Herzlich Willkommen!

Gleich geht es los.

Angular - Der professionelle Einstieg

Dozent: Peter Hecker

Peter Hecker

- Seit 1986: „Entwickler, Trainer“
- Seit 1995: „Web-Entwicklung“
- Seit 2009: „Mobile WebApps“
- @phecker65

Themen

- Angular - Einführung
- Angular - Einstieg
- Angular - Components
- Angular - Templating
- Angular - Data Binding, Pipes
- Angular - Dependency Injection, Services
- Angular - HTTP, RxJS
- Angular - Router, Navigation
- Angular - Forms, Validation

Ihre Themen

Angular - Einführung

Angular

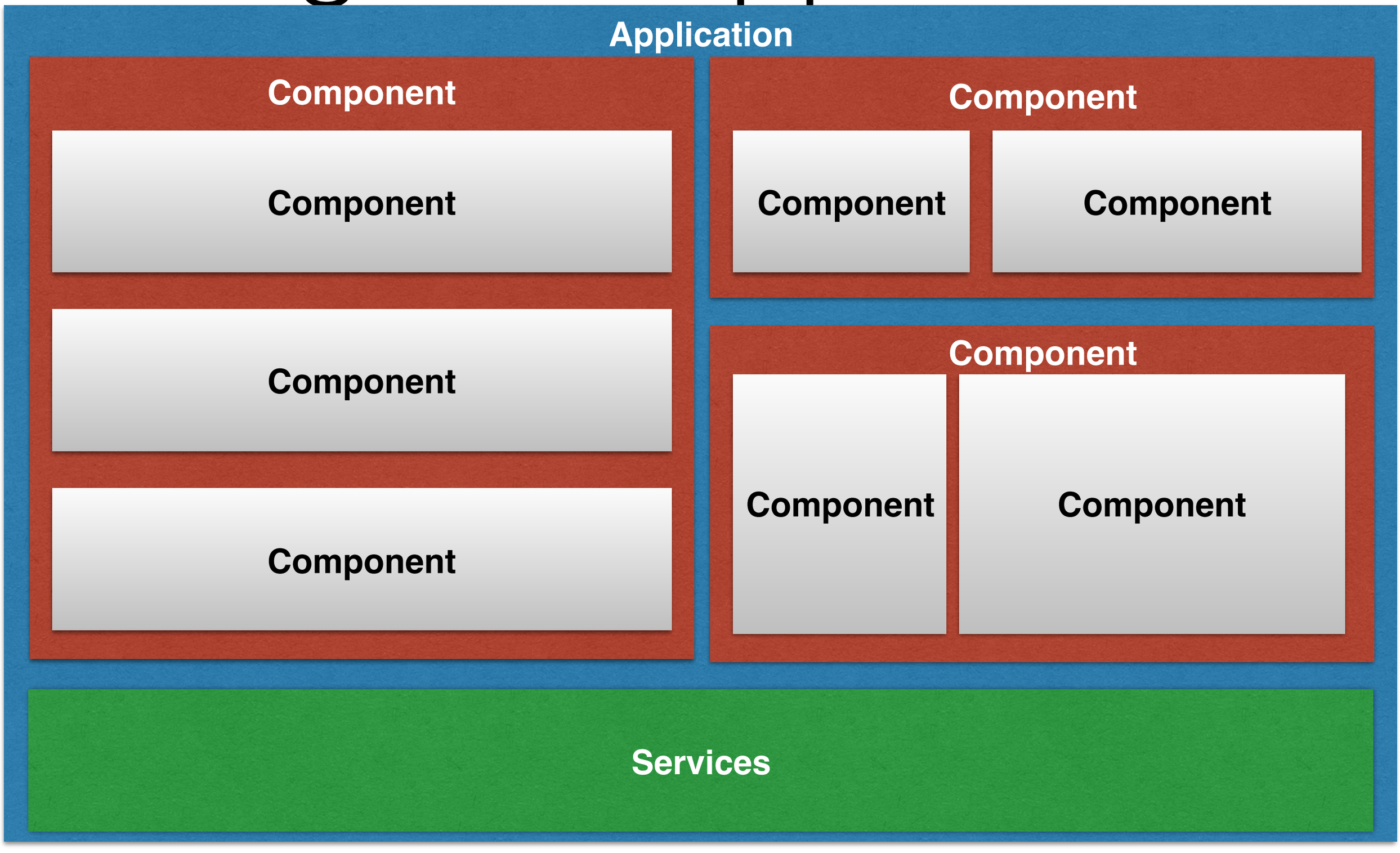
One **Framework**
for **Cross Platform Development**
(Mobile, Desktop, Native)
with **Web-Technologies**
(HTML, CSS, JS)

