

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: 31. 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 12pt 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 made 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	1
2	Grundlagen	1
2.1	Einführung in Frontend-Frameworks für Webapplikationen	1
2.2	Ähnliche Projektgeneratoren	3
2.3	Funktionale Programmierung	5
2.4	Reaktive Programmierung mit Redux	5
3	Konzeptionierung	5
3.1	Auswahl der zu vergleichenden Installationsprozesse	6
3.2	Vergleich der Installationsprozesse	10
3.3	Wahl des generellen Vorgehens	14
3.4	Planung der Erweiterungen	14
3.5	Abwägung über Sonderstellung für TypeScript und Frameworks	14
4	Implementierung	14
4.1	Abhängigkeiten und Exklusivitäten	14

1 Motivation

Entwickelnde haben zu Beginn eines neuen Projektes immer erst die Aufgabe, sich eine geeignete Architektur für das Projekt zu überlegen, die sie dann möglichst von Anfang an verfolgen können. In den meisten Sprachen, insbesondere in der Webentwicklung ist ein Teil dieser Aufgabe die Auswahl von Bibliotheken und Werkzeugen, die bei der Programmierung von Hilfe sein können.

Unter diesen Bibliotheken und Werkzeugen gibt es viele, die besonders häufig verwendet werden oder in bestimmten Kombinationen empfohlen werden. Beispielsweise verwenden 60,4% der von SState of JS 2020¹ Befragten die Library Axios¹ (ein HTTP-Client, die für Browser und Node.js dasselbe Interface bietet)[7]. 89,6% der Befragten nutzen das Tool Node Package Manager (NPM) (wobei dies als offizieller Node.js-Paketmanager eine Sonderstellung genießt), 82,3% nutzen ESLint² (ein Werkzeug zur statischen Codeanalyse) und 70,9% nutzen Prettier³ (ein Codeformatierer).

Geläufige Kombinationen von Werkzeugen oder Bibliotheken ergeben sich oft aus der Installationsanleitung bzw. aus der offiziellen Dokumentation. So

2 Grundlagen

Um die Implementierung der Arbeit vollständig nachvollziehen zu können, ist es wichtig, einige Konzepte verstanden zu haben und zu wissen, worauf die Entwicklung aufbauen wird. Diese Grundlagen werden im Folgenden eingeführt werden.

2.1 Einführung in Frontend-Frameworks für Webapplikationen

Um interaktive Webapplikationen zu entwickeln, müssen diese mit HTML, CSS und JavaScript programmiert werden. Browser stellen den mit HTML definierten Inhalt dar, manipulieren das Aussehen der Seite anhand der durch CSS definierten Regeln und verfügen zudem über einen JavaScript (JS) Interpreter, der den übermittelten JS Code ausführen kann und darüber eine Manipulation der Seiteninhalte erlauben.

Durch diese Möglichkeit ergibt sich das Problem, dass die Daten, die das Programm berechnet (oder auf andere Weise erlangt) Usern korrekt dargestellt werden und umgekehrt Eingaben von Usern sich in den Daten widerspiegeln sollen.

Beim Entwickeln solcher Funktionalität wiederholen sich häufig bestimmte Probleme wie die Synchronisierung von Daten mit der Anzeige oder die Wiederverwendung von Teilen der Nutzeroberfläche. Diese Probleme können mit Hilfe von Frontend-Frameworks gelöst werden.

¹<https://www.npmjs.com/package/axios>

²<https://www.npmjs.com/package/eslint>

³<https://www.npmjs.com/package/prettier>

Zur Zeit gibt es vor allem drei Frameworks, die häufig verwendet werden: React, Angular und Vue [7]. Angular baut dabei auf dem klassischen Model View Controller (MVC)-Pattern auf. In einer Variation von HTML, in der bestimmte TypeScript-Ausdrücke interpretiert werden können, kann die View aufgebaut werden. Die Funktionalität des Controllers wird in einer TypeScript-Klasse implementiert und das Model kann über Services dargestellt werden, die dann automatisch instanziiert und in die Controller injected werden. **ZITAT FEHLT**

Vue und React verfolgen nicht so strikt das MVC-Pattern. Vue verfügt (ähnlich wie Angular) über eine Variation von HTML, in der auch bestimmte Ausdrücke evaluiert werden können, sodass so die View definiert werden kann. Allerdings ist es bei Vue üblich, dass sich noch in derselben Datei wie die View (und das CSS) auch der Controller befindet. Vue gibt im Gegensatz zu Angular keine klare Vorgabe, wie das Model umzusetzen ist.

React hingegen verfügt über einen anderen Ansatz. Hier wird es ermöglicht, eine Variante von HTML-Ausdrücken in JavaScript bzw. TypeScript einzufügen, die dann wie andere Werte auch in Variablen gespeichert werden können. Auf diesem Weg wird parallel zum eigentlichen Document Object Model (DOM) ein sogenanntes Shadow DOM generiert, der dann in das eigentliche DOM überführt wird. Somit ist es in React möglich, View und Controller komplett miteinander zu verbinden. Zudem verfügt auch React nicht über einen klaren Ansatz um das Model umzusetzen.

Alle drei dieser Frameworks ermöglichen eine komponentenbasierte Entwicklung. Das bedeutet, dass mit sehr geringem Aufwand die View in Komponenten unterteilt werden kann, die dann wiederverwendet werden können.

Aus (unter anderen) diesen Gründen werden aktuelle Webapplikationen in der Regel auf Grundlage eines Frontend-Frameworks aufgebaut **ZITAT FEHLT**. Da jedes Frontend aber mehr oder weniger klare Strukturen, Datenflüsse, Syntaxen und vieles mehr vorgibt, ist die Entscheidung eines Frameworks in Bezug auf die Suche weiterer Abhängigkeiten eine sehr folgenschwere Entscheidung.

