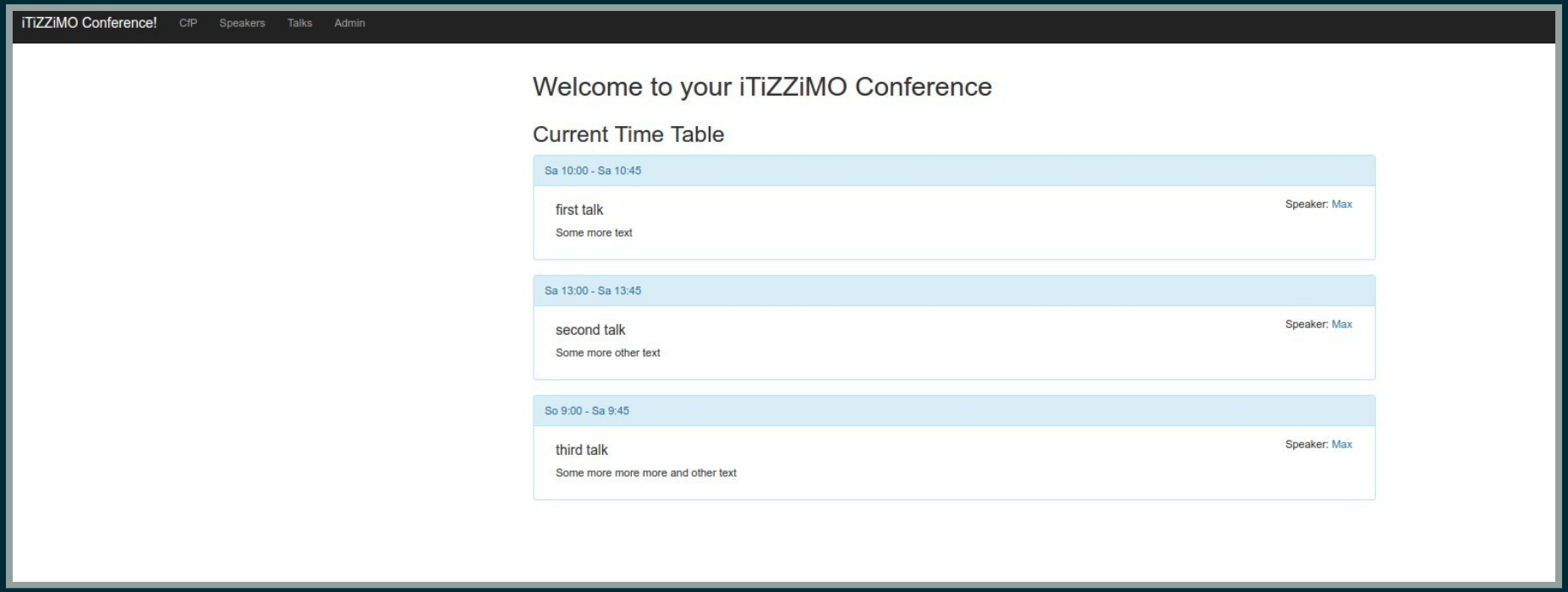
ANGULAR WORKSHOP

MAXIMILIAN BERGHOFF - 26.07.2017 - ITIZZIMO

- Maximilian Berghoff
- @ElectricMaxxx
- github.com/electricmaxxx
- Maximilian.Berghoff@mayflower.de
- Mayflower GmbH - Würzburg

AUSBLICK


iTiZZiMO Conference!
CfP
Speakers
Talks
Admin
Welcome to your iTiZZiMO Conference
Current Time Table
Sa 10:00 - Sa 10:45
first talk
Some more text
Speaker: Max
Sa 13:00 - Sa 13:45
second talk
Some more other text
Speaker: Max
So 9:00 - Sa 9:45
third talk
Some more more more and other text
Speaker: Max

HISTORY

- Angular 1.x heißt jetzt AngularJS
- Angular 2 und 4 sind Angular
- Entwickelt in Community, der auch Google angehört

ANGULARJS VS. ANGULAR

ANGULAR JS

- Controller als Standard für ein "MVC"
- Komponenten durch Direktiven möglich
- Data-Binding everywhere

ANGULAR

- Fokus auf Komponenten
- Data-Binding kann/muss in der Verantwortung des Entwicklers
- Modul Struktur
- Typescript als Basis

- faktisch keine Migration von AngularJS auf Angular möglich (außer Rewrite)
- Angular 2 lässt sich einfach auf 4 migrieren

AUSBLICK 2 - THEMEN

- Bootstraping - Installation einer Skeleton App mit dem CLI
- Basics:
- ▪ Templating, Komponenten, App-Struktur, Routing, DI
- Advanced:
- ▪ Forms - Data-Binding oder Reactive
- ▪ Events - Inter-Komponenten-Kommunikation
- ▪ Async - Kommunikation mit einem Server

