

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Bachelorarbeit Informatik

Titel der Arbeit

Mike Hengge

Datum

Gutachter

Name Gutachter

Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Betreuer

Name Betreuer

Adresse
Universität Tübingen

Nachname, Vorname:

Titel der Arbeit

Bachelorarbeit Informatik

Eberhard Karls Universität Tübingen

Bearbeitungszeitraum: 15.11.2018-15.03.2019

Zusammenfassung

Hier kommt die Zusammenfassung hin!!!

Danksagung

Hier kommen die Danksagungen hin!!!

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface, hier: Schnittstelle zwischen Anfrage und Programm
DOM	Document Object Model, JavaScript Programmierschnittstelle für dynamische Webentwicklung

Kapitel 1

Einleitung

In den letzten Jahrzehnten hat sich das Internet über die ganze Welt verbreitet. Aktuell dient das Internet vielerorts als primäres Informations- und Kommunikationsmedium. Dabei wandelten sich die Ansprüche an dieses Medium mit dem technologischen Fortschritt. Mit leistungsfähigerer Hardware, sowohl auf Nutzer als auch auf Entwicklerseite, und schnelleren Internetverbindungen wurden neue Inhalte und neue Wege der Darstellung möglich.

Dank dieser neuen Möglichkeiten werden mehr und mehr Anwendungen über das Web angeboten, welche vorher nur als Desktop Applikationen zur Verfügung standen. Um solche Webapplikationen zu entwickeln werden oftmals spezielle JavaScript-Bibliotheken und -Frameworks verwendet. Beispiele sind Angular, Vue, React noch viele mehr. React zeichnet sich dabei derzeit durch das größte Wachstum aus.

React ist eine open-source Java Bibliothek und wurde von Facebook entwickelt um möglichst performante Oberflächen zu gestalten. Seine Beliebtheit gründet sich auf der einfachen und schnellen Nutzung, mit hochperformanten Ergebnissen. Dabei beschränkt sich React auf den View-Teil der klassischen Model-View-Controller Struktur. Diese View besteht aus einzelnen Komponenten von denen jeder seinen eigenen 'State' verwaltet. Ändert sich diese States werden nur die betroffenen Komponenten neu gerendert, einer der Gründe für Reacts hohe Effizienz.

Die vorliegende Arbeit erfüllt zwei Funktionen und ist daher auch inhaltlich zweigeteilt. Zum einen sind in ihr Ergebnisse der Recherche und Evaluation von React aufgeführt, während sie sich zum anderen mit der Umsetzung des im Folgenden beschriebenen Projekts beschäftigt.

Ziel des Recherche-Teils ist es, Reacts Eigenschaften, Vorteile, Nachteile und generelle Nutzbarkeit darzustellen. Dieser Teil soll dem Anspruch gerecht werden, eine fundierte Entscheidung für oder gegen eine eigene Nutzung von React zu ermöglichen.

Bei dem Projekt handelt es sich um die Erstellung einer web-basierten Anwendung mittels JavaScript und React aus einer bestehenden stand-alone Java Implementation „Logik Lehrtools“. In dieser werden in einer Nutzeroberfläche die Algorithmen Resolution, Backward Dual Resolution und Variablen-Elimination aus der Aussagenlogik auf dort eingegebene Formeln angewandt und die Zwischenergebnisse und Endergebnisse der Algorithmen ausgegeben.

Das entstehende Online-Tool soll für die Lehre einsetzbar sein und eine Möglichkeit für Studenten bieten Beispiele selbst nachzuvollziehen, den Ablauf der einzelnen Algorithmen zu beobachten und eigene Lösungen zu kontrollieren. Hierfür wird es nach Fertigstellung über einen Webserver im Universitätsnetzwerk zur Verfügung gestellt.

Kapitel 2

React - Theorie und Evaluation

2.1 Motivation

Die Entstehung von React ist direkt verknüpft mit JavaScripts Problemen, die speziell bei großen Anwendungen auftreten. Das von JavaScript verwendete DOM ist bei oftmaligen Zustandsänderungen langsam und ineffizient. Durch den Aufbau des DOM muss dieses auch bei kleinen Änderungen in Gänze neu geladen werden. Es gilt außerdem allgemein als fehleranfällig und unnötig kompliziert und ist für die Webentwicklung nicht sehr gut geeignet.

Hinzu kommt, dass JavaScript-Objekte nicht funktional sind, das heißt ihr Zustand ist nicht nur vom Input und darauf ausgeführten Funktionen abhängig. Stattdessen teilen sich Objekte oftmals Zustände und verändern sich so miteinander gegenseitig. Daher war Frontendcode vor React oft unverständlich und schwer wartbar.

Das Ziel der Entwicklung von React war es deswegen weniger komplexen und gut lesbaren Code zu ermöglichen, der Zustände persistent und effizient verwaltet. Die sonst oft in der Webentwicklung verwendeten Templates sollten ersetzt werden, durch eine Lösung die mehr Flexibilität bietet und es gleichzeitig durch kompaktere Architektur erleichtert Applikationen auszuweiten.

