BABEŞ-BOLYAI UNIVERSITÄT CLUJ-NAPOCA

FAKULTÄT FÜR MATHEMATIK UND INFORMATIK

INFORMATIK IN DEUTSCHER SPRACHE

BACHELORARBEIT

Umwandlung von Angular zu React

Betreuer

Assistent Doktor Manuela-Andreea Petrescu

Eingereicht von

Lucaci Daniel

2021

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ ÎN LIMBA GERMANĂ

LUCRARE DE LICENŢĂ

Conversie din Angular în React

Conducător ştiinţific

Asistent Doctor Manuela-Andreea Petrescu

Absolvent

Lucaci Daniel

2021

Abstract

As we live in a constantly growing era of web, the need to find better ways of developing interactive web applications is a significant issue and this is where JavaScript frameworks come to the picture. ReactJs and Angular are the two most used JavaScript frameworks, that can be used to build single page applications. Although almost the same can be attained with both of them, they use a slightly different syntax and a different approach regarding data manipulation and state management. To help inexperienced developers gain a better understanding of the two aforementioned frameworks, I propose a web application, which is able to convert Angular projects in their React equivalent. Converting any type of project regardless of its complexity is not an easy task, as Angular is in a permanent development, which means that the new features introduced over time must be adapted and updated for this application. To avoid this problem, the application I present covers only a limited set of standard Angular functionalities.

The algorithm used for converting Angular applications is very similar to the first three phases of a compiler (lexical analysis, parsing and semantic analysis), i.e. the content of the files will be first divided into individual units called *tokens* and then checked according to the rules of a formal grammar. This type of approach has several advantages, the most important being the ability to analyze the content of the files in detail and to extract all possible information for creating the react project. Another advantage is the exception handling mechanism it provides, which helps users to precisely identify where a certain error was found. As there is currently no other applications of this kind, it can be considered a prototype.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

Abkürzung	*Bedeutung*	*Seite*
API	Application Programming Interface	4
CLI	Command Line Interface	**7**
CSS	Cascading Style Sheets	1
DFA	Deterministic Finite Automaton	16
DOM	Document Object Model	3
EBNF	Extended Backus-Naur Form	18
HTML	Hypertext Markup Language	1
IE	Internet Explorer	16
JSX	JavaScript XML	4
MVC	Model-View-Controller	3
PHP	Hypertext Preprocessor	1
SPA	Single Page Application	1
UI	User Interface	2
UML	Unified Modeling Language	21
XML	Extensible Markup Language	4

Einführung

Die traditionellen Webseiten, die in den Anfängen des Internets eingeführt wurden, hatten den einzigen Zweck, statische Seiten zu liefern, die Inhalte wie Bilder, CSS, JavaScript usw. umfassten. Im traditionellen Ansatz, wenn es irgendeine Interaktion mit einer Seite gibt, wie z.B. das Anklicken einer Schaltfläche, wird eine neue Seite geladen und von Grund auf neu generiert, um eine Antwort im Fenster des Benutzers anzuzeigen.

In der modernen Ära der Webtechnologie werden die meisten Webseiten als Single-Page Webanwendung (engl. *Single Page Application*) erzeugt, das heißt eine Webanwendung, die aus einem einzigen HTML-Dokument besteht und deren Inhalte dynamisch nachgeladen werden. Der Unterschied zwischen einer Einzelseiten-Webanwendung und einer SPA besteht darin, dass in SPA alle Komponenten wie Bilder, CSS, Skripte und alle anderen erforderlichen Ressourcen beim ersten Laden der Seite geladen und dann die entsprechenden Inhalte dynamisch je nach der Benutzerinteraktion angezeigt werden. Der Hauptvorteil einer SPA ist, dass sobald der Benutzer auf eine erste Version der Seite zugegriffen hat, benötigt jede nachfolgende Anfrage sehr wenig Zeit, weil nun ein bestimmter Teil der Seite aktualisiert wird, anstatt die ganze Seite neu zu laden.

1.1 Motivation

Eine Single-Page-Anwendung (abg. *SPA*) besteht eigentlich nur aus einer *index.html*-Datei, zusammen mit einem Javascript- und einem CSS-Bundle, somit ist sie im Vergleich zu traditionelleren serverseitig-gerenderten Anwendungen einfacher zu implementieren. Natürlich muss die Anwendung Aufrufe an das Backend machen, um ihre Daten zu erhalten, aber das ist ein separater Server, der bei Bedarf mit einer völlig anderen Technologie gebaut werden kann: wie NodeJS, Java oder PHP.

Ein weiterer Vorteil der Bereitstellung unseres Frontends als Einzeilseiten-Webanwendung ist die Versionierung und das Rollback. Alles, was wir tun müssen, ist, unsere Build-Ausgabe zu versionieren. Damit kann man den Server, der unsere SPA bereitstellt, mit einem Parameter konfigurieren, der angibt, welche Version der Frontend-Anwendung gebaut werden soll.

1.2 Problemstellung

Es gibt eine große Anzahl von *Frameworks* (engl. für Rahmenwerken), die sich ständig weiterentwickeln, um eine Einzelseiten-Webanwendung zu erstellen. ReactJS und AngularJS sind zwei der meistgenutzten. ReactJS ist ein JavaScript-Framework, die für interaktive Web-Anwendungen verwendet wird. Andererseit ist AngularJS ein der ältesten quelloffenen JavaScript-Framework. Obwohl Angular ein Mehrzweck-Framework ein, wird es von vielen Entwicklern nur für kundenseitige Anwendungen benutzt. AngularJS ist aufgrund seiner Einfachheit und seiner Funktionen die erste Wahl für die Frontend-Entwicklung und dennoch gibt es einen beträchtlichen Prozentsatz von Entwicklern, die React verwenden oder lernen möchten.

1.3 Lösung

Da die beiden Frameworks eine leicht unterschiedliche Syntax verwenden, ist die Entwicklung einer Anwendung, die einfache Angular-Projekte in React umwandelt, möglich. Der Hauptzweck einer solchen Anwendung ist, dass sie anderen hilft, ein besseres Verständnis über beide Frameworks zu erlangen. Es ist wichtig zu erwähnen, dass sie nur wenige Angular-Funktionalitäten unterstützt. Die Anwendung ist nur ein Prototyp und kann damit weiter entwickelt werden.

1.4 Aufbau der Bachelorarbeit

Der Rest dieser Arbeit ist wie folgt gegliedert:

Kapitel 2 stellt die drei Hauptthemen vor, die dieser Arbeit zugrunde liegen: ReactJs, AngularJs und Quellcode-Analyse .

Kapitel 3 beschreibt ausführlich, wie der Umwandlungsvorgang funktioniert. Darüber hinaus werden einige Hinweise über die Benutzeroberfläche und die Funktionen der Anwendung gegeben.

Kapitel 4 schließt die Arbeit mit einer Zusammenfassung dieser Arbeit und einigen Vorschläge für zukünftige Entwicklung.

Grundlagen und Definitionen

2.1 Einzelseiten-Webanwendung

Eine Einzelseitige-Webanwendung besteht aus einzelnen Komponenten, die unabhängig voneinander ersetzt oder aktualisiert werden können, ohne dass die gesamte Seite aktualisiert werden muss. Somit muss die gesamte Seite bei jeder Benutzeraktion nicht neu geladen werden, wodurch Bandbreite gespart wird und nicht jedes Mal externe Dateien wie Bilder oder CSS-Dateien usw. geladen werden müssen. Der Zweck dahinter ist, dass die nachfolgenden Seiten im Vergleich zum traditionellen Request-Response-Zyklus sehr schnell geladen werden [1].

2.2 ReactJS

ReactJS oder einfach React ist eine JavaScript-Bibliothek, die zur Entwicklung wiederverwendbarer Komponenten für die Benutzeroberfläche eingesetzt wird. React ermöglicht im Grunde die Entwicklung großer und komplexer webbasierter Anwendungen, die ihre Daten ändern können ohne anschließendes Aktualisieren der Seite. Es wird als View (V) im Model-View-Controller (MVC, engl. für *Modell-Präsentation-Steuerung*) verwendet [2]. React bietet Unterstützung für mobile Geräte mithilfe von *React Native* und rendert auf der Serverseite mit NodeJS. React implementiert einen unidirektionalen Datenfluss und erweist sich daher als viel einfacher als traditionelle Datenbindung.

2.2.1 Komponenten

Komponenten stellen die primäre Einheit in React dar. Sie verwenden Daten (Eigenschaften und Zustände), um Ihre Benutzeroberfläche als Ausgabe zu rendern. Komponenten in React können mit anderen React-Komponenten integrieren werden; sie folgen einem vorhersehbaren Lebenszyklus, können ihren eigenen internen Zustand beibehalten und mit JavaScript arbeiten. Der Rendern-Prozess (Ausgabe und Aktualisierung einer Benutzeroberfläche

auf Basis Ihrer Daten) ist in React vorhersehbar, und die Komponenten können sich mithilfe der APIs von React in diesen Prozess einklinken.

Komponenten entsprechen oft Aspekten der Benutzeroberfläche, wie z.B. Datumswähler, Kopfzeile, Navigationsleiste und andere, aber man kann auch Verantwortung für Dinge wie kundenseitiges Routing (engl. *Verkehrsführung*), Datenformatierung, Styling und andere Aufgaben einer kundenseitigen Anwendung übernehmen [3]. Bild 2.1 zeigt eine einfache React-Komponente.

```jsx
const Hello = ({ name, age }) => {
    return (
        <div className="centered-box">
            <p>Hello, my name is {name}</p>
            <p>I am {age} years old</p>
        </div>
    )
}

export default Hello;
```

Bild 2.1: *Einfache React-Komponente*

2.2.2 JSX

JSX ist eine Erweiterung der JavaScript-Programmiersprache, die es uns ermöglicht, React-Elemente mit einer HTML-ähnlichen Syntax zu definieren [4]. Es wurde zur gleichen Zeit wie React veröffentlicht und bietet eine prägnante Syntax für die Erstellung komplexer DOM-Bäume mit Attributen.

