ARS

Motivation

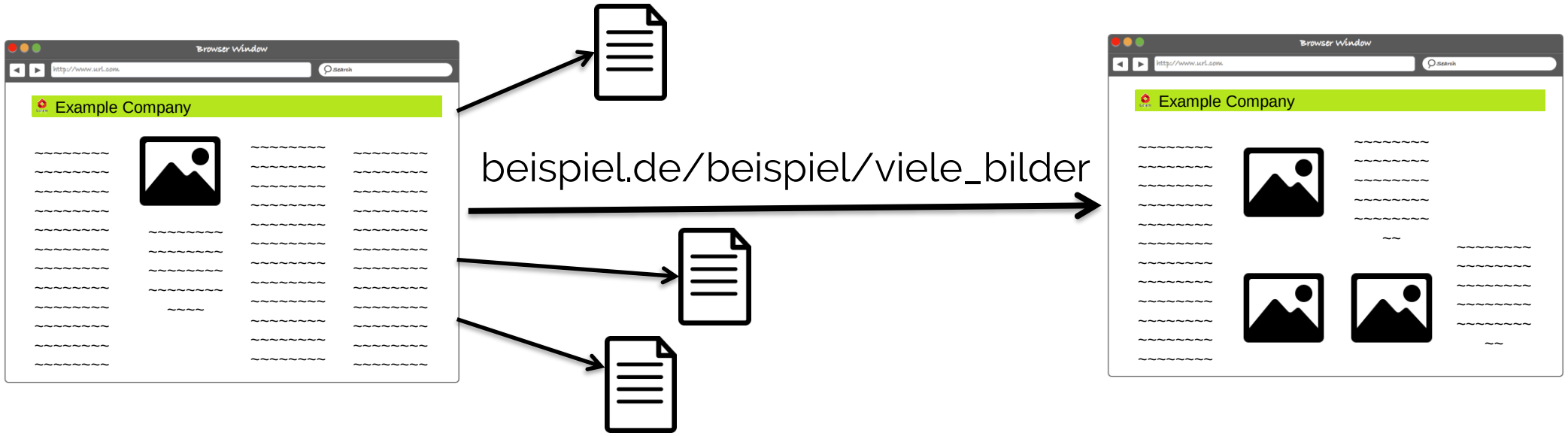
- Abläufe auf mehrere Seiten aufteilen
- Zusammengehörige Abschnitte explizit miteinander verknüpfen
- Navigation zwischen unterschiedlichen Bereichen einer SPA ermöglichen
- Links zu bestimmten Seiten in der Applikation teilen

ARS

Lernziele

- Sie beschreiben den Aufbau einer URL
- Sie erklären den Unterschied zwischen Client- und Server-seitiger Navigation
- Sie begründen, warum Navigation in einer SPA wichtig ist
- Sie beschreiben den Unterschied zwischen Hash-Strategy und Path-Strategy

Navigation im Browser



- Typische Webseite = Sammlung verlinkter Dokumente
- Navigation zwischen Dokumenten mit Uniform Resource Locator (URL) über Links
- Vereinfacht: beispiel.de/beispiel/viele-bilder/das-bild.html
  - Host
  - Pfad zu Ressource
  - Ressource
- Aufruf einer URL → Server erzeugt und liefert eine neue Seite

ARS

Navigation im Browser

- Weiterer Nutzen für Navigation über URLs
  - Erstellung von Lesezeichen
  - Weitergabe von Navigationspunkten an andere
  - Zurück-Navigation zu früher besuchten Seiten
- **Problem:** Navigation bei SPAs
  - Server liefert nur eine Seite aus → Theoretisch nur eine URL
  - Dynamische Erstellung „neuer“ Seiten aus Templates im Browser
- **Lösung:** @angular/router
  - Bildet das Browser-Navigationskonzept für SPA nach
  - SPA leitet weiter zu dynamisch erzeugten Seiten
    - Über Links
    - Eingabe von URLs im Browser
    - Back-Button

ARS

Angular Routing (HTML 5 History API)

- Nutzt den Pfad einer URL zur Navigation
  http://localhost:5555/todo/1
- Default-Verhalten von @angular/router
- Pfad einer URL kann programmatisch geändert werden
- URLs bei SPA schauen wie „echte“ URLs aus, ohne Anfragen an Server zu senden
- Dynamische Änderung der Browser-History bis zur HTML5-Spezifikation nicht möglich
  (https://i/www.w3.org/TR/html5/browsers.html#the-history-interface)

Internet Explorer < 10 ??