LOS GEHT'S

TASK 1: BOOTSTRAPING

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

# Create new skeleton app
ng new conference-app
cd conference-app

# run it
ng serve -o
```

ALLES AUF NULL

```
git clone git@github.com:ElectricMaxxx/itizzimo-conference-app.git

git checkout T1
```

BASICS - STRUKTUR

BASIS - TEMPLATING

TEMPLATING - INTERPOLATION

```
    {{ 'ich bin ein string'}}
    <!-- String -->

    {{ ichBinEineVariable}}
    <!-- public property in component -->

    {{ 'ich bin ein string' + ichBinEineVariable}}
    <!-- Concatenation -->

    {{ gibString() }}
    <!-- Method Call -->
```

```
export class AppComponent {
  ichBinEineVariable = 'app';
  gibString(): string {
    return 'Ich bin ein String';
  }
}
```

TEMPLATING - EXPRESSIONS

```
[property]="expression"
```

TEMPLATING - EXPRESSIONS

```
    <span [hidden]="isUnchanged">changed</span>
    <!-- Sichtbarkeit von Span Element -->

    <div *ngFor="let task of tasks">{{task.name}}</div>
    <!-- Angular interne Direktive -->
```

```
export class AppComponent {
  isUnchanged: bool = true;
  tasks: Task[] = [];
}
```

TEMPLATING - STATEMENTS

```
    <button (click)="onClick()">Drück mich</button>
    <!-- Event Handler -->
```

```
export class AppComponent {
  onClick(): void {
    alert('ich wurde gedrückt');
  }
}
```

TEMPLATES - DATA BINDING

Data direction	Syntax	Type
One-Way from data source to view template	{{expression}}, [target]="expression"	Interpolation, Property, Attribute, Class, Style
One-way from view target to data source	(target)="statement"	Event
Two-Way	[(target)]="expression"	Two-Way

TASK 2 - TEMPLATING

- Führe eine neue Komponente `SpeakerDashboardComponent` ein
- Erstelle eine Liste von Speakern und gebe diese aus
- Erstelle ein Formular um einen neuen Speaker anzulegen

BASICS - TESTING

TESTING - UNIT-TESTS*

* TESTEN DER KOMPONENTEN SIND EIGENTLICH KEINE UNIT-TESTS, SOLLEN HIER ABER UNSERE KLEINSTE EINHEIT DARSTELLEN.

```
import { TestBed, async } from '@angular/core/testing';
import { AppComponent } from './app.component';
describe('AppComponent', () => {
  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  }));

  it('should create the app', async(() => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  }));
});
```

-

```
it(`should have as title 'app'`, async(() => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app.title).toEqual('app');
}));

it('should render title in a h1 tag', async(() => {
    const fixture = TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    const compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('h1').textContent)
    .toContain('Welcome to app!');
}));
```

TASK 3 - TESTING

- Erstelle eine TestSuite für die `CallForPapersComponent`
- Teste:
- ▪ Komponente hat Button zum Registrieren
- ▪ Komponente hat leere Tabelle
- ▪ Komponente zeigt das Formular im "Registrier-Modus"
- ▪ Nach dem Anlegen eines neuen Speakers gibt es genau einen Eintrag in der Tabelle

BASICS - ROUTING

ROUTING - DEFINITION

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

const appRoutes: Routes = [
  { path: 'crisis-center', component: CrisisListComponent },
  { path: 'hero/:id',      component: HeroDetailComponent },
  { path: 'heroes',component: HeroListComponent, data: {
      title: 'Heroes List'
  }},
  { path: '', redirectTo: '/heroes', pathMatch: 'full'},
  { path: '**', component: PageNotFoundComponent }
];
@NgModule({
  imports: [RouterModule.forRoot(appRoutes],
  ...
})
export class AppModule { }
```

ROUTING - IM TEMPLATE

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

ROUTING - AUFRUF

```
<nav>
    <a routerLink="/crisis-center" routerLinkActive="active">
        Crisis Center
    </a>
    <a routerLink="/heroes" routerLinkActive="active">
        Heroes
    </a>
</nav>
```

TASK 4 - ROUTING

- Erstelle ein Komponente, die einen einzelnen Speaker darstellen soll – `SpeakerDashboardComponent`
- ▪ auf der Route `speakers` wird die Liste der verfügbaren Speaker mit einem Link auf den Einzel-Speaker angezeigt.
- ▪ Einzelspeaker erreichbar unter der Route `speaker/:id`
- ▪ ◦ Angedeutetes Profil mit Liste seiner Talks
- ▪ ◦ Formular zum hinzufügen eines Talks zu seine Liste
- die ursprüngliche Komponente ist jetzt unter `call-for-papers` erreichbar

BASICS - DEPENDENCY INJECTION

DEPENDENCY INJECTION

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

import { HEROES }     from './mock-heroes';

@Injectable()
export class HeroService {
  getHeroes() { return HEROES; }
}
```

DEPENDENCY INJECTION

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

import { HEROES }     from './mock-heroes';

const heroesPromise = Promise.resolve(HEROES);

@Injectable()
export class HeroService {
  getHeroes(): Promise<Hero[]> { return heroesPromise; }
}
```

DEPENDENCY INJECTION

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

import { HeroService }         from './hero.service';

