TwintPay Systems

Zahlungsabwicklung per Twint  
Projektdokumentation  
Implementierung mit der Camunda-Plattform

Inhaltsverzeichnis

[1. Vorwort 2](#_Toc202267180)

[2. Beschreibung des Auftrags 2](#_Toc202267181)

[2.1 Anforderungen 2](#_Toc202267182)

[2.2 Projektkonzept – TwintPay Systems 2](#_Toc202267183)

[3. Beschreibung des Vorgehens 2](#_Toc202267184)

[3.1 Ideenfindung 2](#_Toc202267185)

[3.2 Initiale Konzeption 2](#_Toc202267186)

[3.3 Iterative Entwicklung 2](#_Toc202267187)

[3.4 Integration, Test und Fehlerbehebung 3](#_Toc202267188)

[4. Technische Umsetzung 3](#_Toc202267189)

[4.1 Modellierung des BPMN-Prozesses 3](#_Toc202267190)

[4.2 Webformulare und Benutzerinteraktion (HTML + JS) 4](#_Toc202267191)

[4.2.1 index.html – Prozessstart 4](#_Toc202267192)

[4.2.2 form.html – Eingabe von Empfänger und Betrag 5](#_Toc202267193)

[4.2.3 confirmation.html – Rückmeldung und Abschluss 5](#_Toc202267194)

[4.3 REST API-Kommunikation 6](#_Toc202267195)

[5. Fehlerbehandlung und Logging 6](#_Toc202267196)

[6. Testing 6](#_Toc202267197)

[6.1 Testfälle 6](#_Toc202267198)

[7. Mögliche Erweiterungen 7](#_Toc202267199)

[8. Erkenntnisse 7](#_Toc202267200)

[Abbildungsverzeichnis 8](#_Toc202267201)

# 1. Vorwort

Im Rahmen des Moduls 254 bestand die Aufgabe darin, selbstständig einen Geschäftsprozess zu modellieren und zu automatisieren. Ziel war es, ein tieferes Verständnis für BPMN 2.0, Prozessausführung mit Camunda und die Integration von Benutzerinteraktionen mittels Formulare zu entwickeln.

# 2. Beschreibung des Auftrags

## 2.1 Anforderungen

* Modellierung eines Geschäftsprozesses mit Camunda Modeler (BPMN)
* Integration eines benutzerfreundlichen Eingabeformulars (HTML/JS)
* Übergabe der Benutzerdaten an die Camunda Engine
* Verknüpfung über REST API
* Nachweis einer vollständigen Prozessinstanz mit Datenübergabe

## 2.2 Projektkonzept – TwintPay Systems

Das Projekt simuliert eine einfache Zahlungsabwicklung mit Twint. Der Kunde soll den Empfänger und den Betrag angeben. Die Daten werden über ein HTML-Formular erfasst und mittels Camunda REST API an eine User Task übermittelt. Der Prozess dient als Proof-of-Concept für eine einfache Zahlungsstrecke mit manuellem Input.

# 3. Beschreibung des Vorgehens

## 3.1 Ideenfindung

Von Beginn an war klar, dass das Projekt einen möglichst realitätsnahen und zugleich technisch überschaubaren Ablauf abbilden sollte. Die Wahl fiel daher auf die Simulation einer Twint-Zahlung – ein alltäglicher Vorgang, der sich gut in Prozessnotationen übertragen lässt und ausreichend Potenzial für Erweiterungen bietet.

## 3.2 Initiale Konzeption

Der erste Prozessentwurf bestand aus einem Start-Event, gefolgt von einer User Task zur Eingabe der Zahlungsdaten. Anschliessend sollte eine Entscheidung oder Prüfung erfolgen, die den Benutzer entweder zur Bestätigung oder zu einer Fehleranzeige führt. Der Prozess endet anschliessend mit einem End-Event. Dieser einfache Ablauf bietet eine gute Grundlage für den Einsatz von Camunda Forms, REST-Variablenübergabe und strukturierte Task-Aufrufe.

