Trivadis AG
Sägereistrasse 29
CH-8152 Glattbrugg
training@trivadis.com

Autoren:
Thomas Gassmann,
Thomas Bandixen,
Thomas Claudius Huber

Advanced Angular Training

Vers. 1.0 / Februar 2018

Course Content

00_Course_Introduction 3
01_TypeScript_Angular_WrapUp 9
02_Directives 21
03_Pipes 30
04_Modules 36
05_Routing 45
06_Http 60
07_TemplateDrivenForms 66
08_ReactiveForms 74
09_UnitTesting 85
10_e2eTesting 99
11_Redux 106
12_Deployment 126
13_WhatsNew 137
14_Course_Outro 141

00_Course_Introduction

Why Angular

- Angular is a powerful Single Page Application (SPA) framework
  - Invented by Google
  - built with TypeScript
- TypeScript brings the power of typed languages like C# / Java to the web-world
  - great to build mid- and large-sized applications
- With Web-technologies you can build apps for web, mobile and desktop

2 2/25/2018 Advanced Angular Training

trivadis
makes IT easier.

Your Trainer

- Thomas Claudius Huber (@thomasclaudiush)
  www.thomasclaudiushuber.com
  thomas.huber@trivadis.com

- Thomas Bandixen (@tbandixen)
  thomas.bandixen@trivadis.com

- Thomas Gassmann (@gassmannT)
  thomas.gassmann@trivadis.com

trivadis makes IT easier.

Course Hours

- Official course hours
  - 09:00 – 10:30
  - 10:45 – 12:00
  - 13:00 – 14:30
  - 15:00 – 16:10
- If you need a break in between, feel free to ask
- Lunch today: Who is going to join?

trivadis makes IT easier.

Course Planning

- **Day 1: Angular**
  - TypeScript und Angular Wrap-Up
  - Directives
  - Pipes
  - Modules
  - Routing
  - Http
  - Template-Driven Forms

trivadis makes IT easier.

Course Planning

- **Day 2: Angular**
  - Reactive-Forms
  - Testing
  - Redux
  - Deployment
  - What is new in Version …

trivadis makes IT easier.

Introduce yourself

- name and function
- experience with Angular / TypeScript / JavaScript
- expectations for this course

trivadis makes IT easier.

Your machine

- Visual Studio Code
  - Microsoft's powerful cross-platform code editor
- Node.js and NPM
  - powerful server and package manager
- Sample-code:
  - Structured like modules in this course
  - Available on
  - https://github.com/TrivadisCloud/AdvancedAngular
  - https://tinyurl.com/tvd-adv-angular

trivadis makes IT easier.

Your machine

- Install Angular CLI

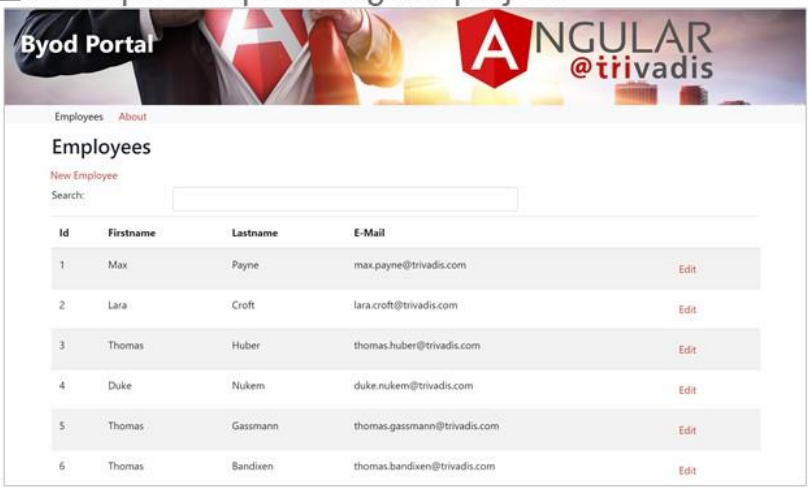
```
npm install -g @angular/cli
```

- VS Code Extension
  - Angular@Trivadis VS Code Essentials
  - https://marketplace.visualstudio.com/items?itemName=trivadis.ngtvd-extensions

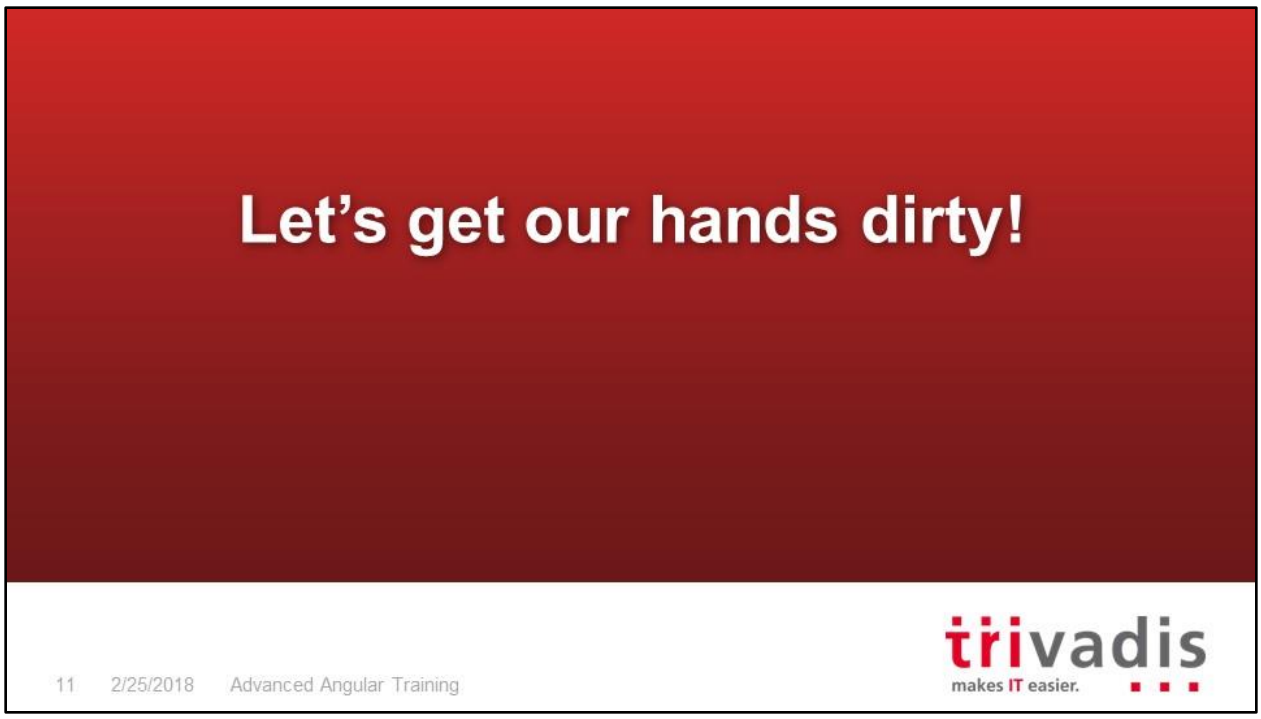
trivadis makes IT easier.

What are we building?

- A simple complete Angular project



trivadis makes IT easier.



01_TypeScript_Angular_WrapUp

TypeScript

- A statically typed superset of JavaScript
  - It «compiles» to plain JavaScript
- Works on any browser, any host, any OS
- Entire project is open source
  - https://github.com/Microsoft/TypeScript
- Future JavaScript-features can be used today
  - TypeScript allows you to use new features and compiles them to older JavaScript-versions

Iterating Arrays: for... of vs. for...in

```
let names:string[] =["Thomas","Sara","Julia"];
```

- **for... of** returns the values

```
for(let name of names){ console.log(name); }
// Output: Thomas, Sara and Julia
```

- **for ... in** returns the keys / indexes

```
for(let name in names){ console.log(name); }
// Output: 0, 1, 2
```

- Note: **for ... in** works not only on arrays, but on any object:
  - Great to iterate properties

var, let and const

- *var* is the classical way to define a variable in JavaScript

```
var name = "Thomas";
```

- *let* and *const* are new types for variable declarations in JavaScript
  – As TypeScript is a superset, they are also available

```
let name: string = "Thomas";
```

- Prefer *let* and *const* over *var*!
  – Why? Let's look at *var* in detail

trivadis makes IT easier.

Var: scoping rules

- You can access var declarations anywhere within their containing function (like function parameters)
  – or within their containing module, namespace, or global scope

```
function getNumber(init)
{
   if(init)
   {
       var x = 9;
   }
   return x;
}
```


trivadis makes IT easier.

Var: scoping rules

- Changing var to let will produce a runtime error in JavaScript
- Changing var to let will produce a compile-time error in TypeScript

```
function getNumber(init)
{
    if(init)
    {
        let x = 9;
    }
    return x; // x is not visible here, as "let" is block-scoped
}
```


Var: scoping rules

Another one! What's the output here?

```
function writeToBody()
{
    for(var i = 0; i<5; i++)
    {
        setTimeout(function() {
            document.write(i+"<br>");
        }, 200);
    }
}
```

```
5
5
5
5
5
```

expected

```
0
1
2
3
4
```

Var: scoping rules

JavaScript workaround: Nest another function expression that is called immediately with a parameter

– the parameter is function-scoped and so captured

```
function writeToBody()
{
    for(var i = 0;i<5;i++)
    {
        (function (i)
        {
            setTimeout(function() {
                document.write(i+"<br>");
            }, 200);
        })(i);
    }
}
```

```
0
1
2
3
4
```


trivadis makes IT easier.

Var: scoping rules

JavaScript workaround: Nest another function expression that is called immediately with a parameter

– the parameter is function-scoped and so captured

```
function writeToBody()
{
    for(var i = 0;i<5;i++)
    {
        (function (val)
        {
            setTimeout(function() {
                document.write(val+"<br>");
            }, 200);
        })(i);
    }
}
```

```
0
1
2
3
4
```


trivadis makes IT easier.

Var: scoping rules

Workarounds are evil. ☺ Just use *let* instead

- this will create a new scope per iteration

```
function writeToBody()
{
    for(let i = 0; i<5; i++)
    {
       setTimeout(function() {
           document.write(i+"<br>");
       }, 200);
    }
}
```

```
0
1
2
3
4
```


trivadis makes IT easier.

let declarations

- Looks similar like a var declaration. Just a different keyword

```
let name: string = "Thomas";
```

- are block-scoped
- are pretty much like declarations in Java / C#

trivadis makes IT easier.

Let declarations

- You can't declare a variable twice in the same scope

```
let name: string = "Thomas";
let name: string = "Bill" // Error
```

- But you can shadow it in another scope

```
let name="Thomas";
{
   let name="Bill";
   console.log(name); // Bill
}
console.log(name); // Thomas
```


Const declarations

- Declared and block-scoped like let

```
const carWheels: number = 4;
```

- Only assignable once:

```
const carWheels: number = 4;
carWheels = 3; // Error: Assignment to const variable
```

- Properties of complex types are still adjustable

```
const car = {wheels: 4,type: "bmw"};
car.wheels = 3; // OK
car.type = "Trike"; // OK
car = {wheels: 2, type: "harley"} // ERROR: Assignment to
const...
```

Summary

- TypeScript is a statically typed superset of JavaScript
  – Any JavaScript is valid TypeScript
- TypeScript transpiles to ES3, ES5 or ES6
- **for... of** returns the values
- As TypeScript is a JavaScript-superset, let and const are available
- Prefer let and const over var

trivadis makes IT easier.

Angular

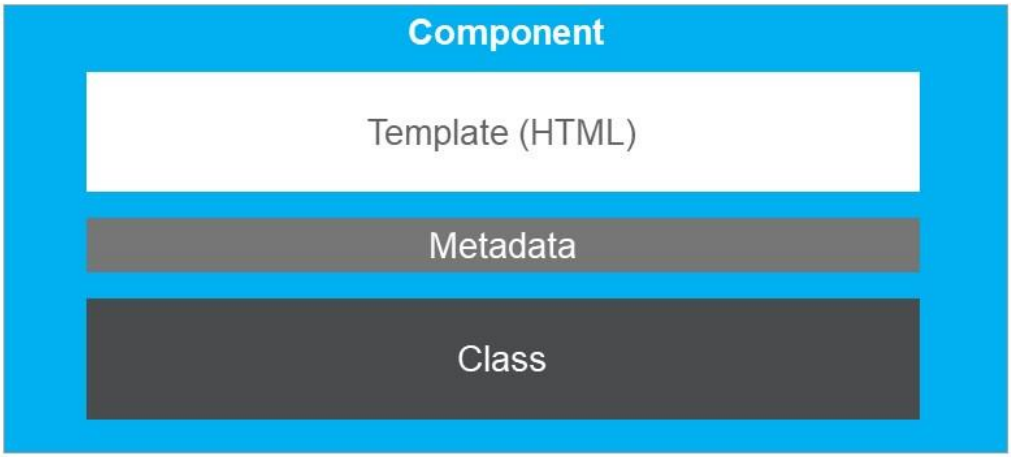
trivadis makes IT easier.

Angular

- Single-Page-Application (SPA) Framework by Google
  - is different to Angular 1.x: No controllers, no $scope
  - Angular is more than a Framework, it's a platform
- Completely rebuilt with TypeScript
- Use your HTML, CSS and TypeScript skills to build apps
  - Instead of TypeScript you could also use plain JavaScript or Dart

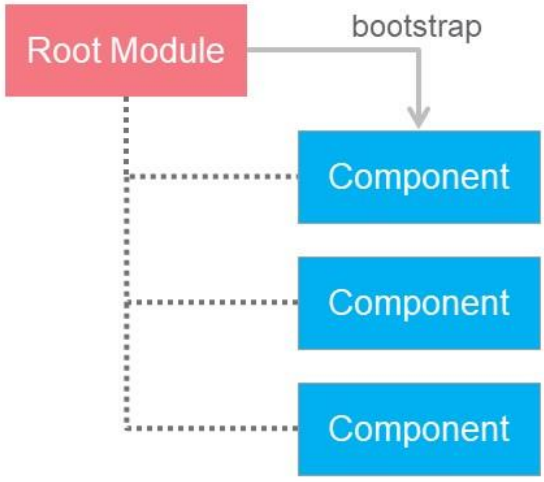
trivadis makes IT easier.

Angular Apps are component-based



trivadis makes IT easier.

Components are structured into Angular-Modules



- Every Angular App has at least one module, the Root Module
- The module can bundle multiple components
- One Component must be bootstrapped for startup
  - only true for the root module

trivadis makes IT easier.

Beside Components an app uses Services

- A service is a class that gets injected into components
  - for example a class that encapsulates access to a REST-Api

trivadis makes IT easier.

Angular CLI

- Install Angular CLI

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

- Generate a new project

```
ng new PROJECT-NAME
```

- Start a project

```
ng serve
```


Advanced Angular CLI commands

- Use SCSS syntax

```
ng new PROJECT-NAME --style=scss | ng set defaults.styleExt scss
```

- Use Service Worker for PWA (Alias: -sw)

```
ng new PROJECT-NAME --service-worker
```

- Build application

```
ng build --prod
```

- Start documentation

```
ng doc
```

Summary

- Angular is component based
- Angular has its own Module-system
- Angular is built with TypeScript
- Angular CLI common tool to start with a new application

trivadis makes IT easier.

02_Directives

What is a Directive?

- A Directive is an Angular Extension for classic HTML
- It can be a custom HTML element or attribute
- Angular components are technically directives

trivadis
makes IT easier.

