Ruhr-Universität Bochum

Bachelorarbeit

# Entwicklung eines erweiterbaren Kommandozeileninterfaces zur Generierung von Web-Applikationen

Schriftliche Prüfungsarbeit
für die Bachelor-Prüfung des Studiengangs Angewandte Informatik an der Ruhr-Universität Bochum

vorgelegt von
Michael David Kuckuk

am
Lehrstuhl für Informatik im Bauwesen
Prof. Dr.-Ing. Markus König

| | |
|---|---|
| Abgabedatum: | 24. August 2021 |
| Matrikelnummer: | 108 017 207 503 |
| 1. Prüfer: | Prof. Dr.-Ing. Markus König |
| 2. Prüfer: | Stephan Embers, M. Sc. |

## Abstract

The ABSTRACT is to be a fully-justified text following the title page. The text will be formatted in 12 pt, single-spaced Computer Modern. The title is "Abstract", set in 12 pt Computer Modern Sans Serif, centered, boldface type, and initially capitalized. Writing the abstract in English is mandatory even if the thesis itself is written in German. The length of the abstract should be roughly about 200 words but must not exceed 250 words.

As usual, the abstract should clearly summarize the aim, the background and the results of your thesis so that an interested reading can decide if he or she wants to further read your (interesting) written report. Also, the abstract should be kept simple. That is, do not include tables, figures or cross references in the abstract. You will have plenty of time in your thesis to explain things more clearly.

## Erklärung

Ich erkläre, dass das Thema dieser Arbeit nicht identisch ist mit dem Thema einer von mir bereits für eine andere Prüfung eingereichten Arbeit.

Ich erkläre weiterhin, dass ich die Arbeit nicht bereits an einer anderen Hochschule zur Erlangung eines akademischen Grades eingereicht habe.

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen der Entlehnung kenntlich gemacht. Dies gilt sinngemäß auch für gelieferte Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen.

## Statement

I hereby declare that except where the specific reference is make to the work of others, the contents of this thesis are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university.

This thesis is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text.

Datum Unterschrift

# Inhaltsverzeichnis

# 1 Motivation

Ziel dieser Arbeit ist es, ein Command Line Interface (CLI) zu entwickeln, was Entwickelnden die initiale Erzeugung des Projektes für eine JavaScript-besierte Webapplikation abnimmt. Im folgenden Kapitel wird zunächst erklärt, warum diese Aufgabe überhaupt automatisiert werden soll. Daraufhin werden dann Anforderungen er

# 2 Stand der Technik

Um dieses Konfigurationsproblem zu automatisieren oder zumindest zu erleichtern, gibt es bereits verschiedene Lösungen. Momentan sind drei Frontend-Frameworks besonders beliebt (React, Angular und Vue) [1] und für jedes dieser Frameworks existiert jeweils ein Programm, was (unter anderem) zur initialen Erstellung von Projekten mit dem jeweiligen Framework empfohlen wird.

## 2.1 React

In der Dokumentation für React wird empfohlen, neue Projekte über das CLI create-react-app (CRA) zu erstellen. Dieses kann per Node Package Manager (NPM) installiert werden und ist dann in der Lage, den Inhalt eines angegebenen Templates (oder des Standardtemplates) in ein angegebenes Verzeichnis zu kopieren. Daraufhin werden die benötigten Abhängigkeiten per NPM oder mittels eines anderen installierten Paketmanagers (wie z.B. Yarn) installiert.

Dritten ist es möglich, eigene Templates für CRA zu erzeugen, die dann wie Erstanbietertemplates zur Erzeugung des neuen Projekts genutzt werden können. Wenn man also mittels CRA ein Projekt erzeugen möchte, das bereits mit gewissen Bibliotheken oder Werkzeugen ausgestattet ist, muss dafür dafür ein Template erstellt worden sein. Die Wahrscheinlichkeit, dass das jedoch passiert ist, sinkt mit der Spezifität der Wünsche.

## 2.2 Angular

Das Angular-Team stellt das sog. Angular-CLI zur Verfügung, das Entwickelnden beim Programmieren verschiedene wiederkehrende und repetitive Aufgaben abnehmen soll (wie z.B. die Erstellung und Einbindung neuer Komponenten). Neben diesen Aufgaben wird das Tool auch zur Erstellung neuer Angular-Projekte genutzt.

Der Befehl ng new <project-name> führt Nutzende in einen Dialog, bei dem einige Fragen zur gewünschten Konfiguration des Projektes gestellt werden. Es können vier Features konfiguriert werden und daraufhin wird das Projekt eingerichtet.

