JET-Frontends beherrschbar machen –

mit Micro Frontends

Hamed Roknizadeh
IT-P Information Technology-Partner GmbH
Hannover

Schlüsselworte

Oracle JET, Micro-Frontend, UI

Einleitung

Bei Web-Anwendungen wird das Frontend immer umfangreicher. Häufig wird auch ein großer Teil der Anwendungslogik im Frontend implementiert. Im Laufe der Zeit wächst die oft von einem separaten Team entwickelte Frontend-Schicht und wird schwieriger zu warten.

So passiert es in vielen Teams, dass neben den Backend-Diensten Frontend-Monolithen erstellt werden. Und ein monolithischer Ansatz für eine große Frontend-Anwendung wird unhandlich.

Es muss eine Möglichkeit geben, das Frontend in kleinere Module aufzuteilen, die unabhängig voneinander agieren können. Anhand von einem Praxisbeispiel zeige ich euch eine Methode, wie der browserbasierte Code einer Oracle JET-Anwendung in Micro Frontends aufgeteilt werden kann.

Bei diesem Ansatz wird die JET Anwendung in ihre Funktionen unterteilt. Jedes Feature kann von einem anderen Team entwickelt werden. Dies stellt sicher, dass jedes Feature unabhängig von anderen Features entwickelt, getestet und bereitgestellt wird.

Dabei gibt es verschiedene Techniken, um am Ende eine aufeinander abgestimmte Web-Anwendung zu erhalten.

Micro Frontend

Micro Frontend erweitert die Idee der Microservices auf den Bereich der Frontend-Entwicklung. Der Begriff Micro Frontend wurde erstmals Ende 2016 im ThoughtWorks Technologie-Radar vorgestellt. Während der großen Projekte wächst die Frontendschicht – welche oft von einem separaten Team entwickelt wird – und wird schwieriger zu warten.

Die Idee des Micro Frontend ist es, eine Website oder Web App als eine Komposition von Features zu erstellen, die unabhängigen Teams gehören. Jedes Team ist für eine einzelne Geschäftsdomäne verantwortlich. Ein Team ist cross functional und entwickelt seine Funktionen Ende-zu-Ende, von der Datenbank bis zur Benutzeroberfläche.

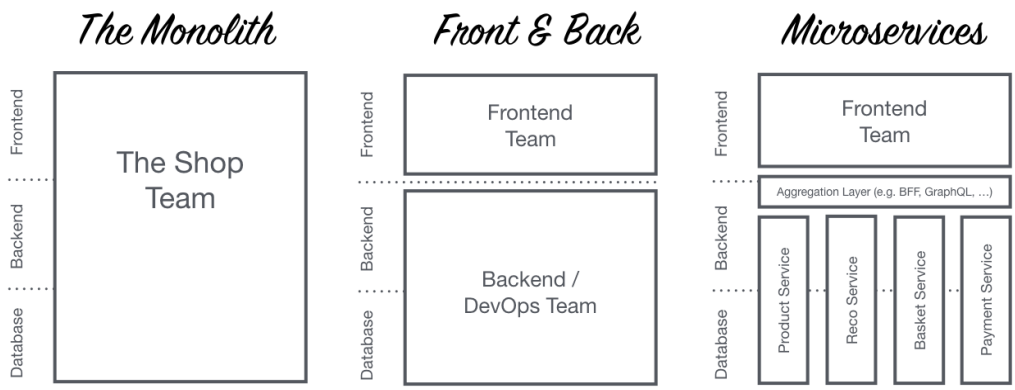

Abb. 1: Monolithic Frontends (Quelle: https://micro-frontends.org)

Was ist ein Monolith?

Eine monolithische Anwendung wird als eine einzige, einheitliche Einheit aufgebaut. Häufig besteht ein Monolith aus drei Teilen: einer Datenbank, einer clientseitigen Benutzeroberfläche (bestehend aus HTML-Seiten und / oder JavaScript, die in einem Browser ausgeführt wird) und einer serverseitigen Anwendung. Die serverseitige Anwendung verarbeitet HTTP-Anforderungen, führt domänenspezifische Logik aus, ruft Daten aus der Datenbank ab, aktualisiert diese und füllt die HTML-Ansichten auf, die an den Browser gesendet werden.