## 3.3 Iterative Entwicklung

Die Umsetzung erfolgte schrittweise. Zunächst wurde das Frontend mit HTML und JavaScript entwickelt, welches die Daten vom Benutzer aufnimmt. Danach wurde die REST-Kommunikation zur Camunda-Engine realisiert, um die User Task automatisch zu beenden. Die Konfiguration der Tasks, das Mapping der Variablen sowie die Prüfung der Prozessinstanzen im Camunda Cockpit erfolgte iterativ. Probleme wurden dabei direkt im Cockpit oder über die Entwicklertools im Browser analysiert und behoben.

## 3.4 Integration, Test und Fehlerbehebung

Jede Komponente – vom Startbutton über die REST-Aufrufe bis zur finalen Rückmeldung – wurde einzeln getestet. Dabei zeigte sich, dass vor allem die Konsistenz der TaskDefinitionKeys und Variablennamen essenziell ist. Auch das Polling der nachfolgenden User Tasks (z. B. Bestätigung oder Fehlermeldung) wurde mehrfach angepasst, um eine möglichst stabile Rückmeldung an den Benutzer zu ermöglichen.

# 4. Technische Umsetzung

## 4.1 Modellierung des BPMN-Prozesses

Die Prozesslogik wurde vollständig mit dem Camunda Modeler erstellt und als ausführbarer Prozess (Executable) gespeichert. Der Ablauf beginnt mit einem Start-Event, das durch die HTML-Seite index.html initiiert wird. Danach folgt eine User Task, in der der Benutzer Empfänger und Betrag angibt. Die Angaben werden als Prozessvariablen gespeichert. Im Anschluss entscheidet der Prozess – hier in Form einer simulierten Entscheidung – ob die Zahlung erfolgreich war oder nicht. Danach führt der Prozess den Benutzer zu einer zweiten User Task, entweder zur Bestätigung oder zur Fehleranzeige. Anschliessend endet der Ablauf mit einem End-Event.