Angular - Designziele

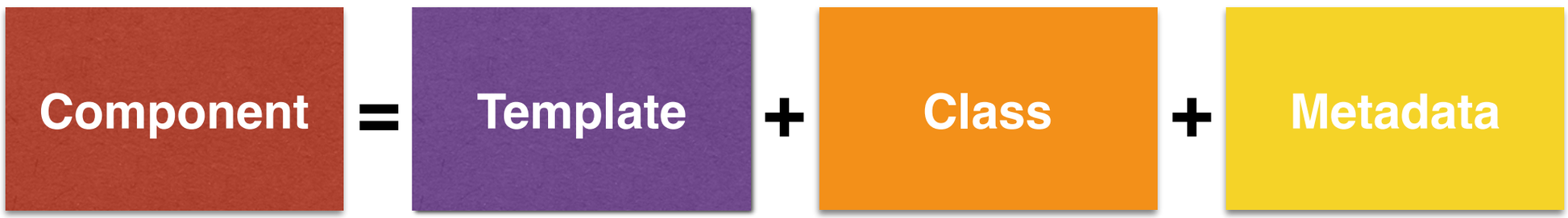
- Modularität
- Einfache API
- Data Binding
- Back-End Integration
- Ausdrucksvoll
- Geschwindigkeit
- Produktivität

