Workshop: Angular 2

Lise K. Haugen
Viktor Johansson

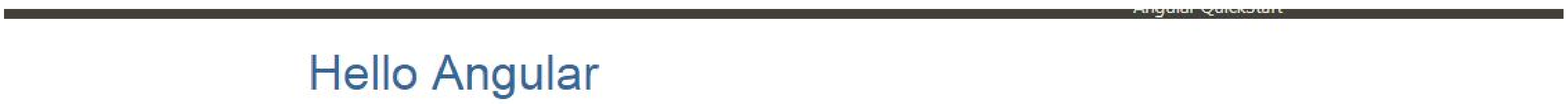
Agenda

1. **Set up workspace**
2. **Angular 2 explanations and tasks**
3. **Play around with the app and api**

Setup workspace

1. Clone GitHub repository: **git clone https://github.com/lisekh/ng2Workshop.git**
2. Download Visual Studio Code
3. Download NodeJS
4. Open command prompt in the project folder and run:
   a. **npm install -g npm@5.3.0**
   b. **npm install**
   c. **npm start**

Your browser should open and display:


Angular QuickStart

Hello Angular

What is Angular 2 ?

Framework for creating JavaScript applications

Single-page applications

Built with Typescript

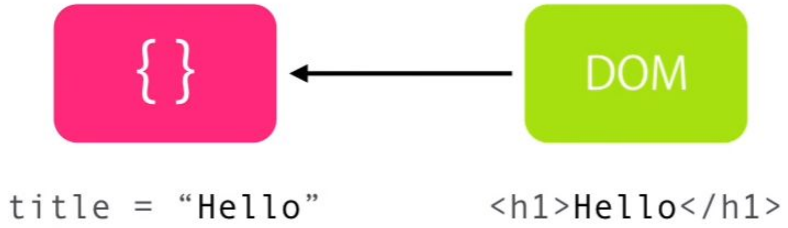
DOM independent

Methods are decoupled from the DOM.

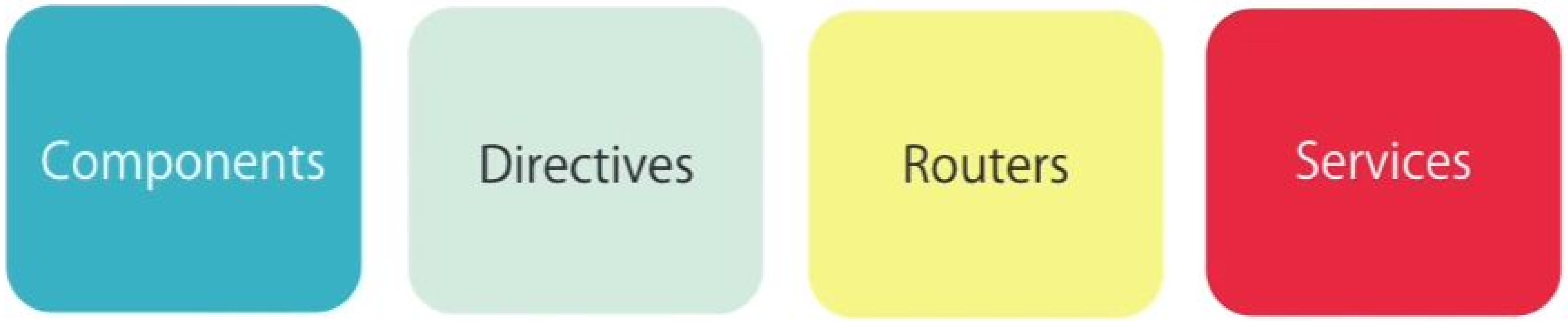
JS and JQuery require reference to DOM elements.

Angular uses “binding”.

View is bound to properties and methods inside the component.



