IN DIESEM KAPITEL

- Werkzeuge kennenlernen
- Entwickeln mit Angular und TypeScript
- Angular CLI
- Die Entwicklungsumgebung einrichten

Kapitel 1
Die Werkzeuge

Wer mit Angular und TypeScript eine Webanwendung entwickeln möchte, braucht als Erstes eine Entwicklungsumgebung. Dieses Kapitels will Ihnen helfen, solch eine Entwicklungsumgebung aufzusetzen. Ich zeige Ihnen die verschiedenen Werkzeuge, die Sie dazu brauchen, und was diese tun. Am Ende des Kapitels lernen Sie die Beispielanwendung kennen.

Die Entwicklungsumgebung

Vor gar nicht so langer Zeit reichten noch ein Texteditor, ein Browser und ein Webserver aus, um zumindest eine kleinere Webanwendung schreiben zu können. Wenn Sie aber mit Angular und TypeScript arbeiten wollen, benötigen Sie auch für die kleinste Webanwendung wesentlich mehr Werkzeuge: Sie brauchen eine Entwicklungsumgebung, die unter anderem die folgenden Programme beziehungsweise Werkzeuge beinhaltet:

- ✔ Node.js
- ✔ npm
- ✔ den TypeScript-Compiler
- ✔ einen Texteditor
- ✔ einen Webserver
- ✔ einen Browser

Dazu kommen oft noch weitere Werkzeuge für das Testen der Webanwendung, das Formatieren und Minimieren des Codes und natürlich für die Überprüfung des Codes auf Fehler.

Leider reicht es nicht aus, die Werkzeuge zu installieren. Diese müssen auch konfiguriert werden. Um den Entwicklern eine bisschen Arbeit abzunehmen, hat das Angular-Team Angular CLI entwickelt. Mithilfe dieses Werkzeugs können Sie mit wenig Aufwand ein neues Angular-Projekt starten. Mehr über Angular CLI erfahren Sie im Abschnitt *Das Hauptwerkzeug: Angular CLI*.

Node.js und npm

Node.js ist eine Laufzeitumgebung für JavaScript. Damit können Sie Kommandozeilenwerkzeuge, Webserver und vieles mehr entwickeln und ausführen. Werkzeuge wie Gulp, Grunt, Karma, ESLint, JSHint kennen Sie vielleicht schon. Eines haben sie alle gemeinsam: Sie wurden in JavaScript geschrieben und werden mit Node.js ausgeführt. Der TypeScript-Compiler und Angular CLI sind da keine Ausnahmen. Aus diesem Grund brauchen Sie Node.js für die Entwicklung einer Angular-Anwendung.

Npm ist ein weiteres wichtiges Element im Werkzeugkasten jedes Webentwicklers. Es wird benutzt, um *Pakete* zu verwalten. Diese Pakete, auch als *npm-Pakete* bekannt, können JavaScript, CSS, HTML und mehr beinhalten. Sie definieren npm-Pakete als *Abhängigkeiten* Ihres Projekts. Beispielsweise hängt ein Angular-Projekt, unter anderem, vom @angular/core-npm-Paket und vom typescript-npm-Paket ab.

In einer Konfigurationsdatei definieren Sie die Abhängigkeiten Ihres Projekts, die Sie im Nachgang dann mit npm installieren. Die Konfigurationsdatei heißt `package.json`. Alle npm-Pakete, die Sie dort definieren, können nur innerhalb des Projekts verwendet werden. Diese werden deshalb *lokale npm-Pakete* genannt. Lokale npm-Pakete befinden sich im Verzeichnis `node_modules` innerhalb Ihres Projektverzeichnisses.

Im Listing 1.1 sehen Sie einen Ausschnitt aus einer `package.json`-Datei. Neben dem Namen (`name`-Eigenschaft) und der Version (`version`-Eigenschaft) sehen Sie dort noch die Eigenschaften `dependencies`, `devDependencies` und `scripts`. Die Ersten beiden sind Objekte, die die Namen und Versionen von Abhängigkeiten beinhalten. Im `dependencies`-Objekt stehen alle Abhängigkeiten, die Ihre Anwendung zum Laufen braucht. Im `devDependencies`-Objekt stehen Abhängigkeiten, die nur bei der Entwicklung, aber nicht mehr zur Laufzeit gebraucht werden. Im `scripts`-Objekt stehen Skripte, die Sie über npm ausführen können. Beispielsweise sind `start` der Name des Skripts und `ng serve` das Kommando, das ausgeführt wird (siehe den Abschnitt *Das ng-serve-Kommando*).

```
{
  "name": "contacts-manager",
  "version": "0.0.0",
  "scripts": {
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    ...
  },
```

```
  "dependencies": {
    "@angular/common": "~7.0.0",
    "core-js": "^2.5.4",
    ...
  },
  "devDependencies": {
    "@angular/cli": "~7.0.3",
    ...
  },
  ...
}
```

Listing 1.1: Ausschnitt aus einer package.json-Datei. Diese wurde von Angular CLI generiert.

Allgemein führen Sie npm-Skripte mit `npm run Skriptname` aus. Es gibt einige Skripte, wie zum Beispiel `start` und `test`, die Sie auch ohne `run` ausführen können. Welche Skripte das genau sind, können Sie auf der Webseite von npm unter `https://docs.npmjs.com/files/package.json` nachlesen. Mit dem Kommando `npm run` sehen Sie alle Skripte, die in der `package.json`-Datei des Projekts definiert sind.

In der `package.json`-Datei haben der Zirkumflex (^) und die Tilde (~) vor einer Versionsnummer eine spezielle Bedeutung: Die Tilde fixiert die ersten beiden Zahlen der Version und nimmt für die dritte einfach die neuste. Der Zirkumflex fixiert die erste Zahl, für die anderen zwei wird die neuste Version genommen. Betrachten wir als Beispiel eine Abhängigkeit `abc`, von der die Versionen `6.0.0`, `6.0.1` und `6.1.0` existieren. Wenn in der `package.json` `6.0.0` steht, wird für die Abhängigkeit die Version `6.0.0` installiert. Wenn dort `~6.0.0` steht, wird die Version `6.0.1` installiert. Wenn dort `^6.0.0` steht, dann wird die Version `6.1.0` installiert.

Neuere npm-Versionen erzeugen zusätzlich zum `node_modules`-Verzeichnis auch eine Datei namens `package-lock.json`. Dort stehen alle npm-Pakete, die sich im `node_modules`-Verzeichnis befinden, mit ihrer jeweiligen Version. Falls Sie ein Werkzeug für die Versionierung Ihres Codes verwenden, wie zum Beispiel git, sollten Sie diese Datei auch hinzufügen. Sie stellt sicher, dass nachträgliche Installationen die exakt gleichen Versionen von den Abhängigkeiten besitzen.

Npm erlaubt auch die Installation von *globalen npm-Paketen*. Diese Pakete können dann in jedem Projekt verwendet werden. Meistens installieren Sie Kommandozeilenwerkzeuge, wie zum Beispiel Angular CLI, global.

Wie Sie Node.js und npm installieren, erfahren Sie im Abschnitt *Beispielanwendung: Umgebung einrichten*.

Mehr über die `package.json` erfahren Sie auf der Webseite von npm unter `https://docs.npmjs.com/files/package.json`. Mehr über die `package-lock.json` erfahren Sie auf `https://docs.npmjs.com/files/package-lock.json`. Auf der npm-Webseite finden Sie auch weitere Informationen über das Werkzeug und seine Features. Falls Sie noch nie mit npm gearbeitet haben, empfehle ich Ihnen, zumindest den *Getting Started*-Teil zu lesen. In der Regel haben Versionen in der `package.json` drei Zahlen, die mit Punkten getrennt werden. Hinter diesen Versionsbezeichnungen steckt eine Versionierungsprozess namens Semantic Versioning. Auch Angular folgt diesem Prozess. Mehr darüber erfahren Sie auf `https://semver.org/lang/de/`.