Das Hauptziel von React ist die Verbesserung der Entwicklererfahrung, das heißt: der Code lesbarer zu machen, dank einer XML-ähnlichen Syntax, die verschachtelte deklarative Strukturen besser darstellen kann. Es ist wichtig zu erwähnen, dass JSX für React nicht erforderlich ist, trotzdem wird es von meisten React-Entwicklern empfohlen [5].

2.2.3 Redux

Redux ist eine eigenständige Bibliothek, die am häufigsten als Zustandsverwaltungsschicht für React verwendet. Ihr Hauptziel besteht darin, Konsistenz und Vorhersagbarkeit für die Daten in Anwendungen zu schaffen [6]. Redux unterteilt die Zustandsverwaltung in zwei separate Einheiten:

- Der Store (engl. für *Speicher*) enthält den gesamten Anwendungsstatus in einem einzigen Objekt, das üblicherweise als Zustandsbaum bezeichnet wird.
- Funktionen, die als *Reducers* (engl. für *Reduzierer*) bekannt sind, legen fest, wie der Store aktualisiert werden soll. Reducers sind Funktionen, die den aktuellen Zustand und einen optionalen *Action*-Objekt als Parameter annehmen und dann den nächsten Zustand zurückgibt, nachdem sie alle Aktualisierungen vorgenommen haben.

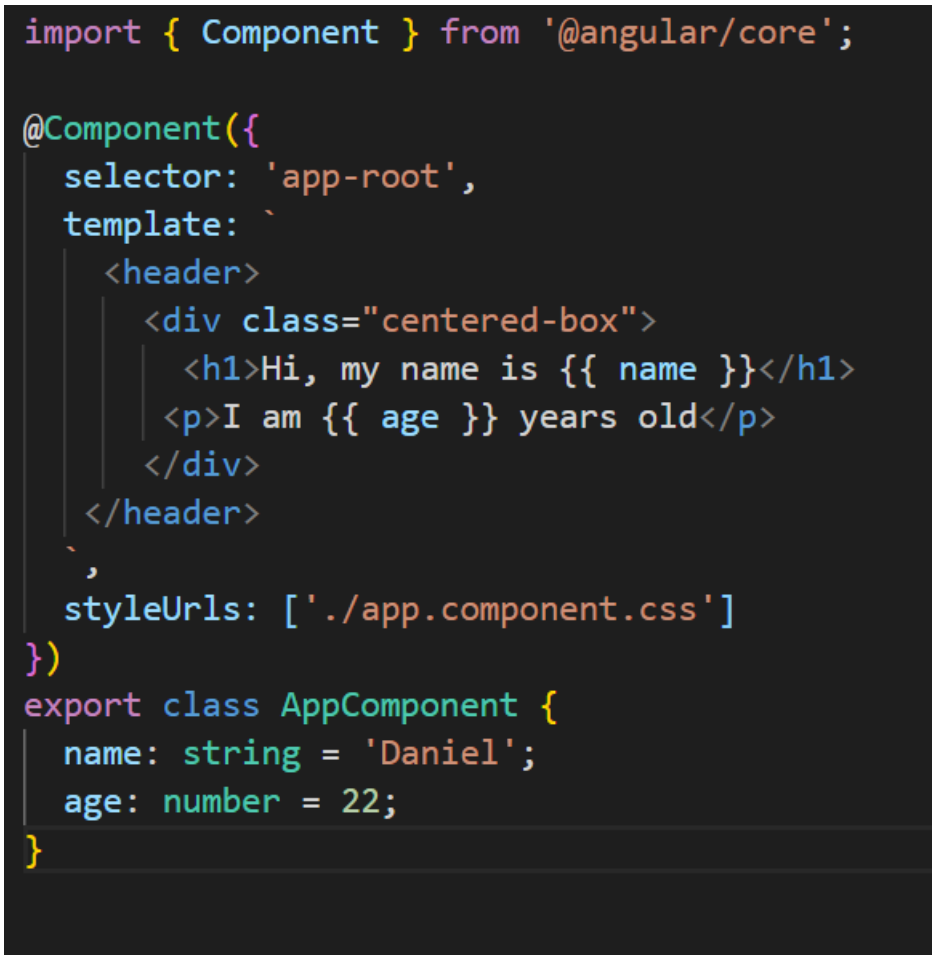
Redux wurde ursprünglich für React entworfen und entwickelt, trotzdem sind die beiden Bibliotheken vollständig entkoppelt [7]. Das Redux-Paradigma kann für die Zustandsverwaltung für die meisten JavaScript-Frameworks, darunter Angular, Backbone, Ember und viele weitere, implementiert werden.

2.3 AngularJS

AngularJS ist ein quelloffenes JavaScript-Framework, das von Google erstellt wurde und Entwicklern bei der Erstellung von Einzelseiten-Webanwendungen hilft. Sein Zweck ist es, die Entwicklung von Webanwendungen mit der Model-View-Controller-Fähigkeit zu unterstützen, um die Entwicklung, Wartung und Prüfung zu erleichtern [8]. AngularJS bringt die MVC-Fähigkeit in die Webanwendung und macht sie dadurch modularer und einfacher zu entwickeln, zu warten und zu testen. AngularJS setzt Direktive ein, die mit dem Präfix 'ng-' vorgesehen und darauf abzielen, die Daten über den Controller an die View oder Templates zu binden [9]. AngularJS-Controller sind in TypeScript geschrieben und fügen die Geschäftslogik zu Views hinzu, die nichts anderes als HTML-Seiten sind.

Angular bietet einige wesentliche Vorteile an, von denen die wichtigsten die folgenden sind:

- **Benutzerdefinierte Komponenten -** Mit Angular kann man eigene Komponenten erstellen, die Funktionalität zusammen mit der Rendering-Logik in wiederverwendbare Teile verpacken. Im Bild 2.2 wird eine einfache Angular-Komponente dargestellt.
- **Datenbindung** – Angular erleichtert die Übertragung der Daten, bzw. Funktionen vom JavaScript-Kerncode zum View.
- **Injektion von Abhängigkeiten (engl. *dependency injection*)** - Mit Angular kann man modulare Dienste (engl. *services*) schreiben und sie überall einsetzen.



```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <header>
      <div class="centered-box">
        <h1>Hi, my name is {{ name }}</h1>
        <p>I am {{ age }} years old</p>
      </div>
    </header>
  `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  name: string = 'Daniel';
  age: number = 22;
}
```

Bild 2.2: *Angular-Komponente*

2.3.1 Angular CLI

Das Angular CLI (*command line interface*, engl. für Befehlszeilenschnittstelle) wird für die Entwicklung von Angular Anwendungen verwendet und kann als Standard-Werkzeug für die Erstellung von Angular-Anwendungen betrachtet werden. Es baut auf NodeJS auf und enthält Skripte zum Erstellen eines Angular-Projekts, zum Hinzufügen von Komponenten und zum Ausführen von Unit-Tests. In der Abbildung 2.3 wird ein Kommando zum Erstellen eines neuen Angular-Projekts angezeigt.

Das CLI übernimmt sogar komplizierte Aufgaben, wie das Kompilieren von TypeScript zu JavaScript, das Erstellen produktionsreifer Builds und die Verwendung anderer Komponenten [10]. Es enthält sogar einen eingebauten Webserver der den Browser automatisch aktualisiert, wenn der Code geändert wird.

```
C:\Projects\Web Development\Angular>ng new basic-app --no-strict
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS
CREATE basic-app/angular.json (2969 bytes)
CREATE basic-app/package.json (1073 bytes)
CREATE basic-app/README.md (1054 bytes)
CREATE basic-app/tsconfig.json (538 bytes)
CREATE basic-app/.editorconfig (274 bytes)
CREATE basic-app/.gitignore (620 bytes)
CREATE basic-app/.browserslistrc (600 bytes)
CREATE basic-app/karma.conf.js (1426 bytes)
CREATE basic-app/tsconfig.app.json (287 bytes)
CREATE basic-app/tsconfig.spec.json (333 bytes)
CREATE basic-app/.vscode/extensions.json (130 bytes)
CREATE basic-app/.vscode/launch.json (474 bytes)
CREATE basic-app/.vscode/tasks.json (938 bytes)
CREATE basic-app/src/favicon.ico (948 bytes)
CREATE basic-app/src/index.html (294 bytes)
CREATE basic-app/src/main.ts (372 bytes)
CREATE basic-app/src/polyfills.ts (2338 bytes)
CREATE basic-app/src/styles.css (80 bytes)
CREATE basic-app/src/test.ts (745 bytes)
CREATE basic-app/src/assets/.gitkeep (0 bytes)
CREATE basic-app/src/environments/environment.prod.ts (51 bytes)
CREATE basic-app/src/environments/environment.ts (658 bytes)
CREATE basic-app/src/app/app.module.ts (314 bytes)
CREATE basic-app/src/app/app.component.html (23332 bytes)
CREATE basic-app/src/app/app.component.spec.ts (965 bytes)
CREATE basic-app/src/app/app.component.ts (213 bytes)
CREATE basic-app/src/app/app.component.css (0 bytes)
- Installing packages (npm)...
```

Bild 2.3: *Erstellen eines neuen Projekts mithilfe der Angular-CLI*

2.3.2 TypeScript

TypeScript ist eine Programmiersprache, die von Microsoft entwickelt wurde und weiterhin unterstützt wird. TypeScript ist syntaktisch eine Verbesserung des EcmaScripts 5, somit kann jedes JavaScript-Programm auch als ein TypeScript-Programm betrachtet werden.

TypeScript enthält zusätzlich ein Modulsystem, Schnittstellen und ein statisches Typsystem. Das Modul- und Typsystem sind flexibel und zielen darauf ab, die Typfehler statisch zu erkennen und die JavaScript-Entwicklung effizienter zu machen (z. B. durch Vorschläge, welche Methoden für ein Objekt aufgerufen werden können) [11].

2.4 Vergleich zwischen AngularJS und ReactJS

Angular und React sind beide UI-Frameworks. Obwohl sie das Gleiche erreichen können, es gibt ein paar Unterscheidungen, wie in der Tabelle 2.1 erwiesen wird.

