2 Moderne Webentwicklung mit Angular

- Clientseitige Web-Programmierung wird bei der Verwendung von Ajax, DOM-Manipulation via jQuery und dem Verzicht auf weitere Hilfsmittel schnell zu komplex. Verhindern sollen das Frameworks wie Angular, React und Co. Deren Verwendung und im Hintergrund arbeitende Mechanismen werden in diesem Kapitel behandelt.

- Inhalt
  1. Angular
  2. Vergleich mit anderen Frameworks
  3. Dependency Management
  4. Bundler
  5. CSS-Präprozessoren
  6. Minifier/Uglyfier
  7. Backend-Client-Entwicklung

2.1 “Moderne“ Webentwicklung mit Angular

- Wiederkehrende Aufgaben vereinfachen
  - Lesen von Formularen
  - Kommunikation mit Server
  - Daten und HTML synchron halten
  - Komponentenstruktur mit definierten Schnittstellen

Geprüft durch einen Compiler, bitte!

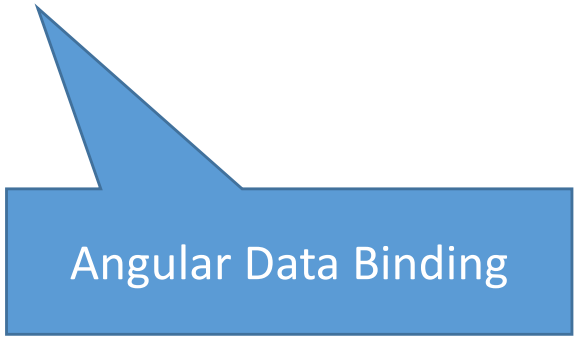
Codeausschnitt einer klassischen Anwendung mit jQuery

```
$("#button").on("click", function () {
   $.ajax({
      type: "GET",
      url: "/data/" + $("#data-input").val(),
      success: function(content) {
         $("#server-response").text(content);
      },
      ...
   });
});
```

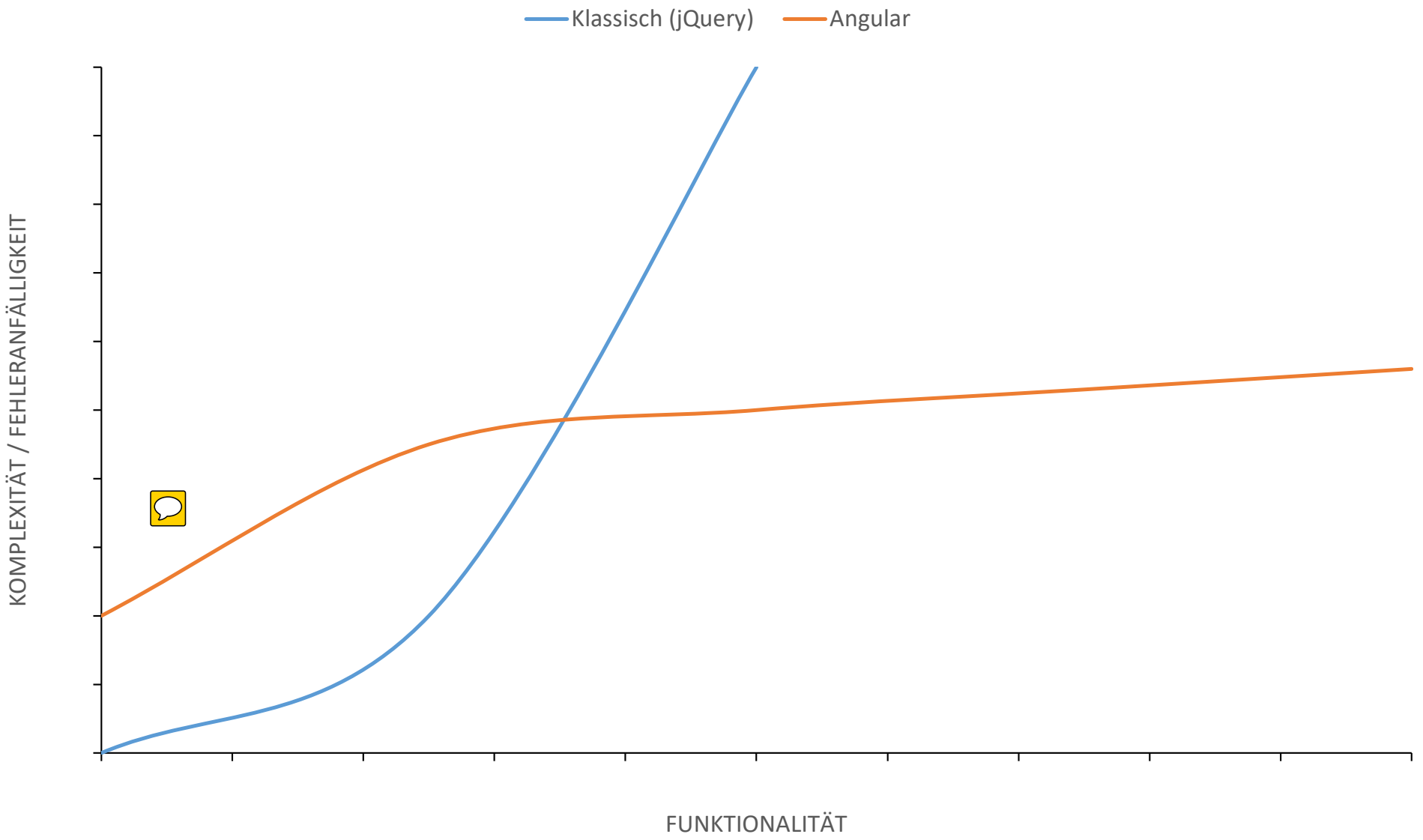
```
let userUL : JQuery = $('#userUL');
userUL.html("");

for (let id in userList) {
  let user : User = userList[id];
  if (user != null) {
    let li = document.createElement('li');
    li.id = String(user.id);
    li.addEventListener('click', handleRead);
    let div = document.createElement('div');
    let divUsername = document.createElement('div');
    divUsername.classList.add("w3-tiny");
    divUsername.textContent = user.username;
    let spanVorname = document.createElement('span');
    spanVorname.classList.add("vornameText");
    spanVorname.textContent = user.vorname;
    let spanDelete = document.createElement('span');
    spanDelete.classList.add('fa', 'fa-trash', "w3-large", "w3-margin-right");
    spanDelete.id = String(user.id);
    spanDelete.addEventListener('click', handleDelete);
    let divDateTime = document.createElement('div');
    divDateTime.classList.add("w3-tiny");
    if (user.time) {
      divDateTime.innerText = user.time;
    }

    li.appendChild(divUsername);
    div.appendChild(spanVorname);
    div.appendChild(spanDelete);
    li.appendChild(div);
    li.appendChild(divDateTime);
    userUL.append(li);

  }
}
```