Texteditoren

Theoretisch können Sie bei der Entwicklung jeden beliebigen Texteditor nutzen. Allerdings sind nicht alle Texteditoren gleich gut für die Entwicklung von Angular-Anwendungen geeignet. Sie sollten einen Texteditor auswählen, der mindestens Syntax-Highlighting und Code-Vervollständigung für TypeScript unterstützt. Auch die Unterstützung von Code-Refactoring ist etwas, worauf Sie bei der Wahl eines Texteditors achten sollten. Gerade letzteres wird Ihnen, vor allem bei größeren Projekten, sehr helfen.

Ich nutze zwar hier den Term »Texteditor«, ich meine damit aber auch integrierte Entwicklungsumgebungen (IDEs). Ich unterscheide nicht explizit zwischen den beiden Ausdrücken, da die meisten modernen Texteditoren sich mithilfe von Plugins zu integrierten Entwicklungsumgebungen umfunktionieren lassen.

In der Tabelle 1.1 sehen Sie vier Texteditoren, die gut für die Entwicklung von Angular-Anwendungen geeignet sind. Alle vier funktionieren unter Windows, Linux und MacOS. Allerdings bieten die vier Editoren unterschiedliche Grade der TypeScript- beziehungsweise Angular-Unterstützung an. Falls Sie genügend Zeit haben, empfehle ich Ihnen, alle vier Texteditoren zu testen und denjenigen auswählen, der am besten zu Ihnen passt. Falls Sie die Zeit nicht haben, empfehle ich Ihnen Visual Studio Code, wozu Sie mindestens das Angular-Language-Service-Plugin installieren sollten.

	Atom	Sublime Text	Visual Studio Code	Webstorm
Open-Source	Ja	Nein	Ja	Nein
Kostenlos	Ja	Nein (kostenlose Evaluation)	Ja	Nein (kostenlose Evaluation)
TypeScript-Unterstützung	durch Plugin	durch Plugin	integriert	integriert
Angular-Unterstützung	derzeit nein	derzeit nein	durch Plugin	integriert

Tabelle 1.1: Überblick über Texteditoren für die Entwicklung von Angular-Anwendungen

Mit TypeScript- beziehungsweise Angular-Unterstützung meine ich konkret die Unterstützung von sogenannten *Language Services*. Diese ermöglichen Dinge wie Code-Refactoring, Code-Vervollständigung und so weiter. Diese Unterstützung ist für mich persönlich bei der Auswahl eines Texteditors besonders wichtig. Natürlich gibt es auch noch weitere Entscheidungskriterien, wie zum Beispiel die Integration mit Angular CLI oder die Unterstützung von Code-Schnipseln.

Weitere Informationen über die vier Editoren finden Sie auf der jeweiligen Webseite. Dort können Sie diese auch herunterladen.

- ✔ **Atom:** `https://atom.io/`.
- ✔ **Sublime Text:** `https://www.sublimetext.com/`.
- ✔ **Visual Studio Code:** `https://code.visualstudio.com/`.
- ✔ **Webstorm:** `https://www.jetbrains.com/webstorm/`.

Mehr über den Angular Language Service erfahren Sie auf `https://angular.io/guide/language-service`.

Suchen Sie in der Plugin-Verwaltung jedes Texteditors nach `angular` beziehungsweise nach `typescript`. Das ist die einfachste Möglichkeit, interessante Plugins zu finden. Am besten installieren Sie keine Plugins mit `AngularJS` in ihrem Namen oder ihrer Beschreibung. Diese wurden für die Version 1.x von Angular geschrieben und sind in der Regel mit Versionen größer/gleich 2.x nicht kompatibel.

Der TypeScript-Compiler

Der *TypeScript-Compiler* hat zwei Aufgaben:

- ✔ Er informiert Sie über Typfehler in Ihrem Code.
- ✔ Er wandelt Ihren TypeScript-Code in JavaScript-Code um.

Bei dieser Umwandlung geht es hauptsächlich um die Syntax. Neuere Syntax, wie zum Beispiel eine Klassendefinition, wird in Syntax umgewandelt, die auch ältere Browser ausführen können. Was genau umgewandelt wird, hängt von der JavaScript-Zielversion ab. Die Zielversion bestimmt im Endeffekt, in welchen Browsern Sie Ihre Anwendung nutzen können. Sie können diese mit der Compiler-Option `target` ändern. Mehr über Compiler-Optionen erfahren Sie im Abschnitt *Compiler-Optionen*.

Der TypeScript-Compiler unterstützt JavaScript-Features, die von den Browsern noch nicht unterstützt werden. TypeScript ist eine Übermenge, ein Superset von JavaScript.

Neue Methoden und Objekte, wie zum Beispiel das Map-Objekt oder die findIndex-Methode von Arrays werden nicht umgewandelt. Wenn Sie diese verwenden wollen, brauchen Sie womöglich Polyfills. Ob Polyfills tatsächlich nötig sind, hängt von den Browsern ab, in denen Ihre Anwendung laufen muss.

Angular CLI nutzt aktuell `ES5` als Zielversion. Das heißt, dass bei die Umwandlung dafür sorgt, dass das JavaScript in allen Browsern funktioniert, die den ECMAScript-5-Standard implementieren. Das sind alle Browser ab Internet Explorer 9.

Sie müssen den TypeScript-Compiler nicht selbst installieren. Das übernimmt Angular CLI für Sie.

Mehr über die Syntax und die Features von TypeScript erfahren Sie im Anhang A. Falls Sie noch keine Erfahrung mit TypeScript haben, empfehle ich Ihnen, jetzt als Erstes diesen Anhang zu lesen.

Polyfills

Ein *Polyfill* ist Code, der ein Feature für Browser implementiert, die dieses Feature noch nicht unterstützen. Das Feature kann bereits Teil eines Webstandards sein oder ein Vorschlag für einen neuen Standard. Der Begriff »Polyfill« wurde von Remy Sharp im Jahr 2009 geprägt.

Ein Beispiel für einen relativ neuen Webstandard ist die fetch-API. Diese wird von modernen Browsern wie Chrome und Firefox unterstützt, aber nicht von Internet Explorer. Wenn Sie also die fetch-API im Internet Explorer nutzen möchten, müssen Sie dafür ein Polyfill nutzen.

Kompilierfehler

Kompilierfehler bemerken Sie, wenn der TypeScript-Compiler Fehler in Ihrem Code findet. Hauptsächlich informiert Sie der Compiler über Typ- und Syntaxfehler.

Ein *Typfehler* entsteht, wenn Sie ein Konstrukt, wie zum Beispiel eine Variable falsch verwenden. Falsch verwenden heißt in diesem Fall, dass Sie den Typ des Konstrukts nicht so verwenden, wie dieser definiert ist. Zum Beispiel ergibt es einen Typfehler, wenn Sie einer Variable vom Typ `string` eine Zahl zuzuweisen versuchen. Auch der Aufruf einer Funktion mit falscher Parameteranzahl oder falschen Parametertypen verursacht einen Typfehler. Obwohl Typfehler nicht immer zur Laufzeitfehlern führen, sollten Sie sich stets bemühen, diese zu korrigieren.

Syntaxfehler entstehen, wenn Sie beim Schreiben von TypeScript-Code einen Fehler machen. Zum Beispiel, wenn Sie ein Komma vergessen oder den Namen bei einer Funktionsdeklaration. Im Browser führen solche Fehler zur einer Exception und das Programm läuft nicht

weiter. In TypeScript führt ein Syntaxfehler zu einem Kompilierfehler. Allerdings versucht der Compiler, den Syntaxfehler im erzeugten JavaScript selbst zu beheben. Der Compiler kann also theoretisch trotz Syntaxfehlern gültiges JavaScript erzeugen.

Darüber hinaus kann der Compiler Sie auch über folgende Fehler informieren:

- ✔ unerreichbare Code-Teile,
- ✔ nicht verwendete Funktionsparameter und Variablen,
- ✔ das Fehlen einer `break`- oder `return`-Anweisung in einer `switch-case`-Anweisung.

Standardmäßig informiert Sie der Compiler nur über unerreichbare Code-Teile, zum Beispiel Code in einer Funktion, der unter einer `return`-Anweisung steht. Damit der Compiler Sie auch auf die anderen zwei Fehler aufmerksam macht, müssen Sie diese Optionen aktivieren (siehe Abschnitt *Compiler-Optionen*).