What kinds of directives do we know?

- Components
  - are referenced by their selector (custom HTML tag)
- Structural Directives
  - are changing the DOM layout by adding/removing DOM elements
- Attribute Directives
  - change the appearance or behavior of an element

trivadis makes IT easier.

Structural directives

- **ngFor** – to generate elements while looping over a collection
- **ngIf** – to show or hide an element with an if-statement
- **ngSwitch** – to show/hide elements based on a condition

trivadis makes IT easier.

About stars (*) and templates

- ngFor, ngIf and ngSwitch use a * for the attribute
  – that's Angular's syntactical sugar
- all these directives add/remove a subtree within a template tag
- the template-tag gets generated behind the scences

trivadis makes IT easier.

About stars (*) and templates

- This is what Angular does with an *ngIf

```
<input *ngIf="lastname.invalid" .../>
```

```
<input template="ngIf:lastname.invalid" .../>
```

```
<ng-template [ngIf]="lastname.invalid">
  <input .../>
</ng-template>
```


trivadis makes IT easier.

How to create an attribute directive

- The most important property of an attribute directive is that it only modifies its own host element

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


@Directive({ selector:"[appSelect]" })
Export class CardHoverDirective(private el: ElementRef) {
   el.nativeElement.style.backgroundColor = "gray";
}
```


Register it in ngModule

- Register all directives separately

```
import { SelectDirective } from './directives/select.directive';

@NgModule({
  imports: [CommonModule],
  exports: [CommonModule,
    SelectDirective],
  declarations: [SelectDirective],
  providers: []
})
export class SharedModule {}
```

Register it in ngModule

- Use ES6 Syntax to register an array of directives

```
import * as fromDirectives from './directives';

@NgModule({
  imports: [CommonModule],
  exports: [CommonModule,
    ...fromDirectives.directives],
  declarations: [...fromDirectives.directives],
  providers: []
})
export class SharedModule {}
```


Selectors

- Selectors is css attribute that identifies a component in a template
- [] make an attribute selector. Angular locates each element in the template that has an attribute name 'appSelect' and applies the logic of this directive to that element

```
@Directive({ selector: '[appSelect]' })
export class SelectDirective {}
```

Types of selectors

- element-name: select by element name
- .class: select by class name
- [attribute]: select by attribute name
- [attribute=value]: select by attribute name and value
- :not(sub_selector): select only if the element does not match the subselector

Directives Input

- The @Input decorator gives you the option to set some property value while applying the directive.

```
@Directive({ selector: '[appSelect]' })
export class SelectDirective {
  @Input() backgroundColor: string;
   constructor(private el: ElementRef, private renderer:
   Renderer){}
```

- Use it in a component

```
<tr *ngFor="let employee of employees$ | async"
          appSelect [backgroundColor]="'gray'">
```

@Hostbinding and @HostListener decorators

- @HostBinding
  - lets you set properties on the element or component that hosts the directive
- @HostListener
  - lets you listen for events on the host element or component.

trivadis makes IT easier.

@HostBinding: possible parameters

- propertyName: references a property of the host with a given property name
- attr.[attributeName]: references an attribute of the host with a given attribute name. Using the null value removes the attribute from the HTML element.
- style.[styleName]: references a property to a style of the HTML element
- class.[className] references a property to a class name of the HTML element.

trivadis makes IT easier.

@HostBinding

```
@Directive({ selector: '[appSelect]' })
export class SelectDirective {

  @HostBinding('class.selected') isSelected = false;

   constructor(private el: ElementRef, private renderer:
        Renderer) {}

   @HostListener('mouseover')
   @HostListener('mouseleave')
   setSelected() {
       this.isSelected = !this.isSelected;
    }
}
```

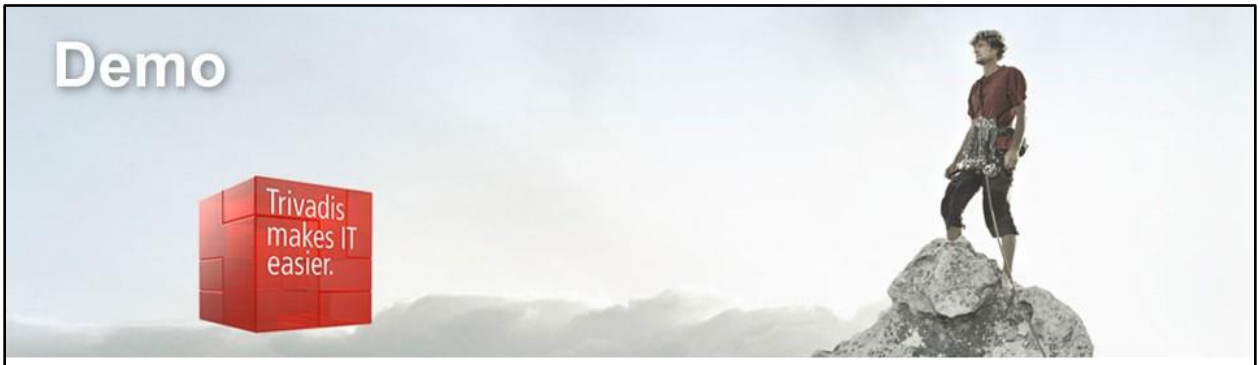

@HostListener

```
@Directive({ selector: '[appSelect]' })
export class SelectDirective {

 constructor(private el: ElementRef, private renderer:
       Renderer) {}
 @Input() backgroundColor: string;

 @HostListener('dblclick')
  doubleClick() {
  this.renderer.setElementStyle(this.el.nativeElement,
       'backgroundColor', this.backgroundColor);
 }
}
```

Summary

- Angular components are directives, but directives with views and templates
- Directives are defined with the @Directive decorator
- Selector is a css attribute which identifies directives within a template
- Use @Hostbinding to bind to the input properties of a host element from within a directive
- User @Hostlistener to listen to the output events

trivadis makes IT easier.



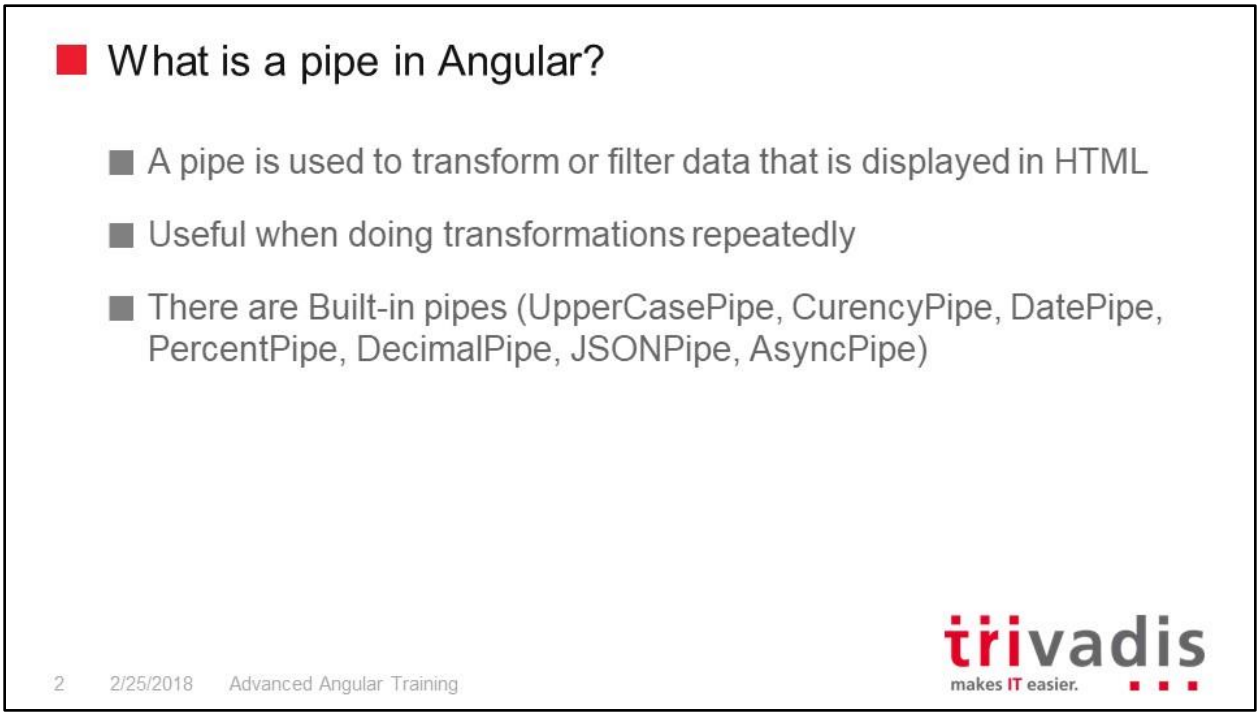
Create a new Attribute Directives (e.g. SelectDirective, create it under shared/directives/)

Add it to the barrel und use it inside the employee-list.component.ts

Add a **`HostBinding`** and **`HostListener`** functionality

trivadis makes IT easier.

03_Pipes



Date pipe explained

```
export class NumberPipeComponent {
        currentDate : number = new Date().getDate();
}
```

```
<p> Today is {{ currentDate| date:"MM/dd/yy" }} </p>
```


trivadis makes IT easier.

Decimal pipe explained

- number_expression | number[:digitInfo[:locale]]
- format: {minIntegerDigists}.{minFractionDigists}-{maxFractionDigits}

```
@Component({
 selector: 'number-pipe',
 template: `<div>
              <!--output '012.63847'-->
              <p>num1 3.2-5: {{num1 | number: '3.2.-5'}}</p>
           </div>`
}
export class NumberPipeComponent {
       num1: number = 12.638467846
}
```


trivadis makes IT easier.

How to create a custom pipe

```
@Pipe({
 name: 'employeeFilter'
})
export class EmployeeFilterPipe implements PipeTransform {

transform(value: Employee[], filterBy: string): Employee[] {
        filterBy = filterBy ? filterBy.toLocaleLowerCase() :
        null;
        return filterBy ? value.filter((e: Employee) =>
        (e.firstname.toLocaleLowerCase() +
        e.lastname.toLocaleLowerCase()).indexOf(filterBy) !== -1)
        : value;
 }
}
```


How to create a custom pipe

```
<input type="search" [(ngModel)]="listFilter" class="form-
control" />

<tr *ngFor="let employee of employees$ | async |
        employeeFilter:listFilter" appSelect>
<td>{{ employee.id }}</td>
<td>{{ employee.firstname }}</td>
<td>{{ employee.lastname }}</td>
<td>{{ employee.email }}</td>

</tr>
```

Pure pipes

- Pure pipes are executed by Angular when a pure change to the input was detected. This can be a primitive input value such as String, Number, Boolean, Symbol or object reference (Array, Function, Object)
- However, changes within objects are ignored. This means changing an object property, modifying an array or chaning input data will not call a pure pipe logic.

Inpure pipes

- Inpure pipes, on the another hand, are raised during every component change detection cycle. They are called more often than pure pipes. For example by every key down or mousedown event.

```
@Pipe({
  name: 'employeefilter',
  pure: false
})
export class EmployeeFilterPipe extends PipeTransform {}
```

Problems with custom FilterPipe and OrderByPipe

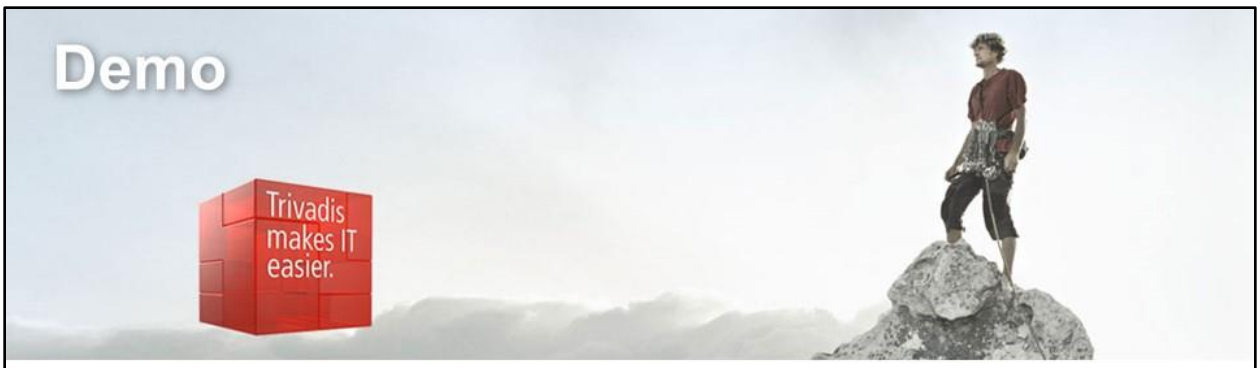
- There are no built in filtering or sorting pipes in Angular on purpose (in contrary to AngularJS). The reason is their poor performance.
- Angular calls impure pipes in every change-detection cycle
- Avoid using them as filtering and sorting are expensive operations

trivadis makes IT easier.

Summary

- Pipes enable you to transform data to be displayed in templates
- Angular comes with many built-in pipes. (Currency, JSON, Decimal, Percent, Date and more…)
- Pure pipes perform much better than inpure pipes a you should consider using inpure pipes

trivadis makes IT easier.



Create a pipe (e.g. FilterEmployeePipe)

Use it inside a component

trivadis
makes IT easier.

04_Modules

What is an Angular module?

- The main purpose of a module is to combine and organize code with similar logical context into one unit
- We can extend our application logic by importing external modules (libraries)
- Modules make it possible to load some funcionality on demand (lazy loading)
- There are 2 types of modules, root and feature modules
- A module can be loaded eagerly when the application starts or lazyly (assynchronously) by the router

2 2/25/2018 Advanced Angular Training

trivadis
makes IT easier.

Root Module

- Every application has at least one module – the root module. It is used to bootstrap your application
- You usually name it AppModule

trivadis makes IT easier.

Feature Modules

- A feature modules purpose is to organize your code and achieve modularity
- Feature modules should only import services from RootModule.
- You should lazy load feature modules whenever possible. Especially for performance reason

trivadis makes IT easier.

Declaration of a module

- The @NgModule decorator defines an Angular Module
- Module configuration:
  - imports
  - exports
  - declarations
  - entryComponents
  - providers
  - bootstrap

trivadis makes IT easier.

Module Bootstraping

- **Imports** array: contains a list of another modules that needs to be imported for your module
- **Exports** array: contains a list of reausable components, directives, pipes in your module that you want to user outside of your module
- **Providers** array: contains a list of services that your app needs
- **Declarations** array: tells Angular which components, directives and pipes (declarables) belong to that module. A declarable can only belong to one module. It is used inside a template
- **Bootstrap** array: Root components that are launched on bootstrapping

trivadis makes IT easier.

Declaration of a module

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

@NgModule({
   imports: [AboutRoutingModule],
   exports: [],
   declarations: [AboutListComponent],
   providers: []
})
export class AboutModule {}
```


trivadis makes IT easier.

Lazy loading modules

- By lazy loading modules Angular creates a child injector for specified providers
- Registering a specified service within a child module can lead to unintended consequences, as more instances of the same service would be created
- This might not be wrong, but sometimes you may need to assure that you get only one instance of a given service e.g. authentication service which delivers the same information across the whole application

trivadis makes IT easier.

