

Angular - Hello World

1 EINFÜHRUNG	2
1.1 Angular Version	2
1.2 Node	2
1.2.1 Releases	3
1.3 Yarn	4
1.3.1 Installation	4
1.3.2 Npm → Yarn	4
1.3.3 Vergleich	5
2 FRONTEND – ANGULAR CLI	6
2.1 Angular CLI	6
2.1.1 Installation	6
2.1.2 Prüfen	6
2.1.3 Path	6
2.1.4 ng	7
2.2 Scaffold app	8
2.2.1 Optionen	8
2.2.2 Restore packages	8
2.2.3 Best Practice	9
2.3 Erster Start	9
2.3.1 ng serve	9
2.3.2 Browser	10
3 ENTWICKLUNGSUMGEBUNG	11
3.1 Plugins	11
3.1.1 Angular Language Service	11
3.1.2 Angular Snippets	11
3.2 Terminal	12
3.3 Scripts	12
3.4 Debugging	12
3.4.1 Firefox	13
3.5 Production build	14
4 DATEIÜBERBLICK	15
4.1 index.html	15
4.2 main.ts – “Hauptprogramm”	16
4.3 app-Folder	16
4.3.1 app.module.ts – Start-Modul	16
4.3.2 app.component.ts – Start-Komponente	17
4.3.2.1 app.component.html	18
4.3.2.2 app.component.scss	18
4.3.2.3 app.component.spec.ts	18
4.4 Infrastruktur	18
4.4.1 package.json	18
4.4.2 styles.scss	19
4.4.3 tsconfig.app.json / tsconfig.json	20

1 Einführung

Angular ist eine Javascript-Bibliothek zum Entwickeln von Web-Apps, die das **MVC-Pattern** implementiert. In Zusammenhang mit Angular wird dabei auch oft von **MVW** gesprochen (**M**odel-**V**iew-**W**hatever). Sie besteht aus mehreren Javascript-Dateien, die man am besten mit Node installiert. Außerdem werden einige zusätzliche Javascript-Pakete benötigt. Die Homepage ist unter <https://angular.io/> zu finden.

Die hier besprochene Version ist **Angular >=13**.

Es wird empfohlen, Angular mit **TypeScript** zu benutzen. Das ist zwar nicht unbedingt notwendig, erleichtert die Codierung meiner Meinung aber doch (auch wenn es zu Beginn vielleicht etwas ungewohnt aussieht). Daher wird in diesem und allen folgenden Tutorials ausschließlich Typescript verwendet.

Die Funktionsweise ist ähnlich wie bei **WPF/MVVM**: Es wird Markup in das HTML-Template eingebaut, das sich auf Properties und Funktionen des dahinterliegenden Modells bezieht. Es wird also ein clientseitiges Model-View-Controller-Pattern verwendet.

Angular ist modular aufgebaut. In jeder Komponente bzw. Modul muss zu Beginn angegeben werden, welche der vielen Module benutzt wird. Dabei können bereits existierende Module angegeben werden, jede App besteht aber eben auch aus selbst programmierten Komponenten, Direktiven und Modulen.

1.1 Angular Version

Angular Versionen sind wie gewohnt aufgebaut in major.minor.patch (Siehe: <https://angular.io/guide/releases>, Stand 2022-06-02).

All major releases are typically supported for 18 months.

SUPPORT STAGE	SUPPORT TIMING	DETAILS
Active	6 months	Regularly-scheduled updates and patches are released
Long-term (LTS)	12 months	Only critical fixes and security patches are released

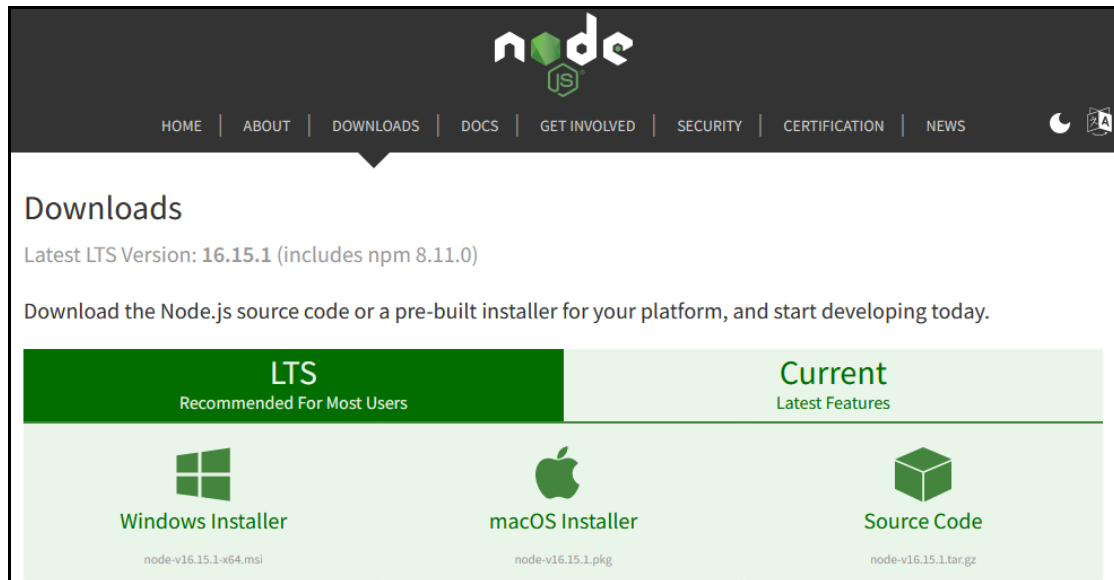
The following table provides the status for Angular versions under support.

VERSION	STATUS	RELEASED	ACTIVE ENDS	LTS ENDS
^13.0.0	Active	2021-11-04	2022-05-04	2023-05-04
^12.0.0	LTS	2021-05-12	2021-11-12	2022-11-12

Angular versions v2 to v11 are no longer under support.

1.2 Node

Voraussetzung für die in diesem Tutorial besprochene Verwendung von Angular ist Node. die aktuelle Version kann von <https://nodejs.org/en/> bzw. <https://nodejs.org/en/download/> installiert werden.



„LTS release status is \"long-term support\", which typically guarantees that critical bugs will be fixed for a total of 30 months.“

Es sollte immer die aktuelle LTS-Version (Long Term Support) installiert werden. Die aktuelle Version ist 16.15.1 (Stand 2022-06-02), das kann mit **node -v** überprüft werden. Es reicht aber auch eine ältere Version:

```
C:\>node -v
v16.15.0

C:\>npm -v
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.
8.11.0
```

Hinweis: Die Warnung besagt, dass globale Module jetzt folgendermaßen installiert werden müssen:

npm install myPackage --location=global

Vorher: **npm install myPackage -g**

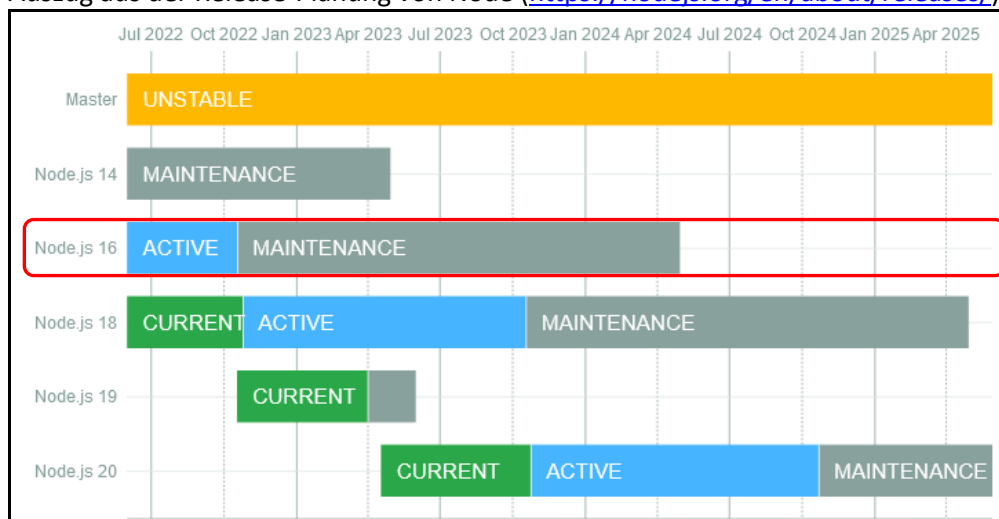
Siehe <https://docs.npmjs.com/cli/v8/commands/npm-install>:

global

- Default: false
- Type: Boolean
- DEPRECATED: `--global`, `--local` are deprecated. Use `--location=global` instead.

