2 Vorüberlegungen

Für die Realisierung des Frontends der Webanwendung sind einige Grundkenntnisse bezüglich Angular und NgRx notwendig. Diese werden in diesem Kapitel näher erläutert.

Das Frontend ist das User Interface (UI) der Webanwendung und bildet somit eine Grafikoberfläche, mit der der User Zugriff auf die Funktionen des Backends hat, ohne das dieser Kenntnisse zu der Funktionsweise oder dem Aufbau des Backends benötigt.

2.1 Angular

Angular ist ein Framework, also ein Programmiergerüst, zur Erstellung von webbasierten Nutzerapplikationen mit Hilfe von Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) und TypeScript (TS). Dabei werden sowohl grundlegende als auch optionale Funktionen als TS-Bibliotheken implementiert, welche dann in Applikationen eingebunden werden können.[5]

2.1.1 Modules

Die als NgModules bezeichneten elementaren Bestandteile einer Angular Applikation erzeugen einen Erstellungskontext für Components. Eine Angular Anwendung besitzt immer ein Hauptmodul. Neben dem Hauptmodul kann eine Angular Anwendung mehrere Funktionsmodule besitzen. Ein Modul besitzt immer mindestens eine Component, bei größeren Anwendungen sind beliebig viele Components möglich.[6]

Das Hauptmodul ermöglicht Bootstrapping, also das Starten komplexerer Abläufe in der Anwendung, wie zum Beispiel das Initialisieren der Anwendung. Ein Modul kann Relationen zwischen den eigenen Components und anderweitig verbundenem Code, wie zum Beispiel Services, aufbauen um Funktionseinheiten zu bil-

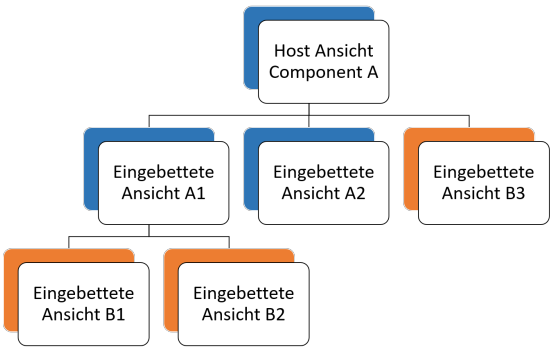
[5] Vgl. Google LLC, Angular Architektur, o.J.
[6] Vgl. Google LLC, Angular Modules, o.J.

den. Funktionen können auch unter Modulen weitergegeben oder wiederverwendet werden, wenn diese entsprechend exportiert und importiert werden.[7]

2.1.2 Components

Eine Component hat die Kontrolle über einen Teil des Bildschirms, dieser Teil wird als Ansicht bezeichnet und entsteht aus dem Zusammenspiel einer in der Component definierten Klasse, welche Metadaten und Logik enthält, und dem dazugehörigen HTML-Template, welches den Aufbau der Ansicht beinhaltet. Die in der Klasse definierten Metadaten beinhalten unter anderem die Referenz zu dem dazugehörigen Template, einen Selector, mit dem die Component im Template aller anderen Components aufgerufen kann sowie einen Constructor, welcher eine Grundlage für Dependency Injection (DI) bildet. Die Gesamtansicht einer Angular Webapplikation im Browser kann aus mehreren Ansichten unterschiedlicher Components zusammengesetzt sein, welche alle in einer übergeordneten Hostansicht liegen und wiederum selbst die Hostansicht ihrer Component sein können, die ebenfalls mehrere eingebettete Ansichten beinhalten kann. Dadurch entstehen Ansichtshierarchien wie in Abbildung 1 dargestellt. Aufgrund dieses modularen Aufbaus können Ansichten leicht unabhängig voneinander geändert oder auch entfernt werden, ohne dabei die anderen Ansichten ändern zu müssen, wodurch sich die Anwendung leichter pflegen lässt.[8]

Abbildung 1: Ansichtshierarchie



Quelle: Eigene Darstellung in Anlehnung an Google LLC, Angular Modules, o.J.

[7] Vgl. Google LLC, Angular Architektur, o.J.
[8] Vgl. Google LLC, Angular Components, o.J.