2.2 Architektur

Laut den Entwicklern Im Vergleich zu den bis dato gängigen JavaScript Frameworks und JavaScript selbst, weist React einige Besonderheiten in seiner Architektur auf. Die wichtigsten dieser Besonderheiten werden hier im Folgenden aufgeführt und diskutiert. Es gibt noch einige weitere Eigenheiten Reacts, doch diese vier sind die ausschlaggebenden Merkmale für dessen Erfolg und stehen deshalb im Fokus dieser Evaluation.

2.2.1 Virtual DOM

In der Webentwicklung hat die DOM-Manipulation eine ausschlaggebende Rolle inne. Mit ihr kann der Elementbaum einer Webseite ausgelesen, verändert und erweitert werden. Wie oben erwähnt ist das JavaScript DOM problematisch.

React löst dieses Problem mittels dem sogenannten Virtual DOM, welche ein entsprechendes virtuelles Model des eigentlichen DOM ist. Jedes DOM-Objekt wird darin über ein ähnliches oder vereinfachtes virtuelles Objekt dargestellt. Obwohl es in seinen Eigenschaften dem DOM stark ähnelt, kann das virtuelle DOM keine direkten Änderungen am Elementbaum durchführen. Stattdessen wird seine erheblich bessere Performanz als Änderungsbuffer benutzt. Wird ein React Komponent (siehe 2.3) gerendert, so wird ein Snapshot des aktuellen Zustands gemacht und alle virtuellen DOM-Objekte werden aktualisiert. Dieser Prozess ist weitaus schneller als das Aktualisieren des eigentlichen DOM, da keine tatsächlich angezeigten Elemente verändert werden und beeinflusst die Performanz des restlichen Systems nicht. Nachdem so alle virtuellen DOM-Objekte aktualisiert wurden, werden diese mittels dem sogenannten 'Diffing' mit dem Snapshot verglichen. Lediglich der Zustand der Objekte der sich zwischen diesen Schritten verändert hat, wird vom virtuellen DOM in das DOM übernommen. In Abbildung 2.1 ist dieser Prozess in einem Schaubild dargestellt.

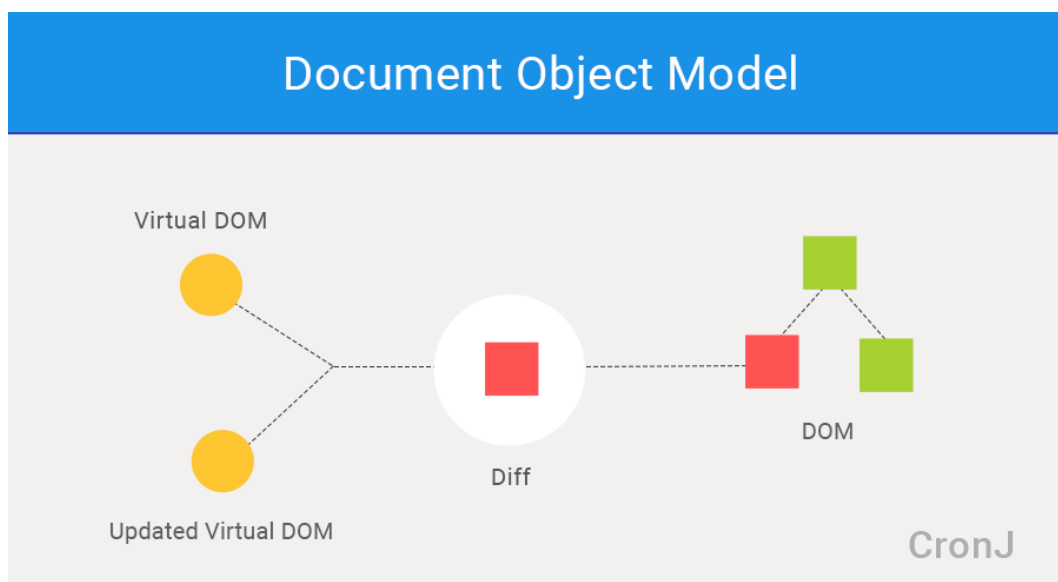


Abbildung 2.1: React Virtual DOM

Resultat ist eine erheblich schnellere, effizientere und performantere DOM-Manipulation als frühere Lösungen. Das Virtual DOM kann fraglos als einer

der Hauptgründe für Reacts Beliebtheit bezeichnet werden.