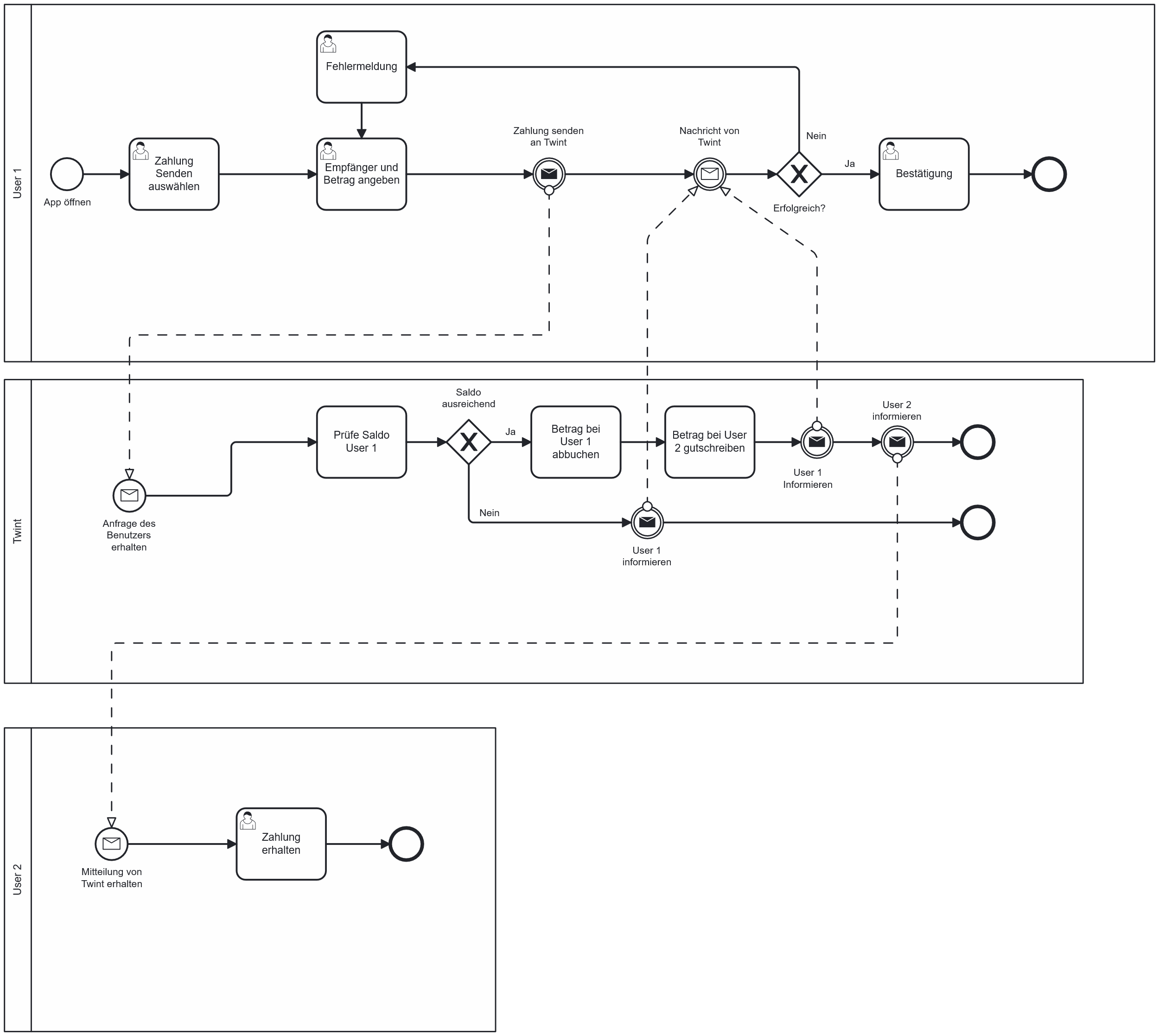


Abbildung 1 camunda file

## 4.2 Webformulare und Benutzerinteraktion (HTML + JS)

### 4.2.1 index.html – Prozessstart

Diese Seite enthält einen Button, mit dem eine neue Instanz des Zahlungsprozesses gestartet wird. Über einen REST-Aufruf an die Camunda Engine wird der Prozess ausgelöst und die erhaltene Prozessinstanz-ID (processInstanceId) an die nächste Seite (form.html) übergeben.Ein Bild, das Text, Screenshot, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 2 index.html

Die Antwort enthält processInstanceId, die an form.html übergeben wird.

### 4.2.2 form.html – Eingabe von Empfänger und Betrag

Ein Bild, das Text, Screenshot, Karte Menü enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 3 form.html

Die Seite enthält ein einfaches Formular mit zwei Feldern – eines für den Empfänger, eines für den Betrag. Nach Absenden des Formulars wird per JavaScript nach der entsprechenden User Task gesucht und diese dann mit den beiden Eingabewerten abgeschlossen. Die Werte werden dabei als Camunda-Prozessvariablen varReceive und varAmount übergeben.

### 4.2.3 confirmation.html – Rückmeldung und Abschluss



Abbildung 4 confirmation

Die gesamte Kommunikation zwischen Frontend und Prozessengine erfolgt über die REST API von Camunda. Dazu zählen u. a. der Start eines Prozesses (POST /process-definition/key/.../start), das Abrufen der offenen Tasks (GET /task?processInstanceId=...) und das Abschliessen einer User Task mit Variablenübergabe (POST /task/{id}/complete).

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 5 error

Je nach Fall erscheint ein Button zum Bestätigen oder zum Zurückkehren.