Das Template einer Ansicht besteht aus regulärem HTML sowie zusätzlichem Angular Syntax, mit dem das HTML entsprechend der in der Component definierten Logik, dem aktuellen Status der Anwendung und auch der Document Object Model (DOM)-Daten verändert werden kann. Die Daten können durch das Template auch vor der Ausgabe verändert oder selektiert werden. Anhand von Abbildung 2 kann man das Zusammenspiel von standardmäßigem HTML Syntax mit Tags wie <ul> und <li> für eine Liste und Angular Syntax wie beispielsweise *ngFor zum Iterieren über die Liste sehen. Angular manipuliert in diesem Beispiel das Template durch die *ngFor Direktive. Obwohl nur ein Listenelement mit <li> vorhanden ist wird dennoch eine komplette Liste angezeigt.[9]

Abbildung 2: Template Beispiel

```
1 <ul>
2     <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
3         {{hero.name}}
4     </li>
5 </ul>
6
7 <app-hero-detail *ngIf="selectedHero"
    [hero]="selectedHero"></app-hero-detail>
```

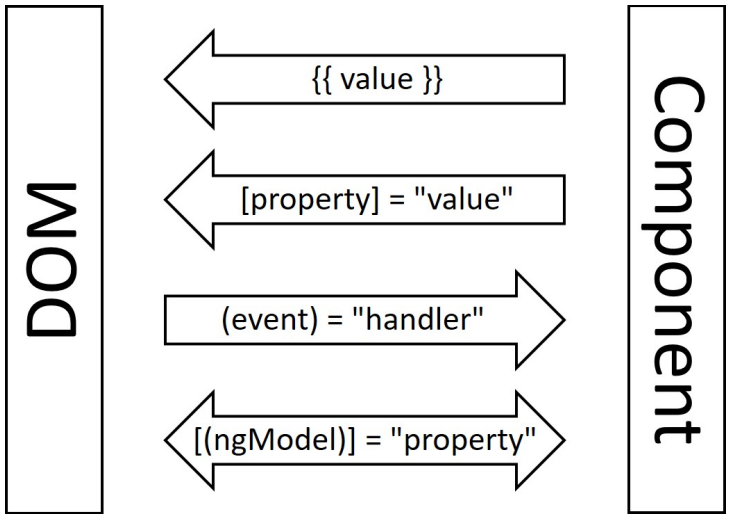
Quelle: Google LLC, Angular Components, o.J.

In Abbildung 2 werden unterschiedliche Möglichkeiten zur Datenbindung in Angular aufgezeigt. Daten können in unterschiedliche Richtungen gebunden werden, entweder zum DOM, vom DOM oder in beide Richtungen. Dies wird in Abbildung 3 veranschaulicht.[10]

Wenn Daten in Richtung DOM gebunden sind, wird auf diese lediglich lesend zugegriffen. Sind sie in Richtung Component gebunden, dann lösen sie einen Vorgang in der Component aus. In Abbildung 2 sorgt das „click“-Event dafür, dass eine Funktion in der Component aufgerufen wird, ebenso ist es auf diese Art auch möglich Daten direkt zu ändern. Falls Daten in beide Richtungen gebunden sind können sie sowohl angezeigt als auch geändert werden. Die Änderungen werden hierbei in Echtzeit wieder an das Template weitergeleitet, wodurch die Än-

[9] Vgl. Google LLC, Angular Components, o.J.
[10] Vgl. ebd.

Abbildung 3: Angular Datenbindung



Quelle: Eigene Darstellung in Anlehnung an Google LLC, Angular Components, o.J.

derungen direkt wieder angezeigt werden können ohne die Website neuladen zu müssen.[11]

Auf diese Weise eingebundene Daten müssen jedoch nicht unbedingt entsprechend ihres Wertes ausgegeben werden, die Ausgabe kann von Angular manipuliert werden. Dazu stellt Angular sogenannte Pipes zur Verfügung, welche über den Pipe Operator „|“ genutzt werden können. Neben bereits von Angular bereitgestellten Pipes können auch eigene Pipes definiert und angewendet werden.[12]

Abbildung 4: Pipe Template

```
1 <div>
2   <p>{{greeting}}</p>
3   <p>{{greeting | uppercase}}</p>
4   <p>{{greeting | lowercase}}</p>
5   <p>{{greeting | titlecase}}</p>
6   <p>{{greeting}}</p>
7 </div>
```

Quelle: Eigene Darstellung

Nimmt man für die Variable „greeting“ aus Abbildung 4 einen Wert von „guten

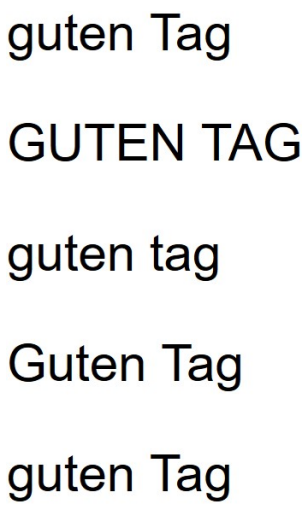
[11] Vgl. Google LLC, Angular Components, o.J.
[12] Vgl. ebd.

