Beispiel App mit Angular/Material

Allgemeines zu Angular/Material

Material sind die Designkomponenten - Icons, Schrift, Eingabefelder, Buttons, ... - für Android-Apps. Angular ist ein Framework basierend auf Typescript zum Entwickeln von Responsive-Webseiten (also Seite, die sich automatisch an die Größe etc. des Bildschirms anpassen) und Smartphone-Apps.

Die Entwicklungsumgebung für Angular/Material basiert auf nodejs/npm. Für das Deployment auf einen Webserver werden daraus dann nur statische Dateien (Javascript/CSS/Fonts/...) erzeugt.Diese Dateien sind zusätzlich über Webpack gepackt, wodurch sich die Anzahl drastisch reduziert und die initiale Ladezeit minimiert wird.

Es ist möglich Angular-Applikationen in einem "Offline-Modus" zu betreiben. Das bedeutet, nach dem laden des Codes in den Browser können mehr oder weniger große Teile der Applikation lokal d.h. ohne Kommunikation zum Webserver laufen. Damit bei der Navigation über Buttons/Links klar ist, ob dieser Link lokal bzw. zu einer externen Resource geht, wird hier ziwschen

- routerLink: Verlinkung innerhalb der Applikation (keine Kommunikatione zum Server)
- URLs: Verlinkung URLs (Kommunikation zum Server)

```
<a href="https://..."></a>
<a href="/help/error.html"></a>
```

unterschieden.

Angular/Material is Komponenten-basiert. D.h. jede Komponente implementiert einen HTML-Tag, der dann an "beliebigen Stellen" verwendet werden kann. Generell wurd hier zwischen Code und Layout getrennt. Das Layout wird in HTML-Templates bzw. CSS-Dateien beschrieben. Im HTML-Code können dann Referenzen zu Variblen bzw. Funktionen aus dem Typescript-Code stehen.

Der Code einer Komponente beschreibt dann welcher Tag implementiert wird und welches Template bzw. welche Stylesheets benötigt werden. Hier z.B. der Tab "app-news", das HTML-Template "news.component.html" und das Stylesheet "news.component.css"

```
@Component({
  selector: 'app-news',
  templateUrl: './news.component.html',
  styleUrls: ['./news.component.css']
})
```

Im HTML-Template ist dann statt "{{title}}" zur Laufzeit der Inhalt der Variablen "title" angezeigt. Beim Click auf den "Cancel"-Button wird hier die Funktion "doCancel()" aufgerufen.

```
<div fxShow fxHide.lt-sm>{{title}}</div>
<button mat-raised-button (click)="doCancel()">Cancel</button>
```

Entwickungsumgebung

IDE

Zum Editieren von Angular/Typscript/HTML/CSS nimmt man am Besten eine IDE mit Syntax-Higlighting und Syntax-Check wie z.B. vscode (Opensource von Microsoft) oder Atom (Opensurce). Beide können durch Plugins erweiter werden, dass die Angular/Typescript/nodejs/HTML/CSS unterstützen. Insbesonders vscode benötigt herzu aber einiges an freiem Memory.

Vorbereiten

Checken ob auf OS-level die notwendigen Komponenten installiert sind (RPM bzw. DEB je nach Linux)

```
nodejs
npm
```

Wenn Webserver

```
docker
docker-compose
```

Angular-CLI instalieren.

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

Anlegen neue App

Erzeugen der "leeren" **CoveApp**

```
ng new CoveApp  # Frage nach Routing: "Yes", Stylesheet: "CSS"
cd CoveApp
npm install
```

Installieren Libraries

Touch-Steuerung für Smartphone

```
npm install hammerjs --save
```

Flexibles Layout for Smartphone und Desktop/Laptop

```
npm install @angular/flex-layout --save
npm install angular-resizable-element--save
```

Angular Material und Lokalisierung

```
ng add @angular/material  # Theme: "indigo-pink", typography styles: No,
Animation: "Yes"
ng add @angular/localize
ng add @angular/pwa
```

Googlefonts und icons lokal installieren, damit beim Aufruf später keine Zugriff auf google erfolgt.

```
npm install typeface-roboto --save
npm install material-design-icons --save
```

Die Liste der Standart-Material-Icons findet sich unter https://material.io/resources/icons/

```
npm install angular-calendar --save
```

Die Flaggen für Sparachauswahl

```
npm install flag-icon-css --save
```

Test setup

```
ng serve
```

Im Browser die URL **http://localhost:4200/**

Anmerkung: **ng serve** übersetzt bei jeder Dateiänderung den geänderten Code neu. Der Browser aktualisiert dann automatisch. Dies funktioniert zu 95%, ab und zu muss man ng serve aber mal abbrechen und neu starten (wenn mal Fehler angezeigt werden, die man sich nicht erklären kann, bevor da nach einem Phantom-Fehler gestzt wird). Das passiert besonders dann, wenn "grundlegende Änderungen" gemacht werden.