ARS

SPA Routing vor History API

- Nutzt den Hashteil einer URL: *http://localhost:5555/#/todo/1*
- Ändert die URL ohne Seite neu zu laden

Angular und ältere Browser

- HashLocationStrategy kann z. Zt. konfiguriert werden, wird aber langfristig nicht mehr unterstützt
- Nutzung eines Polyfills (Shims) für History API

ARS

Grundstruktur

Das Router-Modul ermöglicht das Navigieren zwischen Views, übernimmt das Updaten der Browserhistorie und verhindert ein vollständiges neu laden der Webseite.

Zuständigkeiten der verschiedenen Angular Elemente

index.html

- Präfix für relative Bilder-, CSS- und Skript-URLs festlegen

app.module.ts
app-routing.module.ts

- Import des Router-Modules
- Konfiguration des Routers:
  - Definition der Routen
  - Definition der URL-Strategie

Template (z.B.: app.component.html)

- Festlegen eines zentralen View-Einstiegspunktes (Platzhalter) der definierten Routen

ARS

Ergänzungen index.html

Setzen eines Präfixes für die Verarbeitung von relativen URL-Pfaden (Bilder-, CSS- und Skript-Dateien)

index.html

```
<base href="/"> // Sofern APP-Ordner Root ist, verwende “/“
```

Vorgehen, wenn Zugriff auf index.html nicht möglich:

- Konfiguration des Routers mittels APP_BASE_HREF
- Verwendung von absoluten Pfaden für alle Ressourcen (Bilder, CSS, Skripte und Templates)

ARS

Einfluss base href

Definiert, ab wann der Pfad für den URL relevant wird

```
<base href="/">
```

http://example.com/todo/1

```
<base href="/server-todo/">
```

http://example.com/server-todo/todo/1

ARS

Unterstützte Routendefinitionen

Komponente wird beim Aufruf der exakten Route geladen

```
{path: 'home', component: HomeComponent}
```

Route mit URL-Parametern

```
{path: 'entity/:id', component: EntityDetailComponent}
```

Route mit (statischen) benutzerdefinierten Daten

```
{path: 'dashboard', component: DashboardComponent, data: {title:
'Dashboard' }}
```

Definition von Weiterleitungen

```
{path: '', redirectTo: '/home', pathMatch: 'full' }
```

Definition von Weiterleitungen

```
{path: '**', component: PageNotFoundComponent }
```

Reihenfolge entscheidend

ARS

Routen definieren – app.module.ts

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

const appRoutes: Routes = [
  {path: 'home', component: HomeComponent},
  {path: 'entity/:id', component: EntityDetailComponent},
  {path: '', redirectTo: '/home', pathMatch: 'full'},
  {path: '**', component: PageNotFoundComponent}
];

@NgModule({
  imports: [..., RouterModule.forRoot(appRoutes)],
  ...
})
export class AppModule {}
```

Unschön: Mischen von Routing und Bootstrap Aufgaben

ARS

Auslagerung von Routing in eigenes Modul

- „Separation of concerns“
- Modul kann für Testzwecke einfach entfernt oder ersetzt werden
- Architekturentscheidung, kein Muss

ARS

Routing Module

app-routing.module.ts

```
const appRoutes: Routes = [
      ...
];

@NgModule({
  imports: [..., RouterModule.forRoot(appRoutes)],
  exports: [RouterModule] // Re-Export des RouterModule
})
export class AppRoutingModule {}
```

```
import { AppRoutingModule } from './app-routing.module';

@NgModule({
  imports: [..., AppRoutingModule],
  ...
})
export class AppModule {}
```

app.module.ts

ARS

HTML für Navigation

app.component.html