Tag“ an, so wird für das in Abbildung 4 aufgeführte Template die in Abbildung 5 gezeigte Ansicht im Browser dargestellt. In diesem Beispiel sorgen Pipes dafür, dass der in der Component definierte Wert der Begrüßung geändert wird. Zeile 6 von Abbildung 4 und die korrespondierenden Ausgabe in Abbildung 5 zeigen das lediglich die Ausgabe, nicht jedoch der Wert der Variable direkt, verändert wird, da die Ausgabe wieder dem Ausgangswert entspricht.[13]

2.1.3 Services und Dependency Injection

Im Normalfall werden Daten nicht direkt in der Component gespeichert, es werden vielmehr Modelle definiert, die aufzeigen von welcher Struktur die Daten sind, die dann zum Beispiel von Servern abgerufen werden. Components sollten die Daten nicht direkt selbst vom Server abfragen, da ansonsten in jeder Component, die auf Daten zugreift, eine entsprechende Funktion zur Abfrage von unterschiedlichen Daten vorhanden sein müsste. Dadurch erhöht sich der Aufwand zum Ändern einer solchen Abfrage enorm, da diese dann in jeder Component geändert werden muss. Aus diesem Grund wird diese Aufgabe oft an Services delegiert.[14]

Abbildung 5: Pipe Ausgabe



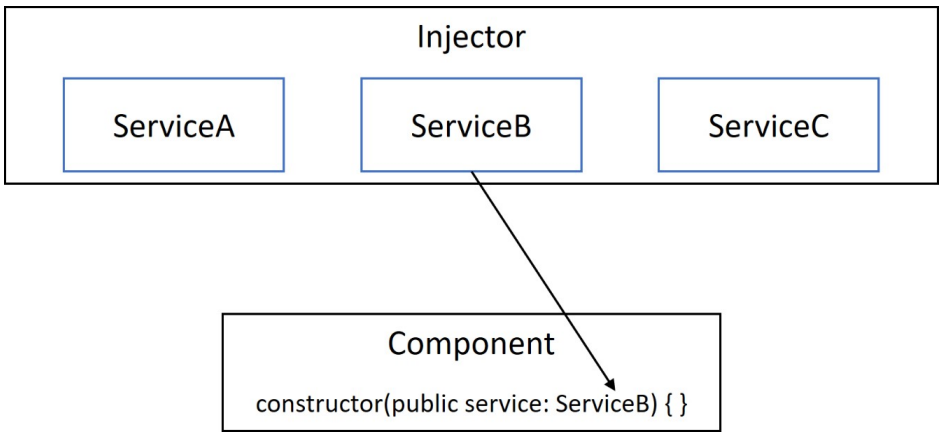
Quelle: Eigene Darstellung

Services sind Klassen mit einem genau definierten Zweck, wie zum Beispiel das Abfragen von Serverdaten. Durch DI können Services allen Components zur Verfügung gestellt werden, die die Funktionalität des Services benötigen. Services können ebenso wie Components untereinander verknüpft werden, sodass die Möglichkeit besteht einen übergeordneten Service zu erstellen, welcher auf andere Services zugreift.[15]

[13] Vgl. Google LLC, Angular Components, o.J.
[14] Vgl. Google LLC, Angular Services, o.J.
[15] Vgl. ebd.

Während des Bootstrapping-Prozesses erstellt Angular automatisch einen Injector, durch den DI möglich wird. Dieser Injector erstellt Abhängigkeiten und behält Instanzen dieser Abhängigkeiten in einer Art Container, dadurch müssen nicht mehrere Instanzen erstellt werden, sofern mehrere Components von dem gleichen Service abhängig sind. Damit der Injector solche Abhängigkeiten erstellen kann, müssen Provider festgelegt werden, die dem Injector mitteilen, wie eine entsprechende Abhängigkeit erstellt werden kann. Bei der Instanziierung einer Component bestimmt Angular anhand des Constructors der Component, welche Abhängigkeiten erstellt werden sollen. Diese Vorgänge werden in Abbildung 6 dargestellt.[16]

Abbildung 6: Service Injection



Quelle: Eigene Darstellung in Anlehnung an Google LLC, Angular Services, o.J.