In der Regel haben unerreichbare Code-Teile keine Auswirkungen auf das Funktionieren Ihrer Anwendung. Dasselbe gilt auch für nicht genutzte Funktionsparameter und Variablen. Allerdings empfehle ich Ihnen, auch solche Fehler zu korrigieren und den überflüssigen Code zu entfernen.

Eine `break`- oder `return`-Anweisung wird manchmal absichtlich ausgelassen, wenn derselbe Code in mehr als einem Fall ausgeführt werden soll. Allerdings rate ich Ihnen trotzdem, diese Fehlermeldung zu aktivieren. Die Fehler, die entstehen können, wenn Sie eine `break`- oder eine `return`-Anweisung vergessen, sind oft schwer zu finden. Da ist es besser, den gemeinsamen Code in einer Funktion zu definieren und diese dann jeweils aufzurufen.

Compiler-Optionen

Mit *Compiler-Optionen* steuern Sie den TypeScript-Compiler. Sie können ihm zum Beispiel sagen, wo das erzeugte JavaScript abgelegt werden soll, ob Sourcemaps erzeugt werden sollen und so weiter.

Es gibt zwei Möglichkeiten, um den Compiler zu konfigurieren: Sie können Parameter beim Aufruf übergeben oder eine Konfigurationsdatei nutzen. Da die Nutzung dieser Parameter ziemlich umständlich ist, konzentriere ich mich hier auf die Konfigurationsdatei. Das Gesagte gilt aber sinngemäß immer für beide Möglichkeiten.

Die Konfigurationsdatei wird im JSON-Format geschrieben und hat den Namen `tsconfig.json`. Diese befindet sich im Hauptverzeichnis des Projekts. Angular CLI erzeugt für Sie die Konfigurationsdatei mit allen nötigen Optionen, die eine Angular-Anwendung braucht. Am besten ändern Sie die Optionen von Angular CLI nicht. Änderungen können dazu führen, dass Angular CLI Ihre Anwendung nicht mehr korrekt zusammenbaut.

Im Listing 1.2 sehen Sie verschiedene Optionen. Ich empfehle Ihnen, diese in die von Angular CLI erzeugte Konfigurationsdatei einzufügen. Eine Beschreibung der Optionen finden Sie in der Tabelle 1.2. In der Beschreibung erkläre ich, was passiert, wenn die Optionen die Werte aus Listing 1.2 haben.

```
{
  "compilerOptions": {
    ...,
    "noFallthroughCasesInSwitch": true,
    "noImplicitAny": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true
  }
}
```

Listing 1.2: Ausschnitt aus der Konfigurationsdatei für den TypeScript-Compiler

Option	Beschreibung
noFallthroughCasesInSwitch	Fehlermeldung, wenn eine switch–case-Anweisung keine break- oder return-Anweisung besitzt
noImplicitAny	Fehlermeldung, wenn der Typ implizit any ist (siehe Anhang A Basistypen)
noUnusedLocals	Fehlermeldung, wenn Variablen definiert sind, die nicht verwendet werden
noUnusedParameters	Fehlermeldung, wenn Funktionsparameter definiert sind, die nicht verwendet werden

Tabelle 1.2: Optionen für den TypeScript-Compiler

Mehr Informationen über die Konfigurationsdatei finden Sie auf `https://www.typescriptlang.org/docs/handbook/tsconfig-json.html`. Weitere Optionen für den TypeScript-Compiler stehen auf `https://www.typescriptlang.org/docs/handbook/compiler-options.html`.

Sourcemaps

Sourcemaps sind eine Möglichkeit, den kompilierten JavaScript-Code dem TypeScript-Code zuzuordnen. Wenn Sie zum Beispiel in der Konsole Ihres Browsers eine Exception bekommen, teilt diese Ihnen mit, in welcher Zeile und Position die Exception aufgetreten ist. Allerdings zeigt diese Position auf den JavaScript-Code. Mithilfe von Sourcemaps ist der Browser in der Lage, diese JavaScript-Position der entsprechenden Position im TypeScript-Code zuzuordnen. Das macht das Finden von Fehlern wesentlich einfacher, da Sie direkt sehen, wo der Fehler im ursprünglichen Code sitzt.

Sourcemaps werden nicht nur von TypeScript genutzt. Sie werden allgemein immer dann verwendet, wenn Sie eine kompilierte und eine ursprüngliche Version des Codes haben, beispielsweise bei SASS und CSS oder minimiertem und nichtminimiertem JavaScript.

Die Sourcemaps werden oft in eine separaten Datei geschrieben. Diese erkennen Sie an der `.map`-Endung.

TSLint, codelyzer und Prettier

TSLint ist ein Werkzeug, mit dem Sie Ihren Code analysieren können. Es überprüft Wartbarkeit, Konsistenz und Schreibstil Ihres TypeScript-Codes. Zusätzlich kann TSLint auch überprüfen, ob Ihr Code frei von Fehlern ist, die die Funktionalität Ihrer Anwendung beeinträchtigen können. Dabei geht es hauptsächlich um TypeScript-Konstrukte, die bei der Nutzung oft zu Bugs führen.

Was genau überprüft wird und was als Fehler betrachtet wird, hängt von der Konfiguration ab. Sie können TSLint mithilfe von Regeln konfigurieren. Von Haus aus hat TSLint über 100 Regeln. In seiner Konfigurationsdatei namens `tslint.json` definieren Sie, welche Regeln für Ihr Projekt aktiv sind. Einige Regeln können Sie auch zusätzlich konfigurieren. Zum Beispiel definieren Sie mit der Regel `quotemark`, ob Ihre Anwendung einzelne oder doppelte Anführungszeichen nutzt.

Regeln, die in der Konfigurationsdatei stehen, gelten prinzipiell für alle TypeScript-Dateien in einem Projekt. Sie können aber auch Regeln für einzelne Dateien und sogar einzelne Zeilen aktivieren und deaktivieren. Das ist manchmal nötig, wenn Sie in bestimmten Dateien Konstrukte benötigen, die Sie aber im Allgemeinen nicht erlauben wollen.

Darüber hinaus gibt es für einige Regeln sogenannte *Fixer*. Das bedeutet, dass diese Regeln Ihren Code automatisch umschreiben dürfen, sodass er regelkonform wird. Damit die Fixer aktiv werden, müssen Sie das Tool mit der Option `--fix` aufrufen.

Alle Regeln von TSLint finden Sie auf `https://palantir.github.io/tslint/rules/`. Dort steht auch, welche Regeln einen Fixer haben.

Obwohl TSLint über 100 Regeln besitzt, gibt es einige Dinge, die nicht überprüft werden können. Für solche Fälle können Sie eigene Regeln implementieren und diese in der TSLint-Konfiguration nutzen. Extraregeln für Angular-Anwendungen gibt es schon. Dafür nutzen Sie das *codelyzer-Plugin*. Die Regeln von codelyzer stellen unter anderem sicher, dass Sie Ihre Angular-Bausteine korrekt benennen und dass Ihr Code den Best Practices für Angular-Projekte folgt.

Alle Regeln für das codelyzer-Plugin finden Sie auf `http://codelyzer.com/rules/`.

Sie müssen TSLint und das codelyzer-Plugin nicht selbst installieren. Das übernimmt Angular CLI für Sie. Auch eine Konfigurationsdatei wird von Angular CLI angelegt.

In einem Angular CLI Projekt nutzen Sie das `ng lint`-Kommando, um den Code zu überprüfen. Mit der Option `--fix` aktivieren Sie die Fixer.

Prettier übernimmt die Formatierung Ihres Codes. Damit brauchen Sie sich keine Gedanken mehr zu machen über die Anzahl an Leerzeichen vor oder nach einer Klammer oder die Länge einer Code-Zeile. Sie lassen Prettier über den Code laufen und schon formatiert es den Code einheitlich durch. Beachten Sie, dass Sie Prettier kaum konfigurieren können. Entweder gefällt Ihnen, wie das Werkzeug Ihren Code formatiert, oder Sie müssen darauf verzichten.

