Ferdinand Malcher • Danny Koppenhagen
Johannes Hoppe

Angular

Das Praxisbuch – von den Grundlagen bis zur professionellen Entwicklung mit Signals

dpunkt.verlag

Hinweise zur Benutzung

Dieses E-Book ist **urheberrechtlich geschützt**. Mit dem Erwerb des E-Books haben Sie sich verpflichtet, die Urheberrechte anzuerkennen und einzuhalten. Sie sind berechtigt, dieses E-Book für persönliche Zwecke zu nutzen. Sie dürfen es auch ausdrucken und kopieren, aber auch dies nur für den persönlichen Gebrauch. Die Weitergabe einer elektronischen oder gedruckten Kopie an Dritte ist dagegen nicht erlaubt, weder ganz noch in Teilen. Und auch nicht eine Veröffentlichung im Internet oder in einem Firmennetzwerk.

Copyright-Vermerk

Nutzungs- und Verwertungsrechte

Sie sind berechtigt, dieses E-Book ausschließlich für persönliche Zwecke zu nutzen. Insbesondere sind Sie berechtigt, das E-Book für Ihren eigenen Gebrauch auszudrucken oder eine Kopie herzustellen, sofern Sie diese Kopie auf einem von Ihnen alleine und persönlich genutzten Endgerät speichern. Zu anderen oder weitergehenden Nutzungen und Verwertungen sind Sie nicht berechtigt.

So ist es insbesondere unzulässig, eine elektronische oder gedruckte Kopie an Dritte weiterzugeben. Unzulässig und nicht erlaubt ist des Weiteren, das E-Book im Internet, in Intranets oder auf andere Weise zu verbreiten oder Dritten zur Verfügung zu stellen. Eine öffentliche Wiedergabe oder sonstige Weiterveröffentlichung und jegliche den persönlichen Gebrauch übersteigende Vervielfältigung des E-Books ist ausdrücklich untersagt. Das vorstehend Gesagte gilt nicht nur für das E-Book insgesamt, sondern auch für seine Teile (z. B. Grafiken, Fotos, Tabellen, Textabschnitte).

Urheberrechtsvermerke, Markenzeichen und andere Rechtsvorbehalte dürfen aus dem E-Book nicht entfernt werden.

Die automatisierte Analyse des Werkes, um daraus Informationen insbesondere über Muster, Trends und Korrelationen gemäß § 44b UrhG (»Text und Data Mining«) zu gewinnen, ist untersagt.

Markenschutz

Die in diesem Werk wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. können auch ohne besondere Kennzeichnung Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.

Haftungsausschluss

Ungeachtet der Sorgfalt, die auf die Erstellung von Text, Abbildungen und Programmen verwendet wurde, können weder Verlag noch Autor*innen, Herausgeber*innen oder Übersetzer*innen für mögliche Fehler und deren Folgen eine juristische Verantwortung oder irgendeine Haftung übernehmen.

Ferdinand Malcher · Danny Koppenhagen · Johannes Hoppe

Angular

Das Praxisbuch – von den Grundlagen bis zur professionellen Entwicklung mit Signals

dpunkt.verlag

Wir hoffen, dass Sie Freude an diesem Buch haben und sich Ihre Erwartungen erfüllen. Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: **service@rheinwerk-verlag.de**.

Informationen zu unserem Verlag und Kontaktmöglichkeiten finden Sie auf unserer Verlagswebsite **www.dpunkt.de**. Dort können Sie sich auch umfassend über unser aktuelles Programm informieren und unsere Bücher und E-Books bestellen.

Autoren: Ferdinand Malcher, Danny Koppenhagen, Johannes Hoppe, *team@angular-buch.com*
Lektorat: Sandra Bollenbacher
Buchmanagement: Julia Griebel
Copy-Editing: Annette Schwarz, Ditzingen
Satz: Da-TeX Gerd Blumenstein, Leipzig, *www.da-tex.de*
Herstellung: Stefanie Weidner, Frank Heidt
Covergestaltung: Eva Hepper, Silke Braun

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

ISBN Print: 978-3-98889-064-1
ISBN PDF: 978-3-98890-331-0
ISBN ePub: 978-3-98890-332-7

1. Auflage 2026

dpunkt.verlag ist eine Marke des Rheinwerk Verlags.

Rheinwerk Verlag GmbH • Rheinwerkallee 4 • 53227 Bonn
service@rheinwerk-verlag.de

Inhaltsverzeichnis

Vorwort

Angular ist eines der populärsten Frameworks für die Entwicklung von Single-Page-Applikationen. Das Framework wird weltweit von großen Unternehmen eingesetzt, um modulare, skalierbare und gut wartbare Applikationen zu entwickeln. Mit Angular in Version 2.0.0 setzte Google im Jahr 2016 einen Meilenstein in der Welt der modernen Webentwicklung: Das Framework nutzt die Programmiersprache TypeScript, bietet ein ausgereiftes Tooling und ermöglicht die komponentenbasierte Entwicklung von Single-Page-Anwendungen für den Browser und für Mobilgeräte.

In kurzer Zeit haben sich rund um Angular ein umfangreiches Ökosystem und eine vielfältige Community gebildet. Angular gilt neben React.js und Vue.js als eines der weltweit beliebtesten Webframeworks. Du hast also die richtige Entscheidung getroffen, als du Angular für die Entwicklung deiner Projekte ausgewählt hast.

Der Einstieg in Angular ist umfangreich, aber die Konzepte sind durchdacht und konsequent. Häufig verwendet man im Zusammenhang mit Angular das Attribut *opinionated*, das wir im Deutschen mit dem Begriff *meinungsstark* ausdrücken können: Angular ist ein meinungsstarkes Framework, das viele klare Richtlinien zu Architektur, Codestruktur und Best Practices definiert. Das kann zu Anfang umfangreich erscheinen, sorgt aber dafür, dass in der gesamten Community einheitliche Konventionen herrschen, Standardlösungen existieren und bestehende Bibliotheken vorausgewählt wurden.

Nach mehr als 10 Jahren Entwicklung ist Angular ein ausgereiftes, aber keineswegs veraltetes Framework: Das Angular-Team bei Google überrascht regelmäßig mit neuen Innovationen und durchdachten Konzepten, aber stets mit einem starken Fokus auf Abwärtskompatibilität und einfache Migration.

Zu diesem Buch

Wir Autoren beschäftigen uns seit 2015 intensiv mit dem Framework und sind als Workshopleiter, Berater und Entwickler für Angular aktiv. Neun Jahre nach Veröffentlichung unseres ersten Standardwerks zu Angular haben wir mit diesem Buch, das du gerade in Händen hältst, einen Neuanfang gewagt: Wir legen den Schwerpunkt auf das *moderne Angular*. Dabei zeigen wir den aktuellen Stand des Frameworks und moderne Best Practices: Signals, Control Flow, Resource API und Signal Forms sind nur ein Teil der umfangreichen Themenauswahl, die wir für dieses Buch getroffen haben. Besonderen Wert legen wir auf Barrierefreiheit und durchgehende Softwaretests.

Dieses Buch zeigt dir, wie du mit Angular komponentenbasierte Single-Page-Applikationen erstellst. Dazu entwickeln wir mit dir gemeinsam eine Anwendung, anhand derer du die Konzepte und Features von Angular lernst und praktisch anwendest. Wir führen dich Schritt für Schritt durch das Framework: vom Projektsetup über Komponenten, Signals, Routing, Formulare und HTTP bis hin zur reaktiven Programmierung mit RxJS. Die umfangreichen Theorieteile eignen sich auch später als Nachschlagewerk im Entwicklungsalltag.

Nach dem Lesen dieses Praxisbuchs bist du in der Lage,

- das Framework sicher und eigenständig zu verwenden,
- modulare, strukturierte und wartbare Webanwendungen mithilfe von Angular zu entwickeln sowie
- durch die Entwicklung von Tests qualitativ hochwertige Anwendungen zu erstellen.

Die Entwicklung mit Angular macht vor allem eines: *Spaß!* Diesen Enthusiasmus für das Framework und für Webtechnologien möchten wir dir in diesem Buch vermitteln – wir nehmen dich mit auf die Reise in die Welt der modernen Webentwicklung!

Versionen und Umgang mit Aktualisierungen

Die Versionsnummer *x.y.z* von Angular basiert auf *Semantic Versioning*.[1] Der Release-Zyklus von Angular ist kontinuierlich geplant: Im Rhythmus von ungefähr sechs Monaten erscheint eine neue Major-Version *x*.

[1] *https://ng-buch.de/d/semver* – Semantic Versioning 2.0.0

Die Minor-Versionen *y* werden monatlich herausgegeben, nachdem eine Major-Version erschienen ist. Jede Major-Version wird planmäßig für 1,5 Jahre unterstützt und weiterentwickelt (Long-Term Support (LTS)).

Das Release einer neuen Major-Version von Angular bedeutet keineswegs, dass alle Ideen verworfen werden und die Software nach einem Update nicht mehr funktioniert. Auch wenn du eine neuere Angular-Version verwendest, behalten die in diesem Buch beschriebenen Konzepte ihre Gültigkeit. Die Grundideen von Angular sind seit Version 2.0.0 konsistent und auf Beständigkeit über einen langen Zeitraum ausgelegt. Alle Updates zwischen den Major-Versionen waren in der Vergangenheit problemlos möglich, ohne dass Breaking Changes die gesamte Anwendung unbenutzbar machten. Gibt es doch gravierende Änderungen, so werden stets ausführliche Informationen und Tools zur Migration angeboten.

Auf der Website zu diesem Buch findest du den Code für das Beispielprojekt und viele weiterführende Informationen. Unter anderem veröffentlichen wir dort zu jeder Major-Version einen Artikel mit den wichtigsten Neuerungen in Angular. Wir empfehlen dir deshalb, unbedingt einen Blick auf die Begleitwebsite zu werfen, bevor du dich mit den Inhalten des Buchs beschäftigst:

Die Begleitwebsite zum Buch

https://angular-buch.com

An wen richtet sich das Buch?

