FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

Titel

Untertitel

Bachelorarbeit

Eingereicht in teilweiser Erfüllung der Anforderungen zur Erlangung des akademischen Grades:

Bachelor of Science in Engineering

an der FH Campus Wien
Studienfach: Computer Science and Digital Communications

Autor:
Florian Mayerhofer

Matrikelnummer:
52007397

Betreuer:
René Goldschmid, MSc

Datum:
tt.mm.yyyy

Erklärung der Urheberschaft:

Ich erkläre hiermit diese Bachelorarbeit eigenständig verfasst zu haben. Ich habe keine anderen Quellen, als die in der Arbeit gelisteten verwendet, noch habe ich jegliche unerlaubte Hilfe in Anspruch genommen

Ich versichere diese Bachelorarbeit in keinerlei Form jemandem Anderen oder einer anderen Institution zur Verfügung gestellt zu haben, weder in Österreich noch im Ausland.

Weiters versichere ich, dass jegliche Kopie (gedruckt oder digital) identisch ist.

Datum: Unterschrift:

Abstract

(E.g. “This thesis investigates...”)

Kurzfassung

(Z.B. “Diese Arbeit untersucht...”)

Abkürzungen

ARP	Address Resolution Protocol
GPRS	General Packet Radio Service
GSM	Global System for Mobile communication
WLAN	Wireless Local Area Network

Schlüsselbegriffe

GSM
Mobilfunk
Zugriffsverfahren

Inhaltsverzeichnis

1 Einführung

Textkörper mit Bild

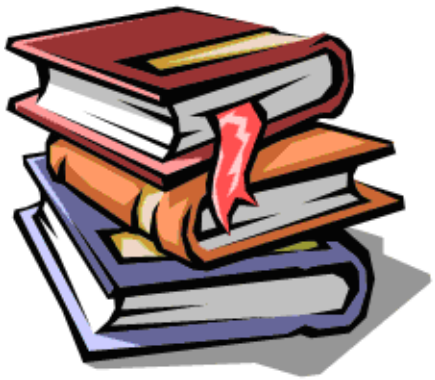

Abbildung 1.1: Ein Stapel Bücher

Textkörper Fortsetzung mit Verweis auf den wundervollen Stapel Bücher in Abbildung 1.1.

1.1 Motivation

1.2 State of the art

1.2.1 Unter-Unterkapitel 11

Textkörper mit direktem Zitat und Seitenanzahl: “It would be very easy to show how technical or report writing differed from other writing” [**?**, p. 3].

1.2.2 Unter-Unterkapitel 12

Textkörper mit Referenzen: Für weiterführende Informationen zum wissenschaftlichen Schreiben siehe "J. Schimel, Writing Science"[**?**]. Es wird empfohlen den Sprachleitfaden der FH Campus Wien [**?**] zu berücksichtigen und die Checkliste für wissenschafltiches Schreiben [**?**] zu verwenden. Beide Leitfäden sind im FH Portal zu finden.

2 Hauptteil

2.1 Frontend Frameworks

Frontend Frameworks sind Tools, die im allgemeinen auf JavaScript basieren. Diese fassen etwaige Funktionalitäten, Prozesse sowie Abhängigkeiten im Web Development Bereich zusammen. Dadurch ist es nicht nötig ständig wiederholende Funktionen, die jede Webseite besitzt, zu implementieren, da dies bereits vom Framework übernommen wurde. Zudem bieten diese Frameworks Hilfestellung bezüglich Deployment, Building und Testing der Webseite, sodass diese Funktionalitäten souverän in einem Projekt implementiert und eingebunden werden können. Da heutzutage Web Applikation immer komplexer und größer werden, ist eine Web Applikation Implementierung ohne Frontend Frameworks undenkbar und mittlerweile Standard in der Entwicklung. Einer der bekanntesten Frontend Frameworks sind Angular, React und Vue.js wobei mittlerweise Erweiterungen für React und Vue.js existieren, nämlich Next.js und Nuxt.js. Next.js baut auf das Frontend Framework React auf und Nuxt.js basiert auf Vue.js. Angular ist ein vollständiges Framework, da es bereits schon sämtliche Funktionen beinhaltet, um eine komplexe Web Applikation zu entwickeln. Wo hingegen React und Vue.js lediglich Libraries sind. Für diese muss man zusätzliche Libraries einfügen, um auf den Funktionsumfang von Angular zu kommen.