Prettier wird von Angular CLI nicht automatisch installiert. Sie müssen es selbst installieren und in Ihr Projekt integrieren. Am besten installieren Sie auch das `tslint-config-prettier`-npm-Paket. Ohne dieses Paket könnten einige TSLint-Formatierungsregeln in Konflikt mit den Regeln von Prettier geraten.

Prettier gibt es auch als Plugin für einige Texteditoren, beispielsweise für Visual Studio Code.

Die Hauptseite von Prettier finden Sie auf `https://prettier.io`. Dort wird erklärt, wie Sie das Werkzeug installieren und konfigurieren.

Das Hauptwerkzeug: Angular CLI

Angular CLI ist ein Werkzeug, das Ihnen das Erstellen von und die Arbeit mit Angular-Projekten erleichtert. Es ist für Sie die einfachste Möglichkeit, weitere Werkzeuge aufzurufen, wie zum Beispiel den TypeScript-Compiler und TSLint. Im Wesentlichen übernimmt Angular CLI die Installation aller benötigten Werkzeuge und konfiguriert diese. Zusätzlich installiert es weitere npm-Pakete, die für eine Angular-Anwendung gebraucht werden. Angular CLI nutzen Sie in der Konsole des Betriebssystems oder des Texteditors, falls dieser eine Konsole besitzt. Wie Sie Angular CLI installieren, sehen Sie im Abschnitt *Beispielanwendung: Umgebung einrichten*.

Die offizielle Dokumentation für Angular CLI finden Sie unter `https://angular.io/cli`.

Falls Sie sich in der Konsole nicht besonders wohl fühlen, können Sie es mit *Angular Console* versuchen, einer grafischen Oberfläche für Angular CLI. Angular Console finden Sie auf `https://angularconsole.com`.

Features von Angular CLI

Die Hauptaufgabe von Angular CLI besteht darin, Ihnen den Aufwand für die Konfigurationen der verschiedenen Werkzeuge, die Sie für Ihre Angular-Anwendung brauchen, abzunehmen. Es richtet Ihnen also einen Build-Prozess ein. Natürlich können Sie eine Angular-Anwendung auch ohne Angular CLI entwickeln. Allerdings müssen Sie in diesem Fall Ihren Build-Prozess selbst konfigurieren. Aus Erfahrung kann ich Ihnen sagen, dass das einige Zeit kosten kann. Mit Angular CLI sparen Sie sich diese Zeit und können direkt loslegen.

Weitere Features von Angular CLI:

- ✔ Integration mit TSLint und codelyzer (siehe Abschnitt *TSLint, codelyzer und Prettier*),
- ✔ Integration mit Karma und Protractor für Unit- und End-to-End-Tests (siehe Teil V),
- ✔ Webserver mit Live-Reload,
- ✔ Integration mit dem TypeScript-Compiler (siehe Abschnitt *Der TypeScript-Compiler*),
- ✔ Integration mit dem Angular-Compiler,
- ✔ Generierung von Angular-Modulen, Komponenten und weiteren Bausteinen.

Bausteine generieren Sie mit dem `ng generate`-Kommando. Mit `ng generate --help` sehen Sie, welche Bausteine Angular CLI generieren kann.

Angular-Compiler

Der *Angular-Compiler* nimmt Ihre Komponenten und ihr Template (siehe Kapitel 2, Abschnitt *Komponenten*) und erzeugt daraus JavaScript-Code. Dieser Code wird ausgeführt, um die Ansicht Ihrer Anwendung aufzubauen.

Es gibt zwei Möglichkeiten, die Komponenten zu kompilieren:

- ✔ **Just-in-Time (JIT): Damit werden die Komponenten zur Laufzeit im Browser kompiliert.**
- ✔ **Ahead-of-Time (AOT): Damit werden die Komponenten beim Programmieren kompiliert.**

Die Wahl von Ahead-of-Time hat einige Vorteile gegenüber Just-in-Time:

- ✔ **Die Anwendung kann schneller angezeigt werden, da die Komponenten schon kompiliert sind.**
- ✔ **Der Angular-Compiler muss nicht vom Browser heruntergeladen werden. Dies macht das Angular-Framework wesentlich kompakter.**
- ✔ **AOT bietet mehr Sicherheit, weil das HTML nicht im Browser evaluiert wird.**
- ✔ **Fehler in den Templates können frühzeitig erkannt werden.**

Allerdings müssen Sie darauf achten, dass Ihr Code AOT-kompatibel ist. Das bedeutet, dass gewisse Sprachkonstrukte nicht erlaubt sind.

Mehr Informationen über AOT und die damit erlaubten Sprachkonstrukte finden Sie auf `https://angular.io/guide/aot-compiler`. Der Code in diesem Buch ist AOT-kompatibel.

Projekt initialisieren

Die Projektinitialisierung mit Angular CLI braucht im Prinzip nur ein Kommando. Sie rufen einfach in einer Konsole `ng new` auf. Angular CLI stellt Ihnen dann verschiedene Fragen, die Ihnen bei der Projektkonfiguration helfen. Nachdem Sie alle Fragen beantwortet haben, legt Angular CLI die nötigen Konfigurationsdateien an und installiert mittels npm alle Abhängigkeiten. Ihr neues Projekt befindet sich in einem Unterverzeichnis des aktuellen Verzeichnisses. Der Name des Projektverzeichnisses ist der gleiche wie der Name des Projekts, den Sie Angular CLI angegeben haben. Außer den Konfigurationsdateien befindet sich darin auch Beispiel-Code für eine Angular-Anwendung. Den Inhalt des Projektverzeichnisses beschreibe ich im Abschnitt *Die Verzeichnisstruktur von Angular CLI*.

Optionen für das Kommando:

- ✔ **`--interactive`**: Wenn Sie `ng new` mit `--interactive=false` aufrufen, stellt Ihnen Angular CLI keine Fragen. In diesem Fall müssen Sie den Namen des Projekts direkt mit dem Kommando angeben. Beispiel: `ng new --interactive=false projekt-name`.
- ✔ **`--style`**: Damit definieren Sie die Endung für Style-Dateien. Standardmäßig wird die Endung `.css` verwendet. Weitere Möglichkeiten sind `.scss`, `.sass`, `.less` und `.styl`. Beispiel: `... --style=scss`. Je nach Dateiendung wird Angular CLI die nötige Konvertierung nach CSS übernehmen.
- ✔ **`--skip-git`**: Standardmäßig erzeugt Angular CLI ein git-Repository für Ihr Projekt. Mit `... --skip-git ...` können Sie das unterbinden. Eine `.gitignore`-Datei wird immer angelegt, egal ob ein git-Repository erzeugt wird oder nicht.

Das `ng new --help`-Kommando zeigt Ihnen weitere mögliche Optionen und was diese tun.

Sie können mit `npm install` alle Abhängigkeiten eines existierenden Angular-CLI-Projekts installieren. Angular CLI installiert die Abhängigkeiten automatisch nur bei der Initialisierung eines neuen Projekts.

Die Verzeichnisstruktur von Angular CLI

Hier bekommen Sie einen kurzen Überblick über die Verzeichnisstruktur, die Angular CLI bei der Initialisierung anlegt. Sie sehen einige der vorkonfigurierten Verzeichnisse und Dateien und lernen, welche Rolle diese in Ihrem Projekt spielen. Ein Ausschnitt aus der Verzeichnisstruktur sehen Sie in Abbildung 1.1.

Auf der höchsten Ebene gibt es zwei Verzeichnisse und einige Dateien. Im `e2e`-Verzeichnis befinden sich Code und Konfigurationsdateien für die End-to-End-Tests. Im `src`-Verzeichnis befindet sich Ihre Anwendung mit allem Nötigem, wie zum Beispiel Bilder, Fonts und so weiter.

