O'REILLY®

3. Auflage

Angular

Das Praxisbuch zu Grundlagen und Best Practices

Manfred Steyer

3. AUFLAGE

Angular

Das Praxisbuch zu Grundlagen und Best Practices

Manfred Steyer

O'REILLY®

Manfred Steyer

Lektorat: Ariane Hesse
Fachliche Unterstützung: Hans-Peter Grahsl
Korrektorat: Sibylle Feldmann, *www.richtiger-text.de*
Satz: III-satz, *www.drei-satz.de*
Herstellung: Stefanie Weidner
Umschlaggestaltung: Michael Oréal, *www.oreal.de*
Druck und Bindung: mediaprint solutions GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

ISBN:
Print 978-3-96009-166-0
PDF 978-3-96010-576-3
ePub 978-3-96010-577-0
mobi 978-3-96010-578-7

3. Auflage 2021

Wieblinger Weg 17
69123 Heidelberg

Dieses Buch erscheint in Kooperation mit O'Reilly Media, Inc. unter dem Imprint »O'REILLY«.
O'REILLY ist ein Markenzeichen und eine eingetragene Marke von O'Reilly Media, Inc. und wird mit Einwilligung des Eigentümers verwendet.

Hinweis:
Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: *kommentar@oreilly.de*.

5 4 3 2 1 0

Inhalt

Vorwort

Vor etwas mehr als zehn Jahren galt für gute Webanwendungen noch die Regel, so viele Aufgaben wie möglich auf dem Server zu erledigen. Inzwischen stützen sich moderne Webanwendungen jedoch auf clientseitige Techniken, allen voran JavaScript. Dies steigert die Benutzerfreundlichkeit und schafft die Möglichkeit, die jeweilige Anwendung an die Auflösungen und Formfaktoren der vielen unterschiedlichen klassischen und mobilen Plattformen anzupassen.

Single Page Applications (SPAs) bilden einen derzeit äußerst beliebten Architekturstil für solche Webanwendungen. Wie ihr Name schon vermuten lässt, bestehen SPAs aus lediglich einer einzigen Seite, die ein Browser auf klassischem Weg abruft und anzeigt. Alle weiteren Seiten und Daten bezieht die SPA bei Bedarf über direkte HTTP-Zugriffe per JavaScript.

Das populäre JavaScript-Framework Angular, das von Google entwickelt wird, hilft Ihnen beim Erstellen von Anwendungen, die diesen Architekturstil verfolgen. Angular bietet unter anderem Unterstützung für die Datenbindung, für das Validieren von Daten sowie das Arbeiten mit Vorlagen. Darüber hinaus stellt Angular Konzepte zur Verfügung, mit denen Sie den Quellcode strukturieren und wartbare, wiederverwendbare sowie testbare Programmteile erschaffen können. Das vorliegende Buch präsentiert die Möglichkeiten von Angular. Dabei beschränkt es sich nicht nur auf die Grundlagen, sondern geht auch auf die zugrunde liegenden Konzepte ein.

Zielgruppe

Das Buch richtet sich an Entwickler, die bereits grundlegende Erfahrungen mit HTML, CSS und JavaScript gesammelt haben und nun mit Angular SPAs entwickeln wollen. Dabei bezieht es sich auf die Version 12 von Angular, die beim Verfassen der Texte die aktuellste Version war. Aufgrund des evolutionären Charakters von Angular gehen wir jedoch davon aus, dass dieses Buch auch für viele darauffolgende Versionen geeignet ist.

Zielsetzung des Buchs

Mit diesem Buch verfolgen wir das Ziel, Ihnen anhand von Beispielen zu zeigen, wie Sie Angular zur Entwicklung von Single Page Applications nutzen können. Dabei gehen wir auf die Möglichkeiten von Angular ein und präsentieren auch Lösungen für Aspekte, die Angular nicht direkt unterstützt.

Quellcodebeispiele, Onlineservices und Errata

Über *http://www.ANGULARarchitects.io/leser* stellen wir Ihnen sämtliche in diesem Buch präsentierten Quellcodebeispiele sowie Web-APIs zur Verfügung, die die Beispiele zu Demonstrationszwecken nutzen und die Sie auch in eigene Projekte einbinden können. Darüber hinaus werde ich auf dieser Seite weitere Informationen zu Angular sowie gegebenenfalls Errata zum vorliegenden Buch veröffentlichen. Daneben bietet die Website Ihnen die Möglichkeit, mit mir als Autor direkt in Kontakt zu kommen.

Konventionen

Kursiv
: Wird genutzt für neue Begriffe, URLs, Dateinamen und E-Mail-Adressen

`Nichtproportionalschrift`
: Programmlistings und Codeelemente im Fließtext wie Methoden, Module o. Ä. werden in dieser Schrift dargestellt.

Dieses Symbol steht für Hinweise und allgemeinere Anmerkungen.

Dieses Symbol steht für Tipps.

Aufbau des Buchs

Das Buch besteht aus 19 Kapiteln. Die folgende Auflistung zeigt, was diese bieten:

- Kapitel 1, *Projekt-Setup*: In diesem Kapitel erfahren Sie, welche Schritte nötig sind, um mit Angular loszulegen. Dazu lernen Sie unter anderem das *Angular Command Line Interface* (CLI) kennen. Damit generieren wir auch schon das Grundgerüst für unsere erste Angular-Anwendung.