Angular - Application

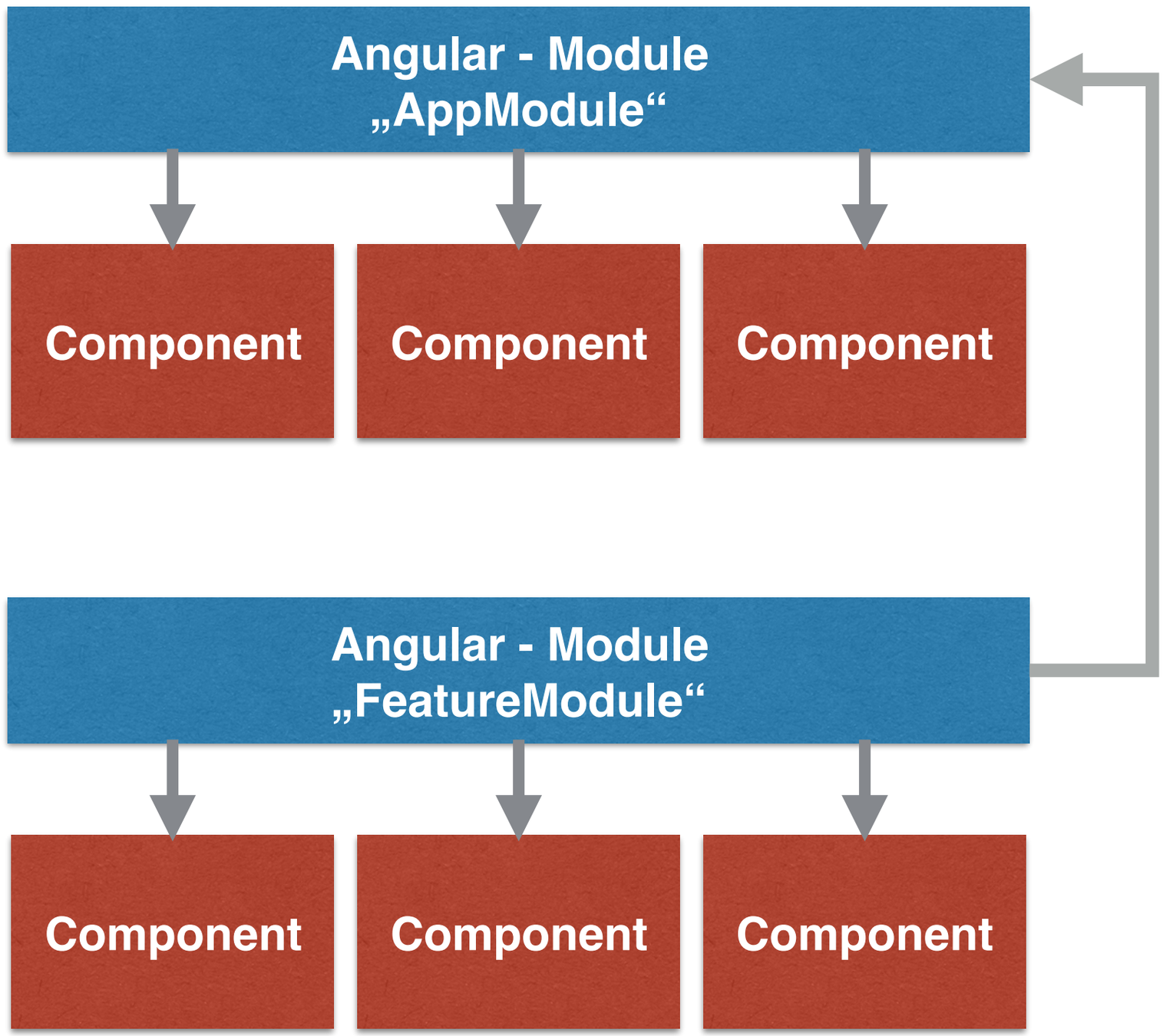
Angular - Application



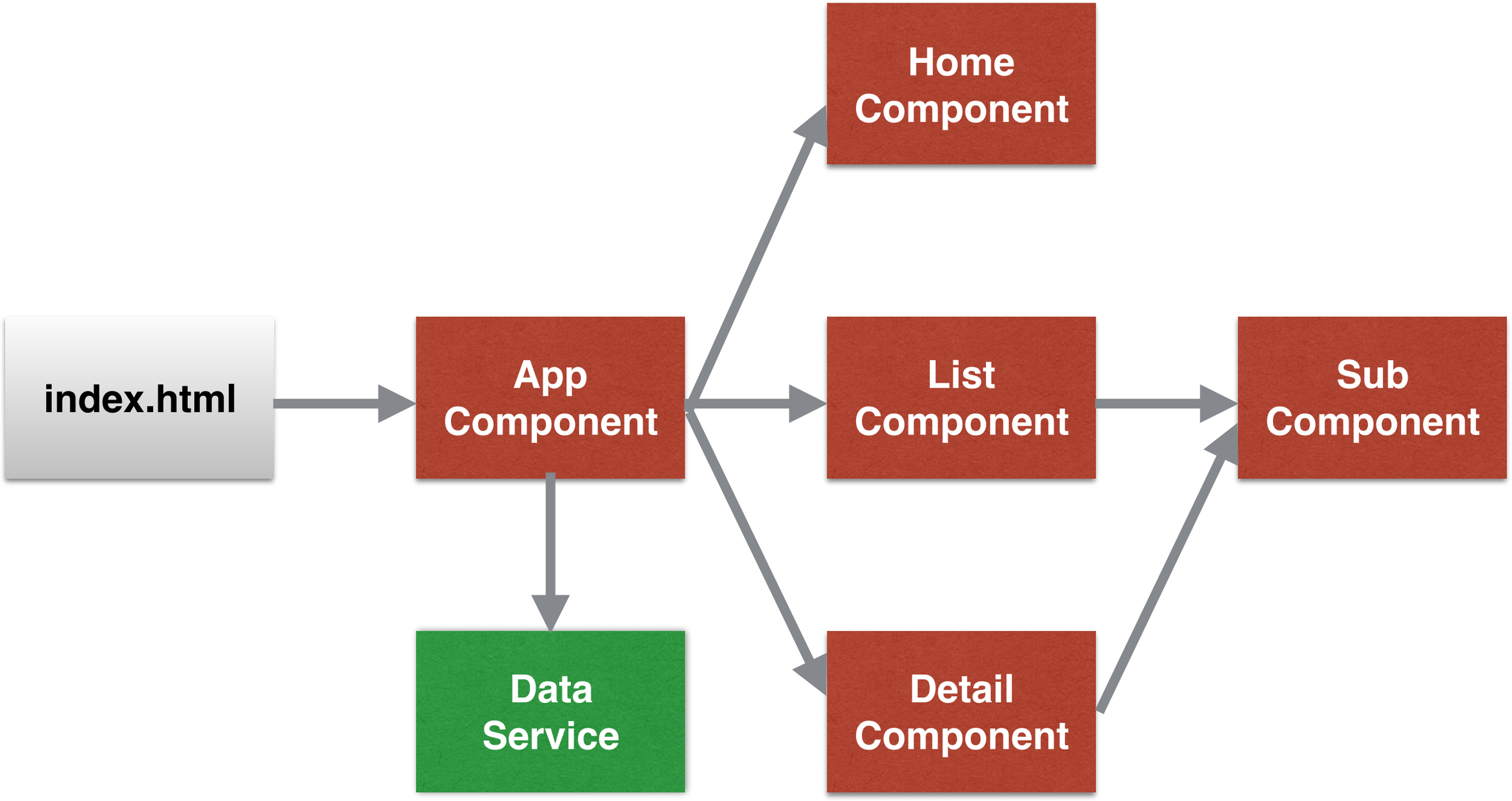
Angular - Component



Angular - Module



Seminarbeispiel



Zusammenfassung

- Angular - Designziele
- Angular - Application
- Angular - Components
- Angular - Seminarbeispiel