Material Icons + lokaler Roboto Font

In Datei **angular.json**

Setzung von "Styles": [...] ersetzen (2mal)

Ergebnis:

```
"styles": [
      "./node_modules/typeface-roboto/index.css",
      "./node_modules/material-design-icons/iconfont/material-icons.css",
      "./node_modules/@angular/material/prebuilt-themes/indigo-pink.css",
      "src/styles.css"
],
```

Löschen der Google-Referenzen in src/index.html

Die beiden Links auf Google in **src/index.html** löschen

```
<link href="https://fonts.googleapis.com/css?
family=Roboto:300,400,500&display=swap" rel="stylesheet">
<link href="https://fonts.googleapis.com/icon?family=Material+Icons"
rel="stylesheet">
```

Leeren src/app/app.component.html

Ersetzen durch

```
<h1>{{title}}</h1>
<router-outlet></router-outlet>
```

Kopieren der Icons

```
src/assets/logo.gif
src/assets/logo.png
src/favicon.ico
```

Die Material-Componenten einbinden

Kopieren von

```
src/material.module.ts
```

in **src/app/app.module.ts** nach den **import {..} ..;** einfügen

```
import { MaterialModule } from '../material.module';
import { FlexLayoutModule } from '@angular/flex-layout';
```

in das Array **imports** 2 neue Einträge

```
imports: [
    MaterialModule,
    FlexLayoutModule,
```

Generiere Model

Zum Speicher von Daten werden Datenstrukturen benötigt, den Rahmen kann man dafür generieren

```
ng generate class model/news
ng generate class model/chat
```

Generiere Service

```
ng generate service services/chat
ng generate service services/news
```

Generiere Komponenten

Die Komponenten für die "Hauptseite" der App

```
ng generate component page-not-found
ng generate component login
ng generate component menu
ng generate component drawer
ng generate component chat
ng generate component confirm-dialog
```

Die Komponenten, die In der App angezeigt werden sollen

```
ng generate component dashboard
ng generate component news
ng generate component calendar
ng generate component email
ng generate component conference
```

Routing von URLs zu Komponenten

Damit verschiedene Seiten auch über URLs direkt erreichbar werden können (z.B. Bookmarks) benötigt man das Routing. Hier werden die URLs auf Komponenten gemapped. Damit das ausserhalb der Entwicklungsumgebung auch funktioniert, muss der Webserver später noch passend konfiguriert werden.

in app-routing.module.ts

```
import { DashboardComponent } from './dashboard/dashboard.component';
import { CalendarComponent } from './calendar/calendar.component';
import { EmailComponent } from './email/email.component';
import { NewsComponent } from './news/news.component';
import { ConferenceComponent } from './conference/conference.component';
import { LoginComponent} from './login/login.component';
import { PageNotFoundComponent } from './page-not-found/page-not-
found.component';
```

Ersetzen von

```
const routes: Route = [];
```

durch

```
const routes: Routes = [
  { path: 'login', component: LoginComponent},
  { path: 'dashboard', component: DashboardComponent},
  { path: 'calendar', component: CalendarComponent},
  { path: 'email', component: EmailComponent},
  { path: 'news', component: NewsComponent},
  { path: 'conference', component: ConferenceComponent},
  { path: '**', component: PageNotFoundComponent },
];
```

Die Zeile **...'dashboard'...** mapped die URL **/dashboard** auf die Komponente **DashboardComponent**. Die letzte Zeile ist der Default, wenn eine ungültige URL aufgerufen wird.

Im Browser führt jetzt **http://localhost:4200/dashboard** zu **dashboard works!** dahingegen **http://localhost:4200/xx** zu **page-not-found works!**

app - Anpassen Hauptseite

Hauptseite nun mit den Teilen Toolbar mit dem Menü (oben) Drawer (erscheint links bei Klick auf "Burger-Icon") Chat (erscheint rechts bei Klick auf Chat-Icon)

app.component.ts

Zusätzliche Imports

```
import { ResizeEvent } from 'angular-resizable-element';
import { SwUpdate, SwPush } from '@angular/service-worker';
import { MatSnackBar } from '@angular/material/snack-bar';
```

Ersetzen von

```
export class AppComponent {
  title = 'CoveApp';
}
```

```
export class AppComponent {
  title = 'Cove-App';
  chatVisible = false;

  onResizeEnd(event: ResizeEvent): void {
    console.log('Element was resized', event.rectangle.left,
event.rectangle.right, event.edges.left);
  }

  receiveMessage($event) {
    console.dir($event)
    this.chatVisible = !this.chatVisible
  }
}
```

app.component.css

```
.mat-toolbar{
  height: 50px;
}

.mat-drawer-content {
    margin-right: 0px !important;
    padding-left: 10px;
    padding-right: 10px;
    height: 100%;
}

.mat-drawer-container{
    height: calc(100% - 70px);
}
```

app.component.html

```
<mat-toolbar>
  <mat-toolbar-row>
    <button mat-icon-button (click)="drawer.toggle()">
      <mat-icon>menu</mat-icon>
    </button>
    <img src="./assets/logo.png" style="width:50px;" />
    <div fxShow fxHide.lt-sm>{{title}}</div>
    <span fxFlex></span>
    <app-menu (messageEvent)="receiveMessage($event)"></app-menu>
  </mat-toolbar-row>
</mat-toolbar>

<mat-drawer-container hasBackdrop="false">
  <mat-drawer #drawer mode="push">
    <app-drawer></app-drawer>
  </mat-drawer>

  <mat-drawer-content>
    <div fxLayout="row" fxLayout.lt-sm="column" fxFlex="100%">
      <div fxFlex="auto">
        <router-outlet></router-outlet>
      </div>
      <div fxFlex="300px" *ngIf="chatVisible">
        <app-chat></app-chat>
      </div>
    </div>
  </mat-drawer-content>
</mat-drawer-container>
```

src/app/drawer

Der Drawer enthält einige Buttons um nach dem Klick in der Mitte die Komponenten erscheinen zu lassen. Der Drawer erscheint, wenn auf das "Burger-Menu" geklickt wird.