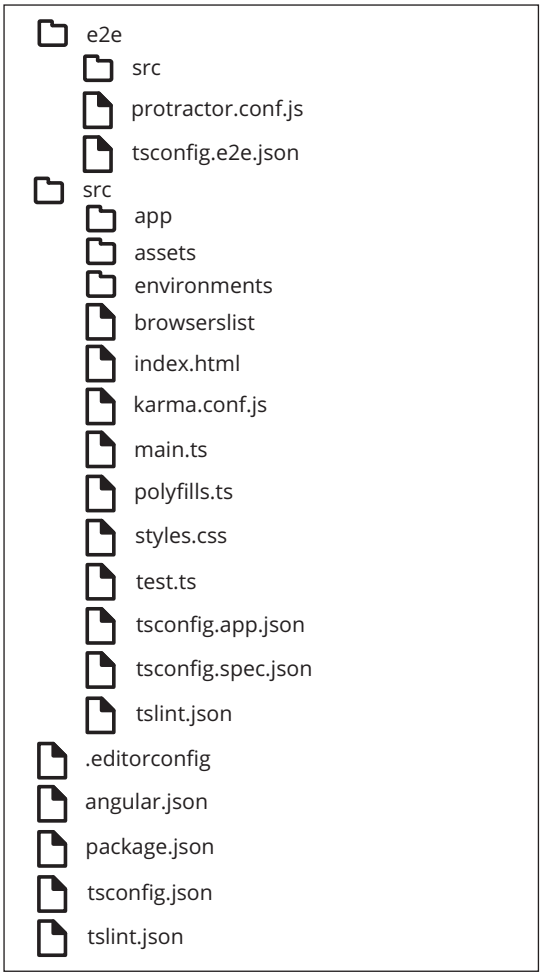

Abbildung 1.1: Verzeichnisstruktur eines Angular-CLI-Projekts

Dateien im Hauptverzeichnis:

- ✔ **.editorconfig**: Eine Konfigurationsdatei für Texteditoren. Diese kann auch in Projekten ohne Angular benutzt werden. Sie können dort allgemeine Codestyles definieren wie zum Beispiel die Anzahl der Einrückungen im Code.
- ✔ **angular.json**: Die Konfigurationsdatei von Angular CLI. Mehr darüber erfahren Sie im Abschnitt *Die Konfigurationsdatei von Angular CLI*.
- ✔ **package.json**: Die Konfigurationsdatei von npm.
- ✔ **tsconfig.json**: Eine von drei Konfigurationsdateien (siehe Abschnitt *Compiler-Optionen*) für den TypeScript-Compiler. Diese wird von Texteditoren mit TypeScript-Unterstützung benutzt.
- ✔ **tslint.json**: Die globale Konfigurationsdatei von TSLint für das Angular-CLI-Projekt. Mehr über TSLint lesen Sie im Abschnitt *TSLint, codelyzer und Prettier*.

Mehr Informationen über die `.editorconfig`-Datei finden Sie auf der Webseite `https://editorconfig.org/`.

Verzeichnisse und Dateien im `e2e`-Verzeichnis:

- ✔ **src**: In dieses Verzeichnis schreiben Sie den Code für Ihre End-to-End-Tests. Wie das geht, erfahren Sie im Kapitel 12 unter *Tests mit Protractor schreiben*.
- ✔ **protractor.conf.js**: Die Konfigurationsdatei von Protractor. Lesen Sie hierzu speziell den Kasten *Die Konfigurationsdatei von Protractor* in Kapitel 12.
- ✔ **tsconfig.e2e.json**: Die Konfigurationsdatei für den TypeScript-Compiler. Diese wird beim Ausführen der End-to-End-Tests ergänzend zur Datei `tsconfig.json` verwendet.

Verzeichnisse und Dateien im `src`-Verzeichnis:

- ✔ **app**: Das Verzeichnis für den Code der Anwendung. Dort werden Sie die meiste Zeit verbringen. Oft ist der Inhalt dieses Verzeichnisses das Einzige, was Sie verändern müssen, um eine Angular-Anwendung zu schreiben. Den Code für die Beispielanwendung schreiben Sie auch dorthin.
- ✔ **assets**: Ein Verzeichnis für statische Ressourcen. Sie können dort zum Beispiel Bilder und Fonts hinterlegen.
- ✔ **environments**: Ein Verzeichnis mit Konfigurationsdateien für das Bauen der Anwendung (siehe Abschnitt *Das ng-build-Kommando*).
- ✔ **browserslist**: Die Datei wird von *autoprefixer* verwendet. Autoprefixer ist ein Werkzeug, das CSS-Regeln mit speziellen Präfixen so ergänzt, dass diese auch in älteren Browsern funktionieren.
- ✔ **index.html**: Die Hauptdatei der Anwendung. Mehr darüber erfahren Sie im Kapitel 2 unter *Die Hauptdatei der Anwendung*.
- ✔ **karma.conf.js**: Die Konfigurationsdatei von Karma. Mehr darüber erfahren Sie im Kapitel 11 im Kasten *Karma*.
- ✔ **main.ts**: Die Haupt-TypeScript-Datei für die Anwendung. Mehr über diese Datei erfahren Sie im Kapitel 2 unter *Die Hauptdatei der Anwendung*.
- ✔ **polyfills.ts**: Definiert Polyfills, die von Angular gebraucht werden, damit es auch in älteren Browsern funktionieren kann. Sie können dort auch eigene Polyfills definieren. Was Polyfills sind, erkläre ich im Abschnitt *Der TypeScript-Compiler*.
- ✔ **styles.css**: Eine Datei, in der Sie globale CSS-Styles definieren. Standardmäßig ist diese leer.
- ✔ **test.ts**: Die Hauptdatei für Unit-Tests. Mehr darüber erfahren Sie im Kapitel 11 unter *Unit-Tests ausführen*.

- ✔ **tsconfig.app.json**: Konfigurationsdatei für den TypeScript-Compiler. Diese wird beim Bauen der Anwendung ergänzend zur Datei `tsconfig.json` benutzt.
- ✔ **tsconfig.spec.json**: Konfigurationsdatei für den TypeScript-Compiler. Diese wird, beim Ausführen der Unit-Tests, ergänzend zu der `tsconfig.json` benutzt.
- ✔ **tslint.json**: TSLint-Konfigurationsdatei für diese Anwendung. Wird ergänzend zur globalen `tslint.json`-Datei im Hauptverzeichnis benutzt.

Mehr Informationen über die `browserslist`-Datei und ihr Format finden Sie auf `https://github.com/browserslist/browserslist`.

Die Konfigurationsdatei von Angular CLI

Bei der Initialisierung legt Angular CLI eine Datei namens `angular.json` an. Darin befinden sich einige Optionen, die das Verhalten von Angular CLI und seine Kommandos steuern. Mithilfe dieser Optionen können Sie Angular CLI anpassen, sodass es sich verhält, wie Sie es für Ihr Projekt brauchen.

Die Konfigurationsdatei ist in verschiedene Bereiche aufgespalten. Jeder Bereich wird durch eine Eigenschaft und ihren Wert definiert. Beispielsweise gibt es die `version`-Eigenschaft (siehe Listing 1.3), die die Version für das Format des Konfigurationsobjekts definiert. Der interessanteste Bereich ist der Bereich für die Projektkonfigurationen. Auf diesen Bereich werde ich mich hier konzentrieren.

Weitere Bereiche und Optionen der Konfigurationsdatei finden Sie auf `https://github.com/angular/angular-cli/wiki/angular-workspace`.

Im Listing 1.3 sehen Sie eine abgespeckte Version der Konfigurationsdatei. Der Code dort stammt aus der `angular.json`-Datei, die Angular CLI bei mir erzeugt hat, als es das Projekt für die Beispielanwendung initialisiert hat (siehe Abschnitt *Beispielanwendung: Umgebung einrichten*).

```
{
  ...,
  "version": 1,
  "projects": {
    "contacts-manager": {
      ...,
      "schematics": {},
      "architect": {...}
    },
    "contacts-manager-e2e": {...}
  },
}
```

Listing 1.3: Ausschnitt aus der angular.json der Beispielanwendung

Der Bereich für die Projektkonfigurationen wird durch die `projects`-Eigenschaft repräsentiert. Der Wert der Eigenschaft ist ein Objekt und seine Eigenschaften sind Projekte, die zum Angular-CLI-Projekt, zu dem sogenannten *Workspace*, gehören. Sie können beliebig viele Projekte zum Workspace hinzufügen. In diesem Fall repräsentiert die `contacts-manager`-Eigenschaft das Projekt für die Beispielanwendung. Die Eigenschaft `contacts-manager-e2e` repräsentiert das Projekt für die End-to-End-Tests der Beispielanwendung (siehe Kapitel 12). Dieses wird automatisch angelegt.