Angular - Einstieg

Angular - Sprachen

SPRACHEN

ES5

ES6/ES7
(ES2015/ES2016)

→ Transpiler:
Traceur, Babel

TypeScript

→ TypeScript-
Compiler

Dart

→ Dart-
Compiler

STANDARDS

ECMAScript 3 (ES3) → ECMAScript 5 (ES5) → ECMAScript 6 / 2015
(ES6, ES2015) → ECMAScript 7
(ES7, ES2016)

PLATTFORMEN

TypeScript im Überblick

- Unterstützt Standard JavaScript Code (ES3, ES5, ES6)
- Bietet Statische Typisierung
- Kapselung durch Module und Klassen
- Unterstützt Konstruktoren, Eigenschaften, Funktionen
- Definition von Schnittstellen
- Lambdas oder Array-Funkionen
- Intellisense und Syntaxüberprüfung durch Werkzeuge

TypeScript Features

- Static Typing
- Interfaces
- Classes (Properties, Methods)

Static Typing

```
let name:string
let age:number
let birthDate:date
```

Interfaces

```
interface ICat {
 name:string
 age:number
}
```

Classes

```
class Cat {
  name:string;
  color:string;

  constructor (name, color) {
    this.name = name;
    this.color = color;
  }
  speak() {
    console.log('meow')
  }
}
```

Werkzeuge:
Type Defintion Files, Typings, TSLint

- Develop
- Type Definition Files
- Typings
- TSLint

DefinitelyTyped

The repository for high quality TypeScript type definitions

Usage

npm

This is the preferred method. This is only available for TypeScript 2.0+ users. For example:

```
npm install --save-dev @types/jquery
```

The types should then be automatically included by the compiler. See more in the handbook.

Triple-Slash Directives

Download a declaration file from the repository and include a line like this:

```
/// <reference path="jquery/jquery.d.ts" />
```

Get the definitions

Contributing

News

DefinitelyTyped

http://definitelytyped.org/

Typings

The TypeScript Definition Manager

> The TypeScript Definition Manager.

Deprecation Notice: Regarding TypeScript@2.0

For users doing `typings install dt~<package> --global` and receiving errors.

Starting from TypeScript 2.0, users can install typings using `npm install @types/<package>`. These typings are coming from DefinitelyTyped. In the future, we hope redirects will be enabled to support existing maintainers to contribute effectively to NPM's `@types` as they did to `typings/registry`.

Typings on DefinitelyTyped have also moved to the external module format supported by TypeScript. This finally solved the real problem that Typings was trying to solve! It also means it will cause errors such as:

```
> typings install dt~angular --global

typings ERR! message Attempted to compile "angular" as a global module,
but it looks like an external module. You'll need to remove the global option to continue.
```

Typings

https://github.com/typings/typings

TSLint

An extensible linter for the TypeScript language.

View on GitHub | Learn More | Download .tar.gz

TSLint is an extensible static analysis tool that checks TypeScript code for readability, maintainability, and functionality errors. It is widely supported across modern editors & build systems and can be customized with your own lint rules, configurations, and formatters.