Eigenschaft	AngularJS	ReactJS
DOM	Reguläres DOM	Virtuelles DOM
Packaging (engl. für Verpackung)	Schwach	Stark
Abstraktion	Schwach	Stark
Ausnahmebehandlung	Während Laufzeit	Während Übersetzungszeit
Datenbindung	Bidirektional	Unidirektional
Templating (engl. für Vorlagenerstellung)	In HTML	In JSX-Dateien
Unterstützung für mobile Geräte	Ionic Framework	React Native

Tabelle 2.1: *Vergleich zwischen Angular und React [12]*

2.5 Quellcode-Analyse

Die Quellcode-Analyse ist der Prozess der Extraktion von Informationen über ein Programm aus seinem Quellcode, die mit automatischen Werkzeugen aus dem Quellcode erzeugt wurden. [13] definiert die Quellcode als jede statische, textuelle, menschenlesbare, vollständig ausführbare Beschreibung eines Computerprogramms, das automatisch in eine ausführbare Form kompiliert werden kann.

2.5.1 Reguläre Ausdrücke und endliche Automaten

Viele technische Sprachen werden mithilfe der regulären Ausdrücke und Grammatike spezifiziert, aber der Entwurf und die Implementierung eines Compilers benötigen eine Möglichkeit, die Algorithmen zu beschreiben, die eine Zeichenkette untersuchen und entscheiden, ob es sich um einen gültigen Satz der Sprache handelt [13].

Häufig muss festgestellt werden, ob ein Text für eine bestimmte Sprache geeignet ist, vor allem als Vorstufe zur Textverarbeitung oder Übersetzung. Ein Dokumentenprozessor prüft die Rechtschreibung von Wörtern, bevor er die Syntax überprüft; ein Compiler untersucht ein Quellprogramm, um sicherzustellen, dass es richtig ist, und eine grafische Benutzeroberfläche muss sicherstellen, dass die Daten korrekt übermittelt werden. Eine solche Kontrolle wird durch ein Erkennungsverfahren durchgeführt, das sich mit Hilfe von minimalistischen Modellen, den so genannten abstrakte Maschinen oder Automaten [14]. Die Vorteile liegen darin, dass die Automaten nicht von den Programmiersprachen und -techniken, d. h. von der Implementierung, abhängen und dass ihre Eigenschaften (z. B. ihre Zeit- und Speicherkomplexität) deutlicher mit der eindeutig mit der Familie der Ausgangssprache verbunden sind.

2.5.2 Lexikalische Analyse

Die lexikalische Analyse (auch *Scanner* gennant) ist die erste Phase der Umwandlung, die den Inhalt der Eingabedateien mithilfe von einem deterministischen Automaten aufteilt, wie definiert in [15]. Als Ausgabe gilt eine Sequenz von Symbole (engl. *tokens*), die weiterhin zur Syntaxanalyse geschickt wird.

Unter Tokens versteht man ein Paar, das aus einem Token-Namen und einem optionalen Attribut Wert besteht [16]. Der Token-Name ist ein abstraktes Symbol, das eine Art lexikalischer Einheit, z. B. ein bestimmtes Schlüsselwort oder eine Folge von Eingabezeichen die einen Bezeichner bezeichnen, darstellt. Die Token-Namen sind die Eingabesymbole, die der Parser verarbeitet. Übliche Token-Namen sind: Operatoren, Separatoren, Variablennamen (Identifikatoren) und Schlüsselwörter. Kommentare, Leerzeichen, Tabulatoren und Zeilumbrüche können nicht als bedeteungsvolle Zeichen betrachtet werden und darum werden sie automatisch ignoriert. Andererseits stellt ein Lexem eine Folge von Zeichen im Quellprogramm dar, die dem Muster eines Token entspricht und vom lexikalischen Analysator als eine Instanz dieses Tokens identifiziert [16].

2.5.3 Parsing

In [17] wird die grammatische Analyse (auch *Parsing* genannt) als der Prozess der Analyse einer Folge von Symbolen, die den Regeln einer formalen Grammatik entsprechen, definiert. Eine formale Grammatik ist definiert als ein Satz von Produktionsregeln für solche Symbolen, die den Syntax einer Programmiersprache beschreiben.

Eine formale Grammatik besteht aus einer Menge von Terminalsymbolen und nichtterminalen Produktionsregeln. Terminalsymbole sind Zeichenkette, die in den Ausgaben der Produktionsregeln einer formalen Grammatik erscheinen können und die nicht durch die Regeln der Grammatik verändert werden können [18]. Auf der anderen Seite, nichtterminale Symbole (auch syntaktische Variablen genannt) sind Variablen, die gemäß den Produktionsregeln durch Gruppen von Terminalsymbolen ersetzt werden können.

2.5.4 Semantische Analyse

Die semantische Analyse (auch kontextsensitive Analyse genannt) ist der Prozess nach dem Parsen , der notwendige semantische Informationen aus dem Quellcode extrahiert [19]. Sie umfasst in der Regel eine Typüberprüfung oder stellt sicher, dass eine Variable vor der Verwendung deklariert wird, die nicht in der erweiterten Backus-Naur-Form beschrieben werden kann und daher beim Parsen nicht leicht zu erkennen ist.

Das Ziel der semantischen Analyse ist das Erzeugen einer Zwischendarstellung, meist häufig ein abstrakter Syntaxbaum oder eine Symboltabelle. Die gebräuchlichsten Arten von Attributen, die wir uns für jedes Symbol im Syntaxbaum merken sollten, sind:

- *Typ* - Verknüpft das Datenobjekt mit der zulässigen Menge von Werten.
- *Zugriffsmodifikator* – verwaltet den Zugang auf eine bestimmte Variable
- *Wert* - normalerweise das Ergebnis einer Zuweisungsoperation.
- *Name* - eindeutiger Bezeichner für eine Variable oder Funktion innerhalb eines Bereichs.
- *Komponente* - Komponenten können aus beliebig vielen Datenobjekten zusammengesetzt sein

Umwandlung von Angular zu React

3.1 Übersicht

Der Prozess der Umwandlung einer Angular-Anwendung in React ist sehr ähnlich mit dem Prozess der Kompilierung bei höheren Programmiersprachen, dass heißt es werden dieseselbe Schritte verwendet: lexicalishe Analyse, Parsing und semantische Analyse.

3.1.1 Benutzeroberfläche und Benutzererfahrung

Die Anwendung ist eine einfache React-Webseite mit dem Namen *Angular2React*, die aus fünf Sektionen besteht:

- *Welcome* – wird direkt nach dem Zugriff auf die Website angezeigt. Bild 3.1 zeigt die Startseite auf einem mobilen Gerät.
- *About* – enthält ausführlichen Informationen und Anweisungen, sowie eine Liste mit häufig gestellten Frage.
- *Upload* – besteht aus einem Behälter in dem man das Archiv mit dem umzuwandelnden Projekt hochladen kann.
- *Converter* – zeigt allgemeine Informationen in Echtzeit über den aktuellen Umwandlungsprozess sowie einen Fortschrittsbalken und eine Schaltfläche zum Abbrechen an.
- *Contact* – zeigt ein Kontaktformular mit Eingabefelder für Namen, Email-Adresse und Nachricht an, mit dem man Kontakt mit dem Ersteller der Anwendung aufnehmen kann. Da die Anwendung über keine Backend-Logik verfügt, werden die Kontaktinformationen nach dem Ausfüllen und Absenden des Formulars in einer Online-Datenbank gespeichert.

Für die Umwandlung einer Angular-Anwendung müssen folgende Schritte beachtet werden:

- Der Benutzer speichert seine Anwendung in einem Archiv, das unbedingt die “src“- Ordner beinhalten muss, ansonsten wird eine Fehlermeldung angezeigt.
- Das Archiv kann per Drag and Drop abgelegt oder mit Hilfe vom Upload-Knopf hochgeladen werden. Der Benutzer hat die Möglichkeit, das zuvor hochgeladene Archiv zu beseitigen und ein anderes hochzuladen.

- Der Umwandlungsprozess beginnt erst dann, wenn man auf der Schaltfläche "Convert" drückt. In diesem Fall wird der Benutzer automatisch auf die Umwandlungsseite weitergeleitet.
- Steckt der Prozess fest oder möchte der Benutzer die Umwandlung abbrechen, so kann er jederzeit auf die Schaltfläche "Cancel" klicken, wodurch die Umwandlung abgebrochen und der Benutzer zur Seite "Upload" zurückgeleitet wird.
- Falls ein Fehler während der Umwandlung gefunden wird, dann wird der Prozess automatisch beendet und die Fehlermeldung wird auf dem Bildschirm angezeigt.
- Am Ende wird ein Archiv mit dem React-Projekt heruntergeladen und der Benutzer wird automatisch auf die “Upload“ – Seite zurückgeleitet. Es ist wichtig zu erwähnen, dass das Archiv nur die Komponenten mit den entsprechenden CSS-Dateien beinhaltet, so müssen alle andere Dateien, wie z. B. Bilder, manuell hinzugefügt werden.

Bild 3.1: *Startseite der Anwendung auf einem mobilen Gerät*

3.1.2 Ausnahmenbehandlung

Obwohl es empfohlen wird, die Anwendung vor dem Hochladen zu testen, besteht die Möglichkeit, dass Fehler auftreten können. Diese können wie folgt unterteilt werden:

- Lexikalische Fehler – wenn bei der lexikalischen Analyse ein unbekanntes Symbol gefunden wird
- Grammatische Fehler – werden auch als Syntaxfehler genannt.
- Semantische Fehler - bezieht sich auf die falsche Verwendung von Variablen und Funktionsaufrufen
- Dateienfehler – diese sind die häufigsten und beruhen sich auf fehlende, bzw. fehlplatzierte Dateien

Wenn ein Fehler während der Umwandlung entdeckt wird, wird diese automatisch abgebrochen und eine Nachricht mit der Beschreibung der Ursache auf dem Bildschirm angezeigt. Bild 3.2 zeigt eine Fehlermeldung, die von einer fehlenden Datei verursacht wurde.