drawer.component.html

```
<mat-selection-list>
    <mat-list-item>
        <button mat-menu-item routerLink="/dashboard" class="dashboard-
icon">
            <mat-icon>dashboard</mat-icon>
            <div class="menu-label" ffxShow fxHide.lt-sm>Dashboard</div>
        </button>
    </mat-list-item>
    <mat-list-item>
        <button mat-menu-item routerLink="/calendar" class="calendar-icon">
            <mat-icon>event</mat-icon>
```

```
            <div class="menu-label" fxShow fxHide.lt-sm>Calendar</div>
        </button>
    </mat-list-item>
    <mat-list-item>
        <button mat-menu-item routerLink="/email" class="email-icon">
            <mat-icon>email</mat-icon>
            <div class="menu-label" fxShow fxHide.lt-sm>EMail</div>
        </button>
    </mat-list-item>
    <mat-list-item>
        <button mat-menu-item routerLink="/news" class="news-icon">
            <mat-icon>rss_feed</mat-icon>
            <div class="menu-label" fxShow fxHide.lt-sm>News</div>
        </button>
    </mat-list-item>
    <mat-list-item>
        <button mat-menu-item routerLink="/conference" class="conference-
icon">
            <mat-icon>video_call</mat-icon>
            <div class="menu-label" fxShow fxHide.lt-sm>Conference</div>
        </button>
    </mat-list-item>
</mat-selection-list>
```

Globale CSS src/style.css

Damit in den Buttons überall das gleiche Layout genutzt wird, die Einstellungen in der globalen **style,css** In style.css anfügen (Text + Icons in "einer Zeile", Button ohne Rahmen).

```
.menu-label{
    display: inline;
}

.mat-menu-item {
    border-width: 0px;
}
```

src/app/menu

menu.component.html

```
<div>
    <button mat-icon-button>
        <mat-icon [matBadge]="emailCount" [matBadgeHidden]="emailCount<=0"
matBadgeColor="warn">email</mat-icon>
    </button>
    <button mat-icon-button>
```

```
        <mat-icon [matBadge]="newsCount" [matBadgeHidden]="newsCount<=0"
matBadgeColor="warn">rss_feed</mat-icon>
    </button>
    <button mat-icon-button (click)="toggleChat()">
        <mat-icon [matBadge]="chatCount" [matBadgeHidden]="chatCount<=0"
matBadgeColor="warn">chat</mat-icon>
    </button>
    <button mat-icon-button>
        <mat-icon>favorites</mat-icon>
    </button>
    <button mat-icon-button>
        <mat-icon>help</mat-icon>
    </button>

    <button mat-icon-button [matMenuTriggerFor]="userMenu">
        <mat-icon>account_circle</mat-icon>
    </button>

    <mat-menu #userMenu="matMenu" xPosition="before">
        <div>
            <button mat-menu-item routerLink="/login">
                <mat-icon>lock_open</mat-icon>
                <div class="menu-label" fxShow fxHide.lt-sm>Login</div>
            </button>
            <button mat-menu-item routerLink="/profile">
                <mat-icon>person</mat-icon>
                <div class="menu-label" fxShow fxHide.lt-sm>Profile</div>
            </button>
            <button mat-menu-item routerLink="/password">
                <mat-icon>security</mat-icon>
                <div class="menu-label" fxShow fxHide.lt-sm>Password</div>
            </button>

            <button mat-menu-item routerLink="/settings">
                <mat-icon>settings</mat-icon>
                <div class="menu-label" fxShow fxHide.lt-sm>Settings</div>
            </button>
            <button mat-menu-item>
                <mat-icon>lock_open</mat-icon>
                <div class="menu-label" fxShow fxHide.lt-sm>Logout</div>
            </button>
        </div>
    </mat-menu>
</div>
```

menu.component.ts

Besehenden Import anpassen

```
import { Component, OnInit, Output, EventEmitter } from '@angular/core';
```

Klasse MenuComponent ersetzen durch

```
export class MenuComponent implements OnInit {
  emailCount: number = 10
  chatCount: number = 5
  newsCount: number = 1
  popoverValue: number = 0
  result: string = '';

  @Output() messageEvent = new EventEmitter<string>();
  constructor() {
  }

  toggleChat() {
    this.chatCount = 0
    this.messageEvent.emit("toggleChat")
  }
}
```

src/app/chat

chat.component.css

```
.chat{
  background-color: lightgrey;
  height: 100%;
}
```

chat.component.html

```
<div class="chat">
  <p>chat works!</p>
</div>
```

src/app/service/news-service.ts

Hier wird ein "Dummy" Service erstellt, der immer die gleichen Daten bereitstellt. Hier kommt dann der Richtie Zugrif auf die REST-API des Application-Servers hin. Dies ist auch eine gängige Vorgehensweise zum Testen (auch Mockup genannt).