HTML Definition in TypeScript → unschön

```
<ul>
  <li (click)="myClick()"
    *ngFor="let user of this.userList; let i = index">
    <div class="w3-tiny">
      {{user.username}}
    </div>
    <span class="vornameText">
      {{user.vorname}}
    </span>
    <span (click)="delete(i)" class="fa fa-trash"></span>
    <div *ngIf="user.time" class="w3-tiny">
      {{user.time}}
    </div>
  </li>
</ul>
```



“Moderne“ Webentwicklung



2.1.1 Angular CLI – Projekt anlegen

```
cd Desktop
```

Verzeichnis wechseln

```
npm install -g @angular/cli
```

Angular-CLI installieren

```
ng new todo
```

Angular-App erzeugen

```
cd todo
```

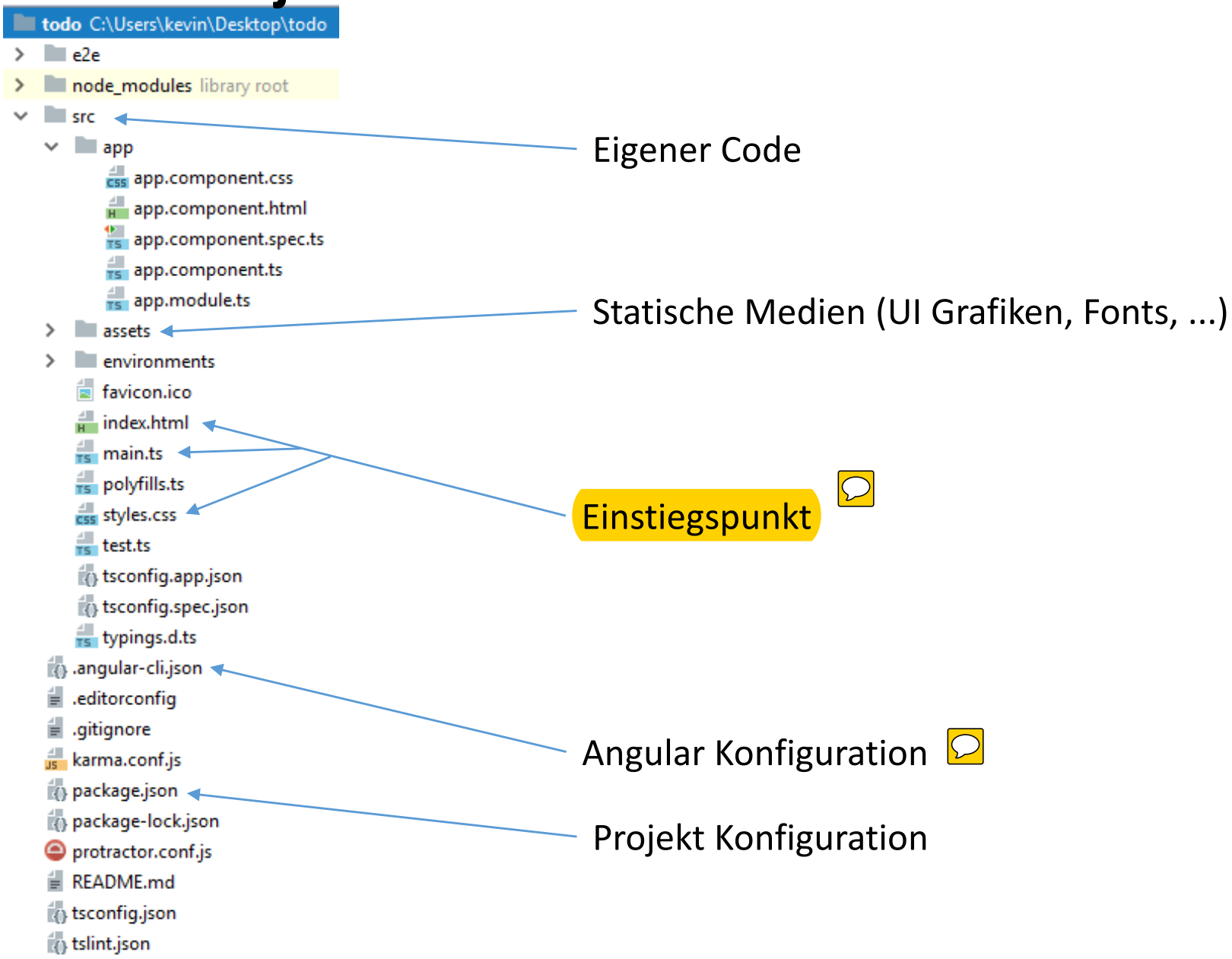
Verzeichnis wechseln

```
ng serve
```

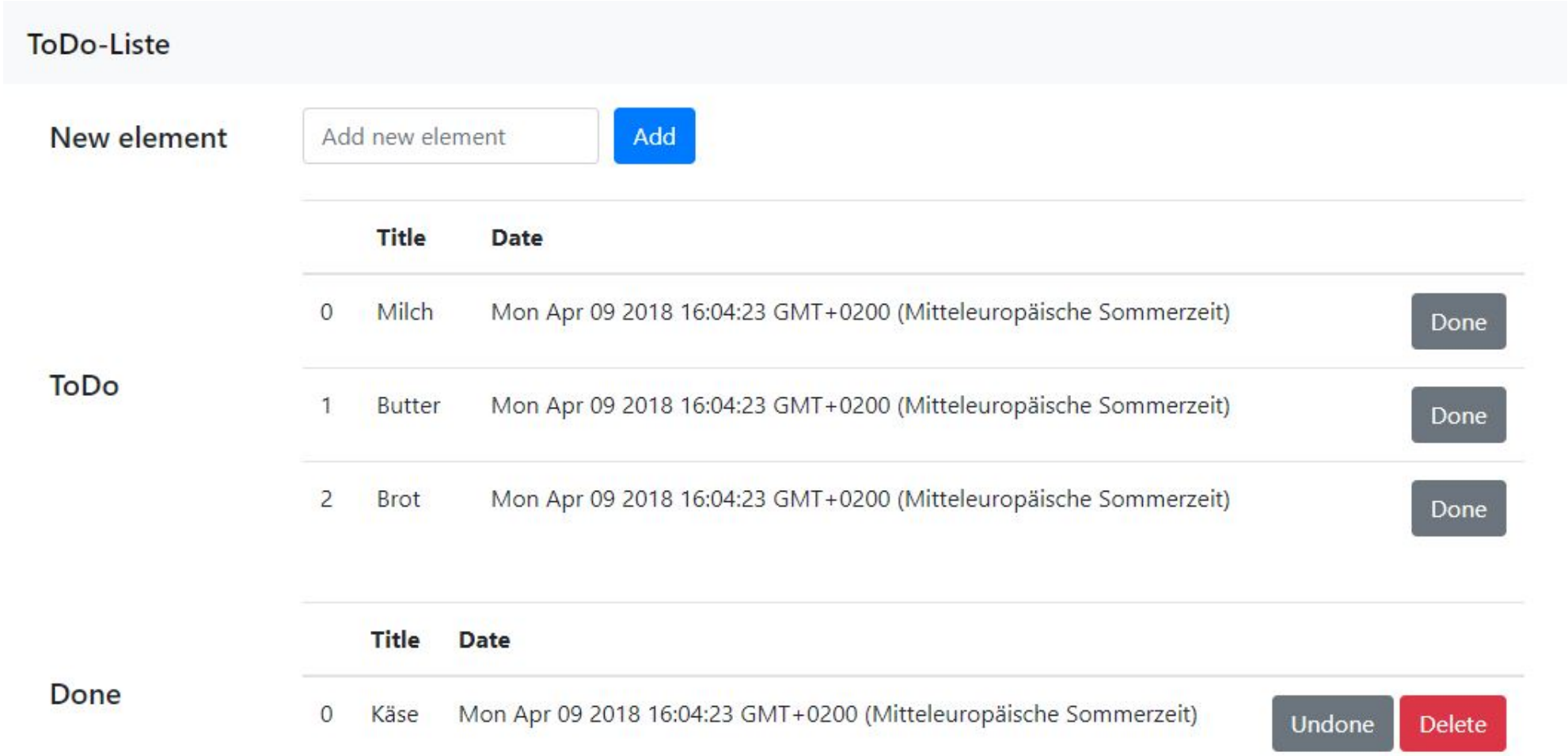
Anwendung ausführen

http://localhost:4200

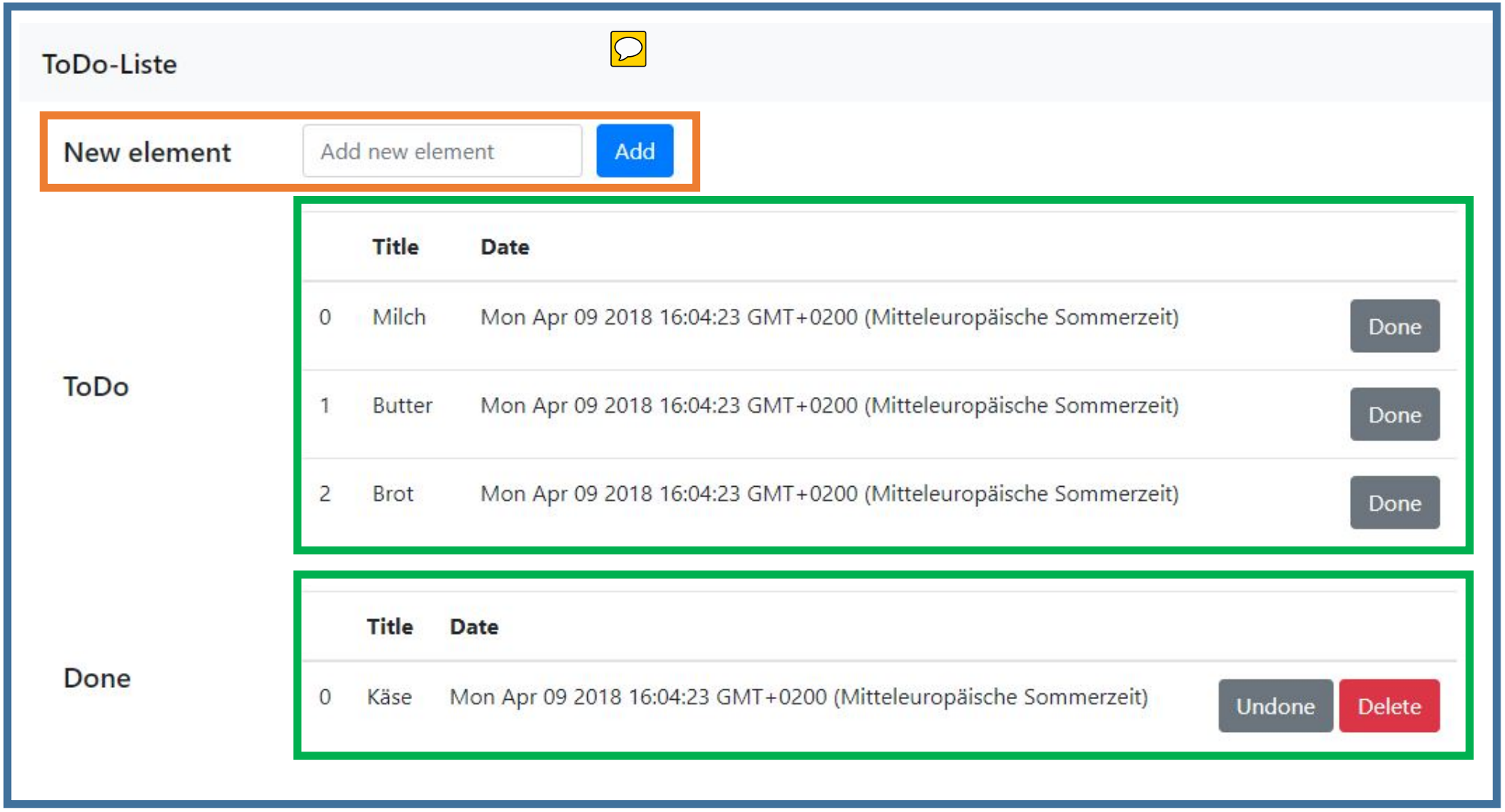
2.1.1 Projektverzeichnis



2.1.1 To-Do-Liste



2.1.1 To-Do-Liste



2.1.1 Komponente

app.component.ts

```
import {Component} from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  title = 'ToDo-Liste';

  constructor() {}
}
```

app.component.html

```
<nav class="navbar navbar-dark bg-dark mb-5">
  <span class="navbar-brand mb-0 h1">{{title}}</span>
</nav>
```

`@Component(...)`

- Metainformationen zur Komponente
- Selector: Name des HTML Tags
- TemplateURL: Zugehörige HTML Datei
- StyleURLs: Zugehörige CSS Datei

index.html

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

