber ein Rest-Client. Dass der Rest-Service zufälligerweise 😉 auch UserService heisst, kann der Client nicht wissen.
3. Neue Methode getUser, welche User zurück gibt oder einen Fehler wirft (später)
4. Bevor da etwas reinkommt, benötigen wir eine Spring-Komponente, welche mit REST-Services kommunizieren kann, so auch etwa mit unserem User Service. Diese Komponente heisst RestTemplate:
    a. Deklarieren eines solchen mit private final RestTemplate restTemplate;
    b. Instanzieren eines solchen beim Konstruieren der UserService-Bean, daher:

```
public UserService() {
    restTemplate = new RestTemplate();
}
```

5. Zurück zur Methode getUser. Unser Ziel ist, dass wir Vornamen und E-Mail-Adresse des Benutzers erhalten, wenn wir seinen Usernamen übergeben. Aber der REST-Service wird uns nicht dies zurückgeben, sondern ein in JSON serialisiertes UserEntity-Objekt.
6. Wir haben zwei Möglichkeiten, daraus Vornamen und E-Mail-Adresse zu extrahieren:
    a. Wir parsen die JSON-Antwort (mühsam).
    b. Wir lassen das RestTemplate die Arbeit für uns machen, indem JSON dies direkt in ein Objekt deserialisiert.

7. Bei der letzteren Variante können wir aber natürlich nicht auf die UserEntity-Klasse zugreifen (ist ja im UserService), sondern wir bauen uns selbst eine User-Klasse in einem Package resources. Diese Klasse muss nicht gleich aufgebaut sein wie UserEntity, sondern kann lediglich diejenigen Attribute enthalten, welche für uns relevant sind (firstName und eMail), der Rest aus dem JSON wird verworfen.
8. Nun können wir mit folgendem Code ein User-Objekt erhalten:

```
User user = restTemplate.getForObject(
            "http://localhost:8070/userapi/v1/users/{userName}",
            User.class,
            userName);
return user;
```

9. Wie wir wissen, kann es ja nicht bloss ein User als Antwort geben, sondern auch eine 404-Exception und in der Realität noch weitere mehr.
10. Daher den gerade erstellten Code in ein try-catch-statement. Gecatcht wird die HttpClientErrorException.
11. Wenn diese den StatusCode 404 hat (http.Status.NOT_FOUND), dann soll unsere Methode null zurückgeben, ansonsten die gecatchte Exception weiter werfen.
12. Was noch nicht gefällt: wir haben den Endpoint (http://localhost:8070/userapi/v1) direkt im Code, obwohl sich das ja ändern kann =>
    a. auslagern in application.properties in neuer Eigenschaft userservice.endpoint
    b. Zugreifen auf diese Eigenschaft über

```
@Value("${userservice.endpoint}")
  private String userServiceEndpoint;
```

    c. userServiceEndpoint nun im getForObject nutzen.

9.3.4 JavaDelegate-Klasse Teil 2

1. Nun, wo wir unseren Service haben, können wir in der execute-Methode fortfahren.
2. Zunächst Autowired für unseren Service
3. Dann die getUser-methode aufrufen und Resultat einem User-Objekt übergeben.
4. Auf null prüfen. Falls null, dann soll ein BpmnError geworfen werden, da dies eigentlich nicht passieren sollte. Könnte man nun auch wieder mit einem angehefteten Fehlereignis abfangen.
5. Ansonsten firstName und email als Prozessvariablen setzen.

9.3.5 Formulare anpassen

1. TweetAnfrageEingereicht: email nicht mehr erforderlich
2. TweetAnfragePruefen: alias nicht mehr erforderlich, dafür firstName

9.3.6 Mail-Versand anpassen

NotifyEmployeeDelegate:

1. firstName ebenfalls auslesen
2. Statt Hallo Mitarbeiter neu mit Vorname ansprechen

9.3.7 Testen

1. Clean & Build & Run (sicherstellen, dass auch der REST-Service läuft)
2. Neue Instanz erstellen und prüfen, ob alles durchläuft
3. Falls ja, super.
4. Falls nicht, zunächst Abweichungen zum Quellcode auf https://github.com/zhaw-gpi/twitter-review-prozess/releases/tag/kapitel-9 herausfinden und sonst Björn fragen