Wie auch die Konfigurationsdatei von Angular CLI wird auch die Konfiguration eines Projekts in verschiedene Bereiche aufgeteilt. Zwei interessante Bereiche werden durch die `schematics-` und die `architect`-Eigenschaft repräsentiert. Mit der *schematics-Eigenschaft* konfigurieren Sie das Verhalten des `ng generate`-Kommandos. Sie können zum Beispiel angeben, dass bei der Generierung von Komponenten nie eine CSS-Datei angelegt wird (siehe Kasten *Komponenten mit Angular CLI erzeugen* in Kapitel 3). Sie sehen das im Listing 1.4.

```
...,
"schematics": {
  "@schematics/angular": {
    "component": {
      "inlineTemplate": true
    }
  }
},
...
```

Listing 1.4: Konfiguration für das `ng generate`-Kommando. Angular CLI soll keine CSS-Datei bei der Generierung von Komponenten anlegen.

Sie können die `schematics`-Eigenschaft auch auf der gleichen Ebene wie die `projects`-Eigenschaft definieren. Auf diese Weise lässt sich das `ng generate`-Kommando projektübergreifend konfigurieren.

Die meisten Optionen, die Sie dem Kommando beim Aufruf übergeben können, können Sie auch in die Konfigurationsdatei schreiben. Beachten Sie nur, dass die Optionen in der Konfigurationsdatei camelCase angegeben werden. Zum Beispiel heißt es `inlineTemplate` in der Konfigurationsdatei und `--inline-template` beim Aufrufen des Kommandos. Das gilt auch für weitere Kommandos.

Schematics

In Angular CLI sind *Schematics* Code-Generatoren. Sie können Dateien erstellen, ändern und auf dem Dateisystem verschieben. Das `ng generate`-Kommando nutzt Schematics, um Komponenten, Services und so weiter zu generieren. Neben den Schematics, die von Angular CLI zur Verfügung gestellt werden, können Sie auch eigene Schematics erstellen. Sie können zum Beispiel Schematics beim Code-Refactoring einsetzen oder wenn Sie immer wieder sehr ähnliche Angular-Bausteine definieren müssen.

Die *architect-Eigenschaft* nutzen Sie für die Konfiguration von weiteren Angular-CLI-Kommandos wie zum Beispiel `ng serve` und `ng build` (siehe Abschnitt *Die Anwendung bauen*). Der Wert der Eigenschaft ist ein Objekt und jede Eigenschaft im Objekt ist der Name eines Angular-CLI-Kommandos. Für jedes Kommando gibt es eine `builder`-, eine `options`- und eine `configurations`-Eigenschaft. Ein Beispiel für das `ng build`-Kommando sehen Sie im Listing 1.5.

```
...,
"architect": {
  "build": {
    "builder": "@angular-devkit/build-angular:browser",
    "options": {...},
    "configurations": {
      "production": {...}
    }
  },
},
...
```

Listing 1.5: Ausschnitt aus der Konfiguration für das `ng build`-Kommando

Die `builder`-Eigenschaft gibt an, welches Kommando aufgerufen wird, wenn Sie `ng build` aufrufen. Der Teil vor dem Doppelpunkt – hier `@angular-devkit/build-angular` – ist der Name des npm-Pakets für den Builder. Der Teil nach dem Doppelpunkt – hier `browser` – ist der tatsächliche Name des Builders.

Im `options`-Objekt befinden sich die Optionen für das Kommando. Welche Optionen Ihnen zur Verfügung stehen, hängt vom jeweiligen Builder ab. Das `configurations`-Objekt wird verwendet, um die Standardoptionen für spezielle Fälle zu überschreiben oder zu ergänzen. Im Listing 1.5 befindet sich dort die `production`-Eigenschaft. Diese beinhaltet die Optionen für das Bauen einer Produktivversion Ihrer Anwendung (siehe Abschnitt *Die Anwendung bauen*). Sie können dort auch eigene Eigenschaften mit ihren eigenen Optionen definieren. Beim Aufrufen des `ng build`-Kommandos (siehe Abschnitt *Das ng-build-Kommando*) können Sie dann angeben, welche Eigenschaft benutzt werden soll beim Bauen der Anwendung.

In der Erklärung der Kommandos gehe ich immer davon aus, dass Sie die Standardkonfiguration verwenden. Falls Sie diese ändern, kann es passieren, dass meine Erklärungen nicht mehr 100 % mit den Resultaten, die Sie sehen, übereinstimmen. Das gilt auch, wenn Sie eine andere Version von Angular CLI verwenden.

Die Anwendung bauen

»Die Anwendung bauen« heißt nichts anderes, als einen TypeScript-Code in JavaScript-Code umzuwandeln. Im Hintergrund passiert zwar noch mehr, aber diese Definition reicht für das grobe Verständnis. Angular CLI hat zwei Kommandos, mit denen Sie Ihre Anwendung bauen können: das `ng serve`- und das `ng build`-Kommando. Diese erkläre ich jetzt.

Das ng-build-Kommando

Wenn Sie in einer Konsole `ng build` aufrufen, wird die Anwendung gebaut und das Ergebnis des Build-Prozesses in das `dist`-Verzeichnis geschrieben. Hauptsächlich verwenden Sie dieses Kommando, wenn Sie schon einen Webserver haben und wollen, dass dieser die Anwendung ausliefert. Standardmäßig wird Ihr gesamter TypeScript-Code in eine Datei geschrieben. Diese Datei nennt man ein *Code-Bündel*. Wenn Sie die Anwendung im Browser laden, wird das Code-Bündel vom Server angefordert und nicht die einzelnen Dateien, die Sie bei der Entwicklung definiert haben. Dieses Standardverhalten können Sie beeinflussen, indem Sie Angular-Module nachladen (siehe Kapitel 10, Abschnitt *Angular-Module nachladen*). Jedes nachgeladene Angular-Modul besitzt sein eigenes Code-Bündel.

Das Kommando können Sie nur innerhalb eines Verzeichnisses mit einem Angular-CLI-Projekt aufrufen. Wenn Sie es in einem anderen Verzeichnis aufrufen, gibt es eine Fehlermeldung.

Optionen für das Kommando `ng build`:

- ✔ **`--configuration`**: Mit dieser Option geben Sie an, welche Eigenschaft aus dem configurations-Objekt (siehe Abschnitt *Die Konfigurationsdatei von Angular CLI*) ausgelesen werden soll. Das definiert dann die Optionen, die beim Bauen der Anwendung verwendet werden. Die Eigenschaft des configurations-Objekts definiert auch, welche environments-Datei aus dem `environments`-Verzeichnis verwedet wird. Wenn Sie die Option nicht verwenden, wird die `environment.ts` gelesen, mit zum Beispiel `... --configuration=production ...` dagegen die Datei `environment.prod.ts`.
- ✔ **`--aot`**: Wenn Sie `... --aot ...` verwenden, wird der Ahead-of-Time-Angular-Compiler verwendet (siehe Kasten *Angular-Compiler*). Diese Option wird automatisch auf `true` gesetzt, wenn Sie die configuration-Option mit `production` verwenden.

Das Kommando `ng build --prod` verwenden Sie, um eine Produktivversion Ihrer Anwendung zu bauen. Es ist äquivalent zu `ng build --configuration=production`.

Das Kommando `ng build --help` zeigt Ihnen weitere mögliche Optionen und was diese tun.

Das ng-serve-Kommando

Wenn Sie in der Konsole `ng serve` aufrufen, wird die Anwendung gebaut und ein Webserver wird gestartet, der die Anwendung ausliefert. Dieses Kommando nutzen Sie bei der Entwicklung der Anwendung. Der erzeugte Code wird nicht auf die Festplatte geschrieben. Dieser befindet sich im Arbeitsspeicher. Jedes Mal, wenn Sie Änderungen im Code vornehmen, wird die Anwendung automatisch neu gebaut und das Browserfenster neu geladen. Für das Neuladen ist eine Websocket-Verbindung zwischen den Browser und den Webserver verantwortlich. Kompilierfehler werden in der Konsole angezeigt. Sie sollten also während der Entwicklung auch immer wieder einen Blick in die Konsole werfen – vor allem, wenn Ihnen Kompilierfehler nicht im Texteditor angezeigt werden.