2.2 Angular

Angular ist zur Zeit einer der bekanntesten Frontend Frameworks, welches für die Entwicklung von Web Applikationen verwendet wird. Angular ist eine sogenannte Single-Page Applikation. Dies bedeutet, dass eine Html-Seite gerendert wird. Wechseln die Nutzende auf eine andere Seite, innerhalb der Web Applikation, so wird die Seite an sich nicht gewechselt, sondern nur der Inhalt. Angular basiert auf Components, welche Bestandteile der Webseite darstellen und benutzt werden, um diese zusammenzubauen. Dieses Framework ist ein vollständiges Framework, da Angular bereits alle relevanten Libraries und Funktionalitäten beinhaltet, um die Implementierung einer komplexen und skalierbaren Webseite zu ermöglichen. Routing, Forms Management, Client-server Kommunikation sind nur einige Beispiele dafür. Angular hatte eine Vorgängerversion namens AngularJS. Nach der Veröffentlichen von Angular 2 im September 2016 wurde AngularJS abgelöst. Dieses Framework basiert auf Typescript, ist open-source und wurde von Teams, die bei Google arbeiten, entwickelt.[1][2] Für die Verwendung von Angular wird ein Node Package Manager benötigt. Dieser kann durch den Download und Installation von Node.js benutzt werden. Mithilfe des Node Package Managers ist es möglich die Angular Command Line Interface (CLI) herunterzuladen. Mit der Angular CLI kann man Angular Projekte erstellen. Mit dem Keyword 'ng' führt man ein Command von der Angular CLI aus. Um ein neues Projekt anzulegen ist es notwendig folgenden Command auszuführen:

```
ng new NameDesProjekts
```

2.2.1 Components

Components sind unter anderem die Basis von Angular. Ein Component stellt ein Bestandteil einer Web Applikation dar. Im Prinzip erstellt man sich seine eigenen HTML-Tags und lässt diese nach belieben auf der Webseite anzeigen. Dies könnte zum Beispiel eine Menüleiste, Fußleiste, Kärtchen oder ein beliebiges UI Element sein. Sinn dahinter ist zu einem Reuseable Code. Components die erstellt wurden, kann man innerhalb vom Projekt mehrmals verwenden und muss sie nicht noch einmal erstellen, wodurch man Zeit spart. Zm Anderen behält der Code eine gewisse Struktur und ist dadurch übersichtlicher, denn das Aufrufen einer Componente nur das HTML-Tag im Code angezeigt wird. Erstellt man ein Component, werden vier verschiedene Files generiert. Zwei Typescript, ein HTML und ein CSS File. Im ersten Typescript File namens "name.component.ts"befindet sich die Logik von diesem Component. Klickt man auf einen Button zum Beispiel, so wird diese Methode im "name.component.ts"File implementiert. Das zweite Typescript File namens "name.component.specs.ts"widmet sich den Tests eines Components. In diesem File wird für jede Methode im "name.component.ts"File, etwaige Testfälle abgedeckt, sodass garaniert werden kann, dass sämtliche Funktionalitäten und Methoden ordnungsgemäß funktionieren und keine Bugs aufweisen. Im HTML File befindet sich die Struktur und Aufbau der Component und im CSS File die dazugehörigen Styles.