2.1.1 NgModule

- Angular Apps sind modular → NgModule fasst z.B. Workflows zusammen
- Fasst Komponenten, Services, Direktiven und Pipes zusammen
- Jede Angular Anwendung besteht aus einem Startmodul
- Zusätzliche Module können als Framework eingebunden werden
  - Bootstrap
  - Material
  - Firebase

app.module.ts

```
@NgModule({
    declarations: [
        AppComponent,
        ListComponent
    ],
    imports: [
        BrowserModule,
        FormsModule
    ],
    providers: [],
    bootstrap: [AppComponent]
})
export class AppModule { }
```

2.1.1 Angular Bootstrap

```
npm install --save @ng-bootstrap/ng-bootstrap
```

```
npm install --save font-awesome
```

Definieren, dass die Bootstrap CSS eingebunden werden soll.

Einbinden des Bootstrap Moduls für TypeScript Funktionalität (bsp. Modalfenster)

.angular-cli.json

```
...
"main": "main.ts",
 "polyfills": "polyfills.ts",
 "styles": [
   "style.css",
   "../node_modules/bootstrap/dist/css/bootstrap.css",
   "../node_modules/font-awesome/css/font-awesome.css"
 ],
 ...
```

app.module.ts

```
import {NgbModule} from '@ng-bootstrap/ng-bootstrap';
import {AppComponent} from './app.component';
import {FormsModule} from '@angular/forms';
import {BrowserModule} from '@angular/platform-browser';

@NgModule({
    declarations: [AppComponent, ...],
    imports: [NgbModule.forRoot(), FormsModule, ...],
    bootstrap: [AppComponent]
})
export class AppModule {