Ein weiteres Merkmal eines Monolithen ist, dass es oft eine massive Code-Basis ist. Serverseitige Anwendungslogik, Client-seitige Front-End-Logik, Hintergrundjobs usw. sind alle in derselben Codebasis definiert. Das bedeutet, wenn Entwickler Änderungen oder Aktualisierungen vornehmen möchten, müssen sie den gesamten Stack auf einmal bauen und deployen.

Monolith Vorteil:

Weniger Querschnittsthemen: Ein großer Vorteil der monolithischen Architektur besteht darin, dass sich nur um Querschnittsthemen wie Protokollierung oder Zwischenspeicherung für eine Anwendung gekümmert werden muss.

Geringerer operativer Overhead: Bei einem Monolithen muss nur eine Anwendung erstellt werden, für die Protokollierung, Überwachung und Tests eingerichtet werden. Ein Monolith ist im Allgemeinen auch weniger komplex bereitzustellen, da nicht mehrere Deployments organisiert werden müssen.

Einfachere Tests: Mit einem Monolith lassen sich automatisierte Tests einfacher einrichten und ausführen, da alles unter einem Dach ist. Bei Microservices müssen Tests für verschiedene Anwendungen in verschiedenen Laufzeitumgebungen berücksichtigt werden, was sehr komplex sein kann.

Performance: Ein Monolith ist gegenüber Microservices häufig performanter. Das liegt meist daran, dass dieser lokale Anrufe anstelle eines API-Anrufs über ein Netzwerk verwendet.

Monolith Nachteil:

Übermäßige Kopplung: Mit einem Monolith können Verstrickungen einerseits vermieden werden, andererseits wird er jedoch anfälliger dafür, je größer er wächst. Da alles so eng gekoppelt ist, wird die Isolierung von Diensten innerhalb des Monolithen schwierig, was die Entwicklung erschwert, wenn es um unabhängige Skalierung oder Codepflege geht.

Schwieriger zu verstehen: Meist ist es so, dass Monolithen im Vergleich zu Microservices schwieriger zu verstehen sind. Dies ist z.B. ein Problem bei der Einarbeitung neuer Team-Mitglieder. Dies ist häufig eine direkte Folge der engen Kopplung sowie der Tatsache, dass es Abhängigkeiten und Nebeneffekte geben kann, die nicht offensichtlich sind, wenn ein bestimmter Dienst oder Controller betrachtet wird.

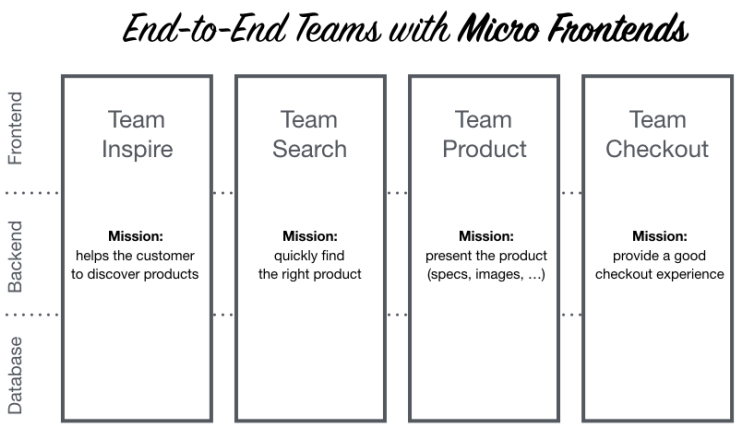

Abb. 2: Organisation in Verticals (Quelle: https://micro-frontends.org)

Micro Frontend Idee

Sei Technologie-Agnostiker

Jedes Team sollte seinen Stack auswählen und aufbauen können, ohne sich mit anderen Teams zu koordinieren.

Team-Code isolieren

Es sollten keine Laufzeitobjekte geteilt werden, auch wenn alle Teams das gleiche Framework verwenden. Es sollten unabhängige Apps erstellt werden, die eigenständig sind. Somit sollten keine gemeinsamen Status oder globale Variablen verwendet werden.

Robuste Web-Anwendung (unabhängig im Fehlerfall)

Funktion sollte verwendbar sein, auch wenn z.B. JavaScript fehlgeschlagen ist oder noch nicht ausgeführt wurde.