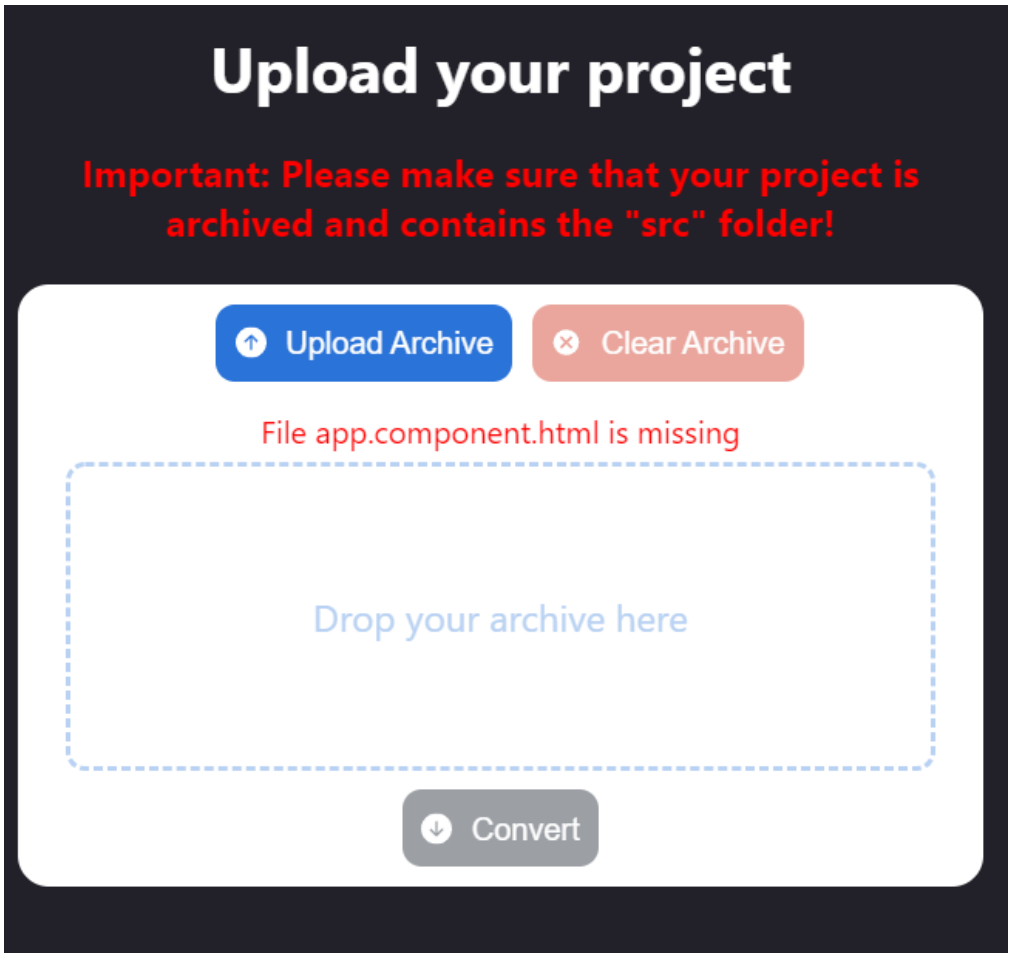


Bild 3.2: *Fehlermeldung wegen einer fehlenden Datei*

3.2 Arbeitsprinzip

3.2.1 Umwandlungsprozess

Der Prozess der Umwandlung eines Projekts von Angular zu React läuft folgendermaßen ab: zunächst wird das Vorhandensein der wesentlichsten Dateien (*app.module.ts, app.component.ts, app.component.html*) überprüft. Wenn diese nicht vorhanden sind, dann kann die Umwandlung nicht stattfinden und eine Fehlermeldung wird auf dem Bildschirm angezeigt.

- Falls diese vorhanden sind, wird erstmals der Inhalt der in der *app.module.ts*-Datei befindenden *AppModule*-Klasse analysiert und daraus wichtige Informationen über alle Komponenten extrahiert.
- Alle Komponenten werden der Reihe nach von oben nach unten (von der höchsten zur niedrigsten Ebene) durchgelaufen
- Für jede Komponente wird zuerst der Inhalt der entsprechenden Klasse analysiert und dann der Inhalt der *HTML*-Datei. Falls *CSS*-Dateien vorhanden sind, werden sie als solche kopiert, ohne ihren Inhalt zu überprüfen. Anhand der extrahierten Informationen wird den entsprechenden React-Komponenten erzeugt und im Archiv hinzugefügt werden.
- Falls ein Fehler gefunden wird, dann wird der Vorgang automatisch beendet und eine Fehlermeldung mit der Beschreibung des Fehlers angezeigt.
- Am Ende wird ein Archiv namens „*converted.zip*“ mit dem erzeugten React-Projekt automatisch heruntergeladen.
- Zusätzliche Dateien, wie z.B. Bilder, müssen vom Benutzer manuell zum Projekt hinzugefügt werden. Falls nötig, müssen weitere Anpassung bei der *index.html* – Datei vorgenommen werden.

Der Diagramm im untenen Bild schildert eine Kurzfassung des zuvor vorgestellten Umwandlungvorgangs.

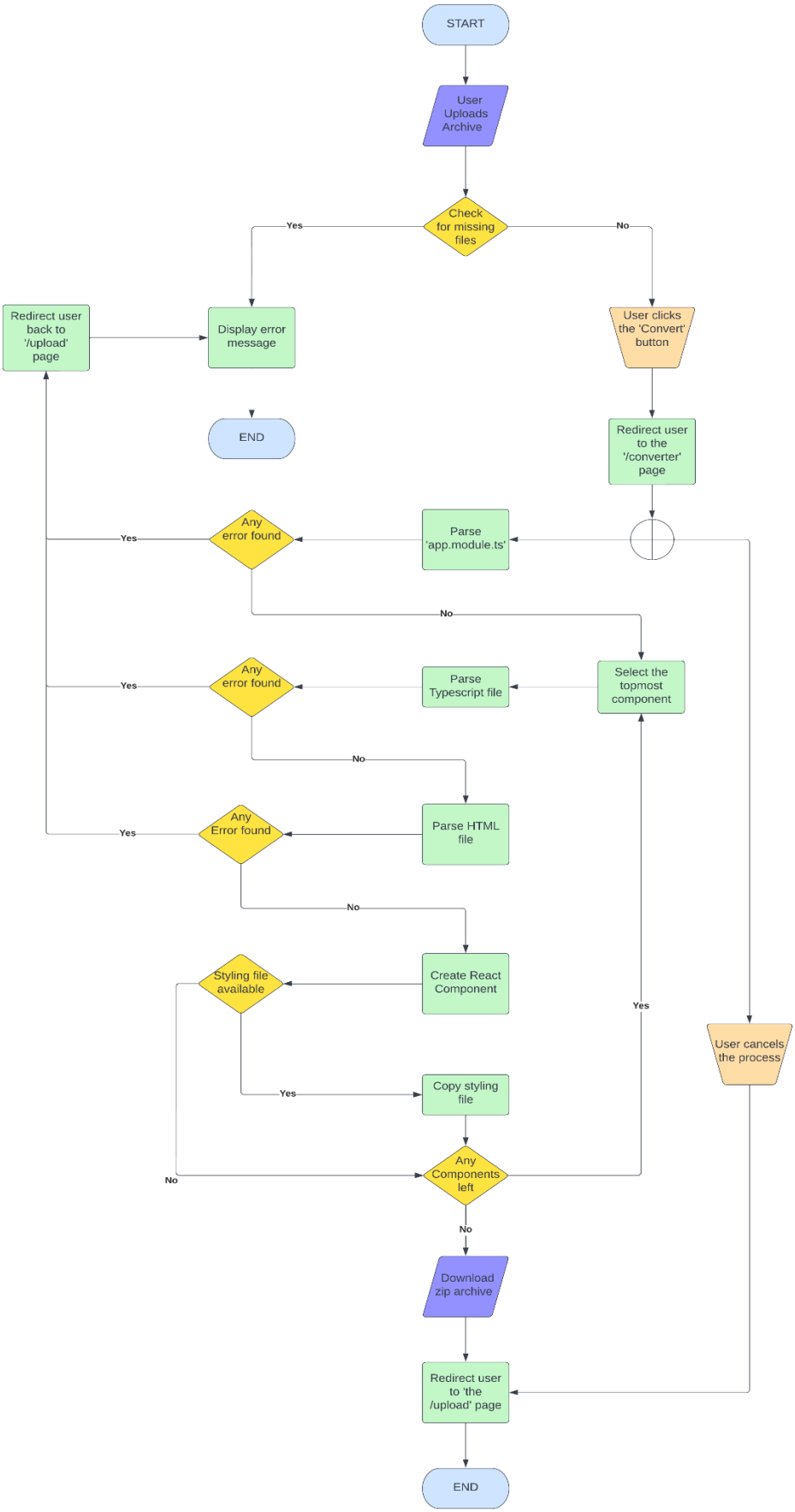


Bild 3.3: *Flussdiagramm für den Umwandlungsvorgang*

3.2.2 Bearbeitung von Archivinhalten

JSZip ist eine Javascript-Drittanbieterbibliothek zum Erstellen, Lesen und Bearbeiten von *.zip*-Dateien, die auch für dieses Projekt verwendet wurde. Es unterstützt viele Webbrowser, darunter Opera, Safari und Firefox. Abbildung 3.4 zeigt eine Funktion für die Erstellung eines Archivs.

Obwohl JSZip einfach und intuitiv ist, gibt es auch ein paar Einschränkungen, die beachtet werden müssen. Zuallererst funktioniert JSZip ausschließlich mit *ZIP*-Archiven, daher können andere Formate wie *.rar, .7z, .tar* und so weiter nicht verwendet werden. Außerdem werden nicht alle Funktionen von Zip-Dateien unterstützt. Klassische Zip-Dateien funktionieren, aber verschlüsselte Zip-Dateien und mehrbändige Dateien werden nicht unterstützt.

Eine weitere Einschränkung ergibt sich aus dem Browser (und dem Rechner, auf dem der Browser läuft). Eine komprimierte Zip-Datei von 10 MB wird problemlos von Firefox/Chrome/ Opera/IE10+ geöffnet, bringt aber ältere IE-Versionen zum Absturz.

```
createArchive() {
    this.root.generateAsync({type: 'blob'}).then(function(blob) {
        saveAs(blob, 'converted-app');
    })
}
```

Bild 3.4: *Funktion zum Erstellen eines Archivs*

3.2.3 Tokenisierung

Die Tokenisierung der Inhalten der Dateien findet mithilfe einer deterministischen endlichen Automaten (abg. *DFA*) statt, der eine gegebene Zeichenfolge akzeptieren oder ablehnen kann, indem er eine durch die Zeichenfolge eindeutig bestimmte Zustandsfolge durchläuft. Einer der Vorteile dieses Automaten ist, dass er einfach zu implementieren und zu benutzen ist. Ein weiterer Vorteil dafür ist die Möglichkeit, mit Genauigkeit festzustellen, in welcher Zeilen- und Spaltennummer ein Fehler gefunden wurde. Im Bild wird ein deterministischer endlicher Automat für Javascript-Operatoren dargestellt.