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

export class News {
  id: number;
  title: string;
```

```
  source: string;
  foundat: Date;
  url: string;
}

@Injectable({
  providedIn: 'root'
})
export class NewsService {
  newsList: News[] = [
    {
      id: 1,
      title: "Sicherheits-Update: Apple schließt kritische Lücken auf alten
Macs",
      source: 'Heise',
      foundat: new Date("2020-11-12 12:00"),
      url: "https://www.heise.de/news/Sicherheits-Update-Apple-schliesst-
kritische-Luecken-auf-alten-Macs-4960291.html"
    },
    {
      id: 1,
      title: "Bauvorschlag 2020: Leiser Allround-PC mit Ryzen-5000-
Aufrüstoption",
      source: 'Heise',
      foundat: new Date("2020-11-12 14:00"),
      url: "https://www.heise.de/ratgeber/Bauvorschlag-2020-Leiser-
Allround-PC-mit-Ryzen-5000-Aufruestoption-4946260.html?
wt_mc=intern.red.plus.plus_buehne.startseite.buehne.buehne"
    },
    {
      id: 1,
      title: "Wetter", source: 'Heise', foundat: new Date("2020-11-12
08:00"), url: "https://www.t-online.de/wetter/"
    }
  ];

  constructor() { }

  public getNews(): News[] {
    return this.newsList;
  }
}
```

src/app/news

Die News erscheinen im mittleren "Work-Bereich"

news.component.ts

```
import { Component, OnInit } from '@angular/core';
import { NewsService } from '../services/news.service';
```

```
@Component({
  selector: 'app-news',
  templateUrl: './news.component.html',
  styleUrls: ['./news.component.css']
})
export class NewsComponent implements OnInit {

  constructor(public newsService: NewsService) { }

  ngOnInit(): void {
  }

  getNews():: News[]{
    return this.newsService.getNews();
  }
}
```

news-component.html

Die Verbindung des HTML-Templates zu den Daten ist hier die Zeile * die in der Classe NewsComponent die Methode **getNews()** aufruft, die ein Array mit den News zurückliefert und dann für jeden eintrag einen

erzeigt. Hier wird auch ein Link auf eine externe Webseite **href="{{news.url}}"** erzeugt, der dann im Browser in einem neuen Tab angezeigt wird **target="_blank"** Eine besoderheit ist hier das **{news.foundat | date: 'medium'}}** hier wird das Feld news.foundat als Datum im Format medium formatiert.

```
<div class="news-page-title">News</div>

<div style="width: 100%">

        <div fxLayout="column">
            <div *ngFor="let news of getNews()">
                <div class="news-page-entry">
                    <div class="news-page-entry-title">
                        <div>
                            <a target="_blank" href="{{news.url}}">
                                <mat-icon>open_in_new</mat-icon>
                            </a>
                            {{news.title}}
                        </div>
                    </div>
                    <div class="news-page-entry-source">Quelle:
{{news.source}}</div>
                    <div class="news-page-entry-foundat">Gefunden am:
{{news.foundat | date: 'medium'}}</div>
                </div>
            </div>
        </div>
</div>
```

news.component.css

```
.news-page-title {
    font-size: 1.5em;
    font-weight: bold;
}

.news-page-entry{
   border-style: solid;
   border-width: 1px;
   width: 100%;
}

.news-page-entry-title{
    font-weight: bold;
}

.news-page-entry-title{
}

.news-page-entry-source{
}

.news-page-entry-foundat{
}

.news-viewport {
    height: 400px;
    border: 1px solid black;
  }

.news-item {
    height: 50px;
}
```

Jetzt kann im Browser auch die URL **http://localhost:4200/news** aufgerufen werden um direkt zu den News zu springen.

Progressive-Web-App (PWA)

Damit die Angular-Applikation als PWA lauffähig wird, müssen nur kleine Änderungen gemacht werden.

manifest.webmanifest

```
{
  "name": "CoveApp",
  "short_name": "CoveApp",
  "theme_color": "#1976d2",
```

```
  "background_color": "#fafafa",
  "display": "standalone",
  "scope": "./",
  "start_url": "./",
  "icons": [
    {
      "src": "assets/logo.png",
      "sizes": "144x144",
      "type": "image/png",
      "purpose": "maskable any"
    }
  ]
}
```

src/app/app.component.ts

Imports ändern

```
import { Component, OnInit } from '@angular/core';
import { ResizeEvent } from 'angular-resizable-element';
import { SwUpdate, SwPush } from '@angular/service-worker';
import { MatSnackBar } from '@angular/material/snack-bar';
```

In die Klasse AppComponent einfügen

```
  constructor(
    private snackBar: MatSnackBar,
    private swUpdate: SwUpdate) {
  }

  ngOnInit() {
    if (this.swUpdate.isEnabled) {
      this.setupUpdates();
    }
  }

  setupUpdates() {
    this.swUpdate.available.subscribe(u => {
      // Update wurde entdeckt

      // Update herunterladen
      this.swUpdate.activateUpdate().then(e => {
        // Update wurde heruntergeladen

        const message = 'Application has been updated';
        const action = 'Ok, Reload!';

        // Benutzer auf Update hinweisen und Seite neu laden
        this.snackBar.open(message, action).onAction().subscribe(
```

```
            () => location.reload()
          );
        });
      });

      // Auf Updates prüfen
      this.swUpdate.checkForUpdate();
    }
```

src/app/login

Dies ist ein Login-Dialog mit Einagebfeldern und eine einfache Feldvalidierung (Username länger als 3 Zeichen, Passwort länger als 8 Zeichen). Statt dem Login erfolgt eine Ausgae der Login-Parameter.

Damit Formulare genutz werden können müssen in **src/app.module.ts** die Module für Formulare geladen werden