Dieses Buch richtet sich an Menschen, die bereits grundlegende Kenntnisse in der Softwareentwicklung mitbringen. Vorwissen zu JavaScript/TypeScript und HTML ist von Vorteil – es ist aber keine Voraussetzung, um mit diesem Buch Angular zu lernen. Wenn du jedoch bereits mit der Webentwicklung vertraut bist, wirst du mit diesem Buch schnell starten können. Falls du gar keine Erfahrung mit HTML und TypeScript hast, empfehlen wir dir, zunächst die grundlegenden Kenntnisse in diesen Bereichen zu festigen.

Du benötigst *keinerlei* Vorkenntnisse im Umgang mit Angular. Ebenso musst du dich nicht vorab mit benötigten Tools und Hilfsmitteln für

die Entwicklung von Angular-Applikationen vertraut machen. Das nötige Wissen darüber wird in diesem Buch vermittelt.

Wir erschließen uns die Welt von Angular praxisorientiert anhand eines Beispielprojekts. Jedes Thema wird zunächst ausführlich in der Theorie behandelt, sodass du die Grundlagen auch losgelöst vom Beispielprojekt nachlesen kannst. Wir wollen einen soliden Einstieg in Angular bieten, Best Practices zeigen und das Bewusstsein für barrierefreie Webanwendungen stärken. Die meisten Aufgaben aus dem Entwicklungsalltag wirst du durch die praktischen Beispiele souverän meistern können.

Wir hoffen, dass dieses Buch deine tägliche Begleitung bei der Arbeit mit Angular wird. Für Details zu den einzelnen Framework-Funktionen empfehlen wir immer auch einen Blick in die offizielle Dokumentation.[2]

Wie ist dieses Buch zu lesen?

Im ersten Teil des Buchs lernst du die benötigten Werkzeuge kennen. Wenn du bereits Erfahrung mit moderner Webentwicklung hast, kannst du diesen Teil gerne grob überfliegen.

Im Anschluss werden wir mit dir zusammen eine Beispielanwendung entwickeln. Die Konzepte und Technologien von Angular wollen wir dabei direkt am Beispiel vermitteln. Diesen Teil des Buchs solltest du in der vorgesehenen Reihenfolge durcharbeiten.

Jedes Thema behandeln wir in drei Schritten: Zunächst erklären wir alle Grundlagen, Konzepte und Features in einem Theorieteil. Anschließend wenden wir das Gelernte im durchgehenden Beispielprojekt namens »BookManager« an – je nach Umfang in einem oder mehreren Praxisteilen. Gute Softwarequalität ist uns wichtig. Deshalb endet jedes Praxiskapitel damit, dass wir die Korrektheit der Anwendung mit Softwaretests verifizieren.

Im letzten Teil »Wissenswertes« haben wir weitere Themen aufgegriffen, die für die Arbeit mit Angular besonders relevant sind. Hier findest du unter anderem die ausführlichen Grundlagenkapitel zu Softwaretests und Barrierefreiheit.

Darüber hinaus haben wir auf der Begleitwebsite zum Buch[3] weitere Themen als Online-Kapitel veröffentlicht.

---

[2] *https://ng-buch.de/d/ng-dev* – Angular Docs

[3] *https://angular-buch.com*

Lernen mit Copy & Paste und AI-Agenten

Wir wissen genau, wie groß die Versuchung ist, bei der Arbeit mit dem Beispielprojekt größere Teile des Codes zu kopieren oder von einem AI-Agenten vervollständigen zu lassen. Für die tägliche Arbeit mit dem Framework ist das ein hervorragender Weg zur Steigerung der Produktivität.

Zum Lernen von Angular möchten wir dich aber einladen, die Codebeispiele selbst zu *tippen*. Wir sind uns sicher, dass du so besser verstehen wirst, wie Angular funktioniert und wie das Framework erfolgreich in der Praxis eingesetzt wird.

Wir stellen alle Quelltexte für das Beispielprojekt selbstverständlich auf GitHub bereit, sodass du deinen Stand jederzeit vergleichen und korrigieren kannst.

Angular.Schule: Workshops und Beratung

Wir, die Autoren dieses Buchs, arbeiten seit Langem als Berater und Trainer für Angular. Wir haben die Erfahrung gemacht, dass man Angular in kleinen Gruppen am effektivsten lernen kann. In einem Workshop kann auf individuelle Fragen und Probleme direkt eingegangen werden – und es macht auch am meisten Spaß!

Schau doch einmal auf *https://angular.schule* vorbei. Dort bieten wir Angular-Schulungen in den Räumen deines Unternehmens, in offenen Gruppen oder als Online-Kurs an. Das Angular-Buch verwenden wir dabei in unseren Kursen zur Nacharbeit. Wir freuen uns auf deinen Besuch!

https://angular.schule

Danksagung

Dieses Buch hätte nicht seine Reife erreicht ohne die Hilfe und Unterstützung verschiedener Menschen. Ein großer Dank gilt unseren Familien, die viel Verständnis aufgebracht haben, wenn wir mal wieder am Buch saßen!

Wir danken besonders **Jan-Hendrik Hausner**: Er hat den gesamten Praxisteil durchgearbeitet, zahlreiche konstruktive Verbesserungsvorschläge gemacht und viele Fehler aufgespürt. **Jan Sebestyén**, **Maximilian Franzke** und **Milan Wanielik** danken wir ebenso für ihr wertvolles Feedback als Testleser.

Dem Team vom dpunkt.verlag, insbesondere **Sandra Bollenbacher**, danken wir für die gute Zusammenarbeit. Für das gewissenhafte Korrektorat unseres Manuskripts danken wir **Annette Schwarz**. Den professionellen Buchsatz mit LaTeX verdanken wir **Jens Dittmar**.

Besonderer Dank gilt dem **Angular-Team** und der Community dafür, dass sie eine großartige Plattform geschaffen haben, die uns den Entwicklungsalltag angenehmer macht. **Matthieu Riegler**, **Kirill Cherkashin** und **Miles Malerba** vom Angular-Team waren unsere Ansprechpartner für schnelle Antworten auf unsere Fachfragen – vielen Dank!

Viele weitere Menschen haben uns E-Mails mit persönlichem Feedback zu den Büchern der ersten Reihe zukommen lassen. Vielen Dank für diese wertvollen Rückmeldungen – sie haben uns motiviert, dieses neue Buch noch besser zu machen!

Teil I

Einführung

1 Benötigte Werkzeuge: Editor, Node.js und Co.

Im ersten Schritt richten wir die Entwicklungsumgebung für unsere Anwendung ein. Dazu prüfen wir, ob alle notwendigen Programme und Werkzeuge einsatzbereit sind.

Falls du bereits fit auf der Konsole bist, eine integrierte Entwicklungsumgebung (IDE) für die Webentwicklung nutzt und *Node.js* sowie *NPM* kennst, kannst du dieses Kapitel gerne überspringen!

1.1 Konsole, Terminal und Shell

Wir werden in diesem Buch einige Befehle auf der Konsole (Terminal) ausführen. Dafür solltest du mit der Eingabe von Konsolenbefehlen auf deinem Betriebssystem vertraut sein. Unter Microsoft Windows nutzt du z. B. die klassische Eingabeaufforderung (cmd) oder das Windows Terminal. Unter macOS oder Linux empfehlen wir das Terminal mit der *bash* oder *zsh*.

1.2 Paketverwaltung mit Node.js und NPM

Node.js ist eine Laufzeitumgebung zur Ausführung von JavaScript außerhalb des Browsers. Es basiert auf derselben JavaScript-Engine, die auch in Google Chrome zum Einsatz kommt. Mit Node.js können unter anderem serverbasierte Dienste und Kommandozeilenwerkzeuge mit JavaScript implementiert werden. Viele Tools im Umfeld der Webentwicklung basieren auf Node.js: die Angular CLI, der Testrunner Vitest oder CSS-Präprozessoren wie Sass und Less. Wir verwenden Node.js in diesem Buch zum Betrieb der Werkzeuge, die wir für die Entwicklung mit Angular benötigen.

Eng verbunden mit Node.js ist der Node Package Manager (NPM)[1]: Mit diesem Paketmanager haben wir Zugriff auf eine Vielzahl von Paketen für Node.js und den Browser – er ist das Eingangstor zum vielfältigen Ökosystem von JavaScript. Wir verwenden NPM, um die Abhängigkeiten für unsere Angular-Projekte zu installieren. Dazu gehören nicht nur Tools wie die Angular CLI oder der TypeScript-Compiler, sondern auch das Angular-Framework selbst.

Node.js und NPM installieren

Auf der Downloadseite von Node.js findest du Installationspakete für alle verbreiteten Betriebssysteme.[2] Wähle dort am besten eine Version mit Long-Term Support (LTS), denn sie bietet hohe Stabilität und wird langfristig gepflegt.

Windows

Unter Windows empfehlen wir den offiziellen Installer von der Node.js-Website. Lade die `.msi`-Datei herunter und folge dem Installationsassistenten. Alternativ kannst du Node.js über *winget* installieren. Dieser offizielle Paketmanager von Microsoft ist unter Windows 10 und 11 bereits vorinstalliert.

Listing 1.1 *Node.js installieren mit winget*

```
$ winget install -e --id OpenJS.NodeJS.LTS
```

macOS

Auch für macOS empfehlen wir den offiziellen Installer von der Node.js-Website. Wenn du den Paketmanager Homebrew[3] nutzt, kannst du Node.js alternativ darüber installieren:

Listing 1.2 *Node.js installieren mit Homebrew*

```
$ brew install node
```

Linux

Unter Linux kannst du Node.js über den Paketmanager deiner Distribution installieren. Da die Standard-Paketquellen vieler Distributionen jedoch nur ältere Versionen enthalten, empfehlen wir die Paketquellen von

[1] *https://ng-buch.de/d/npm* – NPM

[2] *https://ng-buch.de/d/nodejs* – Node.js

[3] *https://ng-buch.de/d/homebrew* – Homebrew

NodeSource.[4] Diese bieten aktuelle Pakete für Debian, Ubuntu, Fedora und weitere verbreitete Distributionen an.