Browserunterstützung

Offiziell werden derzeit folgende Browser unterstützt: Chrome, Firefox, Internet Explorer, Internet Explorer Mobile, Edge, Safari (Mac OS und iOS) und der Android-Browser. »Offiziell« heißt in diesem Zusammenhang, dass die Bibliothek auf diesen Browsern getestet wird. Bei Chrome und Firefox wird nur die neuste Version offiziell unterstützt, allerdings funktioniert Angular auch mit älteren Versionen. Bei Safari und Edge werden die zwei neusten Versionen unterstützt. Internet Explorer wird ab Version 9 und Internet Explorer Mobile ab Version 11 unterstützt. Der Android-Browser wird ab Version 4.4 unterstützt.

Allerdings werden in Angular viele neue Features verwendet, aus diesem Grund funktioniert nicht jeder unterstützte Browser ohne Weiteres. Bei einigen Browsern, vor allem älteren, müssen Sie gegebenenfalls Polyfills aktivieren. Das können Sie in der `polyfills.ts`-Datei tun (siehe Abschnitt *Die Verzeichnisstruktur von Angular CLI*). Für manche Angular-Features müssen Sie auch bei modernen Browsern Polyfills aktivieren.

Angular ist genau wie das Web sehr schnelllebig. Daher sollten Sie sich die aktuelle Browserunterstützung auf `https://angular.io/guide/browser-support` anschauen, bevor Sie ein Projekt mit Angular starten.

Der Webserver von Angular CLI ist nur für die Entwicklung gedacht. Dieser sollte nicht über das Internet erreichbar sein.

Standardmäßig können Sie im Browser die URL `http://localhost:4200` verwenden, um Ihre Anwendung anzuschauen.

Neben allen Optionen vom `ng build`-Kommando unterstützt das `ng server`-Kommando weitere Optionen. Ein paar davon sehen Sie hier:

- **`--port`**: Damit können Sie einen anderen Port für den Webserver von Angular CLI angeben, zum Beispiel `... --port=8080 ...`
- **`--ssl`**: Mit `... --ssl=true ...` ist Ihre Anwendung über `https` erreichbar. Ein Zertifikat wird von Angular CLI bereitgestellt.
- **`--ssl-key`**: Damit geben Sie den SSL-Schlüssel für die ssl-Option an.
- **`--ssl-cert`**: Damit geben Sie das SSL-Zertifikat für die ssl-Option an.

Wenn Sie das SSL-Zertifikat von Angular CLI verwenden, werden Sie in Ihrem Browser vermutlich eine Warnung sehen. Das ist normal und braucht Sie nicht zu beunruhigen.

Das Kommando `ng serve --help` zeigt Ihnen weitere mögliche Optionen und was diese tun.

Die Beispielanwendung tritt auf

Das Beispiel, an dem ich in diesem Buch die verschiedenen Features und Aspekte von Angular vorstelle, ist eine Webanwendung für die Verwaltung von Kontakten. In jedem Kapitel werden Sie diese Beispielanwendung weiterentwickeln und anpassen. Sie werden damit Angular kennenlernen und sehen, wie Sie mit diesem Framework Webanwendungen bauen können.

Funktionalität der Anwendung:

- ✔ Unterstützung von einer oder mehreren Kontaktlisten, zum Beispiel eine Liste für private und eine für geschäftliche Kontakten. Die Liste von Kontaktlisten sehen Sie in der Abbildung 1.2. Das ist die Hauptseite der Anwendung.
- ✔ Jede Kontaktliste kann null oder mehr Kontakte beinhalten. In der Abbildung 1.3 sehen Sie die Kontakte für die ausgewählte Kontaktliste.
- ✔ Die Anwendung erlaubt das Hinzufügen, Bearbeiten und Löschen von Kontaktlisten und Kontakten. In der Abbildung 1.4 sehen Sie das Formular, mit dem der Benutzer Kontakte hinzufügen kann. Das gleiche Formular wird auch verwendet, um die Daten eines Kontakts zu ändern.
- ✔ Sie hat eine rudimentäre Login-Funktionalität. In der Abbildung 1.5 sehen Sie das Login-Formular.

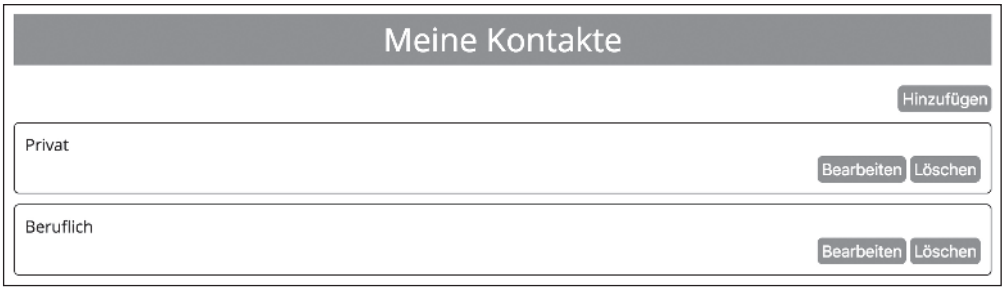

Abbildung 1.2: Die Hauptseite der fertigen Beispielanwendung zeigt die Kontaktlisten an.

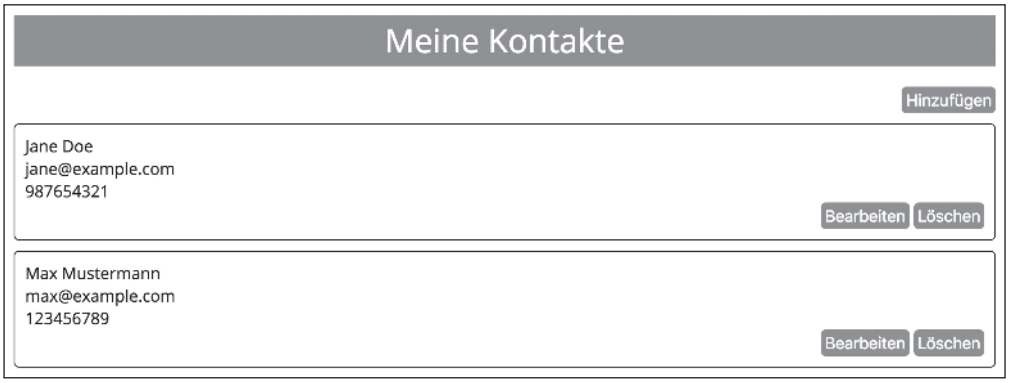

Abbildung 1.3: Liste von Kontakten für die Kontaktliste »Privat«

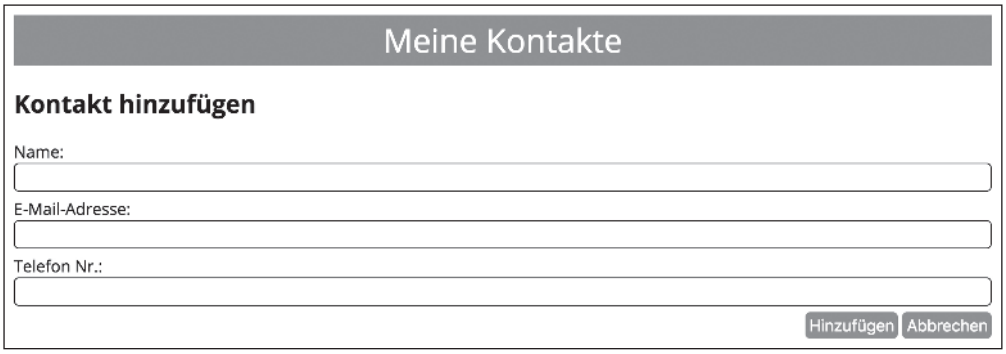

Abbildung 1.4: Formular, um Kontakte hinzuzufügen

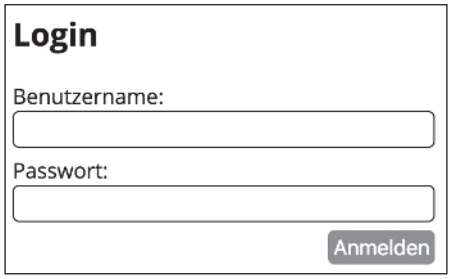

Abbildung 1.5: Login-Formular für die Beispielanwendung

Die Beispielanwendung ist nicht sehr umfangreich, enthält aber alles, was Sie brauchen, um ein Basiswissen und -verständnis von Angular zu erlangen. Natürlich steht es Ihnen frei, weitere Features zu implementieren. Das Aussehen Ihrer Anwendung definieren Sie. Die Abbildungen sind nur ein Beispiel, wie solch eine Anwendung aussehen könnte.