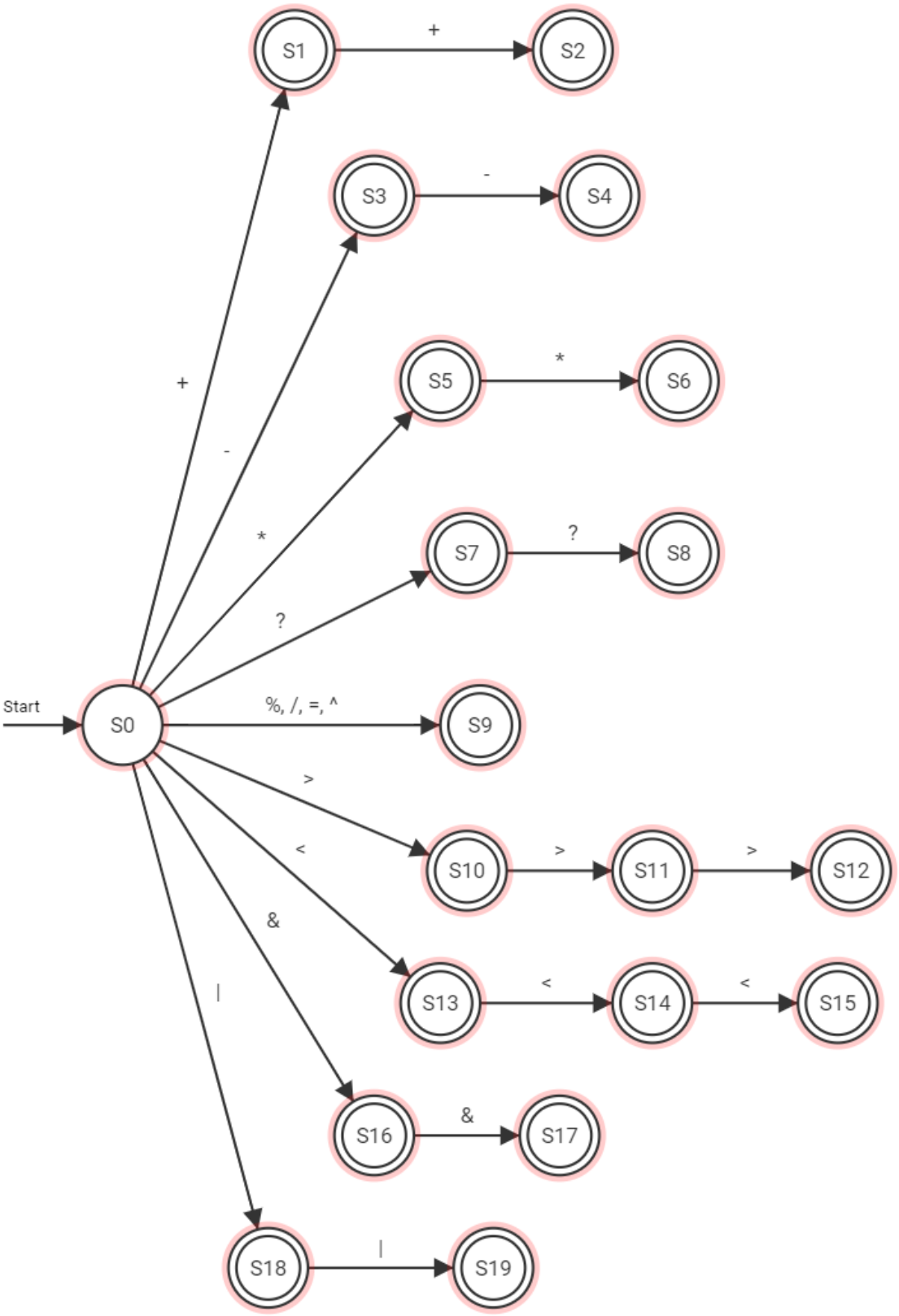


Bild 3.5: *Deterministischer endlicher Automat für Javascript-Operatoren*

3.2.4 Parsing

Für dieses Projekt wurde Extended Backus – Naur – Form (abg. *EBNF*) als formale Grammatik gewählt. Die Produktionsregeln wurden für AppModule, sowie für HTML- und Typescript-Dateien definiert. Für die CSS-Dateien ist die lexikalische und syntaktische Analyse nicht nötig, da diese als solche kopiert werden. Im Bild 3.6 werden die EBNF-Produktionsregel für eine HTML-Datei aufgezählt. Die Terminalsymbole werden mit grüner Farbe und die Nichtterminalsymbole mit rot hervorgehoben.

```
LETTER = 'a' | 'b' | 'c' | 'd' | ... | 'z' | 'A' | 'B' | ... | 'Z';
NON_ZERO_DIGIT = '1' | '2' | '3' | '4' | ... | '9';
DIGIT = NON_ZERO_DIGIT | '0';
BOOLEAN_VALUE = 'true' | 'false';
INTEGER_VALUE = ['-'] NON_ZERO_DIGIT {DIGIT};
FLOAT_VALUE = INTEGER_VALUE '.' {DIGIT};
NUMBER = INTEGER_VALUE | FLOAT_VALUE;
TAG_OPEN = '<';
TAG_CLOSE = '>';
TAG_SLASH_CLOSE = '/';
HTML_TAG_NAME = 'p' | 'div' | 'h1' | 'h2' | ... | 'h6' | 'span' | 'img' | 'em' | 'del' | 'i' | ....;
HTML_ATTRIBUTE_NAME = 'class' | 'id' | ....;
HTML_COMMENT_OPEN = '<' '!' '-' '-' ;
HTML_COMMENT_CLOSED = '-' '-' '>';
HTML_DOCUMENT = HTML_ELEMENT;
HTML_ELEMENT = [HTML_COMMENT_OPEN] TAG_OPEN HTML_TAG_NAME {HTML_ATTRIBUTE}
TAG_CLOSE CONTENT TAG_OPEN TAG_SLASH HTML_TAG_NAME TAG_CLOSE
| TAG_OPEN HTML_TAG_NAME {HTML_ATTRIBUTE} TAG_SLASH_CLOSE TAG_CLOSE
| TAG_OPEN HTML_TAG_NAME {HTML_ATTRIBUTE} TAG_CLOSE [HTML_COMMENT_CLOSED];
HTML_CONTENT = {HTML_ELEMENT | TEXT};
HTML_ATTRIBUTE = HTML_ATTRIBUTE_NAME '=' HTML_ATTRIBUTE_VALUE | HTML_ATTRIBUTE_NAME;
HTML_ATTRIBUTE_VALUE = STRING | BOOLEAN_VALUE | NUMBER;
```

Bild 3.6*: Produktionsregel für eine einfache HTML-Datei*

3.3 Implementierung

3.3.1 Komponentenhierarchie

Die Benutzeroberfläche der Anwendung wurde vollständig mit dem React-Rahmenwerk erstellt. Die folgende Abbildung zeigt eine grafische Darstellung der Zerlegung der Anwendung in Komponenten. Neben diesen Komponenten gibt es drei weitere Hilfskomponenten (*Background*, *Box* und *LoadingSpinner*), die nur für bestimmte Zwecke verwendet werden.