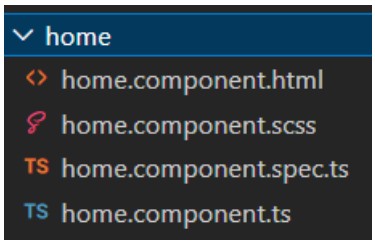

Abbildung 2.1: Component Struktur [Quelle: Autor]

Um ein Component zu erstellen verwendet man in Angular folgenden Command:

```
ng generate component NameDerComponente
```

Dieser Command kann auch abgekürzt werden, welcher dann folgendermaßen aussieht:

```
ng g c NameDerComponente
```

In der Praxis ist es gängig, dass man alle Components in ein Unterverzeichnis namens 'components' speichert, so dass die gesamte Projektstruktur eine gewisse Ordnung behält. Nachdem ein Component erstellt wurde, kann diese in einem HTML File mithilfe des sogenannten Component Selector angezeigt werden. In der Abbilding 2.2 wurde ein Filter Component erstellt, dessen Selector als 'app-filter' bezeichnet wird. Um dieses Component aufzurufen deklariert man den Selector der Component im jeweiligen HTML File mit '<app-filter></app-filter>'.

Abbildung 2.2: Component Selector [Quelle: Autor]

2.2.2 Services

Services oder auch als Serviceklassen bezeichnet, spielen eine bestimmte Rolle in Angular. Zu einem können sie als Schnittstelle zwischen Angular und dem Backend verwendet werden, zum anderen um die Kommunikation zwischen Components zu ermöglichen, wobei dieses Konzept später weiter erläutert wird.
Components müssen häufig bestimmte Daten von einer Datenbank zum Beispiel anzeigen. Für dies muss zuerst ein HTTP Request an ein Backend geschickt werden, welches schlussendlich auf die Datenbank zugreift und die bestimmten Daten an das Frontend schickt. Services dienen hierbei als Schnittstelle zwischen Frontend und Backend. Sie kümmern sich in diesem Fall um das Abschicken und Empfangen von Requests und Responses. Dadurch ist eine klare Arbeitsaufteilung innerhalb der Projektstruktur von Angular möglich. Components kümmern sich um das Darstellen von Daten und Services, um diese Daten zum Beispiel von einem Backend Server abzurufen.
Für das Erstellen von Services kann man die Angular cli verwenden. Ähnlich wie bei der Component Generierung, verwendet man für Services folgenden Command:

```
ng generate services NameDesServices
```

Hier ist es möglich den Command in gekürzter Form anzugeben:

```
ng g s NameDesServices
```

In der Praxis ist es üblich, Services in einem Unterverzeichnis namens 'services' zu speichern, damit eine klare Trennung zwischen Components und Services besteht. Für die Verwendung von Services kann ein Component dieses in sein Konstruktor innjecten. Dieses Konzept nennt man Dependency Injection. Dabei deklariert man im Konstruktur des Components den Service, der benutzt werden soll. Wenn eine Service als Injectable angegeben ist, kann diese in einem Component injeziert werden.

2.2.3 Routing

In einer Single Page Aplikation ändern man, was die Nutzende sehen, indem Teile der Webseite ein- oder ausgeblendet werden, die bestimmten Components entsprechen. In diesem Fall wird nicht der Server aufgerufen, um eine neue Seite zu erhalten. Wenn Nutzende Anwendungsaufgaben ausführen, müssen sie zwischen den verschiedenen Ansichten wechseln, die definiert wurden. Für die Navigation von einer Ansicht zur nächsten, verwendet man den Angular Router. Der Router ermöglicht die Navigation, indem er eine Browser-URL als Anweisung zum Ändern der Ansicht interpretiert.
Für die Verwendung des Angular Routers, muss dieser im 'app.module.ts' File importiert werden.[3]

```
import { RouterModule, Routes } from '@angular/router';
```

Abbildung 2.3: Importing Angular Router [Quelle: Autor]