Main building blocks



Components

Components
Directives
Routers
Services

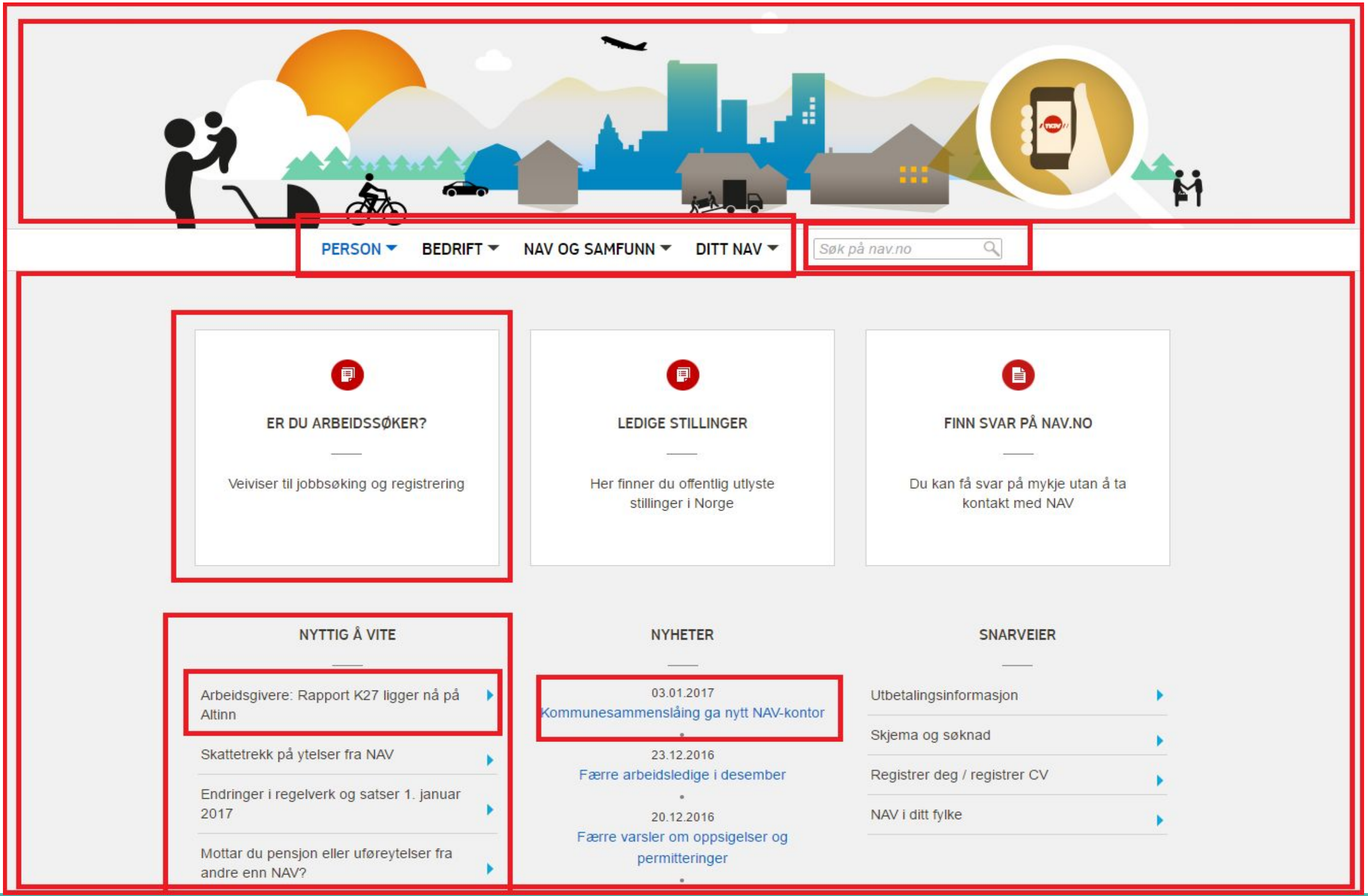
Typescript file

Data, view template, behaviour

Minimum one component - “Root component”

Re-usable

What can be a component?



PERSON BEDRIFT NAV OG SAMFUNN DITT NAV

Søk på nav.no

ER DU ARBEIDSSØKER?

Veiviser til jobbsøking og registrering

LEDIGE STILLINGER

Her finner du offentlig utlyste stillinger i Norge

FINN SVAR PÅ NAV.NO

Du kan få svar på mykje utan å ta kontakt med NAV

NYTTIG Å VITE

- Arbeidsgivere: Rapport K27 ligger nå på Altinn
- Skattetrekk på ytelser fra NAV
- Endringer i regelverk og satser 1. januar 2017
- Mottar du pensjon eller uføreytelser fra andre enn NAV?

NYHETER

03.01.2017
Kommunesammenslåing ga nytt NAV-kontor

23.12.2016
Færre arbeidsledige i desember

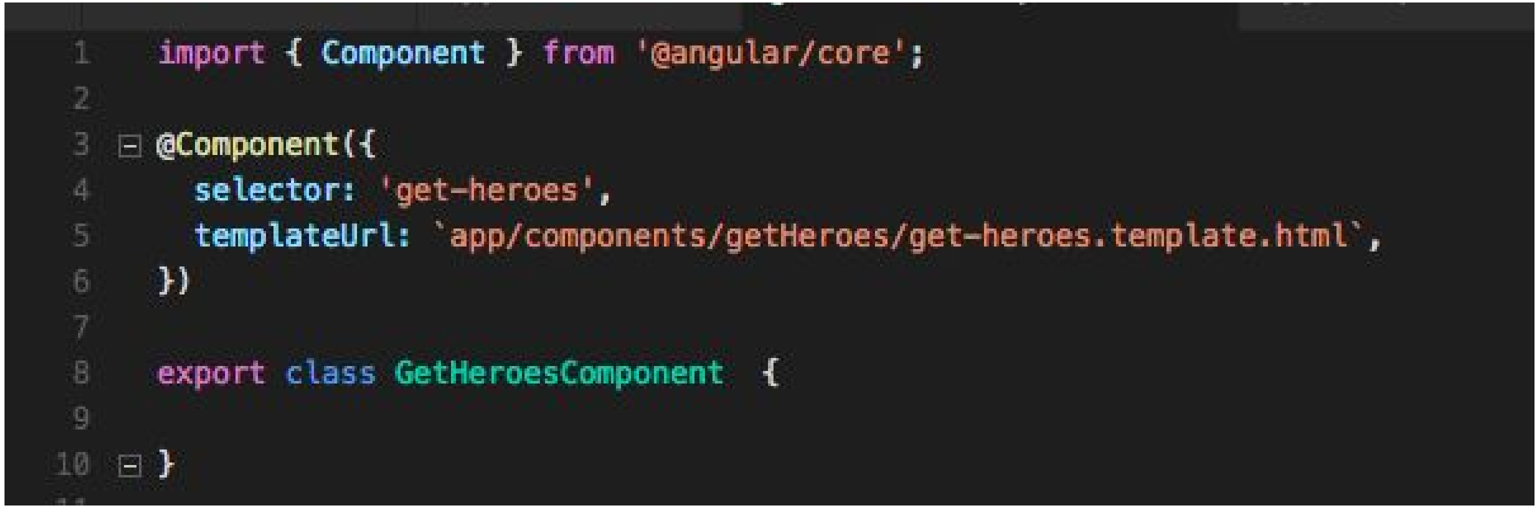
20.12.2016
Færre varsler om oppsigelser og permitteringer

SNARVEIER

- Utbetalingsinformasjon
- Skjema og søknad
- Registrer deg / registrer CV
- NAV i ditt fylke

Task 0: **Create a component class**

1. Create this folder structure inside **app** folder:
   - **components**
     - **getHeroes**
2. Inside **getHeroes** folder create a file called: **get-heroes.component.ts**



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

@Component({
  selector: 'get-heroes',
  templateUrl: `app/components/getHeroes/get-heroes.template.html`,
})