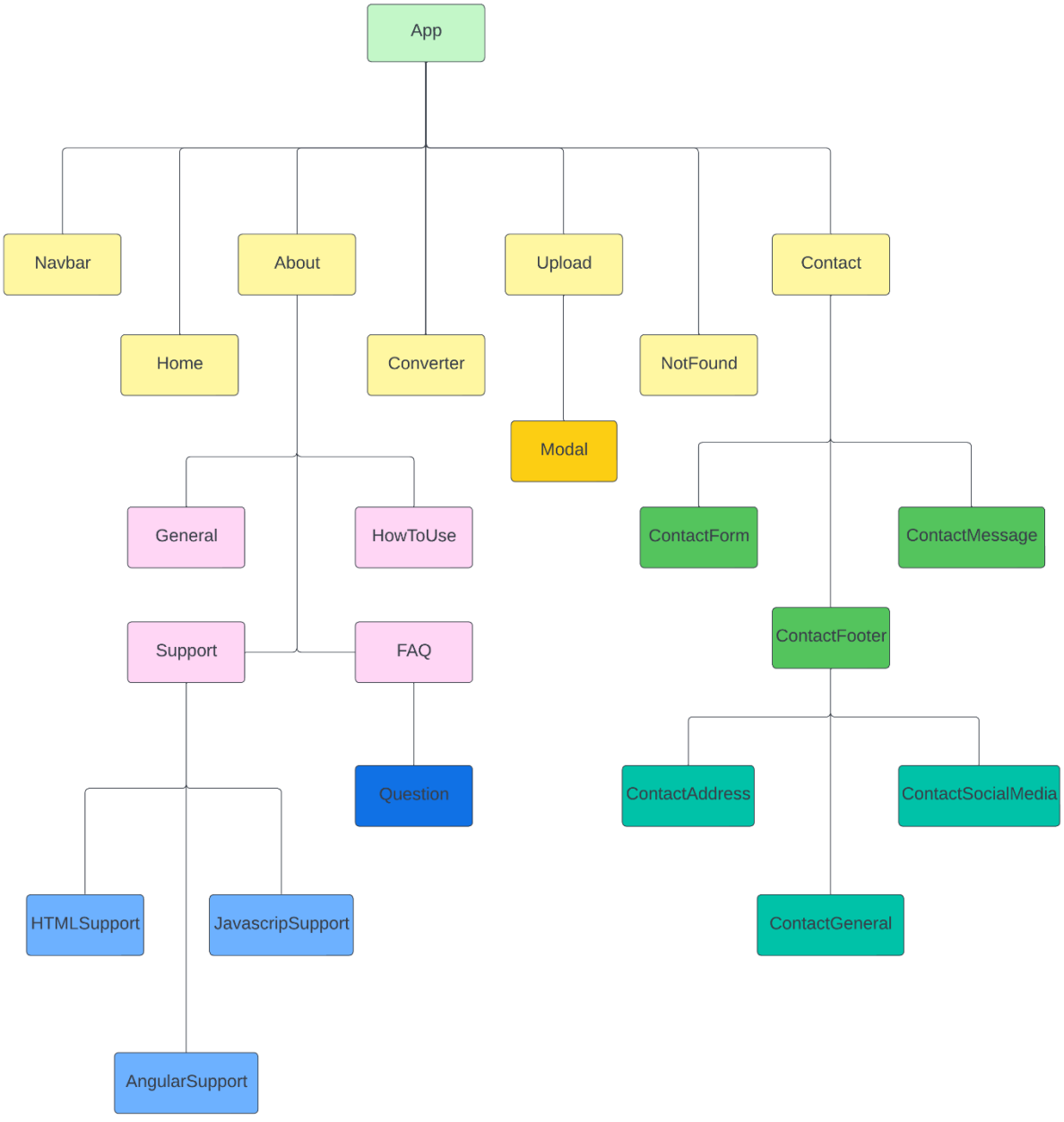


Bild 3.7: *Graphische Darstellung des Komponentenbaums*

3.3.2 Redux Store

Das Projekt basiert auf Redux als Werkzeug zur Zustandsverwaltung und verwendet ein einziges *Slice* (engl. für *Scheibe*) für den Umwandlungsvorgang. Abbildung 3.8 zeigt das zuvor genannte *Slice* zusammen mit dem Anfangszustand und den *Reducers*.

Einer der Hauptvorteile der Verwendung von Redux gegenüber der eingebauten Context-API ist, dass der Redux-Store nicht nur innerhalb der Komponenten, sondern auch in der zusätzlichen Javascript-Dateien zugegriffen werden kann.

Bild 3.8: *Conversion-Slice*

3.3.3 Klassenhierarchie

Die Umsetzung des Umwandlungsvorgangs wurde vollständig in Javascript geschrieben und gründet sich auf drei Hauptklassen, sowie auf vier Paketen. Ein UML-Diagramm mit der Strukturierung des Projekts ist in der Abbildung 3.9 zu sehen. Die Hauptklassen sind:

- *Project* – stellt die wichtigste Klasse dar und verfügt über alle Hauptfunktionen
- *Archive* – kümmert sich mit dem vom Benutzer hochgeladenen Archiv
- *Folders* – wird zum Erstellen des Archivs mit dem umgewandelten React-Projekt eingesetzt.

Die Paketen werden in den folgenden Kapiteln ausführlich beschrieben.

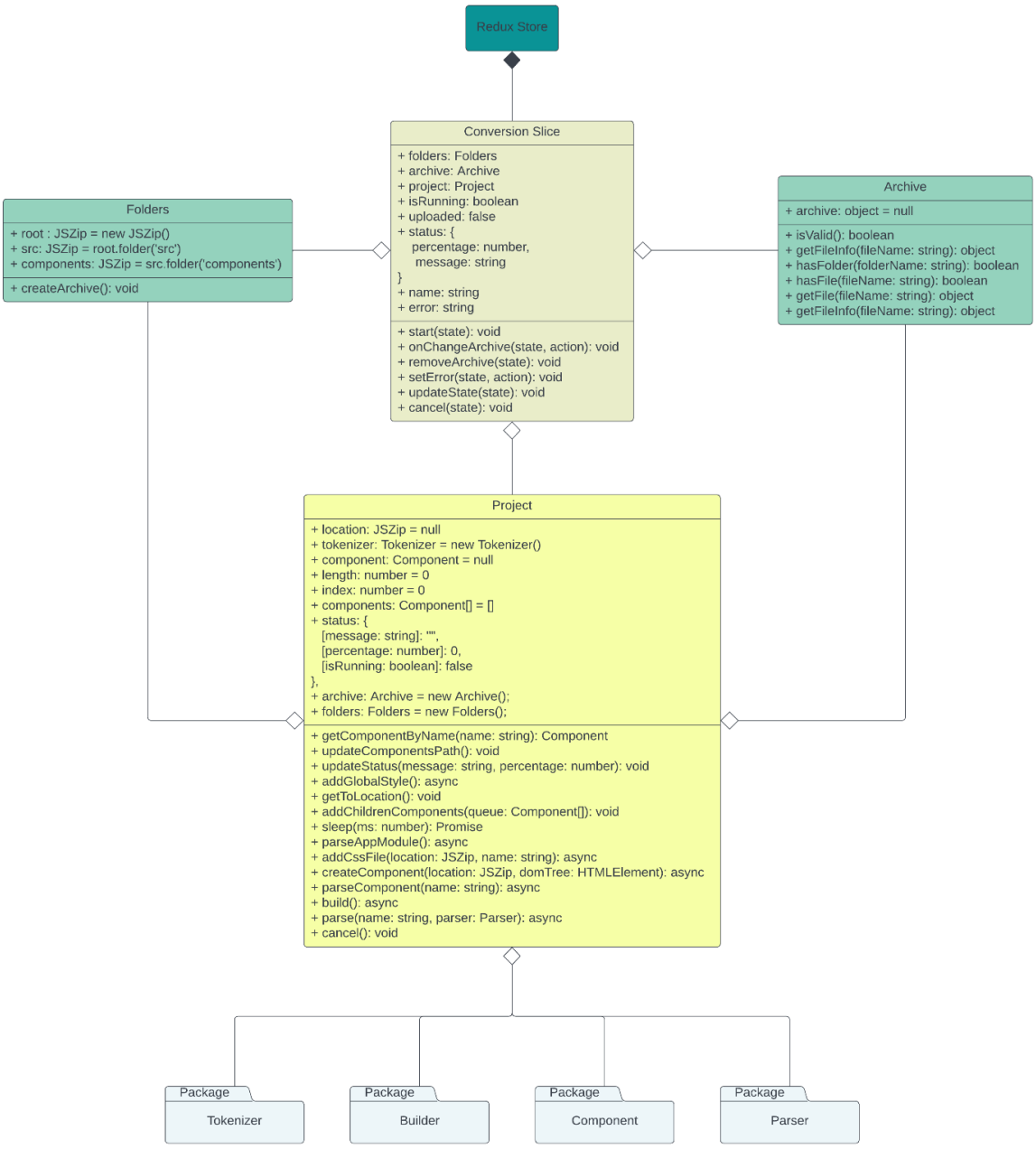


Bild 3.9: *UML-Diagramm für das ganze Projekt*

3.3.4 Tokenizer-Paket

Das Tokenizer-Paket entspricht der lexikalischen Analyse und hat den einzigen Zweck, die Inhalten der Dateien durchzulaufen und diese in Tokens zu zerlegen. Das Paket besteht aus den folgenden vier Klassen:

- *Tokenizer* - benutzt ein DFA-Objekt um Zeichenkette in Tokens aufzuteilen und diese in einer Liste zu speichern
- *DFA* – Setzt einen deterministischen endlichen Automat um
- *Token* – speicher allgemeine Informationen über Tokens
- *State* – enthält eine Reihe von möglichen Zuständen, die der Automat erreichen kann, und einen zusätzlichen Zustand, um zu markieren, dass ein Fehler gefunden wurde.

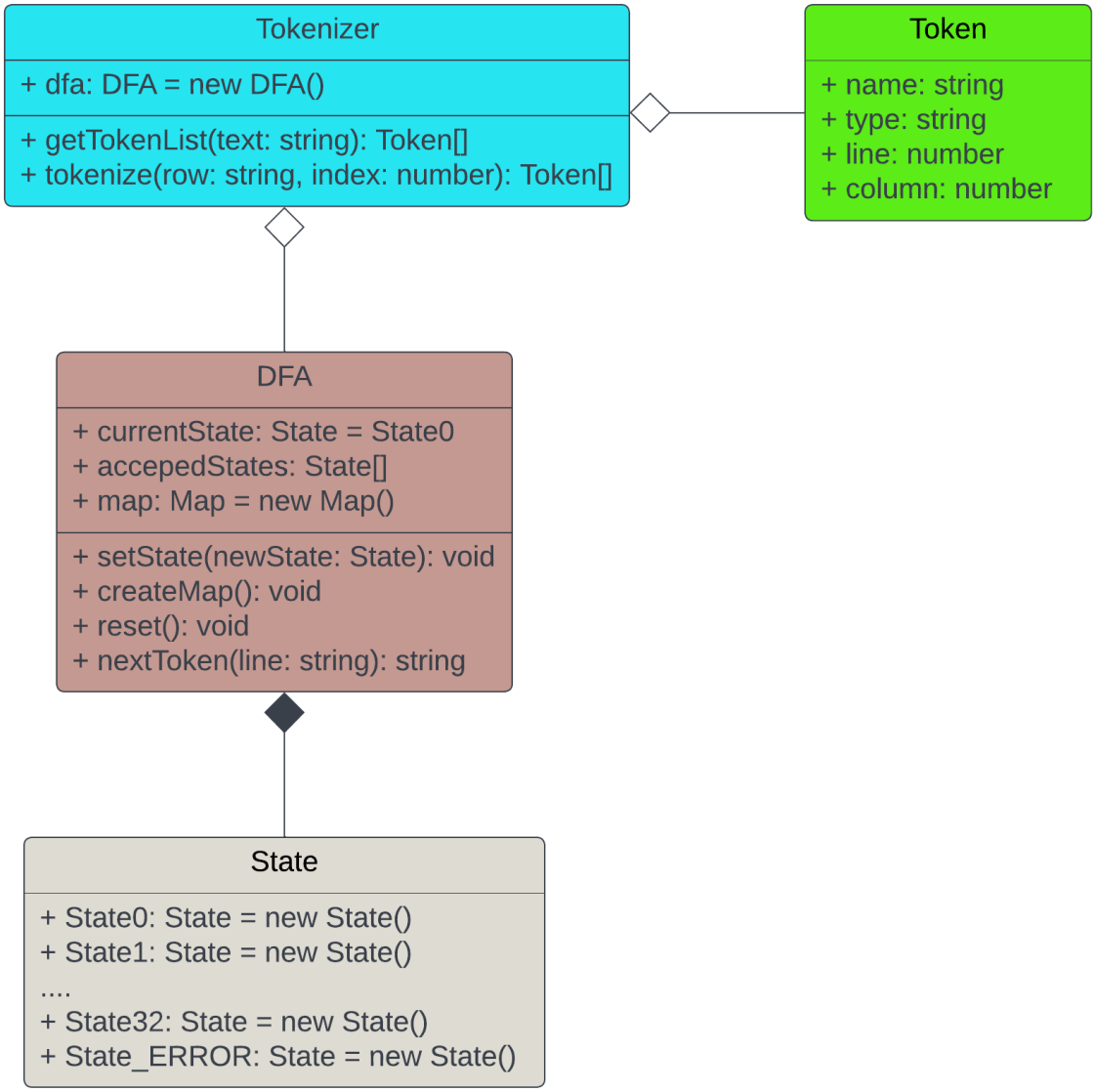


Bild 3.10: *UML-Diagramm fürs Tokenizer-Paket*

3.3.5 Component-Paket

Das Komponentenpaket liefert allgemeine Informationen über die Komponenten hinsichtlich der darin enthaltenen Funktionen und Attribute. Wie man in der Abbildung 3.11 sehen kann, besteht eine Funktion aus einer oder mehreren Anweisungen, und jeder Anweisungstyp verfügt über seine eigene Klasse.