Lazy loading modules

- To ensure that a module will be loaded on demand, we replace the component property with the loadChildren property in the route definition and we pass a string instead of a symbol as a path to the module. We define the class of the module as well.

```
const routes: Routes = [
 { path: '', redirectTo: 'eager', pathMatch: 'full' },
 { path: 'eager', component: EagerComponent },
 { path: 'lazy', loadChildren: 'lazy/lazy.module#LazyModule' }
];
```


Dealing with instances

- Loading services multiple times can be avoided by defining a custom static forRoot() or forChild() method that returns a configuration object that defines the services we want to export

```
export class CoreModule {
static forRoot(): ModuleWithProviders {
        return {
        ngModule: CoreModule,
        providers: [MyService]
  };
 }
}
```

Dealing with instances

- To register a feature module in the root application module we call the forRoot() method defined for example in CoreModule.

```
@NgModule({
   imports: [
        CoreModule.forRoot()
   ]
})
```


Dealing with instances

- forRoot() should only be called once in the AppModule. To make sure it is not called multiple times, you can check it inside the constructor

```
export class CoreModule {
  constructor(@Optional() @SkipSelf() parentModule: CoreModule)
{
    if (parentModule) {
      throw new Error(`Core has already been loaded. Import Core
                          modules in the AppModule only.`);
    }
  }
}
```

Shared modules

- Shared modules are meant to share functionality across your application
- All the „dumb“ components and pipes should be implemented as a shared modules
- A shared module should not import and inject services from the root module

trivadis makes IT easier.

How to create a module with Angular CLI

- ng generate module [name].
  Creates a new NgModule with given name.
- ng g module my-module --routing.
  With the routing option, a separate file my-module-routing.module.ts will be created, where module routes can be specified.

trivadis makes IT easier.

Outlook Angular V7

Statement from Igor Minar (Angular Team Lead)

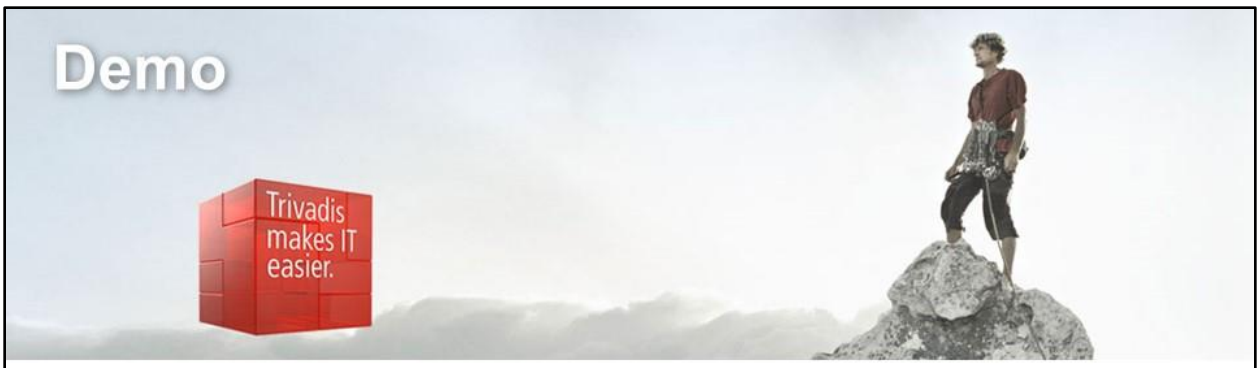
"But we made progress across the board, which unlocked new possibilities, and we now believe that we'll be able to keep the guarantees and features without NgModules in the (near) future. I'm *hoping* in v7."

trivadis makes IT easier.

Summary

- Modules allows you to organize and streamline your code.
- You can put commonly used directives, pipes and components into one module and import just that module
- Modules can be loaded eagerly or on lazyly.
- By calling a custom forRoot() method in the application module you can make sure that services inside this module will be created as singletons and can be used accross other modules

trivadis makes IT easier.



Create a Employee Module

Place all employee components inside this module

Make sure Core Module is only loaded inside AppModule and services are singleton

trivadis
makes IT easier.

05_Routing

What is Routing?

- Angular is a Single Page Application (SPA) framework
- Somehow users need to navigate
- Navigation means Routing the user to different components

trivadis
makes IT easier.

Importing Angular Router

- Setup the Router service provider
  - call the RouterModule's forRoot-method to create a provider

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

@NgModule({
  imports: [BrowserModule, FormsModule,
  RouterModule.forRoot([
  { path: 'languages', component: ProgrammingLanguagesComponent},
  { path: 'persons', component: PersonListComponent }
    ])],
  ...
})
export class AppModule { }
```


Importing Angular Router

- The Router Service can only be imported once. (forRoot-Method)
- Routing inside a feature module is different.
- Import this module before calling forRoot()

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

@NgModule({
  imports: [SharedModule,
    RouterModule.forChild([…])
  ],
  ...
})
export class FeatureModule { }
```


Defining a Routing Module

- Better organization and easier to find
- Separation of concerns
- <NAME>-routing.module.ts

```
const routes: Routes = […];

@NgModule({
   imports: [RouterModule.forChild(routes)],
   exports: [RouterModule],
})
export class EmployeeRoutingModule { }
```


trivadis makes IT easier.

Parameterized Routes: Setting up the route

- **Step 1:** Setting up the route with a parameter

```
const routes: Routes = [
  { path: 'employees', component: EmployeeListComponent },
  { path: 'employees/new', component: EmployeeComponent },
  { path: 'employees/:employeeId', component: EmployeeComponent }
];
```


trivadis makes IT easier.

Parameterized Routes: Navigate to the details

Step 2: Navigate to the details

```
import { Router } from '@angular/router';
...
@Component(...)
export class EmployeeListComponent implements OnInit {
   ...
   constructor(private _service: EmployeeService,
      private _router: Router) { }

   onPersonClick(e: Employee) {
          let link = ['/employee', e.id];
          this._router.navigate(link);
    }
}
```


Parameterized Routes: Get parameter in details

Step 3: Grabbing the parameter

```
export class EmployeeComponent implements OnInit {
  employee$: Observable<Employee>;

  constructor(private route:ActivatedRoute,
    private service: EmployeeService){}

    ngOnInit() {
      this.route.paramMap.subscribe((p) => {
        const eId = +p.get('id');
        this.employee$ = this.service.getEmployee(eId);
      }
    }
}
```

Query Parameters

- Can be important for search url
- We won't define a route for each search field

trivadis makes IT easier.

Query Parameters: Defining

- **Step 1: Define query params**

```
<a [routerLink]="['/employee',  employee.id]"
[queryParams]="{filterBy: listFilter}">Edit </a>
```

```
this.router.navigate(['/employee', id], {
      queryParams: { filterBy: this.listFilter }
});
```


trivadis makes IT easier.

Query Parameters: Retaining query params

Step 2: Add attribute when navigating back

```
<a [routerLink]="['/employees']"
[queryParamsHandling]="preserve">Back</a>
```

```
this.router.navigate(['/employees'], {
       queryParamsHandling: 'preserve'
});
```


trivadis makes IT easier.

Query Parameters: Reading

Step 3: Reading query parameters

```
constructor(
    private route: ActivatedRoute,
    private router: Router
  ) {}

  ngOnInit() {
    this.listFilter =
               this.route.snapshot.queryParams['filterBy'] || '';
  }
```


trivadis makes IT easier.

Providing Data with a Route

- Passing data to a route with the data property

```
const routes: Routes = [
  { path: 'employees', component:EmployeeListComponent },
  { path: 'employees/new', component:EmployeeComponent },
  {
    path: 'employees/:employeeId',
    component: EmployeeComponent,
    data: { title: 'Employee' }
  }
];
```

```
this.route.snapshot.queryParams['title'];
```


Providing Data with a Route Resolver

- Problem that a page loads and after that id loads the data. So it is just partially loaded
- This can be avoided by using a router resolver

```
@Injectable()
export class ProductResolver implements Resolve<IProduct> {
        constructor(private productService: ProductService,
                private router: Router) { }

        resolve(route: ActivatedRouteSnapshot,
                state: RouterStateSnapshot): Observable<IProduct>{
             let id = route.paramMap.get('id');
             // load data and return it
        }
}
```

Add a resolver to a Route

- The Resolver can be added to the route definition

```
const routes: Routes = [
…
{
        path: ':id',
        component: ProductDetailComponent,
        resolve: { product: ProductResolver }
}
];
```


trivadis makes IT easier.

Reading data from a resolver

- Instead of reading from a snapshot, reading from Observables is much better to get informed about route changes

```
export class ProductDetailComponent implements OnInit {
        constructor(private route: ActivatedRoute) {}

        ngOnInit(): void {
           this.route.data.subscribe(data => {
                this.product = data['product'];
           });
        }
}
```


trivadis makes IT easier.

Child routes

- With a child route <router-outlet> can be nested
- Can be used to easyly change the basic layout

```
const routes: Routes = [
{
  path: '',
  component: StandardLayoutComponent,
  children: [
   { path: '', component: WelcomeComponent },
   { path: 'employees', loadChildren:
        './employee/employee.module#EmployeeModule' },
   { path: 'about', loadChildren:
        './about/about.module#AboutModule' }
  ]
}];
```


Guards

- Guards can be helpful for:
  - limiting access to a route
  - warning before leaving a route
  - Retrieving data before accessing a route
- Guards processing:
  - canDeactivate
  - canLoad
  - canActivateChild
  - canActivate
  - resolve

Guards

- A simple class which implements an interface

```
@Injectable()
export class AuthGuard implements CanActivate {
    constructor(private authService: AuthService,
                private router: Router) { }

    canActivate(route: ActivatedRouteSnapshot, state:
                RouterStateSnapshot): boolean {
        return isLoggedIn;
    }
}
```


trivadis makes IT easier.

Guards - CanDeactivate

- Another example:

```
@Injectable()
export class EmployeeEditGuard implements
                CanDeactivate<EmployeeComponent> {

  canDeactivate(component: EmployeeComponent): boolean {
    return confirm(`Navigate away and lose all changes?`);
  }

}
```


trivadis makes IT easier.

Guards - CanDeactivate

- Guards need to be registered in our ngModule or in our project in our barrel

```
@NgModule({
  imports: [
    SharedModule,
    EmployeeRoutingModule
  ],
  exports: [],
  declarations: [...fromContainers.containers,
                 ...fromComponents.components, ...fromPipes.pipes],
  providers: [...fromServices.services, ...fromGuards.guards]
})
export class EmployeeModule {}
```


Guards - CanDeactivate

- Another example:

```
const routes: Routes = [
   {
     path: ':employeeId',
     component: fromContainer.EmployeeComponent,
     canActivate: [fromGuards.AuthGuard],
     canDeactivate: [fromGuards.EmployeeEditGuard]
   }
];
```

Preloading Feature Module

- Our feature module is configured for lazy loading
- Sometimes it is useful to preload it behind the scene.
- This can be done with Preloading (Eager Lazy Loading)
- There are three possibilites:
  - No preloading
  - Preload all
  - Custom (own preload strategy)

Preloading all

- To enable we pass a preload strategy to the forRoot() method
- CanLoad Guard can block preloading

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

Custom Preloading Strategy

- This allows us that just some of the modules are preloaded
- A custom preloader is just a simple Angular service

```
export class AppCustomPreloader implements PreloadingStrategy {

  preload(route: Route, load: Function): Observable<any> {
    return route.data && route.data.preload ? load() : of(null);
  }

}
```


trivadis makes IT easier.

Custom Preloading Strategy

- Register our custom preloader to the forRoot() method

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


trivadis makes IT easier.

Custom Preloading Strategy

- And now mark the module which has to be preloaded

```
const routes: Routes = [{
    path: '', component: StandardLayoutComponent,
    children: [
      { path: '', component: WelcomeComponent },
      { path: 'employees', loadChildren:
               './employee/employee.module#EmployeeModule',
        data: { preload: true }
      },
       …
    ]
}];
```

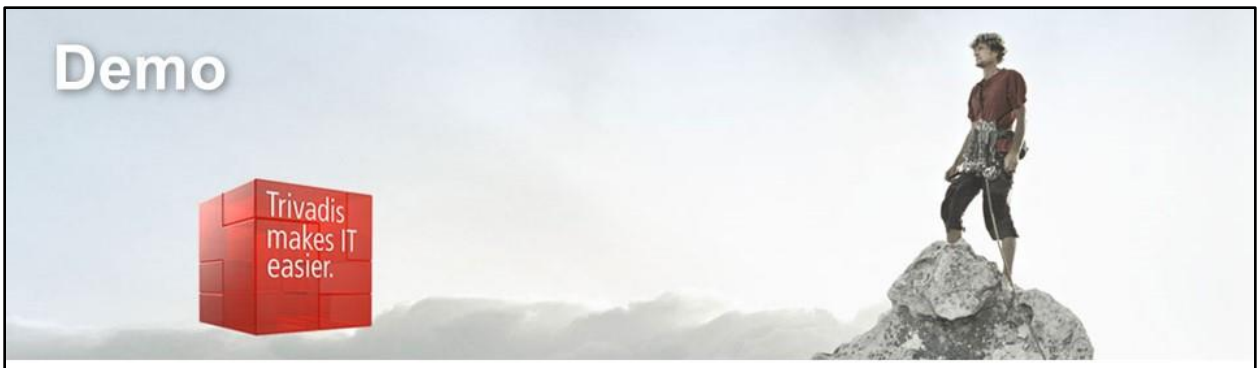

trivadis makes IT easier.

Summary

- Angular has an integrated Router
- Routing means navigating to different components
- Providing data with a resolver
- Guards can be helfpul
- With a Custom Preloading Strategy we can eager lazy load some modules

trivadis makes IT easier.

Demo



Move all employee routes to a employee-routing.module.ts

Make sure lazy loaded is implemented

Add Query Parameter for search functionality

Create a Route Guard (e.g. when leaving our edit form and ask the user for it)

Create a custom preloading strategy

trivadis
makes IT easier.

HttpClient

- In Angular 4.3, a new module for accessing REST Services has been introduced.
- It is a new implementation of the former *HttpModule*
- The new *HttpClient* service was introduced
- The old service Http is deprecated since Angular 5
- Working with Observables is the default . In Angular 5, the **pipe** operator was introduced

trivadis
makes IT easier.

Using HTTPClient

- **Step 1:** Register the Http Service Provider => The HttpModule

```
...
import { HttpClientModule } from '@angular/common/http';
...
@NgModule({
  imports: [CommonModule],
  exports: [CommonModule,
        HttpClientModule],
  declarations: [...],
  providers: [...],
})
export class SharedModule { }
```


Using HttpClient

```
import { HttpClient } from '@angular/common/http';

@Injectable()
export class EmployeeService {
    constructor(private _httpClient: HttpClient) { }

     getEmployees(): Observable<Employee[]> {
        return this.http
          .get<Employee[]>(`${environment.apiBaseUrl}/employees`)
          .pipe(catchError((error: any) =>
                      Observable.throw(error)));
      }
}
```

Using HTTP with Observables

```
export class EmployeeListComponent implements OnInit {

    employees: Employee[];

    constructor(private employeeService: EmployeeService){ }

    ngOnInit() {
        this.employeeService.getEmployees()
        .pipe(
          map(list => this.employees = list)
        );
    }
...
}
```


Interceptors

- Interceptors are sitting in between your application and the backend.
- By using interceptors you can transform a request coming from the application before it is actually submitted to the backend.

