ANGULAR CLI

Schnell zur eigenen Web-App

Angular CLI nimmt dem Entwickler viel Arbeit beim Aufsetzen eines neuen Projekts ab.

Wer eine neue Angular-2-Applikation aufsetzt, muss einen hohen initialen Aufwand treiben. Eine Projektstruktur muss definiert und Pakete müssen geladen werden, und welcher Task Runner soll überhaupt zum Einsatz kommen?

Bei all dem unterstützt Angular CLI. Das Command Line Interface erstellt auf Knopfdruck ein komplettes Projekt inklusive lokaler Buildstruktur und Dev Server. Außerdem bietet es eine ausgereifte Scaffolding-Funktion, mit der vordefinierte Strukturen wie Components, Services und Directives strukturiert angelegt werden. Dies geschieht alles ohne großen Aufwand und unter Berücksichtigung des offiziellen Angular Style Guides. Auch das Konfigurieren von Testing und Linting erledigt das Programm. Im Vergleich zum manuellen Aufsetzen eines Angular-Projekts spart der Entwickler hierdurch viel Zeit und kann nach dem Anlegen direkt mit der Entwicklung beginnen.

Die Unterschiede zu einem Angular-Starter-Projekt liegen in der Konfigurierbarkeit von verschiedenen Varianten. So kann beim Anlegen über Flags gesteuert werden, wie das Projekt und der Buildprozess aufgebaut werden.

Da Angular CLI in JavaScript für Node.js geschrieben wurde und auf der Kommandozeile läuft, ist es zum einen Cross-Platform-fähig und auch Editor-unabhängig. Voraussetzung ist dabei nur Node.js ab Version 6.9.0. Wer Node.js mit Visual Studio erhalten hat, muss unter Umständen die neueste Version manuell nachinstallieren.

Angular CLI wird über NPM als globales Paket installiert (siehe Listing 1). Nach der Installation können die CLI Commands von überall ausgeführt werden. Mit dem Befehl *ng help* kann man sich eine Liste der verfügbaren Befehle und Parameter anzeigen lassen.

Listing 1: Install, Help, New, Generate, Serve

```
1: npm install -g @angular/cli
2:
3: ng help
4: ng new <ProjektName> --skip-git --routing
   --skip-install
5:
6: ng generate component <ComponentName>
7:
8: ng serve --port 4201
```

Foto: Shutterstock / Robuart

Neues Projekt anlegen

Der erste Schritt zur eigenen Web-App besteht im Anlegen eines Angular-Projekts. Wer dies bereits von Hand machen musste, weiß, dass es sehr aufwendig sein kann. Die Angular-Pakete müssen geladen und Bootstrap-Code muss angelegt werden, ein sauberer Buildprozess darf ebenfalls nicht fehlen. Zusätzlich muss man sich mit Testing, Linting und Auslieferung beschäftigen. All das nimmt einem Angular CLI bereits beim Anlegen des Projekts ab.

Durch den Befehl *ng new* (siehe Listing 1) wird ein neues Projekt angelegt. Standardmäßig wird dabei auch ein neues Git-Repository erstellt, und alle NPM-Pakete werden installiert. Um diese Schritte zu überspringen, können Sie die Flags *--skip-git* und *--skip-install* verwenden. Über das Flag *--routing* lässt sich zusätzlich das Routing aktivieren.

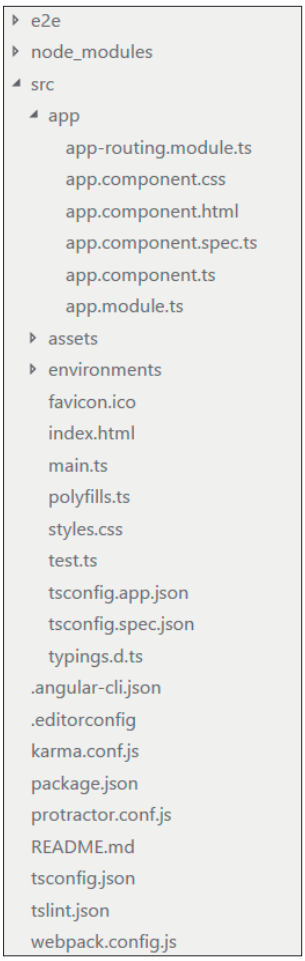
Das Überspringen der Paketinstallation bringt den Vorteil, dass Sie im Nachhinein noch den Package Manager wechseln können. Wer hier eine Vorliebe für yarn [1] als Package-Manager-Alternative zu NPM hat, kann dies bequem konfigurieren.