- Kapitel 2, *Erste Schritte mit TypeScript*: Die bevorzugte Sprache zum Entwickeln mit Angular ist TypeScript. Sie orientiert sich am ECMAScript-Standard und bietet zusätzlich ein statisches Typsystem, um Fehler frühzeitig erkennen zu können. In diesem Kapitel lernen Sie die wichtigsten Merkmale dieser JavaScript-Erweiterung kennen.
- Kapitel 3, *Eine erste Angular-Anwendung*: Dieses Kapitel zeigt Ihnen, wie Sie mit Angular eine erste einfache Anwendung erstellen können, die via HTTP Daten abruft und darstellt. Darüber hinaus lernen Sie hier die Grundlagen zu Komponenten und Modulen kennen. Auch in das Thema Datenbindung wird eingeführt.
- Kapitel 4, *Komponenten und Datenbindung*: Angular-Anwendungen sind Komponenten, die aus weiteren Komponenten bestehen. In diesem Kapitel erfahren Sie, wie Sie eigene Komponenten erstellen können, die über Datenbindung mit anderen Komponenten kommunizieren.
- Kapitel 5, *Services und Dependency Injection*: Services sind unter Angular wiederverwendbare Codeeinheiten. Dank Dependency Injection können Sie sie austauschbar gestalten. Dieses Kapitel zeigt die Möglichkeiten zum Entwickeln solcher Services auf. Außerdem erfahren Sie hier, wie Sie Services global oder auch nur für bestimmte Komponenten registrieren können.
- Kapitel 6, *Pipes*: Pipes helfen dabei, Daten im Rahmen der Datenbindung zu transformieren. Sie kommen zum Beispiel zum Formatieren oder Umschlüsseln von Informationen zum Einsatz. Dieses Kapitel informiert über die in Angular enthaltenen Pipes und erklärt, wie Sie in Ihren Projekten eigene Pipes schreiben können.
- Kapitel 7, *Module*: Zum Strukturieren von Anwendungen setzt Angular auf Module. In diesem Kapitel lernen Sie eine typische Modulstruktur für Angular-Anwendungen kennen. Außerdem erfahren Sie, wie Sie eigene Module umsetzen können.
- Kapitel 8, *Routing*: Mit Routing lassen sich in einer Single Page Application mehrere Seiten simulieren und so Navigationsstrukturen etablieren. Dieses Kapitel geht auf den Angular-Router ein, der sich um diese Aufgabe kümmert.
- Kapitel 9, *Template-getriebene Formulare und Validierung*: In diesem Kapitel lernen Sie, Template-getriebene Formulare mit Angular aufzubauen. Außerdem erfahren Sie, wie Sie Eingaben validieren und eigene Formularsteuerelemente anbieten können.
- Kapitel 10, *Reaktive Formulare*: Reaktive Formulare bilden eine sehr mächtige Alternative zu den im vorangegangenen Kapitel präsentierten Template-getriebenen Formularen. Hier erfahren Sie, was es damit auf sich hat.
- Kapitel 11, *Reactive Extensions Library for JavaScript (RxJS)*: Angular nutzt die von der Bibliothek *RxJS* angebotenen Observables zur Darstellung asynchroner Operationen. In diesem Kapitel werfen wir einen etwas genaueren Blick auf

RxJS und die zugrunde liegenden Konzepte. Wir machen uns mit den verschiedenen Gruppen von Operatoren vertraut und lernen populäre Vertreter dieser Gruppen kennen. Außerdem beschäftigen wir uns mit den Schritten des reaktiven Entwurfs, mit Hot und Cold Observables bzw. Multicasting sowie mit Fehlerbehandlung, Flattening und dem Kombinieren verschiedener Datenflüsse.