Oracle JET mit Micro-Frontend

In dem heutigen Beispiel bestehen kleine eigenständige Anwendungen mit Micro-Frontend, welche von den Hauptanwendungen verwendet werden. Die Micro-Frontend wurden als JET Composite Components implementiert. Da die Hauptanwendungen die Micro-Applications per URL aufrufen, können sie auf anderen Servern bereitgestellt werden.

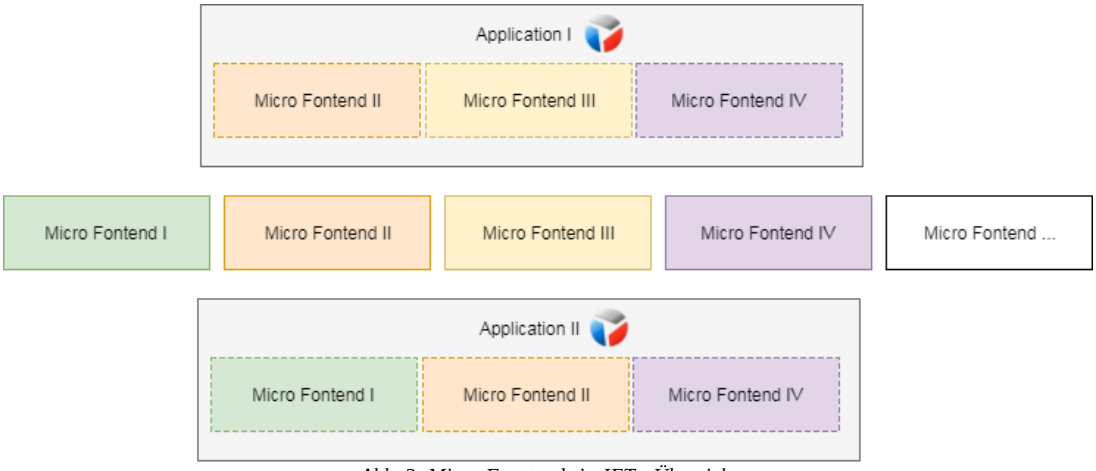

Abb. 3: Micro Frontends in JET - Übersicht

Beim Ausführen der JET-Anwendung lädt der Browser viele HTML-, JS- und CSS-Dateien vom Server herunter. Bei einer Gesamtanwendung werden besagte Dateien von einem einzigen Host zur Verfügung gestellt. Bei der Verwendung von Micro-Frontends aber laden Sie diese Ressourcen von mehreren Servern parallel.

Oracle JET Composite-Komponenten sind als eigenständige Module verpackt, die Ihre Anwendung mithilfe von RequireJS laden können. Das Framework stellt APIs zur Verfügung, welche es Ihnen ermöglicht die Composite-Komponenten zu registrieren. Knockout bietet derzeit ein ein- und zweiseitiges Data-Binding, ein Template-Markup und die Composite-Component-Activation[1].