}
```

Übung

- Bringen Sie eine leere Angular Anwendung zum Laufen.
- Binden Sie Bootstrap (ng-bootstrap) ein.
- Ändern Sie die Hauptkomponente, sodass diese eine Titelleiste anzeigt.
  - Orientieren Sie sich an den bisherigen Folien.
  - https://getbootstrap.com/docs/4.0/components/navbar/

2.1.2 Listen-Komponente

```
ng generate component list
```

list.component.ts

```
import { Component } from '@angular/core';
import { ToDoEntry } from '../ToDoEntry';

@Component({
  selector: 'app-list',
  templateUrl: './list.component.html',
  styleUrls: ['./list.component.css']
})
export class ListComponent {

  public todoList: ToDoEntry[]  = [
    new ToDoEntry('Milch'),
    new ToDoEntry('Butter'),
    new ToDoEntry('Brot')
  ];

  constructor() {}

  done(index: number): void {
    this.todoList.splice(index, 1);
  }

}
```

list.component.html

```
<div class="container">
  <h5>ToDo</h5>

  <table class="table">
    <thead>
    <tr>
      <th scope="col"></th>
      <th scope="col">Title</th>
      <th scope="col">Date</th>
      <th scope="col"></th>
    </tr>
    </thead>
    <tbody>
    <tr *ngFor="let item of todoList; let i = index">
      <td>{{i}}</td>
      <td>{{item.title}}</td>
      <td>{{item.date}}</td>
      <td class="text-right">
        <button type="button" class="btn btn-
            secondary" (click)="done(i)">Done</button>
      </td>
    </tr>
    </tbody>
  </table>
</div>
```