- Kapitel 12, *Testautomatisierung*: Wie die Qualität von Angular-Code mit automatisierten Tests geprüft werden kann, erfahren Sie in diesem Kapitel. Wir gehen dazu auf das populäre Testing-Framework Jasmine in Kombination mit den Angular Testing Utilities ein und zeigen Ihnen, wie Sie damit Tests für Ihren Angular-Code schreiben können. Begriffe und Techniken wie Mocks und Spys werden ebenfalls thematisiert.
- Kapitel 13, *Performancetuning*: In diesem Kapitel betrachten wir zwei elementare Stellschrauben, die die Performance von Angular-Anwendungen beeinflussen, und lernen dabei auch die Art und Weise, wie das Framework arbeitet, besser kennen. Zum einen betrachten wir die Datenbindungsstrategie OnPush und nutzen sie gemeinsam mit Observables und Immutables, zum anderen beschäftigen wir uns mit Lazy-Loading- und Preloading-Strategien.
- Kapitel 14, *Querschnittsfunktionen*: Querschnittsfunktionen sind meist technische Anforderungen, die es immer und immer wieder zu berücksichtigen gilt. Beispiele dafür sind unter anderem Authentifizierung, Protokollierung und die Behandlung von Fehlern. In diesem Kapitel beschäftigen wir uns mit Mechanismen zur Implementierung solcher Querschnittsfunktionen, unter anderem mit Guards, Resolver und HttpInterceptoren.
- Kapitel 15, *Authentifizierung und Autorisierung*: Die wenigsten Anwendungen kommen ohne Authentifizierung und Autorisierung aus. Bei Single Page Applications bieten sich hierzu neben Cookies auch tokenbasierte Verfahren an. Dieses Kapitel beschreibt beide Optionen. Für Letztere geht es auf die populären Standards OAuth 2 und OpenID Connect ein und zeigt, wie Sie damit ein Security-Token anfordern können. Anschließend erfahren Sie, wie sich dieses Vorgehen in Ihren Angular-Anwendungen umsetzen lässt.
- Kapitel 16, *Internationalisierung*: In diesem Kapitel zeigen wir Ihnen, wie das Anpassen von Angular-Anwendungen für Benutzer verschiedener Länder und Sprachen funktioniert. Die dafür in den Angular-Compiler integrierten Möglichkeiten betrachten wir dabei genauso wie den Einsatz der populären Bibliothek *ngx-translate*.
- Kapitel 17, *Reaktive Zustandsverwaltung mit NGRX (Redux)*: State Management hilft beim Verwalten lokal vorgehaltener Daten, auf die unter anderem mehrere Komponenten zugreifen. Die wohl populärste Bibliothek, die dabei unterstützt, ist *NGRX*. In diesem Kapitel erfahren Sie, was sich dahinter verbirgt und wie Sie diese Bibliothek in Ihrer Angular-Anwendung nutzen können.

- Kapitel 18, *Details zu Komponenten und Direktiven*: Obwohl seit der Angular-Einführung in Kapitel 3 ständig Komponenten zum Einsatz kommen, gibt es doch noch einige Details, die unter anderem bei der Entwicklung wiederverwendbarer Komponenten nützlich sind. Diese lernen Sie hier kennen. Außerdem erfahren Sie, wie sich mit Direktiven wiederkehrende Aufgaben umsetzen lassen.
- Kapitel 19, *Wiederverwendbare Bibliotheken und Monorepos*: In diesem Kapitel erfahren Sie, wie sich Ihre Angular-Anwendungen mittels Bibliotheken in einem mit der Angular CLI generierten Monorepo untergliedern lassen, aber auch, wie sie diese Bibliotheken als wiederverwendbare npm-Pakete für andere Projektteams veröffentlichen können.

Schulungen und Beratung

Der Autor bietet Schulungen und Beratung zu Angular an. Informationen dazu finden Sie unter *http://www.ANGULARarchitects.io*.

Danksagungen

Meinen Dank für ihre Mitwirkung an diesem Buch möchte ich aussprechen an

- Ariane Hesse, Lektorat, und Sibylle Feldmann, Korrektorat, die das Manuskript bearbeitet haben.
- Hans-Peter Grahsl, der sehr sorgfältig und unter vollstem Einsatz seines berüchtigten Adlerblicks äußerst wertvolles inhaltliches sowie fachliches Feedback zu den Texten und den Beispielen gegeben hat.

KAPITEL 1

Projekt-Setup

Moderne JavaScript-Projekte gleichen immer mehr klassischen Anwendungen: Sie nutzen Compiler, um moderne typsichere Sprachen wie TypeScript in handelsübliches JavaScript zu übersetzen. Zusätzlich verwenden sie Werkzeuge, mit denen sie optimierte Bundles erzeugen. Damit sind JavaScript-Dateien gemeint, die sich aus mehreren einzelnen Dateien zusammensetzen.

Durch dieses Vorgehen müssen nur noch wenige Dateien auf dem Server platziert sowie in den Browser geladen werden. Ersteres erhöht den Komfort beim Deployment, und Letzteres verbessert die Startgeschwindigkeit der Anwendung. Außerdem kommen in modernen JavaScript-Projekten auch Werkzeuge zur Testautomatisierung zum Einsatz.

Dieses Kapitel zeigt, wie sich ein Projekt-Setup für Angular, das diesen Kriterien genügt, einrichten lässt. Es beginnt mit der Installation und Einrichtung von Visual Studio Code, der in diesem Buch verwendeten Entwicklungsumgebung. Danach wendet sich das Kapitel dem *Angular Command Line Interface* (CLI) zu. Dabei handelt es sich um eine vom Angular-Team bereitgestellte Konsolenanwendung, die viele Aufgaben rund um die Angular-Entwicklung automatisiert. Schließlich erfahren Sie in diesem Kapitel auch, wie ein mit der CLI generiertes Projekt aufgebaut ist.

Visual Studio Code

Wir nutzen in diesem Buch die freie Entwicklungsumgebung Visual Studio Code (*https://code.visualstudio.com*). Sie funktioniert auf allen wichtigen Betriebssystemen (Linux, macOS, Windows) und ist äußerst leichtgewichtig. Visual Studio Code unterstützt auch die Sprache TypeScript. Bei dieser handelt es sich um eine typsichere Obermenge von JavaScript, die für die Angular-Entwicklung verwendet wird.

Außerdem existieren zahlreiche Erweiterungen, die die Arbeit mit Frameworks wie Angular vereinfachen. Um Erweiterungen zu installieren, klicken Sie auf das Symbol *Extensions* in der linken Symbolleiste. Anschließend können Sie nach Erweiterungen suchen und diese installieren (siehe Abbildung 1-1).