1.2.1 Releases

Auszug aus der Release-Planung von Node (<https://nodejs.org/en/about/releases/>):



RELEASE	STATUS	CODENAME	INITIAL RELEASE	ACTIVE LTS START	MAINTENANCE LTS START	END-OF-LIFE
v14	Maintenance LTS	Fermium	2020-04-21	2020-10-27	2021-10-19	2023-04-30
v16	Active LTS	Gallium	2021-04-20	2021-10-26	2022-10-18	2024-04-30
v18	Current		2022-04-19	2022-10-25	2023-10-18	2025-04-30
v19	Pending		2022-10-18		2023-04-01	2023-06-01
v20	Pending		2023-04-18	2023-10-24	2024-10-22	2026-04-30

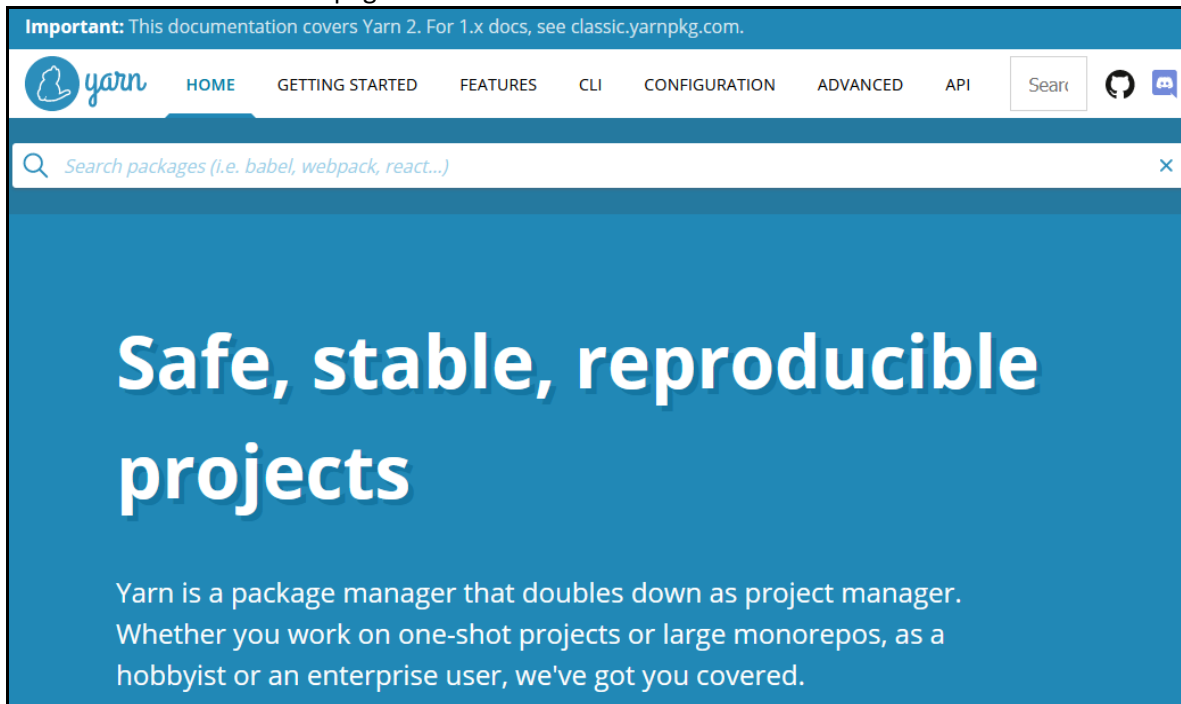
1.3 Yarn

Auf manchen PCs (meinem Laptop, aber da bin ich nicht alleine) dauert die Installation eines neuen Angular-Projekts ca. 1 Stunde!

Als Alternative bietet sich dann Yarn an: <https://yarnpkg.com/lang/en/>

1.3.1 Installation

Den Installer von der Homepage laden und damit Yarn installieren.



1.3.2 Npm → Yarn

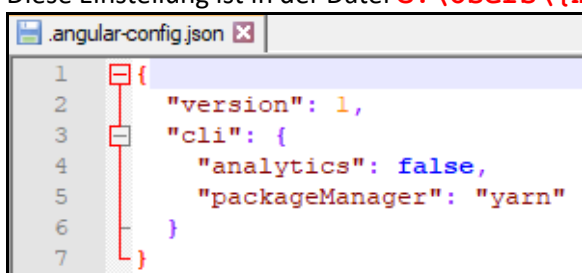
Um jetzt anstelle von Node Yarn für Angular-Projekte (angular-cli muss dazu installiert sein – siehe unten) zu verwenden muss folgende Zeile eingegeben werden:

ng config -g cli.packageManager yarn

Um wieder auf Node zurückzuschalten gibt man sinnigerweise folgenden Befehl ein:

ng config -g cli.packageManager npm

Diese Einstellung ist in der Datei **C:\Users\{myUserName}\.angular-config.json** gespeichert



Damit dauert die Erstellung eines neuen Angular-Projekts wieder nur mehr 1-2 Minuten.

Im generierten Projekt ändert sich dabei aber nichts, insbesondere sieht **package.json** so aus wie bei npm.

1.3.3 Vergleich

Auf <https://yarnpkg.com/en/docs/migrating-from-npm> ist folgender Vergleich der Befehle von npm und yarn aufgelistet:

npm (v5)	Yarn
npm install	yarn install
-	yarn install --flat
-	yarn install --har
npm install --no-package-lock	yarn install --no-lockfile
-	yarn install --pure-lockfile
npm install [package] --save	yarn add [package]
npm install [package] --save-dev	yarn add [package] --dev
-	yarn add [package] --peer
npm install [package] --save-optional	yarn add [package] --optional
npm install [package] --save-exact	yarn add [package] --exact
-	yarn add [package] --tilde
npm install [package] --global	yarn global add [package]
npm update --global	yarn global upgrade
npm rebuild	yarn add --force
npm uninstall [package]	yarn remove [package]
npm cache clean	yarn cache clean [package]
rm -rf node_modules && npm install	yarn upgrade
npm version major	yarn version --major
npm version minor	yarn version --minor
npm version patch	yarn version --patch

2 Frontend – Angular CLI

2.1 Angular CLI

Zum Erzeugen der Angular-Struktur greift man am besten auf das Tool **@angular/cli** zurück, das man mit **npm** installiert – üblicherweise global.

2.1.1 Installation

Befehl: **npm install @angular/cli --location=global**

Danach hat man über **ng** einige neue Befehle zur Verfügung.

```
C:\>npm install @angular/cli --location=global
npm WARN config global '--global', '--local' are deprecated. Use '--location=global' instead.
added 13 packages, changed 183 packages, and audited 197 packages in 17m
24 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
```

Es wird die aktuelle Version des CLI installiert (wenn man nicht explizit eine gewünschte Version angibt).