Beispielsweise ist es nicht möglich, Angular ohne TypeScript oder React ohne ESLint zu verwenden. Sowohl für React als auch für Angular gibt es Komponentenbibliotheken⁴, die Google's Material Design System folgen; allerdings sind diese Bibliotheken sehr verschieden, haben unterschiedliche Features und gehen nicht aus demselben Code hervor.

Die Entscheidung eines Frameworks ist daher eine der ersten Entscheidungen, die Entwickelnde beim Einrichten ihrer Projekte treffen müssen. Sie schränkt die weitere Auswahl an Werkzeugen und Bibliotheken ein. Außerdem können, bis auf in besonderen Kontexten wie Micro-Frontends oder WebComponents **ZITAT FEHLT**, nur mit hohem Aufwand oder gar nicht zwei Frameworks zusammen verwendet werden.

⁴Material-UI für React: <https://material-ui.com/>; Angular Material: <https://material.angular.io/>

2.2 Ähnliche Projektgeneratoren

Um das Problem der initialen Projektkonfiguration zu automatisieren oder zumindest zu erleichtern, bieten die drei oben erwähnten Frameworks jeweils ein Programm an, was (unter anderem) zur initialen Erstellung von Projekten mit dem jeweiligen Framework empfohlen wird.

2.2.1 React

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

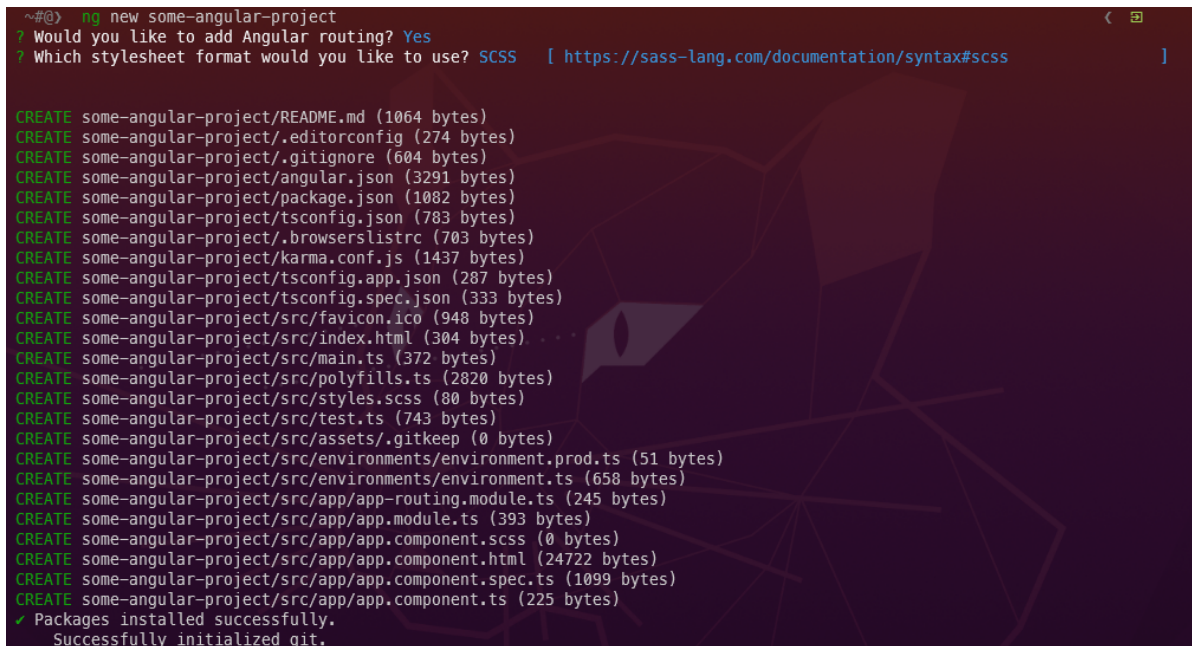
Dritten ist es möglich, eigene Templates für CRA zu erzeugen, die dann wie Erstanbietertemplates zur Erzeugung des neuen Projekts genutzt werden können. Wenn also mittels CRA ein Projekt erzeugt werden soll, das bereits mit gewissen Bibliotheken oder Werkzeugen ausgestattet ist, muss dafür zuvor ein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

Die Erstanbietertemplates geben bereits die empfohlene Ordner- und Dateistruktur für React-Projekte vor; durch Drittanbietertemplates kann hiervon aber abgewichen werden. Außerdem installieren die Erstanbietertemplates bereits einige weitere Werkzeuge wie ESLint, Jest (ein Werkzeug zum Schreiben und Ausführen von automatisierten Tests, in der Regel Unit Tests) und die sogenannten react-scripts. Diese sind eine Sammlung von kleinen Skripten und Abhängigkeiten, die Prozesse automatisieren, die für die Entwicklung notwendig sind. Beispielsweise wird so ein Skript zur Verfügung gestellt, welches einen lokalen Server startet, der den aktuellen Stand der Webapplikation hostet und bei jeder Codeänderung automatisch aktualisiert. Dieser Server gibt zudem auch die Warnungen aus, die ESLint erzeugt.

Außerdem ist es bei der Verwendung dieser Templates mit geringem Aufwand möglich, bestimmte empfohlene Bibliotheken und Werkzeuge nachzurüsten. So gibt es zum Beispiel Anleitungen für die Installation von Prettier oder von CSS Präprozessoren, die aus jeweils einem in der Kommandozeile auszuführenden Befehl bestehen. Für Prettier muss zudem noch eine bestimmte, bereits existierende Datei um einige Zeilen ergänzt werden.