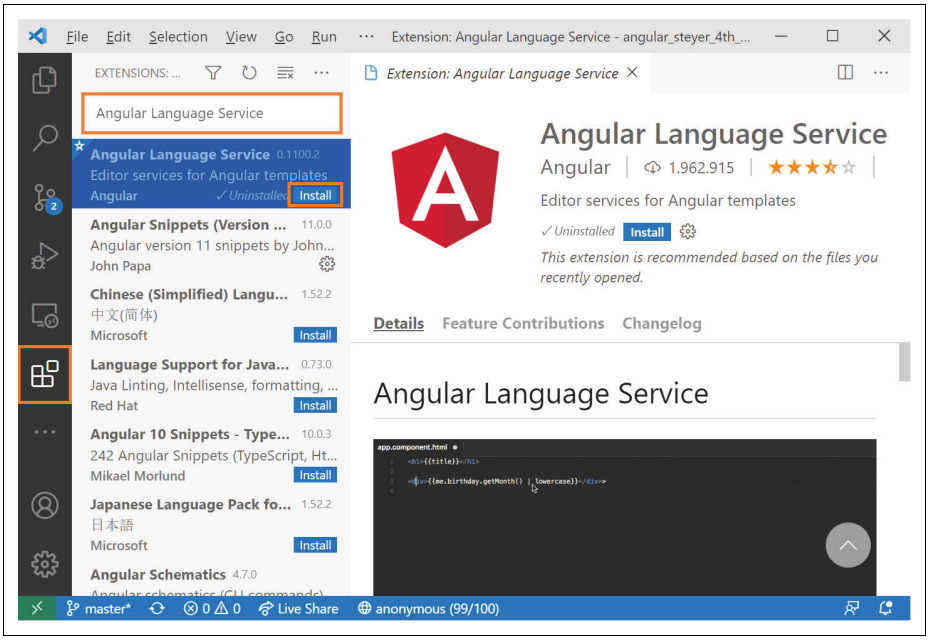

Abbildung 1-1: Erweiterungen in Visual Studio Code installieren

Für die Entwicklung von Angular-Lösungen empfehlen wir die folgenden Erweiterungen:

Angular Language Service
: Der Angular Language Service wird vom Angular-Team bereitgestellt und erlaubt Angular-bezogene Codevervollständigungen in HTML-Templates. Außerdem weist der Language Service auch auf mögliche Fehler in HTML-Templates hin.

Angular Schematics
: Erlaubt das Generieren von Building-Blocks wie Angular-Komponenten über das Kontextmenü von Visual Studio Code.

Debugger for Chrome
: Erlaubt das Debuggen von JavaScript-Anwendungen, die in Chrome ausgeführt werden.

Bitte installieren Sie diese Erweiterungen. Wir werden bei Bedarf in den einzelnen Kapiteln darauf zurückkommen.

Neben Visual Studio Code haben wir auch mit den kommerziellen Produkten WebStorm, PhpStorm bzw. IntelliJ von Jetbrains (*https://www.jetbrains.com/*) sehr gute Erfahrungen gemacht. Es handelt sich bei diesen drei Lösungen eigentlich um das gleiche Produkt in verschiedenen Ausprägungen. PhpStorm unterstützt zum Beispiel darüber hinaus PHP, während IntelliJ zusätzlich viele Annehmlichkei-

ten für Java-Entwickler mit sich bringt. Diese Lösungen sind zwar etwas schwergewichtiger als Visual Studio Code, bieten dafür jedoch ab Werk zahlreiche Features, wie umfangreiche Refactoring-Möglichkeiten oder Test-Runner für Unit-Tests.

Tatsächlich sind Visual Studio Code und WebStorm/IntelliJ mit Abstand die am häufigsten eingesetzten Entwicklungsumgebungen, auf die wir bei unseren Kundenprojekten im Angular-Umfeld stoßen.

Angular CLI

Um keine Zeit mit dem Einrichten aller benötigten Werkzeuge zu verlieren, bietet das Angular-Team das sogenannte *Angular Commandline Interface*, kurz Angular CLI (Angular CLI (*https://cli.angular.io*)), an. Die CLI generiert nicht nur das Grundgerüst der Anwendung, sondern auf Wunsch auch die Grundgerüste weiterer Anwendungsbestandteile wie z. B. Komponenten.

Außerdem kümmert sie sich um das Konfigurieren des TypeScript-Compilers und einer Build-Konfiguration zur Erzeugung optimierter Bundles. Werkzeuge für die Testautomatisierung richtet die CLI ebenfalls ein.

Node.js und Angular CLI installieren

Die CLI lässt sich leicht über den Package-Manager `npm` beziehen, der sich im Lieferumfang von Node.js (*nodejs.org*) befindet. Außerdem nutzt sie Node.js als Laufzeitumgebung. Deswegen sollten Sie zur Vorbereitung eine aktuelle Node.js-Version von Node.js (*https://nodejs.org*) herunterladen und installieren. Die Autoren haben gute Erfahrungen mit den jeweiligen Long-Term-Support-Versionen (LTS-Versionen) gemacht. Der Einsatz älterer Versionen kann zu Problemen führen.