Interceptors

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  intercept (req: HttpRequest<any>, next: HttpHandler):
       Observable<HttpEvent<any>> {

    const authReq = req.clone({
      headers: req.headers.set('Authorization', 'Bearer: XYZ')
    });
    return next.handle(authReq);
  }

}
```


Providing the Interceptor

```
@NgModule({
  imports: [CommonModule],
  exports: [CommonModule, HttpClientModule],
  declarations: [...],
  providers: [
    {
      provide: HTTP_INTERCEPTORS,
      useClass: AuthInterceptor,
      multi: true
    }
  ],
})
export class SharedModule { }
```

Global Error Handler

- Create a global http error handler
- http handle is an observable
- catchError was introduced with lettable operators

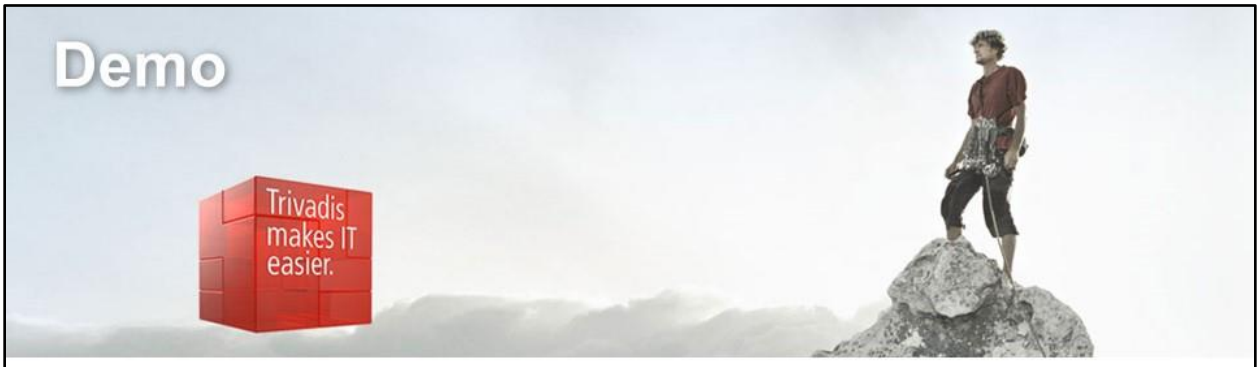
Global Error Handler

```
intercept(req: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
    return next.handle(req).pipe(
      catchError((error, caught) => {
        if (error instanceof HttpErrorResponse) {
          const httpError: HttpErrorResponse = error;
          console.log(`${httpError.message}`);
        }

        // Important! Always rethrow it
        return Observable.throw(error);
      })
    );
  }
```

Summary

- Angular's HttpClient-Services are in @angular/common/http
- You can use Promises or Observables
- Observables are especially powerful if you chain multiple calls

trivadis makes IT easier.

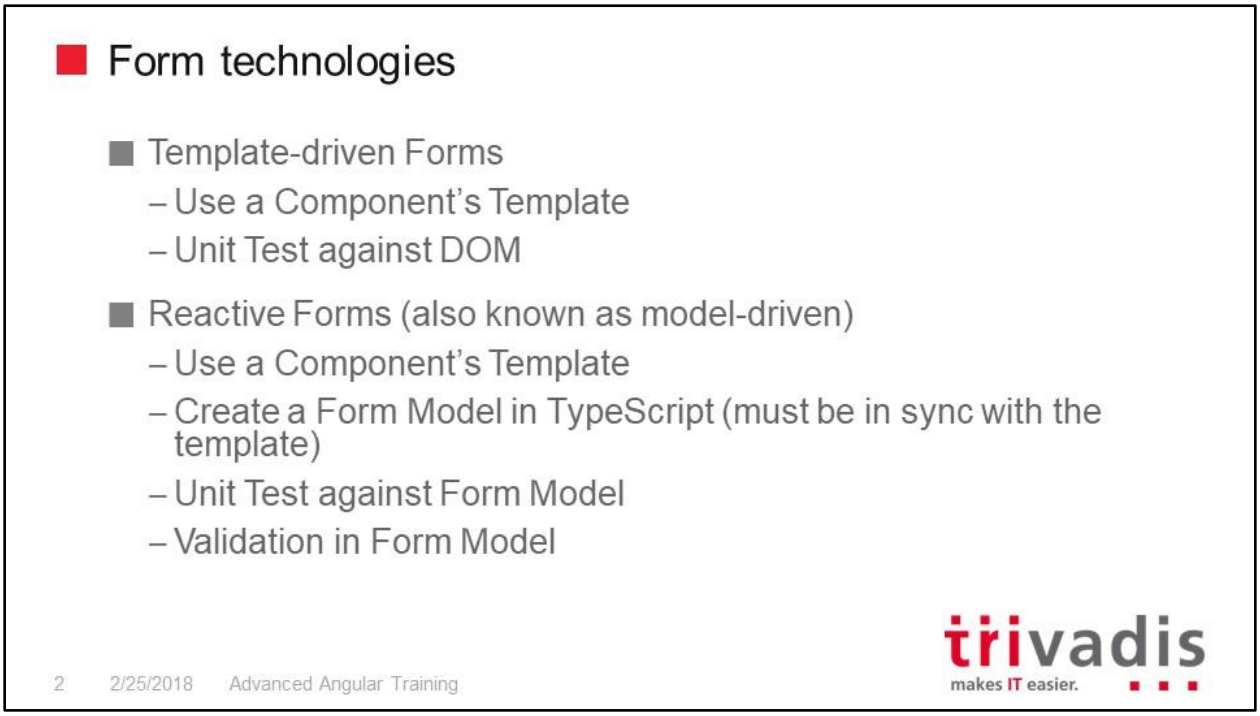


Create a new interceptor

Create a global http error handler

trivadis makes IT easier.

07_TemplateDrivenForms



Template-driven Forms

- Kind of forms used with Angular 1.x
- Bind directives and behavious to your templates
- Used with ngModel, required, minlength etc.
- The template is doing the work

Starting Position

```
<form novalidate> <!- only for Angular<4 -->
    <div class="form-group row">
        <label>Firstname</label>
        <input type="text" class="form-control">
    </div>
    <div class="form-group row">
        <label>Lastname</label>
        <input type="text" class="form-control">
    </div>
    <div class="form-group row">
        <label>Email</label>
        <input type="text" class="form-control">
    </div>
    <div class="form-group row">
        <label>Confirm Email</label>
        <input type="text" class="form-control">
    </div>
    <button type="submit"> Save </button>
</form>
```

Template-driven Forms

- Add the FormsModule to your AppModule to use ngModel

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

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


Template-driven Forms

- Or better put FormsModule to your SharedModule

```
import { NgModule }      from '@angular/core';
import { CommonModule } from '@angular/common';
import {FormsModule} from '@angular/forms';

@NgModule({
  imports: [CommonModule],
  exports: [CommonModule, HttpClientModule, FormsModule],
  declarations: []
})
export class SharedModule { }
```

Template-driven Forms

- Declare our model class

```
@Component({...
  templateUrl: 'employee-form.component.html'
})
export class EmployeeFormComponent {
 employee:Employee = {firstname: 'Thomas',
                lastname: 'Gassmann',
                email: 'thomas.gassmann@trivadis.com'};
}
```


trivadis makes IT easier.

Template-driven Forms

- Binding ngForm and ngModel

```
<form novalidate #f="ngForm">
   <div class="form-group row">
        <label>Firstname</label>
        <input type="text"
               name="firstname"
               [(ngModel)]="employee.firstname"
               class="form-control">
    </div>
</form>
```


trivadis makes IT easier.

Template-driven Forms

- Add submit functionality by adding the ngSubmit event directive

```
<form (ngSubmit)="onSubmit(f)" #f="ngForm">
   …
</form>
```

```
export class AppComponent {
 employee:Employee= {…};
 onSubmit({ valid }: { valid: boolean }) {
    console.log(valid)
 }
}
```


trivadis makes IT easier.

Template-driven error validation

- Similar approach as in Angular 1.x.
- Let's start by disabling our submit button until the form is valid

```
<form (ngSubmit)="onSubmit(f)" #f="ngForm">
   …
   <button type="submit" [disabled]="f.invalid">Save</button>
</form>
```


trivadis makes IT easier.

Template-driven error validation

- Mark each <input> as required and show error message if needed

```
<form (ngSubmit)="onSubmit(f)" #f="ngForm">
  <label>
   <span>Firstname</span>
   <input type="text" name="firstname"
        [(ngModel)]= "employee.firstname" required>
  </label>
  <div *ngIf="f.controls.firstname?.errors?.required">
        Firstname is required
  </div>
   …
   <button type="submit" [disabled]="f.invalid">Save</button>
</form>
```


Template-driven error validation

- Extend our forms with nested objects and data

```
<form (ngSubmit)="onSubmit(f)" #f="ngForm">
  <label>
   <span>Firstname</span>
   <input type="text" name="firstname" #firstname="ngModel"
        [(ngModel)]= "employee.firstname" required>
  </label>
  <div *ngIf="firstname.errors?.required && firstname.touched"
        class="error">
        Firstname is required
  </div>
   …
   <button type="submit" [disabled]="f.invalid">Save</button>
</form>
```

Template-driven custom validation

- A custom validator is just a directive

```
@Directive({
  selector: '[appIsEmails][ngModel]',
  providers: [
    { provide: NG_VALIDATORS, useExisting: IsEmailValidator, multi:
true }
  ]
})
export class IsEmailValidator implements Validator {
  validate(c: AbstractControl) {
      return EmployeeValidators.emailValidator(c);
  }
}
```


Template-driven custom validation

```
export class EmployeeValidators {
  static emailValidator(control: AbstractControl):
                                        ValidationErrors | null {
    const val: string = control.value;
    if (!val || val.indexOf('@') > 0) {
      return null;
    }
    return { invalidemail: true };
  }
}
```

Template-driven custom validation

- Use it in markup

```
…
 <div class="form-group row">
        <label>Email</label>
        <input type="text" class="form-control"
               name="email" #email="ngModel"
               [(ngModel)]="employee.email" appIsEmails>

     <div *ngIf="email.errors?.invalidemail && email.touched">
            Invalid email
        </div>
 </div>
```


trivadis makes IT easier.

Summary

- Bind directives and behavious to our templates
- Used with ngModel, required, minlength etc.
- Template is doing the work

trivadis makes IT easier.

08_ReactiveForms

Reactive Forms

- Also known as model-driven
- Avoiding directives such as ngModel, required etc.
- Use the underlying APIs to do the work by creating a instance inside a component class and construct JavaScript models.
- Very testable
- Keeps all logic in the same place

trivadis
makes IT easier.

Starting Position

```
<form novalidate> <!- only for Angular<4 -->
    <div class="form-group row">
        <label>Firstname</label>
        <input type="text" class="form-control">
    </div>
    <div class="form-group row">
        <label>Lastname</label>
        <input type="text" class="form-control">
    </div>
    <div class="form-group row">
        <label>Email</label>
        <input type="text" class="form-control">
    </div>
    <div class="form-group row">
        <label>Confirm Email</label>
        <input type="text" class="form-control">
    </div>
    <button type="submit"> Save </button>
</form>
```


Reactive Forms

- Add the **ReactiveFormsModule** to your AppModule

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

@NgModule({
  imports: [CommonModule],
  exports: [CommonModule, HttpClientModule, ReactiveFormsModule],
  declarations: []
})
export class SharedModule { }
```

Reactive Forms – FormControl

- FormControl
  - A class that powers an individual form control
  - Tracks the value
  - Validation status

```
ngOnInit() {
  this.myNameCtrl = new FormControl('Thomas Gassmann');
}
```


Reactive Forms – FormGroup

- FormGroup
  - A group of FormControl instances
  - Tracks the value
  - Validation status
  - Can be nested (FormGroup in FormGroup)

```
ngOnInit() {
  this.myGroup = new FormGroup({
    name: new FormControl('tga'),
    email: new FormControl('thomas.gassmann@trivadis.com')
  });
}
```

But how we use them?

Reactive Forms – use of FormGroup

- For binding declare **formGroup** on the form and **formControlName** as a directive
  Note: ngModel and name attribute are no longer required

```
<form novalidate [formGroup]="employee">
  <label>
   <span>Firstname</span>
   <input type="text" formControlName="firstname">
  </label>
  …
</form>
```


Reactive Forms – implementing the FormGroup model

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

@Component({...})
export class EmployeeFormComponent {
 employee:FormGroup;
  ngOnInit() {
    this.employee = new FormGroup({
      firstname: new FormControl(''),
      lastname: new FormControl(''),
      email: new FormControl(''),
    });
  }
}
```

Reactive Forms – set form value

- To set a value from our model we can use setValue or patchValue
- With setValue all controls need to be set whereas patchValue can be used to set only some of the form values.

```
this.employee.setValue({ firstname: 'Thomas',
            lastname: 'Gassmann', email: 'tg' });

// OR


this.form.patchValue({ firstname: 'Thomas' });
```


Reactive Forms

- Exactly the same as the template-driven approach. Add submit functionality by adding ngSubmit event directive

```
<form (ngSubmit)="onSubmit()" [formGroup]="employee">
   …
</form>
```

```
export class EmployeeFormComponent {
 employee:FormGroup;
 onSubmit() {
    console.log(this.employee.valid)
 }
}
```

Reactive Forms error validation

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

@Component({...})
export class EmployeeFormComponent {
 employee:FormGroup;
  ngOnInit() {
    this.employee = new FormGroup({
      firstname: new FormControl('', Validators.required),
      email: new FormControl('',
                [Validators.required, Validators.minLength(2)]),
    });
  }
}
```


Reactive Forms error validation

- Let's start by disabling our submit button until the form is valid

```
<form novalidate (ngSubmit)="onSubmit()" [formGroup]="employee">
   …
   <button type="submit"
                [disabled]="employee.invalid">Save</button>
</form>
```

Reactive Forms error validation

- Use the .controls property on the FormGroup object to check for errors

```
<div *ngIf="employee.controls.firstname?.errors?.required">
       Firstname is required
</div>
```

- Or use .get() method that will lookup for that control

```
<div *ngIf="employee.get('firstname').hasError('required')
         && employee.get('firstname').touched">
       Firstname is required
</div>
```


trivadis makes IT easier.

Reactive Forms - FormBuilder

- Usage of FormGroup and FormControl is complex
- Let's simplify it with FormBuilder
- **FormBuilder** is a helper class to build forms
- It is a flexible and common way to configure forms

So, let's change our existing form

trivadis makes IT easier.