app.component.html

```
...
<app-list></app-list>
```

Selector

2.1.2 Ausflug: TypeScript Module

list.component.ts

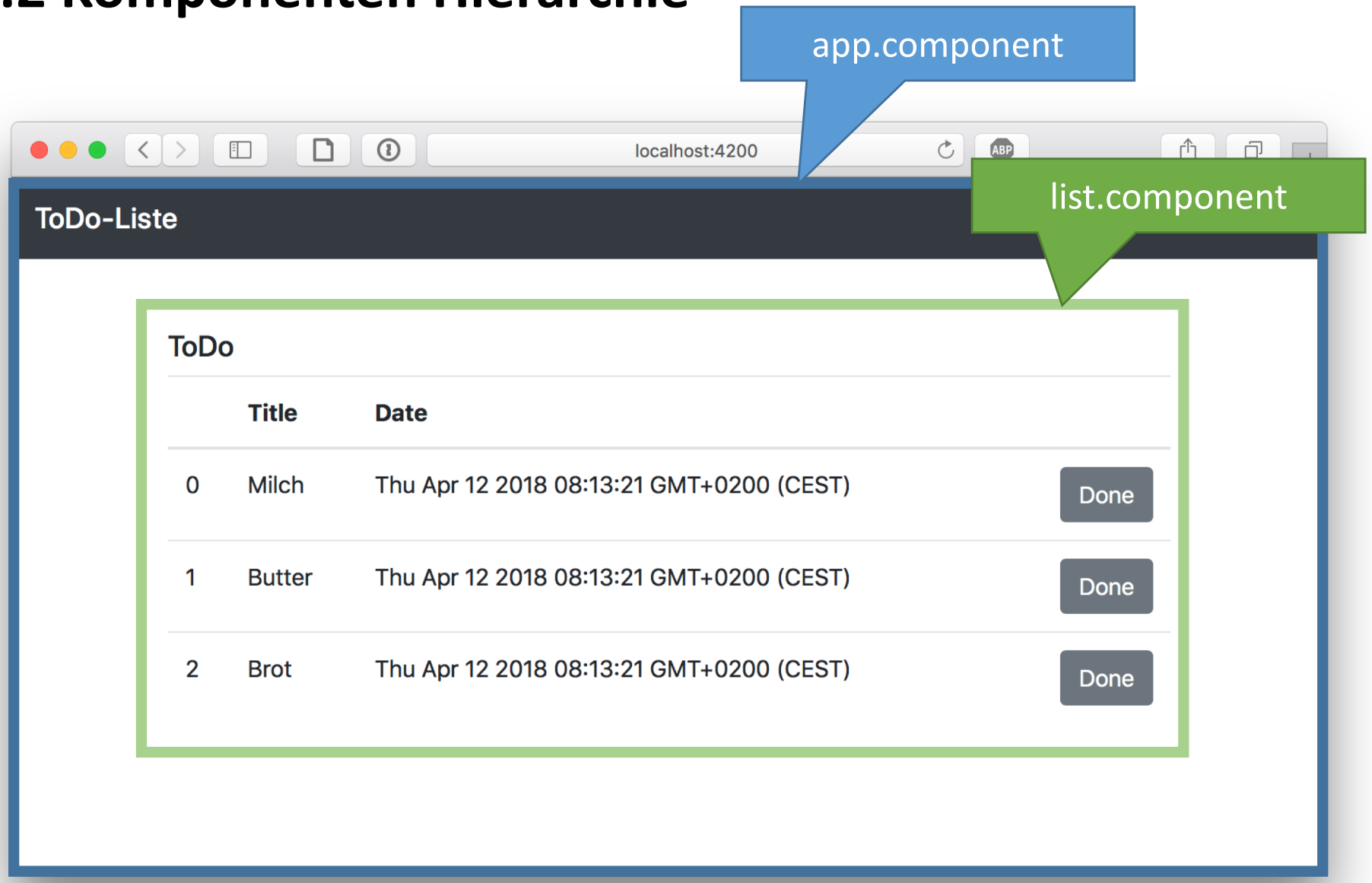
```
import { Component } from '@angular/core';
import { ToDoEntry } from '../ToDoEntry';

@Component({
  selector: 'app-list',
  templateUrl: './list.component.html',
  styleUrls: ['./list.component.css']
})
export class ListComponent {

  public todoList: ToDoEntry[]  = [
    new ToDoEntry('Milch'),
    new ToDoEntry('Butter'),
    new ToDoEntry('Brot')
  ];

  constructor() {}

  done(index: number): void {
    this.todoList.splice(index, 1);
  }

}
```

ToDoEntry.ts

```
export class ToDoEntry {
  title: string;
  done: boolean;
  date: Date;

  constructor(title: string) {
    this.title = title;
    this.date = new Date();
    this.done = false;
  }
}
```

- `export class`
  - Macht die Klasse sichtbar nach außen und damit importierbar
  - „Sichtbar“ auf TypeScript Ebene

2.1.2 Komponenten Hierarchie



2.1.2 Direktiven („erweitern HTML“)

- *ngFor
  - „For-Schleife für HTML“
- *ngIf
  - “If-Verzweigung für HTML“
- Binding Markup
  - „Automatische Synchronisation von HTML Inhalten und TypeScript Werten“

2.1.2 *ngFor, *ngIf

Ziel: Tabelle anhand von Daten

list.component.ts

```
...
public items: ToDoEntry[];
...
```

list.component.html

```
<tr *ngFor="let item of items; let i =
index">
  <td>{{i}}</td>
  <td>{{item.title}}</td>
  <td>{{item.date}}</td>
  <td class="text-right">
    <button
      *ngIf="!item.done"
      (click)="done(i)">
      Done
    </button>
  </td>
</tr>
```

Laufzeitsicht

```
<tr>
  <td>0</td>
  <td>Milch</td>
  <td>Mon Apr 09 2018 19:21:22</td>
  <td class="text-right"> </td>
</tr>
<tr>
  <td>1</td>
  <td>Butter</td>
  <td>Mon Apr 09 2018 19:21:22</td>
  <td class="text-right">
    <button type="button"
      class="btn" (click)="done(1)">
      Done
    </button>
  </td>
</tr>
...
```

=> Ersetzt eigenhändiges $(„#container“).append(„<tr>“); usw...