Nach dem Erstellen erhalten Sie eine Projektstruktur, wie sie in Bild 1 dargestellt ist. Im Hauptordner befinden sich die Konfigurationsdateien für unterschiedliche Frameworks wie

NPM, Karma (JS-Unit-Test-Runner), Protractor (E2E-Test-Runner), TypeScript und TS Lint. Das *src*-Verzeichnis enthält alle Dateien, die für die Entwicklung relevant sind. Wer die Konfiguration nicht anpassen möchte, kann sich direkt dem *app*-Ordner widmen. Dieser beinhaltet, wie man es von Angular gewohnt ist, das App Module.

In der *index.html* befindet sich das Layout der Seite. Alle Skripts und Stylesheets werden automatisch beim Build mit eingebunden und müssen hier nicht mehr angegeben werden.

Die *main.ts*-Datei enthält das Bootstrapping der Angular-App, lädt automatisch das App Module und aktiviert den Production Mode, sofern dieser in den Environments aktiviert wurde. Wer alte Browser unterstützen muss, sollte einen Blick in die Datei *polyfills.ts* werfen. Hier werden alle Browser-Polyfills eingebunden, die später in einem zentralen Bundle liegen.



Die Projektstruktur einer neuen Web-App (Bild 1)

Um Third-Party Libraries oder Stylesheets hinzuzufügen, gibt es in der *.angular-config.json* unter dem Key *apps* die Keys *scripts* und *styles*, die eine Liste von Pfaden zu den einzelnen Bibliotheken enthalten. Alles, was im *assets*-Ordner landet, wird automatisch mit verteilt und beinhaltet Bilder, Fonts oder andere statische Dateien.

Scaffolding

Nachdem die grundlegenden Projektstrukturen angelegt wurden und bereits ein rudimentäres, lauffähiges Gerüst darstellen, sollen einige Komponenten das doch recht leere Gerüst füllen. An dieser Stelle lohnt sich ein Blick sowohl in die Angular- als auch die Angular-CLI-Dokumentation [2][3]. Dort sind einerseits die unterschiedlichen Komponenten mit einfachen bis komplexen Beispielen übersichtlich zusammengefasst. Andererseits werden die Konsolenbefehle sowie deren Argumente gelistet, was beim Einstieg ungemein hilfreich ist.

Das Scaffolding bietet die Möglichkeit, je nach Bedarf und Anforderung die passende Komponente in das Gerüst zu integrieren. Beispielhaft erstellen Sie nun Ihre erste Angular Component und damit verbunden Ihre erste View. Navigieren Sie mithilfe der Konsole in das Root-Verzeichnis des Projekts und erstellen Sie die Component mittels des Befehls in Zeile 6 aus Listing 1. Ersetzen Sie dabei den Parameter <*ComponentName*> durch den gewünschten Namen der Komponente. Zunächst wird die Konsole anzeigen, dass vier Dateien innerhalb das App-Verzeichnisses in einem namensgleichen Unterverzeichnis angelegt wurden. Angular CLI übernimmt aber zusätzliche Arbeit für Sie, da die *app.module.ts*-Datei um die fehlenden Referenzen erweitert wurde.

Es existiert eine Datei mit der Endung **.component.ts*, die für die Interaktion mit der View verantwortlich ist. Das Template selbst wurde als **.component.html* zur Modellierung der DOM-Elemente hinzugefügt, und ebenso deren Styling in der Datei **.component.css*. Haben Sie keine weiteren Argumente beim Aufruf angefügt, so wird zusätzlich die Datei **.spec.ts* erzeugt. Sie dient der Definition der Tests für die Component.

Das Beispiel gilt nicht nur für die Component. Die Erstellung anderer Komponenten läuft semantisch ähnlich ab, da in Abhängigkeit von der benötigten Komponente ein *ng generate* aufgerufen wird und die Komponente mit ihrer Bezeichnung und ihrem individuellen Namen erzeugt wird.

Interessant ist auch die Parametrisierung der Erstellung, da so den eigenen Bedürfnissen nachgekommen werden kann. Ist die Erzeugung der **.spec.ts*-Datei innerhalb des App-Ordners nicht erwünscht, da beispielsweise ein eigenes Testprojekt an einer anderen Stelle existiert, kann dies mit dem Zusatz *--spec false* verhindert werden. Gleiches gilt für das Template (*--inline-template true*) oder auch für den korrespondierenden Style (*--inline-style true*).

Hinzu kommt, dass Komponenten in Abhängigkeit vom aktuellen Pfad relativ zum App-Verzeichnis erstellt werden. So können Subkomponenten einfach angelegt werden, indem man in den Komponentenpfad navigiert. Im Gegensatz dazu kann durch Setzen des Parameters *--flat true* und der Pfadposition im Root-Verzeichnis die Erzeugung eines extra Ordners für die Komponente unterbunden werden.