2.1.2 Prüfen

Normalerweise sollte jetzt der Befehl **ng** auf der Commandline verfügbar sein. Falls nicht, liegt das (vor allem auf den Schul-PCs) daran, dass der die Variable **PATH** nicht richtig erweitert wurde (siehe unten).

```
C:\>ng version

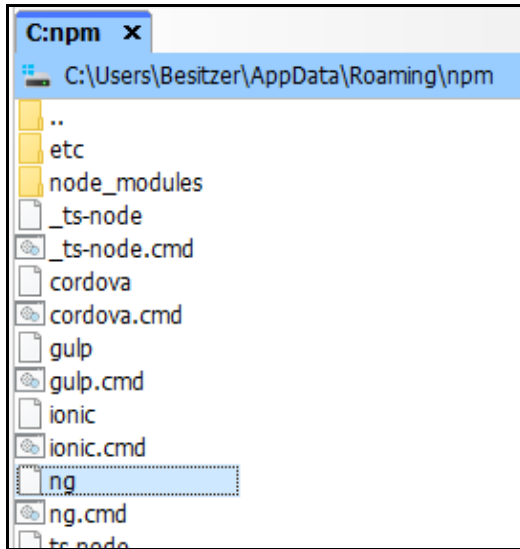
Angular CLI
Angular CLI: 13.3.7
Node: 16.15.1
Package Manager: yarn 1.22.18
OS: win32 x64

Angular:
...

Package                                  Version
-----
@angular-devkit/architect                0.1303.7 (cli-only)
@angular-devkit/core                     13.3.7 (cli-only)
@angular-devkit/schematics               13.3.7 (cli-only)
@schematics/angular                      13.3.7 (cli-only)
```

2.1.3 Path

Die Start-Scripts aller global installierten Module, also z.B. **ng.cmd**, liegt nämlich im User-Verzeichnis:

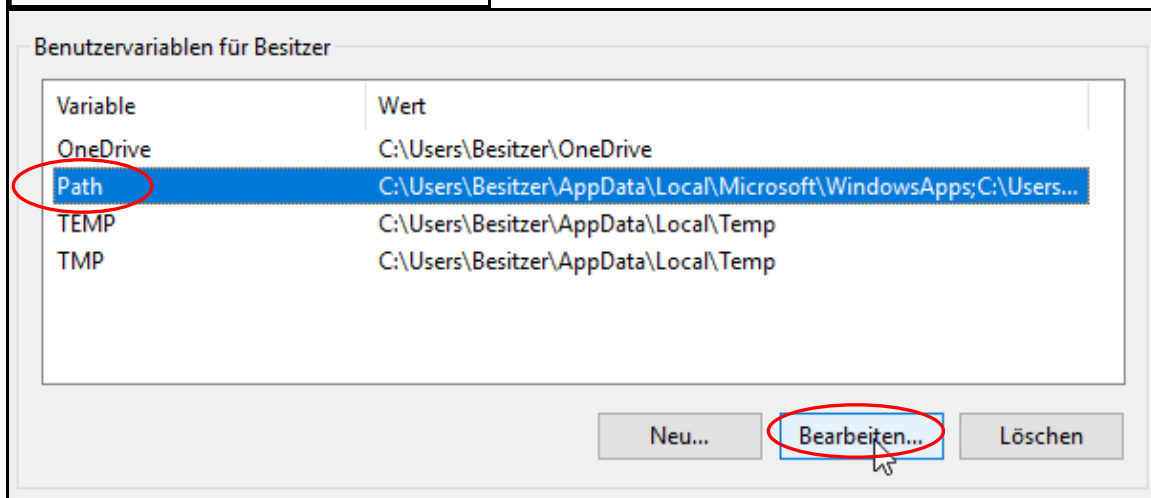
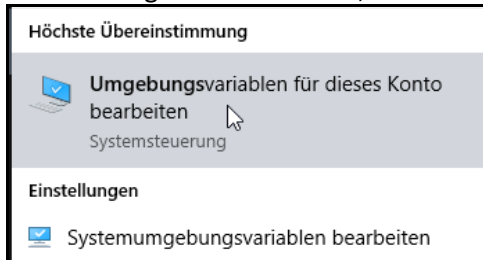


```
C:\Users\Besitzer>set path
Path=C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\In
2\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Program File
el\Intel(R) Management Engine Components\DAL;C:\Program Files\Inte
\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files\
2;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsP
nn\;C:\Program Files\dotnet\;C:\Program Files\Git\cmd;C:\Program F
Users\Besitzer\AppData\Local\Android\Sdk\platform-tools;C:\Program F
cal\Microsoft\WindowsApps;C:\Users\Besitzer\AppData\Roaming\npm;
```

Sollte das nicht der Fall sein, muss der Pfad erweitert werden (gilt dann aber nur für das aktuelle Konsolenfenster!):

```
C:\>set path=%path%;%appdata%\npm
```

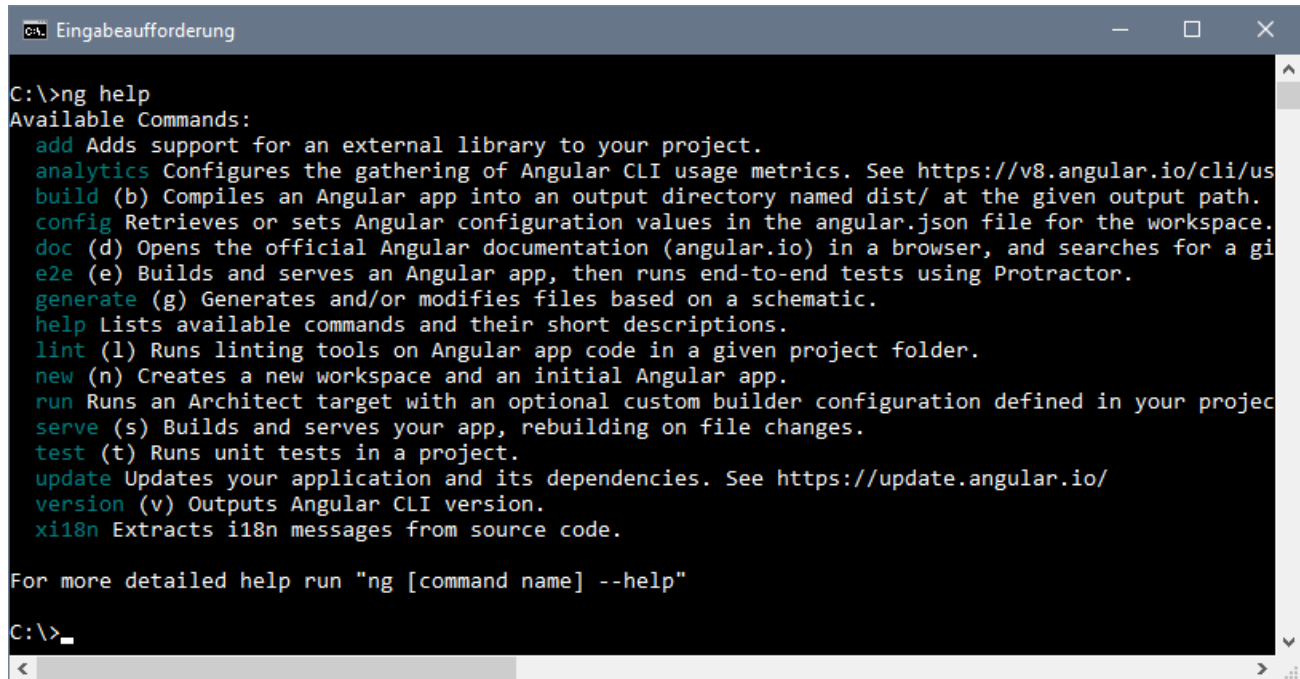
Dasselbe Ergebnis erhält man, wenn man die Umgebungsvariablen ändert:



Diese Einstellungen gelten dann für alle (künftigen) Konsolenfenster.