Definition der Routen
Damit zwischen einzelnen Routen gewechselt werden kann, ist es notwendig zunächst die Routen zu definieren. Dabei gibt man an welches Component bei einer bestimmten Route angezeigt wird. Man weist einen URL Pfad zu einem bestimmten Component. Abbildung 2.4 stellt ein Array von verschiedenen Routen dar. Das Array besteht aus Objekten, welches einen Pfad und ein Component beinhaltet. Im Pfad wird der Name von der Route beschrieben und beim Component, welches Component angezeigt werden soll, wenn auf diesen bestimmten Pfad navigiert. Navigiert man zum Beispiel zur '/home' URL, wird das Home Component geladen und angezeigt.[3]

```
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'registration', component: RegistrationComponent },
  { path: 'login', component: LoginComponent },
  { path: '**', redirectTo: 'home', pathMatch: 'full' }
]
```

Abbildung 2.4: Route Definition [Quelle: Autor]

Angular Router Konfiguration
Nachdem die einzelnen Routen erstellt wurden, muss der Router mit diesen konfiguriert werden. Dies geschieht mithilfe der 'RouterModule.forRoot()' Methode. Dieser Methode übergibt man das vorhin erstellte Array, die die Routen beinhaltet. Somit weiß der Angular Router über die einzelnen Routen bescheid und kann nach einer Navigation zu einem bestimmten Pfad, das entsprechende Component laden.[3]

```
RouterModule.forRoot(routes),
```

Abbildung 2.5: Angular Router Konfiguration [Quelle: Autor]

Route mit Parameterübergabe
Navigiert man zu einem bestimmten Pfad, kann ein Parameter mitgegeben werden. Dieser muss im vorhinein bei der Routen Definition angegeben. Der Parameter dient hierbei als Platzhalter. Zum Beispiel kann eine ID mitgegeben werden. Beim Laden des jeweiligen Components kann vom Pfad diese ID ausgelesen werden, um anschließend bestimmte Daten im Component anzeigen zu lassen, die dieser ID entsprechen. Oft ist hier der Fall, dass die ID vom Pfad ausgelesen wird und anschließend ein HTTP Request an ein Backend Server gesendet wird, um zu einer bestimmten Datei zusätzliche Informationen anzeigen zu lassen. In der Abbildung 2.6 ist eine Routen Definition mit einem Platzhalter namens 'ID' zu sehen, der das oben Beschriebene umsetzt.[3]

```
{ path: 'searchResults/:id', component: SearchResultsComponent }
```

Abbildung 2.6: Route Definition mit Parameter [Quelle: Autor]

Für das Zugreifen auf mitgegebene Parameter bei der Navigation, wird in Angular das ActivatedRouter Service injected. Mithilfe des Services kann man auf etwaige Properties von der Route Zugriff erhalten. Folgende Properties können bei der Navigation mitgegeben und anschließend abgerufen werden:

- **url:** Ein Observable der Routenpfade, dargestellt als ein Array von Zeichenfolgen für jeden Teil des Routenpfads.
- **data:** Ein Observable, das das für die Route bereitgestellte Datenobjekt enthält. Enthält auch alle aufgelösten Werte aus dem Resolve Guard.
- **params:** Ein Observable, das die für die Route spezifischen erforderlichen und optionalen Parameter enthält.
- **paramMap:** Ein Observable, das eine Karte der für die Route spezifischen erforderlichen und optionalen Parameter enthält. Die Zuordnung unterstützt das Abrufen einzelner und mehrerer Werte aus demselben Parameter.
- **queryParamMap:** Ein Observable, das eine Karte der für alle Routen verfügbaren Abfrageparameter enthält. Die Zuordnung unterstützt das Abrufen einzelner und mehrerer Werte aus dem Abfrageparameter.
- **queryParams:** Ein Observable, das die für alle Routen verfügbaren Abfrageparameter enthält.
- **fragment:** Ein Observable des URL-Fragments, das für alle Routen verfügbar ist.
- **outlet:** Der Name des RouterOutlet, das zum Rendern der Route verwendet wird. Bei einem unbenannten Ausgang ist der Name des Ausgangs primär.
- **routeConfig:** Die Routenkonfiguration, die für die Route verwendet wird, die den Ursprungspfad enthält.
- **parent:** Die übergeordnete ActivatedRoute der Route, wenn diese Route eine untergeordnete Route ist.
- **firstChild:** Enthält die erste ActivatedRoute in der Liste der untergeordneten Routen dieser Route.
- **children:** Enthält alle unter der aktuellen Route aktivierten untergeordneten Routen.