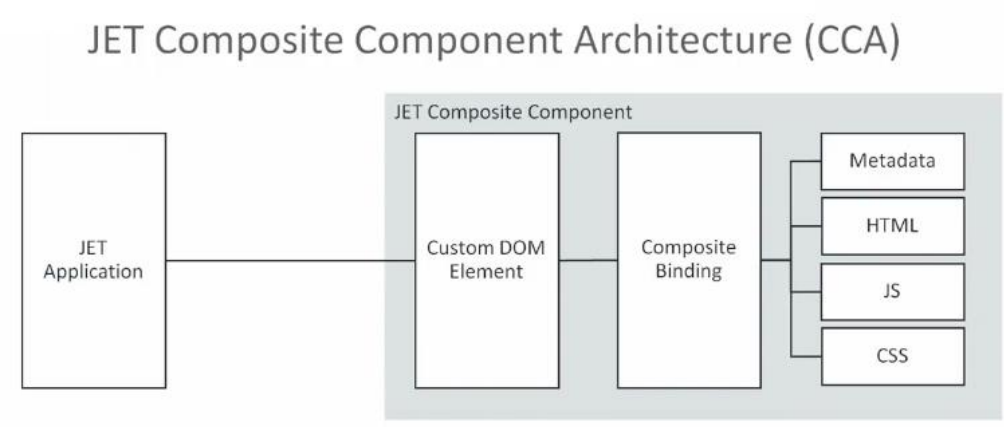

Abb. 4: Quelle: https://docs.oracle.com

[1] https://docs.oracle.com/middleware/jet320/jet/developer/, Aufruf 27.09.2018

Zum Laden der CCA muss das requireJS-Text-Plugin verwendet werden. Standardmäßig verhindern die Sicherheitsrichtlinien des Browsers das Abrufen von Ressourcen durch dieses Plugin (z.B. Ansicht und Metadaten). Nur der Abruf von .js-Dateien ist von einem anderen Origin möglich. Um dies zu beheben, ist eine einfache Änderung in der requireJS-Konfiguration der verwendeten Anwendung nötig. Die Konfiguration für das Text-Plugin sollte eine UseXhr-Funktion definieren, die dem Quellserver für die Komponente den Wert true zurückgibt[2].

Im Folgenden finden Sie ein allgemeines Beispiel für die Implementierung einer requireJS-Konfiguration:

```
requirejs.config(
  {
    baseUrl: 'js',
    // Path mappings for the logical module names
    paths:
    {
      'knockout': 'libs/knockout/knockout-3.4.0.debug',
      'jquery': 'libs/jquery/jquery-3.1.0',
      'jqueryui-amd': 'libs/jquery/jqueryui-amd-1.12.0',
      'promise': 'libs/es6-promise/es6-promise',
      'hammerjs': 'libs/hammer/hammer-2.0.8',
      'ojdnd': 'libs/dnd-polyfill/dnd-polyfill-1.0.0',
      'ojs': 'libs/oj/v2.2.0/debug',
      'ojL10n': 'libs/oj/v2.2.0/ojL10n',
      'ojtranslations': 'libs/oj/v2.2.0/resources',
      'text': 'libs/require/text',
      'css' : 'libs/require-css/css',
      'signals': 'libs/js-signals/signals'
    },
    // Shim configurations for modules that do not expose AMD
    shim:
    {
      'jquery':
      {
        exports: ['jQuery', '$']
      }
    },
    ,
    //Additional Configuration
    config:
    {
      text:
        {
          useXhr: function (url, protocol, hostname, port) {
            // Allow cross-domain requests to get Text resources
            // Remote server must set Access-Control-Allow-Origin header
            return true;
          }
        }
    }
  }
);
```

Abb. 5: requireJS-Konfiguration

[2] vgl. https://blogs.oracle.com/groundside/jet-composite-components-xii-revisiting-the-loader-script, Aufruf 27.09.2018

Technische Realisierung (Micro-Frontends)

Zuerst muss eine CCA erstellt werden. Diese erzeugt die in */js/jet-composites* benötigten Kernartefakte.

Kernkomponentenartefakte:

- *loader.js*: Registrierung der Komponenten (*metadata*, *viewModel*, *view*, *css)*
- *component.json*: Definiert die API der Komponente (Properties, Methods, Events, Slots)
- *view.html*: Die Benutzeroberfläche für die Komponente
- *viewModel*: Das Modell, das die UI unterstützt und für die Implementierung von Methoden etc. verantwortlich ist
- *README.md*: Dokumentation (Markdownformat)

ojet create component micro-frontend-i

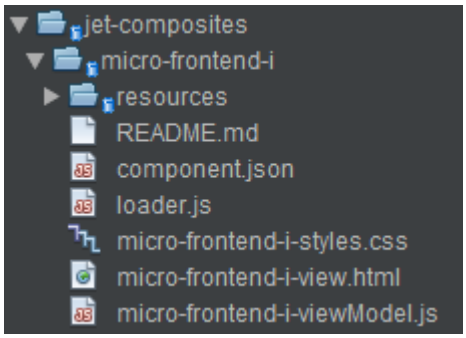

Abb. 6: CCA-File-Structure

Technische Realisierung (Hauptanwendung)

Jede JET-Anwendung besteht aus zwei Dateien: Eine HTML- und eine Javascript-Datei. In der Datei *application i.html* wird *micro-frontend-i*, *micro-frontend-ii* und *micro-frontend-iv* aufgerufen. Um die aktuelle Auswahl innerhalb des *micro-frontend-i* verwenden zu können, besitzt *micro-frontend-i* einen *selectionListener*.