Projekt starten

Das Projekt ist erstellt, die notwendigen Konfigurationsdateien wurden von der Angular CLI angelegt, und zusätzlich haben Sie bereits eine Component eingefügt. Alles in allem existiert – ob selbst oder mithilfe von *ng generate* angelegt – ausreichend lauffähiger Code, um die Anwendung einmal zu starten und zu debuggen. Das Stichwort, mit dem Sie die Kompilierung und Auslieferung im lokalen Webserver starten, lautet hier *ng serve*. Beim verwendeten Webserver handelt es sich im Übrigen um den *Webpack Dev Server* [4]. Der Befehl *ng serve* (siehe Zeile 8 aus Listing 1) startet den Webserver und lädt die Ressourcen.

Die Konsole müsste nach der Eingabe nun darüber informieren, dass bei den Default-Einstellungen der Server unter *http://localhost:4201* erreichbar ist. Andererseits sollte sie auch anzeigen, dass die Komponenten zusammengeführt wurden. Durch das Anhängen von *--open* wird zusätzlich der Browser geöffnet.

Mit dem Befehl *ng help serve* können Argumente für das Kommando *serve* im Überblick eingesehen werden. Das Flag *--target* spezifiziert die adressierte Zielumgebung mit dem Standardwert *Development* und der Option *Production*. ▶

Der Parameter beeinflusst die Form der nach der Kompilierung gelieferten Ressourcen sowie die Ausführlichkeit (Verbosity) der innerhalb Angulars aufgezeichneten Fehler.

Unter Angabe von *--watch true* werden der Build und die damit verbundene Kompilierung der Ressourcen neu getriggert und auf der Konsole ausgegeben, sobald gespeicherte Änderungen erkannt werden. Sollte es geschehen, dass der gespeicherte Code Fehler aufweist, so sind diese einsehbar und kein neuer Stand wird ausgeliefert. Das bedeutet, dass über den Browser immer die letzte lauffähige Version abrufbar ist. In Verbindung damit kann *--poll* unter Angabe eines Intervalls in Millisekunden verwendet werden; dies spezifiziert, in welchem zeitlichen Abstand die Dateien auf Änderungen geprüft werden sollen. Das kann unter Umständen hilfreich sein, sobald die Kompilierung einen längeren Zeitraum in Anspruch nimmt, damit sich die ausgelösten Prozesse nicht gegenseitig beeinflussen.

Sollte die Anwendung später in einer minifizierten und gebündelten Form ausgeliefert werden, so ist es möglich, die Generierung der Sourcemaps für das Mapping vom eigentlichem TypeScript-File hin zum gebündelten File zu unterdrücken. Dies geschieht mit dem Argument *--sourcemap false*, was sich direkt im Debugger des Browsers bemerkbar macht, da das Verzeichnis *webpack://* nicht mehr angezeigt wird.

Es fällt auf, dass bei Änderungen, die innerhalb des Editors an den Projektdateien gemacht werden, zum einen der durch den *--watch true* gestarteten Recompile-Vorgang angestoßen wird, und dass zum anderen das Browserfenster ohne Zutun des Benutzers neu geladen und so mit den aktuellen Ressourcen versorgt wird. Dies geschieht durch das Argument *--live-reload true*, das standardmäßig bereits aktiviert ist. Dadurch wird festgelegt, ob der Refresh der Seite manuell durchgeführt werden muss oder ob dies in die Verantwortlichkeit des Angular CLI übergeht.

Dies sind nur einige interessante Parameter für die Anpassung des Angular CLI an die persönlichen Bedürfnisse und den Projektkontext. Ein schneller Blick genügt aber, um sich einen Eindruck zu verschaffen, was ergänzend zur Konfiguration bereitsteht. Hot Module Replacement oder Ahead of Time Compilation sind nur zwei weitere Stichwörter, die mit dem Tool adressierbar sind.

Lint – Test – Deploy

Neben der Unterstützung für das Erstellen, Bauen und Debuggen von Angular-Applikationen bringt Angular CLI noch weitere praktische Features für die Durchsetzung von einheitlichem Code-Style oder für das Konfigurieren und Ausführen von Tests mit. So besteht die Option, den TypeScript Linter für das Aufzeigen von Style- und Code-Problemen sowie für das Ausführen der angelegten Unit- und End-to-End-Tests zu nutzen.