@Component({
  selector: 'my-heroes',
  providers: [HeroService],
  template: `
  <h2>Heroes</h2>
  <hero-list></hero-list>
  `
})
export class HeroesComponent {
    constructor (private service: HeroService) {}
 }
```

TASK 5 - BACKENDSERVICE

- Erstelle einen `BackendService`, der sowohl Speaker als auch Talks vorhält
- Benutze den Service für:
- ▪ Speakers Liste auf der CfP Seite
- ▪ Speakers Liste auf der Speaker Seite
- ▪ Speaker Profil und dessen Talk Liste

BASICS - CONCLUSION

ADVANCED

ADVANCED - EVENTS

EVENTS - EINE IDEE

MESSAGEBUS

MESSAGEBUS

```
import { Injectable } from '@angular/core';
import { Observable, Subject } from 'rxjs/Rx';
interface Message {channel: string; data: any;}
@Injectable()
export class MessageService {
    private subject = new Subject<Message>();

    public publish<T>(message: T): void {
        const channel = (<any>message.constructor).name;
        this.subject.next({ channel: channel, data: message });
    }
    public of<T>(messageType: { new(...args: any[]): T }): Observable<ar
        const channel = (<any>messageType).name;
        return this.subject.filter(m => m.channel === channel).map(m =>
    }
}
```

EVENTS - INNER COMPONENT

```
import {Component, EventEmitter, Input, Output} from "@angular/core";
@Component({
    selector: 'inner-component',
    template: '<h1>{{title}}</h1>'
})
export class InnerComponent {
    @Input() title: string = '';
}
```

EVENTS -- INNER COMPONENT - SUBMIT EVENT

```
import {Component, EventEmitter, Input, Output} from "@angular/core";
@Component({
    selector: 'inner-component',
    template: `
        <h1>{{title}}</h1>
        <p><button (click)="onClick()">Klick mich</button></p>
        `
})
export class InnerComponent {
    @Input() title: string = '';
    @output onClick: EventEmitter<void> = new EventEmitter();
    onClick() {
        this.onClick.emit();
    }
}
```

EVENTS - OUTER COMPONENT

```
import {Component} from "@angular/core";
@Component({
    selector: 'outer-component',
    template: '<inner-component [title]="title"></inner-component>'
})
export class OuterComponent {
    title: string = 'Title';

    onInnerClick(): void {
        alert('Innen wurde gedrückt');
    }
}
```

EVENTS - OUTER COMPONENT - CATCHING

```
import {Component} from "@angular/core";
@Component({
    selector: 'app',
    template: `
        <outer-component (onClick)="onInnerClick()">
        </outer-component>`
})
export class AppComponent {
}
```

ADVANCED - FORMS

FORMS

```
import {Component} from "@angular/core";
@Component({
    selector: 'app',
    template: `
        <form>
            <input
                type="text"
                [(ngModel)]="name"
                name="name"
                id="name" />
        </form>
    `
})
export class AppComponent {
    name: string = 'Max';
}
```

FORMS - REACTIVE FORMS

FORMS - REACTIVE FORMS

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

import { AppComponent }         from './app.component';
@NgModule({
  imports: [
    BrowserModule,
    ReactiveFormsModule
  ],
  declarations: [
    AppComponent
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

FORMS - REACTIVE FORMS - TEMPLATE

```
<h2>Hero Detail</h2>
<h3><i>FormControl in a FormGroup</i></h3>
<form [formGroup]="heroForm" novalidate>
 <div class="form-group">
   <label class="center-block">Name:
     <input class="form-control" formControlName="name">
   </label>
 </div>
</form>
```

FORMS - REACTIVE FORMS - COMPONENT

```
import { Component }              from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';
@Component({
    selector: 'hero-detail',
    templateUrl: 'hero-detail.html'
})
export class HeroDetailComponent {
  heroForm: FormGroup;

  constructor(private fb: FormBuilder) { this.createForm()}

  createForm() {
    this.heroForm = this.fb.group({name: ''});
  }
}
```

FORMS - REACTIVE FORMS

```
<p>Form value: {{ heroForm.value | json }}</p>
<p>Form status: {{ heroForm.status | json }}</p>
```

FORMS - REACTIVE FORMS - DOCS LESEN

TASK 6 - REACTIVE FORMS UND EVENTS

- Ersetze Eingabe-Formular zum Anlegen eines Speakers durch ein `Reactive Form`
- Zerteile das Formular dazu in mehrere Komponenten

```
<speaker-form>
      <form>
            <speaker-name></speaker-name>
            <speaker-email></speaker-email>
            <speaker-buttons></speaker-buttons>
      </form>
</speaker-form>
```

CONCLUSION

TASK 7 - ZUSATZ AUFGABE

- Erstelle eine Listenansicht für die Talks
- Erstelle ein Admin zum Annehmen und Disponieren von Talks
- Erstelle ein Dashboard in dem alle angenommen Talks in ihren Timeslots dargestellt werden

Q&A - EURE PLATTFORM-APPLICATION

LINKS

- RxJs
- Jasmine
- Reactive Forms
- Reactive Forms on steroids
- RxJs Slides