Bernhard Niedermayer

Software Development
bernhard.niedermayer@catalysts.cc
playing around with Angular2 since alpha.something

Angular2

Going Mobile with common web technologies

TypeScript

JavaScript just got better

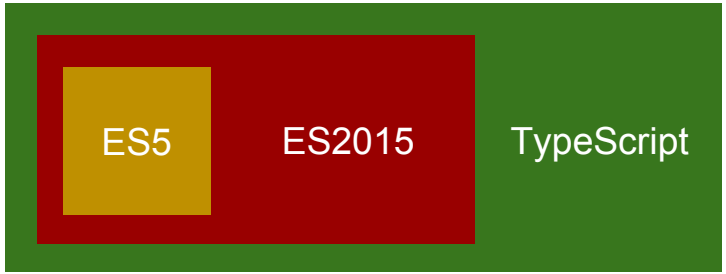
Languages

how they relate to one another

TypeScript adds types and annotations to **ES6**/**ES2015**

ES6/**ES2015** adds classes, block scoped variables, fat arrow functions, template strings and generators to **ES5**

ES5 is the version of **JavaScript** you're currently using and loving ;)



Transpiling

how TypeScript becomes JavaScript

Transpiling refers to the action of taking a source in a specific language and converting it into the source (as opposed to bytecode) of another language.

There are multiple examples of such transpiling operations:

LESS / **SASS** / **SCSS**	=>	**CSS**
TypeScript / **ES2016** **Dart** / **CoffeeScript**	=>	**JavaScript** / **ES5**
Markdown	=>	**HTML**
C#	=>	**Java**

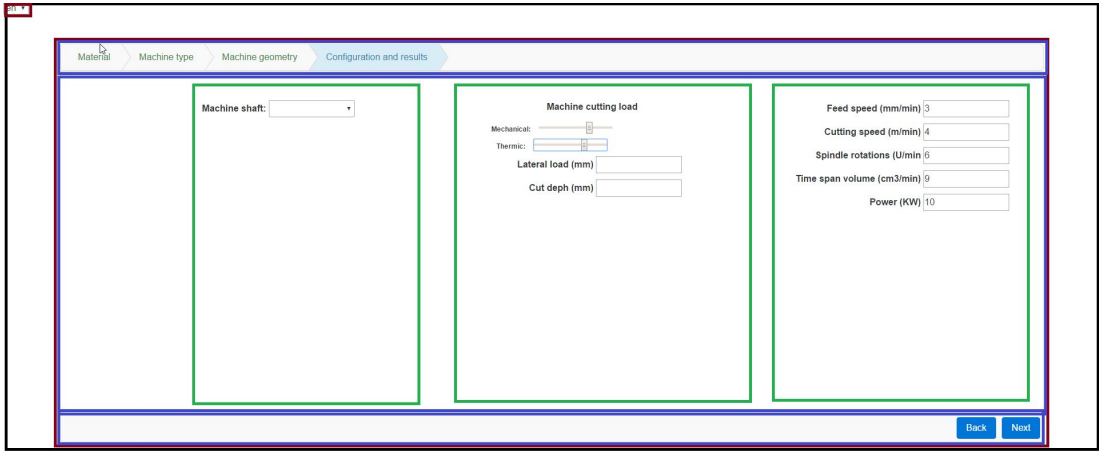
Components

The central concept of your Angular2 application!

Hierarchie

how to work with components

Components are your central building block of any **Angular2** application. They declare inputs and outputs and are structured in a tree, meaning a single component can have multiple child components.



Code example

overview

A component consists of 3 parts:

- A component decorator
  @Component()
- A view
  template: ``
- A controller
  class HelloWorld {}

```
import {bootstrap} from 'angular2/platform/browser';
import {Component, Input} from 'angular2/core;

@Component({
    selector: 'hello-world',
    directives: [],
    template: `
<div class="title" *ngIf="name">Hello {{name}}!</div>
<div class="title" *ngIf="!name">Hello world!</div>
`
})
class HelloWorld {
    @Input() name: String;
}

bootstrap(HelloWorld);
```

Code example

imports

```
import {bootstrap} from 'angular2/platform/browser';
```

ES2015 comes with module support replacing current solutions like **AMD**, **CommonJS** and **global** scripts. **Angular2** (when written in **ES2015** or **TypeScript**) leverages this functionality and exports several Components, helper functions and constants.

Code example

@Component

Note: **template** and **templateUrl** are mutually exclusive!

```
@Component({
    selector: 'hello-world',
    directives: [],
    providers: [],
    viewProviders: [],
    pipes: [],
    styles: '',
    styleUrls: [],
    template: `
<div class="title" *ngIf="name">Hello {{name}}!</div>
<div class="title" *ngIf="!name">Hello world!</div>
`,
    templateUrl: undefined
})
```

Code example

Controller / class

Controllers in Angular2 are represented by classes. A class can have public and private properties and functions as well as a constructor.

```
class HelloWorld {
    @Input() name: String;
}
```

```
class HelloWorld implements OnInit {
    name: String;

    constructor(name?: String) {
        this.name = name || 'World';
    }

    onInit() {
        // lifecycle callback function
    }
}
```

Code example

bootstrapping

```
bootstrap(HelloWorld);
```

By calling **bootstrap** you tell Angular which **component** should be considered the root of your application.

Directives

...where have they gone?

```
import {Directive} from 'angular2/core';
@Directive({
    selector: 'button[click-logger]',
    host: {
        '(click)': 'onClick($event.target)'
    }
})
export class ClickLogger {
    onClick(target) {
        console.log(target, 'clicked');
    }
}
```

```
<button click-logger>Click me</button>
```

Also, built in directives, e.g.