[3]

Default Route

Die letzte Route Definition in der Abbildung 2.4 stellt einen Default Pfad dar. Navigiert man zu einem Pfad der nicht existiert bzw. vom Router nicht gefunden wird, so lädt der Router das Component welches unter dem Default Pfad angegeben ist. Wenn ein Nutzende die Webseite lädt, wird zu diesem Pfad weitergeleitet. Bevor der Router zum Einsatz kommen kann, muss das sogenannte RouterOutlet Directive im HTML File angegeben werden. Typischerweise passiert dies im HTML File vom App Component. Hierbei definiert man das RouterOutlet directive als HTML Tag '<router-outlet></router-outlet>'. Dieses directive dient als Platzhalter, für die definierten Routen im Angular Router. Dort wo das RouterOutlet directive platziert wird, werden die Components angezeigt, wenn zu deren Pfade navigiert wird.[3]

Router Links

Für die Navigation an sich bietet Angular eine bestimmte Funktionalität namens Router Links an. Abbildung 2.7 1. Zeile, sieht man eine einfache Routing Navigation. Dem Anchor Tag wird ein Attribut namens 'routerLink' hinzugefügt. Diesem Attribut übergibt man einen Pfad, zu dieser der Angular Router navigiert, wenn auf den Anchor Tag geklickt wird.

```
<a class="nav-link active text-light" aria-current="page" routerLink="accountinfo" i18n>Accountinfo</a>
<a [routerLink]="['searchResults']"[queryParams]="{product: 'notebook'}">Suchen</a>
```

Abbildung 2.7: Router navigation [Quelle: Autor]

Müssen zusätzliche Properties oder Query Parameter dem Pfad beim Navigieren mitgegeben werden, kann dies mit '[routerLink]' durchgeführt werden. Durch die eckigen Klammern, wird dem Angular Router Bescheid gegeben, dass neben dem Pfad auch zusätzliche Informationen mitgegeben werden können. Hierbei könnte es sich um eine definierte Variable in einem Component File handeln, oder dynamische Werte. In Abbildung 2.7 2. Zeile, würde dieser Pfad folgende Route entsprechen, '/searchResults?product=notebook'. Mit [queryParams] teilt man den Angular Router mit, dass er neben dem Pfad zusätzlich den Query Parameter namens 'product', mit dem Wert 'notebook' übergeben soll.[3]

Activated Route

Befinden sich die Nutzende auf einer bestimmten Seite, kann das RouterLinkActive Directive benutzt werden, um CSS Klassen aktiv zu machen. Einem Anchor Tag kann man CSS Styles hinzufügen, so dass die Nutzende erkennen, auf welcher Seite sie sich gerade befinden und dies auch zum Beispiel im Menü ersichtlich markiert wird. Damit ein Anchor Tag, nachdem zu einer bestimmten Seite navigiert wurde, CSS Styles erhält, fügt man dem Anchor Tag das RouterLinkActive Directive hinzu. Als Wert übergibt man eine oder mehrere CSS Klassen. In Abbildung 2.8 wird für den Anchor Tag die CSS Klasse 'activeSite' hinzugefügt, wenn die Anwendende zur Startseite navigieren.

```
<a routerLinkActive="activeSite" routerLink="home" i18n>Startseite</a>
```