export class GetHeroesComponent  {

}
```

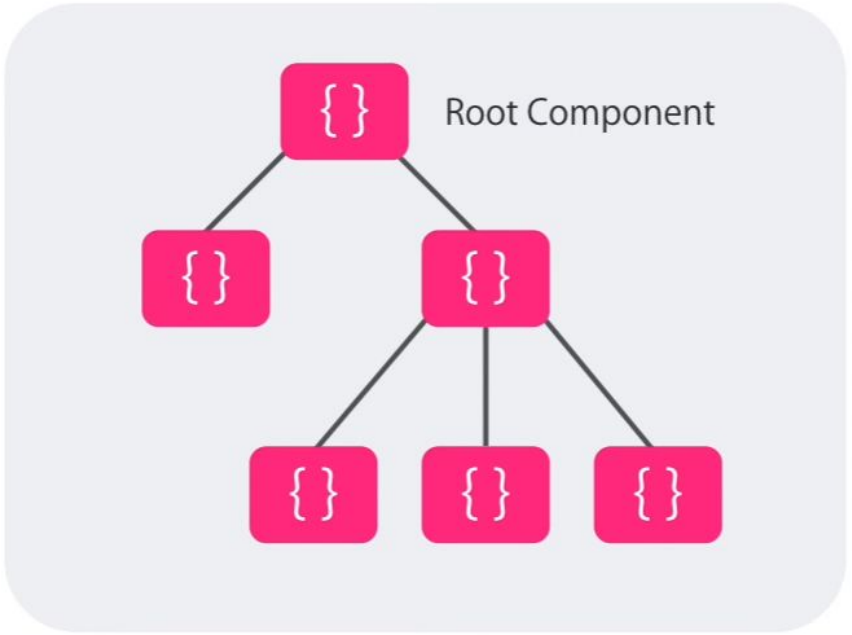
Task 1: **Create a component template**

1. Create a file inside **getHeroes** folder called: **get-heroes.template.html**
2. Put this content in file: **<button class="btn btn-default" type="button" style="width:100%;">Get Heroes</button>**

Component hierarchy

Root component can have many child components

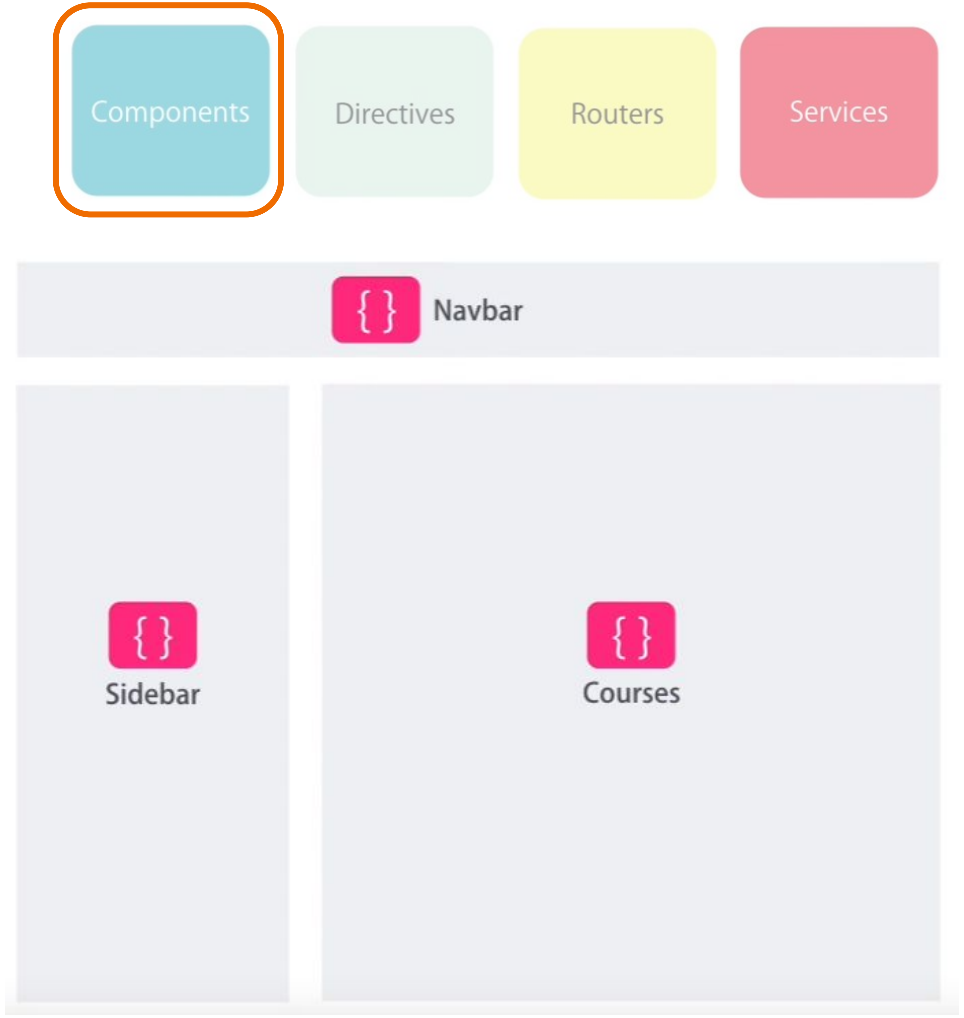
Child components can also have many children



Traditional setup

Three components:

1. Navbar
2. Sidebar
3. Main area
   a. Ex. to display courses



Task 2: **Create a sidebar component**

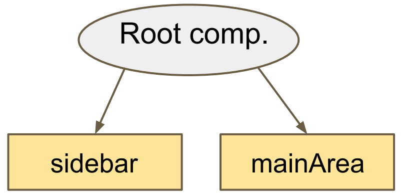
1. Create a **sidebar** component
2. Template content: **<div class="col-md-2" style="margin-top:20px;padding:2em; background-color:gray;">Sidebar area</div>**