Quick start

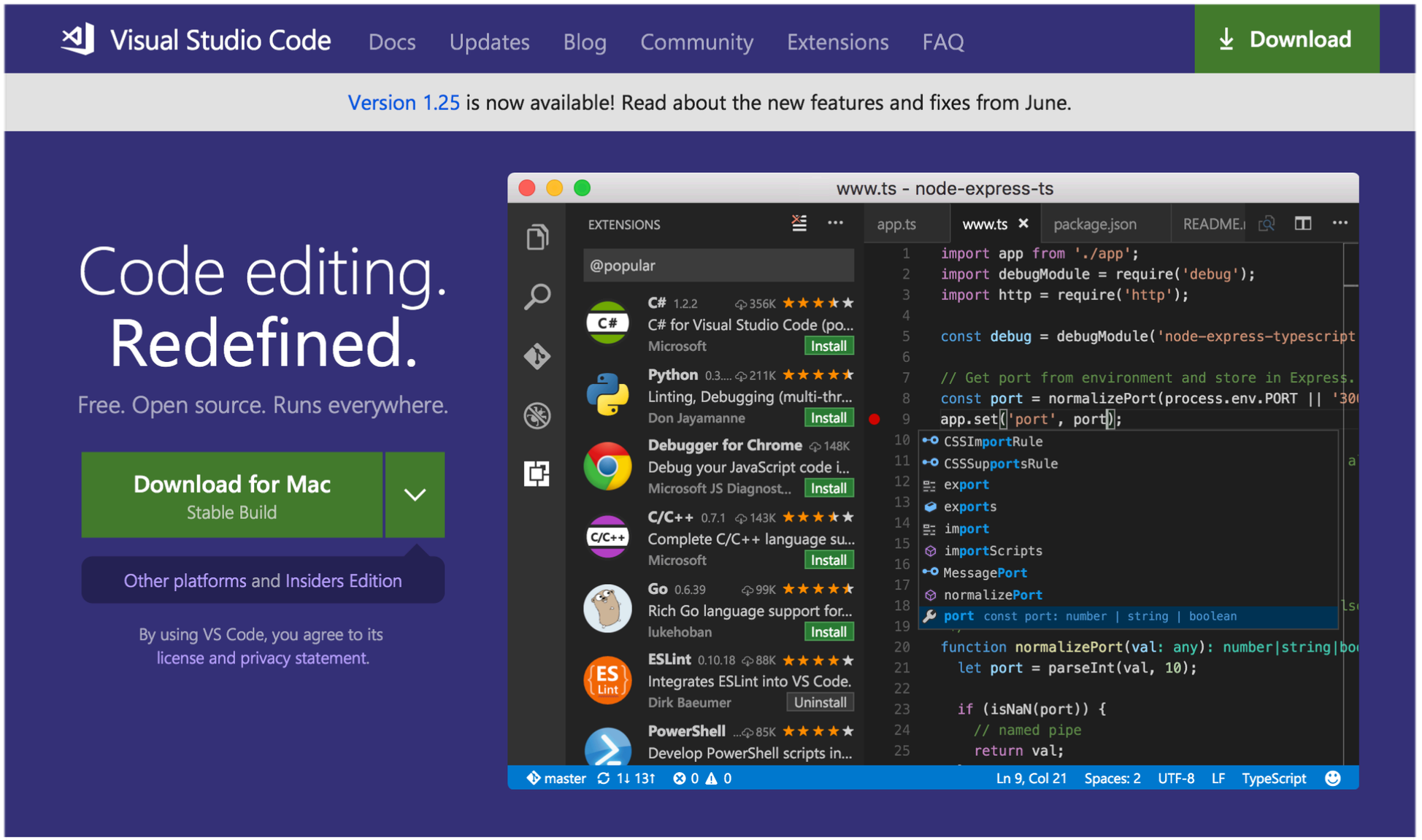
```
# Install the global CLI and its peer dependency
yarn global add tslint typescript

# Navigate to to your sources folder
cd path/to/project

# Generate a basic configuration file
tslint --init
```

TSLint

http://palantir.github.io/tslint/

IDE’s und Editoren

- IDE’s
  - Visual Studio, Eclipse, Netbeans, WebStorm
- Editoren
  - Visual Studio Code, Atom, Sublime