## 4.3 REST API-Kommunikation

|  |  |
| --- | --- |
| Aktion | REST-Endpunkt |
| Prozess starten | POST /engine-rest/process-definition/key/{key}/start |
| Tasks abrufen | GET /engine-rest/task?processInstanceId=...&taskDefinitionKey=... |
| Task abschliessen | POST /engine-rest/task/{taskId}/complete |

# 5. Fehlerbehandlung und Logging

Die Fehlerbehandlung erfolgt direkt im JavaScript-Code durch try...catch-Blöcke. Bei REST-Fehlern, nicht auffindbaren Tasks oder ungültigen Rückgaben wird eine passende Fehlermeldung im Browser angezeigt. Zudem wird das Polling automatisch wiederholt, wenn noch keine passende Aufgabe gefunden wurde. Dies sorgt für Robustheit auch bei asynchronen Prozessabläufen.

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 6 chatch e

Polling-Probleme werden mit Timeouts behandelt und automatisch erneut versucht.

# 6. Testing

Zur Sicherstellung der Funktionalität wurden gezielt Tests durchgeführt. Es wurden sowohl typische Abläufe („Happy Path“) als auch Fehlerfälle geprüft. Dabei wurde z. B. getestet, ob eine Weiterleitung nach Prozessstart korrekt erfolgt, ob Variablen korrekt gesetzt werden, ob Tasks abgeschlossen werden können, und ob Rückmeldungen zuverlässig erscheinen. Der gesamte Ablauf war reproduzierbar testbar.

## 6.1 Testfälle

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. | Szenario | Erwartung | Ergebnis |
| 1 | Prozessstart | Weiterleitung zu Eingabeformular | ✅ |
| 2 | Gültige Eingabe | Zahlung wird bestätigt | ✅ |
| 3 | REST-Aufruf schlägt fehl | Fehlermeldung im Browser | ✅ |
| 4 | Bestätigung wartet | Button erscheint nach einigen Sekunden | ✅ |
| 5 | Fehlerhafte Zahlung (simuliert) | Fehler-Button erscheint mit Rückleitung | ✅ |

# 7. Mögliche Erweiterungen

Für eine produktive Version oder weiterführende Entwicklung sind zahlreiche Erweiterungen denkbar. So könnten z. B. Eingaben durch zusätzliche Validierung abgesichert werden, etwa mit IBAN-Prüfung oder Telefonnummernvalidierung. Eine Integration mit der echten Twint-API würde eine realitätsnahe Zahlungssimulation ermöglichen. Auch die Speicherung der Zahlungen in einer Datenbank und die Anzeige einer persönlichen Zahlungshistorie im Frontend wären denkbar. Schliesslich könnten auch Benutzerrollen, etwa durch eine Login-Funktion, sowie automatisierte E-Mail-Benachrichtigungen eingeführt werden.

# 8. Erkenntnisse

Durch dieses Projekt konnte ich ein vollständiges Verständnis für das Zusammenspiel zwischen Benutzeroberfläche und Prozessmodellierung entwickeln. Besonders der Einsatz der Camunda REST API in Kombination mit JavaScript war lehrreich und zeigte auf, wie Geschäftsprozesse mit externen Schnittstellen verbunden werden können. Zudem wurde mir bewusst, wie entscheidend klare Namenskonventionen, saubere Prozessstrukturen und testbare Teilabschnitte für die fehlerfreie Umsetzung sind. Die Umsetzung dieses Projekts hat mir nicht nur praktische Programmiererfahrung vermittelt, sondern auch methodisch-konzeptionelle Kompetenzen im Umgang mit Geschäftsprozessen geschult. Die klare Trennung zwischen Logik und Benutzeroberfläche war entscheidend für Übersichtlichkeit und Erweiterbarkeit.

# Abbildungsverzeichnis

[Abbildung 1 camunda file 3](#_Toc202267156)

[Abbildung 2 index.html 4](#_Toc202267157)

[Abbildung 3 form.html 5](#_Toc202267158)

[Abbildung 4 confirmation 5](#_Toc202267159)

[Abbildung 5 error 5](#_Toc202267160)

[Abbildung 6 chatch e 6](#_Toc202267161)