Reactive Forms - FormBuilder

- FormBuilder is injected into constructor

```
import { Component, OnInit } from '@angular/core';
import {FormBuilder,FormGroup,Validators} from '@angular/forms';

@Component({...})
export class EmployeeFormComponent {
 employee:FormGroup;
 constructor(private fb: FormBuilder) {}
  ...
}
```


Reactive Forms - FormBuilder

```
@Component({...})
export class EmployeeFormComponent {
 employee:FormGroup;

 constructor(private fb: FormBuilder) {}

 ngOnInit() {
    this.employee = this.fb.group({
      firstname: ['', Validators.required],
      email: ['',[Validators.required,Validators.minLength(4)]],
    });
  }
}
```

Reactive Forms – Read values from FromGroup

```
@Component({...})
export class EmployeeFormComponent {
 form:FormGroup;

 onSubmit() {
   if (this.form.touched && this.form.valid) {
     const newModel = Object.assign({}, this.model,
                                    this.form.value);
     const newModel2 = { ...this.model, ...this.form.value };

     this.update.emit(newModel);
    }
  }
}
```


Reactive Forms – Custom validation

- Just a simple method

```
export class EmployeeValidators {
  static emailValidator(control:
                        AbstractControl):ValidationErrors | null {
    const val: string = control.value;
    if (!val || val.indexOf('@') > 0) {
      return null;
    }
    return { invalidemail: true };
  }
}
```

Reactive Forms – Custom validation

- synchronous and asynchronous validations are possible

```
this.fb.group(
    {
        …
        email: [
           '',
            [Validators.required,
            EmployeeValidators.emailValidator],
            EmployeeValidators.checkEmailUniqueAsync(this.service)
        ]
});
```


trivadis makes IT easier.

updateOn option

- For Forms with heavy validation requirements, updating on every keystroke can sometimes be too expensive
- Angular 5 provides a new option that improves performance by delaying form control updates until the blur or the submit event.

```
email: new FormControl(null, {
        validators: Validators.required,
        updateOn: 'blur'
}),
```

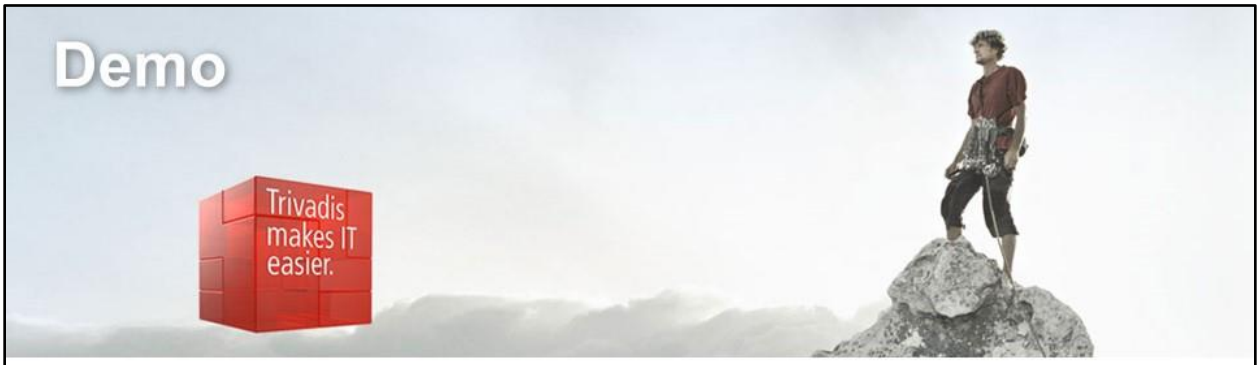
- **Not working with FormBuilder. Open bug!**

trivadis makes IT easier.

Summary

- Reactive-Forms are super powerful
- Avoid directives such as ngModel, required etc.
- Import ReactiveFormsModule

trivadis makes IT easier.



Create a employee form with reactive forms

trivadis makes IT easier.

What kind of tests exists?

Unit Tests

- Test certain functions or units of code
- Super fast
- Easy to write

e2e Tests

- Runs the real application and simulates the users behavior
- Very slow
- Very fragile

trivadis
makes IT easier.

Test Strategy

- Use the smallest test type possible
- Write readable tests
- Do not mock unless there is a reason to
- Test all the time
- Make tests fail

Tools

KARMA

- Test runner
- Test code on various browser
- Integrate with various continuous integration services like Jenkins
- Complete configuration with Angular CLI
- *.spec.ts files are considered

```
ng test
```

```
ng test --code-coverage
```


Jasmine

- Behavior-driven development framework
- Fast
- Dependency free
- Already integreated with Angular CLI

Protractor

end to end testing for AngularJS

- End-to-end test framework
- Built on top of Selenium WebDriverJS
- Runs in a real browser
- Test like a user
- Protractor is also configured with the Angular CLI

```
ng e2e
```


trivadis makes IT easier.

Insiders' tip

Wallaby.js

- Integrated Continuous Testing Tool for JavaScript

trivadis makes IT easier.

Wallaby.js

```
21      it("is defined", () => {
22          console.log("Hallo Basta!");  Hallo Basta!
23          expect(comp).toBeTruthy();
24      });
```

```
21      it("is value equals", () => {
22          expect(1).toBe(2);  Expected 1 to be 2.
23      });
```


trivadis makes IT easier.

Unit Test vs. Integration Test

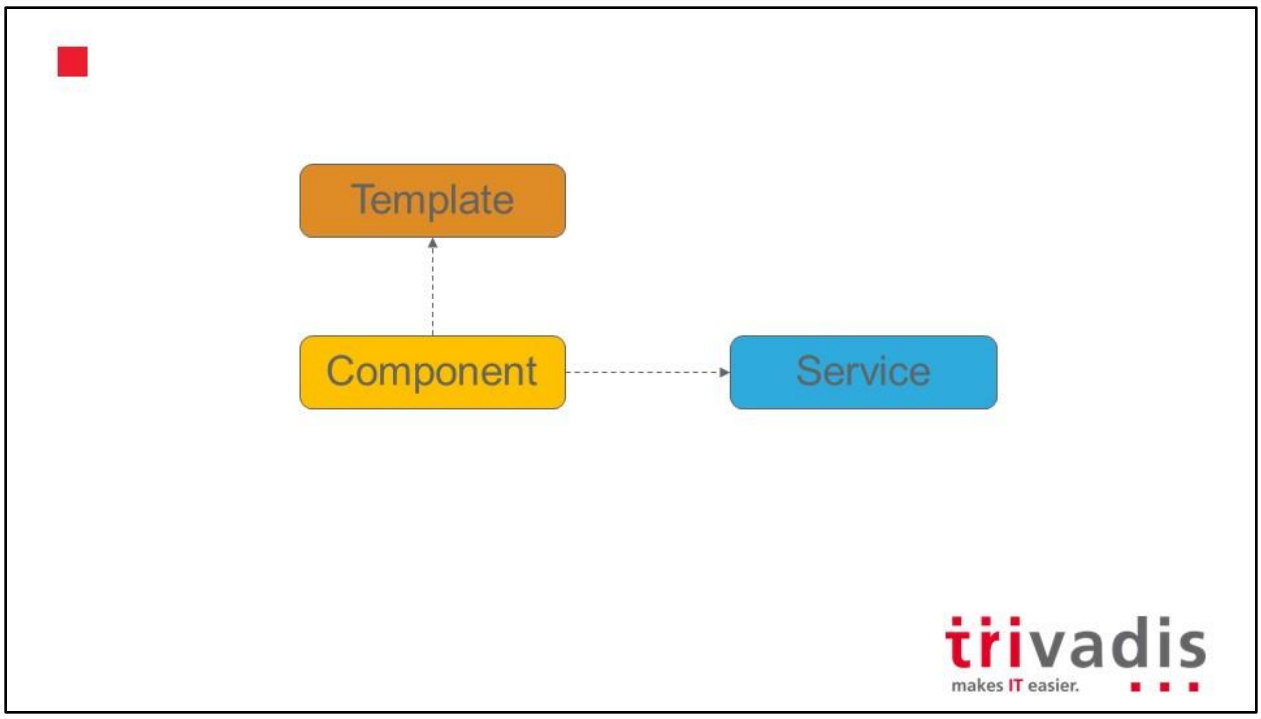
Unit Test

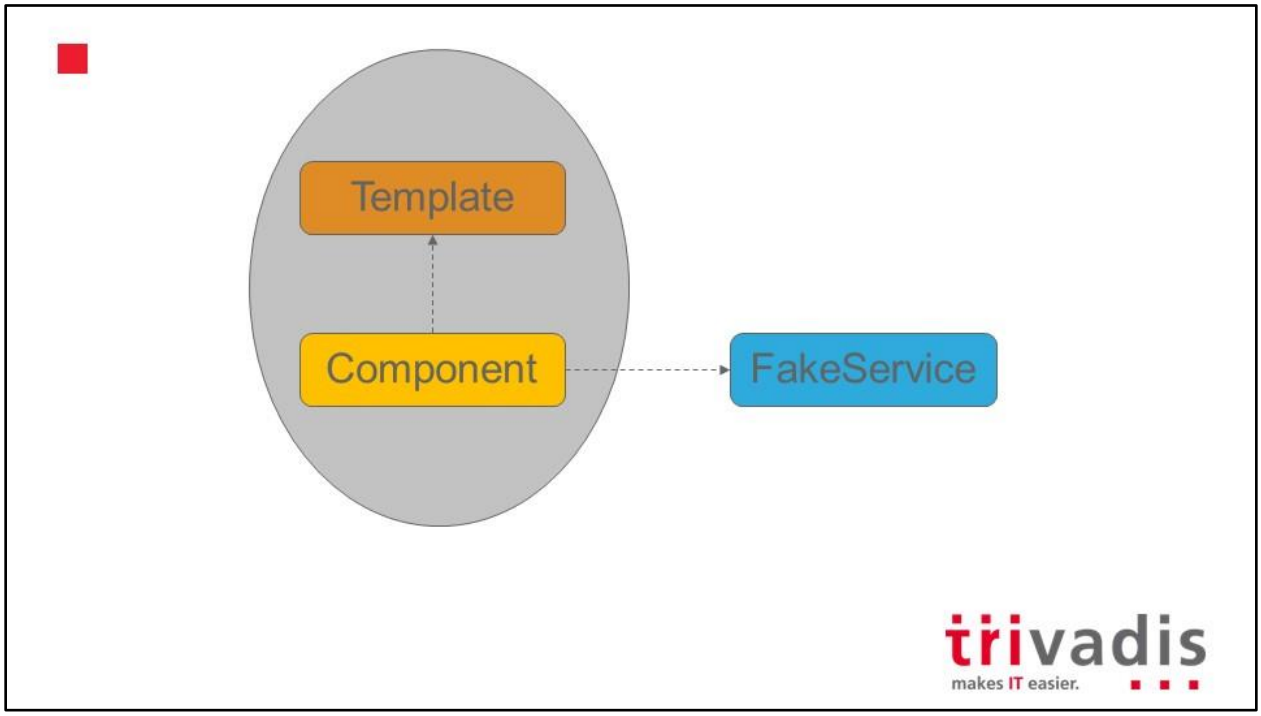
- Test in isolation, without external resources (e.g. templates)
- Just a simple function

Integration Test

- Test with external resources (e.g. templates)

trivadis makes IT easier.





Jasmin Basics

```
describe('Test Suite Name', () => {

        beforeAll(() => {});
        beforeEach(() => {});
        afterEach(() => {});
        afterAll(() => {});

        it('A Spect / Unit Test', () => {

        });

});
```


Jasmin Basics

```
describe('Test Suite', () => { // Test Suite
    it("A single spec", () => { // Spec or test
        // Arrange
        let component = new AppComponent();

        // Act
        component.ngOnInit();

        // Assert
        expect(component.title).toBe("hallo thomas");
    });
});
```

Test a pipe

- Pipe is only a class which implements an interface.

```
describe('FilterPipeTests', () => {
        let pipe: EmployeeFilterPipe;
        let employeees: Employee[];

        beforeEach(() => {
                pipe = new EmployeeFilterPipe();
        });

        it('should return all data', () => {
                const result = pipe.transform(employeees, null);
                expect(result.length).toBe(2);
        });
});
```


Test a Component

- Try to test with hardly any dependency to Angular
- When possible create a new instance directly

```
beforeEach(() => {
   comp = new PersonDetailComponent();
});
```

Test a Component

- Otherwise use the TestBed to configure a new TestingModule

```
beforeEach(
   async(() => {
     TestBed.configureTestingModule({
       imports: [RouterTestingModule, FormsModule],
       declarations: [EmployeeListComponent],
       schemas: [NO_ERRORS_SCHEMA]
     }).compileComponents();
   })
 );
```


Test a Component

- Fixture has a collection of helpful functions

```
beforeEach(() => {
    fixture = TestBed.createComponent(PersonDetailComponent);
    comp = fixture.componentInstance;
    fixture.detectChanges(); // triger change detection
});

 it('should render title in a h2 tag',
    async(() => {
      const result =
       fixture.nativeElement.querySelector('h2').textContent;
      expect(result).toContain('Employees');
    })
  );
```

Test a Component

- fixture.detectChanges triggers a change detection cycle
- Especially necessary when accessing bindings (e.g. {{ xyz }})
- whenStable() will be called when detectChanges is over

```
it('should render title in a h2 tag',
   async(() => {
       fixture.detectChanges(); // triger change detection
       fixture.whenStable().then(() => {
              expect(…)
       });
   })
 );
```


Test a Service

- Try to test with fewest possible dependencies to Angular
- We define our testing module and import HttpClientModule

```
beforeEach(() => {
   TestBed.configureTestingModule({
     imports: [HttpClientModule],
     providers: [EmployeeService]
   });
 });
 beforeEach(() => {
   service = TestBed.get(EmployeeService);
 });
```

Test a service

- When using an async operation, always add the async function to your test
- Karma is not waiting for async operator
- A test without an assertion is always a passed test

```
it("should get a person async", async(() => {
    service.getPersonAsync(1).then((result) => {
        expect(result.firstname).toBe("Thomas");
    });
}));
```


FakeAsync

- The main advantage of fakeAsync over async is that the test appears to be synchronous.
- Calling tick() simulates the passing of time until all the pending asynchronous activities finish

```
it('should test fakeAsync',
  fakeAsync(() => {
      let flag = false;
      setTimeout(() => { flag = true; }, 50);
      expect(flag).toBe(false);
      tick(50);
      expect(flag).toBe(true);
  })
);
```

Mock a Service

- What if the test has no access to the REST Service
- We have to mock our service call

```
import { HttpClientTestingModule } from '@angular/common/http/testing';
import { HttpTestingController } from '@angular/common/http/testing';

beforeEach(() => {
  TestBed.configureTestingModule({
    imports: [HttpClientTestingModule],
    providers: [EmployeeService]
  });
});
```


Mock a Service

```
beforeEach(() => {
  service = TestBed.get(EmployeeService);
  httpMock = TestBed.get(HttpTestingController);
});

it('should create the app', async(() => {
  const dummy: Employee[] = [...];

  service.getEmployees().subscribe(list => { expect(list.length).toBe(2); });

  const request = httpMock.expectOne(`${environment.apiBaseUrl}/employees`);
  expect(request.request.method).toBe('GET');

  request.flush(dummy);
 })
);
```

Use Firefox as a second test browser

- Install Firefox launcher for karma

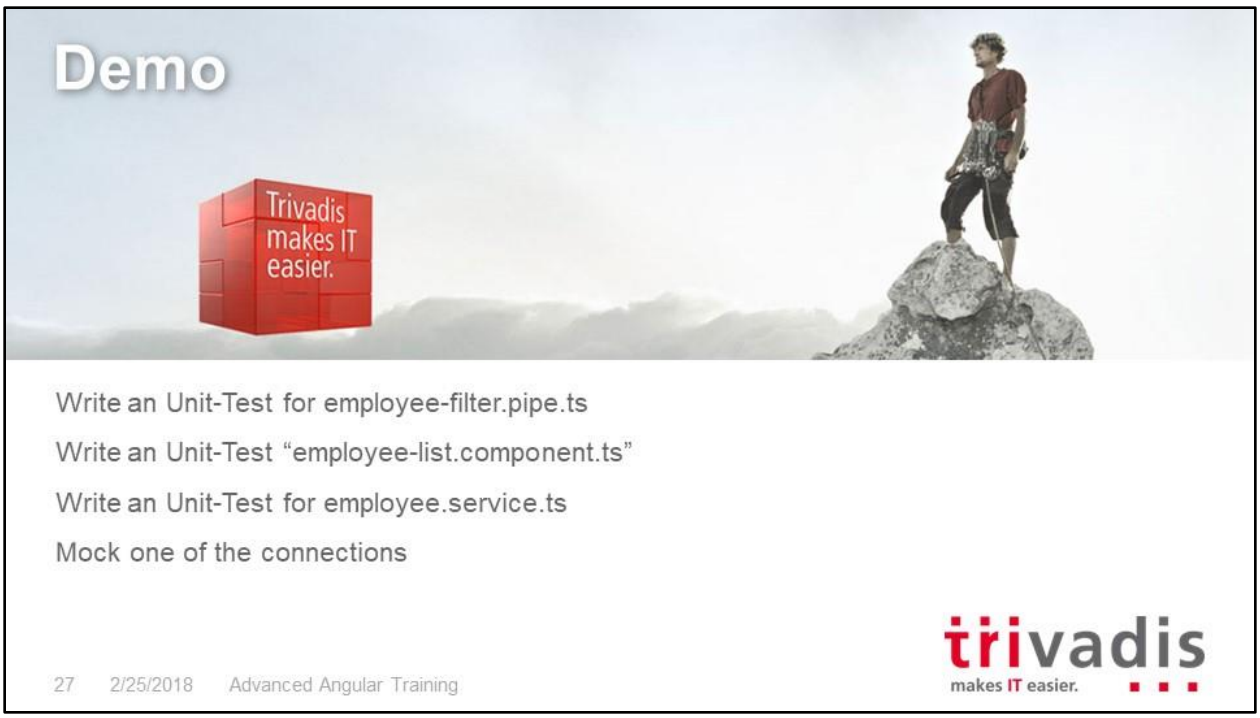
```
npm install karma-firefox-launcher --save-dev
```