2.2.2 Angular

Das Angular-Team stellt das sog. Angular-CLI zur Verfügung, das Entwicklenden beim Programmieren verschiedene wiederkehrende und repetitive Aufgaben abnehmen soll



```

~#@> ng new some-angular-project
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? SCSS [ https://sass-lang.com/documentation/syntax#scss ]

CREATE some-angular-project/README.md (1064 bytes)
CREATE some-angular-project/.editorconfig (274 bytes)
CREATE some-angular-project/.gitignore (604 bytes)
CREATE some-angular-project/angular.json (3291 bytes)
CREATE some-angular-project/package.json (1082 bytes)
CREATE some-angular-project/tsconfig.json (783 bytes)
CREATE some-angular-project/.browserslistrc (703 bytes)
CREATE some-angular-project/karma.conf.js (1437 bytes)
CREATE some-angular-project/tsconfig.app.json (287 bytes)
CREATE some-angular-project/tsconfig.spec.json (333 bytes)
CREATE some-angular-project/src/favicon.ico (948 bytes)
CREATE some-angular-project/src/index.html (304 bytes)
CREATE some-angular-project/src/main.ts (372 bytes)
CREATE some-angular-project/src/polyfills.ts (2820 bytes)
CREATE some-angular-project/src/styles.scss (80 bytes)
CREATE some-angular-project/src/test.ts (743 bytes)
CREATE some-angular-project/src/assets/.gitkeep (0 bytes)
CREATE some-angular-project/src/environments/environment.prod.ts (51 bytes)
CREATE some-angular-project/src/environments/environment.ts (658 bytes)
CREATE some-angular-project/src/app/app-routing.module.ts (245 bytes)
CREATE some-angular-project/src/app/app.module.ts (393 bytes)
CREATE some-angular-project/src/app/app.component.scss (0 bytes)
CREATE some-angular-project/src/app/app.component.html (24722 bytes)
CREATE some-angular-project/src/app/app.component.spec.ts (1099 bytes)
CREATE some-angular-project/src/app/app.component.ts (225 bytes)
✓ Packages installed successfully.
  Successfully initialized git.

```

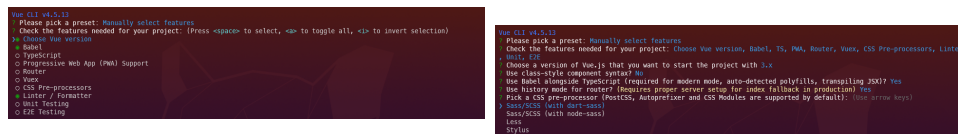
Abbildung 1: Die Ausgabe des Angular CLI's bei der Erzeugung eines neuen Projektes.

(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 zwei Fragen zur gewünschten Konfiguration des Projektes gestellt werden. Über explizit gesetzte Kommandozeilenoptionen können dann insgesamt Features konfiguriert werden, mit denen das Projekt eingerichtet wird. Diese Features sind jedoch allesamt Angular-intern, d.h. es gibt keine Möglichkeit weitere Libraries oder ähnliches installieren zu lassen, die nicht direkt vom Angular Team kommen. Selbst einige Libraries (z.B. Angular Material), die vom Angular-Team direkt entwickelt werden, sind nicht in diese initiale Projektgenerierung eingebunden.

Im Gegensatz zu CRA sind hier die Möglichkeiten, die Nutzenden bei der initialen Projekterstellung geboten werden, sehr eingeschränkt. Es gibt die Möglichkeit, bei der Einrichtung eine sogenannte "Collection" anzugeben, die dann die Erstellung des eigentlichen Projektes übernimmt. Es könnte also eine Collection entwickelt werden, die weitere Libraries je nach Eingabe einbindet. Jedoch scheint die Umsetzung dieses Features abzunehmen: von den überprüften Collections für ESLint, Prettier und Apollo-Angular hat nur eine dieses Feature (also die Installation bei der Initialisierung statt hinterher) jemals unterstützt, hat es aber mittlerweile wieder eingestellt [4] [6] [8].

Im Gegensatz zu der React-Lösung 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 durch die gestellten auch Anfänger:innen die Entdeckung und der Einstieg



- (a) Das Vue CLI fragt explizit nach zehn verschiedenen Features. Davon kann eine beliebige Teilmenge ausgewählt werden.
- (b) Für jede der Auserwählten Erweiterungen werden Detailliertere Fragen gestellt.

Abbildung 2: Zwei Bilder von dem Prozess der Erstellung einer neuen Vue-Applikation.

in Angular erleichtert, da gewisse Features automatisch an sie herangetragen werden.

2.2.3 Vue

Von den drei Frameworks bietet Vue in Bezug auf die Projekterstellung das umfangreichste CLI. Zu aller erst besteht die Möglichkeit, ein Preset auszuwählen. Dies kann eines von zwei Standardpresets sein oder eines, was zuvor auf dem lokalen Computer erstellt und gespeichert wurde. Es ist auch möglich, ein bereits existierendes Preset oder ein komplett neues Preset als Kommandozeilenparameter anzugeben.

Falls man kein Preset auswählt, trifft man nun 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. Die soeben erstellte Konfiguration kann anschließend als neues Preset gespeichert werden.