Beispielanwendung: Umgebung einrichten

In diesem Abschnitt richten Sie Ihre Entwicklungsumgebung ein und nutzen Angular CLI, um ein neues Projekt anzulegen, das dann als Basis für die weiteren Kapitel dient. Sie werden es nach und nach weiterentwickeln und am Ende haben Sie dann die Anwendung vor sich, die ich gerade im Abschnitt *Die Beispielanwendung tritt auf* skizziert habe.

Ich gehe hier davon aus, dass Sie schon einen Texteditor haben. Falls nicht, finden Sie im Abschnitt *Texteditoren* ein paar Vorschläge.

1. Installieren Sie Node.js und npm.

   Sie können Node.js von der offiziellen Node.js-Webseite herunterladen unter `https://nodejs.org/de/download/current`. Npm wird bei der Installation von Node.js mitinstalliert.

Installieren Sie eine möglichst neue Version von Node.js, am besten mindestens Version 10.x.x. Manche ältere Versionen funktionieren nicht mit Angular CLI.

2. Führen Sie `node --version` und `npm --version` in einer Konsole aus.

   Damit testen Sie, ob Node.js und npm erfolgreich installiert sind. Falls alles geklappt hat, wird Ihnen die Version für das jeweilige Werkzeug angezeigt.

3. Installieren Sie Angular CLI mit `npm install -g @angular/cli@7.0.3`.

   Die Option `-g` von npm bewirkt, dass das Werkzeug global wird. Nur wenn Angular CLI global installiert ist, können Sie es in der Konsole über `ng` aufrufen.

Mit `@x.x.x` nach einem npm-Paketnamen installieren Sie die Version angegeben durch `x.x.x`, hier Version `7.0.3` von `@angular/cli`. Wenn Sie die Version nicht explizit angeben, wird einfach die neuste Version des Pakets installiert.

4. Überprüfen Sie mit `ng --version`, ob die Installation erfolgreich war.

5. Initialisieren Sie das Projekt für die Beispielanwendung (siehe Abschnitt *Projekt initialisieren*). Nutzen Sie dafür das Kommando `ng new --interactive=false contacts-manager`.

   Die Initialisierung kann einige Minuten dauern. Zeit für eine kleine Kaffeepause.

Wenn Sie schon ein initialisiertes Angular-CLI-Projekt haben, reicht es, wenn Sie mittels `npm install` seine Abhängigkeiten installieren. Das Kommando führen Sie im Verzeichnis des Projekts aus.

6. Navigieren Sie über die Konsole in das `contacts-manager`-Verzeichnis.
7. Passen Sie die `tsconfig.json`-Datei an und fügen Sie die Optionen aus dem Abschnitt *Compiler-Optionen* hinzu.
8. Starten Sie, mit `ng serve`, den Webserver von Angular CLI.
9. Schauen Sie sich, über `http://localhost:4200`, die Anwendung im Browser an. Sie sollten das gleiche sehen wie in der Abbildung 1.6.

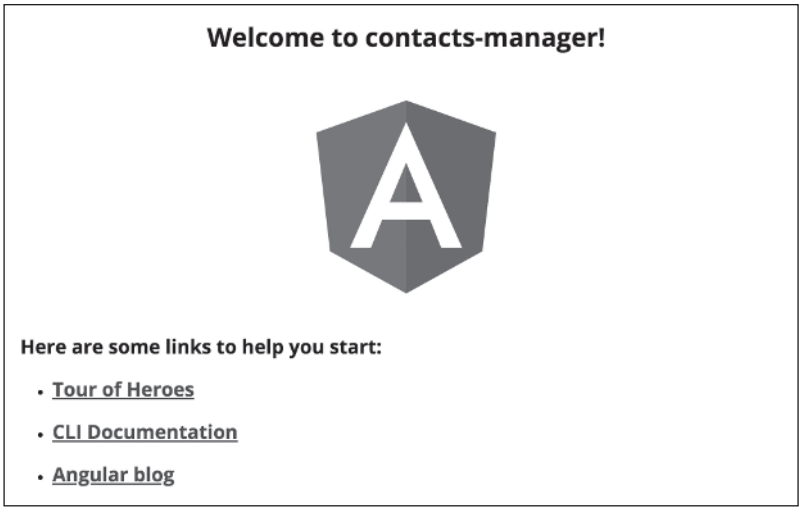

Abbildung 1.6: Screenshot von einem frisch initialisiertem Angular-CLI-Projekt

10. Führen Sie irgendeine Änderung im Code aus. Ändern Sie zum Beispiel den HTML-Code der `src/app/app.component.html`.

    Die Änderung sollte im Browser sichtbar sein, ohne dass Sie die Seite neu laden müssen.

Solange der Webserver von Angular CLI in der Konsole läuft, können Sie in diesem Fenster keine weiteren Kommandos ausführen. Sie können den Webserver mit Strg+C anhalten.

11. Probieren Sie auch die Kommandos `ng lint` und `ng build` aus. Ändern Sie den Code und versuchen Sie, Fehler zu verursachen. Führen Sie die Kommandos aus und schauen Sie sich die Fehlermeldungen in der Konsole an.

Ein für dieses Buch initialisiertes Angular-CLI-Projekt finden Sie unter `beispielanwendung/contacts-manager-01`.

Alle Code-Beispiele und Lösungsvorschläge sind Angular-CLI-Projekte. Damit Sie diese ausführen und im Browser anschauen können, müssen Sie ihre Abhängigkeiten installieren. Dies machen Sie mittels `npm install`. Sie müssen das Kommando in jedem Verzeichnis mit einer `package.json`-Datei ausführen.

Wenn Sie auf die Schnelle etwas ausprobieren wollen, können Sie dafür `https://stackblitz.com` verwenden. Das ist ein Online-Editor für Webanwendungen und Sie können dort mit wenig Aufwand ein neues Angular-CLI-Projekt initialisieren.

Die Angular-Version aktualisieren

Bei jedem Angular-Projekt kommt die Zeit, wo Sie die Angular-Version aktualisieren wollen oder müssen. Wenn Sie in einem Projekt schon mal die Versionen Ihrer Abhängigkeiten aktualisiert haben, wissen Sie, dass dieses Unterfangen Zeit kostet und schwierig sein kann. Das Angular-Team lässt Sie aber nicht im Stich. Mithilfe von Angular CLI können Sie die Versionen Ihrer Abhängigkeiten aktualisieren. Das tun Sie mit `ng update`. Falls nötig, versucht dieses Kommando auch Konfigurationsdateien anzupassen, die Angular CLI angelegt hat.

Das ist allerdings nur die halbe Miete. Vor allem bei größeren Versionssprüngen wie zum Beispiel von Version 4.0.0 auf Version 6.0.0 müssen Sie auch Ihren Code anpassen, damit weiterhin alles funktioniert. Auch dafür hat das Angular-Team eine Lösung parat: Auf `https://update.angular.io` geben Sie Ihre aktuelle und die neue Angular-Version an. Die Seite gibt Ihnen Tipps, wie Sie Ihren Code anpassen müssen, sodass dieser auch mit der neuen Angular-Version reibungslos funktioniert. Sie müssen nicht immer alles, was dort steht, umsetzen und manchmal müssen Sie Anpassungen machen, die nicht dort beschrieben sind. Was nötig ist, jeweils hängt von Ihrem Projekt ab.