Installation prüfen

Nach der Installation prüfen wir auf der Kommandozeile, ob `node` und `npm` richtig installiert sind, indem wir die Versionsnummern ausgeben:

```
$ node -v
$ npm -v
```

Listing 1.3
Versionen von Node.js und NPM prüfen

Die Angular CLI verlangt klar festgelegte Node.js-Versionen, die mit neuen Major-Versionen von Angular aktualisiert werden. Welche Version von Node.js jeweils mindestens benötigt wird, ist in der Angular-Dokumentation zu finden.[5] Solltest du eine nicht unterstützte Version verwenden, wirst du beim Ausführen von CLI-Befehlen eine Fehlermeldung erhalten. Du kannst also nicht viel falsch machen und musst dann einfach nur die richtige Version von Node.js installieren.

Versionsmanager für Node.js

Wenn du an mehreren Projekten arbeitest, die unterschiedliche Node.js-Versionen erfordern, lohnt sich der Einsatz eines Versionsmanagers. Mit Tools wie *nvm*[a] (Node Version Manager) oder *fnm*[b] (Fast Node Manager) kannst du mehrere Versionen parallel installieren und per Befehl eine bestimmte Version auswählen. Für die Arbeit mit diesem Buch ist ein Versionsmanager jedoch nicht notwendig.

[a] *https://ng-buch.de/d/nvm* – GitHub: Node Version Manager (nvm)

[b] *https://ng-buch.de/d/fnm* – GitHub: Fast Node Manager (fnm)

NPM-Pakete installieren

Wenn wir Pakete mit NPM installieren, unterscheiden wir zwischen einer *lokalen* und einer *globalen* Installation.

Installieren wir Pakete *lokal*, werden diese im Dateisystem unseres jeweiligen Projekts abgelegt. Dafür wird automatisch ein Unterordner mit dem Namen `node_modules` erstellt, der die installierten Pakete enthält. Diese Variante ist empfehlenswert, wenn wir direkte Abhängigkeiten für

[4] *https://ng-buch.de/d/nodesource* – NodeSource: Node.js Distributions

[5] *https://ng-buch.de/d/ng-versions* – Angular Docs: Version compatibility

ein Projekt installieren möchten. Im Hauptverzeichnis eines Projekts existiert in der Regel eine Datei mit dem Namen `package.json`, in der alle NPM-Abhängigkeiten verzeichnet sind. Das behandeln wir auf Seite 52 ausführlicher, wenn wir unser Beispielprojekt anlegen.

Generell gilt, dass eine lokale Installation der globalen vorzuziehen ist. Wenn wir auf demselben System mehrere Softwareprojekte parallel entwickeln, kann so jedes Projekt seine eigenen Versionen der Abhängigkeiten besitzen.

```
$ npm install <paketname>
```

Bei der *globalen* Installation wird das Paket in einem zentralen Ordner auf dem System installiert. Diese Variante bietet sich an, wenn die Pakete ausführbare Kommandozeilenbefehle beinhalten. Die Befehle sind dann aus jedem Verzeichnis heraus aufrufbar. Wir werden später die Angular CLI kennenlernen (ab Seite 29) und global installieren – auf diese Weise können wir das Werkzeug auf dem gesamten System verwenden. Die Option `-g` ermöglicht eine globale Installation:

```
$ npm install -g <paketname>
```

Warnungen und Hinweise zu Sicherheitslücken

Beim Installieren von NPM-Paketen können Warnungen erscheinen, die vielfältige Ursachen haben. In den meisten Fällen beeinträchtigt das aber nicht die Entwicklung der Angular-Anwendung. Folgende Warnungen begegnen uns besonders häufig:

- **SKIPPING OPTIONAL DEPENDENCY**: Eine irrelevante Abhängigkeit wurde übersprungen. Die Warnung kann ignoriert werden.
- **found X vulnerabilities:** In einer der Abhängigkeiten wurde eine Sicherheitsschwachstelle entdeckt. Eine solche Sicherheitswarnung kann nicht immer direkt korrigiert werden, weil das Tooling womöglich bei einer neueren Version in der Konstellation nicht mehr funktioniert. Das Problem ist aber selten tatsächlich schwerwiegend: Angular erzeugt beim Build ein kompiliertes Bundle, das später ausgeliefert wird. Die Warnung betrifft normalerweise nur das Tooling, das Bundle ist dann in der Regel nicht betroffen. In einem frischen Projekt können diese Meldungen ignoriert werden. Bei einem etablierten Projekt mit vielen Abhängigkeiten zu Fremdpaketen ist es hingegen

wichtig, die Schwachstellen tatsächlich regelmäßig zu prüfen und betroffene Pakete zu aktualisieren.

Bitte lass dich also von den Warnungen nicht verunsichern. Das ganze Tooling rund um Angular hat eine große Liste an Abhängigkeiten, sodass es immer wieder zu Warnungen kommen kann.

1.3 Visual Studio Code

Visual Studio Code (kurz: VS Code)[6] ist eine kostenlose und quelloffene Entwicklungsumgebung. Lass dich vom ähnlichen Namen nicht täuschen: Diese IDE ist ein eigenständiges Produkt und nicht mit dem bekannten *Visual Studio* verwandt.

VS Code ist unter Windows, macOS und Linux lauffähig, hat eine moderne Oberfläche und unterstützt zahlreiche Programmiersprachen. Da die TypeScript-Integration sehr ausgereift ist, verwenden wir vorzugsweise diese IDE für die Entwicklung von Angular-Anwendungen. VS Code bringt eine sehr gute automatische Codevervollständigung mit und zeigt die zugehörige Dokumentation an, wenn du mit der Maus über Code fährst. Außerdem besitzt der Editor von Haus aus eine Git-Integration[7] und bietet ein Terminal an. So kannst du die im Buch verwendeten Befehle direkt ins integrierte Terminal eingeben, ohne das Fenster wechseln zu müssen.

Andere Entwicklungsumgebungen nutzen

Wir empfehlen dir, Visual Studio Code zu verwenden, um mit Angular zu entwickeln. Wenn du allerdings bereits mit einer anderen geeigneten IDE vertraut bist, kannst du diese selbstverständlich weiter nutzen. Das wichtigste Kriterium ist eine direkte Unterstützung für TypeScript und Angular, ggf. mithilfe von Plug-ins. Aus unserer Erfahrung sind die IDEs der Firma JetBrains (WebStorm oder IntelliJ) ebenfalls gut geeignet.

Empfohlene Erweiterung: Angular Language Service

Über den Reiter *Extensions* in der Seitenleiste von Visual Studio Code können wir Erweiterungen für den Editor installieren. Welche Erweite-

[6] *https://ng-buch.de/d/vscode* – Visual Studio Code

[7] Git ist ein Tool zur Versionsverwaltung von Quellcode.

rungen im Entwicklungsalltag tatsächlich praktisch sind, ist sehr individuell. Deshalb wollen wir keine umfangreichen Empfehlungen aussprechen.

Eine Erweiterung ist für die Arbeit mit Angular aber unverzichtbar: der *Angular Language Service*. Dieses Werkzeug wird direkt vom Angular-Team bereitgestellt und integriert die Frameworkunterstützung in Visual Studio Code. Wir erhalten damit Typprüfungen, Autovervollständigung und korrektes Syntax-Highlighting in den HTML-Templates der Anwendung.

Den Angular Language Service findest du mit der Extension-Suche am besten über seinen Paketnamen `Angular.ng-template`. Versichere dich vor der Installation, dass du das richtige Paket ausgewählt hast. Die Anzahl der Downloads ist ein guter Indikator. Alternativ kannst du auch direkt unserem Link folgen.[8] Es ist nie verkehrt, hier gründlich zu sein, um keine Schadsoftware zu installieren – das gilt auch für andere Erweiterungen, z. B. für den Browser.

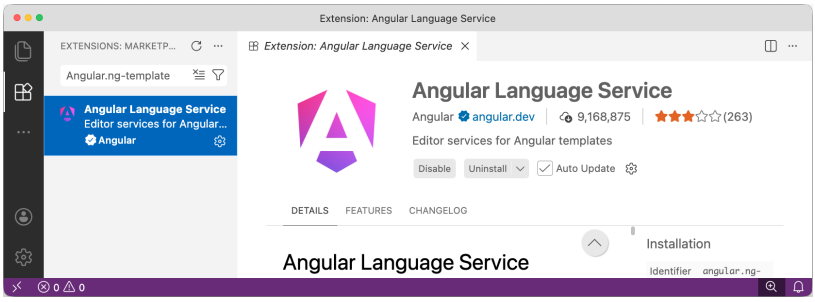


Abb. 1.1 *Erweiterungen in VS Code*

1.4 Google Chrome

Zur Darstellung der Angular-Anwendung und für das Debugging empfehlen wir Google Chrome.[9] Dieser Browser bringt unter anderem leistungsfähige Entwicklertools mit, die das Debugging von Webanwendungen erleichtern. Selbstverständlich kannst du auch einen anderen modernen Browser für die Entwicklung verwenden, z. B. Mozilla Firefox, Microsoft Edge oder Safari.

[8] *https://ng-buch.de/d/ng-langservice* – VS Code Marketplace: Angular Language Service

[9] *https://ng-buch.de/d/chrome* – Google Chrome

1.5 Angular Developer Tools

Die Angular DevTools[10] sind eine Browsererweiterung, die speziell für das Debugging und die Performanceanalyse von Angular-Anwendungen entwickelt wurde. Diese Tools sind für Google Chrome, Microsoft Edge und Mozilla Firefox verfügbar und erweitern die Standard-Entwicklertools des Browsers um Angular-spezifische Funktionen.

Nach der Installation der Erweiterung findest du in den Browser-DevTools einen zusätzlichen Tab »Angular«. Hier können wir die Komponentenhierarchie der Anwendung erkunden, den Zustand einzelner Komponenten inspizieren und zur Laufzeit bearbeiten sowie die Performance der Anwendung analysieren.

Wichtig: Die Angular DevTools funktionieren nur mit Development-Builds (d. h. bei der lokalen Entwicklung mit `ng serve`). Bei produktiv kompilierten Anwendungen werden die für die DevTools notwendigen Debug-Informationen entfernt, um die Performance zu optimieren.