Unter Eingabe von *ng help lint* gewinnt man wie auch bei den vorherigen Eingaben einen Überblick über die zur Verfügung stehenden Argumente. Bei der Verwendung des Linters werden die im Root-Pfad des erstellten Projekts angelegten Regeln innerhalb der *tslint.json*-Datei herangezogen. Es können weitere Regelsätze neben den Standardregeln, die durch das Angular CLI angelegt wurden, nach Belieben hinzugefügt werden. Je nach Projektsituation und Anforderungen bezüglich Clean Code lohnt sich ein Blick in die verfügbaren Regeln auf GitHub [5], um das eigene Projekt anzupassen. Sobald alle Regeln nach Wunsch konfiguriert sind, kann man sich durch die Eingabe von *ng lint* einen Überblick über Verstöße verschaffen. Anschließend werden die aufgedeckten Probleme dargestellt, oder aber der Linter läuft ohne Beanstandung und schließt mit einem grünen Output.

Listing 2: Lint, Test, Deploy

```
1: ng lint --type-check
2:
3: ng test --browsers [Browsers]
4: ng e2e

5: ng build --target=production
```

Zudem können einige Verstöße – auf der TS-Lint Seite [5] mit *Has Fixer* gekennzeichnete Regeln – durch Angabe von *ng lint -fix* direkt korrigiert werden. Dabei werden die angepassten Dateien und die enthaltene Anzahl der Änderungen ausgegeben. Bei der Überprüfung der Fehler ist es oft hilfreich, wenn sowohl die Ausführlichkeit der Fehlerbeschreibung als auch das Format angepasst werden, da so schneller die Ursache des Problems erkannt werden kann.

Dies geschieht mit der Angabe des Arguments *ng lint --format <Wert>* und dem entsprechenden Formatierungswert. Voreingestellt ist in Angular CLI die Ausgabe *prose* für die Ausgabe in Prosa-Text. Für Nutzer, die den Output weiterverarbeiten oder den Check als Teil eines automatisierten Prozesses verwenden, kann die Formatierung als *json* oder *checkstyle* in Frage kommen. Sollte die Ausgabe direkt leserlich gestaltet werden, kann aber auch *msbuild* oder *stylish* verwendet werden.

Im vorherigen Abschnitt zum Scaffolding mit dem Angular CLI wurde kurz die automatisierte Erstellung der Spec-Dateien für spätere Unit-Test-Implementierung angesprochen. Auch hier wird der Entwickler durch Angular CLI bei der Ausführung mithilfe des simplen Aufrufs von *ng test* unterstützt. Dabei werden die vorhandenen Tests innerhalb der *spec.ts*-Dateien unter Verwendung des Testrunners Karma und des Unit-Tests-Frameworks Jasmine ausgeführt. Die Tests laufen nach dem Bauen des Projekts innerhalb des sich öffnenden Browserfensters. Das Resultat ist direkt einsehbar.

Sollte die Ausführung innerhalb verschiedene Browser gewünscht sein, wird mit dem Parameter *ng test --browsers [Browser,Browser]* und den Bezeichnern für die Browserhersteller gearbeitet (siehe Listing 2 in Zeile 3).

Im Vorfeld ist es natürlich nötig, neben dem durch das CLI mitgelieferten Chrome-Browser noch weitere zu installieren. Dies geschieht durch das Hinzufügen in der *package.config* für die Verwaltung der NPM-Abhängigkeiten und der Ausführung von *npm install* in der Konsole auf gleicher Ebene.

Darüber hinaus muss in der Konfigurationsdatei von *karma.conf.js* der hinzugefügte Browser gegenüber dem Testrunner bekannt gemacht werden, was durch das Hinzufügen des Plug-ins geschieht.

Angular CLI unterstützt darüber hinaus die Ausführung des End-to-End-Test-Frameworks Protractor. Dabei stehen bereits bekannte Argumente für die Konfiguration des Webservers zur Verfügung.

Wer seine Applikation deployen möchte, kann das über den Befehl *ng build* vorbereiten. Dadurch werden alle relevanten Dateien in den *dist*-Order kopiert. Von dort kann man sie entweder manuell oder über einen automatisierten Prozess auf die Zielumgebung kopieren.

Konfiguration und Customizing

Das initiale Projekt kommt vorkonfiguriert und beinhaltet alles, was für den Start benötigt wird. Wer die verwendeten Frameworks (zum Beispiel TypeScript oder Karma) konfigurieren möchte, kann direkt die verschiedenen Konfigurationsdateien der Frameworks bearbeiten. Zusätzlich bietet Angular CLI eine eigene Konfigurationsdatei mit dem Namen *.angular-cli.json*, in der verschiedene Optionen zur Verfügung stehen.