2.3 Funktionale Programmierung

2.4 Reaktive Programmierung mit Redux

3 Konzeptionierung

Angeichts dessen, dass das geplante CLI verschiedenste Bibliotheken und Werkzeuge unterstützen können soll und dabei auch zukunftssicher bleiben soll, ist es unabdingbar, die Entwicklung im Vorhinein zu planen. Hierfür muss zunächst untersucht werden, welche Gemeinsamkeiten und welche Unterschiede bei verschiedenen Installationsprozessen vorliegen. Daraufhin kann dann ein System erarbeitet werden, was die Gemeinsamkeiten fördert und gleichzeitig Raum für die bekannten, aber auch möglichst für erwartbare unbekannte Unterschiede lässt.

3.1 Auswahl der zu vergleichenden Installationsprozesse

Um zu bestimmen, welche Schritte bei der Einrichtung von neuen Projekten durchführbar sein müssen, wurden zunächst verschiedenste Projekte erstellt, zu denen dann möglichst verschiedene Abhängigkeiten und Werkzeuge hinzugefügt wurden.

Aufgrund der bereits erläuterten Beliebtheit von Frontend-Frameworks wurde für jedes der drei bekanntesten Framework ein Projekt erstellt, in dem dann die entsprechenden Installationen durchgeführt wurden. Für diese initiale Erstellung wurde jeweils das bereits vorgestellte CLI verwendet.

Bei der Auswahl dessen, was zu diesen Projekten hinzugefügt wurde, lag der Fokus zum einen auf Bekanntheit und Verbreitung und zum anderen auf Vielseitigkeit. Da sich ähnliche Tools oft auch in ihrem Installationsprozess ähneln (beispielsweise waren die Schritte zur Installation von ESLint fast identisch zu den Schritten zur Installation von TSLint⁵), ist es für diese Vorbereitung also nicht nötig, eine Abhängigkeit hinzuzufügen, wenn bereits eine andere ausprobiert wurde, die einen Ähnlichen Funktionsumfang hat.

Aus diesen Gründen wurden folgende Werkzeuge installiert:

- **ESLint** ist ein Werkzeug zur statischen Codeanalyse für JavaScript. Es gibt Warnungen aus, wenn bestimmte Ausdrücke verwendet wurden, die häufig zu Bugs führen (z.B. die Verwendung von einem doppelten Gleichheitszeichen anstelle eines dreifachen Gleichheitszeichens, d.h. die typunsichere Gleichheitsüberprüfung anstelle Gleichheitsprüfung mit Prüfung auf Typgleichheit).

ESLint verfügt auch über die Möglichkeit, Formatierungspräferenzen anzugeben. Diese werden in die Analyse mit einbezogen und darüber können beispielsweise Warnungen ausgegeben werden, wenn Zeilen falsch eingerückt sind. Einige Warnungen, darunter die meisten Warnungen zur Formatierung, können von ESLint auf Wunsch automatisch behoben werden. Standardmäßig ist ESLint jedoch nur passiv, also idempotent.

ESLint wird stellvertretend für alle Tools zur statischen Codeanalyse überprüft. Insbesondere entfällt deshalb die Überprüfung von TSLint; zumal das Projekt veraltet ist und die Entwicklung zugunsten von ESLint aufgegeben wurde [2] [1]. Da es aber noch automatisch mit Angular zusammen installiert wird, hätte man seine Überprüfung dennoch in Betracht ziehen können.

- **Prettier** ist ein Codeformatierer, der unter freiwilliger Angabe von Konfigurationseigenschaften Code einheitlich formatiert (d.h. Einrückungen werden korrigiert, Leerzeichen vor oder hinter Klammern eingefügt oder entfernt etc.). Gegenüber ESLint hat Prettier den Vorteil, nicht nur JavaScript-basierte Dateien formatieren zu können, sondern es kann auch mit anderen Dateiformaten wie Markdown oder CSS umgehen.

⁵<https://palantir.github.io/tslint/>

Darüber hinaus kann Prettier auch ohne Konfiguration (d.h. ohne Festlegung von Präferenzen) aufgerufen werden um Code zu formatieren. ESLint hingegen muss ausgiebig konfiguriert werden, damit der Code formatiert wird. Zudem hat Prettier mehr Einstellmöglichkeiten als ESLint (zumindest in Bezug auf Formatierung).

- **SCSS / Sass**⁶ sind zwei Dialekte eines CSS-Präprozessors. Dieser Präprozessor erweitert die CSS-Syntax um eine weitere Art von Variablen, Verschachtelung, sogenannten "Mixins" und weitere Features. Zur Kompilierzeit wird daraus normales CSS erzeugt. Dank der zusätzlichen Funktionen kann kompakteres, wiederholungs-freieres und oft auch leserlicheres CSS geschrieben werden.

Zugunsten von SCSS / Sass wurde auf die weitere Überprüfung anderer Präprozessoren wie Less⁷ verzichtet.

- **Jest** ist eine Bibliothek zum Schreiben und Ausführen automatisierter Tests. Diese werden in Node.js ausgeführt und verfügen daher a priori nicht über Browserspezifische APIs (wie z.B. Zugriff auf Canvas-Elemente). Viele dieser Funktionalitäten sind aber ggf. über weitere Bibliotheken nachrüstbar. In der Regel wird Jest für Unittests verwendet.

Im Vergleich zu anderen Tools vereint Jest mehrere Funktionalitäten, die sonst über mehrere Abhängigkeiten aufgeteilt sind. Wo sonst der Ausführer der Tests von der Bibliothek zur Aufstellung und Überprüfung von Annahmen (Engl.: „Assertion Library“) und der Mockingbibliothek getrennt ist und zur Erzeugung eines Berichtes über die Testabdeckung eine weitere Bibliothek notwendig ist, bietet Jest alle diese Features gleichzeitig an.