```
<nav>
 <a routerLink="/home" routerLinkActive="active">Home</a>
 <a routerLink="/search" routerLinkActive="active">Search</a>
</nav>

<router-outlet></router-outlet>
```

- RouterOutlet: Platzhalter für Views der definierten Routen
- RouterLink: Verlinkung zu (Routing-)Views
- RouterLinkActive: Zuweisung einer CSS-Klasse im aktiven Zustand der Route

ARS

Bestandteile des Router Modules

- Router Outlet
  - Angular Komponente
  - Markiert, wo Templates dynamisch erzeugt werden
- Router Link
  - Angular Direktive
  - Router Link bindet Navigations-URL an HTML-Elemente
  - Navigationspfad kann statisch oder dynamisch sein
    - Statisch → one time binding (string)
    - Dynamisch → über Property-Binding (Code im Template)

ARS

Bestandteile des Router Modules

Beispiel: Anwendung unter http://example.com

```
<nav>
  <!- http://example.com/dashboard -->
  <a routerLink="/dashboard">Dashboard</a>
  <!- http://example.com/hero/1 -->
  <a [routerLink]="['/hero', firstHero.id]">First Hero</a>
</nav>
<router-outlet></router-outlet>
```

ARS

Programmatische Navigation

- Einbindung des Router Services im Konstruktor
- Aufruf von *navigate* mit Parameter
  - Pfad (Path) definiert im Router
  - Optional: Dynamische Bestandteile einer URL (hier id)

ARS

Möglichkeit der Navigation

```
constructor(private router: Router) {}

onSelect(entity: Entity) {
   this.router.navigate(['/entity', entity.id]);
}
```

```
<ul class="entities">
   <li *ngFor="let entity of entities" (click)="onSelect(entity)">
     <span class="badge">{{entity.id}}</span> {{entity.name}}
   </li>
</ul>
```

ARS

Klassen für den Zugriff auf Routendaten

```
import { Router, ActivatedRoute, ParamMap } from '@angular/router';

constructor(private route: ActivatedRoute,
            private router: Router) {}
```

Router	Bereitstellung der Navigations- und URL-Manipulations-Funktionaliät
ActivatedRoute	Ermöglicht Zugriffe auf Daten der aktuell aktiven Route: Z.B. URL, Parameter, Komponente, Fragmentdaten, Benutzerdefinierte Daten, Kindelemente, etc.
ParamMap	Map mit Parametern; parameter: string -> any

ARS

Parameter auslesen - Snapshot

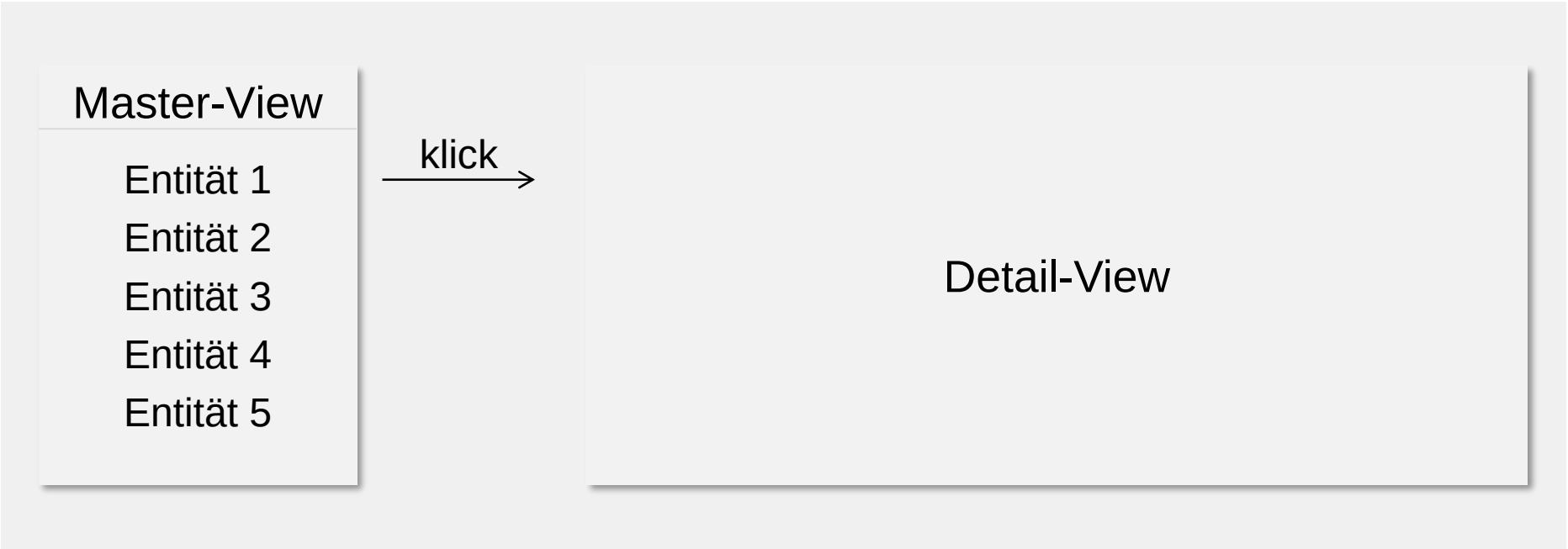
```
ngOnInit() {
  // (+) Wandelt den String 'id' in numerischen Typ (number) um
  const id = +this.route.snapshot.paramMap.get('id');