2.2.2 Komponenten

2.2.3 Unidirektionaler Datenfluss

Anders als viele der beliebten Frameworks verwendet React keinen bidirektionalen Datenfluss. Ebenso erstellt es in dem bekannten Model-View-Controller Konzept nur die View-Komponente und überlässt dem Entwickler die Wahl, ob und welche weiteren Komponenten verwendet werden. In einem bidirektionalen Datenfluss sind Model und View in direktem gegenseitigem Austausch, in dem Änderungen an einer Seite entsprechende Änderungen auf der anderen Seite auslösen. Diese gängige Lösung wird in den meisten Fällen gute Leistungen und Ergebnisse erzielen. Jedoch kann es hier zu unvorhersehbaren Datenflüssen kommen, eben weil sowohl Model als auch View Einfluss auf den Zustand der Anwendung haben. Es kann so zu nicht beabsichtigten Änderungen in nicht bearbeiteten Model-Bereichen und zu Kettenreaktionen von Updates kommen.

In React können Daten stattdessen nur in eine Richtung gegeben und verarbeitet werden, React ist also unidirektional. So wird ein Jonglieren der Zustände vermieden.

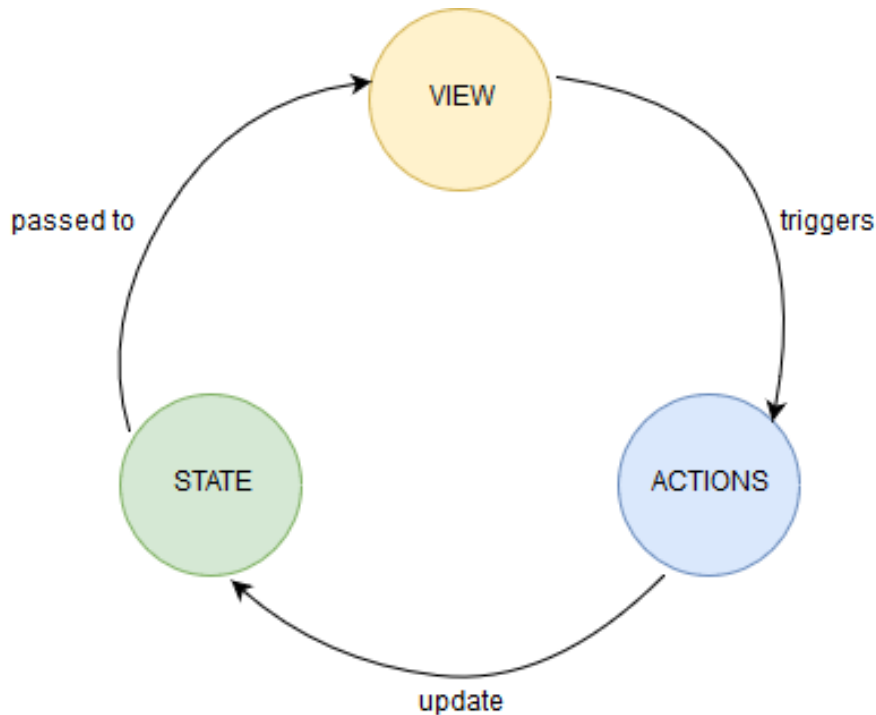


Abbildung 2.2: Unidirektionaler Datenfluss in React

Wie in Abbildung 2.2 zu sehen ist, fließen Daten hier in einem Kreislauf durch Zustand, View und Aktionen. Der Anwendungszustand, der die Zustände der Komponenten beinhaltet, wird an die View und ihre Child-Komponenten weitergegeben. In dieser werden Aktionen getriggert, beispielsweise durch den Nutzer, welche den Zustand verändern können.

Wie in 2.1 erwähnt konnte es vor React oftmals schwierig sein den Datenfluss in einer Anwendung nachzuverfolgen. Der unidirektionale Datenfluss Reacts schafft hier eine größere Nachvollziehbarkeit und begünstigt so Analyse und Fehlersuche.

2.2.4 JSX


JSX ist eine Syntax-Erweiterung zu JavaScript die für React empfehlenswert ist, da sie durch ihre Ähnlichkeit mit HTML-Syntax leicht zu lesen und verwenden ist. Der Typ eines Elements wird wie in HTML in einem öffnenden Tag festgelegt. Zwischen diesem öffnendem und dem abschließend folgenden schließendem Tag können die Kinder des Elements hinzugefügt werden. In Abbildung 2.3 ist als Beispiel eine ungeordnete Liste in JSX dargestellt. In

```
<ul>
  <li>1 lb Salmon</li>
  <li>1 cup Pine Nuts</li>
  <li>2 cups Butter Lettuce</li>
  <li>1 Yellow Squash</li>
  <li>1/2 cup Olive Oil</li>
  <li>3 cloves of Garlic</li>
</ul>
```

Abbildung 2.3: Ungeordnete Liste in JSX

JSX werden JavaScript-Ausdrücke in geschweifte Klammern gestellt. Die Anzeige des Titel-Felds eines Elements wäre beispielsweise wie in Abbildung 2.4 umzusetzen.