```
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
```

Diese müssen auch noch im Array **imports** importiert werden

```
...
  imports: [
    FormsModule,
    ReactiveFormsModule,
...
```

login.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';
import { FormGroup, FormControl, Validators } from '@angular/forms';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  public loginForm: FormGroup;
  hide = true;
  loginError = false;
  retUrl: string = "home";

  constructor(
    private router: Router,
```

```
    private activatedRoute: ActivatedRoute) { }

  ngOnInit(): void {
    this.activatedRoute.queryParamMap
      .subscribe(params => {
        this.retUrl = params.get('returnUrl');
        console.log('LoginComponent/ngOnInit ' + this.retUrl);
      });
    this.loginForm = new FormGroup({
      username: new FormControl('', [Validators.required,
Validators.minLength(3)]),
      password: new FormControl('', [Validators.required,
Validators.minLength(8)])
    })
  }

  public hasError = (controlName: string, errorName: string) => {
    return this.loginForm.controls[controlName].hasError(errorName);
  }

  doCancel() {
    this.router.navigate([this.retUrl || '/']);
  }

  doRegister() {
    console.log('register')
  }

  doLogin() {
    let credentials = this.loginForm.value;
    this.loginError = false;

    alert(`login ${credentials.username} ${credentials.password}`);
  }
}
```

login.component.html

```
<div class="login-wrapper" fxLayout="row" fxLayoutAlign="center center">
  <mat-card class="box">
    <mat-card-header>
      <mat-card-title>Log in</mat-card-title>
    </mat-card-header>

    <form [formGroup]="loginForm" autocomplete="off" novalidate
(ngSubmit)="doLogin()" fxLayout="column wrap"
      fxLayoutAlign="center center" fxLayoutGap="10px" class="form">
      <mat-card-content>
        <mat-form-field>
          <input matInput required placeholder="Username"
formControlName="username">
```

```
          <mat-error *ngIf="hasError('username', 'required')">Username ist
Mussfeld</mat-error>
          <mat-error *ngIf="hasError('username', 'minlength')">Username
muss mindestens 3 Zeichen enthalten</mat-error>
        </mat-form-field>

        <mat-form-field>
          <mat-label>Passwort</mat-label>
          <input matInput #password required placeholder="Passwort"
formControlName="password">
          <mat-error *ngIf="hasError('password', 'required')">Passwort ist
Mussfeld</mat-error>
          <mat-error *ngIf="hasError('password', 'minlength')">Passwort
muss mindestens 8 Zeihen enthalten</mat-error>
        </mat-form-field>
      </mat-card-content>
      <mat-card-actions align="center">
        <button mat-raised-button color="primary"
[disabled]="!loginForm.valid">Login</button>
        <button type="button" mat-raised-button
routerLink="/register">Register</button>
        <button type="button" mat-raised-button
(click)="doCancel()">Cancel</button>
      </mat-card-actions>
      <mat-error *ngIf="loginError">Username/Passwort ungültig</mat-error>
    </form>
  </mat-card>
</div>
```

src/app/conference

Dies ist eine Demo für Webcam und Bildschirmteilen. Es wir die lokale Webcam und zur "Demmo" als Remote-Video nocheinmal die lokale Webkam angezeigt. Beim Bildschirmteilen der eigene geteilte Bildschirm bzw. das eigene geteilte Fenster angezeigt.

Im lokalne Bild der Webcam kännen Darstelulungseffekte ausgewählt werden und es kann der Spiegelmodus (eigenes Bild wie im Spiegel) ausgewählt werden.

Da nur lokaler nur Ton Rückkopplungen erzeugt, wurden hier nur die Bedienelemente implementiert.

Hier werden die Typscript-Typen der Webrtc-Schnittstelle benätigt.

```
npm install @types/webrtc --save
```

Der Code dieses Beispieles ist schon komplexer.