VS Code

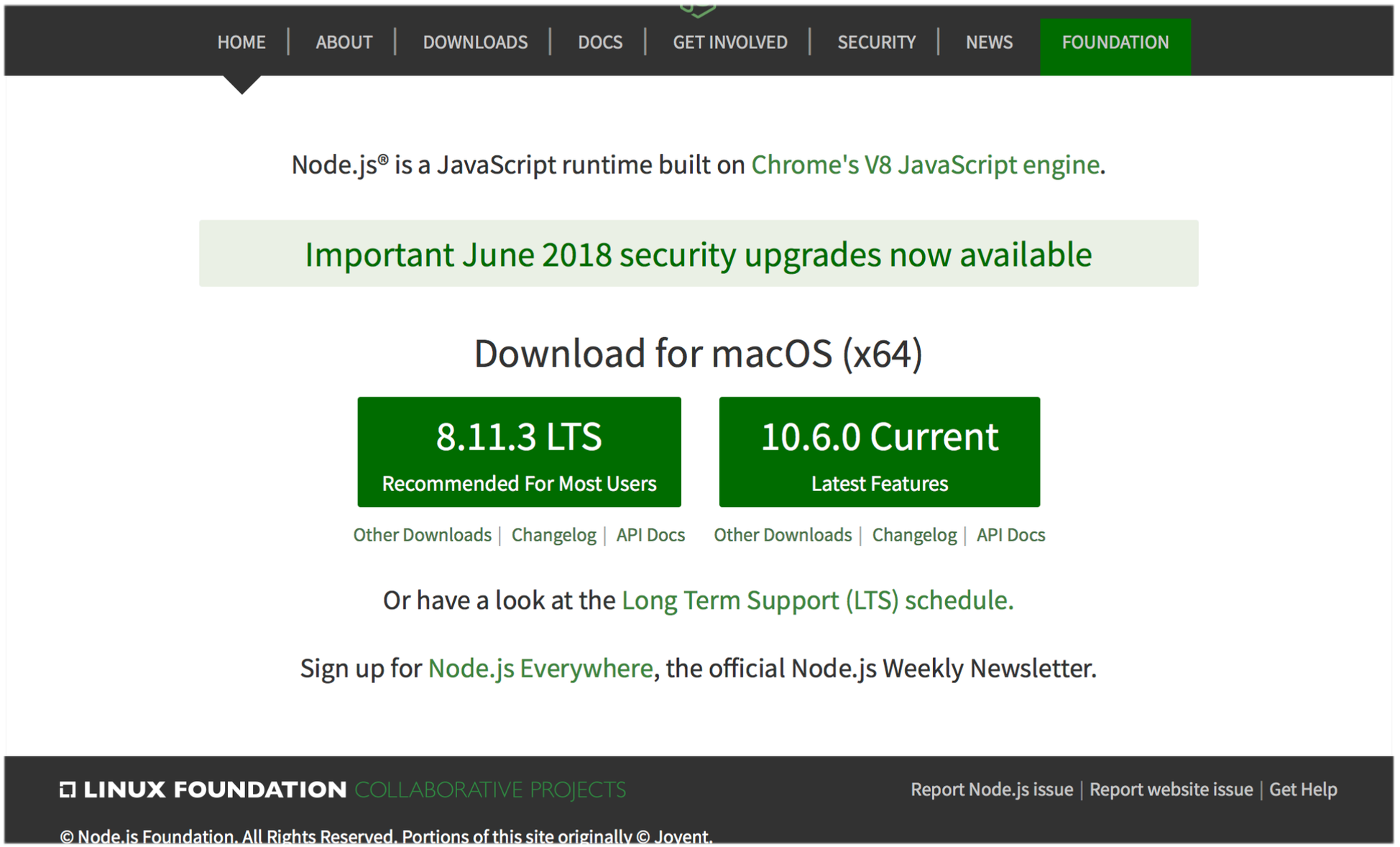
https://code.visualstudio.com/

Atom

https://atom.io/
https://atom.io/packages/atom-typescript

Konventionen: Style Guides

- https://github.com/johnpapa/angular-styleguide
- https://github.com/excelmicro/typescript



Node.js

https://nodejs.org/

Eine Angular - Anwendung erstellen

1. Anwendungsorder erstellen
2. **package.json, tsconfig.json, systemjs.config.js** erstellen
3. Bibliotheken und Typen installieren mit **npm install**
4. **app.module.ts (class AppModule)** erstellen
5. **app.component.ts (class AppComponent)** erstellen
6. **main.ts** („bootstrapper“) erstellen
7. **index.html** erstellen

```
{
  "name": "angular-quickstart",
  "version": "1.0.0",
  "description": "QuickStart package.json from the documentation, supplemented with testing support",
  "scripts": {
    "build": "tsc -p src/",
    "build:watch": "tsc -p src/ -w",
    "build:e2e": "tsc -p e2e/",
    "serve": "lite-server -c=bs-config.json",
    "serve:e2e": "lite-server -c=bs-config.e2e.json",
    "prestart": "npm run build",
    "start": "concurrently \"npm run build:watch\" \"npm run serve\"",
    "pree2e": "npm run build:e2e",
    "e2e": "concurrently \"npm run serve:e2e\" \"npm run protractor\" --kill-others --success first",
    "preprotractor": "webdriver-manager update",
    "protractor": "protractor protractor.config.js",
    "pretest": "npm run build",
    "test": "concurrently \"npm run build:watch\" \"karma start karma.conf.js\"",
    "pretest:once": "npm run build",
    "test:once": "karma start karma.conf.js --single-run",
    "lint": "tslint ./src/**/*.ts -t verbose"
  },
  "keywords": [],
  "author": "",
  "license": "MIT",
  "dependencies": {
    "@angular/common": "~2.4.0",
    "@angular/compiler": "~2.4.0",
    "@angular/core": "~2.4.0",
    "@angular/forms": "~2.4.0",
    "@angular/http": "~2.4.0",
    "@angular/platform-browser": "~2.4.0",
    "@angular/platform-browser-dynamic": "~2.4.0",
    "@angular/router": "~3.4.0",

    "angular-in-memory-web-api": "~0.2.4",
    "systemjs": "0.19.40",
    "core-js": "^2.4.1",
    "rxjs": "5.0.1",
    "zone.js": "^0.7.4"
  },
  "devDependencies": {
    "concurrently": "^3.2.0",
    "lite-server": "^2.2.2",
    "typescript": "~2.0.10",

    "canonical-path": "0.0.2",
    "tslint": "^3.15.1",
    "lodash": "^4.16.4",
    "jasmine-core": "~2.4.1",
    "karma": "^1.3.0",
    "karma-chrome-launcher": "^2.0.0",
```

package.json

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "lib": [ "es2015", "dom" ],
    "noImplicitAny": true,
    "suppressImplicitAnyIndexErrors": true
  }
}
```

tsconfig.json