Die Ausdrücke werden als vollwertiges Skript ausgeführt, es ist daher auch möglich vorher definierte Funktionen und vorimplementierte JavaScript Operationen an dieser Stelle zu verwenden. Da JSX eine Erweiterung zu JavaScript ist, kann in derartigen Funktionen auch JSX-Syntax verwendet werden. Wie in Abbildung 2.5 gezeigt, könnte beispielsweise eine Funktion



```
<h1>{this.props.title}</h1>
```

Abbildung 2.4: JavaScript in JSX

setResult erstellt werden, die einen erhaltenen String untersucht und abhängig von dessen Inhalt zwei unterschiedliche JSX-Sequenzen ausgibt.

2.3 React Native

2.4 Stärken

2.5 Kritik

2.6 Alternative Frameworks und Bibliotheken

2.7 Fazit

Kapitel 3

React in der Praxis

3.1 Entwicklungsumgebung

3.1.1 Nodejs

Nodejs ist ein JavaScript Runtime Environment für die serverseitige Entwicklung mit JavaScript. Für Projekte mit React ist es vor allem wegen dem enthaltenen Paketmanager NPM wichtig. Dieser ist aktuell de-facto Standard zur Verteilung von Paketen in JavaScript und bietet die Möglichkeit die React-Applikation während der Entwicklung direkt über einen integrierten Webserver anzuzeigen. Ein weiterer wichtiger Vorteil von NPM ist die einfache Installation vieler Erweiterungsmodule, die spezielle Funktionen bieten und häufige Problemstellungen lösen können.

Der oben erwähnte Webserver ist die einfachste Möglichkeit eine React-Applikation außerhalb einer Produktionsumgebung anzuzeigen. Anders als bei statischen Webseiten ohne Skriptanteil, können dynamische Webseiten nicht ohne eine Server-Client-Umgebung aufgerufen werden. Grund dafür ist, dass der dynamische Inhalt erst beim Aufruf vom Server generiert wird. Soll die React-Applikation so angezeigt werden reicht dank NPM der kurze Befehl `'npm run start'` um den Webserver zu starten und die Applikation anzuzeigen. Veränderungen im Code der Applikation werden dabei live compiliert, was die Fehlersuche und das Testen neuer Komponenten stark erleichtert. Wie in Abbildung 4.1 zu sehen ist, steht die Applikation unter einer lokalen URL zur Verfügung und könnte durch einen weiteren kurzen Befehl `'npm run build'` für die Produktion optimiert werden.

```
Compiled successfully!  
  
You can now view react-hello-world in the browser.  
  
Local:      http://localhost:3000/  
On Your Network:  http://192.168.56.1:3000/  
  
Note that the development build is not optimized.  
To create a production build, use npm run build.
```

Abbildung 3.1: Laden von React (2 Zeilen) & Button.js (1 Zeile)

3.1.2 Toolchains für React

Für React ist eine Vielzahl von Toolchains zur Vereinfachung der Entwicklung verfügbar. Während es problemlos möglich ist ohne diese Werkzeug-Programme eine React Applikation zu entwickeln, sind sie in bestimmten Fällen sehr empfehlenswert. Besonders bei sehr große Projekte mit vielen Komponenten und/oder Drittpartei-Modulen sind entsprechende Toolchains wertvoll. Andere Vorteile sind die Vermeidung typischer Fehler im Projektaufbau und eine einfache Optimierung des Projekts für die Produktion.

Bekannte Toolchains sind unter anderem Create React App, Next.js und Gatsby. Create React App ist für die Entwicklung von single-page Applikationen wie dem LogikLehrtool-Projekt sehr gut geeignet. Next.js dagegen wird eher für die Entwicklung von serverseitig gerenderten Webseiten und Gatsby für statische Seiten verwendet. Darüber hinaus gibt es eine Vielzahl an weiteren Toolchains die sehr viele Anwendungsgebiete abdecken und auch für ungewöhnlichere Projekte eine passende Konfiguration erzeugen können.

3.2 React in HTML-Seiten einfügen

Es ist sehr einfach React einer bestehenden HTML-Seite hinzuzufügen. Im folgenden Beispiel wird einer HTML-Seite ein Like-Button hinzugefügt.

```
<!-- ... existing HTML ... -->
<div id="like_button_container"></div>
<!-- ... existing HTML ... -->
```

Abbildung 3.2: Erstellen eines Dom-Containers

```
'use strict';
const e = React.createElement;
class LikeButton extends React.Component {
  constructor(props) {
    super(props);
    this.state = { liked: false };
  }
  render() {
    if (this.state.liked) {
      return 'You liked this.';
    }
    return e(
      'button',
      { onClick: () => this.setState({ liked: true }) },
      'Like'
    );
  }
}
const domContainer = document.querySelector('#likeButton');
ReactDOM.render(e(LikeButton), domContainer);
```

Abbildung 3.3: Button.js: React-Script zur Verwendung des Dom-Containers

Wie in Abbildung 4.1 zu sehen ist, wird als Erstes ein Dom-Container in der HTML-Datei eingefügt. Dieser Container wird anschließend bei der Erstellung eines React-Scripts mittels seiner ID verwendet. Abbildung 4.2 zeigt dieses Script, welches einen Button mit einer Klick-Reaktion erzeugt. Wird der Button geklickt, so ändert sich sein Status und es wird stattdessen der Satz 'You liked this' angezeigt.