Seit 2019 ist Jest das meist verbreitetste Werkzeug zum Schreiben von JavaScript-Tests [7] und unterstützt dabei alle drei Frameworks. Außerdem ist seine Installation vergleichsweise sehr simpel, da hier nicht mehrere Tools zusammen installiert und miteinander eingerichtet werden müssen. Da diese höhere Komplexität also vermeidbar ist, werden andere Library-Kombinationen für Unittests nicht näher betrachtet.

- **Cypress** ist ein weiteres Werkzeug zum Schreiben automatisierter Tests, allerdings mit dem wichtigen Unterschied zu Jest, dass die Tests im Browser ausgeführt werden. Aktuell ist Cypress in der Lage, Chromium-basierte Browser und Firefox zu kontrollieren, um das Laden und die Interaktion mit einer Seite nicht nur per Node.js zu simulieren, sondern entsprechende Ereignisse in einem Browser so auszulösen, dass sie nicht (in relevantem Ausmaß) von tatsächlichen Userinteraktionen unterschieden werden können. Außerdem stehen in diesen Tests die Browser-APIs in vollem Umfang zur Verfügung.

⁶<https://sass-lang.com/>

⁷<https://lesscss.org/>

Seit 2020 wird Cypress öfter als vergleichbare Projekte wie Puppeteer oder WebdriverIO verwendet [7], weshalb diese zugunsten von Cypress nicht weiter berücksichtigt werden.

- **Husky und lint-staged** ist eine Kombination von Tools, die die Ausführung anderer Tools als Reaktion auf bestimmte Ereignisse ermöglichen.

Einige der zuvor aufgeführten Tools bieten Leistungen an, die dabei helfen können, sicherzustellen, dass in ein Versionskontrollsystem bzw. Engl.: Version Control System (VCS) eingepflegter Code immer gewissen Standards entspricht. Die automatischen Tests können beispielsweise garantieren, dass kein zuvor existierendes (und von Tests abgedecktes) Feature kaputt gemacht worden ist, und Prettier und ESLint können garantieren, dass der Code einem gewissen Stil entspricht. Andere Werkzeuge wie CSS-Präprozessoren hingegen bieten in dieser Hinsicht keine weiteren Vorteile.

Um die Qualität des Codes auf einem möglichst hohen Niveau zu halten, kann es also sinnvoll sein, diese entsprechenden Tools automatisch vor der Einpflegung des Codes in ein VCS ausführen zu lassen. Genau diese Möglichkeit bietet Husky, sofern das VCS Git verwendet wird. Diese Annahme lässt sich durchaus treffen, da Git mit einem Abstand von ??? das am weitesten verbreitete VCS ist **ZITAT FEHLT**.

Mithilfe von Husky lassen sich mit wenig Konfigurationsaufwand Skripte festlegen, die vor dem Speichern von Code in Git ausgeführt werden sollen. Scheitert eines dieser Skripte, so wird der Speichervorgang abgebrochen. Lint-staged erleichtert hierbei das Überprüfen der geänderten Dateien, sodass nicht bei jeder Änderung der gesamte Code geprüft werden muss.

Aufgrund dessen, dass diese beiden Tools empfohlen werden, um Prettier automatisch vor entsprechenden Vorgängen laufen zu lassen, und da ihre Verwendung sich leicht auf zusätzliche Tools erweitern lässt, werden Husky und lint-staged vergleichbaren Alternativen vorgezogen.

Außerdem wurden die folgenden Bibliotheken installiert:

- **Router** - Jedes der drei Frameworks verfügt über eine Unterbibliothek um verschiedene Routen einzurichten. So kann man ohne ein Neuladen der Seite die URL wechseln, diese Änderung auch im Browserverlauf widerspiegeln lassen und auch schon beim Aufruf einer Unter-URL direkt die entsprechende Komponente anzeigen lassen. Schon die Tatsache, dass die Installation eines Routers in Angular- und Vueprojekten schon bei der initialen Einrichtung ausgelöst werden kann, zeigt, dass Router sehr häufig verwendet werden. Für ein jeweiliges Framework gibt es außerdem in der Regel genau einen geläufigen Router, sodass sich hier keine besondere Bevorzugung stattgefunden hat. Stattdessen wurden alle drei Router gleichermaßen untersucht.

- **Redux** - Wie bereits erläutert, verfügt React nicht über eine besonders empfohlene Methode zum zentralen Statemanagement. Diese Lücke kann jedoch von Redux gefüllt werden. Mithilfe von Redux kann ein zentraler sogenannter SStore erzeugt werden. Dieser kann modifiziert werden, indem man an anderer Stelle eine sogenannte Action erzeugt, die an den Store weitergeleitet wird. Daraufhin werden sogenannte "Reducer aufgerufen, die den aktuellen Wert des Stores und die aktuelle Action entgegennehmen und einen neuen, entsprechend der Action aktualisierten Store zurückgeben.

Diese Reducer sind stets pure Funktionen⁸ und sind daher (da sie keinen Initialisierungsprozess voraussetzen und nichts anderes beeinflussen) sehr leicht testbar. Außerdem ermöglicht Redux in Kombination mit einer Browsererweiterung mehrere interessante Features, darunter ein Feature namens SZeitreise", mit dem man den Store zu einem beliebigen vorherigen Stand zurücksetzen kann.

Redux lässt sich als eine Implementierung des Kommandopatterns **ZITAT FEHLT** betrachten. Hierbei stellen die Actions die Kommandos dar und der Store ist der Ausführer, der auch die Kommandohistorie verwaltet. Entsprechend stellt Redux auch automatisch eine Aufwandslose Undo-Redo-Funktionalität zur Verfügung.

