Frontend-Integration mit JavaScript-Events und Self-Contained Systems

Im Zuge der Weiterentwicklung unserer Omnichannel-Plattform und der Migration des Thalia Webshop-Monolithen sind bereits zahlreiche [Self-Contained Systems](#) und Microservices entstanden. Immer wieder stellt sich dabei die Frage nach der „richtigen“ Integrationsform zwischen einzelnen Systemen. Die Lösungsmöglichkeiten sind vielfältig: HTTP/REST, Messaging und Datenreplikation im Backend oder HTML-Transklusion und JavaScript im Frontend. Eine allgemeingültige Lösung gibt es hier nicht. Je nach Anforderungen und Anwendungskontext sind unterschiedliche Ansätze möglich und sinnvoll. Nach dem Self-Contained Systems-Architekturstil wird eine Integration auf Frontend-Ebene bevorzugt, um die Kopplung zwischen den Systemen möglichst lose zu halten. Aus diesem Grund setzen wir bei Thalia bei der Integration einiger unserer Systeme auf die Frontend-Integration.

In diesem Beitrag geht es um eine spezielle Form der Frontend-Integration unter Verwendung von Client-Side-Includes und JavaScript-Events, die wir bei Thalia umgesetzt haben. Anhand eines fiktiven Beispiels soll die Umsetzung konkret demonstriert werden. Das lauffähige [Beispiel](#) findet sich auf Github.

Fallbeispiel und Problemstellung

Als fachliches Beispiel soll ein fiktiver Kaufprozess in einem Online-Shop dienen. Der Kaufprozess wird vereinfacht realisiert und besteht lediglich aus der Auswahl einer Zahlungsart und dem Kaufabschluss. Die beiden Prozesse „Zahlungsartenauswahl“ und „Kauf“ werden von zwei verschiedenen Self-Contained Systems getrennt voneinander realisiert: dem Zahlungsartenservice und Checkoutservice. Beide Systeme sind Webanwendungen auf Basis von Spring-boot mit einer eigenen Datenhaltung, Businesslogik und Präsentationslogik inklusive eigenem Frontend.

Es besteht die Anforderung, dass der gesamte Kaufprozess (Auswahl der Zahlungsart und Kaufabschluss) auf einer einzigen Seite durchlaufen werden

kann, ohne diese zu verlassen. Außerdem soll die Auswahl der Zahlungsart dem „Kauf“-Prozess bekannt gemacht werden, da diese für die anschließende Auftragsverarbeitung benötigt wird. Es ist also eine Integration der Systeme notwendig, die in diesem Beispiel ausschließlich über das Frontend erfolgen soll.

HTML-Transklusion

Damit die Auswahl der Zahlungsart und der Kaufabschluss auf einer Seite möglich ist, ist es notwendig die entsprechenden HTML-Fragmente aus den beiden Systemen zusammenzuführen. In der Literatur wird dieser Ansatz auch als [HTML-Transklusion](#) bezeichnet. HTML-Transklusion beschreibt den Vorgang HTML-Fragmente zur Laufzeit nachzuladen und in einer bestehenden HTML-Seite einzubetten. Die Fragmente können als Ressourcen von unterschiedlichen Quellen bereitgestellt werden. Eingebettet bilden die einzelnen Fragmente zusammen mit der einbettenden HTML-Seite eine neue Seite. Die Granularität der Fragmente kann dabei variieren.

Die Transklusion lässt sich auf zwei verschiedene Arten realisieren: serverseitig oder clientseitig. Bei der serverseitigen Transklusion werden die referenzierten HTML-Fragmente von einem Proxy-Server vor der Auslieferung der HTML-Seite an den Client nachgeladen und eingebettet. Im Gegensatz dazu werden die Fragmente bei der clientseitigen Variante erst im Browser des Clients geladen und verarbeitet.

In dem Fallbeispiel wird das HTML-Snippet zur Auswahl der Zahlungsart in die HTML-Seite des Kaufprozesses clientseitig nachgeladen und eingebettet. Folgender Quellcodeausschnitt zeigt dafür die Implementierung der einbettenden Seite, in diesem Fall dem Checkoutservice:

```
function includeZahlungsartenauswahl(){
    fetch('http://localhost:8081/zahlungsarten')
        .then(function(response) {
            return response.text();
        })
        .then(function(html){
            $("#zahlungsarten-checkout-content" ).html(html);
        });
}
```

Mithilfe der [fetch-API](#) wird die Ressource „zahlungsarten“ aus dem Zahlungsservice (hier erreichbar unter localhost:8081) geladen. Der Body aus der Antwort wird mithilfe von jquery in das HTML im Checkoutservice in das Element mit der Id „zahlungsarten-checkout-content“ eingebettet. In der Antwort enthalten ist sowohl HTML als auch JavaScript. Entscheidend ist hierbei, dass beim Einbetten der Antwort auch das mitgelieferte JavaScript als solches interpretiert und ausgeführt wird. Dies wird durch die [html](#)-Funktion von jquery automatisch gewährleistet.

