Technology
Arts Sciences
TH Köln

Erstellung einer isolierten Vue-Komponenten Bibliothek im Zusammenspiel mit dem Utility-Class-Framework Tailwindcss

Entwicklung und Dokumentation anpassbarer Vue-Komponenten für kontinuierlich wachsende Webplattformen am Beispiel einer instanzierbaren Plattformlösung der AVACO GmbH.

Exposé zum Praxisprojekt

ausgearbeitet von

Marvin Christian Klimm

vorgelegt an der

Technischen Hochschule Köln
Campus Gummersbach
Fakultät für Informatik und
Ingenieurwissenschaften

im Studiengang
Medieninformatik

Hochschul-Betreuer:	Prof. Christian Noss Technische Hochschule Köln
Unternehmensansprechpartner:	Thomas Brambring AVACO GmbH

Gummersbach, im August 2020

Adressen: Marvin Christian Klimm
Am Hepel 61
51643 Gummersbach
studies@marvinklimm.de

Prof. Christian Noss
Technische Hochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
christian.noss@th-koeln.de

Thomas Brambring
AVACO GmbH
Löhestraße 18
53773 Hennef (Sieg)
t.brambring@avaco.io

Inhaltsverzeichnis

1 Kontext und Problemfeld

Derzeit sind Singlepage-Application-Frameworks wie Vue.js und React.js laut Github mit die meist verfolgten Open Source Projekten weltweit. Sie finden ihren Einsatz in zahlreichen Webapplikationen und Websites jeglicher Größe und Art. Seit 2017 wird das Frontend des Hauptproduktes My.PORTAL der AVACO GmbH vollständig auf Vue.js Basis entwickelt.

Laut eigenen Angaben des Unternehmens waren die Entwicklungsfreundlichkeit, die solide Community, die Performance, die vielen nützlichen Bibliotheken, die leichte Erlernung sowie die Unternehmensunabhängigkeit des Frameworks für die Einführung des Frameworks entscheidend gewesen.

1.1 Über das Produkt My.PORTAL

Mit My.PORTAL bietet die AVACO GmbH ein anpassbares Portal, das als Handels-, Kommunikations- und Kollaborationslösung in unterschiedlichen Nutzungskontexten eingesetzt werden kann. Besonders hervorgehobene Einsatzzwecke sind die Nutzung als Stadtportal, Unternehmensportal und Bildungsportal, in denen die Plattformbetreiber eine eigene für sie angepasste DSGVO-konforme Portallösung erhalten.

1.2 Kommunikation zwischen Backend und Frontend

Das in .NET implementierte Backend basiert auf einer Microservices-Architektur in welcher jeder Microservice gesondert voneinander agiert. Das Frontend kann über unterschiedliche Endpoints auf die einzelnen APIs der Services zugreifen. Zu Beginn des Projektes wurden RESTful APIs erstellt, nach c.a. 18 Monaten Entwicklung wurde begonnen erste GraphQl APIs zu implementieren. Daher kommunizieren heute ältere Services über REST-APIs und neuere über GraphQl APIs.

1.3 Frontend-Architektur

Das Frontend wird in einer modularisierten Weise entwickelt und als Single Page Application bereitgestellt. Derzeit besteht das Frontend aus den folgenden Modulen:

- portal-frontend: Haupt-Applikation, die außerdem auch alle Instanzkonfigurationen beinhaltet.
- portal-global-components: Bibliothek von wiederverwendbaren Vue-Komponenten
- portal-tracker: Framework um Aktivitäten dem Backend zu übermitteln

- portal-statistics: Weitere Vue-Applikation um Portal Administratoren Nutzungsstatistiken anzuzeigen

1.4 Projektrelevante Herausforderungen im Frontend

Über den Entwicklungszeitraum des Projektes haben sich einige Probleme in der Frontend Entwicklung herausgestellt. Diese werden im Folgenden erläutert.

1.4.1 Redundante Style Definition

Das Portal Styling wurde von Anfang an in SASS entwickelt. Jede Komponente hat in der Regel ihre komplett eigenen „Scoped" Style Definitionen innerhalb einer .vue Datei erhalten. Konstante Attributwerte wie beispielsweise Abstände, Gridgrößen und Farben wurden per SASS-Mixins oder SASS-Variablen eingefügt was zu redundanten Style Definitionen im transpilierten CSS Code geführt hat.