```
/**
 * System configuration for Angular samples
 * Adjust as necessary for your application needs.
 */
(function (global) {
  System.config({
    paths: {
      // paths serve as alias
      'npm:': 'node_modules/'
    },
    // map tells the System loader where to look for things
    map: {
      // our app is within the app folder
      app: 'app',

      // angular bundles
      '@angular/core': 'npm:@angular/core/bundles/core.umd.js',
      '@angular/common': 'npm:@angular/common/bundles/common.umd.js',
      '@angular/compiler': 'npm:@angular/compiler/bundles/compiler.umd.js',
      '@angular/platform-browser': 'npm:@angular/platform-browser/bundles/platform-browser.umd.js',
      '@angular/platform-browser-dynamic': 'npm:@angular/platform-browser-dynamic/bundles/platform-browser-dynamic.umd.js',
      '@angular/http': 'npm:@angular/http/bundles/http.umd.js',
      '@angular/router': 'npm:@angular/router/bundles/router.umd.js',
      '@angular/forms': 'npm:@angular/forms/bundles/forms.umd.js',

      // other libraries
      'rxjs':                      'npm:rxjs',
      'angular-in-memory-web-api': 'npm:angular-in-memory-web-api/bundles/in-memory-web-api.umd.js'
    },
    // packages tells the System loader how to load when no filename and/or no extension
    packages: {
      app: {
        defaultExtension: 'js'
      },
      rxjs: {
        defaultExtension: 'js'
      }
    }
  });
})(this);
```

systemjs.config.js

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent }  from './app.component';

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

app.module.ts

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

@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>`,
})
export class AppComponent  { name = 'Angular'; }
```


app.component.ts

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

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

platformBrowserDynamic().bootstrapModule(AppModule);
```

main.ts

```
<!DOCTYPE html>
<html>
  <head>
    <title>Angular QuickStart</title>
    <base href="/">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="styles.css">

    <!-- Polyfill(s) for older browsers -->
    <script src="node_modules/core-js/client/shim.min.js"></script>

    <script src="node_modules/zone.js/dist/zone.js"></script>
    <script src="node_modules/systemjs/dist/system.src.js"></script>

    <script src="systemjs.config.js"></script>
    <script>
      System.import('main.js').catch(function(err){ console.error(err); });
    </script>
  </head>

  <body>
    <my-app>Loading AppComponent content here ...</my-app>
  </body>
</html>
```


index.html

```
h1 {
  color: #369;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 250%;
}
```

styles.css

Eine Angular - Anwendung erstellen

- Angular Quickstart
  - https://angular.io/guide/quickstart
- Github
  - https://github.com/angular/quickstart
- AngularCli
  - https://cli.angular.io/
- Angular Seed
  - https://mgechev.github.io/angular-seed/

JavaScript und Module

- Auf Sprachebene:
  - AMD, CommonJS, UMD, SystemJS
  - TypeScript
  - ES6 / ES2015
- Auf Anwendungsebene:
  - AngularJS
  - Angular

JavaScript und Module

	<script>	RequireJS	Browserify	SystemJS	TSC
AMD	-	X	-	X	X
CommonJS	-	-	X	X	X
UMD	-	-	-	X	X
SystemJS	-	-	-	X	X
ES6 / ES2015	-	-	-	X	X
Browser	X	X	X	X	X
NodeJS	-	-	-	X	X

TypeScript und Module

	TypeScript Design-Time	TypeScript Runtime
~~**External Modules**~~	~~X (< 1.5)~~	~~AMD, CommonJS, UMD, SystemJS~~
AMD	-	X
CommonJS	-	X
UMD	-	X
SystemJS	-	X
ES6 / ES2015	**X (>= 1.5)**	**X**

SystemJS

build passing | gitter join chat | support SystemJS 10%

Configurable module loader enabling dynamic ES module workflows in browsers and NodeJS.

SystemJS 0.20 release notes

- Loads any module format when running the ~15KB development build.
- Loads ES modules compiled into the `System.register` module format for production with exact circular reference and binding support
- Supports RequireJS-style map, paths, and bundles configuration.

Built with the ES Module Loader project, which is based on principles and APIs from the WhatWG Loader specification, modules in HTML and NodeJS.

Supports IE9+ provided a promises polyfill is available in the environment.

For discussion, join the Gitter Room.

Documentation

- Getting Started
- Module Formats
- Production Workflows

SystemJS

https://github.com/systemjs/systemjs

Ohne SystemJS

index.html