- Modify karma.conf.js

```
plugins: [ …
        require('karma-firefox-launcher'),
…
],
…
browsers: ['Chrome', 'Firefox'],
…
```


Summary

- Jasmin is the default testing framework. It works perfectly with all other tools
- Angular CLI configures Karma and Protractor
- Async functions should always be mapped inside an async keyword



Write an Unit-Test for employee-filter.pipe.ts

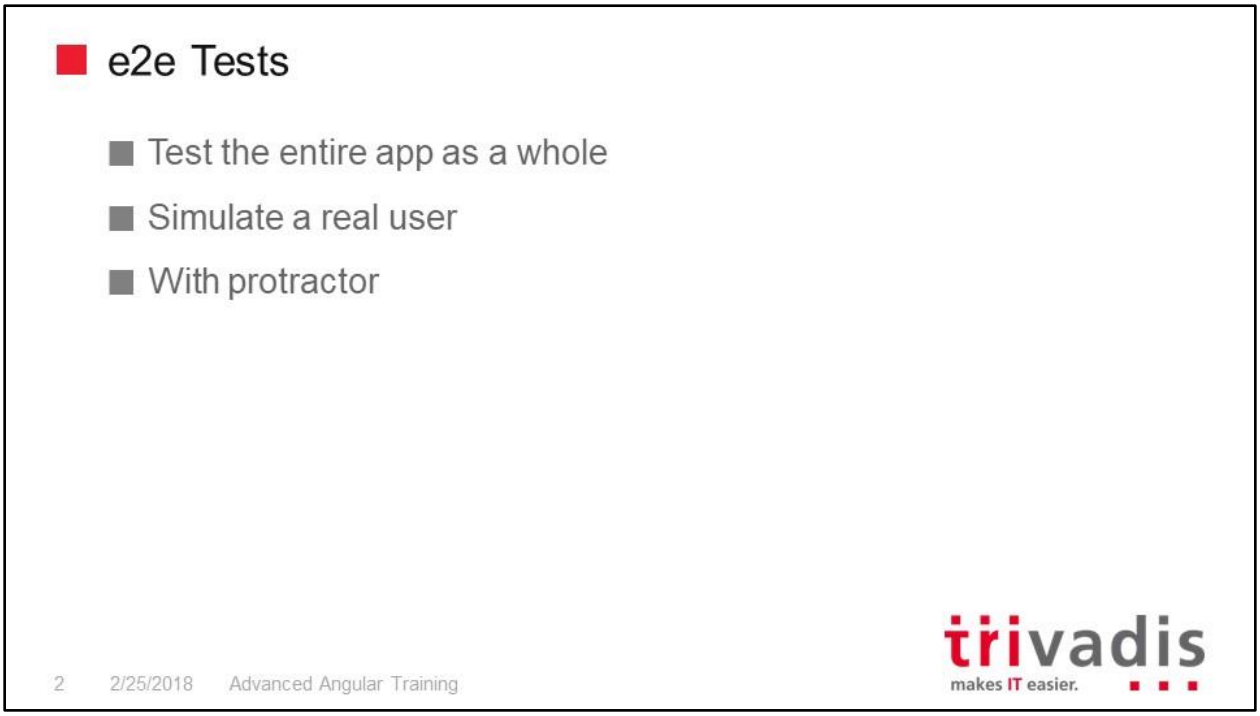
Write an Unit-Test “employee-list.component.ts”

Write an Unit-Test for employee.service.ts

Mock one of the connections

trivadis
makes IT easier.

10_e2eTesting



Basics

```
import { browser, element, by } from 'protractor';

describe('QuickStart E2E Tests', function () {

    beforeEach(function () {
        browser.get('/');
    });

    it('should display title', function () {
        expect(element(by.css('h1')).getText()).toEqual("hallo basta");
    });

});
```


trivadis makes IT easier.

Basics

- "Page object"-classes are like ViewModels to interact with
- Each end-to-end test spec should have a "page object"-class
- Decapsulation from a specific framework like protractor
- Tests are located in a separate folder, e.g. "/e2e"

trivadis makes IT easier.

Page Object

```
import { AppPage } from './app.po';

describe('byod-portal App', () => {
    let page: AppPage;

    beforeEach(() => { page = new AppPage(); });

    it('should display welcome message', () => {
        page.navigateTo('/');
        expect(page.getText('app-root h2'))
            .toEqual('Byod Portal');
    });
});
```


trivadis makes IT easier.

Browser, Element, By

- The browser represents the WebDriver-Instance
- With by and element we can access HTML Elements

```
import { browser, by, element } from 'protractor';

export class AppPage {
    navigateTo(url: string) {
        return browser.get(url);
    }
    getText(selector: string) {
        return element(by.css(selector)).getText();
    }
}
```


trivadis makes IT easier.

Test navigation link

- Click simulates a real user click on an element

```
it('should navigate to byod page', () => {
    page.navigateTo('/');

    page.getElement('[ng-reflect-router-link="/employees"]')
        .click();

    expect(page.getCurrentUrl()).toContain('/employees');
});
```


trivadis makes IT easier.

Test navigation link – Page Object

- On the WebDriver-Instance (browser) the url can be read or set

```
getCurrentUrl() {
    return browser.getCurrentUrl();
}

…
```


trivadis makes IT easier.

Test the search field

```
it('should search correct', () => {
    page.navigateTo('/employees');

    page.search('Thomas');

    expect(page.getResult().count()).toEqual(3);
});
```


Test the search field – Page Object

- On an element, sendKeys can be used to send keys to an input

```
search(search: string) {
    this.getElement('input[type="search"]').clear();
    this.getElement('input[type="search"]').sendKeys(search);
}

getResult() {
    return element.all(by.css('table.table tbody tr'));
}
…
```


Other useful function

- Pause is for debugging during development

```
browser.pause();
```

- If the application is not an Angular App

```
browser.ingoreSynchronization = true
```

- Take a screenshot

```
browser.takeScreenshot().then(png => {
  const stream = fs.createWriteStream(`screenshot-1.png`);
  stream.write(new Buffer(png, 'base64'));
  stream.end();
});
```

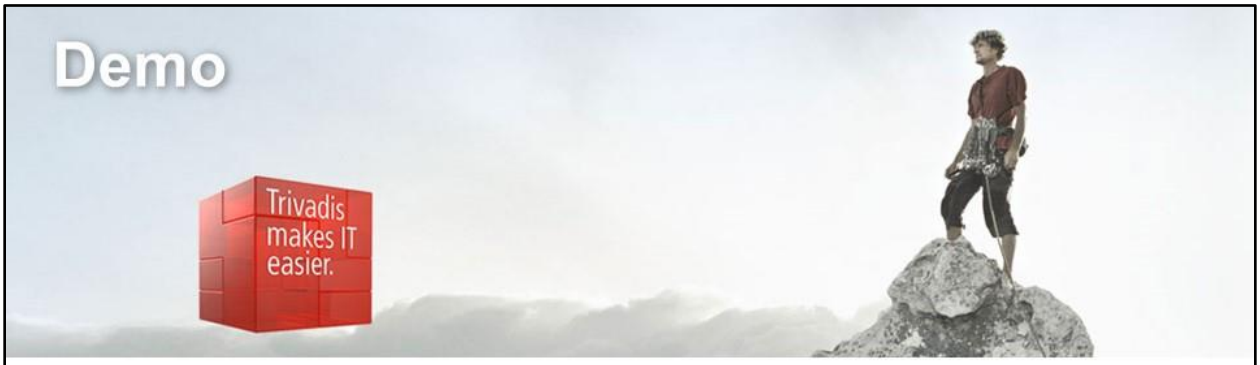

Settings

- Each time we execute end-to-end tests, an instance of chrome gets started, to powerup you can use the PhantomJS-plugin

Summary

- Protractor enables e2e tests
- browser, element and by are the most used commands for e2e tests
- e2e are an addition for unit testing.

trivadis makes IT easier.



Create an e2e test

- Test the click on employee link
- Test search functionality

trivadis makes IT easier.

11_Redux

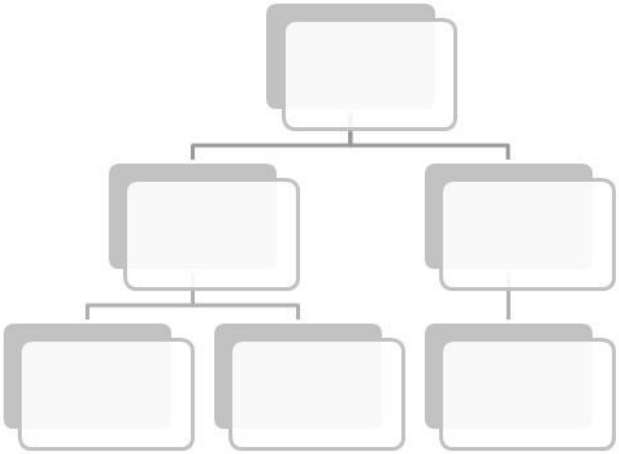
Redux

«Redux is not great for making simple things quickly. It's great for making really hard things simple.»

Jani Eväkal

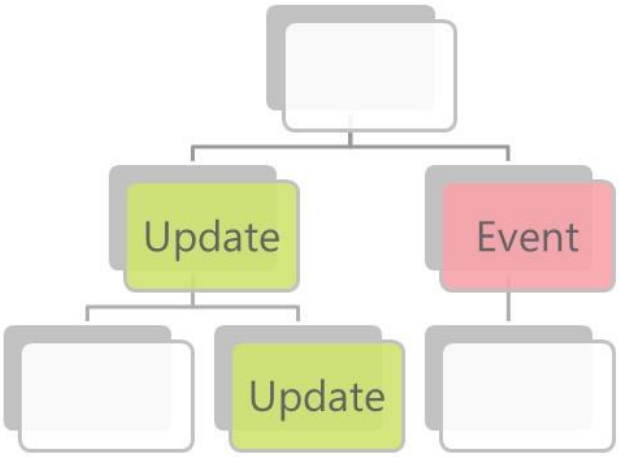
trivadis
makes IT easier.

Typical Angular Application



trivadis makes IT easier.

Typical Angular Application



trivadis makes IT easier.

State Management

With a state management system we would like to have:

- A single-source of truth of our applications state and
- encapsulate our business logic from our view logic

What is an Application State?

- Server response
- User input
- UI state
- Router / location state

The state is composed in our Store (and only in the store)

Redux Architecture: Core Concepts

- Single state tree
- Actions
- Reducers
- Store
- One-way dataflow

Redux: Single state tree

- One «big» JavaScript Object
- It is composed of reducers

```
// initial state
const state = {
        employees: []
};
```

Redux: Actions

- Actions have two properties: type and payload
- Dispatch actions to reducers

```
// an example of an action
const action = {
        type: 'ADD_EMPLOYEE',
        payload: {
                firstname: 'Thomas',
                lastname: 'Gassmann'
        }
};
```


Redux: Reducers

- Pure functions
- Given dispatched action
- Composes and returns new state

```
function reducer(state, action) {
    switch(action.type) {
        case 'ADD_EMPLOYEE': {
            const empl = action.playload;
            const newState = [...state.employees, empl];
            return { newState };
        }
    }
    return state;
}
```

Redux: Store

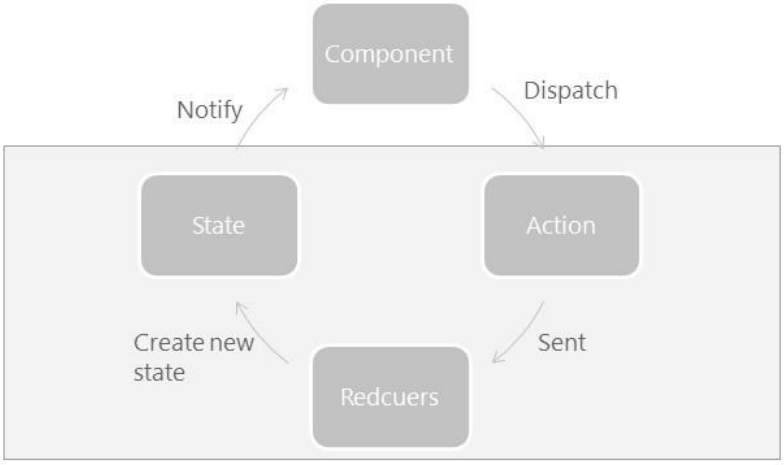
- State container
- Components subscribe to slices of state
- Components dispatch Actions to the store
- Reducers compose new State
- Store is updated and notifies subscribers
- Everything is immutable

Redux: Store

- Example of immutable operations

```
const person = {firstname: 'Thomas'};

const newP1 = Object.assign({}, person, {lastname: 'Gassmann'});
const newP2 = {…person, {lastname: 'Gassmann'});

console.log(person);  // {firstname: 'Thomas'}

console.log(newP1);  // {firstname: 'Thomas', lastname: 'Gassmann'}

console.log(newP2);  // {firstname: 'Thomas', lastname: 'Gassmann'}
```

One-way dataflow



trivadis makes IT easier.

Redux in Angular

- angular-redux

  https://github.com/angular-redux/store

- **ngrx**

  https://github.com/ngrx/platform

trivadis makes IT easier.

How to start

- Install npm package

```
npm install @ngrx/store --save
```

- Install effects module

```
npm install @ngrx/effects --save
```

- For debugging

```
npm install @ngrx/store-devtools --save
```

- Install «Redux» Chrome Extension

trivadis makes IT easier.

Integration

- Add references to your ngModule

```
@NgModule({
imports: [
        StoreModule.forRoot(fromStore.reducers),
        EffectsModule.forRoot(fromStore.effects),
        environment.production  ? [] : StoreDevtoolsModule.instrument({
                maxAge:  10
        }),
],
…
})
export class AppModule  {}
```


trivadis makes IT easier.