0	Milch	Wed Apr 18 2018 18:27:25 GMT+0200 (Mitteleuropäische Sommerzeit)	
1	Butter	Wed Apr 18 2018 18:27:25 GMT+0200 (Mitteleuropäische Sommerzeit)	Done

2.1.2 Binding Markup

- Interpolation
  - „spätere Einfügung in einen abgeschlossenen Text“
  - One-way text binding

```
<td>{{item.title}}</td>
<img src="{{item.url}}" style="height:30px">
```

- Property Binding
  - One-way data binding
  - Objects, Arrays, ...

```
<button [disabled]="isUnchanged">Save</button>
<app-list [items]="getData()"></app-list>
```

- Event Binding
  - Auf Benutzereingabe reagieren

```
<button type="button" class="btn btn-secondary"
(click)="done(10)">Done</button>
```

- Model Binding
  - Two-way data binding

```
<input [(ngModel)]="textFieldContents" type="text">
```

2.1.2 Binding: Interpolation

HTML

```
<p>{{item.title}}</p>
<img src="{{item.url}}" style="height:30px">
```

TypeScript (Komponente)

```
export class TestComponent {

  public item = {
    title: 'Hallo, Welt',
    url: 'https://www.thm.de/_thm/logos/thm.svg'
  };

  constructor() { }
}
```

Ergebnis

TypeScript → HTML

2.1.2 Property Binding

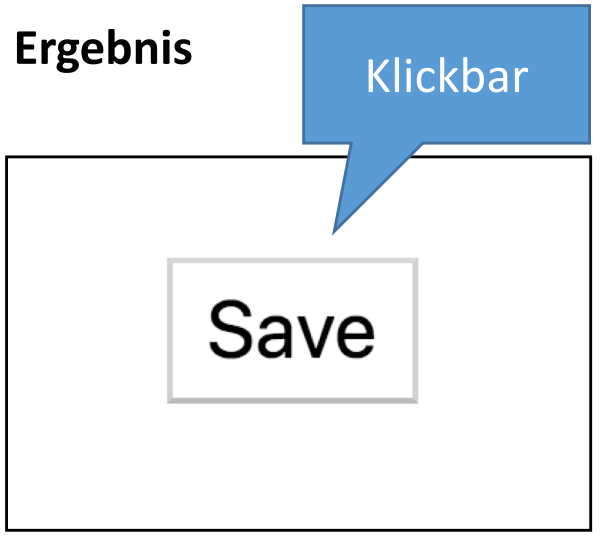
HTML

```
<button [disabled]="isUnchanged">Save</button>
```

TypeScript (Komponente)

```
export class TestComponent {

  public isUnchanged = false;

  constructor() { }
}
```

Ergebnis



TypeScript → HTML

Unterschied zur Interpolation: Property Binding erlaubt das Übergeben komplexer Werte wie Objekte oder Arrays.

2.1.2 Event Binding

HTML

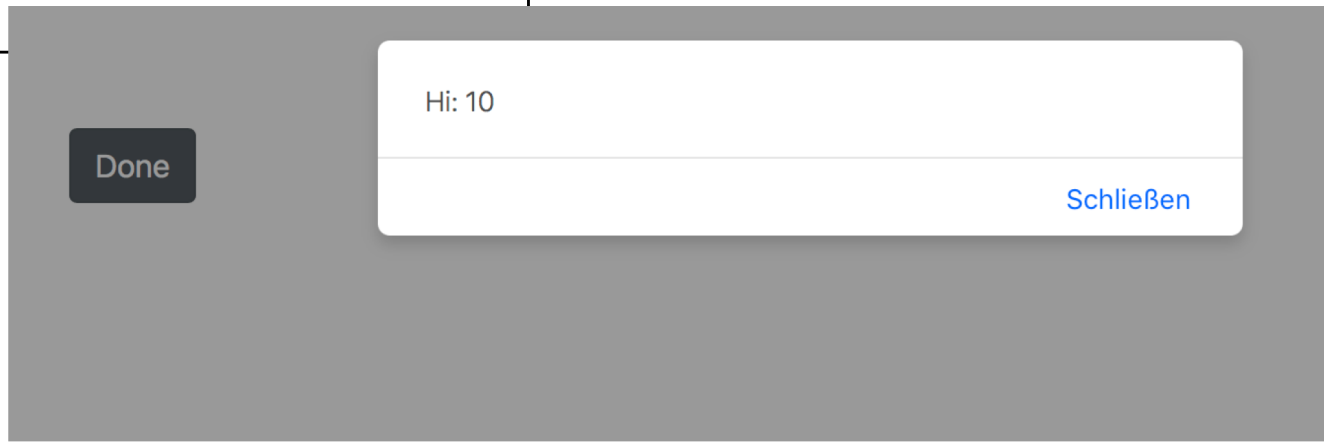
```
<button type="button" class="btn btn-secondary"
(click)="done(10)">Done</button>
```

TypeScript (Komponente)

```
export class TestComponent {

  public done(index: number): void {
    alert('Hi: ' + index);
  }

  constructor() { }
}
```

TypeScript ← HTML

Ergebnis



2.1.2 Model Binding

HTML

```
<input [(ngModel)]="textFieldContents" type="text">
<p>Der Text ist: {{textFieldContents}}</p>
```

TypeScript (Komponente)

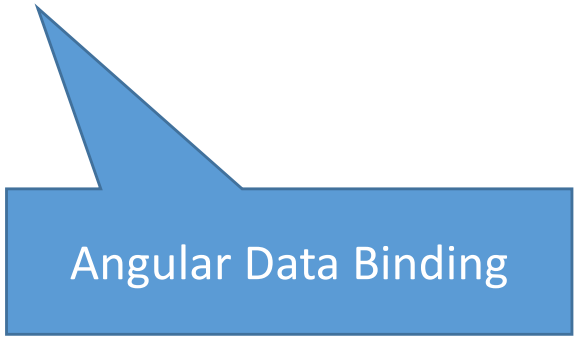
```
export class TestComponent {

  public textFieldContents: string = "Wert";

  constructor() { }
}
```