1.4.2 Property Hell

Zu Beginn des Projektes wurden viele Komponenten spezifisch für den ersten Einsatzkontext entwickelt. Für ein konsistentes UI-Design war es immer wieder nötig bestehende Komponenten in einer leicht-abgewandelten Form im Portal wiederzuverwenden. Dazu wurden regelmäßig neue Properties zu Komponenten hinzugefügt, um sie an neue Kontexte anzupassen. Diese Properties steuern oft nur einen einzigen Use-Case und erschweren es den Entwicklern die Dokumentation sowie den Code einer Komponente überblicken zu können.

1.4.3 Komponentenmanagement

Bisher wurden Komponenten gemäß dem Atomic Design Prinzip von Brad Frost entwickelt und geordnet. Dabei werden Komponenten in die Kategorien „Atom", „Molekül", „Organismus", „Template" und „Page" kategorisiert. Dabei haben sich für dieses Projekt zwei Schwächen herausgestellt.

1. Der Unterschied zwischen Molekülen und Organismen ist oft für die Komponenten nicht strikt definierbar.

2. Es kann passieren, dass größere Komponenten, wie beispielsweise Organismen Moleküle benötigen, die sonst keine andere Komponente benötigt. Diese Moleküle werden jedoch in dem Ordner Moleküle abgespeichert, was zu einer Unübersichtlichkeit der Repositories geführt hat.

1.4.4 Nachträgliches Komponenten Splitting

Es kommt vor, dass Abschnitte einer Vue-Komponente zur Wiederverwendung eines Markup Abschnitts in eine neue Komponente ausgelagert werden sollen. Durch die Scoped-SASS Style Definitionen ist es mühsam und fehleranfällig die richtigen Definitionen zu identifizieren und zu verschieben.

1.4.5 Ungenutzte Stylings

Über die Entwicklungszeit hinweg hat sich gezeigt, dass es Komponenten gibt, die gegenüber ihrer ersten Ausführung geändert wurden. Dabei konnte es passieren, dass ungenutzte Style Definitionen im Code verblieben sind.

1.4.6 Umständliche Dokumentation, der globalen Komponenten

Eine Dokumentation von Komponenten mit Code Beispielen und Property Auflistungen muss manuell vom Entwickler verantwortet werden. Dies führt dazu, dass Dokumentationen durch menschliche Fehler unvollständig sind. Zudem ist die manuelle Dokumentation aufwendig.

1.4.7 Portal-Config Abhängigkeit für Global Components

Jede Portal Instanz wird im portal-frontend-Repository konfiguriert und diese Konfiguration gespeichert. Das Repository portal-global-components ist zur Verwendung auf eine gültige Portal-Konfiguration aus portal-frontend angewiesen. Dadurch sind die Komponenten in Zukunft nicht in anderen Projekten ohne gültige Portal-Konfiguration nutzbar.

2 Schnittstelle zur Medieninformatik

Die folgenden Disziplinen aus dem Studiengang der Medieninformatik können sich positiv auf den Projekterfolg auswirken.

- Webbasierte Anwendungen 1: Das Projekt wird in modernen Frontend-Technologien umgesetzt
- Grundlagen der visuellen Kommunikation: Die Verwendung von Tailwindcss kann sich auf den Designprozess auswirken.
- Mensch-Computer-Interaktion: Die neuen Komponenten sollen positiv in das Nutzererlebnis einwirken. Außerdem soll das Entwicklungs-Erlebnis der Frontend-Entwickler verbessert werden.
- Webbasierte Anwendungen 2: Das Endprodukt in welchem die neuen Komponenten genutzt werden kommuniziert über APIs mit dem Backend.

3 Arbeitsergebnis

Für einen Abschluss des Projektes soll das Framework Tailwindcss in allen Fontend-Repositories nutzbar sein und ein neues im Production Stage nutzbares Vue-Komponenten-Repository angefertigt werden. Dieses Repository soll eine Verbesserung gegenüber des bereits existierenden *portal-global-components* - Repository darstellen um dieses langfristig vollständig ablösen zu können. Der Verbesserungsgrad soll anhand der in Abschnitt 1.4 genannten Herausforderungen bewertet werden. Außerdem sollen Chancen und Risiken bzgl. der Nutzung von Tailwindcss in einem Vue (v.2) Projekt dargelegt werden und ein Ausblick auf die Nutzung in Bezug auf die neue Vue-Version 3 gegeben werden.

Literaturverzeichnis

[Github 2020] Github, Suche: *GitHub Star Filtering Search @ONLINE*. `https://github.com/search?q=stars%3A%3E100&s=stars&type=Repositories`. Version: März 2020. – Last accessed 4 Mai 2020