Abschließend müssen in der HTML-Datei, wie in Abbildung 4.3, nur noch die entsprechenden Skripte geladen werden. Dies beinhaltet zwei Skripte welche die React-Bibliothek laden und das eben erstellte Button-Skript welches unsere React-Komponente beinhaltet.

```
<!-- ... other HTML ... -->

<!-- Load React. -->
<!-- Note: when deploying, replace "development.js" with "production.min.js". -->
<script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>
<script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>

<!-- Load our React component. -->
<script src="like_button.js"></script>

</body>
```

Abbildung 3.4: Laden von React (2 Zeilen) & Button.js (1 Zeile)

Kapitel 4

Logik Lehrtools mit React

Dieses Kapitel beinhaltet den eingangs erwähnten zweiten Teil dieser Arbeit, das Logik-Lehrtools-Projekt. Dieser zweite Teil ist aufgeteilt in die Teilaufgaben die das Projekt mit sich brachte. Zunächst wird über ein Ablaufdiagramm gezeigt wann die einzelnen Komponenten bei der Nutzung zusammenspielen. Anschließend werden diese Komponenten detailliert aufgeführt. Dabei werden sowohl ihre Rolle im Ablauf, als auch Entscheidungen und Probleme der Projektdurchführung erläutert. Zuerst werden hier die durchgeführten Anpassungen am bereits vorhandenen Java-Programm vorgelegt und erklärt. Anschließend folgen die zwei Webentwicklungskomponenten: der verwendete Webserver und die React-Applikation. Abschließend dann die Server-Schnittstelle, die die Kommunikation zwischen Webseite und Java-Programm regelt.

4.1 Ablauf

In Abbildung 4 ist der Ablauf vom Aufruf der Seite bis zur Anzeige der Resultate dargestellt. Deutlich sichtbar ist, wie nach den ersten beiden Schritten die Kommunikation nicht mehr über den Webserver abläuft. Stattdessen wird die neben dem Server laufende API nunmehr zentral für den Austausch.

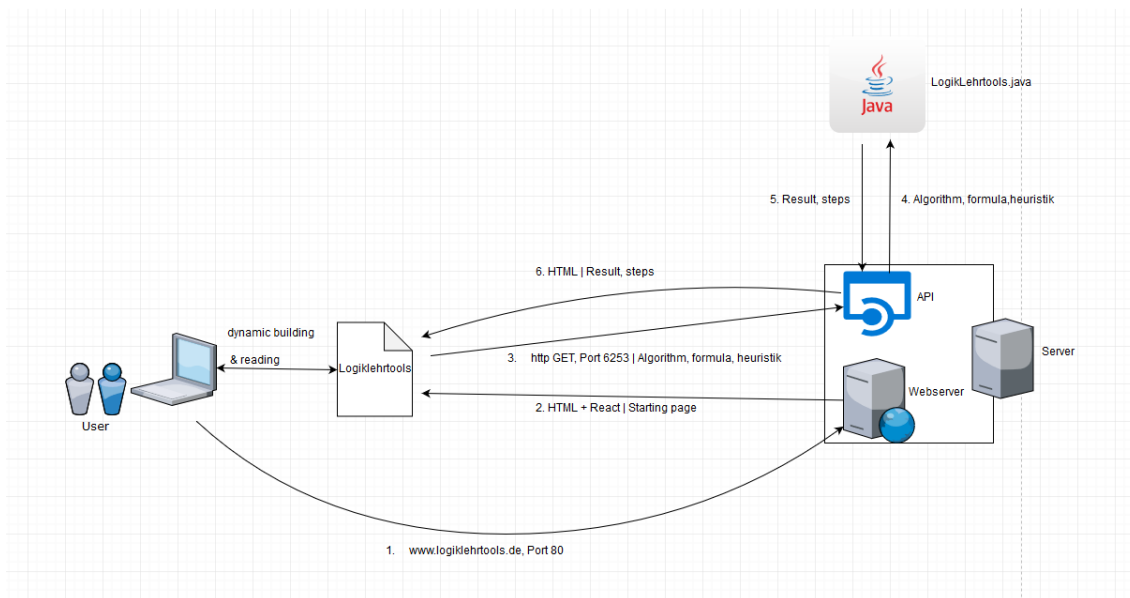


Abbildung 4.1: Ablauf der Anwendung

4.2 Anpassung der Java Software

Um das bereits vorliegende Java-Programm von Herrn Espinoza verwenden zu können waren einige Modifikationen daran notwendig. Herr Espinoza hatte seine Applikation mit Benutzeroberfläche entworfen und die logischen Abläufe an eine solche angepasst. Die Aktionen der Applikation wurden alleinig durch die Zustände und Nutzeraktionen dieser Benutzeroberfläche gesteuert. Während das für ein Standalone-Programm durchaus sinnvoll war, warf es Schwierigkeiten für die Webapplikation auf. Sind die Aktionen einer Webapplikation von früheren Aktionen oder Inputs abhängig, so ist eine sehr viel größere Menge an Kommunikation zwischen Server und Client nötig, ebenso wie eine komplexere Benutzeroberfläche und ein höheres Maß an Handlungen der Nutzer.

