AngularJS

AngularJS?

AngularJS!

- Clientseitiges Javascript Framework
- Entwicklung von Single-Page-Webapps
- MVM: Model-View-ViewModel
- Typescript
  - Klassen, Interfaces, Vererbung, Generics
  - Modularisierung
  - Kompatibel zu Plain-Javascript
  - Wird zu Plain-Javascript kompiliert
- Dependency Injection

Architektur

- https://angular.io/docs/ts/latest/guide/architecture.html
- `Component`
  - Bilden Teile der UI ab
  - Angular Applikation = Menge von Components
  - Definieren Aussehen und Verhalten
- `Module`
  - Jede Angular-App hat mindestens ein Modul: `AppModule` (per Konvention)
  - *Modules consolidate components, directives and pipes into cohesive blocks of functionality [...] Modules can also add services*
  - Binden Components: Eine Component gehoert immer zu einem Module
  - Bieten Services an
- `Service`
  - Services koennen injiziert werden
  - Stehen in der gesamten App zur Verfuegung

Entwicklungsumgebung

Einrichten einer Entwicklungsumgebung

- NodeJS

- NodeJS-Plugins:
  - `lodash` : Javascript-Utilities (Vergleichbar mit Apache Commons)
  - `gulp` : Automatisierung sich wiederholender Tasks (Bundling, Refreshing, Running, ...)
  - `angular-cli`

```
npm init .
npm install --save lodash
npm install --save-dev gulp
npm install
npm install -g angular-cli
```

- `init` erstellt initiale `package.json`
- `--save` speichert Abhaengigkeit und schreibt sie zusaetzlich in `package.json`
- `-g` = *global*: Eine Abhaengigkeit wird systemweit installiert

Angular-CLI

- NodeJS-Plugin
- Hilfe bei Erstellung, Entwicklung, Deployment
- "Hello World"-Angular-Applikation ohne angular-cli: https://angular.io/docs/ts/latest/quickstart.html
- Konsolenbefehl: `ng`

Erstellen eines Angular-Projekts

Erstellen eines Projekts

```
ng help
ng new APPLICATIONNAME
cd APPLICATIONNAME
ng serve
```

Dateistruktur

- Komplex!
  - Viele unterschiedliche Technologien
  - Viele Konfigurationsdateien
  - Wer versteht das?!

```
.
|   angular-cli.json
|   karma.conf.js
```

```
|   package.json
|   protractor.conf.js
|   README.md
|   tslint.json
|
+---e2e
|       app.e2e-spec.ts
|       app.po.ts
|       tsconfig.json
|
+---node_modules
|
...
|
\---src
    |   favicon.ico
    |   index.html
    |   main.ts
    |   polyfills.ts
    |   styles.css
    |   test.ts
    |   tsconfig.json
    |   typings.d.ts
    |
    +---app
    |       app.component.css
    |       app.component.html
    |       app.component.spec.ts
    |       app.component.ts
    |       app.module.ts
    |       index.ts
    |
    +---assets
    |
    \---environments
            environment.prod.ts
            environment.ts
```

- Dank angular-cli muss man das nicht verstehen!

Entwicklung #1: Components und Modules

Angular Applikation

```
\---src
    |   index.html
    |   main.ts
    |
```

```
    +---app
    |       app.component.css
    |       app.component.html
    |       app.component.ts
    |       app.module.ts
```

`app.component.ts`

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

- `@Component` -Decorator
  - Metadaten: Markiert eine Klasse als "Angular Component"
  - Beschreibt, wie/wann/wo eine Klasse prozessiert, instanziiert und verwendet wird
  - Component-Klasse wird exportiert
- `selector` : Element im DOM, das durch das Template ersetzt werden soll
- `templateUrl` : Angabe des Templates
- `styleUrl` : Angabe des Stylesheets fuer das Template

`app.module.ts`

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

- `@NgModule`
- `declarations` : Definiert eine Component als Member eines Moduls
- `imports` : Exportierte Deklarationen (z.B. Services) aus anderen Modulen werden importiert
- `providers` : Definiert fuer Dependency Injection verwendbare Services
- `bootstrap` : Angegebene Components werden in den DOM eingehangen, wenn das Modul beim

Startup erstellt wird

Component, Module, Class erzeugen

```
ng generate module MODULENAME
```

- Erzeugt:
  - `MODULENAME.module.ts`
  - `MODULENAME.component.ts`
  - `MODULENAME.component.html`
  - `MODULENAME.component.css`

```
ng generate component COMPONENTNAME
```

- Erweitert: `app.module.ts`
- Erzeugt (in eigenem Unterordner):
  - `MODULENAME.component.ts`
  - `MODULENAME.component.html`
  - `MODULENAME.component.css`

```
ng generate class CLASSNAME
```

- Erzeugt: `CLASSNAME.ts`

Entwicklung #2: UI

Databinding

- One-way: Component => UI
  - `{{VARIABLE}}`

    ```
    <p>{{werIstSchuld}} ist schuld!</p>
    ```

  - `werIstSchuld` ist ein Klassenmember von *.component.ts
- Two-Way: Component <=> UI
  - `[(ngModel)]='VARIABLE'`

    ```
    <textarea [(ngModel)]='werIstSchuld'></textarea>
    ```

- `werIstSchuld` ist ein Klassenmember von *.component.ts

Directives

- `*ngIf`

```
*ngIf="list > 5"
```

- `*ngSwitch`
- `*ngFor`

```
*ngFor="let element of list"
```

- `(click)`

```
(click)="onNewElementClick()"
```

- But wait, there is more! https://angular.io/docs/ts/latest/api/#!?type=directive

Routing

- Mapping zwischen URL <> Component
- Routing-Module anlegen:

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

const routes: Routes = [
    { path: 'main', component: MainComponent },
    { path: 'settings', component: SettingsComponent }
];

@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule],
    providers: []
})
export class AppRoutingModule { }
```

- Anpassen der `app.component.html`:

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

- Verwendung:

```
<a routerLink="/main">Show Main</a>
```

Entwicklung #3: Services

Erzeugung eines Services

```
ng generate service SERVICENAME
```

- Erzeugt: `SERVICENAME.service.ts`
- `@Injectable()`-Decorator
- Muss *provided* werden: @NgModule.providers

Entwicklung #4: Backend

RxJS

- https://github.com/Reactive-Extensions/RxJS
- *asynchronous and event-based programs using observables*
- Reaktive Programmierung
- Asynchrone Antworten
- Aufruf von Webschnittstellen

Beispiel

```
import { Http, Headers, Response } from '@angular/http';
import { Observable } from 'rxjs';
import 'rxjs/Rx';

httpService: Http;

public addTodo(todo: Todo): Observable<boolean> {
    let body = JSON.stringify(todo);
    let headers = new Headers();
    headers.append("Content-Type", "application/json");
    return this.httpService.post('http://localhost:3000/todos', body, { headers: heade
        .map(resp => resp.json() as boolean);

}

public getTodos(): Observable<Todo[]> {
    return this.httpService.get('http://localhost:3000/todos')
```

```
        .map(resp => resp.json() as Todo[]);
}
```

Deployment

via angular-cli

```
ng build
```

- Optimiert, Minifiziert und Packt
- Ausgabe ins `dist/` Verzeichnis
  - `index.html`
  - `main.bundle.js` : Alle Abhaengigkeiten minifiziert und in einem js-File gepackt
  - `styles.bundle.js` : Alle Styles minifiziert und in einem js-File gepackt
  - `inline.bundle.js`
- Kann auf jedem beliebigen Webserver deployed werden!