Sobald Node.js installiert ist, kann die CLI mittels `npm` eingerichtet werden:

```
npm install -g @angular/cli
```

Der Schalter `-g` bewirkt, dass `npm` das Werkzeug systemweit, also global, einrichtet, sodass es überall zur Verfügung steht. Ohne diesen Schalter würde `npm` das adressierte Paket lediglich für ein lokales Projekt im aktuellen Ordner einrichten. Nach der Installation steht die CLI über das Kommando `ng` zur Verfügung.

Ein Projekt mit der CLI generieren

Ein Aufruf von

```
ng new flight-app
```

generiert das Grundgerüst einer neuen Angular-Anwendung, die den Namen *flight-app* erhält. Dazu stellt es uns ein paar Fragen (siehe Abbildung 1-2):

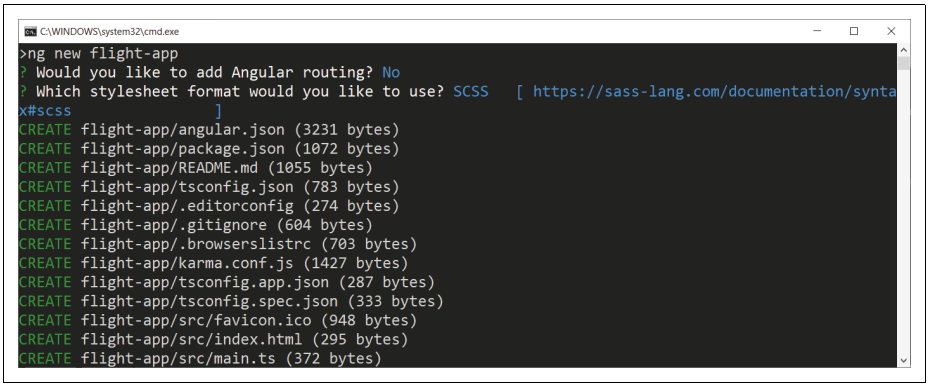

Abbildung 1-2: ng new stellt ein paar Fragen, bevor es ein neues Projekt generiert.

Je nach Angular-Version können diese Fragestellungen etwas variieren. Wir gehen hier von folgenden Einstellungen aus:

Add Angular Routing
: Diese Frage beantworten wir hier mit *No*. Um das Thema Routing kümmert sich Kapitel 8.

Stylesheet Format
: Wir empfehlen hier SCSS, eine Obermenge von CSS. Die Angular CLI kompiliert diese Dateien für den Browser nach CSS.

Da `ng new` auch zahlreiche Pakete via `npm` bezieht, kann der Aufruf etwas länger dauern.

Seit Version 12 verwendet die CLI standardmäßig den sogenannten Strict Mode. In diesem Modus führen sowohl der TypeScript-Compiler als auch Angular selbst strengere Code-Prüfungen durch. Hierdurch sollen Programmierfehler rascher entdeckt werden.

Falls Sie diese strengeren Prüfungen nicht verwenden wollen, verwenden Sie den Schalter `--strict`:

```
ng new flight-app --strict false
```

Wir gehen in in diesem Buch jedoch davon, dass der Strict Mode aktiviert ist. Das entspricht auch den Empfehlungen des Angular-Teams.

Angular-Anwendung starten

Um Ihre Anwendung zu starten, wechseln Sie in den generierten Projektordner. Dort bauen Sie mit `ng serve` die Anwendung und stellen sie über einen Demowebserver bereit:

```
cd flight-app
ng serve -o
```

Der Schalter `-o` öffnet einen Browser, der die Anwendung anzeigt. Standardmäßig findet sich diese Anwendung unter *http://localhost:4200*. Ist Port 4200 schon belegt, erkundigt sich `ng serve` nach einer Alternative. Außerdem nimmt der Schalter `--port` den gewünschten Port gleich beim Start von `ng serve` entgegen:

```
ng serve -o --port 4242
```

Die im Browser angezeigte Anwendung sieht wie in Abbildung 1-3 aus. Auch hier kann es von Version zu Version zu Abweichungen kommen.