  this.service.getEntity(id)
    .then((entity: Entity) => this.entity = entity);
}
```

- Einfache Variante initiale URL-Parameter auszulesen
- Künftige Änderungen der URL-Parameter werden nicht erkannt
- Nutzung nur wenn Komponente nicht direkt wiederverwendet wird, z.B. Navigation von einer Detailansicht direkt zu einer anderen (mit anderen Worten: Die Instanz der Komponente wird nicht wiederverwendet)

ARS

Snapshot Nachteil



Obige Umsetzung mit Snapshots nicht möglich:

- NgOnInit wird nur beim ersten initialen Aufruf ausgeführt
- Änderungen bei weiteren Klicks werden nicht erkannt
- Lösung: Umsetzung mit Observables

ARS

Parameter auslesen - Wertänderungen

```
ngOnInit() {
  // Auf Änderungen des URL-Parameter reagieren
  this.route.paramMap
    .subscribe(paramMap => {
      this.service.getEntity(paramMap.get('id'))
        .then((entity: Entity) => this.entity = entity);
  });
}
```

- Künftige Änderungen der URL-Parameter werden erkannt
- Komponente kann wiederverwendet werden

ARS

Statische Daten

app.module.ts

```
{path: 'home', component: HomeComponent, data: {title: 'Dashboard'}}
```

dashboard.component.ts

```
constructor (private activatedRoute: ActivatedRoute) {
  this.title = activatedRoute.snapshot.data['title'];
}
```

dashboard.component.html

```
<h1>{{title}}</h1>
```

- Nutzung des „data“-Attributes für Routen-spezifische Daten: Seitentitel, Breadcrumb-Texte, etc.
- Nutzung ausschließlich für statische Daten

ARS

Zusammenfassung

Um was ging es in diesem Modul?

- Grundlagen über Navigation im Browser
- Aufbau einer URL
- Ansätze für Navigation in einer SPA (Router-Module in Angular)

Wozu brauche ich das? Was will ich damit machen?

- Aufteilung komplexer Abläufe auf mehrere Seiten
- Navigation zwischen unterschiedlichen Bereichen einer SPA
- Explizite Verknüpfung zusammengehöriger Abschnitte

ARS

Kontrollfragen

- Wie kann in HTML über Links navigiert werden?
- Wie ist ein URL aufgebaut?
- Welche Besonderheiten sind bzgl. der Navigation in SPAs zu berücksichtigen?
- Was ist der Unterschied zwischen Hash-Strategy und Path-Strategy?
- Was ist die Bedeutung des Routing-Modules?
- Wie wird das Routing-Module konfiguriert?
- Ist es möglich auf programmatische Weise Routen aufzurufen?