Ergebnis

TypeScript ←→ HTML

```
let userUL : JQuery = $('#userUL');
userUL.html("");

for (let id in userList) {
  let user : User = userList[id];
  if (user != null) {
    let li = document.createElement('li');
    li.id = String(user.id);
    li.addEventListener('click', handleRead);
    let div = document.createElement('div');
    let divUsername = document.createElement('div');
    divUsername.classList.add("w3-tiny");
    divUsername.textContent = user.username;
    let spanVorname = document.createElement('span');
    spanVorname.classList.add("vornameText");
    spanVorname.textContent = user.vorname;
    let spanDelete = document.createElement('span');
    spanDelete.classList.add('fa', 'fa-trash', "w3-large", "w3-margin-right");
    spanDelete.id = String(user.id);
    spanDelete.addEventListener('click', handleDelete);
    let divDateTime = document.createElement('div');
    divDateTime.classList.add("w3-tiny");
    if (user.time) {
      divDateTime.innerText = user.time;
    }

    li.appendChild(divUsername);
    div.appendChild(spanVorname);
    div.appendChild(spanDelete);
    li.appendChild(div);
    li.appendChild(divDateTime);
    userUL.append(li);

  }
}
```

```
<ul>
  <li (click)="myClick()"
    *ngFor="let user of this.userList; let i = index">
    <div class="w3-tiny">
      {{user.username}}
    </div>
    <span class="vornameText">
      {{user.vorname}}
    </span>
    <span (click)="delete(i)" class="fa fa-trash"></span>
    <div *ngIf="user.time" class="w3-tiny">
      {{user.time}}
    </div>
  </li>
</ul>
```



2.1.2 Konfiguration

src/tsconfig.app.json

Strenge Templatekontrolle aktivieren

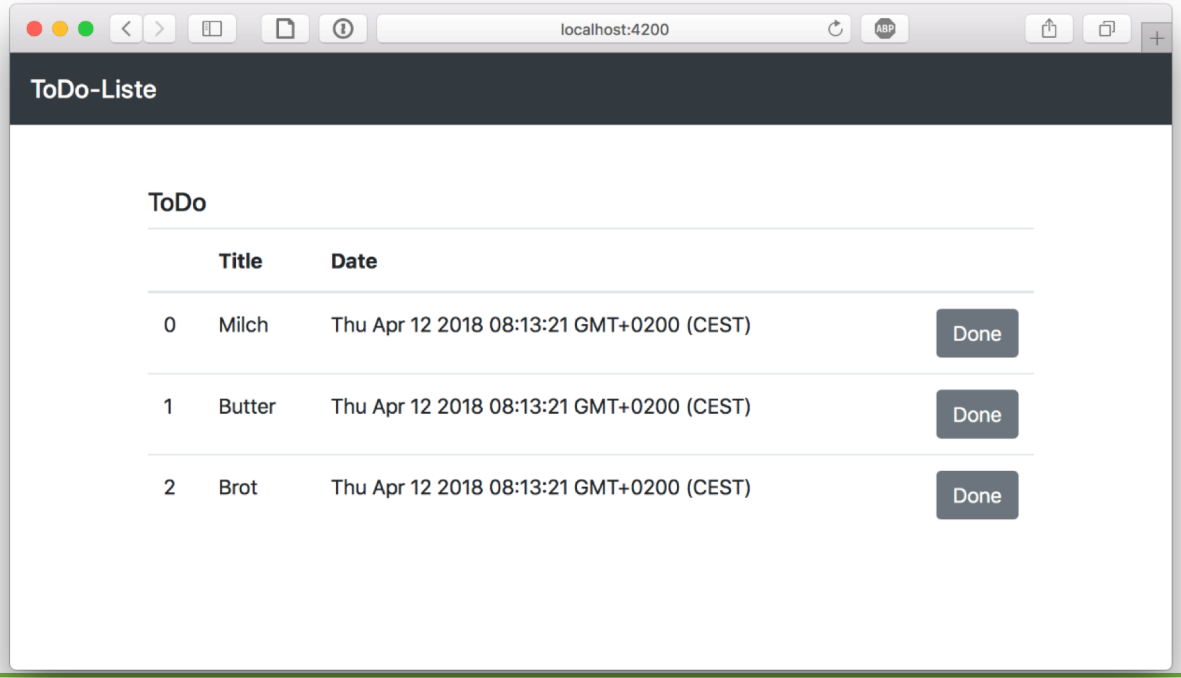
```
{
  "extends": "../tsconfig.json",
  "compilerOptions": {
    "outDir": "../out-tsc/app",
    "baseUrl": "./",
    "module": "es2015",
    "types": []
  },
  "exclude": [
    "test.ts",
    "**/*.spec.ts"
  ],
  "angularCompilerOptions": {
    "fullTemplateTypeCheck": true
  }
}
```

package.json

```
{
  "name": "todo",
  "version": "0.0.0",
  "license": "MIT",
  "scripts": {
    "ng": "ng",
    "start": "ng serve --aot",
    "build": "ng build --prod --aot",
    "test": "ng test",
    "lint": "ng lint"
  },
```