3. Copy the component class from get-heroes.component.ts and rename to:
    a. Filename: **sidebar.component.ts**
    b. Class name: **SidebarComponent**
    c. Selector: **‘sidebar’**

Task 2: **Create a mainArea component**

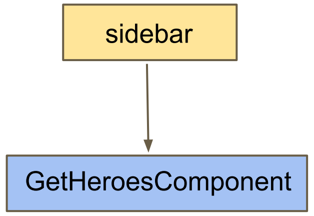
1. Create a **mainArea** component
2. Template content: **<div class="col-md-10" style="margin-top:20px;padding:2em; background-color:lightgray;">Main area</div>**
3. Copy the component class from get-heroes.component.ts and rename to:
   a. Filename: **main-area.component.ts**
   b. Class name: **MainAreaComponent**
   c. Selector: **‘main-area’**

Task 3: **Arrange the components in the hierarchy**

1. Arrange so that **sidebar** and **mainArea** are *child components* of the root component (app.component.ts)



2. Arrange so that **GetHeroesComponent** is a *child component* of **SidebarComponent**

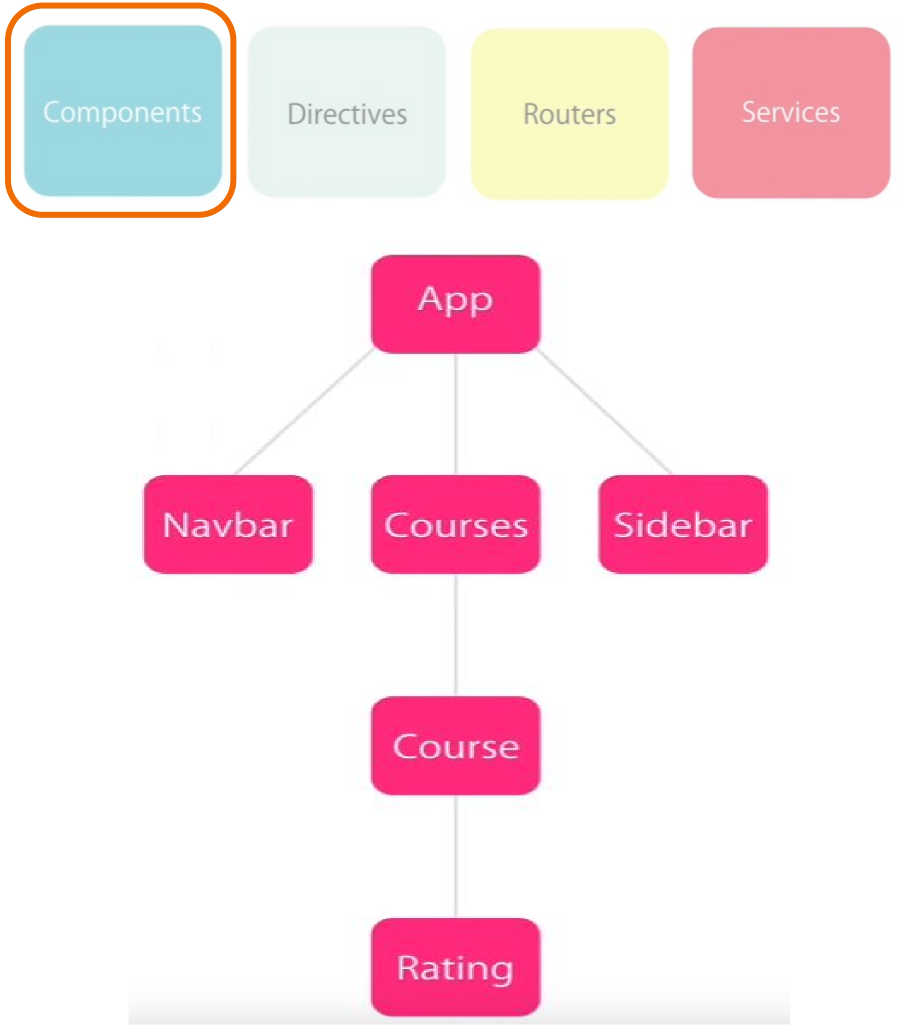


Example hierarchy

Nesting components

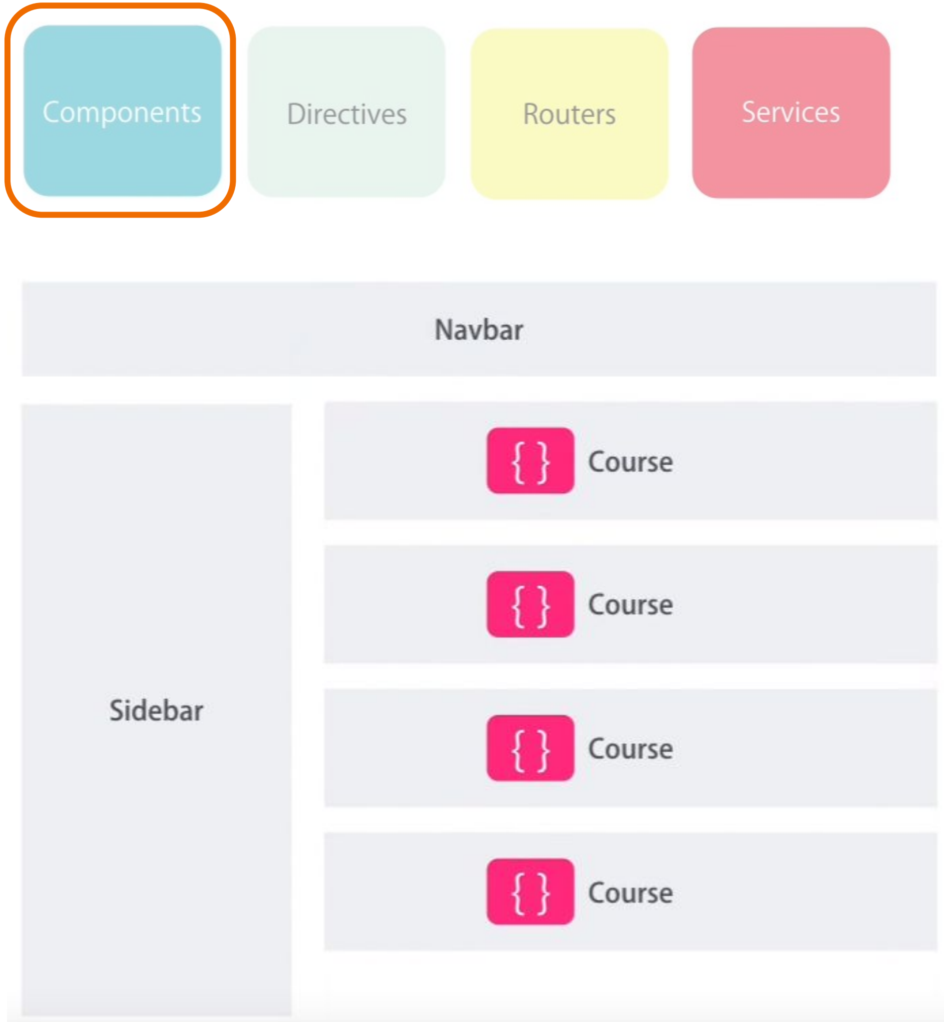
Reuse components

Root component



Reuse components

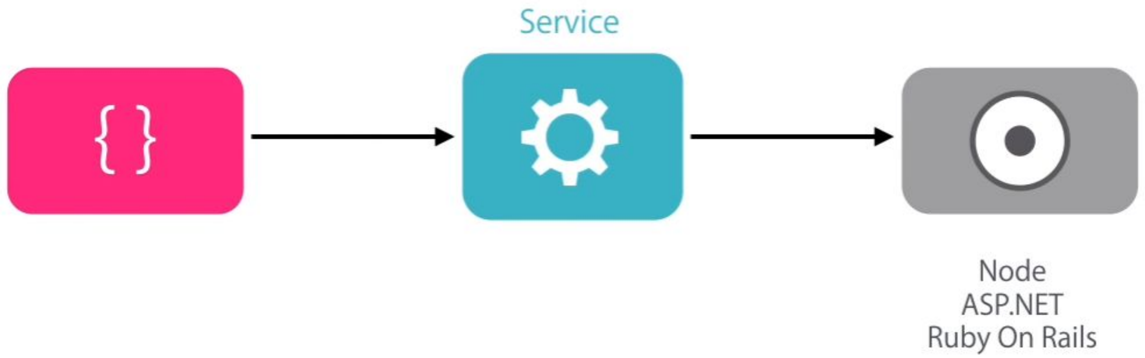
Inside “Courses” component we display many courses by using the the same “Course” component



Services

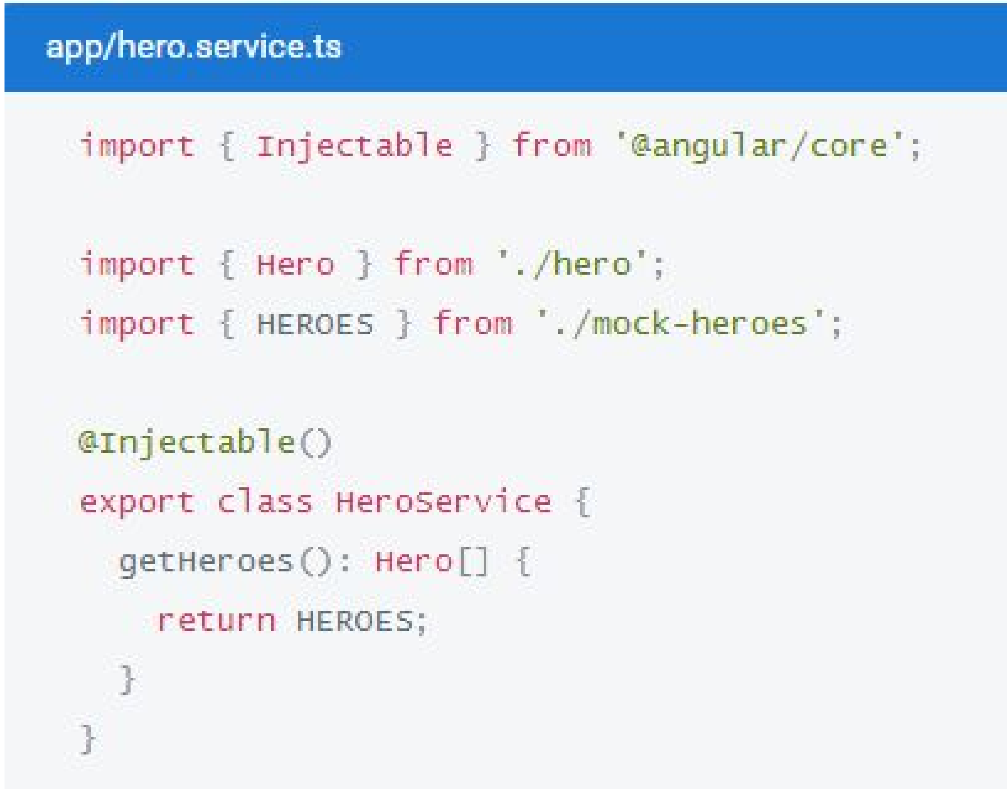
All logic that is not related to user interaction with the UI.

Handles backend communication i.e. REST calls and logging.



Example service

- Import Injectable
- Decorate class with @Injectable()
- Export the class
- Create a method which can be called by an component



app/hero.service.ts

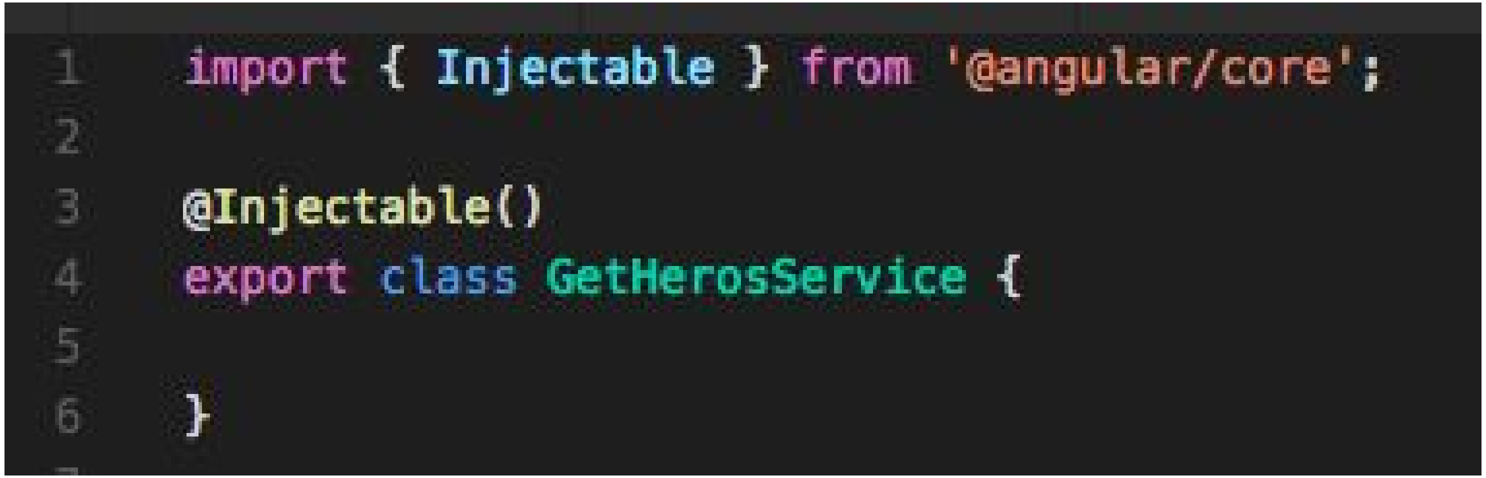
```
import { Injectable } from '@angular/core';