Im Gegensatz zu CRA sind hier die Möglichkeiten, die Nutzenden geboten werden, sehr eingeschränkt, da es nicht wie bei den Templates für Dritte die Möglichkeit gibt, neue

Libraries mitsamt entsprechender Konfiguration in den Erstellungsprozess einzubinden. Dafür können sich Nutzende hier (im Rahmen der beschränkten angebotenen Optionen) eine beliebige Konfiguration aussuchen und sind nicht darauf angewiesen, dass jemand vor ihnen schon denselben Wunsch hatte. Außerdem wird so auch Anfänger:innen die Entdeckung und der Einstieg in neue Libraries erleichtert.

## 2.3 Vue

Von den drei Frameworks bietet Vue in Bezug auf die Projekterstellung das umfangreichste CLI. Hier trifft man zunächst eine Vorauswahl von zehn Features, die man haben oder nicht haben möchte. Daraufhin werden zu den ausgewählten Features detailliertere Fragen gestellt. Insgesamt stehen einem durch dieses Tool über 20 verschiedene Libraries ohne weiteren Konfigurationsaufwand zur Verfügung.

## 2.4 Vergleich der Möglichkeiten zwischen Frameworks

Alle drei Tools bieten Entwickelnden die Möglichkeit, eigene Erweiterungen zu erarbeiten und zu veröffentlichen. Im Falle von React muss dies in Form eines Templates geschehen. Da nur ein Template zur Erstellung eines Projektes genutzt werden kann, sind derartige Erweiterungen hier also exklusiv.

Bei Angular und React ist es jedoch möglich, auch Erweiterungen zu entwickeln, die zusätzlich zu anderen Optionen und Erweiterungen nutzbar sind. Für das Angular CLI kann man sogenannte Schematics entwickeln, die das Hinzufügen und Einbinden einer Bibliothek vollautomatisch übernehmen. Diese Schematics müssen aber leider nach der Installation ausgeführt werden und müssen insbesondere von Nutzenden entdeckt werden. Hierfür gibt es keine eigene Plattform o.ä. und die Auswahl der Schematics, die schon bei der Projekterstellung auswählbar sind, beschränkt sich auf Angular-interne Features (z.B. die Einfügung eines Routers). Selbst Libraries, die auch vom Angular-Team betreut werden (z.B. Angular Material) müssen später per Schematic nachgerüstet werden.

Im Rahmen des Vue CLI's ist es immerhin möglich, auch Libraries von Dritten direkt bei der Projekterstellung einzubinden. Allerdings ist auch hier die anfängliche Auswahl nicht erweiterbar. Dafür können, wie auch schon bei Angular, anschließend automatisch über Drittanbieterplugins neue Libraries heruntergeladen, importiert und demonstriert werden.

Tabelle

Außerdem bieten alle drei der genannten Tools keine Unterstützung für andere Frameworks. Daher ist beispielsweise der große Umfang der Features des Vue CLI's leider nicht in einem Angular-Projekt nutzbar.

Tabelle 1: Automatische und initiale Installierbarkeit verschiedener Features in Angular und Vue Projekten

| | Angular | | Vue | |
|---|---|---|---|---|
| Feature | Automatisch installierbar | Ititial auswählbar | Automatisch installierbar | Initial auswählbar |
| TypeScript | wird erzwungen | | ✓ | ✓ |
| Router | ✓ | ✓ | ✓ | ✓ |
| PWA-Support | ✓ | × | ✓ | ✓ |
| Linter | ✓ | × | ✓ | ✓ |
| Formatierer | ✓ | × | ✓ | ✓ |
| CSS Reset | × | × | × | × |
| CSS Präprozessor | ✓ | ✓ | ✓ | ✓ |
| Design Framework | ✓ | × | ✓ | × |
| State Management | wird erzwungen | | ✓ | ✓ |
| Unit Testing Framework | ✓ | × | ✓ | ✓ |
| E2E Testing Framework | ✓ | × | ✓ | ✓ |