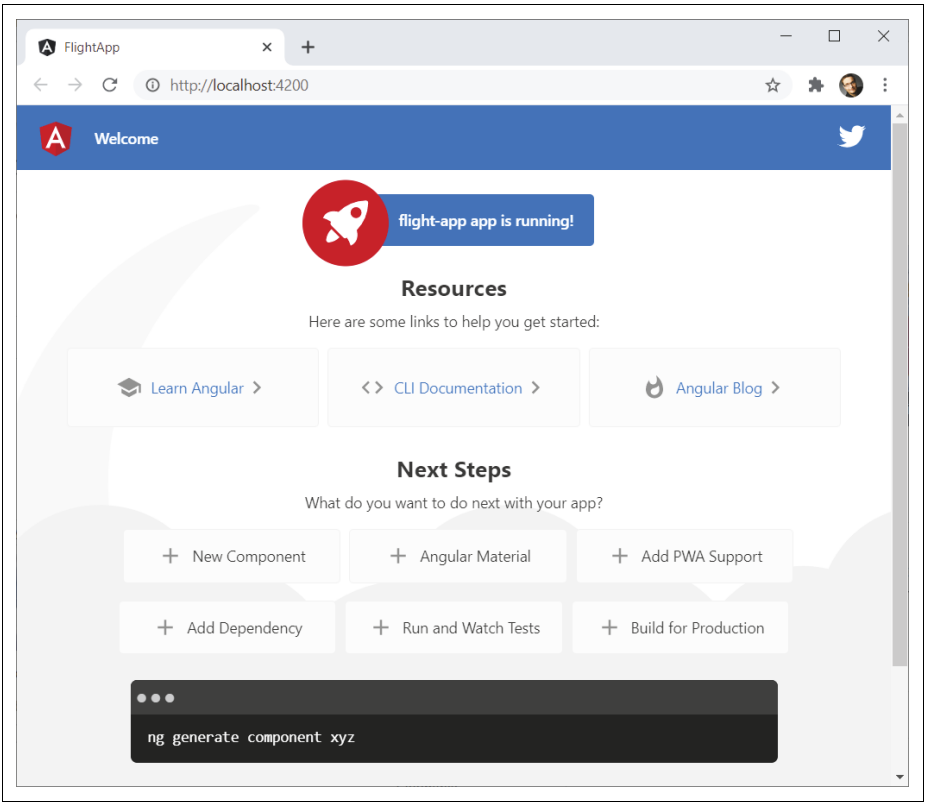

Abbildung 1-3: Generierte Angular-Anwendung

Der für die Entwicklung gedachte Befehl `ng serve` macht aber noch ein wenig mehr: Er überwacht sämtliche Quellcodedateien und stößt das Kompilieren sowie Generieren der Bundles erneut an, wenn sie sich ändern. Danach aktualisiert er auch das Browserfenster.

Um das auszuprobieren, können Sie mit Visual Studio Code die Datei *src\app\app.component.html* öffnen und das erste Vorkommen von *Welcome* durch *Hello World!* ändern. Nach dem Speichern der Datei sollte `ng serve` den betroffenen Teil der Anwendung neu kompilieren, bundeln und den Browser aktualisieren (siehe Abbildung 1-4).

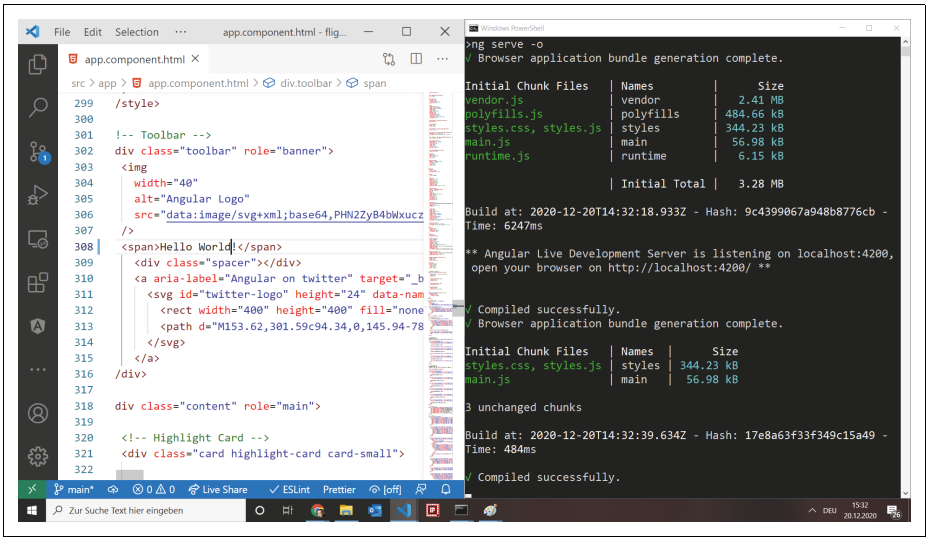

Abbildung 1-4: Generierte Angular-Anwendung ändern

Wenn Sie Visual Studio Code verwenden, sollten Sie zunächst den Hauptordner Ihrer Angular-Anwendung mit dem Befehl *File/Open Folder* öffnen. Der Hauptordner ist der, der auch die Datei *package.json* beinhaltet. Das stellt sicher, dass Visual Studio Code sämtliche Konfigurationsdateien findet und keine unnötigen Fehler anzeigt.

Danach können Sie die gewünschte Datei über den links angezeigten Explorer suchen und öffnen. Alternativ dazu bietet sich die Tastenkombination *Strg+P* an. Sie öffnet ein kleines Fenster, mit dem man nach der gewünschten Datei suchen kann.

Mit *Strg+Umschalt+C* können Sie übrigens jederzeit eine externe Konsole im aktuellen Ordner öffnen, um z.B. die Angular CLI auszuführen. Die Tastenkombination *Strg+Umschalt+Ö* öffnet hingegen die Konsole als Terminal direkt in Visual Studio Code.

Die automatische Generierung der Bundles nach einer Änderung am Programmcode funktioniert meist ganz gut, aber ab und an kommt die CLI aus dem Tritt. Das ist unter anderem dann der Fall, wenn Sie mehrere Dateien rasch hintereinander speichern. Auch das Umbenennen von Dateien bringt diesen Mechanismus aus dem Konzept.