2.1.4 ng

Mit dem (mit obigen Einstellungen sicher global verfügbaren) Befehl ng kann man dann von der Konsole aus Projekte erstellen und verändern.



```

C:\>ng help
Available Commands:
  add Adds support for an external library to your project.
  analytics Configures the gathering of Angular CLI usage metrics. See https://v8.angular.io/cli/us
  build (b) Compiles an Angular app into an output directory named dist/ at the given output path.
  config Retrieves or sets Angular configuration values in the angular.json file for the workspace.
  doc (d) Opens the official Angular documentation (angular.io) in a browser, and searches for a gi
  e2e (e) Builds and serves an Angular app, then runs end-to-end tests using Protractor.
  generate (g) Generates and/or modifies files based on a schematic.
  help Lists available commands and their short descriptions.
  lint (l) Runs linting tools on Angular app code in a given project folder.
  new (n) Creates a new workspace and an initial Angular app.
  run Runs an Architect target with an optional custom builder configuration defined in your projec
  serve (s) Builds and serves your app, rebuilding on file changes.
  test (t) Runs unit tests in a project.
  update Updates your application and its dependencies. See https://update.angular.io/
  version (v) Outputs Angular CLI version.
  xi18n Extracts i18n messages from source code.

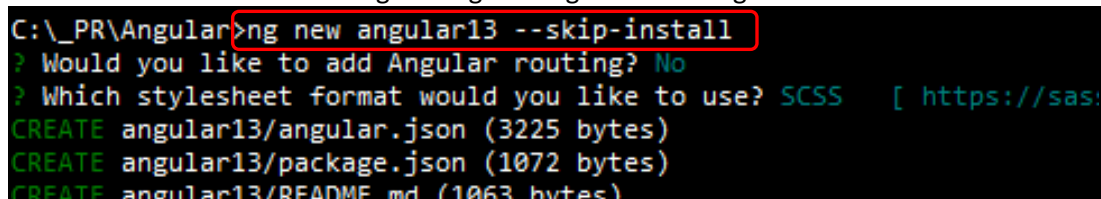
For more detailed help run "ng [command name] --help"
C:\>_

```

2.2 Scaffold app

Im ersten Schritt erzeugt man mit dem oben installierten @angular/cli eine Hello-World-Struktur. Der Befehl lautet **ng new**. Als Parameter gibt man den Projektnamen an, und zwar üblicherweise in Kebab Case, also z.B. **ng new angular13**. Um vorerst die sehr speicherintensiven node_modules nicht zu installieren kann man die Option **--skip-install** angeben.

In beiden Fällen muss man einige wenige Konfigurationen angeben:



```

C:\_PR\Angular>ng new angular13 --skip-install
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? SCSS [ https://sas:
CREATE angular13/angular.json (3225 bytes)
CREATE angular13/package.json (1072 bytes)
CREATE angular13/README.md (1063 bytes)

```

Man wird über zwei Optionen gefragt:

- Routing brauchen wir vorerst nicht
- By Stylesheets sollte man SCSS auswählen

Etwasged Warnings am Ende kann man ignorieren.

2.2.1 Optionen

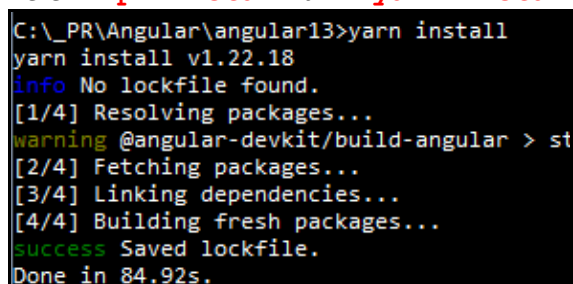
Im AngularCLI-Wiki (<https://angular.io/cli/new>) kann man alle Optionen nachschlagen.

So könnte man mit **--skip-tests** auch das Erzeugen der .spec.ts-Testdateien verhindern.

2.2.2 Restore packages

Falls die Node-Module noch nicht automatisch heruntergeladen wurden (bzw. wegen **-skip-install**), sollte man das jetzt nachholen. Dazu muss man aber **ins Verzeichnis des Projekts wechseln**.

Befehl: **npm install** bzw. **yarn install**

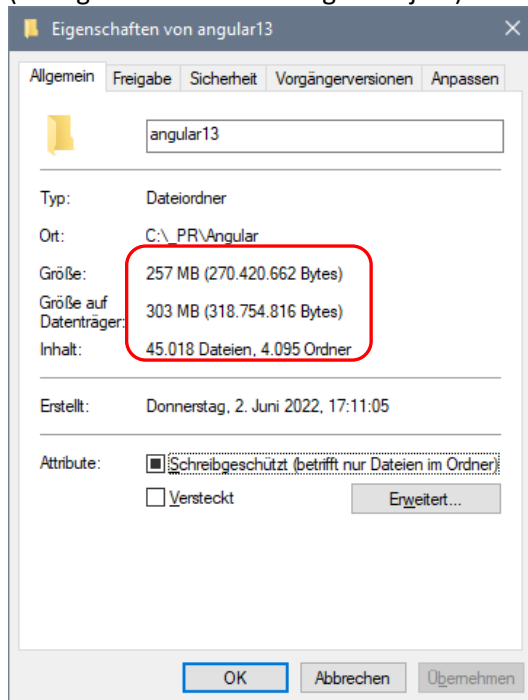


```

C:\_PR\Angular\angular13>yarn install
yarn install v1.22.18
info No lockfile found.
[1/4] Resolving packages...
warning @angular-devkit/build-angular > st
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...
success Saved lockfile.
Done in 84.92s.

```


Es kann doch einige Zeit dauern. Es werden ca. 200MB an ins Verzeichnis `node_modules` heruntergeladen (wohlgemerkt für ein einziges Projekt)!



2.2.3 Best Practice

Für die vielen Übungen ist es aufgrund des Speicherbedarfs und der langen Installationsdauer am besten, einmal ein Projekt anzulegen und dieses immer wieder zu verwenden. Dabei ist dann immer nur der **src**-Ordner zu archivieren bzw. auszutauschen.

2.3 Erster Start

Die so erzeugte Applikation ist ohne Änderung sofort lauffähig.

2.3.1 ng serve

Zum Starten muss man im Projekt-Ordner **ng serve** eingeben:

```
C:\_PR\Angular\angular13>ng serve
✓ Browser application bundle generation complete.

Initial Chunk Files | Names          | Raw Size
vendor.js           | vendor         | 1.70 MB |
polyfills.js        | polyfills      | 294.84 kB |
styles.css, styles.js | styles        | 173.69 kB |
main.js             | main           | 47.99 kB |
runtime.js          | runtime        | 6.52 kB |
                    | Initial Total  | 2.21 MB

Build at: 2022-06-02T15:17:24.989Z - Hash: 51af0a14fef9c99f - Time: 8318ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

✓ Compiled successfully.
```

Dieser Befehl fungiert automatisch auch als Watcher, d.h. wenn sich Dateien ändern (Html, Typescript, CSS, ...) wird der Browser aktualisiert (ohne <F5> drücken zu müssen).