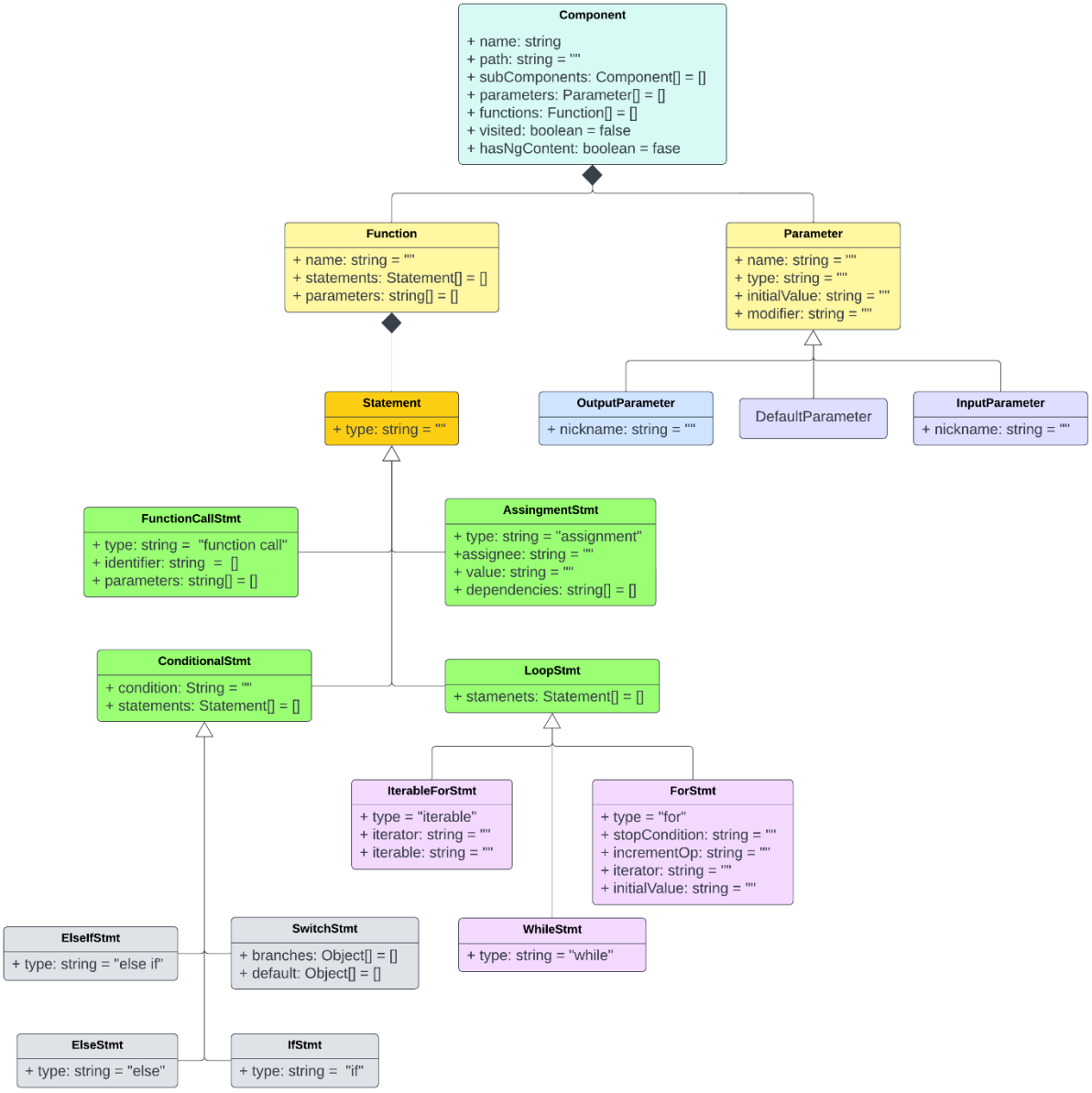


Bild 3.11: *UML-Diagramm fürs Component-Paket*

3.3.6 Builder- und Parser-Paket

Nach der Tokenisierung des Inhalts einer Datei wird die Tokenliste je nach dem Dateityp an eine der drei Parsing-Klassen (*TypeScriptParser*, *HTMLParser* und *AppModuleParser*) weitergeleitet, um die Reihenfolge der Token zu überprüfen und wertvolle Informationen daraus zu extrahieren. Es ist wichtig zu erwähnen, dass in diesen Klassen die in einem früheren Kapitel vorgestellten EBNF-Produktionsregeln umgesetzt sind, wobei die Nichtterminalsymbole mit Großbuchstaben hervorgehoben werden.

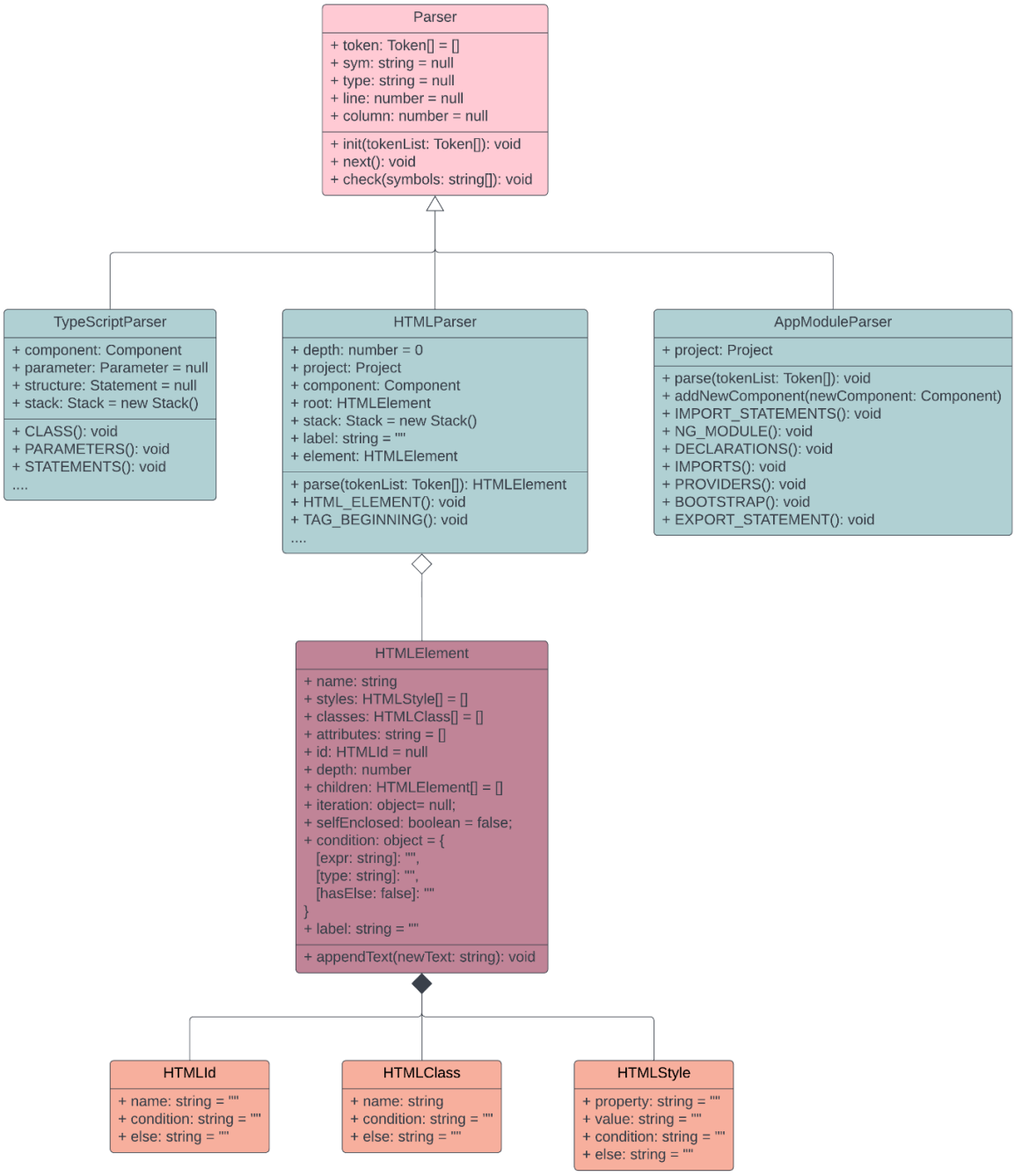


Bild 3.12: *UML-Diagramm fürs Parser-Paket*

Das Builder-Paket hingegen übernimmt die vom Parsing-Paket bereitgestellten Informationen und verwendet eine Reihe von Funktionen, um die entsprechenden React-Komponenten zu erstellen. Es besteht aus zwei Hauptklassen: *DOMBuilder*, der für die Erstellung des HTML-Baums zuständig ist, und *ComponentBuilder*, der die Komponenten selbst erstellt. Das UML-Diagramm dieses Pakets ist unten abgebildet.