conference.component.ts

```
import { Component, OnInit, Renderer2, ElementRef, ViewChild } from
'@angular/core';
import { WebcamInitError } from 'ngx-webcam';

@Component({
  selector: 'app-conference',
  templateUrl: './conference.component.html',
  styleUrls: ['./conference.component.css']
})
export class ConferenceComponent implements OnInit {
  @ViewChild('remoteScreen', { static: true }) remoteScreen: ElementRef;
  @ViewChild('localVideo', { static: true }) localVideo: ElementRef;
  @ViewChild('remoteVideo', { static: true }) remoteVideo: ElementRef;
  public error: String;

  public micOn = false;

  public speakerOn = false;

  public screenShare: boolean = false;
  public mediaStream = null;

  public showWebcam: boolean = false;
  public localVideoStream = null;
  public mirrorWebcam = false;
  public localVideoFilter = 'none';

  public remoteVideoStream = null;

  public webcamResolution = 2;
  public webcamResolutions = [
    {
      name: 'QVGA (160*120)',
      video: { width: 160, height: 120 }
    },
    {
      name: 'HVGA (320*240)',
      video: { width: 320, height: 240 }
    },
    {
      name: 'VGA (640*480)',
      video: { width: 640, height: 480 }
    },
    {
      name: 'XVGA (1024*768)',
      video: { width: 1024, height: 768 }
    },
    {
      name: 'HD (1280*720)',
      video: { width: 1280, height: 720 }
    },
    {
      name: 'fullHD (1920*1080)',
```

```
        video: { width: 1920, height: 1080 }
      },
      {
        name: '4K (4096*2160)',
        video: { width: 4096, height: 2160 }
      },
      {
        name: '8K (7680*4320)',
        video: { width: 7680, height: 4320 }
      }
    ];

    constructor(private renderer: Renderer2) {
    }

    ngOnInit(): void {
    }

    public handleInitError(error: WebcamInitError): void {
      if (error.mediaStreamError && error.mediaStreamError.name ===
"NotAllowedError") {
        this.error = "Kamera ist nicht erlaubt worden!";
        this.showWebcam = false;
      }
    }

    public toggleMirror(): void {
      this.mirrorWebcam = !this.mirrorWebcam;
    }

    public webcamOff(ev) {
      console.dir(ev);
      this.localVideoStream = null;
      this.showWebcam = false;

      // DEMO: gebe locales Video auch als Remote Video aus
      this.remoteVideoStream = null;
    }

    public webcamOn() {
      // @ts-ignore
      navigator.mediaDevices.getUserMedia({ video:
this.webcamResolutions[this.webcamResolution].video })
        .then(this.handleWebcam.bind(this), this.handleError);
    }

    public handleWebcam(stream) {
      let obj = this;
      this.showWebcam = true;
      this.localVideoStream = stream;
      this.localVideoStream.oninactive = function (ev) { obj.webcamOff(ev) };
      console.dir(this.localVideoStream);
      this.renderer.setProperty(this.localVideo.nativeElement, 'srcObject',
stream);
```

```
    // DEMO: gebe locales Video auch als Remote Video aus
    // Hack: Bei Firefox hat die Funktion captureStream den Namen
mozCaptureStream
    if (this.localVideo.nativeElement.mozCaptureStream) {
      this.remoteVideoStream =
this.localVideo.nativeElement.mozCaptureStream();
    } else {
      this.remoteVideoStream =
this.localVideo.nativeElement.captureStream();
    }
    this.renderer.setProperty(this.remoteVideo.nativeElement, 'srcObject',
this.remoteVideoStream);
  }

  public handleShareScreen(stream) {
    let obj = this;
    this.screenShare = true;
    this.mediaStream = stream;
    this.mediaStream.oninactive = function (ev) { obj.screenShareOff(ev) };
    console.dir(this.mediaStream);
    this.renderer.setProperty(this.remoteScreen.nativeElement, 'srcObject',
stream);

  }

  public handleError(error) {
    console.log('Error: ', error);
  }

  public screenShareOff(ev) {
    this.mediaStream = null;
    this.screenShare = false;
    console.dir(ev);
  }

  public screenShareOn() {
    // @ts-ignore
    navigator.mediaDevices.getDisplayMedia({ video:
this.webcamResolutions[this.webcamResolution].video })
      .then(this.handleShareScreen.bind(this), this.handleError);
  }

  public toggleMic(): void {
    this.micOn = !this.micOn;
  }

  public toggleSpeaker(): void {
    this.speakerOn = !this.speakerOn;
  }

  public toggleWebcam(): void {
    let elem = this.localVideo.nativeElement.srcObject;
    if (elem) {
```

```
      let tracks = elem.getTracks();
      tracks.forEach(track => track.stop());
      this.renderer.setProperty(this.localVideo.nativeElement, 'srcObject',
null);

      // DEMO: gebe locales Video auch als Remote Video aus
      this.renderer.setProperty(this.remoteVideo.nativeElement,
'srcObject', null);
    }

    if (!this.showWebcam) {
      this.webcamOn();
    } else {
      this.webcamOff(null);
    }
  }

  public toggleScreenShare() {
    let elem = this.remoteScreen.nativeElement.srcObject;
    if (elem) {
      let tracks = elem.getTracks();
      tracks.forEach(track => track.stop());
      this.renderer.removeAttribute(this.remoteScreen.nativeElement, null);
    }

    if (!this.screenShare) {
      this.screenShareOn();
    } else {
      this.screenShareOff(null);
    }
  }
}
```

conference.component.css

```
video {
  background-color: #f5f5f5;;
  padding: 5px;
}

.red-icon {
    color: red;
}

.green-icon {
  color: greenyellow;
}

.border {
  border-width: 1px;
  border-color: black;
```

```
  border-style: solid;
  height: 40px;
}


.none {
  -webkit-filter: none;
  filter: none;
}

.blur {
  -webkit-filter: blur(3px);
  filter: blur(3px);
}

.grayscale {
  -webkit-filter: grayscale(1);
  filter: grayscale(1);
}

.invert {
  -webkit-filter: invert(1);
  filter: invert(1);
}

.sepia {
  -webkit-filter: sepia(1);
  filter: sepia(1);
}
```