1.6 Codebeispiele in diesem Buch

Wir entwickeln in diesem Buch eine praktische Anwendung. Die dazugehörigen Projekte haben wir zentral zur Verfügung gestellt. Auf der folgenden Website findest du eine Übersicht über alle Zwischenstände des Beispielprojekts *BookManager*:

Quelltext, Online-Demo und Differenzansicht:
https://bm1.angular-buch.com

Zu jedem Praxiskapitel findest du dort den Quellcode, eine Live-Demo, eine Differenzansicht und einen Stackblitz-Link zum Experimentieren. Alle Projekte sind auf der Entwicklungsplattform GitHub gehostet, sodass du den Code bei Bedarf herunterladen und lokal bearbeiten kannst. Du kannst das Repository entweder mit Git klonen oder den gesamten Quelltext als ZIP-Archiv herunterladen.

[10] *https://ng-buch.de/d/ng-devtools* – Angular Docs: DevTools Overview

2 Angular CLI: der Codegenerator für unser Projekt

Eine »rohe« Angular-Anwendung braucht verhältnismäßig viele Vorbereitungen, die wir nicht per Hand ausführen möchten. Angular bringt dafür ein ausgereiftes Tool mit, das uns bei der Arbeit mit unseren Projekten durchgehend unterstützt: die *Angular CLI*. In diesem Kapitel werden wir die Installation und die wichtigsten Befehle des Kommandozeilentools kennenlernen.

2.1 Das offizielle Tool für Angular

Die Angular CLI ist das offizielle Werkzeug für unsere Angular-Anwendungen. Das Tool stellt Vorlagen und Befehle für alle wiederkehrenden Aufgaben bereit, vom Anlegen eines Projekts über Codegenerierung und einen Entwicklungswebserver bis hin zur Ausführung von Unit-Tests und zum finalen Deployment. Der generierte Code orientiert sich stets am offiziellen Styleguide von Angular.[1] Somit folgen alle Angular-Projekte einer konsistenten Struktur und nutzen dasselbe Tooling.

Beim Anlegen eines neuen Projekts wird alles Nötige vorbereitet: Alle Dateien werden angelegt, NPM-Pakete werden installiert und das Projekt wird unter Versionsverwaltung (Git) gestellt. Während der Entwicklung können wir die Grundgerüste für unsere Komponenten, Services, Pipes und weitere Features automatisch generieren lassen. Wiederkehrende Aufgaben wie das Bauen des Projekts oder die Ausführung von Unit-Tests sind bereits eingerichtet. Auch ein Webserver für die Entwicklung wird mitgeliefert, sodass wir das Projekt ohne zusätzliche Abhängigkeiten direkt auf dem lokalen System ausführen können. Eine Überwachung

[1] *https://ng-buch.de/d/ng-styleguide* – Angular Docs: Style Guide

des Dateisystems (Watch-Modus) sorgt dafür, dass die Anwendung bei jedem Speichervorgang automatisch im Browser aktualisiert wird. Außerdem werden Skripte angeboten, um eine Anwendung automatisch zu aktualisieren oder andere Pakete in das Projekt zu integrieren. Kurz: Die Angular CLI unterstützt uns durch die Automatisierung bei allen Routineaufgaben rund um unser Angular-Projekt. Wir wollen das Tool nun einrichten und verwenden!

2.2 Installation

Um die Angular CLI nutzen zu können, sollten wir das Tool als globales NPM-Paket installieren:

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

Danach können wir die Angular CLI mit dem Befehl `ng` auf der Kommandozeile ausführen. Um sicherzustellen, dass alles geklappt hat, können wir uns zum Beispiel die installierte Version oder die Hilfe mit der Auflistung aller verfügbaren Befehle anzeigen lassen:

```
$ ng version # Anzeige der Versionen
$ ng help    # Hilfe mit Auflistung der Befehlsreferenz
```

Wir werden im Verlauf dieses Buchs mehrere Befehle und Codegeneratoren der Angular CLI kennenlernen.

Analytics

Sobald wir das Kommando `ng` zum ersten Mal ausführen, bittet die Angular CLI um Erlaubnis für die Analytics: Aktivieren wir diese Funktion, werden anonymisierte Nutzungsdaten an das Angular-Team bei Google übermittelt. Die Informationen sollen zur Verbesserung von Angular verwendet werden. Welche Daten erhoben werden, ist in der Dokumentation beschrieben.[2]

Ob du dieses Feature aktivierst oder auf die Datensammlung verzichten möchtest, darfst du selbst entscheiden. Du kannst die Einstellung jederzeit ändern und die Analytics so auch nachträglich ein- oder ausschalten:

[2] *https://ng-buch.de/d/ng-analytics* – Angular Docs: ng analytics

```
$ ng analytics info     # Aktuellen Status anzeigen
$ ng analytics disable  # Analytics deaktivieren
$ ng analytics enable   # Analytics aktivieren
```

Listing 2.1
Analytics konfigurieren

2.3 Autovervollständigung einrichten

Die Angular CLI bietet eine automatische Befehlsvervollständigung für die Shell an. Um diese Funktion einzurichten, müssen wir einmalig dieses Kommando ausführen:

```
$ ng completion
```

Dadurch wird die Konfiguration der Shell ergänzt (z. B. die Datei `.bashrc` oder `.zshrc`), sodass für die Autovervollständigung die Angular CLI aufgerufen wird. Im Anschluss müssen wir das Terminal einmal neu starten.

Tippen wir jetzt in der Kommandozeile den Befehl `ng` ein und drücken Tab, erhalten wir automatisch passende Vorschläge. Die Autovervollständigung bezieht sich dabei immer auf den aktuellen Kontext: Zum Beispiel liefert sie für `ng generate` (oder kurz: `ng g`) alle möglichen Generatoren, die im aktuellen Projekt verfügbar sind.

Du kannst die Angular CLI auch ohne die Autovervollständigung bedienen. Wir empfehlen jedoch, das Feature zu aktivieren – es verbessert die Developer Experience enorm!

3 Syntax & Konzepte: Grundlagen für modernes Angular

Bevor wir mit Angular starten, wollen wir in diesem Kapitel einige Features und Sprachelemente von TypeScript und modernem JavaScript besprechen. Alle diese Konzepte werden wir im Verlauf des Buchs praktisch einsetzen.

3.1 TypeScript

Für die Entwicklung mit Angular verwenden wir die Programmiersprache *TypeScript*.[1] Die Sprache wurde von Microsoft entwickelt und implementiert die neuesten Sprachelemente aus dem Standard *ECMAScript*, auf dem JavaScript basiert. Wir können also sämtliches Wissen zu JavaScript auch in einem TypeScript-Programm einsetzen.

Zusätzlich bietet TypeScript ein statisches Typsystem. Das bedeutet praktisch, dass die Typen von Variablen, Funktionsparametern und Klassen-Propertys direkt im Code aufgeschrieben werden. So erhalten wir schon während der Entwicklung eine gute Unterstützung im Editor und können unsere Software typsicher entwickeln. Autovervollständigung, Typprüfungen und Rename Refactoring sind nur einige der Vorteile, die sich aus einem statischen Typsystem ergeben.

Angular setzt seit den frühen Tagen auf TypeScript. Solltest du noch keine Erfahrung mit der Sprache haben, empfehlen wir einen Blick in unser Online-Kapitel zu TypeScript:

https://angular-buch.com/material/typescript

Die wichtigsten Sprachelemente, die man für die Entwicklung mit Angular benötigt, stellen wir dir im Folgenden vor.

[1] *https://ng-buch.de/d/typescript* – TypeScript

3.2 Template Strings

Mit einem normalen String in einfachen Anführungszeichen ist es nicht möglich, einen Text über mehrere Zeilen anzugeben. Um das zu ermöglichen, können wir einen *Template-String* einsetzen: Er wird mit schrägen `Hochkommata` (*Gravis* oder *Backtick* genannt) eingeleitet und beendet, nicht mit Anführungszeichen. Der String kann sich schließlich über mehrere Zeilen erstrecken und endet erst beim schließenden Backtick.

Mit Template-Strings können wir außerdem Ausdrücke direkt in einen String einbetten.

Listing 3.1 *Template-String mit eingebetteten Ausdrücken*

```
const text = `Mein Name ist ${firstname}.
Ich bin ${age} Jahre alt.`;

const url = `${apiBaseUrl}/user/${id}/friends?page=${page}`;
```

Wir werden Template-Strings vor allem nutzen, um URLs mit Parametern zusammenzubauen. Außerdem können wir die Syntax auch in den Template Expressions von Angular verwenden, z. B. im Property Binding – dazu später mehr.

3.3 Arrow Functions

Eine *Arrow-Funktion* ist eine kompakte Alternative zur klassischen `function()` in JavaScript. Auch die Bezeichnung *Lambda-Ausdruck* ist verbreitet.

Die Definition einer anonymen Funktion verkürzt sich damit elegant zu einem Pfeil =>. Besitzt die Funktion genau einen Parameter ohne Typ, können die runden Klammern auf der linken Seite sogar weggelassen werden. Auch die geschweiften Klammern auf der rechten Seite können eingespart werden: Lässt man die Klammern weg, ist das Ergebnis des rechtsseitigen Ausdrucks der Rückgabewert für die Funktion. Wir müssen also kein `return`-Statement verwenden. Diese vier Definitionen sind gleichwertig:

```
function (foo) { return foo + 1; }

(foo) => { return foo + 1; }

foo => { return foo + 1; }

foo => foo + 1;
```

Damit können anonyme Funktionen mit nur wenig Tipparbeit definiert werden. Das folgende Beispiel zeigt, wie wir alle geraden Zahlen aus einer Liste ermitteln können. Zuerst wird eine herkömmliche Funktionsdeklaration verwendet. Die Arrow-Funktion im zweiten Beispiel ist allerdings wesentlich kompakter: Es ist möglich, einen einzeiligen Ausdruck einzusetzen, um die Funktion zu deklarieren.