Abhilfe schafft hier ein erneutes Speichern der betroffenen Dateien oder – wenn alle Stricke reißen – ein Neustart von `ng serve`.

Build mit CLI

Während `ng serve` für die Entwicklung sehr komfortabel ist, eignet es sich nicht für den Produktiveinsatz. Um Bundles für die Produktion zu generieren, nutzen Sie die Anweisung

```
ng build
```

Seit Angular CLI 12 führt `ng build` zahlreiche Optimierungen, die zu kleineren Bundles führen, automatisch durch. Davor musste man diese Optimierungen explizit mit dem Schalter `--prod` anfordern.

Ein Beispiel für eine solche Optimierung ist die Minifizierung, bei der unnötige Zeichen wie Kommentare oder Zeilenschaltungen entfernt sowie Ihre Anweisungen durch kompaktere Darstellungsformen ersetzt werden. Ein weiteres Beispiel ist das sogenannte Tree-Shaking, das nicht benötigte Framework-Bestandteile identifiziert und entfernt. Diese Optimierungen verlangsamen natürlich den Build-Prozess ein wenig.

Die generierten Bundles finden sich im Ordner *dist/flight-app*. Im Rahmen der Bereitstellung müssen Sie diese Dateien lediglich auf den Webserver Ihrer Wahl kopieren. Da es sich aus Sicht des Webservers hierbei um eine statische Webanwendung handelt, müssen Sie dort auch keine zusätzliche Skriptsprache und kein Web-Framework installieren.

Projektstruktur von CLI-Projekten

Die von der CLI generierte Projektstruktur orientiert sich an Best Practices, die sich auch in anderen Projekten finden. Für einen ersten Überblick präsentiert Tabelle 1-1 die wichtigsten Dateien. Wir gehen im Laufe des Buchs auf diese und weitere Dateien bei Bedarf genauer ein.

Tabelle 1-1: Projektstruktur

Ordner/Datei	Beschreibung
src/	Beinhaltet alle Quellcodedateien (TypeScript, HTML, CSS etc.).
src/main.ts	Dieser Quellcodedatei kommt besondere Bedeutung zu. Die CLI nutzt sie als Einstiegspunkt in die Anwendung. Deswegen wird ihr Code beim Programmstart zuerst ausgeführt. Standardmäßig beinhaltet sie ein paar Zeilen zum Starten von Angular. Normalerweise müssen Sie diese Datei nicht anpassen.
src/styles.scss	Hier können Sie Ihre eigenen globalen Styles eintragen. Die Dateiendung, z. B. *css* oder *scss*, hängt von der beim Generieren des Projekts gewählten Option ab.
src/app/	Dieser Ordner und seine Unterordner beinhalten die entwickelten Programmdateien wie zum Beispiel Angular-Komponenten.
src/assets/	Ordner mit statischen Dateien, die die CLI beim Build in das Ausgabeverzeichnis kopiert. Hier könnten Sie zum Beispiel Bilder oder JSON-Dateien ablegen.
dist/	Beinhaltet die von `ng build` generierten Bundles für die Auslieferung auf einen Server. Der Einsatz von `ng serve` schreibt diese Bundles hingegen nicht auf die Platte, sondern hält sie lediglich im Hauptspeicher vor.
node_modules/	Beinhaltet sämtliche Module, die über `npm` bezogen wurden. Dazu gehören der TypeScript-Compiler und andere Werkzeuge für den Build, aber auch sämtliche Bibliotheken für Angular.
tsconfig.json	Konfigurationsdatei für TypeScript. Hier wird zum Beispiel festgelegt, dass der TypeScript-Compiler Ihren Quellcode nach ECMAScript 2015 kompilieren soll. Das ist jene JavaScript-Version, die von allen modernen Browsern der letzten Jahre unterstützt wird.

Tabelle 1-1: Projektstruktur (Fortsetzung)

Ordner/Datei	Beschreibung
.browserslistrc	Listet sämtliche Browser, die die Angular-Anwendung unterstützen soll. Aus dieser Liste ermittelt die Angular CLI nötige CSS-Präfixe, die es beim Kompilieren einfügt. Bis Angular 12 wurde aus dieser Datei auch abgeleitet, ob zusätzlich Legacy-Bundles (ECMAScript 5) für Browser wie Internet Explorer 11 zu erzeugen sind.
package.json	Referenziert sämtliche Bibliotheken, die benötigt werden, inklusive der gewünschten Versionen. Wenn Sie einen Blick auf die Abschnitte `dependencies` und `devDependencies` werfen, sehen Sie alle von `ng new` installierten Pakete. Da der Ordner *node_modules* mit diesen Paketen nicht in die Quellcodeverwaltung eingecheckt wird, sind diese Hinweise notwendig. Ihre Entwickler-Kolleginnen und -Kollegen müssen lediglich die Anweisung `npm install` ausführen, um sie in den *node_modules*-Ordner zu laden.
index.html	Die Startseite. Der Build-Prozess erweitert sie um Referenzen auf die generierten Bundles.
angular.json	Mit dieser Datei lässt sich das Verhalten der CLI anpassen. Beispielsweise referenziert sie globale Styles oder Skripte, die es einzubinden gilt.