# 3 Implementierung

## 3.1 Abhängigkeiten und Exklusitiväten

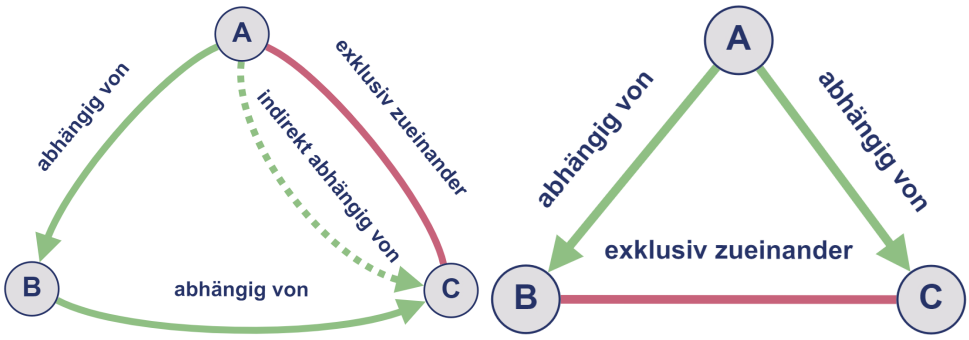
Im Rahmen der Abhängigkeiten und Exklusivitäten von Erweiterungen zueinander gibt es mehrere Probleme, die gelöst werden müssen. Zum einen kann es schnell passieren, eine unmögliche Kombination von Bedingungen zu stellen. Außerdem muss nach der Auswahl einer Kombination von Erweiterungen geprüft werden, ob diese Kombination zulässig ist.

### 3.1.1 Überprüfung der Umsetzbarkeit

Ein triviales Beispiel für unmögliche Kombinationen von Bedingungen wäre ein Paar von Abhängigkeiten $A$ und $B$, wobei $A$ und $B$ exklusiv zueinander sind und gleichzeitig $A$ von $B$ abhängt. Es können in diesem Beispiel niemals beide Anforderungen gleichzeitig erfüllt werden.

Natürlich lassen sich aber auch kompliziertere Beispiele erzeugen. In Abbildung 1 werden zwei solche Konstellation dargestellt. Bei der einen hängt $A$ von $B$ und $B$ von $C$ ab, aber $A$ ist zu $C$ exklusiv. Diese Kombination von Anforderungen lässt sich ebensowenig wie das erste Beispiel gleichzeitig erfüllen, aber das Problem ist hier nicht sofort offensichtlich. Man muss hierfür erkennen, dass Abhängigkeiten transitiv sind. Ist also die Erweiterung $A$ von $B$ und $B$ von $C$ abhängig, so ist auch indirekt $A$ von $C$ abhängig.

Der zweite in Abbildung 1 dargestellte Konflikt beruht im Gegensatz zu den bisherigen Beispielen nicht darauf, dass eine Erweiterung zugleich (indirekt) abhängig und exklusiv




(a) Eine Möglichkeit, wie ein Widerspruch zwischen indirekter Abhängigkeit und Exklusivität entstehen kann.

(b) Hier entsteht ein Widerspruch dadurch, dass eine Erweiterung zwei Abhängigkeiten hat, die zueinander exklusiv sind.

Abbildung 1: Zwei Graphen zur Veranschaulichung von Konfliktmöglichkeiten zwischen Abhängigkeiten und Exklusivitäten.

zu einer anderen Erweiterung ist. Hier wird eine weitere Kategorie von Problemen dargestellt: sind zwei (indirekte) Abhängigkeiten einer Erweiterung zueinander exklusiv, so können nicht beide dieser Abhängigkeiten erfüllt werden. Auch solche Konfigurationen sind also unzulässig.

Aus diesen fachlichen Überlegungen heraus ergibt sich eine erste Lösung. Für jede Erweiterung $E$ sind zunächst alle direkten sowie indirekten Abhängigkeiten zu bestimmen. Für alle diese Abhängigkeiten ist dann zu prüfen, dass zum einen die Abhängigkeit $A$ nicht zu $E$ exklusiv ist, aber auch dass $A$ zu keiner weiteren (indirekten) Abhängigkeit $A'$ von $E$ exklusiv ist.

Bei näherer Betrachtung dieser Lösung lässt sich jedoch einiges an Ineffizienz feststellen. Zum einen werden für jede Erweiterung erneut die transitiven Abhängigkeiten berechnet. Aufgrund eben dieser Transitivität sind jedoch die Berechnungen für alle Erweitungen, die Abhängigkeit einer anderen Erweiterung sind, redundant. Im schlimmsten Fall (nämlich, wenn eine Erweiterung von insgesamt $n$ Erweiterungen von allen anderen Erweiterungen (indirekt) abhängt) ist also genau eine der durchgeführten Berechnungen wirklich notwendig und $n - 1$ Berechnungen werden unnötigerweise durchgeführt. Aus ähnlichem Grund ist auch die Überprüfung der Exklusitivät zweier paarweise unabhängiger (indirekter) Abhängigkeiten häufig überflüssig.