Da Redux die meistverwendete Bibliothek für Statemanagement ist [7], wird ihre Installation stellvertretend für Alternativen wie MobX betrachtet. Ausnahme hiervon bildet jedoch Vuex, da es eine sehr hohe Ähnlichkeit zu Redux aufweist, sich mit besonders wenig Aufwand in einem Vue-Projekt installieren lässt und es für Vue-Projekte anstelle von Redux empfohlen wird [3].

- **Komponentenlibraries** - Gerade für kleine Projekte, in denen keine eigenen Designer angestellt sind, kann es sich sehr lohnen, eine Komponentenlibrary einzubinden. Da diese jeweils für ein Framework Komponenten bereitstellen muss, gibt es in der Regel für verschiedene Design Systeme jeweils eine Bibliothek pro Framework.

Stellvertretend für andere Komponentenlibraries wurden drei Material-Design-Libraries untersucht: Material UI für React, Angular Material für Angular und Vuetify für Vue.

- **Paper.js** - Um abschließend noch eine Bibliothek zu untersuchen, die an sich keinerlei Bindung an Frameworks hat aber trotzdem in die Applikation eingebunden werden muss, wurde willkürlich als Beispiel paper.js ausgewählt. Diese Library erleichtert den Umgang mit dem HTML Canvas.

⁸D.h. sie sind seiteneffektfrei und ergeben bei gleicher Eingabe immer die gleiche Ausgabe.

3.2 Vergleich der Installationsprozesse

Beim manuellen Installieren der verschiedenen Tools und Libraries sind einige Gemeinsamkeiten und vor allem in den Details viele Unterschiede aufgefallen. Auf einer allgemeineren, konzeptuellen Ebene lässt sich aber vor allem (wie bereits geschehen) zwischen Werkzeugen und Bibliotheken unterscheiden.

Allgemeinhin müssen für beides zunächst NPM-Pakete installiert werden. Daraufhin benötigen aber Werkzeuge meist die Änderung von Konfigurationsdateien, während zur Einbindung von Bibliotheken eher der produktive Code modifiziert werden muss.

3.2.1 Installationen von Werkzeugen

Zur Installation von Prettier ist beispielsweise lediglich noch die Erstellung einer Konfigurationsdatei für Prettier notwendig. Theoretisch kann auch auf diese Verzichtet werden, aber da ein Ziel des Projektes ist, dass Nutzende alles Wichtige für die meiste Arbeit eingerichtet bekommen, ist es sinnvoll, die Datei bereits zu erstellen und mit sinnvollen Standardwerten zu füllen. Darüber hinaus ist es ratsam, zur leichteren Ausführung von Prettier (wie bei den meisten anderen Werkzeugen auch) ein NPM-Script zu erzeugen. Hierfür muss die package.json-Datei modifiziert werden können.

Ähnlich ist bei der Installation von ESLint vorzugehen. In der Regel bedarf es lediglich der Installation des Paketes, der Einrichtung eines entsprechenden Scripts und der Erzeugung der Konfigurationsdatei. Leider wird der Installationsprozess bei der Benutzung von Angular dadurch verkompliziert, dass hier standardmäßig zunächst das veraltete TSLint eingebunden ist. Allerdings gibt es bereits ein NPM-Paket⁹, was den Wechsel auf ESLint automatisch durchführen kann.

Mit überdurchschnittlich wenig Aufwand lässt sich Sass / SCSS installieren. Sowohl Angular als auch Vue verfügen hierzu über einen Parameter bei der initialen Installation. Bei React muss man zwar das node-sass Paket installieren, aber weitere Einrichtung muss nicht vorgenommen werden.

Überdurchschnittlich viel Aufwand verursachten dahingegen Jest und Cypress. In Kombination mit React wird Jest zwar automatisch installiert und im Vue-CLI gibt es hierfür eine entsprechende Option. Allerdings setzt Angular bisher auf Karma als Testausführer und Mocha als Assertion Library, weshalb hier nicht nur eine Installation von Jest sondern auch eine Anpassung bestehender Scripte und alter Tests notwendig ist. Wie auch schon bei ESLint gibt es ein Paket, was den Umstieg bis auf die Aktualisierung der Tests selber automatisch durchführen kann. Da im Kontext dieses Projekts bei der Ausführung noch keine speziellen Tests entstanden sein können, kann diese Aufgabe hier durch ein schlichtes Austauschen von festgelegten, vorgefertigten Dateien erledigt werden.

Die Installation von Cypress ist vergleichsweise zwar simpel und benötigt lediglich das Installieren neuer NPM-Pakete, das erstellen eines Scripts in der package.json-Datei und

⁹<https://github.com/angular-eslint/angular-eslint>

das Erzeugen neuer Dateien. Dieser Prozess kann im Kontext von Vue sogar von der initialen Projekterstellung abgenommen werden und auch in Angular-Applikationen kann er mithilfe eines NPM-Pakets automatisiert werden. Allerdings ergibt sich in genau der Kombination (d.h. in Angular-Projekten) eine Komplikation, wenn auch Jest installiert wurde. Dieses Problem lässt sich durch Modifikation der `jest.config.js`-Datei beheben [5].

Da das Schreiben von Test sowohl für Jest als auch für Cypress sowohl in JavaScript als auch in TypeScript erfolgen kann, sollten die automatisch eingerichteten Tests in derjenigen der beiden Sprachen sein, die auch für den Produktivcode verwendet wird. Es ist also bei der Installation notwendig zu wissen, ob TypeScript verwendet wird oder nicht.