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

@Injectable()
export class HeroService {
  getHeroes(): Hero[] {
    return HEROES;
  }
}
```

Task 4: **Create a GetHeroesService class**

1. Create a new folder inside app folder named **services**
2. Create file: **get-heroes.service.ts**
3. Make the service available for all future components



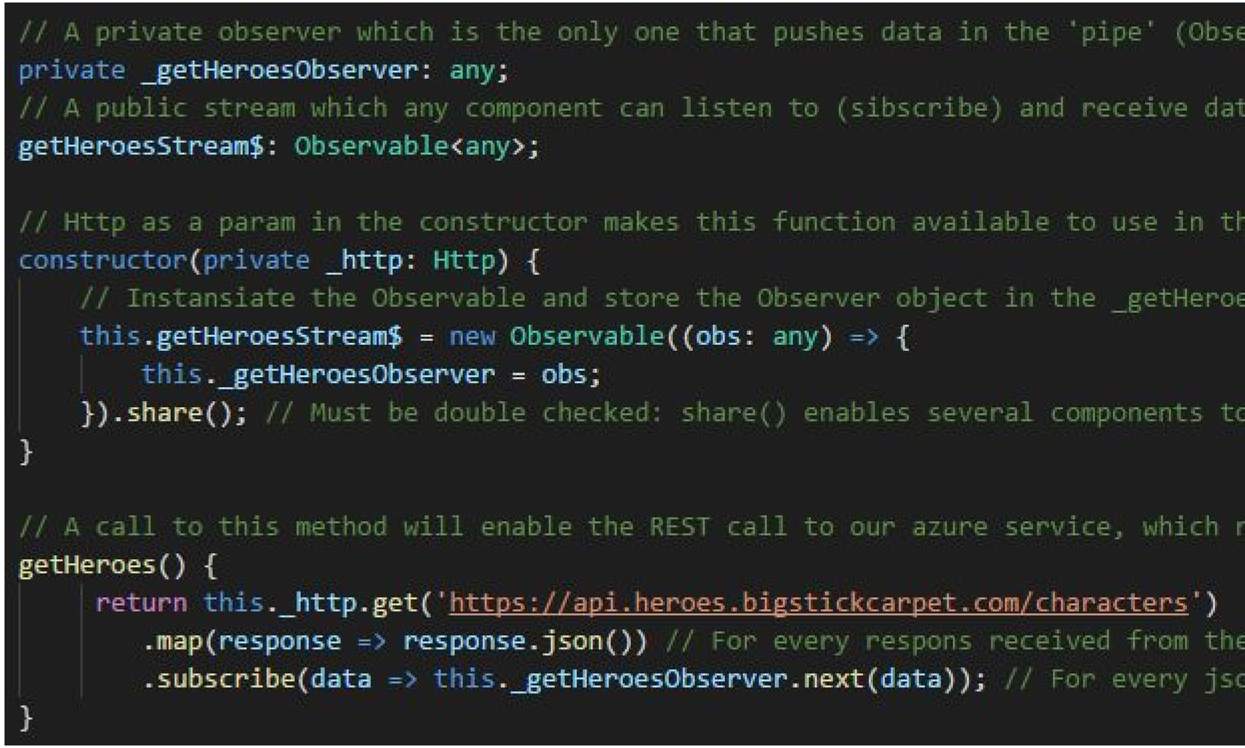
```
import { Injectable } from '@angular/core';

@Injectable()
export class GetHerosService {

}
```

Task 5: **Create method calling a REST service**

1. Inside the **GetHeroesService** create a method which calls the REST service:



```
// A private observer which is the only one that pushes data in the 'pipe' (Obse
private _getHeroesObserver: any;
// A public stream which any component can listen to (sibscribe) and receive dat
getHeroesStream$: Observable<any>;

// Http as a param in the constructor makes this function available to use in th
constructor(private _http: Http) {
    // Instansiate the Observable and store the Observer object in the _getHeroe
    this.getHeroesStream$ = new Observable((obs: any) => {
        this._getHeroesObserver = obs;
    }).share(); // Must be double checked: share() enables several components to
}

// A call to this method will enable the REST call to our azure service, which r
getHeroes() {
    return this._http.get('https://api.heroes.bigstickcarpet.com/characters')
        .map(response => response.json()) // For every respons received from the
        .subscribe(data => this._getHeroesObserver.next(data)); // For every jso
}
```

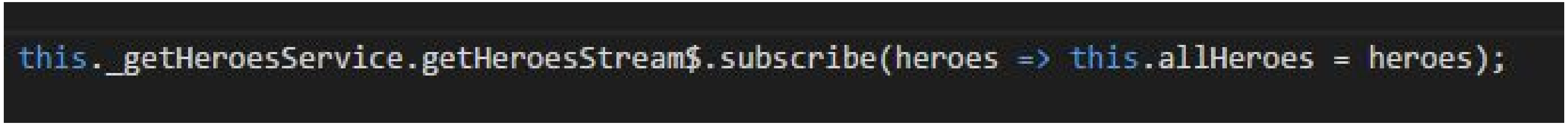
Task 5.5: **Using services in components**

1. Import the service class in the component
   a. **import {GetHeroesService} from '../../services/get-heroes.services';**

2. Add the service as a parameter to the constructor

Task 6: **Listen to the data stream (Observable)**

1. Inside the **GetHeroesComponents**' constructor start listening to the data stream when the component is initialized:



```
this._getHeroesService.getHeroesStream$.subscribe(heroes => this.allHeroes = heroes);
```

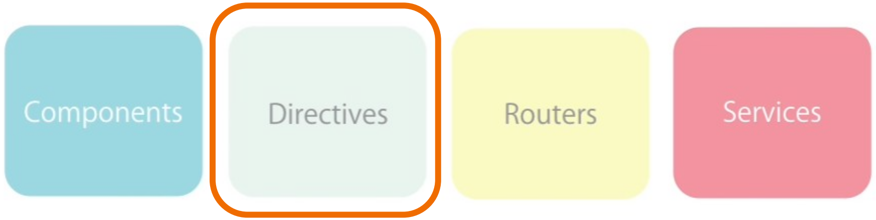
Promise vs Observable

Promise

- A Promise handles a **single event,** when an async event completes or fails
- A Promise can <u>not</u> be cancelled

Observable

- For managing async data flows
- It's like a stream which allows to pass zero or more events where the callback is called for each event
- Provides more features than Promises (has many operators)
- Can handle 0, 1 or multiple events
- Cancelable



Directives

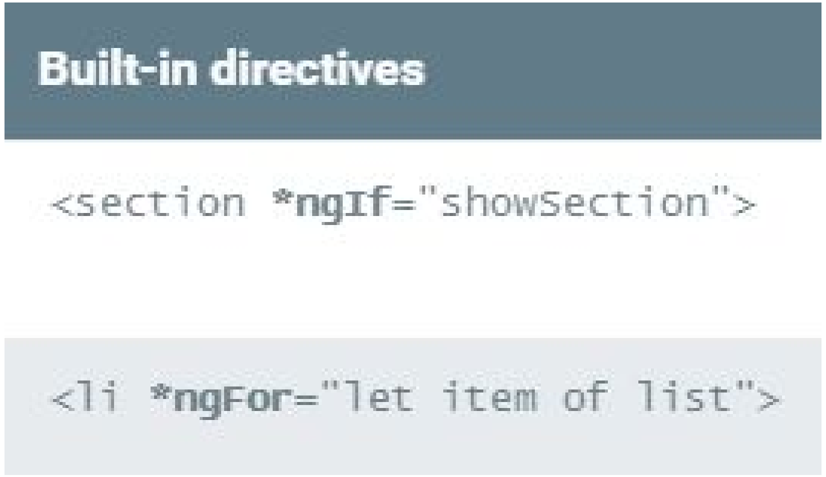
For changing the DOM and extending its behaviour

Many built-in directives, ex.:

- Adding classes
- Repeating classes

Create your own:

```
<input autoGrow />
```



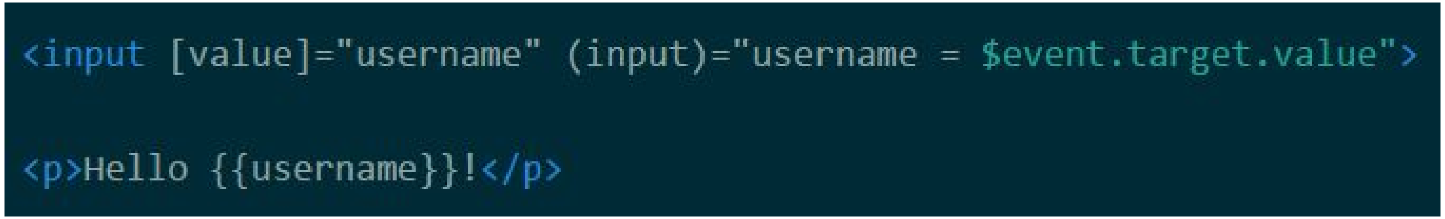
Built-in directives

```
<section *ngIf="showSection">
```

```
<li *ngFor="let item of list">
```

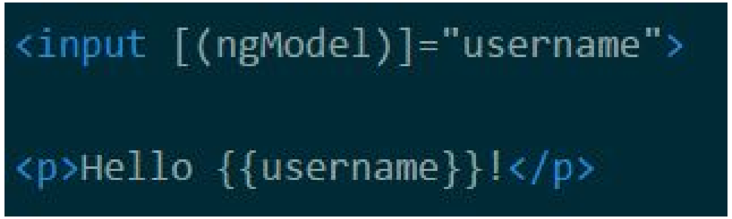
Data binding

- **{{ … }}** equals **rendering**
- **[…]** equals **input**
- **(…)** equals **output**
- **[(…)]** equals **input/output (two-way data binding)**



```
<input [value]="username" (input)="username = $event.target.value">

<p>Hello {{username}}!</p>
```

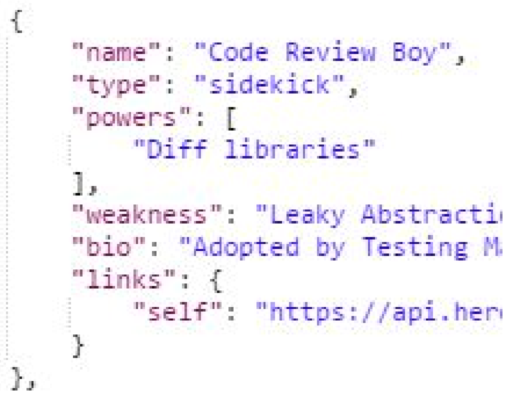


```
<input [(ngModel)]="username">

<p>Hello {{username}}!</p>
```

Task 7: **Calling the service and using a directive to display heroes**

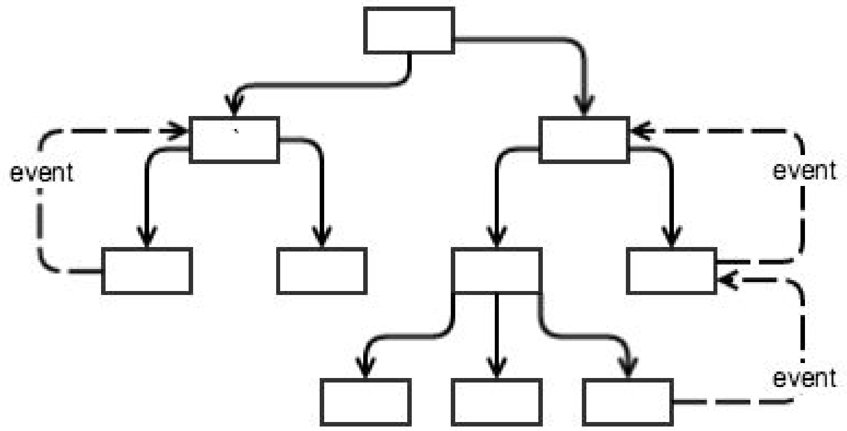
1. Adding an action to the **Get Heroes** button: **(click)=”getHeros();”**
2. Displaying the data received from the **GetHeroesService** with the directive ***ngFor**
   a. ***ngFor=”let hero of allHeroes”**