Für dieses Projekt war eine stateless Implementation, also eine in der das Output nur vom Input und nicht von verschiedenen Zuständen abhängig ist besser geeignet. Da die Grundidee des Tools die Anwendung verschiedener Algorithmen war. Da die Anwendung und Darstellung der Algorithmen festen

Regeln folgen, war eine Implementation, die dies widerspiegelt, eine passendere Wahl.

Einzelne Funktionen wie die Wahl der nächsten Resolutionsvariable zwischen einzelnen Schritten im gewählten Algorithmus und die Möglichkeit einen Schritt zurück zu gehen wurden daher entfernt. Stattdessen ist nun vor der Anwendung des Algorithmus die Angabe einer Heuristik möglich. So ist ebenfalls die Reihenfolge der Resolutionsvariablen frei wählbar, allerdings nur einmal vor Ausführung des Tools. Ist eine andere Reihenfolge gewünscht, muss der Algorithmus erneut mit einer anderen Heuristik gestartet werden.

Ebenso wurde die Wahl der Subsumption-Reihenfolge entfernt. Bisher konnte die Reihenfolge der beiden Möglichkeiten (forward und backward subsumption) gewählt werden. Da aber nach Ausübung einer von beiden stets die anderen zu folgen hatte und das Ergebnis nach Anwendung beider Varianten ohnehin dasselbe war, war diese Wahl unnötig. Stattdessen wird nun immer zuerst forward und dann backward subsumption durchgeführt.

Ebenso wurde die Auswahl der Eingabesyntax entfernt, da nicht zu erwarten ist, dass ein Nutzer verschiedene Eingabesprachen verwenden wird. Stattdessen scheint es wahrscheinlicher, dass sich Nutzer leichter an eine einzige Syntax gewöhnen um diese dann flüssig verwenden zu können.

Durch diese Anpassungen ist die Nutzerinteraktion des Tools auf wenige Schritte beschränkt. Der Nutzer kann lediglich seine Formel eingeben, optional eine Heuristik wählen und dann den gewünschten Algorithmus starten. Alle folgenden Schritte bis zur Anzeige des Resultats sind nicht beeinflussbar und daher immer gleich. Unter dem Ergebnis werden die einzelnen Schritte bis zur Lösung angezeigt. Diese Lösung mit weniger Interaktion konzentriert sich auf ein effizientes Angebot der Grundfunktionalität des Tools, ohne unnötige Wahlmöglichkeiten und bietet so besonders bei oftmaliger Nutzung große Zeitersparnis.

Unabhängig von den Änderungen am logischen Ablauf wurde die Art des In- und Outputs ebenfalls abgewandelt. Das Tool liegt als jar-Datei auf dem Server und arbeitet nun ausschließlich mit den Argumenten, die ihm bereits beim Start durch die API (siehe 5.4) übergeben werden. Danach werden keine zusätzlichen Parameter benötigt.

Die Teilschritte und das Endergebnis werden über die Kommandozeile ausgegeben, wo sie von der API gelesen und verarbeitet werden. In Abbildung 5.1 ist eine beispielhafte Anwendung des Java-Programms auf eine Formel zu sehen.

```

• mike@testserver: ~/Desktop
File Edit View Search Terminal Help
mike@testserver:~/Desktop$ java -jar Main.jar "(x1+x2+x3) * (~x1) * (~x2) * (~x3)"
0:Formula clauses as NNF: { 1) {~x1} , 2) {~x2} , 3) {~x3} , 4) {x1,x2,x3} }
0:Resolvents with variable x1: { [1,4]] {x2,x3} }
0:Resolvents with variable x2: { [2,4]] {x1,x3} }
0:Resolvents with variable x3: { [3,4]] {x1,x2} }
0: Non-tautological resolvents: { [3,4]] {x1,x2} , [2,4]] {x1,x3} , [1,4]] {x2,x3} }
0:Forward Subsumption -Deletion of subsumed clauses: { [3,4]] {x1,x2} , [2,4]] {x1,x3} , [1,4]] {x2,x3} }
0:Backward Subsumption - Deletion of subsumed clauses: { [3,4]] {x1,x2} , [2,4]] {x1,x3} , [1,4]] {x2,x3} }
1:Formula clauses as NNF: { 1) {~x1} , 2) {~x2} , 3) {~x3} }
1:Resolvents with variable x1: { [1,[3,4]] {x2} , [1,[2,4]] {x3} }
1:Resolvents with variable x2: { [2,[3,4]] {x1} , [2,[1,4]] {x3} }
1:Resolvents with variable x3: { [3,[2,4]] {x1} , [3,[1,4]] {x2} }
1: Non-tautological resolvents: { [2,[3,4]] {x1} , [3,[2,4]] {x1} , [1,[3,4]] {x2} , [3,[1,4]] {x2} , [1,[2,4]] {x3} }
1:Forward Subsumption -Deletion of subsumed clauses: { [2,[3,4]] {x1} , [1,[3,4]] {x2} , [1,[2,4]] {x3} }
1:Backward Subsumption - Deletion of subsumed clauses: { [2,[3,4]] {x1} , [1,[3,4]] {x2} , [1,[2,4]] {x3} }
2:Formula clauses as NNF: { 1) {~x1} , 2) {~x2} , 3) {~x3} }
2:Resolvents with variable x1: { [1,[2,[3,4]]]{} }
2:Resolvents with variable x2: { [2,[1,[3,4]]]{} }
2:Resolvents with variable x3: { [3,[1,[2,4]]]{} }
2: Non-tautological resolvents: { [1,[2,[3,4]]]{} , [2,[1,[3,4]]]{} , [3,[1,[2,4]]]{} }
2: End Resolution: empty clause in resolvents
End: Result: Formula is unsatisfiable
mike@testserver:~/Desktop$