Eine letzte Besonderheit ergibt sich bei der Installation von Husky und lint-staged. Hier sind zwar ähnliche Schritte wie bei anderen Tools notwendig, aber zusätzlich wird hier eine Liste von Befehlen bzw. Scripten benötigt, die vor bestimmten Git-Prozessen ausgeführt werden sollen.

Zusammenfassend ist für die Installation von Tools also eine Kenntnis über die Verwendung von TypeScript nötig. Es muss auch bekannt sein, ob unter den anderen ausgewählten Werkzeugen solche dabei sind, die vor Git-Prozessen ausgeführt werden sollen. Darüber hinaus ist die Installation von NPM-Paketen sowie die Modifikation einiger Dateien notwendig. Diese Dateien sind entweder im JavaScript Object Notation (JSON)-Format oder sind JavaScript- / TypeScript-Dateien. Außerdem muss es möglich sein, neue Dateien mit vorbestimmtem Inhalt zu erzeugen, der sich an den JavaScript-Dialekt anpassen können muss.

3.2.2 Installationen von Bibliotheken

Auch bei der Installation von Bibliotheken gibt es viele Gemeinsamkeiten, aber hier sind deutlich größere Unterschiede zwischen den Frameworks zu vermerken. Dies ist vorallem dadurch zu begründen, dass die Benutzung der Bibliotheken innerhalb bestimmter Komponenten geschieht und daher entweder bestehende Komponenten modifiziert oder neu erstellte Komponenten eingebunden werden müssen. Aufgrund dieser Bindung an Komponenten muss das entsprechende Vorgehen an jedes Framework einzeln angepasst werden.

Ein gutes Beispiel hierfür ist die Installation von `paper.js`. Da es sich hier nur um eine spezielle Library handelt, ist sie die einzige, die in keinem Framework vorinstallierbar ist, und eignet sich daher besonders gut für diesen Vergleich. Zur sinnvollen Benutzung von `paper.js` ist es notwendig, einen Canvas auf der Webseite anzuzeigen. Sobald dieser Canvas existiert, kann `paper.js` initialisiert werden und etwas auf dem Canvas zeichnen (zu Demonstrationszwecken genügt ein einfaches Rechteck).

Hierfür lässt sich in allen drei Frameworks eine Komponente schreiben, die dann in die Hauptkomponente eingebettet werden muss. Die dafür erforderlichen Dateiveränderungen sind jedoch sehr verschieden, was am Beispiel von Vue in Listing 1 dargestellt wird.

```

1  // HelloWorld.vue
2  <template>
3    <div class="hello">
4      <!-- ... -->
5
6  +    <PaperJsExample />
7
8      <h3>Installed CLI Plugins</h3>
9      <!-- ... -->
10   </div>
11 </template>
12
13 <script lang="ts">
14   import { defineComponent } from "vue";
15 + import PaperJsExample from "@/components/PaperJsExample.vue";
16
17   export default defineComponent({
18     name: "HelloWorld",
19     props: {
20       msg: String,
21     },
22 +   components: {
23 +     PaperJsExample,
24 +   },
25   });
26 </script>
27
28 <!-- ... -->

```

Listing 1: In Vue notwendige Änderungen, um eine Komponente hinzuzufügen

Wie man hier sieht, muss die zu verwendende Komponente zum einen dort eingefügt werden, wo sie anschließend im DOM erscheinen soll. Zum anderen muss sie im Script-Block importiert werden. Danach folgt eine Besonderheit von Vue: es muss deklariert werden, dass die Komponente verwendet werden wird (vgl. Zeilen 22 - 24 in Listing 1). Dieses Attribut des Konfigurationsobjektes muss bei der ersten Komponente, die eingefügt werden soll, neu erzeugt werden, während es bei allen weiteren neuen Komponenten nur ergänzt werden muss.

Dieses eine Codebeispiel zeigt bereits, dass es schon für simple Installationen von Bibliotheken notwendig ist, framework-spezifischen Code modifizieren zu können. Hierfür wäre es hilfreich, ein tiefes Verständnis des zu modifizierenden Codes zu haben. Da aber die Struktur des vorher existierenden Codes bekannt ist (unter der Annahme, dass sich die initialen Setups nicht auf relevante Art und Weise verändern werden) und da die vorzunehmenden Änderungen sich innerhalb eines Frameworks immer ähneln, kann diese geringe Menge von Änderungen bzw. Änderungsarten auch ohne ein solches tiefes

Verständnis umgesetzt werden. Genauer wird hierauf in Kapitel 4 eingegangen.

Die Komponentenlibraries lassen sich mehrheitlich ohne bereits bekannte Probleme einbinden; häufig sogar automatisch bei der initialen Projekterzeugung mittels des Framework-spezifischen CLI's.

Neue Probleme weisen jedoch die Einbindung von Redux und Routern auf. Diese Libraries bauen beide darauf auf, dass gewisse Informationen global verfügbar sind (in Redux der Store und in Routern die aktuelle Route).

In React ist das umsetzbar, indem die oberste Komponente von sogenannten "Providern" umgeben wird, die einen Wert und einen zugehörigen Schlüssel entgegennehmen und dann jeder ihnen unterliegenden Komponente den Wert unter dem jeweiligen Schlüssel zur Verfügung stellen. Das Umgeben der obersten Komponente ist also notwendig, damit alle Komponenten der gesamten Applikation Zugriff auf diese Werte haben. Falls dies nicht gewünscht ist, kann der zugehörige Provider verschoben werden.