Listing 3.2
Callback-Funktion verwenden

```
const numbers = [0, 1, 2, 3, 4];

const even1 = numbers.filter(
  function(value) {
    return value % 2 === 0;
  }
);

const even2 = numbers.filter(
  value => value % 2 === 0
);
```

Bei komplexerer Logik kann auch ein mehrzeiliger Block verwendet werden. In diesem Fall muss der Rückgabewert allerdings mit `return` aus der Funktion herausgegeben werden.

Ein weiterer Vorteil der Arrow-Funktion ist, dass sie keinen eigenen `this`-Kontext besitzt. Das ist besonders dann interessant, wenn wir die Funktion innerhalb einer Klasse verwenden und mit `this` auf die Instanz der Klasse zugreifen möchten. Mit einer normalen `function()` ist das nicht ohne Mehraufwand möglich, denn `this` verweist dort nicht automatisch auf die Klasseninstanz. Mit Arrow-Funktionen hingegen wird die Variable `this` aus dem übergeordneten Kontext verwendet. Definieren wir etwa eine Arrow-Funktion innerhalb einer Klasse, so zeigt `this` auf die Klasseninstanz. Dies entspricht genau dem, was wir beim Programmieren mit Klassen üblicherweise benötigen.

Listing 3.3
In einer Arrow-Funktion zeigt this *auf den übergeordneten Kontext, z. B. die Klasse.*

```
class User {
  name: string;
  // ...

  showFriends() {
    this.friends.forEach(friend => {
      console.log(`${this.name} kennt ${friend.name}.`);
    });
  }
}
```

3.4 Immutability

In JavaScript werden Objekte und Arrays stets nur als Referenzen auf eine zugehörige Speicherstelle gespeichert. Ändern wir also die Inhalte direkt im Objekt, so ändert sich die Referenz nicht! Das bedeutet auch, dass bei der Zuweisung eines Objekts zu einer Variable lediglich ein Verweis auf das ursprüngliche Objekt erzeugt wird. Dadurch erhalten wir einen oft unerwünschten Nebeneffekt: Ändern wir einen Wert im neu zugewiesenen Objekt, so ändert sich auch der Wert im ursprünglichen Objekt.

Listing 3.4 *Es wird nur die Objektreferenz kopiert.*

```
const book = { title: 'Angular', year: 2016 };

const copy = book;
copy.year = 2026;

console.log(copy === book); // true
console.log(copy); // { title: 'Angular', year: 2026 }
console.log(book); // { title: 'Angular', year: 2026 }
```

Vergleichen wir also zwei Variablen mit Objekten, werden nur die Referenzen gegeneinander geprüft. Diese sagen jedoch nichts über den Inhalt der Objekte aus. Um sicher festzustellen, dass sich ein Objekt geändert hat, ist es gute Praxis, stets ein neues Objekt mit neuer Referenz zu erzeugen. So kann man beim direkten Vergleich der Referenzen sofort erkennen, dass es sich um unterschiedliche Objekte handelt.

Signals von Angular, die wir in Kapitel 5 ab Seite 61 kennenlernen werden, erkennen Änderungen nur, wenn sie einen komplett neuen Wert erhalten.

Um gut wartbaren Code zu erhalten, dürfen wir niemals die Werte eines Objekts oder Arrays direkt verändern. Wir behandeln ein Objekt oder Array als *unveränderlich* (engl. *immutable*) und erzeugen bei einer Änderung immer eine Kopie. Hierfür nutzen wir in der Regel die Spread-Syntax, die wir im nächsten Abschnitt kennenlernen werden.

Merke: Objekte und Arrays sollten nie direkt verändert werden. Stattdessen sollte immer eine Kopie mit neuer Referenz erzeugt werden, die die gewünschten Änderungen enthält.

3.5 Spread-Syntax und Rest-Parameter

In JavaScript können wir eine Syntax mit drei Punkten verwenden (`...`). Diese Schreibweise hat zwei Bedeutungen, je nachdem, wo sie eingesetzt wird: Die *Spread-Syntax* breitet Elemente aus, während *Rest-Parameter* übrige Argumente einsammeln.

Mithilfe der Spread-Syntax können Teile eines Objekts oder Arrays expandiert werden, um

- mehrere Eigenschaften in ein Objekt zu kopieren,
- mehrere Elemente in ein Array zu kopieren oder
- mehrere Argumente bei Funktionsaufrufen anzugeben.

Praktisch verwenden wir die Spread-Syntax vor allem, um Objekte und Arrays zu klonen. Das wollen wir uns an ein paar Beispielen anschauen.

Objekteigenschaften kopieren

Wir erinnern uns noch einmal an die Problemstellung aus dem letzten Abschnitt (Listing 3.4 auf Seite 36): Dort existiert das Objekt `book` einmalig im Speicher. Wir wollen davon eine Kopie erzeugen und die Kopie verändern. Das originale Objekt soll dabei unverändert bleiben, denn wir wollen die Regeln der Immutability berücksichtigen.

Auf den ersten Blick scheint diese Aufgabe einfach zu sein: Wir weisen das Objekt einer neuen Variable `copy` zu und ändern die Eigenschaft `year`. Bei der Ausgabe sehen wir allerdings, dass diese Herangehensweise nicht funktioniert.

Die Ursache hier haben wir bereits zuvor kennengelernt: Die Variable enthält nur die *Referenz* auf das Objekt, nicht das Objekt selbst. Die beiden Variablen `book` und `copy` zeigen also auf dieselbe Speicherstelle. Ändern wir etwas an den Daten, wird das originale Objekt im Speicher verändert.

Mit der Spread-Syntax können wir dieses Problem elegant lösen:

```
const book = { title: 'Angular', year: 2016 };

const copy = { ...book, year: 2026 };

console.log(copy); // { title: 'Angular', year: 2026 }
console.log(book); // { title: 'Angular', year: 2016 }
```

Wir initialisieren die Variable `copy` mit einem neuen (leeren) Objekt. Anschließend kopieren wir mit der Spread-Syntax `...` alle Eigenschaften von `book` in das neue Objekt. Im letzten Schritt setzen wir die Eigenschaft `year` auf den Wert `2026`. Existiert die Eigenschaft bereits, wird sie überschrieben. Damit haben wir die Eigenschaften von `book` in ein anderes Objekt expandiert. Praktisch wurde also der Inhalt des Objekts kopiert.

Bitte beachte, dass diese Idee nur für *Plain Objects* funktioniert. Als Grundlage erzeugen wir immer ein leeres untypisiertes Objekt `{ }`, in das wir die Eigenschaften eines anderen Objekts kopieren. »Klonen« wir also auf diese Weise ein Objekt, das eine Instanz einer Klasse ist, so werden nur die Eigenschaften kopiert, nicht aber die klasseneigenen Methoden. Die inhaltliche Verbindung mit der Klasse geht verloren, und es wird lediglich eine flache Kopie (engl. *Shallow Copy*) erzeugt.

Diese Eigenschaft ist vor allem interessant, wenn wir es mit komplexeren Objekten zu tun haben: Es wird stets nur die obere Ebene eines Objekts kopiert. Tiefere Zweige eines Objekts oder Arrays müssen wir zunächst mit der Spread-Syntax einzeln klonen und anschließend neu zusammenbauen.

Wird diese Aufgabe zu kompliziert, können wir die native Funktion `structuredClone()`[2] verwenden. Sie erzeugt eine *Deep Copy*, indem sie alle Zweige des Objekts vollständig kopiert. Das ist in den meisten Fällen aber nicht notwendig und sollte eine Ausnahme bleiben.

Array-Elemente kopieren

Die Spread-Syntax funktioniert ähnlich auch für Arrays. Wir können damit die Elemente eines Arrays in ein anderes Array kopieren. Dieses Feature können wir nutzen, um eine Kopie eines Arrays zu erzeugen, aber auch, um mehrere Arrays zusammenzufügen.

```
const numbers = [1, 2, 3, 4, 5];

const numbers1 = [...numbers, 6];
const numbers2 = [...numbers, 0, ...numbers];

console.log(numbers1); // [1, 2, 3, 4, 5, 6]
console.log(numbers2); // [1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5]
```

[2] *https://ng-buch.de/d/structuredclone* – MDN: structuredClone()

Funktionsargumente übergeben

Die Spread-Syntax lässt sich nicht nur in Arrays einsetzen, sondern auch für Funktionsargumente. Wollen wir die Elemente eines Arrays einzeln als Argumente an eine Funktion übergeben, können wir also die Spread-Syntax nutzen.

```
const args = [5, 'foo', true];

doThings(args[0], args[1], args[2]);

doThings(...args);
```

Funktionsargumente einsammeln

In einem anderen Kontext haben die drei Punkte eine andere Bedeutung: Erhält eine Funktion mehrere Argumente, so können wir diese elegant in einem Array erfassen. Ein solcher Parameter heißt *Rest-Parameter*, weil er den »Rest« der übergebenen Argumente aufnimmt, also alle übrig gebliebenen. Die Schreibweise mit `...` ist identisch zur Spread-Syntax, aber der Effekt ist umgekehrt: Anstatt ein Array zu expandieren, führen wir alle übergebenen Argumente in einem Array zusammen.

```
function doThings(...args) {
  console.log(args); // [5, 'foo', true]
}
```

3.6 Private Class Fields

Mit *Private Class Fields* in JavaScript können wir Datenkapselung in Klassen realisieren. Ein privates Feld wird durch ein vorangestelltes #-Symbol definiert und ist nur innerhalb der Klasse zugänglich, in der es definiert wurde. Ein direkter Zugriff von außen ist nicht möglich.

In TypeScript existiert außerdem der Access Modifier `private`, der die Sichtbarkeit von Propertys und Methoden einschränkt. Der Schutz ist allerdings zur Laufzeit nicht garantiert, da TypeScript zu JavaScript umgewandelt wird und das Schlüsselwort `private` dort nicht existiert. Wir haben uns in diesem Buch für die moderne JavaScript-Variante entschieden, um private Propertys und Methoden von Klassen abzubilden.