Abbildung 2.8: Router navigation [Quelle: Autor]

Zusätzlich zu 'routerLinkActive' kann man auch '[routerLinkActive]' verwenden, um den Wert vom Directive auf ein Property des Components zu setzen.[3]

Route Protection

Gewisse Routen sind für die Anwendende nicht zugänglich. Sei es eine Dashboard Seite nachdem man sich angemeldet hat oder eine Admin Seite. Bestimmte Routen sollen nicht für die Nutzende zugänglich sein. Daher ist es wichtig diese Routen zu schützen und nur Zugang zu gewährleisten, wenn sich Nutzende erfolgreich authentifizieren. Für die Implementierung kommt das CanActivate Interface zum Einsatz. Dieses Interface wird in einer Guard Klasse verwendet. Durch das hinzufügen des CanActivate Interfaces, wird eine Methode namens canActivate in den Guard implementiert. Diese Methode bestimmt durch das Returnen von true oder false, ob eine Route zugänglich ist oder nicht. Dazu wird bei der Route Definition ein weiteres Property namens canActivate benutzt. Als Wert für das Property kann ein oder mehrere Guards verwendet werden. Returnen alle Guards true, können die Nutzende zu der geschützten Route navigieren. Wenn ein Guard false returned, ist die Route nicht zugänglich.

```typescript
@Injectable()
export class AuthGuard implements CanActivate {

    constructor(private authService: AuthService, private router: Router) {}

    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean |
    UrlTree | Observable<boolean | UrlTree> | Promise<boolean | UrlTree> {
        const isAuth = this.authService.getIsAuth();
        if(!isAuth) {
            this.router.navigate(['/']);
        }
        return isAuth;
    }

}
```

Abbildung 2.9: CanActivate Methode [Quelle: Autor]

In Abbildung 2.9 wird ein Beispiel für die CanActivate Methode demonstriert. In diesem Beispiel lautet die Bedingung für das Zugreifen auf Routen, die mit diesem Guard gesichert sind, das die Anwendende angemeldet sein müssen. Ist dies der Fall, wird true returned, im anderen Fall wird auf die Default Route navigiert. Dieser Guard muss in der Route Definition im app.module.ts File eingefügt werden, indem der Wert des entsprechenden Propertys mit diesem gesetzt wird.

```typescript
{ path: 'accountinfo', component: AccountinfoComponent, canActivate: [AuthGuard] },
```

Abbildung 2.10: Guard in der Route Definition [Quelle: Autor]

Child Routes

Child Routes oder auch Nested Routes genannt, sind Routen die nicht vom Root Component abgeleitet sind, sondern von einem anderen Component. Wenn die Web Applikation komplexer wird, macht es Sinn auf Child Routes zurückzugreifen. Denn sämtiche Seiten sind erst aufrufbar, wenn die Nutzende auf eine bestimmte Seite navigiert haben, von dieser eine andere Seite abgeleitet wird. Ein Beispiel dafür wäre eine Dashboard Seite. Von der Dashboard Seite kann man auf eine Account Information Seite navigieren. Würde man in der URL nur den Pfad von der Account Information Seite angeben, so wäre diese nicht aufrufbar, da man sich zuvor auf der Dashboard Seite befindet muss. Dadurch entsteht eine Baumstruktur von Routen, wobei Child Routen als Leaf Nodes agieren oder als Parent Node. Für die Implementierung geht man genau so vor, wie bei der Routendefinition. Zusätzlich kann man nun zu einem Eintrag im Array der Routendefinition, neben Path und Component, ein Children Array angeben. Dieses Children Array beinhaltet die Childrenrouten von einer Parent Route. Diese Childrenrouten stellen jene Routen dar, die von der darüber geordneten Route abhängig ist. Für die Childrenrouten benötigt man auch einen Platzhalter, nämlich das 'routerOutlet' HTML-Tag. Dieses Tag wird in der Seite der Parent Route platziert, von dieser die Childrenrouten abgeleitet werden. Dort wo sich das 'routerOutlet' HTML-Tag befindet, werden die Seiten der Childrenrouten geladen und angezeigt. Für die Defintion der Childrenrouten stellt Abbildung 2.11 ein Beispiel dar. Hier existiert für eine Produkt Details Seite zwei Childrenrouten. Die Erste dient als Übersichtsseite für ein bestimmtes Produkt, wobei die zweite Childroute eine detaillierte Ansicht eines Produkts darstellt. Auch wurde eine Default spezifiziert, die zur Übersichtsseite weiterleitet, falls zu einer nicht vorhandenden Childroute