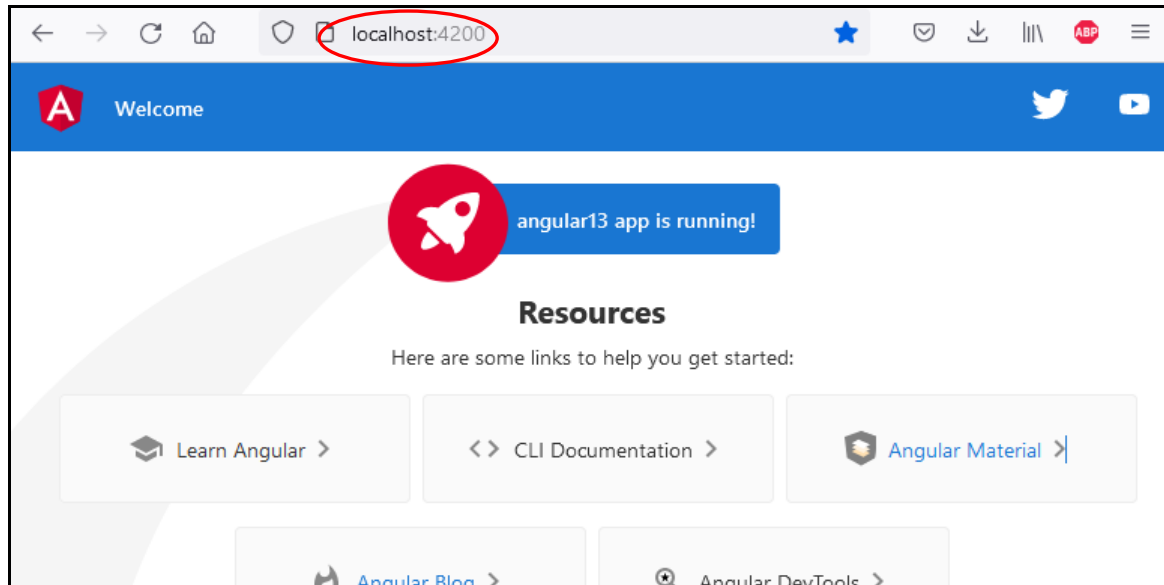
Falls man (wie manchmal am Schul-PC) Probleme mit PowerShell hat, kann man das möglicherweise mit folgendem Befehl beheben:

```
set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

2.3.2 Browser

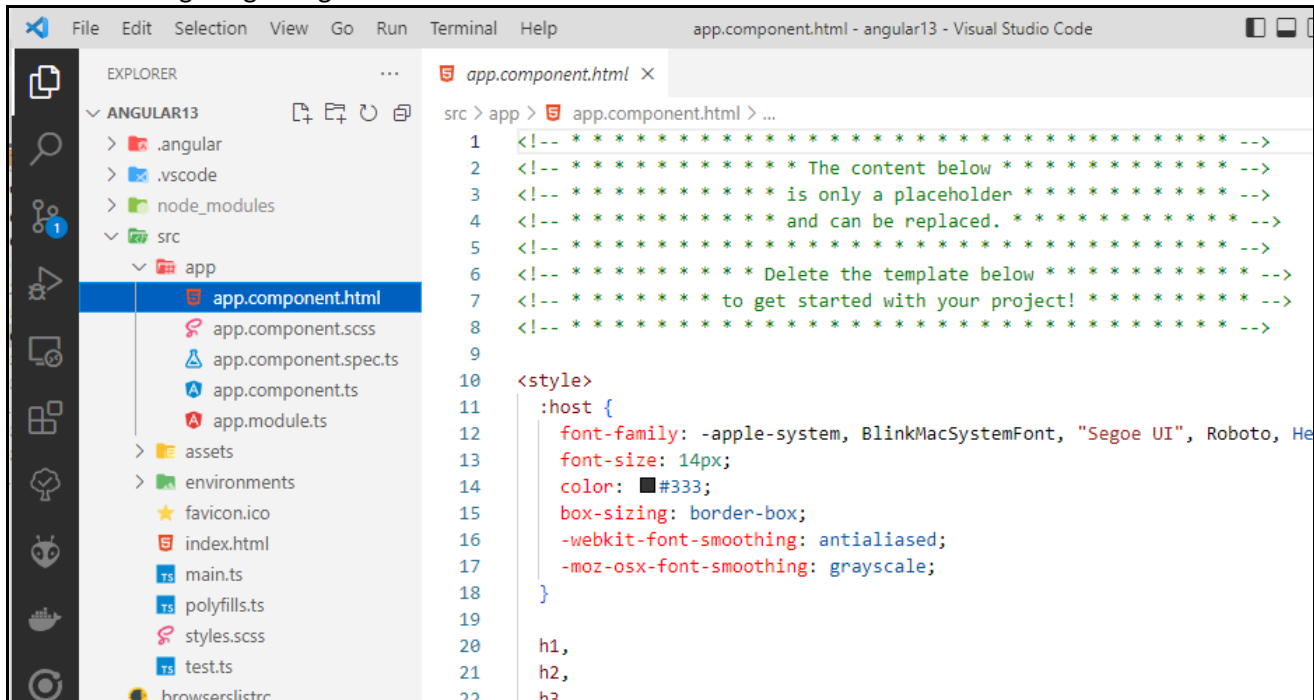
Es muss nur beim ersten Mal die URL in den Browser eingegeben werden, ab dann aktualisiert sich die App wie oben erwähnt automatisch.

Und so sieht es dann aus:



3 Entwicklungsumgebung

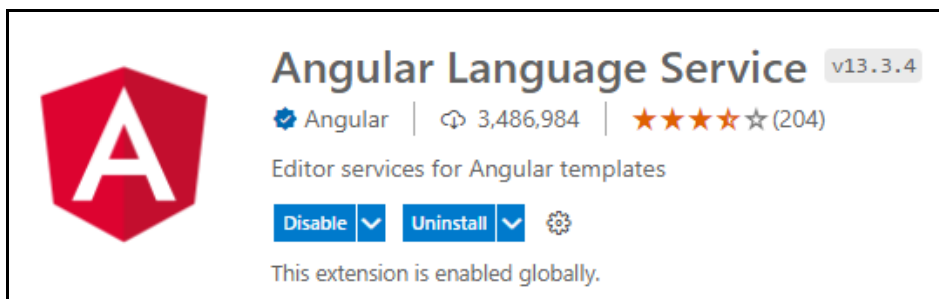
Als Entwicklungsumgebung bietet sich Visual Studio Code an.



3.1 Plugins

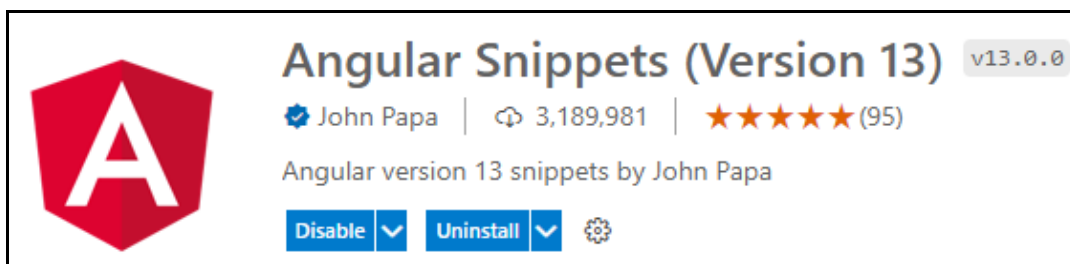
Folgende Extensions sollte man auf jeden Fall für Visual Studio Code installieren:

3.1.1 Angular Language Service



Dadurch wird im HTML-Editor erkannt, welche Properties und Funktionen in der zugehörigen Komponentenklasse existieren.

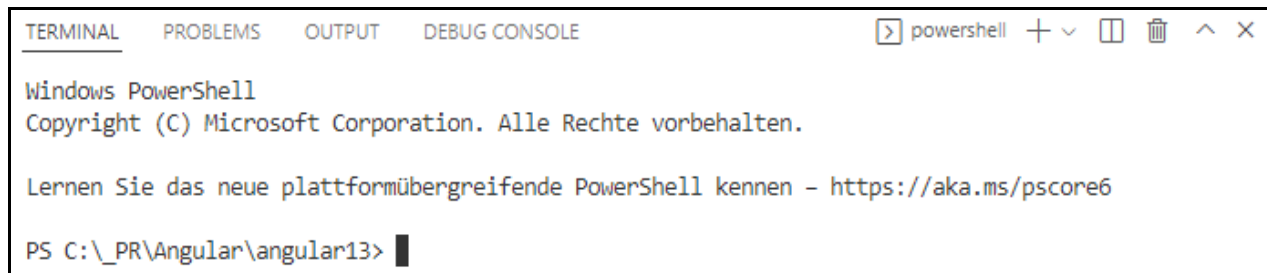
3.1.2 Angular Snippets



Derartige Plugins gibt es mehrere. Sie stellen Shortcuts zur Verfügung, mit denen Code-Blöcke eingefügt werden können.

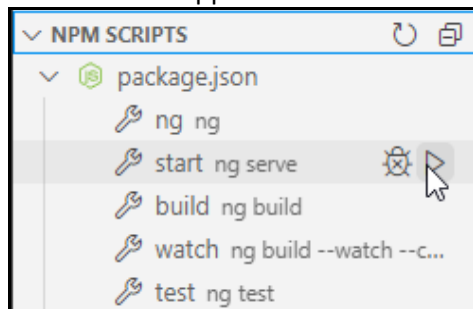
3.2 Terminal

Mit **<Strg>ö** kann man das Terminal einblenden, in dem man ebenfalls alle Befehle anstelle Konsole eingeben kann.



3.3 Scripts

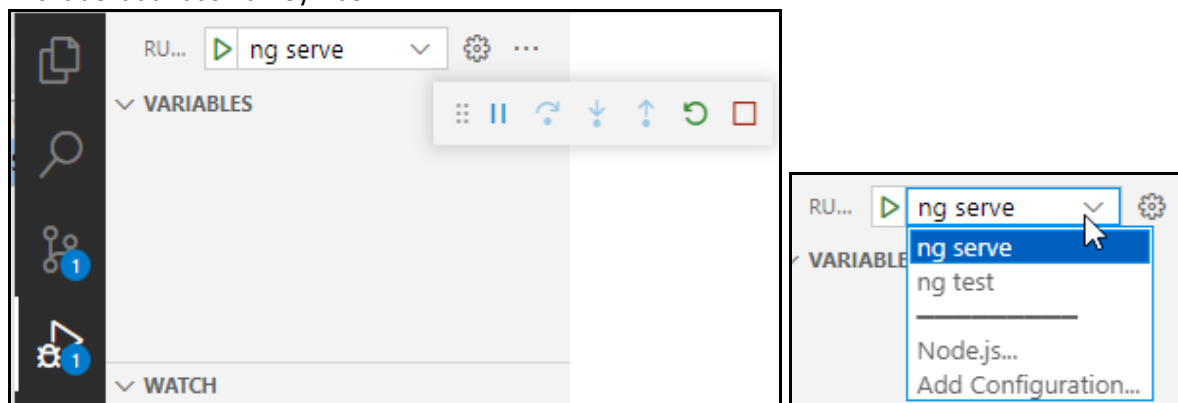
Man kann die App aber auch über ein npm-Script starten:



3.4 Debugging

Seit einiger Zeit braucht man keine Erweiterung mehr, um mit Visual Studio Code eine Angular-App debuggen zu können.