Listing 3.5 *Private Class Fields in JavaScript und TypeScript*

```
class MyClass {
  #myProp = 'Angular';

  #doThings(param: string): void {
    console.log(this.#myProp);
  }

  private myProp = 'Angular';
  private doThings(param: string): void { /* ... */ }
}
```

3.7 Property Modifiers: `readonly` und `protected`

TypeScript stellt uns eine Reihe von sog. *Property Modifiers* zur Verfügung, mit denen wir die Sichtbarkeit und den Zugriff auf Eigenschaften und Methoden von Klassen steuern können. Im Laufe des Buchs werden wir vor allem zwei Modifier verwenden:

Der Modifier `protected` sorgt für eine eingeschränkte Sichtbarkeit von Propertys und Methoden. Er ist ein Kompromiss zwischen privater und öffentlicher Sichtbarkeit: Ein Protected Property ist nicht von außen sichtbar, sondern kann nur innerhalb derselben Klasse und in vererbten Kindklassen verwendet werden. Dazu gehört auch das Template einer Angular-Komponente.

Mit `readonly` können wir sicherstellen, dass eine Eigenschaft einer Klasse nach der Initialisierung nicht mehr verändert werden kann. Eine mit `readonly` deklarierte Eigenschaft kann nur einmalig mit einem Wert beschrieben werden. Die Zuweisung muss entweder direkt bei der Property-Initialisierung oder im Konstruktor erfolgen. Der Compiler wertet jede nachträgliche Zuweisung als Fehler, das Programm lässt sich so nicht mehr übersetzen. Das Verhalten ist vergleichbar mit einer Variable, die mit `const` erzeugt wurde. Grundsätzlich lässt sich `readonly` auch mit einem anderen Modifier kombinieren. Es ist also möglich, ein Property mit `protected readonly` zu markieren.

Listing 3.6 *Propertys mit dem Modifier* `readonly`

```
export class MyClass {
  readonly book = { title: 'Angular', year: 2016 };

  updateBook() {
    // FEHLER:
    // Cannot assign to 'book' because it is a read-only
    this.book = { title: 'Angular', year: 2026 };
  }
}
```

Angular definiert in seinem *Coding Style Guide*[3] mehrere Regeln zu Property Modifiers. Wir haben daraus für dieses Buch die folgenden Konventionen abgeleitet:

- Propertys und Methoden, die nur innerhalb der Klasse verwendet werden, werden grundsätzlich als privat markiert. Das betrifft insbesondere injizierte Services.
- Propertys, die im Template einer Komponente genutzt werden, werden mit `protected` gekennzeichnet. Das gilt vor allem für Signals.
- Propertys, die von Angular verwaltet werden, werden auf `readonly` gesetzt. Das betrifft Propertys mit `input()`, `output()`, `model()` und Querys (`viewChild()`, `contentChild()`).

Wir werden diese Konventionen im Verlauf des Buchs aufgreifen, wenn wir die genannten Themen behandeln.

Diese Empfehlungen sind übrigens nur eine Leitlinie und keine strikte Pflicht. Wir halten uns in diesem Buch an die offiziellen Empfehlungen von Angular. Bei Bedarf kannst du in deinen Projekten aber andere Konventionen vereinbaren.

3.8 Decorators

Mit Decorators können wir Klassen, Methoden und Eigenschaften dekorieren und damit Metadaten hinzufügen. Man erkennt einen Decorator am @-Zeichen zu Beginn des Namens.

```
@Component({
  // Objekt mit Konfiguration
})
export class MyComponent { }
```

Angular nutzt dieses Sprachkonzept, um Klassen eine Semantik zu geben. Durch den Decorator `@Component()` wird eine Klasse z. B. als Komponente behandelt. Alle Decorators von Angular sind Funktionen, daher darf man die Funktionsklammern bei der Verwendung nicht vergessen.

[3] *https://ng-buch.de/d/ng-styleguide* – Angular Docs: Style Guide

3.9 Generic Types

Mit *Generics* können wir Typparameter für Funktionen, Klassen, Interfaces und Type Aliases definieren. Sie sind ein wichtiges Konzept in TypeScript, um wiederverwendbaren und flexiblen Code zu erstellen: Dazu wird ein generischer Typ `T` eingesetzt, der erst bei der konkreten Verwendung festgelegt werden muss. So kann derselbe Code mit verschiedenen Typen arbeiten, ohne dass diese bei der Implementierung schon bekannt sein müssen.

In Angular verwenden wir Generics vor allem bei der Erzeugung von Signals oder in Services, die Daten über HTTP abrufen. Das folgende Beispiel zeigt, wie wir Generics in einer Funktion verwenden können: Die Signatur der Funktion `signal()` arbeitet mit einem generischen Typ `T`. Übergeben wir an das Argument `initialValue` einen Wert, wird daraus automatisch der Typ ermittelt. Einfache Typen wie `string` oder `number` können problemlos erkannt werden. Übergeben wir ein Objekt, erkennt TypeScript die Struktur, ordnet sie aber nicht automatisch einem benannten Interface zu. Wollen wir explizit den Typ `Book` verwenden, müssen wir die Typinformation manuell in spitzen Klammern angeben.

Listing 3.7
Generic Types in einer Funktion verwenden

```
// Signatur der Funktion
function signal<T>(initialValue: T): WritableSignal<T>

interface Book {
  title: string;
  year: number;
}

// Verwendung:
// WritableSignal<string>
const headline = signal('Hello world');

// WritableSignal<{ title: string; year: number }>
const book1 = signal({ title: 'Angular', year: 2026 });

// WritableSignal<Book>
const book2 = signal<Book>({ title: 'Angular', year: 2026 });
```

Wir werden diese Schreibweise später noch ausführlicher im Zusammenhang mit Signals kennenlernen.

3.10 Promises und async/await

Eine *Promise* ist ein natives Objekt in JavaScript, das einen asynchronen Vorgang repräsentiert. Sie liefert entweder einen Wert zurück, wenn die Operation erfolgreich war, oder einen Fehler, wenn die Ausführung fehlgeschlagen ist. Das Ergebnis einer Promise kann mit der Methode `then()` verarbeitet werden. Geben wir aus `then()` einen Wert oder eine Promise zurück, wird das Ergebnis automatisch in eine neue Promise verpackt – so lassen sich mehrere asynchrone Operationen verketten.

Mit den Schlüsselwörtern `async` und `await` können wir asynchronen Code schreiben, der wie synchroner Code aussieht. Der Code wird dabei aber weiterhin asynchron ausgeführt. Die synchrone Schreibweise ist lediglich »syntaktischer Zucker«, der die Lesbarkeit verbessert. Dazu markieren wir eine Funktion mit `async`. Innerhalb dieser Funktion können wir dann `await` vor einer Promise verwenden, um auf das Ergebnis zu warten. Verkettete `then()`-Aufrufe sind dann nicht mehr notwendig.

Promises werden ohne externe Bibliotheken direkt vom Browser unterstützt. Zum Beispiel arbeitet die native Fetch API mit Promises, wie Listing 3.8 zeigt. Auch Angular setzt für asynchrone Operationen an verschiedenen Stellen auf Promises.

Listing 3.8
Promises verarbeiten mit then() *oder* async/await

```
// Promise mit then()
function loadData() {
  fetch('https://example.org')
    .then(response => response.json())
    .then(data => console.log(data));
}

// Promise mit async/await
async function loadData() {
  const response = await fetch('https://example.org');
  const data = await response.json();
  console.log(data);
}
```

Eventbasierte Szenarien werden in Angular häufig mit Observables aus der Bibliothek RxJS umgesetzt. Ein Observable repräsentiert einen Datenstrom, es unterscheidet sich also konzeptionell von der Promise. Wir werden in Kapitel 31 zu RxJS ab Seite 412 noch genauer auf diese Unterschiede eingehen.

Teil II

BookManager:
Schritt für Schritt zur App

4 Projektvorstellung und Einrichtung

Wir wollen uns nun dem größten Abschnitt dieses Buchs widmen und Angular kennenlernen. Dazu wollen wir ein umfangreiches Beispielprojekt entwickeln. Dabei werden wir alle wichtigen Konzepte des Frameworks behandeln und praktisch anwenden.

4.1 Unser Projekt: BookManager

Unser Projekt ist eine Webanwendung, mit der wir Bücher verwalten können: der *BookManager*. Wir wollen Bücher in einer Übersicht und auf einer Detailseite darstellen. Ebenso wollen wir nach Büchern suchen können und interessante Bücher auf einer Favoritenliste sammeln. Außerdem wollen wir Möglichkeiten schaffen, den Buchbestand selbst zu verwalten, indem wir neue Einträge über ein Formular erstellen. Die Sprache der Anwendung ist Englisch. Wir haben einige Skizzen angefertigt, an denen wir uns orientieren werden.