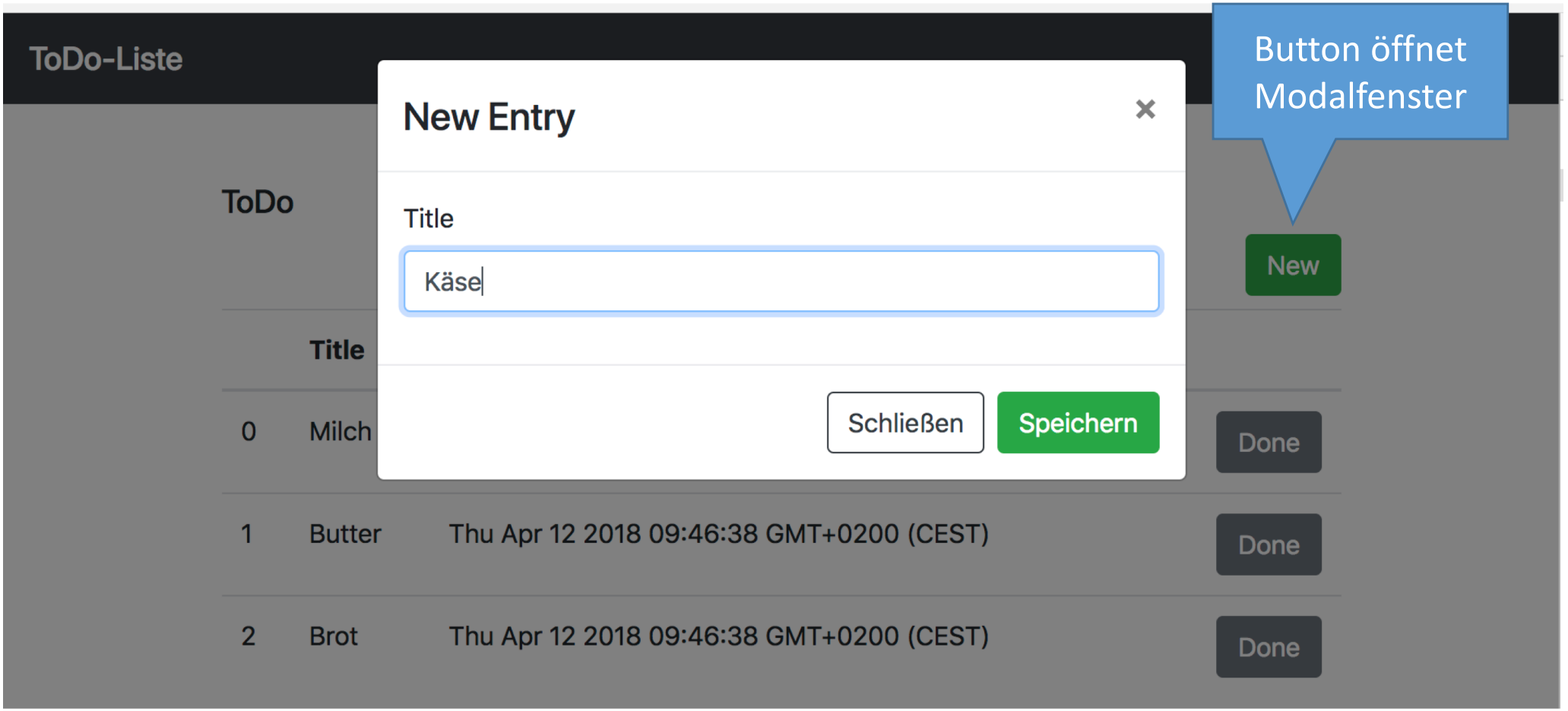
Übung

- Implementieren Sie die ToDo Liste, wie Sie in den Folien gezeigt ist.
  - Die Hauptkomponente (app.component) soll die Listenkomponente (list.component) enthalten.
    - Die Hauptkomponente ist für die Titelleiste zuständig („ToDo-Liste“).
    - Die Listenkomponente zeigt die Liste.
  - Drei Einträge sollen vorprogrammiert ein, welche durch „Done“ gelöscht werden.
  - Anlegen neuer Einträge ist vorerst nicht nötig.



2.1.3 Bootstrap + Angular

- Modalfenster
- Ziel:



2.1.3 Modalfenster

list.component.ts

```
import {NgbModal} from '@ng-bootstrap/ng-bootstrap';
import {AddentryComponent} from './addentry/addentry.component';

export class ListComponent {
  ...

  constructor(private modalService: NgbModal) {}

  addToDoButtonClicked() {
    const modalReference = this.modalService.open(AddTodoComponent);
    modalReference.result
      .then((result: any) => {
        this.todoList.push(new ToDoEntry(result as string));
      })
      .catch((error) => {
        console.log('Window closed: ' + error);
      });
  }
  ...
}
```

Dependency Injection

Neue Komponente

Promise

Cancel oder X führen zu „weichem Fehler“

Es muss kein leerer Container erstellt werden. (= Outlet)

2.1.3 Modalfenster

add-todo.component.ts

```
import {NgbActiveModal} from '@ng-bootstrap/ng-bootstrap';

@Component({
  selector: 'app-add-todo',
  templateUrl: './add-todo.component.html',
  styleUrls: ['./add-todo.component.css']
})
export class AddTodoComponent {

  public title: string;

  constructor(public activeModal: NgbActiveModal) {
    this.title = '';
  }

  save() {
    if (this.title.trim().length > 0) {
      this.activeModal.close(this.title);
    }
  }
}
```

ngModel des TextField

Context des Modal Systems

Setzt das result → triggert .then

2.1.3 Modalfenster

add-todo.component.html

```
<div class="modal-header">
  <h4 class="modal-title">New Entry</h4>
  <button type="button" class="close" aria-label="Close"
(click)="activeModal.dismiss()">
    <span aria-hidden="true">&times;</span>
  </button>
</div>
<div class="modal-body">
  <div class="form-group">
    <label for="itemInput">Title</label>
    <input type="text" class="form-control" id="itemInput" [(ngModel)]="title">
  </div>

</div>
<div class="modal-footer">
  <button type="button" class="btn btn-outline-dark"
(click)="activeModal.dismiss()">Schließen</button>
  <button type="button" class="btn btn-success"
(click)="save()">Speichern</button>
</div>
```

Triggert .catch

Wird im Verlauf zu ‚result'

Triggert .catch

Triggert letztendlich .then

2.1.3 Modalfenster

app.module.ts

```
import {BrowserModule} from '@angular/platform-browser';
import {NgModule} from '@angular/core';

import {AppComponent} from './app.component';
import {FormsModule} from '@angular/forms';
import {AddTodoComponent} from './add-todo/add-todo.component';

export let moduleConfig = {
  declarations: [
    AppComponent,
    AddTodoComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent] ,
  entryComponents: [
    AddTodoComponent
  ]
};

@NgModule(moduleConfig)
export class AppModule {
}
```

2.1.3 Dokumentation

- https://angular.io/docs
- https://ng-bootstrap.github.io/#/components/
- https://getbootstrap.com/docs/4.0/

2.1 Kompetenz: Angular – Teil 1

- Sie verstehen die Vorteile von Web-Frontend-Frameworks
- Sie können eine Angular-Anwendung einrichten und starten
- Sie können Bootstrap einbinden und verwenden
  - Sie kennen die Besonderheiten von Bootstrap in Angular
  - Sie können ein Bootstrap-Modalfenster in Ihrer Anwendung verwenden
- Sie wissen, wie Angular-Komponenten verschachtelt werden können
- Sie können Direktiven und Property Binding zur Implementierung von Funktionalität verwenden
  - *ngFor, *ngIf
  - Sie kennen die Unterschiede von Interpolation, Property Binding, Event Binding, Model Binding

Ende VL 3