```

Abbildung 4.2: Anwendung des Java-Programms mit Ausgabe

4.3 Webserver

Um die Webseite anzuzeigen wird sie über einen Webserver ausgegeben. Für dieses Projekt wurde ein Apache Webserver verwendet, es wäre aber problemlos möglich kleinere und ressourcenärmere Webserver zu implementieren. Der Webserver ist bei diesem Projekt von eher geringer Bedeutung, da React dynamische Webseiten erzeugt deren Inhalte nicht vom Server, sondern clientseitig berechnet werden. Der Server liefert lediglich den Programmiercode der Seiten, kann diesen jedoch selbst nicht oder nur teilweise interpretieren.

Aus diesem Grund sind für den Webserver in diesem Fall auch keine der üblichen Optimierungen notwendig. Sogar bei sehr großen Nutzerzahlen wird pro Anfrage nur ein sehr kleiner Aufwand entstehen. Da wie in 5.1 bereits erwähnt auch keine Zustände verwaltet werden müssen, besteht nur ein geringer Kommunikationsaufwand zwischen Nutzern und Server. Sogar bei Benutzung des Tools entsteht keinerlei Rechenaufwand für den Webserver, da die Berechnung der Ergebnisse von der API und dem Java Programm und die Anzeige der Ergebnisse wiederum von den Nutzergeräten erledigt wird. Pro Nutzer wird die Seite also nur einmal ausgeliefert und dann verwendet, bis die Seite neu geladen oder geschlossen wird.

4.4 React Applikation

Die React Applikation wurde mithilfe der-Toolchain Create React App erstellt. Wie bereits in Kapitel 4 ist diese ist für derartige single-page Apps besonders

gut geeignet. Dank Create React App sind alle nötigen Abhängigkeiten und Ordnerstrukturen von Projektbeginn an gegeben.

Bei Aufruf der Seite wird die Eingabemaske angezeigt:

Abbildung 4.3: Eingabemaske

Dank JSX ähnelt der Code in React, beispielsweise für das Eingabefeld der Formel, dabei stark HTML:

```
<label>Please enter your formula</label>
<div className="inputFormula">
  <input type="text" id="formulaInput" autoFocus={true} value={this.state.value} onChange={this.handleChange}
    placeholder="+ for 'OR', * for 'AND' & - for 'NOT' Example: (x1 + x2) * -x3">
</input>
</div>
```

Abbildung 4.4: Eingabefeld der Formel in JSX-Syntax

Wie in Abbildung 5.3 zu erkennen ist, verfügt das Feld über einen Zustand 'this.state'. Dessen Wert wird bei Laden der Seite mit "", also leer, initialisiert. Die Funktion 'onChange', zu sehen in Abbildung 5.4, die diesem Komponenten hier unter anderem übergeben wird sorgt dafür, dass die vom Nutzer eingegebene Formel den neuen Wert von 'this.state.value' bildet, welcher gleichzeitig auch im Eingabefeld angezeigt wird.

```
handleChange(event) {  
  this.setState({value: event.target.value});  
}
```

Abbildung 4.5: handleChange-Funktion

Nachdem der User die Formel, eventuell eine Heuristik und den Algorithmus eingegeben hat, werden diese Angaben als Daten eines HTTP GET-Requests an den Server geschickt. Um diesen Request zu erzeugen wird Axios verwendet, ein schlanker und weit verbreiteter HTTP Client. Dessen Installation und Nutzung ist dank der Modularität von React nur eine Sache von wenigen Zeilen. Nach Installation mittels eines Paketmanagers muss lediglich eine entsprechende Import-Zeile in der Javascript-Datei eingefügt werden. Anschließend kann der HTTP Client sofort mit allen Funktionen genutzt werden.