Das Debugger-Fenster wird mit dem „Käfer“-Symbol geöffnet (oder View → Debug oder **<Strg><Shift>D**) und klickt dort auf das Run-Symbol:



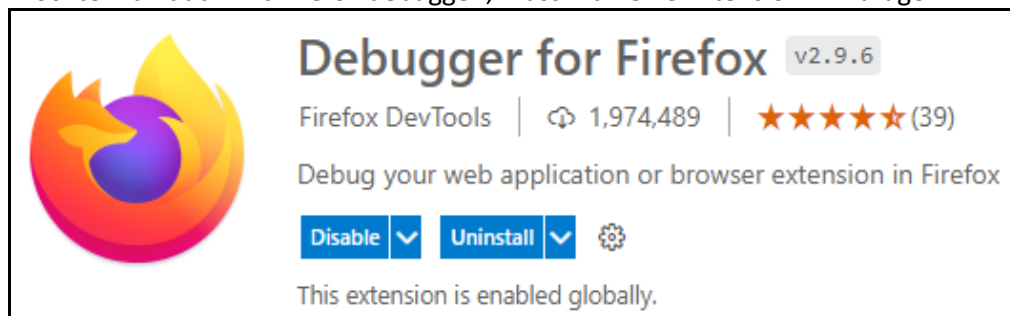
Die Start-Optionen sind in der Datei **.vscode\launch.json** gespeichert.

Per Default wird Chrome gestartet.

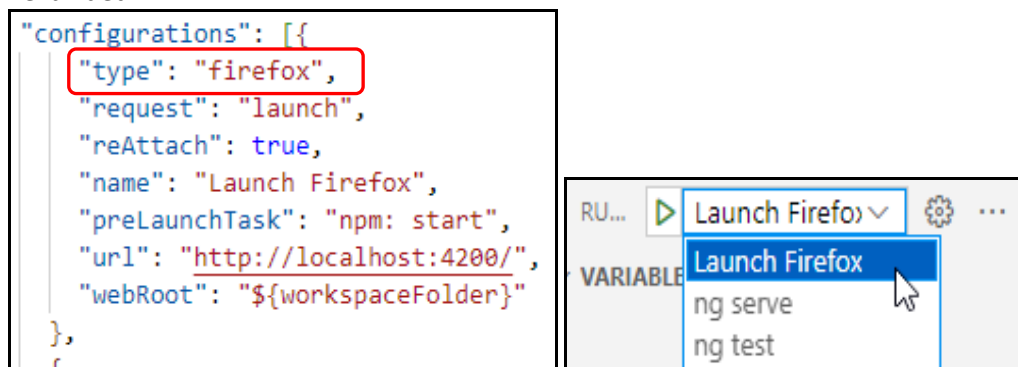
```
{
  // For more information, visit: https://go.micro
  "version": "0.2.0",
  "configurations": [
    {
      "name": "ng serve",
      "type": "pwa-chrome",
      "request": "launch",
      "preLaunchTask": "npm: start",
      "url": "http://localhost:4200/"
    },
    {
      "name": "ng test",
      "type": "chrome",
      "request": "launch",
      "preLaunchTask": "npm: test",
      "url": "http://localhost:9876/debug.html"
    }
  ]
}
```

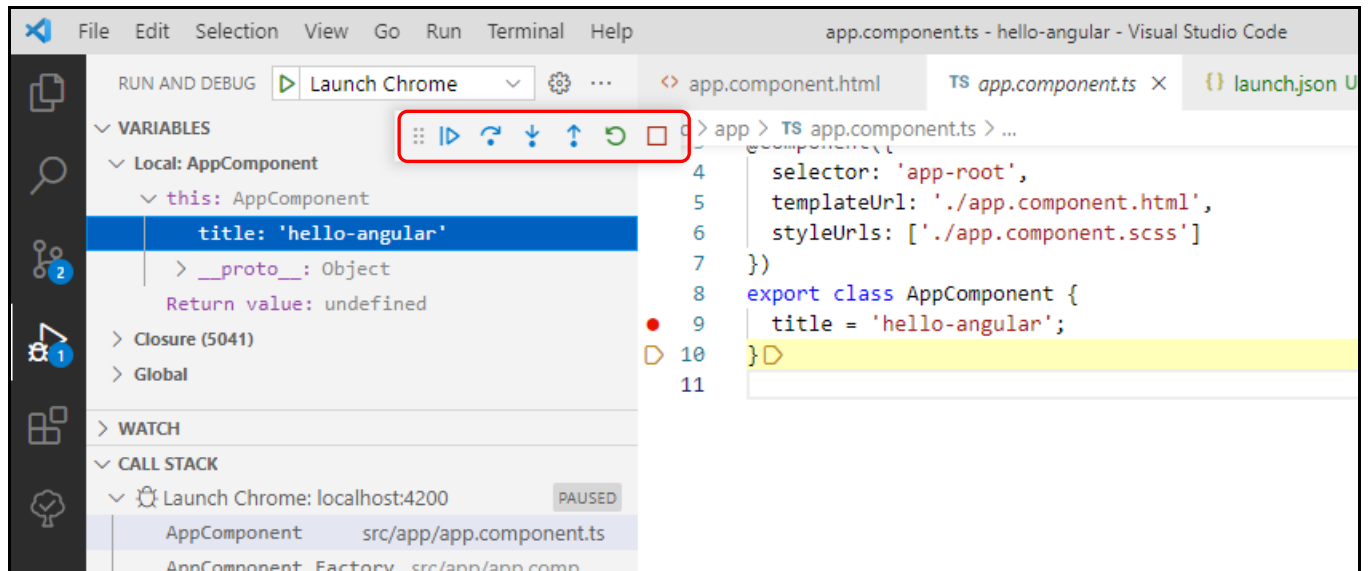
3.4.1 Firefox

Möchte man auch mit Firefox debuggen, muss man eine Extension hinzufügen:



Dort muss man dann die Launch-Configuration bearbeiten, damit sich der Debugger auf <http://localhost:4200> verbindet:





Die Bedienung ist wie von Debuggern gewohnt (Variable, Watch, Breakpoints, ...) sowie Step Over, Continue, Links hat man unter den Variablen auch den Callstack sowie weitere Tabs.

3.5 Production build

Damit man die App auch aus dem Explorer starten kann, schlage ich vor, folgende Batch Files zu erzeugen:

ng build --base-href=. --configuration production --optimization