conference.component.html

```
<div>
  <div class="toolbar" fxLayout="row">
    <div class="border">
      <mat-form-field>
        <mat-select [(value)]="webcamResolution">
          <mat-option *ngFor="let resolution of webcamResolutions; index as
i" [value]="i">
            {{resolution.name}}
          </mat-option>
        </mat-select>
      </mat-form-field>
    </div>
    <div class="border">
      <button mat-icon-button [ngClass]="{'red-icon' : showWebcam}"
(click)="toggleWebcam();">
        <mat-icon>{{showWebcam?'videocam':'videocam_off'}}</mat-icon>
      </button>
      <button mat-icon-button (click)="toggleMirror();" *ngIf="showWebcam">
        <mat-icon>swap_horiz</mat-icon>
```

```
      </button>
      <mat-form-field *ngIf="showWebcam">
        <mat-select [(value)]="localVideoFilter">
          <mat-option value="none">Kein Filter</mat-option>
          <mat-option value="invert">Invers</mat-option>
          <mat-option value="grayscale">Schwarzweiß</mat-option>
          <mat-option value="blur">Blur</mat-option>
          <mat-option value="sepia">Sepia</mat-option>
        </mat-select>
      </mat-form-field>

    </div>
    <div class="border">
      <button mat-icon-button [ngClass]="{'red-icon' : micOn}"
(click)="toggleMic();">
        <mat-icon>{{micOn?'mic':'mic_off'}}</mat-icon>
      </button>
      <mat-slider *ngIf="micOn" min="1" max="100" step="5" value="50">
</mat-slider>
    </div>
    <div class="border">
      <button mat-icon-button [ngClass]="{'red-icon' : speakerOn}"
(click)="toggleSpeaker();">
        <mat-icon>{{speakerOn?'volume_mute':'volume_off'}}</mat-icon>
      </button>
      <mat-slider *ngIf="speakerOn" min="1" max="100" step="5" value="50">
</mat-slider>
    </div>

    <div class="border">
      <button mat-icon-button [ngClass]="{'red-icon' : screenShare}"
(click)='toggleScreenShare()'>
        <mat-icon>{{screenShare?'screen_share':'stop_screen_share'}}</mat-
icon>
      </button>
    </div>
    <hr />
  </div>
  <div>
    <div fxLayout="row" fxLayout.lt-md="column">
      <div>
        <video #localVideo playsinline autoplay
[ngClass]="localVideoFilter" [ngStyle]="mirrorWebcam ? {'transform':
'rotateY(180deg)'}: ''">Local
          video</video>
        <video #remoteVideo playsinline autoplay>Remote video</video>
        <video #remoteScreen playsinline autoplay>Remote screen</video>
      </div>
    </div>
  </div>
</div>
```

Erzeuge Distribution

```
ng build --prod
```

Erzeugt alle statischen Dateien für den Webserver mit der Produktionskonfiguration. Das ergebnis steht dann im Verzeichnis **dist**.

Der Inhalt sieht dann z.B. wie folgt aus. Es fallen die generierten Dateinamen auf und dass die Struktur bis auf das Unterverzeichnis **assets** "flach" ist.

```
3rdpartylicenses.txt                                roboto-latin-
400.176f8f5bd5f02b3abfcf.woff2
assets/                                             roboto-latin-
400.49ae34d4cc6b98c00c69.woff
favicon.ico                                         roboto-latin-
400italic.b1d9d9904bfca8802a63.woff
index.html                                          roboto-latin-
400italic.d022bc70dc1bf7b3425d.woff2
main.c7c00b4d0896522a7cf9.js                        roboto-latin-
500.cea99d3e3e13a3a599a0.woff
manifest.webmanifest                                roboto-latin-
500.f5b74d7ffcdf85b9dd60.woff2
MaterialIcons-Regular.4674f8ded773cb03e824.eot      roboto-latin-
500italic.0d8bb5b3ee5f5dac9e44.woff2
MaterialIcons-Regular.5e7382c63da0098d634a.ttf      roboto-latin-
500italic.18d00f739ff1e1c52db1.woff
MaterialIcons-Regular.83bebaf37c09c7e1c3ee.woff     roboto-latin-
700.2267169ee7270a22a963.woff
MaterialIcons-Regular.cff684e59ffb052d72cb.woff2    roboto-latin-
700.c18ee39fb002ad58b6dc.woff2
ngsw.json                                           roboto-latin-
700italic.7d8125ff7f707231fd89.woff2
ngsw-worker.js                                      roboto-latin-
700italic.9360531f9bb817f917f0.woff
polyfills.939905d55339054d2452.js                   roboto-latin-
900.870c8c1486f76054301a.woff2
roboto-latin-100.a45108d3b34af91f9113.woff          roboto-latin-
900.bac8362e7a6ea60b6983.woff
roboto-latin-100.c2aa4ab115bf9c6057cb.woff2         roboto-latin-
900italic.c20d916c1a1b094c1cec.woff
roboto-latin-100italic.451d4e559d6f57cdf6a1.woff    roboto-latin-
900italic.cb5ad999740e9d8a8bd1.woff2
roboto-latin-100italic.7f839a8652da29745ce4.woff2
runtime.acf0dec4155e77772545.js
roboto-latin-300.37a7069dc30fc663c878.woff2         safety-worker.js
roboto-latin-300.865f928cbabcc9f8f2b5.woff
styles.e08a8ede6a24c1015a97.css
roboto-latin-300italic.bd5b7a13f2c52b531a2a.woff    worker-basic.min.js
roboto-latin-300italic.c64e7e354c88e613c77c.woff2
```