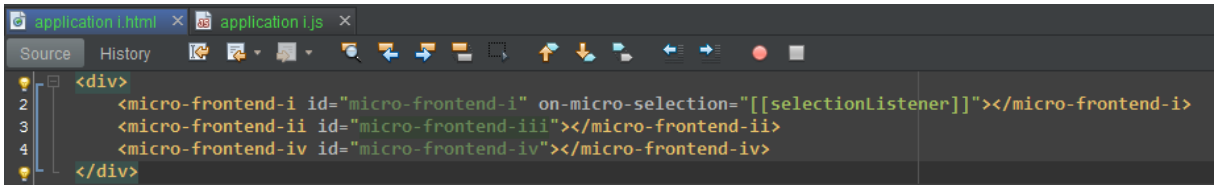

Abb. 7: application i.html

In der *application i.js* wird *loader.js* von *micro-frontend-i*, *micro-frontend-iii* und *micro-frontend-iv* aufgerufen. Hier können die Micro-Frontend-Komponenten von verschiedenen Hosts geladen werden. Die *selectionListener*-Funktion erhält die Auswahl von *micro-frontend-i* und aktualisiert den entsprechenden Parameter von *micro-frontend-iii*.

```
application i.html | application i.js
Source | History
1  define(['ojs/ojcore'
2      , 'knockout'
3      , 'jquery'
4      , '//host/micro-frontend-i/js/jet-composites/micro-frontend-i/loader.js'
5      , '//host/micro-frontend-iii/js/jet-composites/micro-frontend-iii/loader.js'
6      , '//host/micro-frontend-iv/js/jet-composites/micro-frontend-iv/loader.js'],
7        function (oj, ko, $) {
8
9            function application_i() {
10               var self = this;
11
12               self.selectionListener = function (event) {
13                   document.getElementById('micro-frontend-iii').synchMicro(event.detail.value);
14               };
```

Abb. 8: application i.html

Wie auch in *application i.html* werden in der *application ii.html* drei Micro-Frontends aufgerufen. In diesem Fall jedoch *micro-frontend-i*, *micro-frontend-ii* und *micro-frontend-iv*.

```
application ii.html | application ii.js
Source | History
<div>
    <micro-frontend-i id="micro-frontend-i" on-micro-selection="[[selectionListener]]"></micro-frontend-i>
    <micro-frontend-ii id="micro-frontend-ii"></micro-frontend-ii>
    <micro-frontend-iv id="micro-frontend-iv"></micro-frontend-iv>
</div>
```

Abb. 9: application ii.html

Wie zuvor bei *application i.js* werden die *loader.js*-Dateien von *micro-frontend-i*, *micro-frontend-ii*, *micro-frontend-iv* aufgerufen. Nun jedoch werden die Informationen des *selectionListener*s mit micro-frontend-ii synchronisiert.

```
application ii.html | application ii.js
Source | History
1  define(['ojs/ojcore'
2      , 'knockout'
3      , 'jquery'
4      , '//host/micro-frontend-i/js/jet-composites/micro-frontend-i/loader.js'
5      , '//host/micro-frontend-ii/js/jet-composites/micro-frontend-ii/loader.js'
6      , '//host/micro-frontend-iv/js/jet-composites/micro-frontend-iv/loader.js'],
7        function (oj, ko, $) {
8
9            function application_ii() {
10               var self = this;
11
12               self.selectionListener = function (event) {
13                   document.getElementById('micro-frontend-ii').synchMicro(event.detail.value);
14               };
15           }
16       });
```

Abb. 10: application ii.js

Innerhalb der Präsentation wird der komplette Aufbau der Anwendung, die Micro-Frontends und dessen Kommunikation untereinander näher erläutert.

Kontaktadresse:

Hamed Roknizadeh

IT-P Information Technology-Partner GmbH

Seligmannallee 4-6

30173 Hannover

Telefon:	+49 (511) 61 68 04 - 390
Fax:	+49 (511) 61 68 04 - 17
E-Mail	h.roknizadeh@it-p.de
Internet:	www.it-p.de