navigiert wird.

```
{ path: 'product-details/:id', component: ProductDetails,
  children: [
    { path: '', redirectTo: 'overview', pathMatch: 'full' },
    { path: 'overview', component: Overview },
    { path: 'specs', component: Specs }
  ]
}
```

Abbildung 2.11: Angeben von Childrouten [Quelle: Autor]

2.2.4 Component Kommunikation

Angular ist ein Component-based Framework, daher ist es wichtig zu verstehen, wie eine Kommunikation zwischen einzelnen Components möglich ist. Da Components oft Daten von anderen Compoments bekommen, um diese zu verarbeiten bzw. anzuzeigen, bietet Angular verschiedene Ansätze an, um ein Datenaustausch zu ermöglichen. Im nachfolgenden werden Kommunikationsmöglichen für Components beschrieben und praktisch angewendet.

Parent-to-child Kommunikation
...
Child-to-parent Kommunikation
...
Kommunikation über Serviceklassen
Components werden nicht immer im Parent-Child Prinzip implementiert. Oft kommt es vor, das Components miteinander Daten austauschen ohne das diese in einer Art und Weise in Verbindung stehen. Für solche Fälle verwendet man zu einem Serviceklassen und zum anderen RxJS. Angular macht Gebrauch von Reactive Extensions Library for JavaScript (RxJS). Mithilfe dieser Library können Components miteinander interagieren, ohne das diese direkt in Verbindung stehen. Falls die Components nicht in einer Parent-Child Umgebung dargestellt werden, ist RxJS eine Alternative für den Datenaustausch. "RxJS ist eine Bibliothek zum Erstellen asynchroner und ereignisbasierter Programme unter Verwendung von beobachtbaren Sequenzen. Es bietet einen Kerntyp, das Observable, Satellitentypen (Observer, Schedulers, Subjects) und Operatoren, die von Array-Methoden (map, filter, Reduce, every usw.) inspiriert sind, um die Behandlung asynchroner Ereignisse als Sammlungen zu ermöglichen".[4]. Die wichtigsten Konzepte von RxJS, für das Ausführen von asynchronen Event, sind folgende:

- **Observable:** stellt die Idee einer aufrufbaren Sammlung zukünftiger Werte oder Ereignisse dar.
- **Observer:** ist eine Sammlung von Rückrufen, die weiß, wie man auf die vom Observable gelieferten Werte hört.
- **Subscription:** Stellt die Ausführung eines Observable dar, ist in erster Linie nützlich, um die Ausführung zu stornieren.
- **Operators:** sind reine Funktionen, die einen funktionalen Programmierstil für den Umgang mit Sammlungen mit Operationen wie map, filter, concat, Reduce usw. ermöglichen.

- **Subject:** entspricht einem EventEmitter und ist die einzige Möglichkeit, einen Wert oder ein Ereignis per Multicasting an mehrere Observer zu senden.
- **Scheduler:** sind zentralisierte Dispatcher zur Steuerung der Parallelität, die es uns ermöglichen, zu koordinieren, wann die Berechnung auf z. setTimeout oder requestAnimationFrame oder andere.