```
{
    "name": "Code Review Boy",
    "type": "sidekick",
    "powers": [
        "Diff libraries"
    ],
    "weakness": "Leaky Abstracti
    "bio": "Adopted by Testing M
    "links": {
        "self": "https://api.her
    }
},
```

Data sharing between components

- Sending the data between components
  - “Down stream”
    - **@Input**
  - “Up stream”
    - **@Output (EventEmitter)**



Component lifecycles

Directive and component change detection and lifecycle hooks	(implemented as class methods)
`constructor(myService: MyService, ...) { ... }`	Called before any other lifecycle hook. Use it to inject dependencies, but avoid any serious work here.
`ngOnChanges(changeRecord) { ... }`	Called after every change to input properties and before processing content or child views.
`ngOnInit() { ... }`	Called after the constructor, initializing input properties, and the first call to `ngOnChanges` .
`ngDoCheck() { ... }`	Called every time that the input properties of a component or a directive are checked. Use it to extend change detection by performing a custom check.
`ngAfterContentInit() { ... }`	Called after `ngOnInit` when the component's or directive's content has been initialized.
`ngAfterContentChecked() { ... }`	Called after every check of the component's or directive's content.
`ngAfterViewInit() { ... }`	Called after `ngAfterContentInit` when the component's view has been initialized. Applies to components only.
`ngAfterViewChecked() { ... }`	Called after every check of the component's view. Applies to components only.
`ngOnDestroy() { ... }`	Called once, before the instance is destroyed.

Routers

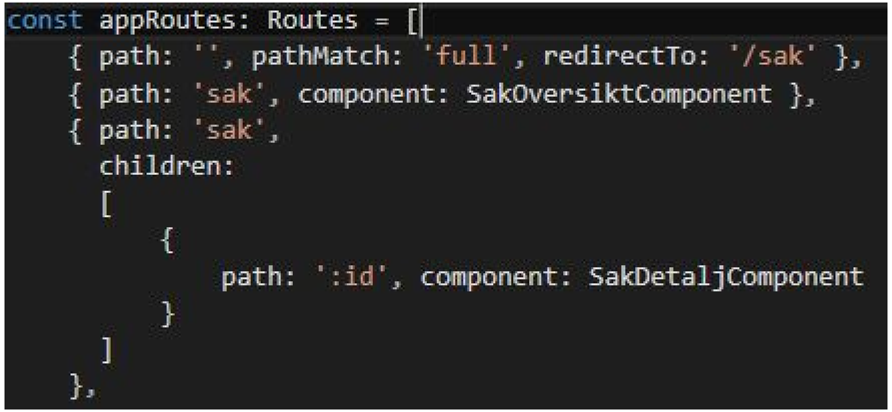
Responsible for all navigation

Located in its own typescript file (eg. app.routing.ts)

Based on URL the router will display the relating components

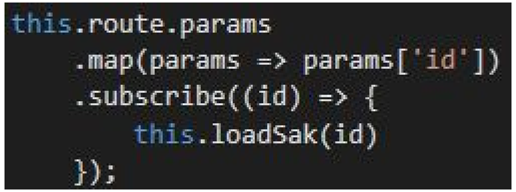
Routing

Set routes in a separate file app.routing:



```
const appRoutes: Routes = [
    { path: '', pathMatch: 'full', redirectTo: '/sak' },
    { path: 'sak', component: SakOversiktComponent },
    { path: 'sak',
      children:
      [
          {
              path: ':id', component: SakDetaljComponent
          }
      ]
    },
```

In case of dynamic url (IDs), subscribe on url params on component init:



```
this.route.params
    .map(params => params['id'])
    .subscribe((id) => {
        this.loadSak(id)
    });
```

More tasks

1. Create a component for creating a **new hero**
   a. A component with a form template
   b. A service for posting the form to the database
2. Create a component for **updating a hero**
   a. Create a new component or reuse the “NewHero” component
   b. Create a service for updating the hero

API: https://documenter.getpostman.com/view/220187/super-tech-heroes-api/77cf6KB

More info...

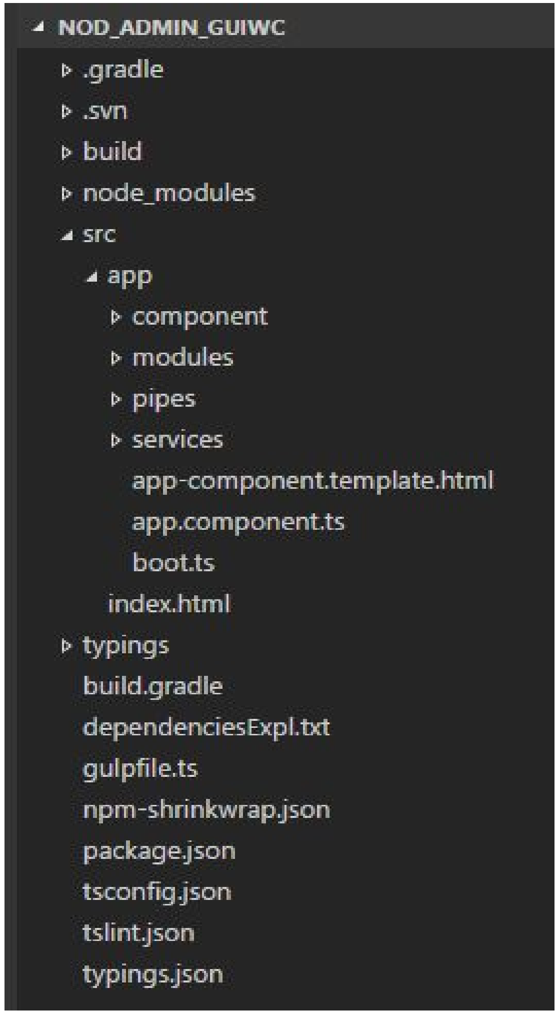
Package structure

"App" folder - all application related files.

- Components, services, modules, pipes.
- Root component

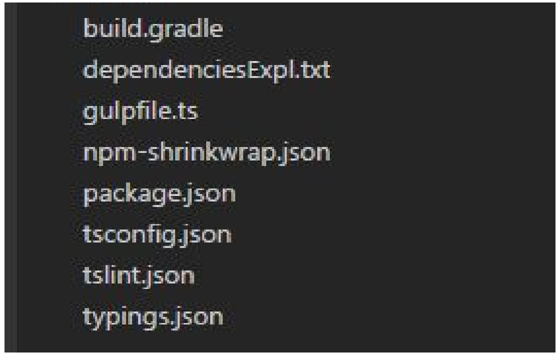
Index.html

Configuration files



```
◢ NOD_ADMIN_GUIWC
  ▷ .gradle
  ▷ .svn
  ▷ build
  ▷ node_modules
  ◢ src
    ◢ app
      ▷ component
      ▷ modules
      ▷ pipes
      ▷ services
        app-component.template.html
        app.component.ts
        boot.ts
      index.html
  ▷ typings
    build.gradle
    dependenciesExpl.txt
    gulpfile.ts
    npm-shrinkwrap.json
    package.json
    tsconfig.json
    tslint.json
    typings.json
```

Configuration files



build.gradle
dependenciesExpl.txt
gulpfile.ts
npm-shrinkwrap.json
package.json
tsconfig.json
tslint.json
typings.json

Package.json

- App name, dependencies, script commands

Tsconfig.json

- Config file for the typescript compiler

Typings.json

- When using external javascript libraries in Typescript, must import a Typescript definition file
- Static type checking and intellisense

Why Typescript

Angular 2 is built using Typescript.

Classes, data types, interfaces, access modifiers, intellisense, compile time checking.

“Strictly typed”.

Using JavaScript or DART as language is also possible.

Compiling Typescript

Use commands defined in package.json.

Starts typescript compiler in watch mode.

.js and .map file is created.

Map file is used for debugging.