Durch die HTML-Transklusion befinden sich beide HTML-Teile auf einer Seite und das aus dem Zahlungsservice nachgeladene JavaScript hat Zugriff auf das gleiche Document Object Model wie das JavaScript aus dem Checkoutservice. Die Voraussetzungen für den Datenaustausch über JavaScript-Events sind somit geschaffen. Die folgende Abbildung zeigt das Frontend des Checkouts nach erfolgreicher HTML-Transklusion:

Checkout

Bücher-Abo

Preis: 9.99 €

Zahlungsart

Bitte wählen Sie ihre Zahlungsart aus

Vorkasse

[Auswählen](#)

Rechnung

[Auswählen](#)

Bankeinzug

[Auswählen](#)

Kaufen

Ergebnis der HTML-Transklusion im Frontend

Datenaustausch über Custom-JavaScript-Events

In unserem Fallbeispiel besteht die Anforderung, dass die Auswahl der Zahlungsart an den Kaufprozess übermittelt werden sollen. Um diese Information von einem Self-Contained System zum Anderen zu überreichen werden sogenannte [CustomEvents](#) verwendet. Mit diesem speziellen Event-Typ ist es möglich, benutzerdefinierte Daten zum Objekt hinzuzufügen und zusammen mit dem Event zu transportieren. Die CustomEvent-API definiert dazu einen Konstruktor, der einen Event-Namen und einen Hash mit dem Key „detail“ als Parameter erwartet. Der folgende Quellcode zeigt die Implementierung eines CustomEvents:

```
let event = new CustomEvent('zahlungsart-ausgewaehlt',
    {detail: zahlungsartEventData.toJSON()});
// event auf dem Window-Objekt veröffentlichen
window.dispatchEvent(event);
```

Zunächst erfolgt die Instanziierung des CustomEvents mit dem namen „zahlungsart-ausgewaehlt“ und den Zahlungsinformationen im JSON-Format. Anschließend wird das Event über das Window-Objekt veröffentlicht. Ausgelöst wird die Event-Verarbeitung in unserem Beispiel durch die Selektion einer Zahlungsart.

Damit das Event konsumiert werden kann ist es lediglich notwendig einen EventListener für das definierte Event an dem Window-Objekt zu registrieren. Hierbei ist es wichtig, dass der EventListener vor der Veröffentlichung von Events registriert wird. Die Implementierung im Checkoutservice-Frontend sieht dazu folgendermaßen aus:

```
window.addEventListener('zahlungsart-ausgewaehlt', function
(event) {
    //Event verarbeiten
    checkoutData.setZahlungsart(event.detail.zahlungsart);
}, false);
```

Über das Event-Objekt und den Key „detail“ hat die Funktion aus dem Checkoutservice-JavaScript Zugriff auf die transportierten Informationen. Diese Daten können nun von dem Checkoutservice für die weitere Auftragsverarbeitung genutzt werden.

Fazit und Ausblick

Durch den Einsatz von HTML-Transklusion und Custom-JavaScript-Events ist es möglich Daten zwischen mehreren Self-Contained Systems über das Frontend auszutauschen. Die Integrationsform ähnelt dem aus Messaging-Systemen bekannten Publish-Subscribe-Pattern. Das „zahlungsort-ausgewählt“-Event wird nach dem „fire-and-forget“-Prinzip veröffentlicht, sodass der Sender den Empfänger nicht kennen muss. Die Kopplung zwischen den Self-Contained Systems reduziert sich daher auf den Event-Namen und den auszutauschenden Daten. Beides muss beiden Systemen bekannt sein und wie eine API behandelt werden.

Beim Austausch der Daten über JavaScript-Events im Frontend ist zu beachten, dass diese Daten dem Browser im Klartext zur Verfügung stehen. Es gilt daher genau zu prüfen, ob die Integrität und Geheimhaltung der ausgetauschten Daten notwendig ist. In unserem Beispiel bestimmt der Zahlungsortenservice welche Zahlungsorten der Kunde zur Auswahl erhält. Diese Auswahl könnte beim Transport zum Checkoutservice im Frontend manipuliert werden, sodass eine zusätzliche Validierung im Backend notwendig wäre. Dies wird deutlicher, wenn die Anforderung besteht mittels Lastschriftverfahren zahlen zu können. Die an den Checkoutservice übertragenen Bankverbindungsdaten müssten ebenfalls erneut validiert werden.

Um doppelte Validierungslogik zu vermeiden besteht die Möglichkeit, entsprechende Daten zu signieren und als Hash zu übertragen. Der Checkoutservice könnte somit feststellen, ob die transportierten Daten manipuliert wurden. Dadurch wäre die Integrität der transportierten Daten sichergestellt. Dieser Ansatz fand bei der Realisierung des [Thalia-Club-Anmeldeprozesses](#) Anwendung, damit die für den Kaufabschluss benötigten Daten zwischen zwei Self-Contained Systems über das Frontend ausgetauscht werden konnten.