In Angular ist es für eine sinnvolle Verwendung von Redux sinnvoll, einen Angular-Service einzurichten. Dies ermöglicht die einfache Erstellung eines Singleton **ZITAT FEHLT** Store-Objektes und erlaubt es gleichzeitig, mit Angular-typischen Mitteln wie RxJS¹⁰-Methoden auf den Store zuzugreifen. Im Grunde genommen handelt es sich hierbei aber erneut nur um das Erstellen einer Datei mit vorbestimmtem Inhalt.

Im Zusammenhang mit Vue fallen keine weiteren Probleme auf, da hier für Redux Vuex substituiert wird und sowohl Vuex als auch der Vue Router bereits automatisch installierbar sind.

3.2.3 Zusammenfassung der notwendigen Installationsschritte

Über die Werkzeuge und Bibliotheken hinweg zeigt sich, dass sich vieles über einfache Befehle installieren lässt, ohne, dass ein weiterer Aufwand betrieben werden muss. Für andere Installationen ist es ausnahmslos notwendig, gegebene Pakete von NPM installieren zu können. Es muss möglich sein, neue Dateien an bestimmten Orten zu erzeugen und diese mit Inhalten zu füllen, die sich nach der getroffenen Auswahl von Werkzeugen und Bibliotheken richten können.

Außerdem muss die Modifikation bestimmter Dateien ermöglicht werden. Zum einen muss es möglich sein, verschiedene JSON-Dateien zu verändern. Zum anderen muss frameworkspezifischer Code ergänzt und modifiziert werden können. Außerdem muss die Reihenfolge der Installation der Tools bzw. Werkzeuge festlegbar sein, da beispielsweise Husky und lint-staged als letzte Werkzeuge installiert werden sollten.

¹⁰<https://rxjs.dev/>

3.3 Wahl des generellen Vorgehens

3.4 Planung der Erweiterungen

3.5 Abwägung über Sonderstellung für TypeScript und Frameworks

4 Implementierung

4.1 Abhängigkeiten und Exklusivitäten

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.

4.1.1 Überprüfung der Umsetzbarkeit

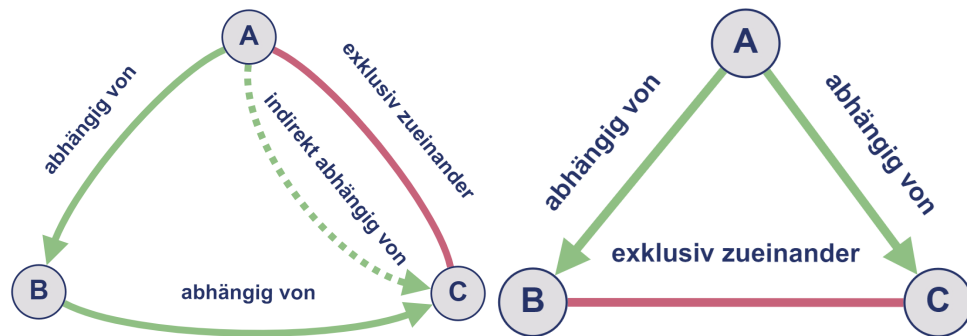
Ein triviales Beispiel für unmögliche Kombinationen von Bedingungen wäre ein Paar von Abhängigkeiten A und B , wobei A und B exklusiv zueinander sind und gleichzeitig A von B abhängt. Es können in diesem Beispiel niemals beide Anforderungen gleichzeitig erfüllt werden.

Natürlich lassen sich aber auch kompliziertere Beispiele erzeugen. In Abbildung 3 werden zwei solche Konstellationen 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 ebenso wenig 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 3 dargestellte Konflikt beruht im Gegensatz zu den bisherigen Beispielen nicht darauf, dass eine Erweiterung zugleich (indirekt) abhängig und exklusiv 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



- (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 3: Zwei Graphen zur Veranschaulichung von Konfliktmöglichkeiten zwischen Abhängigkeiten und Exklusivitäten.

berechnet. Aufgrund eben dieser Transitivität sind jedoch die Berechnungen für alle Erweiterungen, 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 Exklusivität 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 Exklusivität 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 [9]. 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] Archivierungsbanner und Angabe des letzten Commits auf den master-Branch am 25. März 2021. <https://github.com/palantir/tslint> [Zugriff am 29. August 2021].
- [2] R. F. u. S. Y. Adi D., John W., Feb. 2019. <https://blog.palantir.com/tslint-in-2019-1a144c2317a9> [Zugriff am 29. August 2021].
- [3] C. F. et al. State management - information for react developers. <https://vuejs.org/v2/guide/state-management.html#Information-for-React-Developers> [Zugriff am 30. August 2021].
- [4] J. Henry, Apr. 2021. <https://github.com/angular-eslint/angular-eslint/issues/395#issuecomment-821991202> [Zugriff am 28. August 2021].
- [5] L. M. Kerr. Github-issue: Jest schematic + cypress schematic conflicts. <https://github.com/briebug/jest-schematic/issues/51> [Zugriff am 30. August 2021].
- [6] R. Mello, Apr. 2021. <https://github.com/ricmello/prettier-schematics/blob/main/src/collection.json> [Zugriff am 28. August 2021].
- [7] Sacha Greif und Raphaël Benitte. State of JS 2020, 2020. <https://2020.stateofjs.com/en-US/> [Zugriff am 28. August 2021].
- [8] S. V. und Kamil Kisiela, Aug. 2018. <https://github.com/kamilkisiela/apollo-angular/blob/master/packages/apollo-angular/schematics/collection.json> [Zugriff am 28. August 2021].
- [9] 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.