Applikation mit nginx-Webserver als Docker-Container

nginx/default.conf

Die Angular-Applikation besteht aus etlichen statischen Dateien, dies sind die, die in **src/assets** abgelegt wurden unddurch genutzte Module bereitgestellte Dateien (z.B. das Module flag-icons-css ehthält alle Ländernflaggen als SVG-Files). Damit URLs auf Objekte im Javascript-Code zeigen können, müssen im Webserver alle URLs unterhalb der Applikation auf die Datei **index.html** umgelenkt werden. Der Javascript-Code ruft dann die zur URL passende Komponente auf. Dies ist notwendig, damit z.B. für URLs aus Bookmarks die korrekte Seite angezeigt wird. Dies wird in der nginx-Konfiguration durch die Zeile **try_files $uri $uri/ /index.html;** erreicht (versuche zuerst Datei zur angegebene URL, dann Verzeichniss zur angegebene-Url, sonst index.html).

```
server {
    listen       80 default_server;
    listen  [::]:80 default_server;
    listen       443 ssl http2 default_server;
    listen  [::]:443 ssl http2 default_server;
    ssl_certificate /etc/ssl/certs/nginx.crt;
    ssl_certificate_key /etc/ssl/private/nginx.key;

    server_name  localhost;

    #charset koi8-r;
    #access_log  /var/log/nginx/host.access.log  main;

    location / {
        root   /usr/share/nginx/html;
        try_files $uri $uri/ /index.html;
    }

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}
```

Selfsigned SSL-Zertifikat generieren

Datei **generatecert.sh** Hier die Daten für -subj, -addext anpassen.

```
openssl req -x509 -nodes -days 365 -subj "/C=DE/ST=NRW/L=Frankfurt/O=Cove
e.V./CN=linux6.fritz.box" -addext "subjectAltName=DNS:192.168.1.154" -
newkey rsa:2048 -keyout nginx/nginx.key -out nginx/nginx.crt
```

SSL-Zertifikat generieren

```
chmod +x ./generatecert.sh
./generatecert.sh
```

Docker

Der einfachste Weg Webserver zu erstellen und zu installieren ggf. auch in vielen instanzen geht über Docker-Container. Zum Erzeugen werden einige Konfigurationsfiles benötigt. .

Dockerfile

Die Datei **Dockerfile** enthält den "Bauplan" für das Container-Image. Das Container-Image basiert immer auf einem Basis-System, hier ein "vorgefertigtes" Container-Image mit **nginx**, zu dem dann weitere Komonenten zugefügt werden, hier sind es nur Dateien. Es können aber auch weitere Programme installiert werden etc.

Datei anlegen mit Inhalt

```
FROM nginx:stable

COPY nginx/default.conf /etc/nginx/conf.d/default.conf
COPY nginx/nginx.key /etc/ssl/private
COPY nginx/nginx.crt /etc/ssl/certs

COPY ./dist/CoveApp /usr/share/nginx/html
```

.dockerignore

Hier werden die Dateien konfiguriert, die Docker beim Bauen des Images nicht betrachten soll.

Datei **.dockerignore** anlegen mit Inhalt

```
.git
./node_modules
```

Setup docker-compose

Mit Docker-Compose können ggf. auch mehrere Images unter Berücksichtigung der Abhängigkeiten (z.B. erst den DB-Server, dann den Application-Server) mit umgebungsspezifischen Konfigurationen gestartet werden.

Datei **docker-compose.yml anlegen (hier unter Environment bei **NGINX_HOST** den externen Namen für den Web-Server eintragen).

```
    web:
      build: ./
      ports:
       - "8080:80"
       - "8443:443"
      environment:
        - NGINX_HOST=linux6.fritz.box
```

Webserver erzeugen/starten/stoppen

Container erzeugen

Aufruf zum Erzeugen des Images

```
docker-compose build
```

Container starten

Image als Daemon starten

```
docker-compose up -d
```

Test im Browser mit URL **https://localhost:8443/** Es kommt Warnung wegen Selfsigned-certificate

Status der Container

```
docker-compose ps
```

Logfile anzeiugen

```
docker-compose logs --time web
```

Container stoppen

```
docker-compose stop
```

Development-Pipeline

Ein Beipsielhafter Ablauf der Entwicklung sieht wie folgt aus

```
# Mit IDE der Wahl die Angular-Support bietet den Sourcecode modifizieren
# Test in Dev-Umgebung
ng serve

# Wenn für gut befunden für Prod erzeugen (oder andere Umgebung)
ng build --prod

# Docker-Container erzeugen
docker-compose build

# Docker-Container starten
docker-compose up -d
```

Test PWA### Start App

Der Zest ist recht einfach , obwohl es eigentlich mit einem Self-Signed-Certificate nicht geht.

Chrome aufrufen mit

```
chromium --user-data-dir=/tmp/foo --ignore-certificate-errors --unsafely-
treat-insecure-origin-as-secure=https://linux6.fritz.box:8443/
```

Hierbei die URL am Ende des Aufrufes mit der aktuellen ersetzen.

Der Test wäre dann

1. Webseite mit https aufrufen
2. Änderung in **src/app/app.component.ts** in der Zeile

```
title="CoveApp"
```

Den String ändern.

```
ng build --prod
docker-comompose build
docker-compose up -d
```

Spätestens beim nähsten Auruf der URL bzw. beim Browser-Refresh kommt dann die Mitteilung **Application has been updated OK, Reload**. Dies zeigt, das die "lokale Version der App festgestellt hat, dass auf den Webserver eine neue Version installiert ist. Dies funktioniert analog im Chrome auf dem Android-Smartphone.

Next steps

Erzeugen einer "Native Smartphone App". Für Android gibt es hier **Apache cordova**