[4]
Die Konzepte von RxJS können mit Serviceklassen von Angular verbunden werden, um den Austausch von Daten zu implementieren. Wie bereits erwähnt dienen Serviceklassen zum Einem als Schnittstelle zwischen Frontend und Backend, zum Anderem als Kommunikationsschnittstelle zwischen Components. Die Serviceklasse kann dafür verwendet werden, um zu subscriben und um bestimmte Werte auszusenden. Dies macht man mithilfe von Observables bzw. mit Subjects. In der Serviceklasse wird ein Subject definiert, welches als Observable in einer Methode zurückgeliefert wird. Diese Methode können Components benutzen um darauf zu subscriben. Durch das Subscriben haben diese Components nun einen Subscription zu diesem Subject. Anschließend implementiert man eine weitere Methode die benutzt wird, um Werte an die Subscriber auszusenden. Das Subject wird benutzt, um bestimmte Daten als ein Multicast, an andere Component zu senden, die das Subject zuvor subscribed haben. Bildlich kann man sich dieses Szenario folgendermaßen vorstellen. Es existiert eine Serviceklasse und zwei Components namens A und B. Component A benutzt die Serviceklasse, um Daten ausszusenden. Component B verwendet die Serviceklasse, um auf ein Observable zu subscriben. Wenn Component A nun Daten aussenden möchte, benutzt diese die Serviceklasse, damit alle anderen Components die Werte bekommen, die sie benötigen.

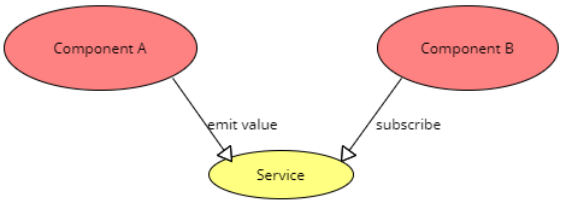

Abbildung 2.12: Component Kommunikation mit Services [Quelle: Autor]

2.3 Next.js

2.3.1 Routing

Wie funktioniert Routing
Definition der Routen
Navigation zwischen unterschiedliche Routen
Route mit Parameterübergabe
Route Protection

2.3.2 Component Kommunikation

Parent-to-child Kommunikation
Properties

Child-to-parent Kommunikation
Function properties

Centralized State

2.4 Unterschiede

2.4.1 Routing Unterscheidungen

2.4.2 Component-interaction Unterscheidungen

3 Schluss

3.1 Zusammenfassung

Wichtigsten Erkenntnisse noch einmal zusammenfassen

3.2 Zukunfsausblicke

Mögliche Themen die in zukünftigen Paper ausgearbeitet werden können

4 Related Work

Literaturverzeichnis

[1] E. Saks, “JavaScript frameworks: Angular vs React vs Vue,” 2019, online erhältlich unter https://www.theseus.fi/bitstream/handle/10024/261970/Thesis-Elar-Saks.pdf; abgerufen am 6. Dezember 2022. 2

[2] C. M. N. G. B. M. O. T. G. Ovidiu Constantin Novac, Damaris Emilia Madar, “Comparative study of some applications made in the Angular and Vue.js frameworks,” 2021, online erhältlich unter https://ieeexplore-1ieee-1org-1tn53mdm6397c.han.fh-campuswien.ac.at/document/9484150; abgerufen am 6. Dezember 2022. 2

[3] Angular, “Angular Router Official Documentation,” Website, 2022, online erhältlich unter https://angular.io/guide/router-reference; abgerufen am 6. Dezember 2022. 4, 5, 6, 7

[4] RxJS, “RxJS Introduction,” 2022, online erhältlich unter https://rxjs.dev/guide/overview; abgerufen am 14. Dezember 2022. 9, 10

Abbildungsverzeichnis

Tabellenverzeichnis

Appendix

(Hier können Schaltpläne, Programme usw. eingefügt werden.)