Hier können die jeweiligen Pfade zu Assets und Komponenten konfiguriert werden, die später in den einzelnen Bundles zusammengefasst werden. Wer zusätzliche Environments (zum Beispiel für die Qualitätssicherung) hinzufügen möchte, kann dies unter dem Key *Environment* tun. Diese können dann beim Build und Serve mitangegeben werden.

Hinter den Kulissen verwendet Angular CLI Webpack zum Auflösen und Bündeln von Abhängigkeiten. Mit dem Befehl *ng eject* kann die Webpack-Konfiguration direkt angepasst werden.

Durch *eject* wird die Webpack-Konfiguration exportiert und alle Angular-CLI-Befehle lassen sich über *npm* ausführen. Hierbei ist jedoch Vorsicht geboten: Einmal exportiert, kann man das Angular CLI nicht mehr verwenden und bekommt nur noch über Umwege den ursprünglichen Stand zurück. Davon abgesehen kann danach jedoch der bestehende Prozess an die persönlichen Bedürfnisse angepasst werden.

Fazit

Die Hürden beim richtigen Einrichten eines Projekts werden von vielen Entwicklern als eine Barriere beim Einstieg in neue Technologien wahrgenommen. Immer mehr Helfer wie Yeoman unterstützen den Entwickler bei der Überwindung solcher Hürden und versuchen, den initialen Overhead so gering wie möglich zu gestalten.

Das Team um Angular hat diese Herausforderung erkannt und bietet neben dem Framework zur Realisierung komplexer Webanwendungen nun auch noch den passenden Generator in Form von Angular CLI an.

Projekte mit grundsätzlichen Komponenten, Referenzen und Modulen lassen sich so schnell und einfach von jedem durchführen. Probleme in Bezug auf fehlende, falsche oder überflüssige Elemente werden beim Scaffolding so gut wie ausgeschlossen. Gleichermaßen gestattet die mögliche umfangreiche Parametrisierung auch, individuelle Konfigurationen zu schaffen.

Neben dem bloßen Scaffolding unter Berücksichtigung des Angular Style Guides bietet Angular CLI weitere Features wie Debugging, Linting oder Testing. Entwicklern wird auf diese Weise ein Tool an die Hand gegeben, das die Out-of-the-Box-Ausführung im lokalen Webserver und damit das Debugging unterstützt.

Zusätzlich können Unit-Tests und End-to-End-Tests ohne weitere Konfigurationshürden ausgeführt werden – stets mit ausführlichem und hilfreichem Feedback des CLI.

Darüber hinaus kann Angular CLI auch in neue und bestehende Projekte eingebunden werden. So lassen sich die Aufrufe via Gulp-Tasks [6] beispielsweise auf dem CLI aufrufen, was zur Erzeugung der notwendigen Artefakte führt. Weiter können diese gleichermaßen im Build-Prozess mit entsprechendem *Production*-Attribut parametrisiert werden. Ganz von allein kommt man so mit Angular CLI zwar zum Release-fertigen Artefakt, dennoch muss man sich bei der Automatisierung mit weiterem Tooling im CI-Build beschäftigen. Wer Herr über seinen eigenen Buildprozess sein möchte, kann selbst das über das CLI erreichen.

Durch die Verfügbarkeit der Sourcen auf GitHub [3] und die rege Verwendung des CLI besteht auch hier eine stetige Weiterentwicklung. Dies führt dazu, dass auch in Zukunft weitere Features hinzukommen werden, was die Verwendung voraussichtlich komfortabler und umfangreicher gestaltet. ■

[1] Yarn, https://github.com/yarnpkg
[2] Architecture Overview, www.dotnetpro.de/SL1710AngularCLI1
[3] angular/angular-cli, www.dotnetpro.de/SL1710AngularCLI2
[4] webpack-dev-server, www.dotnetpro.de/SL1710AngularCLI3
[5] TSLint, www.dotnetpro.de/SL1710AngularCLI4
[6] GulpJs, http://gulpjs.com

Florian Bader
ist Consultant im Bereich Microsoft .NET bei der AIT GmbH & Co. KG. Er berät Kunden im Bereich Application Lifecycle Management mit Schwerpunkt in Softwareentwicklung und –architektur sowie Buildautomatisierung.
florian.bader@aitgmbh.de

Lukas Ochsenreiter
ist Consultant bei AIT GmbH & Co. KG. Er ist als Softwareentwickler mit Schwerpunkt Webentwicklung im industriellen Umfeld tätig.
lukas.ochsenreiter@aitgmbh.de

dnpCode A1710AngularCLI