Die Lösung dieser Probleme ergibt sich aus einer theoretischeren Betrachtung des Problems. Wie bereits aus der Verwendung des Begriffs der Transitivität hervorgeht, lassen sich Abhängigkeit und Exklusitivät als mathematische Relationen über der Menge aller Erweiterungen auffassen. Hierbei ist besonders hervorzuheben, dass die Exklusivität symmetrisch ist (ist $A$ zu $B$ exklusiv, so ist auch $B$ zu $A$ exklusiv) während Abhängigkeit nicht symmetrisch ist (im Gegenteil: bei der anfänglichen Analyse von möglichen Bibliotheken, Frameworks etc. ergab sich, dass Abhängigkeit nie symmetrisch zu sein

scheint). Allerdings ist Exklusivität a priori nicht transitiv (auch, wenn $A$ zu $B$ und $B$ zu $C$ exklusiv ist, können $A$ und $C$ zusammen verwendet werden), während Abhängigkeit sehr wohl transitiv ist (wie bereits erläutert).

Vor diesem Hintergrund lässt sich erkennen, dass die Bestimmung der (indirekten) Abhängigkeiten der Bestimmung der Transitiven Hülle gleichkommt. Diese kann mittels des Floyd-Warshall-Algorithmus (in der Warshall-Variante) berechnet werden [2]. Hierfür muss zunächst ein gerichteter Graph erzeugt werden, in den alle deklarierten Abhängigkeiten als Kante eingefügt werden. Von diesem Graphen wird dann die transitive Hülle bestimmt, in der zwei Knoten $A$ und $B$ genau dann durch eine (von $A$ nach $B$ gerichtete) Kante verbunden sind, wenn die Erweiterung $A$ von $B$ abhängt.

Auch die Relation der Exklusivität lässt sich in einen Graphen überführen. In diesem Graphen gibt es ebenfalls pro Erweiterung einen Knoten und jede Exklusivität wird als Kante dargestellt. Aufgrund der Symmetrie der Exklusivität kann dieser Graph aber ungerichtet sein.

Das Problem der Überprüfung der Restriktionen reduziert sich nun darauf, sicherzustellen, dass es zwischen zwei Knoten (also zwischen zwei Erweiterungen) in maximal einem der beiden Graphen eine Kante gibt, wobei die Richtung keine Rolle spielt (denn wenn die Knoten exklusiv zueinander sind, darf es zwischen beiden keine Abhängigkeit geben – egal, in welche Richtung).

Anders formuliert, darf es im Graphen der Exklusivitäten keine Kante $\{A, B\}$ geben, für die in der transitiven Hülle der Abhängigkeiten die Kante $(A, B)$ oder die Kante $(B, A)$ existiert. Aufgrund dessen, dass die transitive Hülle gerichtet ist, gibt es darin doppelt so viele Kanten wie in dem Graphen der Exklusivitäten. Außerdem liegt sie als Adjazenzmatrix vor, während die Exklusivitäten als Kantenliste vorliegen. Somit kann bei $m$ Exklusivitäten in $\mathcal{O}(m)$ über die Exklusivitäten iterieren und jeweils in $\mathcal{O}(1)$ die Existenz einer (transitiven) Abhängigkeit prüfen. Die Umkehrung, also die Iteration über Abhängigkeiten und dann die Prüfung der Existenz einer Exklusivität, würde bei $n$ Erweiterungen einen Aufwand von $\mathcal{O}(n^2)$ verursachen. Wenn man vorher die Kantenliste der Exklusivitäten in $\mathcal{O}(n^2)$ in eine Adjazenzmatrix überführt, ist die anschließende Existenzprüfung auch wieder in $\mathcal{O}(1)$ möglich, aber insgesamt

Der Algorithmus von Floyd-Warshall sorgt dafür, dass diese Überprüfung eine asymptotische Laufzeit von $\mathcal{O}(n^3)$ hat. Da die oben beschriebenen Probleme in beliebiger Tiefe von Abhängigkeiten auftreten können, ist jedoch die Bestimmung der transitiven Hülle nicht vermeidbar. Allerdings ist damit zu rechnen, dass die Anzahl aller Erweiterungen stets kleiner als 100 sein wird (andernfalls würde die Benutzbarkeit des Programms möglicherweise stark eingeschränkt). Daher ist diese Laufzeit in diesem Fall als unbedenklich einzustufen.

# Literatur

[1] Sacha Greif und Raphaël Benitte. Front-end Frameworks — State of JS 2020, 2020. `https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/#front_end_frameworks_experience_ranking` [Zugriff am 26. Juli 2021].

[2] S. Warshall. A theorem on boolean matrices. *Journal of the ACM (JACM)*, 9(1): 11–12, 1962. ISSN 0004-5411. doi: 10.1145/321105.321107.