Als Ziel des GET-Requests wird ein anderer Port als der des Webservers verwendet, um die Daten an die Server API zu schicken. Die Daten werden im JSON-Format verschickt, da dieses von Axios unterstützt wird und die Daten von der API leicht ausgelesen werden können. In Abbildung 5.5 ist die Implementation am Beispiel Resolution dargestellt.

```
handleRes() {  
  const url = 'http://192.168.133.129:6253';  
  axios.get(url, {  
    params: {  
      'Content-Type': 'application/json',  
      type: 'Resolution',  
      formula: this.state.value,  
      heuristik: this.state.heuristik,  
    })  
  })  
}
```

Abbildung 4.6: GET-Request für die Resolution

Nach Berechnung der Ergebnisse erhält React eine Antwort mit diesen als HTML-Tabelle. Mit Hilfe der React-Funktion 'dangerouslySetInnerHTML' React ist in der Lage diesen HTML-Code ohne weiteres in den bestehenden Code einzugliedern. Der abschreckende Name dieser Funktion wurde gewählt, da es generell gefährlich sein kann Nutzerinput ohne weiteres darzustellen.

An dieser Stelle besteht keine solche Gefahr, da dieses Input von Teilen der Projektarchitektur generiert wurde. Sobald die Daten erhalten und mittels 'dangerouslySetInnerHTML' angezeigt werden, entsteht auf der Webseite die erhaltene Tabelle (siehe Abbildung 5.6).

Um das Ergebnis auch kompakt darzustellen, untersucht React mittels String-matching das erhaltene Ergebnis und gibt je nach Fall an ob die Formel erfüllbar ist oder nicht. Dazu wird Reacts Fähigkeit genutzt nicht nur HTML-Code, sondern sogar React-Code in Variablen speichern und direkt in bestehenden Code einbauen zu können. Die Methode, die das Ergebnis untersucht erstellt eine entsprechende React-Komponente und übergibt diesen an die passende Stelle im React-Code.

Um die Navigation für den Nutzer zu erleichtern, springt die Webseite nach dem Starten des Tools direkt an die Stelle, an der das Ergebnis angegeben wird. Dort gibt es durch einen entsprechenden Button die Möglichkeit zur Tabelle mit den Einzelschritten zu springen.

Iteration	Step	Resulting formula
0	Unprocessed formula	'x1+x2'
0	Formula clauses as NNF	{ 1) {x1,x2} }
0	Resolvents with variable x1	{ }

Abbildung 4.7: Ergebnistabelle

4.5 Server API

Die API stellt die Schnittstelle zwischen Nutzereingabe und Java Applikation dar. Ihre Aufgaben sind das Empfangen der eingegebenen Daten, die Umwandlung dieser Daten in Startbefehle für die Applikation und das Senden der Ergebnisse, die von der Applikation ausgegeben werden. Häufige Lösungen für Anforderungen dieser Art beinhalten das Nutzen des Webservers als Proxy, der bestimmte Anfragen an ein Framework weiterleitet. Da aber in diesem Fall die Anzahl der möglichen Fälle sehr gering ist, bot sich eine direktere und schmalere Lösung an.

Die Schnittstelle wurde daher über ein Python-Skript realisiert. Dieses bedient sich des Python „BaseHTTPServer“ Moduls, welches einen sehr kleinen HTTP-Server erzeugt. Der Server ist in der Lage auf einem angegebenen Port auf HTTP-Requests zu warten, diese zu verarbeiten und angepasste Antworten zu senden.

Für dieses Projekt muss lediglich auf den GET-Request den React mittels Axios versendet gewartet werden. Es sind keinerlei andere Anfragen zu erwarten, weshalb Anfragen die nicht in das Muster passen sofort verworfen werden können.

Handelt es sich um den erwarteten Request, werden die beinhalteten Daten aus JSON geparkt und in einen Kommandozeilenbefehl verarbeitet. Dieser Befehl startet die Java-Applikation mit den erhaltenen Parametern.

Wie bereits in 5.1 erwähnt erzeugt die Applikation Ausgaben für die Kommandozeile. Diese werden im Skript aufgefangen und können so als Datenset verwendet werden. Da React in der Lage ist HTML Code zu empfangen und direkt zu verwenden, kann an dieser Stelle ein weiteres Verpacken der Daten in JSON vermieden werden. Stattdessen wird eine Tabelle aufgebaut, in der die Ergebnisse strukturiert und übersichtlich präsentiert werden können. Der HTML-Code wird nicht mit dem Typ 'text/html', wie möglicherweise zu erwarten wäre, versehen. Stattdessen wird als Typ 'text/plain' angegeben. Grund dafür ist, dass so keine Notwendigkeit besteht, eine komplette HTML-Struktur aufzubauen. So ist es möglich nur den Code für die Tabelle zu senden, so dass er von der React Applikation ohne weiteres in die bereits bestehende Seite eingegliedert wird.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum

Unterschrift