Create Actions

- Create a new File under actions/employee.action.ts
- Define action types according to what happen inside our app

```
export enum EmployeeActionTypes {
   LoadEmployees = '[Employees] Load Employee',
   LoadEmployeeSuccess = '[Employees] Load Employee Sucess',
   LoadEmployeeFail = '[Employees] Load Employee Fail'
}
```


Create Actions

- For loading all employees we have three different actions
- One for loading, one if it loads successful and one if there is an error

```
export class LoadEmployees implements Action {
  readonly type = EmployeeActionTypes.LoadEmployees;
}

export class LoadEmployeeSuccess implements Action {
  readonly type = EmployeeActionTypes.LoadEmployeeSuccess;
  constructor(public payload: Employee[]) {}
}

export class LoadEmployeeFail implements Action {
  readonly type = EmployeeActionTypes.LoadEmployeeFail;
  constructor(public payload: any) {}
}
```

Create Actions

- Export our actions for using it inside our reducer

```
// action types
export type EmployeeActions =
            | LoadEmployees
            | LoadEmployeeSuccess
            | LoadEmployeeFail
            ;
```


Create State Object

- Create a new File under state/app.state.ts
- State is just an interface and consists of other state objects

```
export interface AppState {
   employee: EmployeeState;
}
```

Create State Employee

- Create a new File under state/employee.state.ts
- File consists of the state and an initial state

```
export interface EmployeeState {
   employees: Employee[] ;
   loading: boolean;
   loaded: boolean;
}

export const initialState: EmployeeState = {
   employees : [],
   loading: false,
   loaded: false
};
```


Create Reducer

- Create a file reducers/employee.reducer.ts
- Reducer ist just a simple function which **always** returns a state

```
export function reducer(
  state = fromState.initialState,
  action: fromActions.EmployeeActions
): fromState.EmployeeState {

   switch(…) { …. }

   return state;
}
```

Implement our Reducer

- We have three actions at the moment. So let's implement the corresponding reducer
- LoadEmployee action

```
import * as fromActions from '../actions/employee.action';

switch (action.type) {
   case fromActions.EmployeeActionTypes.LoadEmployees: {
    return {
      ...state,
      loading: true,
    };
}
```


Implement our Reducer

- LoadEmployeeSuccess action
- Return always a new object (state).

```
case fromActions.EmployeeActionTypes.LoadEmployeeSuccess: {
    const employees = action.payload;

    return {
       ...state,
       loading: false,
       loaded: true,
       employees,
    };
 }
```

Implement our Reducer

- LoadEmployeeFail action
- Return always a new object (state).

```
case fromActions.EmployeeActionTypes.LoadEmployeeFail: {
    return {
        ...state,
        loading: false,
        loaded: false,
    };
}
```


trivadis makes IT easier.

Define our Reducer Map

- Every NgRx Store always has at least one ActionReducerMap definition with the definition of all reducers inside this module
- This structure represents our state
- Create a new file reducers/index.ts

```
export const reducers: ActionReducerMap<fromState.AppState> = {
  employee: fromReducer.reducer
};
```


trivadis makes IT easier.

Use NgRx inside our component

- Inject the Store inside the constructor
- And select the appropriate projection

```
@Component({
 templateUrl:  'employee-list.component.html'
})
export class EmployeeListComponent  implements  OnInit {
   employees$: Observable<Employee[]>;
   constructor(private store$: Store<fromStore.AppState>) { }

   ngOnInit() {
      this.employees$ = this.store$.select(fromStore.getEmployees);
   }
}
```


Create Selectors

- We make use of selectors to use a particular object of the state in our angular app.
- Inside our app state file. We export our employee state

```
export const getEmployeeState = (state: AppState) => state.employee;
```

- Now we create an export funtion for our array

```
export const getEmployees = (state: fromState.EmployeeState) => state.employees;
```

Create Selectors

- We can now use these functions to create a selector. We put all our selectors inside a selector.employee.ts file

```
export const getAllEmployees  = createSelector(
   getEmployeeState,
   fromReducer.getEmployees
);
```

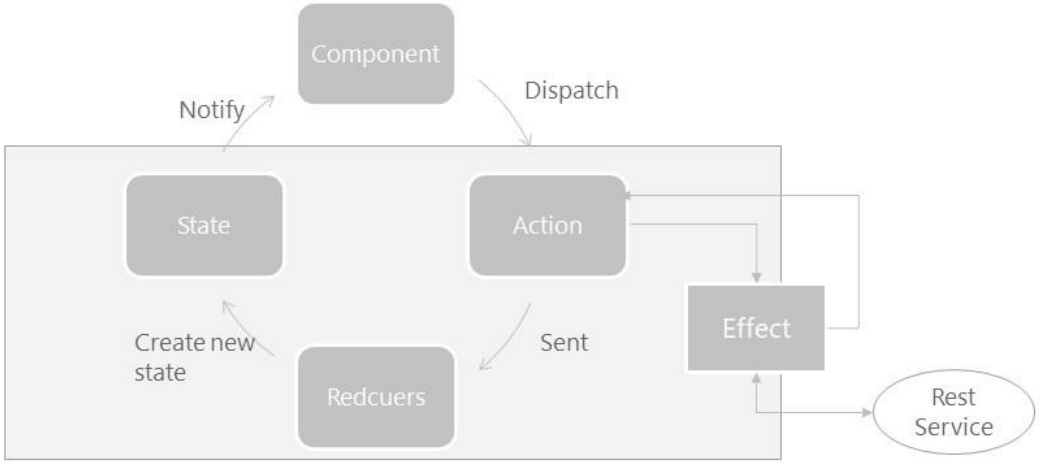

trivadis makes IT easier.

Effects

- Effects are for side effects. When we access data from outside of our store we use effects
- Isolate side effects from components
- Effects are normal TypeScript classes with at least one function
- Functions have an «**@Effect()**» decorator

trivadis makes IT easier.

Effects

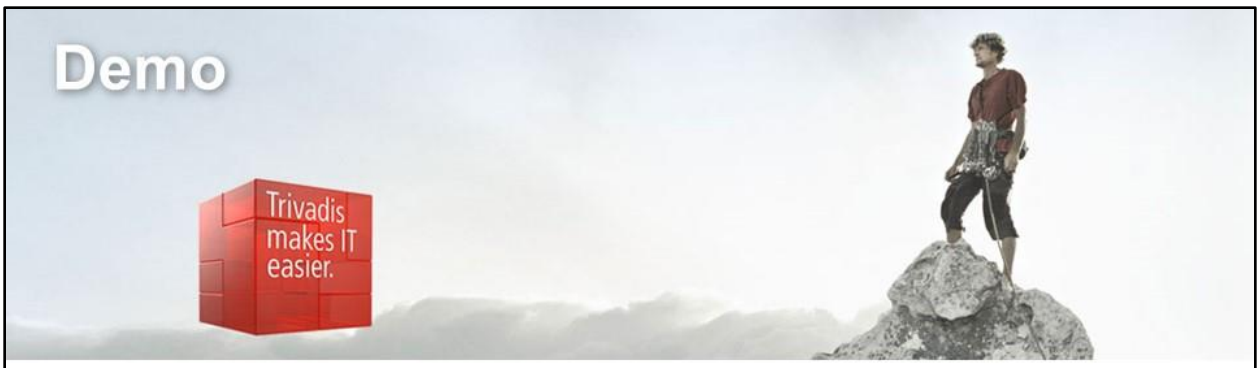


trivadis makes IT easier.

Create Effects

```
@Injectable()
export class EmployeeEffects {
  constructor(private actions$: Actions,
    private employeeService: fromServices.EmployeeService) {}

  @Effect()
  loadEmployees$ =
this.actions$.ofType(employeeActions.EmployeeActionTypes.LoadEmployees).pipe(
    switchMap(() => {
      return this.employeeService.getEmployees()
        .pipe(
          map(employees => new employeeActions.LoadEmployeeSuccess(employees)),
          catchError(error => of(new employeeActions.LoadEmployeeFail(error)))
        );
    })
  );
}
```


makes IT easier.



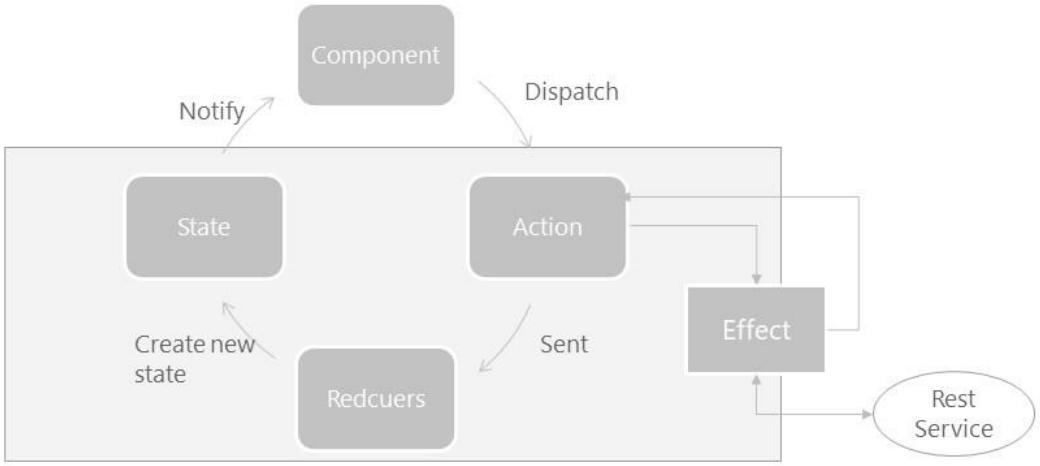
Add store functionality to our existing app

Modify our existing employee-list.component.ts to access our store

Create an effect to fetch the data

trivadis
makes IT easier.

Recap



trivadis
makes IT easier.

Advanced component architecture

Containers

- State-full Components
- Aware of Store
- Read data from store
- Dispatch Actions

Presentational

- State-less Components
- Not aware of Store
- Read data from @Input
- Callbacks via @Output

trivadis makes IT easier.

ChangeDetectionStrategy.OnPush

- OnPush means that the change detector's mode will be initially set to CheckOnce
- Angular only perform ChangeDetection if the reference of the input changes
- For our presentational components we can set ChangeDetectionStrategy to OnPush
  Because we fetch our data through an @Input
- For our container components we can also set ChangeDetectionStrategy to OnPush **if** we load everthing via an Observable .

trivadis makes IT easier.

ChangeDetectionStrategy.OnPush

- Inside our component decorator

```
@Component({
   selector: 'app-employee-form',
   ...
   changeDetection: ChangeDetectionStrategy.OnPush
})
```


trivadis makes IT easier.

Demo

trivadis makes IT easier.

Summary

- NgRx is one of the most popular redux libraries in Angular
- It consists of Actions, Reducers and the Store
- Effects are for fetching data outside of our store

12_Deployment

Steps to a successfull deployment

- TSLint
- Optimizing Code
- Build process

trivadis
makes IT easier.

TSLint

- Extensible static analysis tool
- Helps to catch common errors
- Catches the «oops» moments (debugger, console.log etc.)

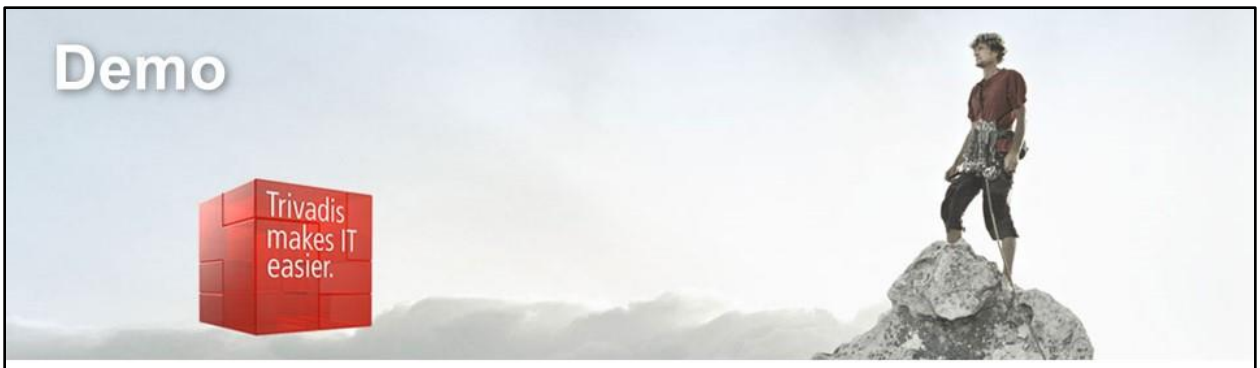
Run TSLint

Install TSLint

```
npm install tslint --save-dev          #local
npm install tslint -g                  #global
```

Run TSLint

```
node_modules/.bin/tslint --init        #local: create tslint.json
tslint --init                          #global: create tslint.json
tslint "app/**/*.ts"                   #run TSLint
ng lint                                #using with Angular CLI
```



Integrate TSLint in your project

Check error and fix it

trivadis
makes IT easier.

Optimiziung RxJS (Angular <= 4)

- Often the whole module «rxjs/Rx» is imported

```
import "rxjs/Rx";
```

- affects the size of the bundle (webpack)
- and the amount of requests to the server (system.js)

trivadis
makes IT easier.

Optimiziung RxJS

- Use lettable operators
- affects the size of the bundle (webpack)

```
this.service.update(employee).pipe(
        filter(newEmpl => newEmpl != null),
        tap(newEmpl => {
          // do something without returning a value
        }),
        map(newEmpl => new action.UpdateEmplSuccess(newEmpl)),
        catchError(error => {
          console.log(error);
          return of(new actions.UpdateEmplFail(error));
        })
      );
```


Build with Angular CLI

- Compile your app with the Angular CLI

  (JIT; since CLI 1.5 AOT default)

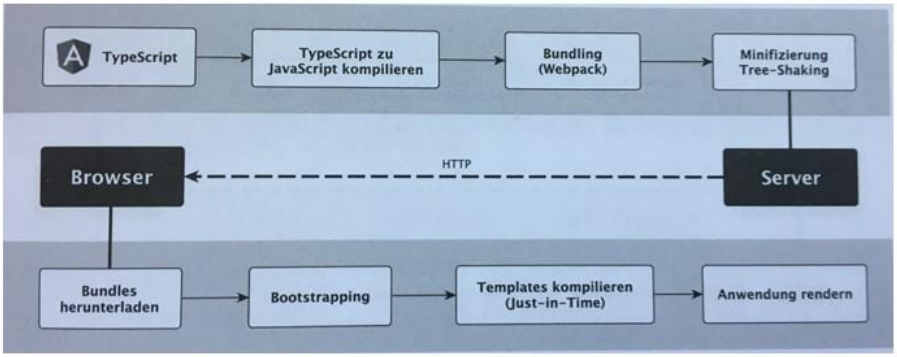
```
ng build                          #dev build
```

- Compile your app with the Angular CLI (AOT)

```
ng build --prod                   #prod build
```


Just in Time Compilation (JIT)

- Compilation is done during execution of our app



Woiwode et al. (2017)
https://angular-buch.com/

trivadis makes IT easier.

Angular in JIT Mode

```
// The browser platform with a compiler
import { platformBrowserDynamic }
       from '@angular/platform-browser-dynamic';


// The app module
import { AppModule } from './app.module';


// Compile (JIT) and launch the module
platformBrowserDynamic().bootstrapModule(AppModule);
```


trivadis makes IT easier.