```

"scripts": {
  "ng": "ng",
  "start": "ng serve",
  "build": "ng build",
  "build:production": "ng build --base-href=. --configuration production --optimization",
  "watch": "ng build --watch --configuration development",
  "test": "ng test"
},
  
```

Bei Ausführen des Skripts wird das Projekt in das Unterverzeichnis **dist** kompiliert. Es besteht aus einem html-File und 3-5 js-Files:

	Name	Größe
angular13		
.angular		
.git		
.vscode		
dist		
angular13		
node_modules		
src		
	3rdpartylicenses.txt	13 KB
	favicon.ico	1 KB
	index.html	1 KB
	main.c209b19b34d79ac0.js	117 KB
	polyfills.12fdf83aea81a901.js	34 KB
	runtime.0de40f8acb10cff6.js	2 KB
	styles.ef46db3751d8e999.css	0 KB

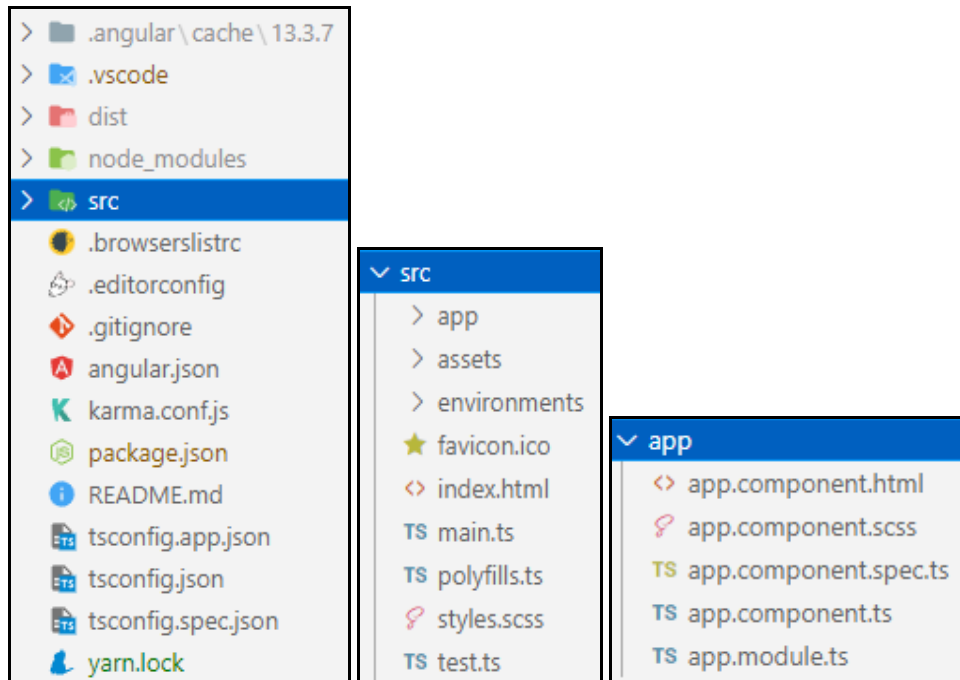
Die App kann aber nicht einfach durch Klick auf index.html im Browser angezeigt werden, sondern sie muss über einen Webserver provided werden, z.B. mit Live-Server:

```

C:\_PR\Angular\angular13\dist\angular13>live-server
Serving "C:\_PR\Angular\angular13\dist\angular13" at http://127.0.0.1:8080
Ready for changes
GET /sw.js 404 2.712 ms - 144
GET /node_modules/sw-toolbox/sw-toolbox.js 404 3.818 ms - 176
  
```

4 Dateiüberblick

Jetzt sollen noch die wichtigsten Dateien des Projekts besprochen werden. Die Client- bzw. Angular-Dateien befinden sich im Verzeichnis **src**:



4.1 index.html

In der einfachsten Form sieht die Startdatei so aus:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>HelloAngular</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>
```

Es fällt auf, dass keine Javascript-Includes vorhanden sind. Es wird alles über das Angular-Framework abgewickelt. Die Frage ist natürlich, was wird tatsächlich geladen?

Die Netzwerk-Anzeige in der Developerkonsole zeigt Folgendes:

Status	Meth...	Host	Datei	Initiator	Typ	Übertragen	Größe
200	GET	localhost:4200	/	document	html	849 B	565 B
200	GET	localhost:4200	styles.css	stylesheet	css	1,46 KB	1,18 KB
200	GET	localhost:4200	runtime.js	script	js	6,81 KB	6,52 KB
200	GET	localhost:4200	polyfills.js	script	js	295,14 KB	294,84 KB
200	GET	localhost:4200	styles.js	script	js	172,80 KB	172,51 KB
200	GET	localhost:4200	vendor.js	script	js	1,70 MB	1,70 MB
200	GET	localhost:4200	main.js	script	js	48,34 KB	48,04 KB
101	GET	localhost:4200	ws	polyfills.js:1021 ...	plain	129 B	0 B
200	GET	localhost:4200	favicon.ico	FaviconLoader.j...	vnd.m...	1,20 KB	948 B

Man sieht es auch, wenn man sich den tatsächlich geladenen Seiten-Quelltext ansieht:

```
<body>
  <app-root></app-root>
  <script src="runtime.js" type="module"></script><script src="polyfills.js" type="mo
</html>
```

Die Bundles entstehen durch WebPack, das alle benötigten Typescript-/Javascript-Module serverseitig bündelt und in Form von üblicherweise fünf Paketen an den Browser liefert.

Man muss sich darum aber keine genaueren Gedanken machen – wichtig ist, dass es funktioniert und die Anzahl der Requests an den Server niedrig hält.

4.2 main.ts – “Hauptprogramm”

Jede Angular-App benötigt zumindest ein Modul, das als Einsprungpunkt fungiert. Üblicherweise heißt diese Datei **main.ts**. Sie sieht zu so aus und muss/soll eigentlich nie verändert werden:

```
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.log(err));
```

Es wird das Start-Modul geladen und mit **bootstrapModule** aktiviert.

4.3 app-Folder

Im Verzeichnis **app** werden alle eigenen Angular-Dateien platziert. Zu Beginn sind fünf Dateien vorhanden:

```

v app
  <> app.component.html
  app.component.scss
  TS app.component.spec.ts
  TS app.component.ts
  TS app.module.ts
```

4.3.1 app.module.ts – Start-Modul

Jede Angular-App besteht aus mindestens einem Modul. Dieses Modul sieht zu Beginn so aus:


```
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 { }
```

Vergleiche main.ts:

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

Es werden zu Beginn alle benötigten Module geladen. Intern wird dazu das oben erwähnte WebPack verwendet. Dann wird mit dem **@NgModule**-Dekorator die Modulklass AppModule erzeugt. Mit einem **Dekorator** wird eine Klasse mit Metadaten erweitert. Dazu wird ein Objekt mit bestimmten Properties übergeben. Die wichtigsten sind dabei:

- **declarations**: welche **Komponenten** werden benötigt? Im Beispiel ist das die selbstprogrammierte Komponente AppComponent, die weiter unten beschrieben wird.
- **imports**: welche anderen **Module** werden benötigt? Das root-Modul benötigt immer (zumindest) das BrowserModule. Sehr oft wird auch das FormsModule benutzt und im Falle eines Zugriffs auf Webservern, auch das HttpClientModule (also praktisch immer).
- **bootstrap**: Welche der angegebenen Komponenten soll als **Start-Komponente** benutzt werden?