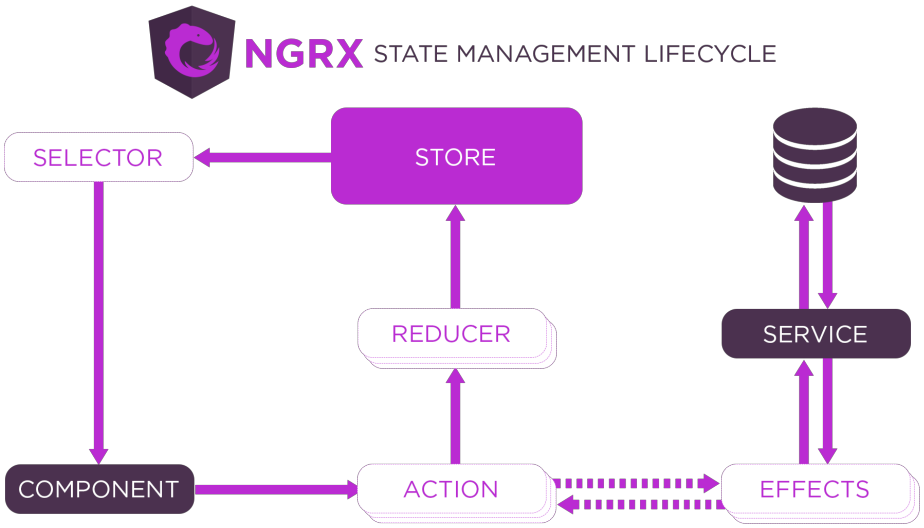
2.2 NgRx

Ebenso wie Angular ist NgRx ein Framework. Mit NgRx lassen sich reaktive Anwendungen einfach und schnell in Angular erstellen. Es stellt unter anderem eine Zustandsverwaltung (State Management), automatische Codegenerierung sowie verschiedene Entwicklertools zur Verfügung. Bei der Zustandsverwaltung wird ein Gesamtzustand bestehend aus mehreren Unterzuständen initialisiert, welcher dann verändert werden kann. Das Framework ist in mehrere eigenständige Module aufgeteilt, sodass jeweils nur die benötigten Teile in die Anwendung einge-

[16] Vgl. Google LLC, Angular Services, o.J.

bunden werden müssen. @ngrx/store ist hierbei die Hauptkomponente, welche die Nutzung von sogenannten „Actions“, „Reducers“ und „Selectors“ ermöglicht.[17] Ein weiteres wichtiges Modul ist @ngrx/effects. Dieses stellt sogenannte „Effects“ zur Verfügung, welche essentiell für die Kommunikation zwischen Store und Services sind. Dementsprechend werden sie für die Kommunikation zu externen Datenquellen, wie zum Beispiel Servern, benötigt.[18] Der Aufbau dieser Gesamtstruktur wird in Abbildung 7 dargestellt.

Abbildung 7: NgRx State Management Lifecycle



Quelle: NgRx, Store, o.J.

2.2.1 Actions

Actions beschreiben unterschiedliche Ereignisse, die innerhalb eines Programmes ablaufen können, so zum Beispiel Interaktionen mit Nutzern, externen Servern oder auch der Austausch mit der Geräte-API. Sie stellen hierbei die Eingaben und Ausgaben verschiedenster Mechanismen innerhalb von NgRx dar. Actions können als Klasse definiert werden. Diese Klassen beinhalten einen Type, der die Action beschreibt und einen Kontext bereitstellt, von wo die Action versendet wird und zu welcher Kategorie die Action gehört.[19]

[17] Vgl. NgRx, Store, o.J.
[18] Vgl. NgRx, Effects, o.J.
[19] Vgl. NgRx, Actions, o.J.

Die Dokumentation zu NgRx gibt für die Nutzung von Actions folgende Leitlinien zur Erstellung an:

- Im Voraus: Actions sollten geschrieben werden, bevor eine Funktionalität entwickelt wird, sodass die Funktionalität an sich besser verstanden wird.
- Geteilt: Actions sollten entsprechend ihrer Ereignisquelle aufgeteilt werden.
- Viele: Das Schreiben von Actions ist mit wenig Aufwand verbunden, daher sollten möglichst viele verwendet werden, damit Abläufe besser abgebildet werden können.
- Ereignisgesteuert: Actions beschreiben Ereignisse und nicht Befehle. Das Beschreiben der Ereignisse ist vom Umgang mit den Ereignissen getrennt.
- Beschreibend: Es soll ein Kontext für das Ereignis zur Verfügung gestellt werden um das Debuggen mit Entwicklerwerkzeugen zu erleichtern.[20]

2.2.2 Reducers

Reducers steuern die Übergänge von einem Zustand in einen anderen. Dabei wird anhand des Typs der Action ermittelt, was gesteuert werden sollen. Sie geben für eine bestimmte Eingabe stets dieselbe Ausgabe zurück, daher sind Reducers sogenannte „pure functions“. Jeder Reducer entscheidet anhand der zeitlich gesehen neuesten Action und dem aktuellen Zustand des Stores ob ein modifizierter Zustand oder der Originalzustand zurückgegeben werden soll. Ein Reducer besteht aus drei Teilen:[21]

- Ein Interface, welches die Form des Zustands beschreibt, also die Art der Daten und Variablen die abgebildet und geändert werden sollen.
- Ein Initialzustand, der den Zustand direkt nach der Initialisierung des Stores darstellt.