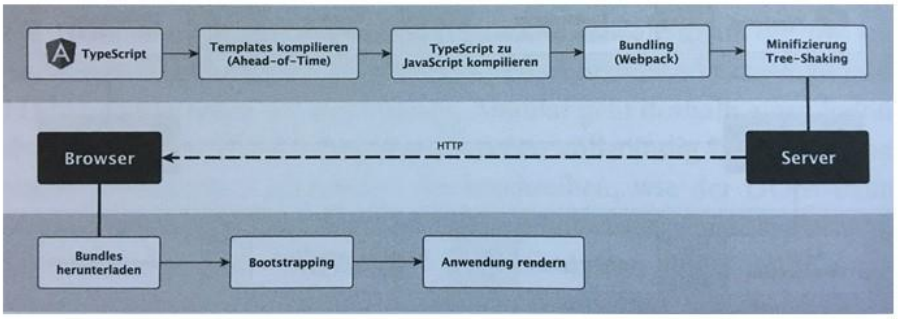
JIT Drawbacks

- Bundle size
- Slower rendering
- Errors in template can only be caught at runtime
- Security

The Angular 2 Compiler by Tobias Bosch (at Angular Connect 2016)
https://www.youtube.com/watch?v=kW9cJsvcsGo

Ahead of Time Compilation (AOT)

- Compilation is done during building our app



Woiwode et al. (2017)
https://angular-buch.com/

Angular in AoT Mode

```
// The browser platform with a compiler
import { platformBrowser } from '@angular/platform-browser';

// The generated app factory (AOT)
import { AppModuleNgFactory }
       from './aot/app.module.ngfactory';

// Launch with the app module factory.
platformBrowser().bootstrapModuleFactory(
   AppModuleNgFactory
);
```


Build for production

```
ng build --prod                #prod build
```

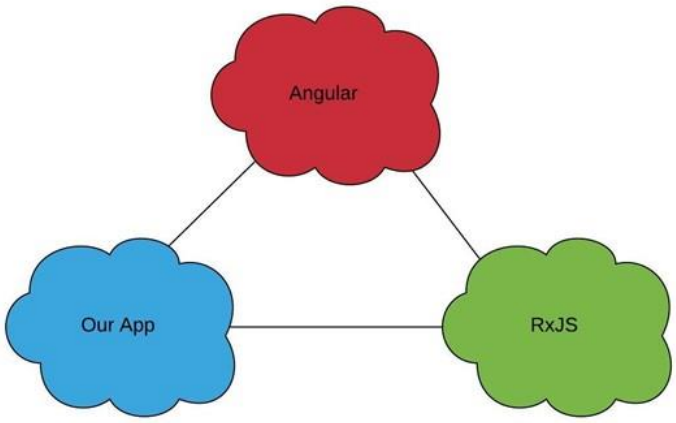
- **AOT** compilation
- **Minification**: Comments, line breakings are removed and variables are shorten
- **(limited) Tree-Shaking:** Drop unused module export code
- **Hashing of files**
- **Source Maps:** false

Minification

```
webpackJsonp([1,5],{"/fcW":function(l,n){function u(l){throw new Error("Cannot find module '"+l+"'.")}u.keys=function(){return[]},
u.resolve=u,l.exports=u,u.id="/fcW"},1:function(l,n,u){l.exports=u("x35b")},"1A80":function(l,n,u){"use strict";function t(l)
{return m._24(0,[(l()(),m._25(0,null,null,2,"a",[["class","navbar-brand"],["routerLink","/registrations"]],[[1,"target",0],[8,
"href",4]],[[null,"click"]],function(l,n,u){var t=!0;if("click"===n){t=!1!==m._26(l,1).onClick(u.button,u.ctrlKey,u.metaKey)&&t}
return t},null,null)),m._27(335872,null,0,g.y,[g.j,g.v,h.f],{routerLink:[0,"routerLink"]},null),(l()(),m._28(null,["Swiss
Microsoft CoWorking Space"]))],function(l,n){l(n,1,0,"/registrations")},function(l,n){l(n,0,0,m._26(n,1).target,m._26(n,1).href)})
}function e(l){return m._24(0,[(l()(),m._25(0,null,null,2,"a",[["class","navbar-brand"],["routerLink","/login"]],[[1,"target",0],
[8,"href",4]],[[null,"click"]],function(l,n,u){var t=!0;if("click"===n){t=!1!==m._26(l,1).onClick(u.button,u.ctrlKey,u.metaKey)&&
t}return t},null,null)),m._27(335872,null,0,g.y,[g.j,g.v,h.f],{routerLink:[0,"routerLink"]},null),(l()(),m._28(null,["Swiss
Microsoft CoWorking Space"]))],function(l,n){l(n,1,0,"/login")},function(l,n){l(n,0,0,m._26(n,1).target,m._26(n,1).href)})}
function i(l){return m._24(0,[(l()(),m._25(0,null,null,9,"div",[["style","color: white;\n        padding: 15px 50px 5px 50px;\n
    float: right;\n        font-size: 12px;"]],null,null,null,null,null)),(l()(),m._28(null,[" Last access:\n      "])),(l()(),
m._25(0,null,null,2,"time",[],null,null,null,null,null)),(l()(),m._28(null,["",""])),m._29(2),(l()(),m._28(null,["   "])),(l()(),
m._25(0,null,null,2,"a",[["class","btn btn-danger square-btn-adjust"],["routerLink","/logout"]],[[1,"target",0],[8,"href",4]],[
[null,"click"]],function(l,n,u){var t=!0,e=l.component;if("click"===n){t=!1!==m._26(l,7).onClick(u.button,u.ctrlKey,u.metaKey)&&t}
if("click"===n){t=!1!==e.authService.logout()&&t}return t},null,null)),m._27(335872,null,0,g.y,[g.j,g.v,h.f],{routerLink:[0,
"routerLink"]},null),(l()(),m._28(null,["Logout"])),(l()(),m._28(null,["\n    "]))],function(l,n){l(n,7,0,"/logout")},function(l,
n){var u=n.component;l(n,3,0,m._30(n,3,0,l(n,4,0,m._26(n.parent,0),u.authService.lastAccess,"dd.MM.yy HH:mm"))),l(n,6,0,m._26(n,7)
.target,m._26(n,7).href)})}function r(l){return m._24(0,[(l()(),m._25(0,null,null,8,"li",[],null,null,null,null,null)),(l()(),
```

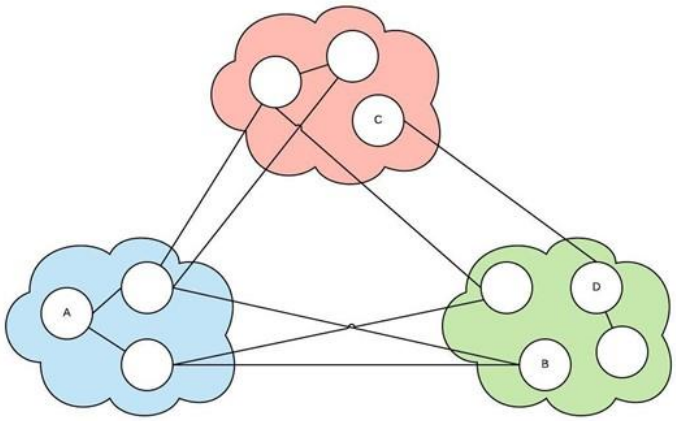

trivadis makes IT easier.

Tree-Shaking



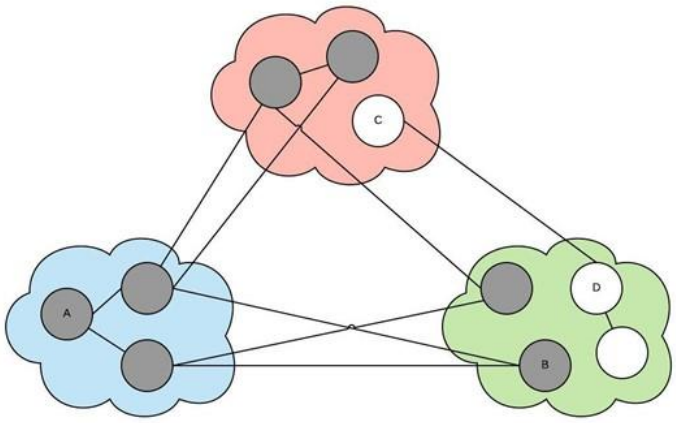
trivadis makes IT easier.

Tree-Shaking



trivadis makes IT easier.

Tree-Shaking



trivadis makes IT easier.

Bundles – Output of build

main.bundle.js	Source from our app
vendor.bundle.js	External libraries (Angular, RxJS)
inline.bundle.js	Logic for loading our bundles
styles.bundle.js	CSS Styles (only on DEV build)

trivadis makes IT easier.

Source Maps

- With source maps the size of the bundle can be analyzed.
- Install source-map-explorer

```
npm install -g source-map-explorer
```

- Run viusalizer

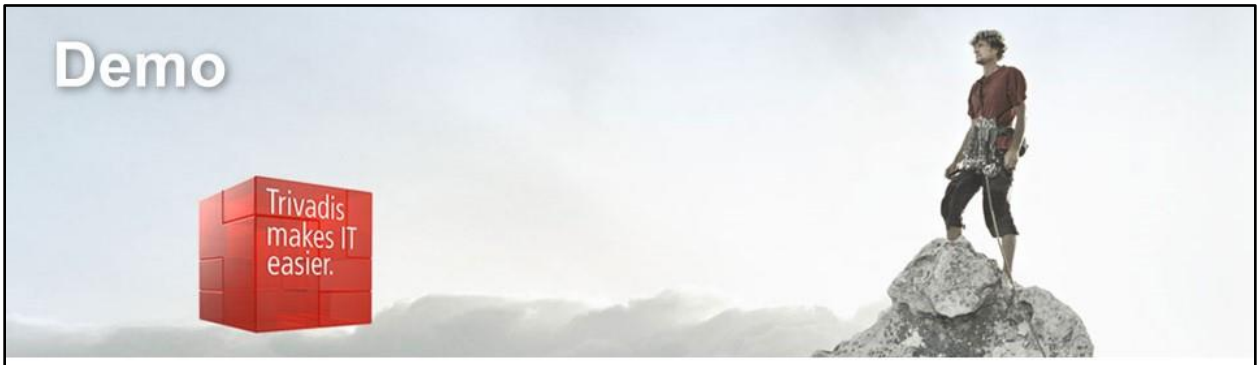
```
source-map-explorer vendor.bundle.js
```


trivadis makes IT easier.

Summary

- Use TsLint during development
- When dealing with RxJs use new lettable operators (e.g. pipe)
- Angular CLI helps to create a dev or prod build
- Always use a prod build for production `ng build --prod`

trivadis makes IT easier.

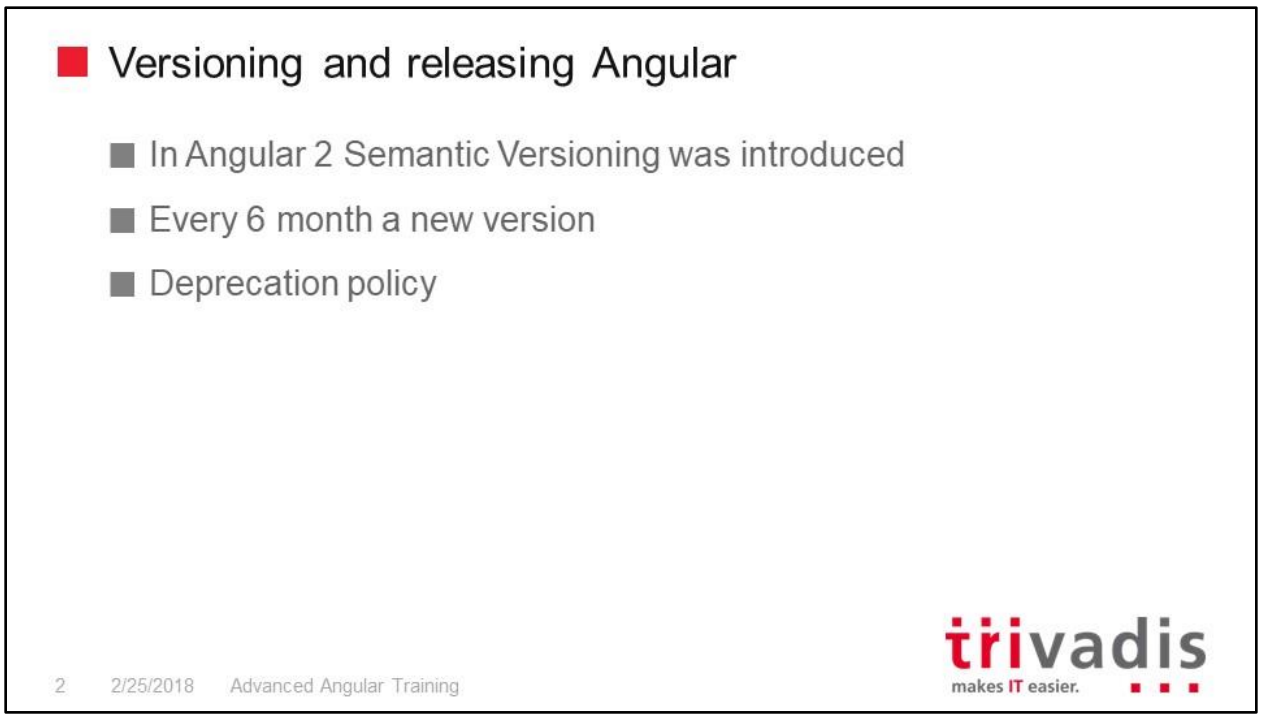


Build your app

Play with setting

trivadis makes IT easier.

13_WhatsNew



What's new in Angular 4

- Performance boost
- Angular Universal
- TypeScript's StrictNullChecks compliancy
- Stand alone animation module
- ngIf supports now else
- SEO optimizations
- httpClient (since v.4.3)
- …

trivadis makes IT easier.

What's new in Angular 4

```
<ng-template #hidden>
  <p>You are not allowed to see our secret</p>
</ng-template>
<p *ngIf="shown; else hidden">
  Our secret is being happy
</p>
```


trivadis makes IT easier.

What's new in Angular 4

```
//new in Angular 4 for SEO
this.title.setTitle(`TechEvent - ${this.pageTitle}`);
this.meta.addTag({
    name: "Description",
    content: `TechEvent - ${this.pageTitle}`
});
```


trivadis makes IT easier.

What's new in Angular 5

- Extended PWA Support
- Performance Boost
- Angular CLI: AoT default Build
- Forms *updateOn mechnism*
- I18N Pipes
- New Router events (ActivationStart, ActivationEnd, ChildActivationStart, ChildActivationEnd)
- …

trivadis makes IT easier.

Angular Labs

- Schematics
- Component Dev Kit
- ABC—The ABC (Angular + Bazel + Closure)
- Angular Elements

trivadis makes IT easier.

Summary

- Two major release per year
- Performance is getting better and better
- Angular CLI is one of the best starting points
- Schematics are super powerful

trivadis makes IT easier.

14_Course_Outro

Useful resources

- https://m.trivadis.com/angular
- https://github.com/TrivadisCloud/Angular-Playground
- https://marketplace.visualstudio.com/items?itemName=trivadis.ngtvd-extensions
- https://swissangular.com/

trivadis
makes IT easier.

Feedback

We are very grateful for your feedback.

- What was good?
- What could be improved?
- Something else?

trivadis makes IT easier.

Feedback

Link on your Desktop ➔ Trivaluation

trivadis makes IT easier.

Feedback

https://www.metricsthatmatter.com/trivad86/onsite

trivadis makes IT easier.

Thank you

Thomas Gassmann
(@gassmannT)

thomas.gassmann@trivadis.com

trivadis makes IT easier.