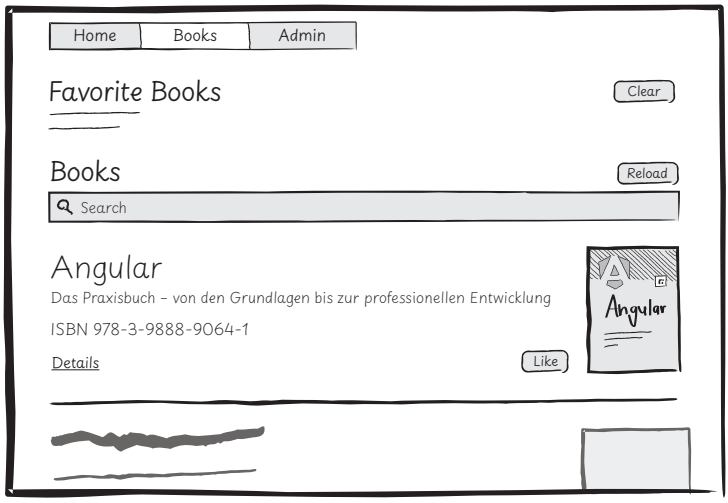


Abb. 4.1
Skizze der Listenansicht

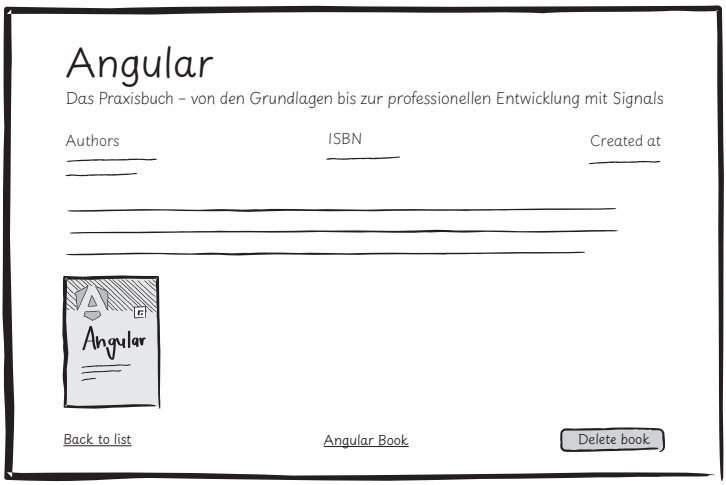


Abb. 4.2
Skizze der Detailansicht

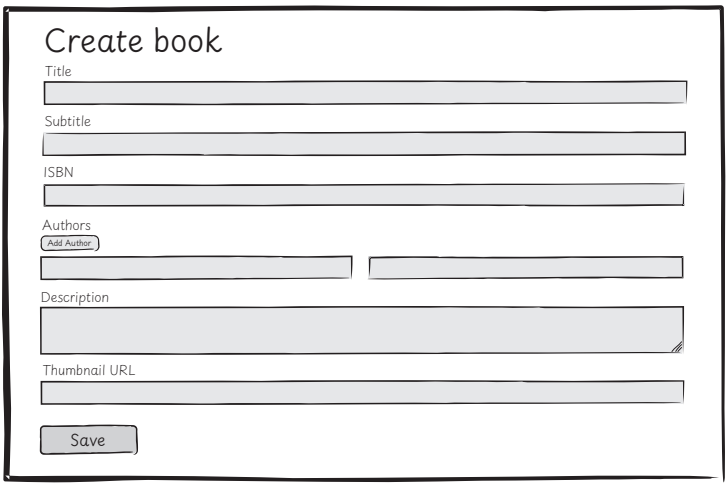


Abb. 4.3
Skizze des Eingabeformulars

4.2 Struktur und Quellcode

In den folgenden Kapiteln werden wir den BookManager Schritt für Schritt erweitern und verändern. Das Projekt wächst damit von einem einfachen Beispiel zu einer vollwertigen Anwendung. Wir haben den BookManager in 15 Einzelschritte unterteilt, in denen wir jeweils ein eigenständiges Thema betrachten. Wir erläutern zunächst immer die theoretischen Grundlagen, bevor wir im Anschluss das Gelernte im Projekt in einem oder mehreren Schritten umsetzen.

Für jeden Teilschritt stellen wir eine Online-Demo zur Verfügung. Den gesamten Quelltext des Projekts findest du auf GitHub, sodass du jeden Schritt einzeln ausprobieren kannst.

Quelltext, Online-Demo und Differenzansicht:
https://bm1.angular-buch.com

Gelegentlich wird es vielleicht passieren, dass eine wichtige Zeile Quelltext vergessen wurde und ein Teilschritt dadurch nicht funktioniert. Als Unterstützung haben wir die Änderungen zwischen den einzelnen Praxiskapiteln als Differenzansicht bereitgestellt. So kannst du detailliert nachvollziehen, was sich verändert hat.

Abb. 4.4
Differenzansicht für den BookManager

Angular wird aktiv weiterentwickelt, und mit neuen Versionen des Frameworks können Änderungen am Code des BookManager notwendig sein. Wir halten den Code auf GitHub und die Differenzansichten stets aktuell, sodass du die Änderungen überprüfen und mit dem gedruckten Stand vergleichen kannst.

Digitale Barrierefreiheit

Bei der Konzeption des Beispielprojekts haben wir großen Wert darauf gelegt, die Anwendung für möglichst viele Menschen ohne Barrieren zugänglich zu machen. In den HTML-Templates werden wir deshalb Themen wie semantisches HTML und ARIA-Attribute bzw. entsprechende Rollen von Anfang an berücksichtigen.

4.3 Projekt mit der Angular CLI initialisieren

Möchten wir ein neues Projekt beginnen, so generieren wir dieses mit dem Befehl `ng new`. Die Angular CLI wird uns mithilfe einer Eingabeaufforderung einige Fragen stellen. Mit den Pfeiltasten können wir eine Auswahl treffen und mit der ↵-Taste bestätigen. Alle notwendigen Dateien und Ordner werden dadurch automatisch in einem eigenen Unterordner generiert. Ebenso werden die notwendigen NPM-Pakete mithilfe von `npm install` installiert. Alle abgefragten Einstellungen können wir übrigens auch als Argument an den Befehl anhängen.

Um die Grundstruktur unseres Projekts anzulegen, geben wir den folgenden Befehl in der Konsole ein.

Listing 4.1 *Neues Projekt BookManager anlegen mit der Angular CLI*

```
$ ng new book-manager --style=css --no-ssr
```

Der Name unseres Projekts lautet `book-manager`. Die verwendeten Kommandozeilenparameter bedeuten Folgendes:

- `--style=css`: Legt das Stylesheet-Format fest, mögliche Optionen sind: `css`, `scss`, `sass`, `less`, `tailwind`. Wir haben uns hier für den Standard CSS entschieden.
- `--no-ssr`: Sorgt dafür, dass kein zusätzlicher Quellcode zur Umsetzung von Server-Side Rendering (SSR) generiert wird.[1]

Die Angular CLI bietet Unterstützung für die Arbeit mit Artificial Intelligence (AI) an: Der interaktive Prompt fragt nach, ob eine Konfiguration für ein bestimmtes AI-Werkzeug generiert werden soll. Alternativ können wir auch direkt die Kommandozeilenoption `--ai-config` verwenden. Dadurch werden Dateien erzeugt, die von AI-Assistenten wie Claude Code, GitHub Copilot, Cursor und vielen weiteren eingelesen werden. Sie enthalten Informationen, die diese Tools mit aktuellen Angular Best Practices versorgen. Du kannst diese Option auch überspringen und die Konfiguration bei Bedarf später generieren. Wir widmen uns dem Thema AI-Unterstützung noch einmal detailliert in Kapitel 34 ab Seite 479.

[1] Server-Side Rendering ist ein fortgeschrittenes Konzept, das wir in diesem Buch nicht behandeln. Wir haben ein Online-Kapitel zu dem Thema bereitgestellt: *https://angular-buch.com/material/ssr*.

4.4 Aufbau des neuen Projekts

Als Nächstes navigieren wir in den neu angelegten Projektordner 🗀`book-manager`.

```
$ cd book-manager
```

Listing 4.2
Ordner wechseln

Außerdem öffnen wir den Ordner im Editor, sodass wir die Dateien bearbeiten können.

Angular Language Service installiert?

Wenn du Visual Studio Code nutzt, stelle bitte sicher, dass du den *Angular Language Service* installiert hast. Diese Editor-Erweiterung sorgt für korrektes Syntax-Highlighting und Typprüfung in den Angular-Templates, siehe Seite 25.

Im Editor schauen wir uns die generierte Ordnerstruktur genauer an:

- 🗀`book-manager`
  - 🗀`.angular` ..... Verzeichnis für Cache und temporäre Dateien
  - 🗀`.vscode` .....Konfigurationen für den Editor Visual Studio Code
  - 🗀`node_modules` ..... installierte NPM-Pakete
  - 🗀`public` ..... Verzeichnis für statische Dateien (Bilder etc.)
  - 🗀`src` ..... beinhaltet den Code unserer Anwendung
  - `.editorconfig` .....Konfiguration für den Codeeditor
  - `.gitignore` ..... von Git ignorierte Dateien und Ordner
  - `.prettierrc` ..... Konfiguration für Codeformatierung mit Prettier
  - `angular.json` .....Konfigurationsdatei der Angular CLI
  - `package-lock.json` .....Lockfile für NPM-Paketinformationen
  - `package.json` ..... Konfiguration des NPM-Projekts
  - `README.md` .....Textdatei zur Hilfe
  - `tsconfig.app.json` .. TypeScript-Konfiguration für die Anwendung
  - `tsconfig.json` ..... projektweite TypeScript-Konfiguration
  - `tsconfig.spec.json` ..... TypeScript-Konfiguration für das Testing

Die Angular CLI legt sofort ein neues Git-Repository an[2] und erstellt alle notwendigen Konfigurationen, sodass wir direkt mit der Entwicklung der Anwendung beginnen könnten. Einige der Konfigurationsdateien wollen wir vorher näher betrachten.

[2] Die Initialisierung des Git-Repositorys kann mit dem Argument `--skip-git` bei `ng new` übersprungen werden. Ein Repository wird nur angelegt, wenn Git auf dem System installiert ist und wenn das Arbeitsverzeichnis noch nicht unter Git-Versionsverwaltung steht.

Datei oder Verzeichnis nicht gefunden?

Durch die stetige Weiterentwicklung der Angular CLI kann es bei neueren Versionen dazu kommen, dass die Konfigurationsdateien für neu angelegte Projekte ggf. entfernt wurden, andere Namen tragen oder sich an einem anderen Ort befinden.

🗀 `.angular`

Das Verzeichnis 🗀 `.angular` ist nach dem Anlegen des Projekts noch nicht vorhanden. Es enthält einen Build-Cache und wird automatisch erzeugt, sobald wir die Anwendung starten. Bei Build-Problemen kann es gegebenenfalls hilfreich sein, das Verzeichnis zu löschen. Es wird beim nächsten Durchlauf neu erzeugt.

`angular.json`

Die Datei `angular.json` beinhaltet die zentrale Konfiguration für unser Angular-Projekt. Hier finden wir unter anderem die Einstellungen wieder, die wir beim Erzeugen des Projekts gesetzt haben, z. B. einen Verweis auf ein global angelegtes CSS-Stylesheet.

Außerdem können wir in der `angular.json` alle Parameter für den Build der Anwendung konfigurieren. Für den Einstieg in Angular ist diese Datei zunächst noch nicht wichtig.