4.3.2 app.component.ts – Start-Komponente

Jede Angular-App benötigt zumindest eine root-Komponente. Komponenten sind die grundlegenden Bausteine einer App, indem sie eine View und die Funktionalität dahinter darstellen. Zu Beginn werden wir den Großteil des Codes in diesen Komponenten programmieren. Später werden wir den Code auf mehrere Module, Services, Pipes und Direktiven auslagern. Die einfachste Variante sieht so aus:

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  title = 'hello-net-core-angular';
  isOk: boolean;
```

```
<body>
| <app-root></app-root>
</body>
```

Zu Beginn werden wieder die benötigten Module geladen. Eine Klasse wird durch den **@Component**-Dekorator zu einer Angular-Komponente. Wie schon beim main-Modul wird der **Dekorator** über ein Objekt konfiguriert. Die wichtigsten Properties sind dabei:

- **selector**: Welcher Tag im HTML-File wird durch diese Komponente ersetzt? Hier kommt also der oben erwähnte Tag in index.html ins Spiel.

Eigentlich ist es ein ganz normaler CSS-Selektor. Für Komponenten sollte aber ein **HTML-Tag** (und kein HTML-Klassenattribut) angegeben werden.

- **template / templateUrl**: Welches HTML soll durch die Komponente erzeugt werden? Das entspricht also der View der Komponente und ist daher ein entscheidender Teil jeder Angular-App. Dabei werden die Daten der Klasse über eine Databinding-Syntax gebunden. Das wird im nächsten Kapitel besprochen.

Die Klasse **AppComponent** wird durch **export class** exportiert und kann daher von anderen Komponenten und Modulen eingebunden werden. Das ist oben bei main.ts der Fall, wo ja die Komponente App als root-Komponente angegeben wurde.

4.3.2.1 app.component.html

Diese Datei stellt den Inhalt der Komponente dar.

```
330 <!-- Highlight Card -->
331 <div class="card highlight-card card-small">
332
333 > <svg id="rocket" xmlns="http://www.w3.org/2000/svg" width="100" height="100">
342 </svg>
343
344 <span>{{ title }} app is running!</span>
345
346 <svg id="rocket-smoke" xmlns="http://www.w3.org/2000/svg" width="100" height="100">
347 <title>Rocket Ship Smoke</title>
348 <path id="Path_40" data-name="Path 40" d="M644.6,141S143.6,141,143.6,141" />
349 </svg>
350
351 </div>
```

4.3.2.2 app.component.scss

Wie der Name vermuten lässt, stehen darin die Styles der Komponente. Die Datei ist zu Beginn leer.

4.3.2.3 app.component.spec.ts

Diese Datei ist für Unit-Tests zuständig.

4.4 Infrastruktur

Neben den eigenen Dateien gibt es einige Dateien, die man nicht verändern muss, die aber (vor allem von Node) benutzt werden. Die wichtigsten sind:

4.4.1 package.json

Diese Datei ist die JSON-Konfigurationsdatei für Node. Hier werden die Node-Module angegeben, die geladen werden sollen. Sie wurde weiter oben bei **Restore Packages** aufgerufen.

Das hat aber nichts mit Angular zu tun, sondern ist bei Node-Modulen prinzipiell so.

Die benötigten Module werden unter **dependencies** (für die Laufzeit) bzw. und **devDependencies** (für die Entwicklung) angegeben. Hier findet man neben den eigentlichen Angular-Modulen auch jene wie zone.js, jasmine oder karma (Unit-Tests). Die Versionen sind die aktuellen mit Stand 2022-06-02 (hängen aber von der installierten Version von @angular/cli ab):

```
"dependencies": {
  "@angular/animations": "~13.3.0",
  "@angular/common": "~13.3.0",
  "@angular/compiler": "~13.3.0",
  "@angular/core": "~13.3.0",
  "@angular/forms": "~13.3.0",
  "@angular/platform-browser": "~13.3.0",
  "@angular/platform-browser-dynamic": "~13.3.0",
  "@angular/router": "~13.3.0",
  "rxjs": "~7.5.0",
  "tslib": "^2.3.0",
  "zone.js": "~0.11.4"
},
```

```
"devDependencies": {
  "@angular-devkit/build-angular": "~13.3.7",
  "@angular/cli": "~13.3.7",
  "@angular/compiler-cli": "~13.3.0",
  "@types/jasmine": "~3.10.0",
  "@types/node": "^12.11.1",
  "jasmine-core": "~4.0.0",
  "karma": "~6.3.0",
  "karma-chrome-launcher": "~3.1.0",
  "karma-coverage": "~2.1.0",
  "karma-jasmine": "~4.0.0",
  "karma-jasmine-html-reporter": "~1.7.0",
  "typescript": "~4.6.2"
}
```

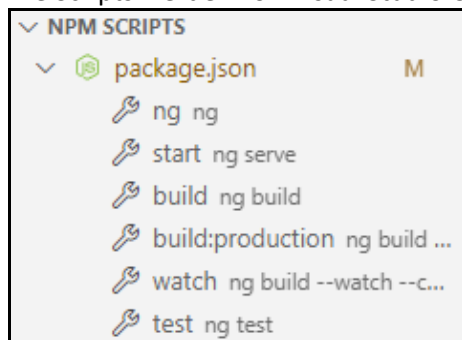
Einige Scripts sind bereits dabei unter der Property **scripts** eingetragen:

```
"scripts": {
  "ng": "ng",
  "start": "ng serve",
  "build": "ng build",
  "watch": "ng build --watch --configuration development",
  "test": "ng test"
},
```

Diese Befehle bedeuten dabei Folgendes:

start	Übersetzt Typescript in Javascript, startet den Server und aktualisiert den Browser, wenn sich eine Date ändert
build	Erzeugt das Angular-Projekt
watch	Erzeugt das Projekt neu bei Code-Änderung
test	Unit-Tests

Die Scripts werden von Visual Studio Code ausgelesen und über den Typ „NPM SCRIPTS“ zur Verfügung gestellt:



Man könnte die einzelnen Skripts aber auch von der Konsole aus als Parameter bei npm ausführen. Der Aufruf erfolgt dann eben z.B. mit **npm run build**.

```
C:\_PR\Angular\angular13>npm run build
npm WARN config global `--global`, `--local` are deprecated
> angular13@0.0.0 build
> ng build

Browser application bundle generation complete
```

4.4.2 styles.scss

Wie gewohnt werden hier die allgemeinen CSS-Styles angegeben. Oben haben wir aber gesehen, dass Komponenten eigene Stylesheets haben können/sollten.

4.4.3 tsconfig.app.json / tsconfig.json

Hier steht die Konfiguration für den Typescript-Compiler. Diese Dateien sind üblicherweise nicht zu verändern.

```
tsconfig.json > ...
1  /* To learn more about this file see: https://
2  {
3    "compileOnSave": false,
4    "compilerOptions": {
5      "baseUrl": "./",
6      "outDir": "./dist/out-tsc",
7      "forceConsistentCasingInFileNames": true,
8      "strict": true,
9      "noImplicitReturns": true,
10     "noFallthroughCasesInSwitch": true,
11     "sourceMap": true,
12     "declaration": false,
13     "downlevelIteration": true,
14     "experimentalDecorators": true,
15     "moduleResolution": "node",
16     "importHelpers": true,
17     "target": "es2017",
18     "module": "es2020",
19     "lib": [
20       "es2018",
21       "dom"
22     ]
23   },
24   "angularCompilerOptions": {
25     "enableI18nLegacyMessageIdFormat": false,
26     "strictInjectionParameters": true,
27     "strictInputAccessModifiers": true,
28     "strictTemplates": true
29   }
30 }
```

```
tsconfig.app.json > ...
1  /* To learn more about this file
2  {
3    "extends": "./tsconfig.json",
4    "compilerOptions": {
5      "outDir": "./out-tsc/app",
6      "types": []
7    },
8    "files": [
9      "src/main.ts",
10     "src/polyfills.ts"
11   ],
12   "include": [
13     "src/**/*.d.ts"
14   ]
15 }
```