NgIf, NgFor, NgSwitch,
NgSwitchWhen, NgSwitchDefault,
NgStyle, NgClass

A component is a special type of directive **having** a **view**.

Databinding, Services, Pipes, etc...

Databinding / Template-Syntax

Property and Event Bindings

event bindings use parenthesis ()

property bindings use brackets []

two-way data binding uses both [()]

interpolation with curly braces {{}}

template private variable using #

```
<div class="item"
     *ngFor="#item in items"
     (click)="selectItem(item);">
    <div>{{item.name}}</div>
    <img [src]="item.imageSrc">
</div>

<form class="new-item" (submit)="addNewItem()">
    <input #myItem [(ng-model)]="newItem">
</form>
```

DI

creating a service

```
import {Injectable} from 'angular2/core;

@Injectable()
export class NameService {
    names: Array<String>;

    constructor() {
        this.names = ['John', 'Jack']
    }

    getName() {
        let idx = Math.floor(Math.random() * this.names.length);
        return this.names[idx];
    }
}
```

DI

using a service

```
import {bootstrap} from
'angular2/platform/browser';
import {HelloWorld} from './hello-world';
import {NameService} from './name-service';

bootstrap(HelloWorld, [NameService]);
```

```
import {Component} from 'angular2/core';
import {NameService} from './name-service';

@Component({
    selector: 'hello-world',
    template: `
            <div class="title">Hello {{name}}!
</div>
      `
})
class HelloWorld {
    name: String;

    constructor(nameService: NameService) {
        this.name = nameService.getName();
    }
}
```

Pipes

Using Pipes

transforms data for display

instead of just using the raw toString()

parametrizable

built in pipes:

DatePipe, UpperCasePipe, LowerCase Pipe, CurrencyPipe PercentPipe, JsonPipe, AsyncPipe

```
@Component({
    selector: 'hello-world',
    pipes: [],
    template: `
      <p> {{conference | json}}</p>
      <p>
      {{ conference.tracks[0].date | date:'fullDate' | uppercase }}
      </p>
      `
})
```

Pipes

Creating your own Pipes

@Pipe annotation

PipeTransform interface

```
import {Component} from 'angular2/core';
import {TopConfPipe} from './pipes/topconf-pipe';

@Component({
    selector: 'my-component',
    pipes: [TopConfPipe]
})
```

```
import {Pipe, PipeTransform} from 'angular2/core';

@Pipe({name: 'topConf'})
export class TopConfPipe implements PipeTransform {
    transform(value: string, args: string[]): string {
        return value + ' (btw... TopConf rulez!)' +
            (args[0] || '');
    }
}
```

Far from the end....

Form-Controls, Router, RxJS, ...

Angular offers a lot more features than covered in this short presentation, among these are:

- **Form-Controls**
  highly sophisticated support for form handling including validation
- **Router**
  now allowing multiple parallel nested routes (all mapped to the url on demand)
- **RxJS**
  support for RxJS is included within the core of Angular2 (Observables instead of Promises)

Angular2

in real-world projects

Transpiling revisited

CI for TypeScript

Transpiling and packaging of TypeScript projects as part of a gradle build

1) get node
2) npm install
3) gulp install

Gulp task for transpiling

- require gulp-typescript
- (require gulp-sourcemaps)
- create and compile project

```
plugins {
    id "com.moowork.gulp" version "0.11"
}
node {
    version = '5.4.1'
    npmVersion = '3.5.3'
    download = true
}
gulp_build.dependsOn 'npmInstall'
gulp_build.dependsOn 'installGulp'
build.dependsOn gulp_build
```

Testing

Unit Tests, End-to-end Tests

Jasmine + Karma

Angular2 Unit Testing framework built upon Jasmin

protractor

```
describe('HelloWorldService', () => {
  it('says hello world', () => {
    let helloWorldService = new HelloWorldService();
    expect(helloWorldService.hi()).toEqual('Hello World!');
  });
});
```

```
describe('schedule', function() {
  it('should display all talks', function() {
    browser.get('http://topconf.com/linz-2016/schedule/2016-02-02/');
    var talkList = element.all(by.repeater('talk in talks'));
    expect(talkList.count()).toEqual(13);
  });
});
```

NativeScript

Going Mobile with Javasript, CSS, and XML

NativeScript

- uses JavaScript VM V8 (Android) and JavaScriptCore (iOS)
- 2-way data binding between js and native components
- css implementation for native components
- difference to Ionic, Xamarin, ReactNative
  - not based upon a DOM
  - no cross-compilation
  - possibility to call native APIs

NativeScript

Hello World ;)

NativeScript CLI

```
tns create
tns platform add android

… implement

tns run android

… iterate
```

```
var applicationModule = require("application");
applicationModule.mainModule = "views/hello-world";
applicationModule.start();
```

```
<Page>
   <Label text="Hello, world!"/>
</Page>
```

Native Code...

...called by JavaScript

JIT compiled

for Android, global object `android` gets injected into V8

native Java code called via reflection, proxy objects on JavaScript side

Android JNI used to also call C++ code of the NativeScript runtime

```
var time = new android.text.format.Time();
```

NativeScript + Angular

NativeScript + Angular

Rendering mechanism

- AngularJS 1 was bound to the DOM
- Angular2 runtime consists of two layers
  - Application layer
  - Rendering layer
- Every component can define its own render
- Different render implementations
  - DomRenderer
  - NativeScriptRenderer
- css implementation for native components

```
import { nativeScriptBootstrap } from
       "nativescript-angular/application";
import {MainPage} from "./main-page";

nativeScriptBootstrap(MainPage);
```

```
"dependencies": {
  "tns-core-modules": "1.6.0-angular-1",
  "nativescript-angular": "0.0.26",
  "angular2": "2.0.0-beta.1",
  …
}
```