[20] Vgl. NgRx, Actions, o.J.
[21] Vgl. NgRx, Reducers, o.J.

- Die Funktion, welche die Zustandsänderungen anhand der anliegenden Actions durchführt.[22]

2.2.3 Selectors

Selectors werden benutzt, um Teile des Zustands auszuwählen, sodass auf die Daten des benötigten Teilzustands zugegriffen werden kann. Es können auch innerhalb eines Selectors mehrere Teilzustände gleichzeitig ausgewählt werden, sofern dies benötigt wird. Da Selectors ebenso wie Reducers zu den pure functions gehören, kann man sich bei ihrer Nutzung eine Technik namens Memoisation zunutze machen. Hierbei werden die Rückgabewerte einer Funktion gespeichert. Dadurch müssen Werte nicht neu berechnet werden, sofern sie bereits einmal berechnet wurden und wieder dieselben Eingabeparameter vorliegen. Somit werden diese Prozesse beschleunigt, da die erneute Rechenzeit entfällt, was sich vor allem bei Selectors mit aufwendigen Berechnungen bemerkbar macht.[23]

2.2.4 Effects

Effects liegen an Datenströmen von sogenannten Observables von Actions aus dem Store an und interagieren mit ankommenden Actions. Je nach Art der Action können entsprechend programmierte Effects neue Actions oder andere Events auslösen. Dieses Auslösen ist unter anderem abhängig von externen Interaktionen, wie beispielsweise Netzwerkanfragen. Sie trennen hierbei, wie in Abbildung 7 zu sehen, Components und Services weiter auseinander, wodurch die Anwendung modularer gestaltet wird und das Ziel von einfachen Components, welche lediglich Logiken zur Verfügung stellen und nicht selbst andere Funktionen übernehmen sollten, weiter erreicht wird.[24]

[22] Vgl. NgRx, Reducers, o.J.
[23] Vgl. NgRx, Selectors, o.J.
[24] Vgl. NgRx, Effects, o.J.

Literaturverzeichnis

Google LLC	(Angular Architektur) Architecture overview, o.J., https://angular.io/guide/architecture (besucht am 27. 08. 2019).
Google LLC	(Angular Modules) Introduction to modules, o.J., https://angular.io/guide/architecture-modules (besucht am 28. 08. 2019).
Google LLC	(Angular Components) Introduction to components, o.J., https://angular.io/guide/architecture-components (besucht am 28. 08. 2019).
Google LLC	(Angular Services) Introduction to services and dependency injection, o.J., https://angular.io/guide/architecture-services (besucht am 05. 09. 2019).
HOMAG Group AG	(HOMAG Geschichte), o.J., https: //www.homag.com/unternehmen/geschichte-innovationen/ (besucht am 21. 08. 2019).
HOMAG Group AG	(HOMAG Informationen), o.J., https://www.homag.com/unternehmen/homag-group/ (besucht am 20. 08. 2019).
Kaib, Michael	(Integration) Enterprise Application Integration - Grundlagen, Integrationsprodukte, Anwendungsbeispiele, Wiesbaden: Deutscher Universitätsverlag, 2002.
NgRx	(Store) @ngrx/store, o.J., https://ngrx.io/guide/store (besucht am 27. 08. 2019).
NgRx	(Effects) @ngrx/effects, o.J., https://ngrx.io/guide/effects (besucht am 06. 09. 2019).
NgRx	(Actions) Actions, o.J., https://ngrx.io/guide/store/actions (besucht am 23. 08. 2019).
NgRx	(Reducers) Reducers, o.J., https://ngrx.io/guide/store/reducers (besucht am 23. 08. 2019).

NgRx (Selectors) Selectors, o.J., https://ngrx.io/guide/store/selectors (besucht am 23. 08. 2019).

tapio GmbH (tapio) Was für Lösungen bietet tapio?, o.J., https://www.tapio.one/de/solutions (besucht am 21. 08. 2019).

Zillgens, Christoph (Responsive Design) Responsive Webdesign - Reaktionsfähige Websites gestalten und umsetzen, München: Carl Hanser Verlag, 2012.