Lassen Sie uns nun ein paar der Programmdateien unter *src/app* etwas genauer betrachten. Starten wir dabei mit der generierten *AppComponent*. Wie die meisten Angular-Komponenten besteht sie aus mehreren Dateien:

app.component.ts
: TypeScript-Datei, die das Verhalten der Komponente definiert.

app.component.html
: HTML-Datei mit der Struktur der Komponente.

app.component.scss
: Datei mit lokalen Styles für die Komponente. Allgemeine Styles können in die besprochene *styles.scss* eingetragen werden.

Beispiel 1-1 zeigt den Inhalt der generierten *app.component.ts*:

Beispiel 1-1: Die generierte app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  title = 'flight-app';
}
```

Es handelt sich dabei um eine Klasse, die lediglich eine Eigenschaft `title` vom Typ `string` besitzt. Letzteres muss hier gar nicht explizit angegeben werden: TypeScript kann sich diesen Umstand aus dem zugewiesenen Standardwert herleiten.

Die Angabe von `export` definiert, dass die Klasse auch in anderen Dateien der Anwendung genutzt werden darf.

Die Klasse wurde mit dem Dekorator `Component` versehen. Dekoratoren definieren Metadaten für Programmkonstrukte wie z. B. Klassen. Diesen importiert die Komponente in der ersten Zeile aus dem Paket `@angular/core`. Bei der Nutzung eines Dekorators wird ihm das Symbol `@` vorangestellt.

Die Metadaten beinhalten den Selektor der Komponente. Das ist in der Regel der Name eines HTML-Elements, das die Komponente repräsentiert. Um die Komponente aufzurufen, können Sie also die folgende Schreibweise in einer HTML-Datei verwenden:

```
<app-root></app-root>
```

Der Dekorator verweist außerdem auf das HTML-Template der Komponente und ihre SCSS-Datei mit lokalen Styles. Letztere ist standardmäßig leer. Die HTML-Datei beinhaltet den Code für die oben betrachtete Startseite. Die ist zwar schön, enthält aber eine Menge HTML-Markup. Ersetzen Sie mal zum Ausprobieren den *gesamten* Inhalt dieser HTML-Datei durch folgendes Fragment:

```
<h1>{{title}}</h1>
```

Wenn Sie nun die Anwendung starten (`ng serve -o`), sollten Sie den Inhalt der Eigenschaft `title` als Überschrift sehen. Die beiden geschweiften Klammernpaare definieren eine sogenannte Datenbindung. Angular bindet also die angegebene Eigenschaft an die jeweilige Stelle im Template.

Mehr Informationen zu TypeScript, Datenbindungen und Angular im Allgemeinen finden Sie in den nächsten beiden Kapiteln. Um diesen Rundgang durch die generierten Programmdateien abzuschließen, möchten wir jedoch noch auf drei weitere generierte Dateien hinweisen. Eine davon ist die Datei *app.module.ts*, die ein Angular-Modul beinhaltet (siehe Beispiel 1-2).

Beispiel 1-2: Das generierte AppModule

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Angular-Module sind Datenstrukturen, die zusammengehörige Building-Blocks wie Komponenten zusammenfassen. Technisch gesehen, handelt es sich dabei um eine

weitere Klasse. Sie ist in den meisten Fällen leer und dient lediglich als Träger von Metadaten, die über den `NgModule`-Dekorator angegeben werden.

Lassen Sie uns einen Blick auf die Eigenschaften von `NgModule` werfen:

declarations
: Definiert die Inhalte des Moduls. Derzeit beschränken sich diese auf unsere `AppComponent`. Sie wird in der dritten Zeile unter Angabe eines relativen Pfads, der auf die Datei *app.component.ts* verweist, importiert. Die Dateiendung *.ts* wird hierbei weggelassen.

imports
: Importiert weitere Module. Das gezeigte Beispiel importiert lediglich das `BrowserModule`, das alles beinhaltet, um Angular im Browser auszuführen. Das ist auch der Standardfall.

providers
: Hier könnte man sogenannte Services, die Logiken für mehrere Komponenten anbieten, registrieren. Kapitel 5 geht im Detail darauf ein.

bootstrap
: Diese Eigenschaft verweist auf sämtliche Komponenten, die beim Start der Anwendung zu erzeugen sind. Häufig handelt es sich dabei lediglich um eine einzige Komponente. Diese sogenannte Root-Component repräsentiert die gesamte Anwendung und ruft dazu weitere Komponenten auf.

Das Modul, das die Root-Component bereitstellt, wird auch als Root-Module bezeichnet. Angular nimmt es beim Start der Anwendung entgegen und rendert die darin zu findende Root-Component. Für diese Aufgabe hat die CLI die Datei *main.ts* eingerichtet (siehe Beispiel 1-3).

Beispiel 1-3: Die generierte Datei main.ts

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

Die Funktion `platformBrowserDynamic` erzeugt eine sogenannte Plattform, die die Ausführung von Angular im Browser möglich macht. Andere Plattformen ermöglichen zum Beispiel die serverseitige Ausführung von Angular oder die Ausführung in mobilen Anwendungen. Die Nutzung im Browser ist jedoch der hier betrachtete Standardfall.