`package.json` und `package-lock.json`: Konfiguration für NPM

In der Datei `package.json` befindet sich die Konfiguration für den Node Package Manager (NPM). Sie enthält Metadaten für das Projekt, unter anderem Name, Versionsnummer und abhängige Pakete.

Im Abschnitt `dependencies` werden alle NPM-Pakete mit den entsprechend installierten Versionen gelistet, die direkt von der Anwendung verwendet werden. Dazu zählt neben Angular auch die Bibliothek RxJS. In `devDependencies` hingegen erscheinen die Pakete, die für die Entwicklung des Projekts benötigt werden. Dies sind zum Beispiel der TypeScript-Compiler und die Angular CLI.

Die Datei `package-lock.json` wird automatisch vom NPM erzeugt. Hier wird festgehalten, welche NPM-Pakete mit welcher exakten Version installiert wurden. Die Datei wird automatisch aktualisiert, wenn Pakete hinzugefügt, entfernt oder aktualisiert werden. Sie sollte unbedingt bei einer Änderung mit versioniert werden, da sie dafür sorgt, dass nach dem Auschecken des Projekts und einer anschließenden Neueinrichtung mittels `npm install` stets wieder der exakt gleiche Abhängigkeitsbaum der Pakete aufgebaut wird.

Bitte beachte, dass Angular nicht mit jeder Version von Node.js kompatibel ist. In der Dokumentation von Angular[3] findest du aktuelle Infos, welche genauen Versionen von Node.js und TypeScript unterstützt werden.

`tsconfig.json`: Konfiguration für TypeScript

Wenn die Angular CLI den Build des Projekts durchführt, wird intern der TypeScript-Compiler ausgeführt, um den TypeScript-Code in JavaScript umzuwandeln. Die Einstellungen für den TypeScript-Compiler werden in unterschiedlichen Konfigurationsdateien festgelegt: Die Datei `tsconfig.json` enthält die Grundeinstellungen. Darauf aufbauend verwendet Angular separate Dateien für den Bau der Anwendung (`tsconfig.app.json`) und für die Ausführung der Tests (`tsconfig.spec.json`).

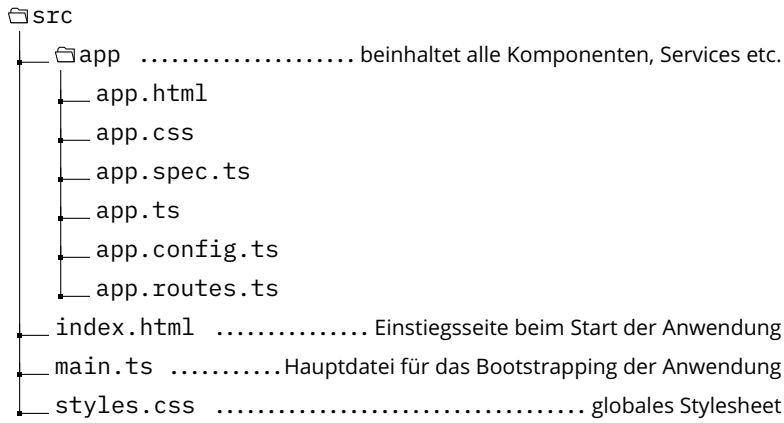
In der Regel müssen wir hier keine Änderungen vornehmen. Beim Update von Angular werden die TypeScript-Konfigurationen automatisch migriert.[4]

🗀`src`: der Ordner für den Quellcode

Die Angular CLI hat uns unser Projekt bereits mit einer ersten Komponente angelegt. Das Verzeichnis 🗀`src` hat in unserem frischen Projekt die folgende Struktur:

[3] *https://ng-buch.de/d/ng-versions* – Angular Docs: Version compatibility

[4] Angular bietet ein geführtes Update für neue Versionen mit dem Befehl `ng update` an. In Kapitel 38 ab Seite 529 findest du dazu weitere Details.



Bevor wir loslegen, wollen wir noch einige der angelegten Dateien im Verzeichnis src etwas genauer betrachten.

`index.html`: die Einstiegsseite

Die Datei `index.html` ist die Einstiegsseite beim Aufruf der Anwendung im Browser. Das ist also der Teil unserer Anwendung, der vom Browser direkt angefordert wird. Wir sehen hier ein einfaches HTML-Grundgerüst ohne viele Inhalte. Im `<body>` der Seite finden wir das Element `<app-root>`: Es handelt sich um das sogenannte Host-Element der Komponente `App`. Darin erzeugt Angular die BookManager-Applikation, die wir in den folgenden Kapiteln entwickeln werden.

Listing 4.3 *Die Datei* index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>BookManager</title>
  <base href="/">
  <meta name="viewport" content="width=device-width,
    ↪ initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Dieses Element gehört nicht zum HTML-Standard, sondern ist ein von Angular definiertes Element für eine Komponente. Wir werden uns im nächsten Kapitel ab Seite 61 ausführlich mit Komponenten beschäftigen.

Bootstrapping der Anwendung

In der Datei `main.ts` finden wir den Einstiegspunkt für Angular, hier wird der sog. *Bootstrapping*-Prozess von Angular angestoßen. Wir sehen den Aufruf der Funktion `bootstrapApplication()` mit dem Argument `App`: Das ist die Komponente, mit der die Anwendung gestartet wird. Im zweiten Argument wird ein Konfigurationsobjekt übergeben, das aus der Datei `app.config.ts` geladen wird. Wir werden später auf diese `ApplicationConfig` zurückkommen, wenn wir uns mit Providers und Dependency Injection befassen. Zunächst reicht es uns zu wissen, dass mit der Funktion `bootstrapApplication` die Angular-Anwendung mit einer Komponente gestartet wird. Im nächsten Kapitel erfahren wir, was sich hinter Komponenten verbirgt und welche besondere Rolle die Komponente `App` dabei spielt.

Listing 4.4
Das Bootstrapping unserer Anwendung (main.ts)

```
import { bootstrapApplication }
        ↪ from '@angular/platform-browser';
import { appConfig } from './app/app.config';
import { App } from './app/app';

bootstrapApplication(App, appConfig)
  .catch((err) => console.error(err));
```

Statische Assets einbinden

Im Ordner 🗀`public` können wir Dateien ablegen, die beim Build statisch ausgeliefert werden: Bilder, Icons und mehr. Dateien aus diesem Ordner können wir in der Anwendung direkt über den Pfad ab dem Wurzelverzeichnis referenzieren. Legen wir also beispielsweise eine Bilddatei `icon.png` unter 🗀`public` ab, so können wir die Datei später wie folgt im Template einbinden:

```
<img src="icon.png" alt="Icon" />
```

Damit der Ordner berücksichtigt wird, ist er in der Datei `angular.json` unter `assets` aufgeführt. Nur die dort konfigurierten Pfade werden beim Build als statische Assets übernommen und können im Template genutzt werden. Wenn wir Assets an anderen Stellen im Projekt ablegen möchten, müssen wir diese Pfade ebenfalls in der `angular.json` eintragen.

4.5 Das Projekt starten

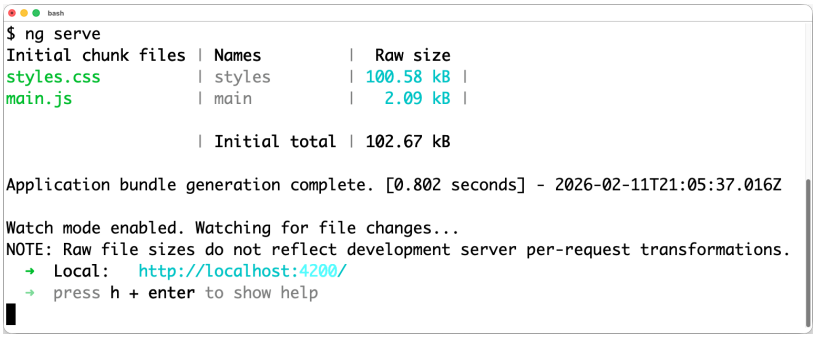
Im Projektordner führen wir nun den Befehl `ng serve` aus, um das Projekt zu starten.

Listing 4.5
Webserver starten mit der Angular CLI

```
$ ng serve
```

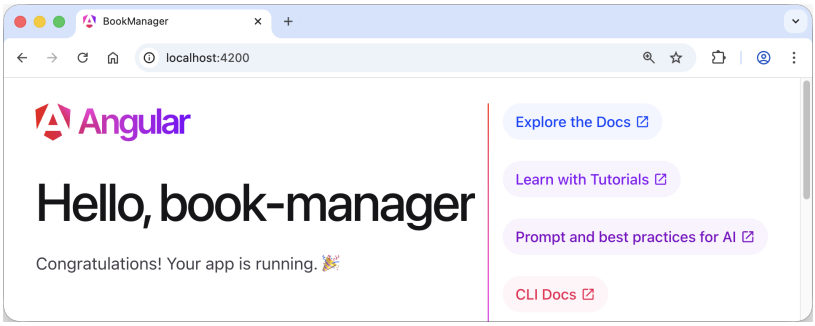
Das Kommando sorgt dafür, dass das gesamte Projekt kompiliert, gebündelt und schließlich von einem Entwicklungswebserver ausgeliefert wird. Sobald wir die Ausgabe der verschiedenen Bundles (`main.js` und `styles.css`) sehen, ist der Build abgeschlossen. Der Webserver läuft so lange, bis wir ihn mit der Tastenkombination Strg + C bzw. ctrl + C wieder beenden.

Abb. 4.5
Aufruf von ng serve *auf der Kommandozeile*



Die fertige Anwendung ist nun über die URL *http://localhost:4200* im Browser erreichbar. Auf der Seite erscheint die Ausgabe *»Hello, book-manager«*. Unsere Anwendung funktioniert!

Abb. 4.6
Die Anwendung im Browser aufrufen



Solange der Server gestartet ist und wir Änderungen an Dateien des Projekts vornehmen und speichern, aktualisiert sich die geöffnete Website stets automatisch im Browserfenster, sodass wir immer eine aktuelle Ansicht vom gerade erstellten Projektstand haben.