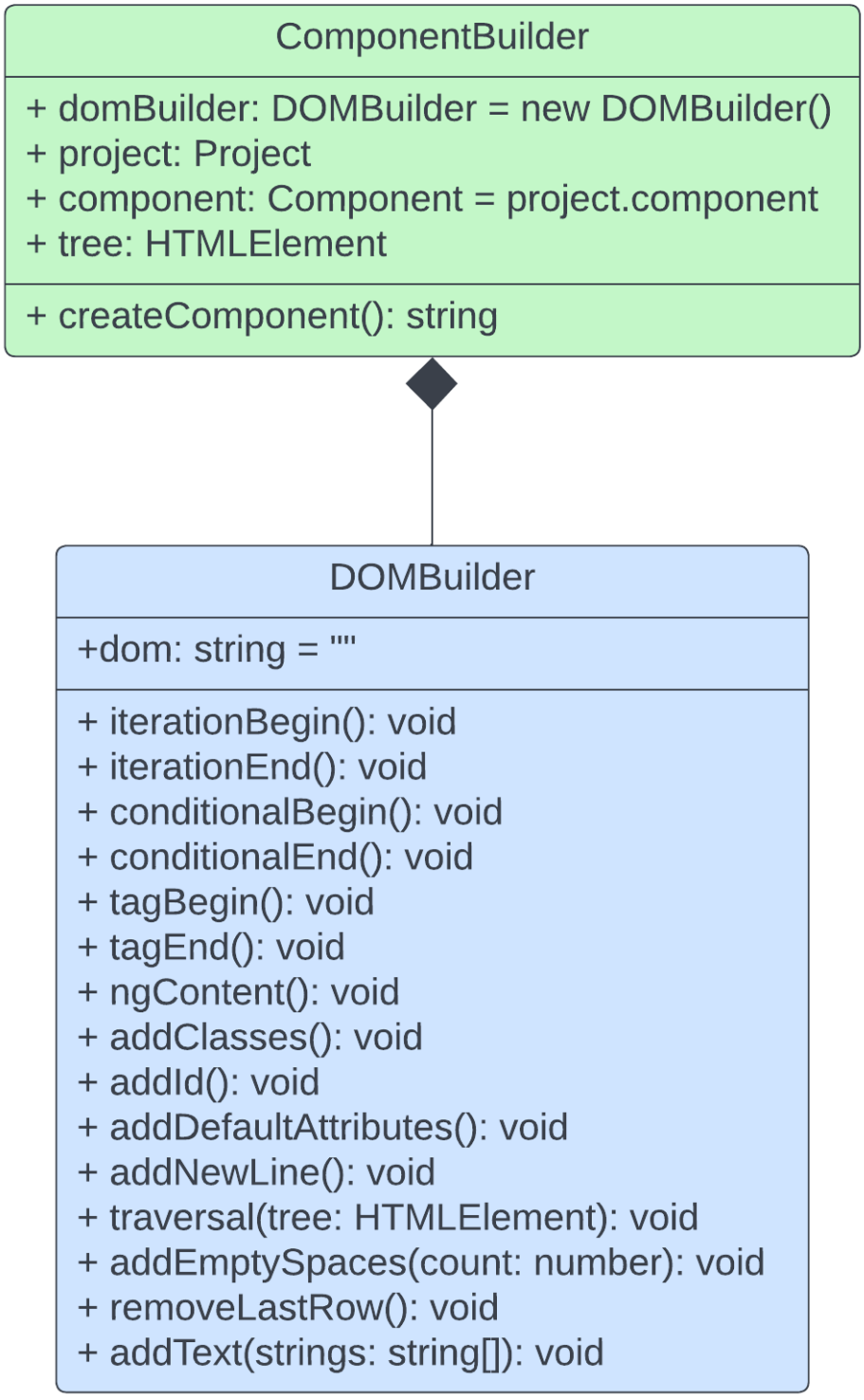


Bild 3.13: *UML-Diagramm fürs Builder-Paket*

3.3.7 Datenbankverbindung

Die Anwendung verfügt über keinen Backend-Server, daher werden alle von den Benutzern empfangenen Nachrichten in einer Firestore-Datenbank gespeichert. Für das Absenden der Anfragen wurde die eingebaute *Fetch API* verwendet. Das Bild 3.14 zeigt eine Codequenz mit dem Aufruf der *fetch()*-Funktion, deren Ziel ist einen neuen Eintrag in die Datenbank hinzuzufügen. Am Ende dieses Aufrufs wird auf dem Bildschirm eine Meldung angezeigt, ob der Eintrag erfolgreich hinzugefügt wurde oder ob ein Fehler aufgetreten ist.

```
const message = {
  name,
  email,
  text,
};

try {
  await fetch(
    "https://angular-converter-default-rtdb.firebaseio.com/messages.json",
    {
      method: "POST",
      body: JSON.stringify(message),
      headers: {
        "Content-Type": "application/json",
      },
    }
  );
```

Bild 3.14: *Das Hinzufügen eines neuen Eintrags in die Datenbank*

Schlusswort

4.1 Weitere Entwicklung

Da diese Anwendung nur eine begrenzte Anzahl von Angular-Funktionen unterstützt, besteht die Möglichkeit, weitere Angular-Funktionen wie Routing, Animationen und Dienste (engl. *services*) hinzuzufügen. Darüber hinaus können solche Umwandlungsanwendungen auch für andere Frameworks erstellt werden (z. B. von React zu Angular oder von Angular zu Vue). Noch besser wäre eine Anwendung, die eine serverseitige Website in eine clientseitig generierte Website umwandelt. Es ist wichtig zu erwähnen, dass eine Anwendung mit vollem Funktionsumfang nicht sehr günstig wäre, da sich die meisten Rahmenwerke derzeit in ständiger Entwicklung befinden (z. B. alle sechs Monate wird eine neue Version von Angular veröffentlicht) und die Anwendung daher ständig gewartet und aktualisiert werden soll.

4.2 Zusammenfassung

Angular2React ist eine Web-Anwendung, die darauf abzielt, einfache Angular-Projekte in ihr React-Äquivalent umzuwandeln. Da es derzeit nur sehr wenige oder gar keine derartige Anwendung gibt, kann dieser als ein Prototyp betrachtet werden. In der voliegenden Arbeit wurden die Besonderheiten dieser Anwendung vorgestellt, von denen die wichtigste die Funktionsweise ist.

Literaturverzeichnis

[1] Madhuri A. Jadhav, Balkrishna R. Sawant and Anushree Deshmukh. Single Page Application using AngularJS. International Journal of Computer Science and Information Technologies (IJCSIT), 6 (3): 2876-2879, 2015.

[2] Sanchit Aggarwal. Modern Web-Development using ReactJS. In International Journal of Recent Research Aspects (IJJRA), 5(1): 133-137, March 2018.

[3] Mark Tielens Thomas. React in action. Manning, 1st edition, Shelter Island, New York, July 2018. ISBN: 978-1617293856.

[4] Alex Banks, Eve Porcello. Learning React: Functional Web Development with React and Redux. O'Reilly Media, 1st edition, Newton, Massachusetts, May 2017. ISBN: 978-1491954621.

[5] Azat Mardan. React Quickly: Painless web apps with React, JSX, Redux, and GraphQL. Manning, 1st edition, Shelter Island, New York. September 2018. ISBN: 978-1617293344.

[6] Anthony Accomazzo, Nate Muray, Ari Lerner. Fullstack React: The complete guide to ReactJs and friends. Fullstack.io, San Francisco, California, July 2017. ISBN: 978-0991344628.

[7] Marc Garreau. Redux in action. Manning, 1st edition, Shelter Island, New York, June 2018. ISBN: 978-1617294976

[8] Shyam Seshadri. Angular: Up and Running: Learning Angular, Step by Step. O'Reilly, Newton, Massachusetts, June 2018. ISBN: 978-1491999837.

[9] Jeffry Houser. Learn With: Angular 4: Collected Essays: Angular CLI, Unit Testing, Debugging TypeScript, and Angular processes. DotComIt LLC, Meriden, Connecticut, October 2017. ASIN: B076ZXN6B9

[10] Jeremy Wilken. Angular in action. Manning, 1st edition, Shelter Island, New York, April 2018. ISBN: 978-1617293313.

[11] Gavin Bierman, Martin Abadi, Mads Torgersen. Understanding TypeScript. In Lecture Notes in Computer Science (LNCS), Washington, SUA, May 2014. ISBN: 978-3-662-44202-9

[12] Anurag Kumar, Ravi Kumar Singh. Comparative Analysis of AngularJS and ReactJS. In International Journal of Latest Trends in Engineering and Technology (IJLTET), 7(4): 225-227, November 2016.

[13] David Binkley. Source Code Analysis. In Future of Software Engineering (FOSE), Loyola College Baltimore, Mariland, USA. June 2007.

[14] Stefano Crespi Reghizzi, Luca Breveglieri, Angelo Morzenti. Formal Languages and Compilation. Springer, 2nd edition, Berlin, May 2009. ISBN: 978-1848820494.
[15] Wuu Yang, Chey-Woei Tsay, Jien-Tsai Chan. On the applicability of the longest-match rule in lexical analysis. Computer Languages, Systems & Structures. 28 (3): 273–288. 2002
[16] Alfred Aho, Jeffrey Ullman, Ravi Sethi, Monica Lam. Compilers: Principles, Techniques and Tools. Addison Wesley, Boston, Massachusetts, 2nd edition, August 2006. ISBN: 978-0321486813.
[17] Vaibhav Kumar Rai: Introduction of lexical analysis, https://www.geeksforgeeks.org/introduction-to-syntax-analysis-in-compiler-design/
[18] John Hopcroft, Rajeev Motwani, Jeffrey Ullman. Introduction to Automata Theory, Languages, and Computation. Pearson, 3rd edition, London, June 2006. ISBN: 978-0321455369.
[19] Reinhard Wilhelm, Helmut Seidl, Sebastian Hack. Compiler Design: Syntactic and Semantic Analysis. Springer, Berlin, May 2013. ISBN 978-3-642-17540-4.