```
<script src="a.js"></script>
<script src="b.js"></script>
<script src="c.js"></script>
```

a.js

```
function a() {}
```

b.js

```
function b() {}
```

c.js

```
function c() {}
```

Mit SystemJS

index.html

```
<script src="system.js">
</script>
<script src="config.js">
</script>
```

config.js

```
System.config({
 paths: {},
 map: {},
 packages: {}
});
```

ES6 / ES2015 Module

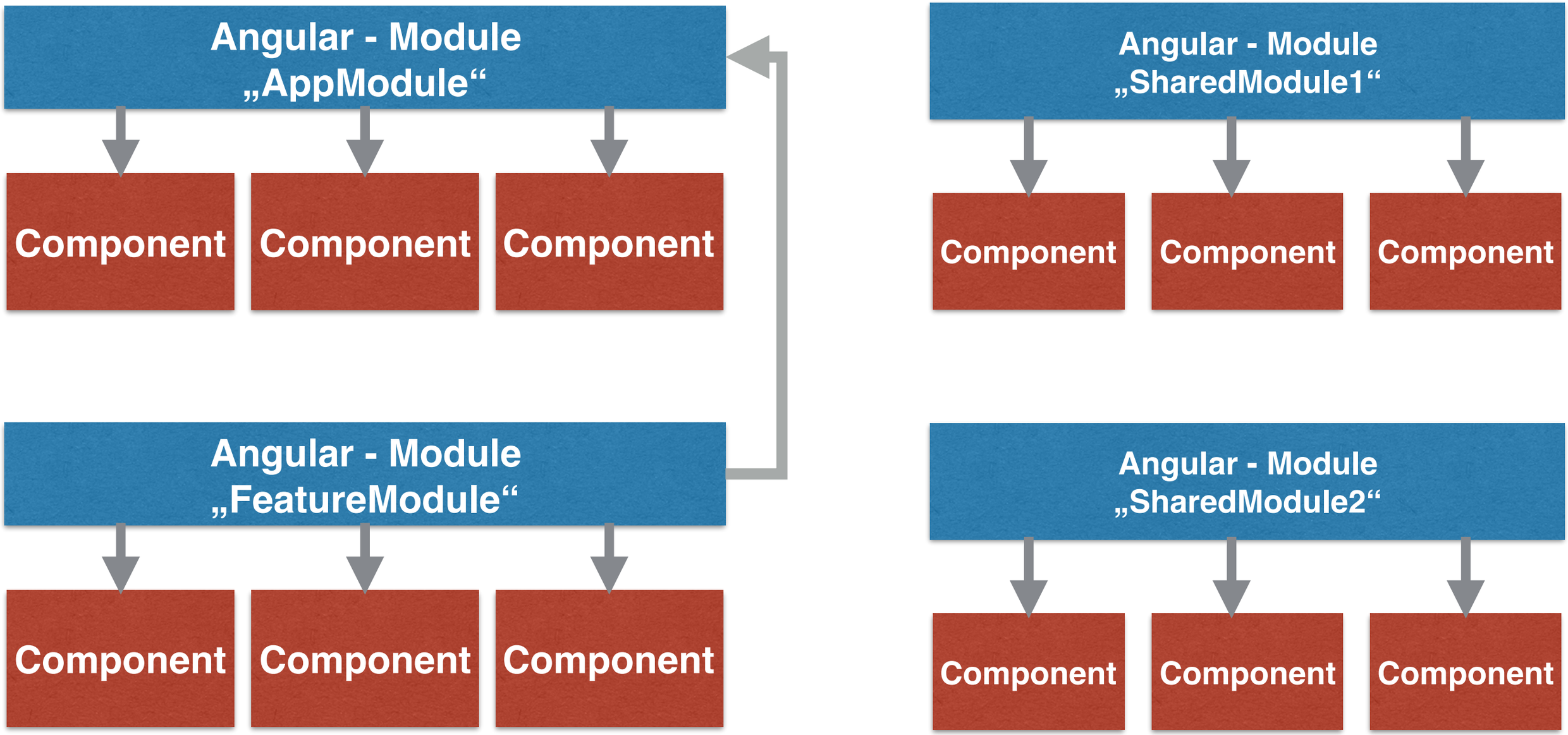
item.ts

```
export class Item {

}
```

item-list.ts

```
import { Item } from './item'


export class ItemList {

}
```

Angular Module



JavaScript und Module

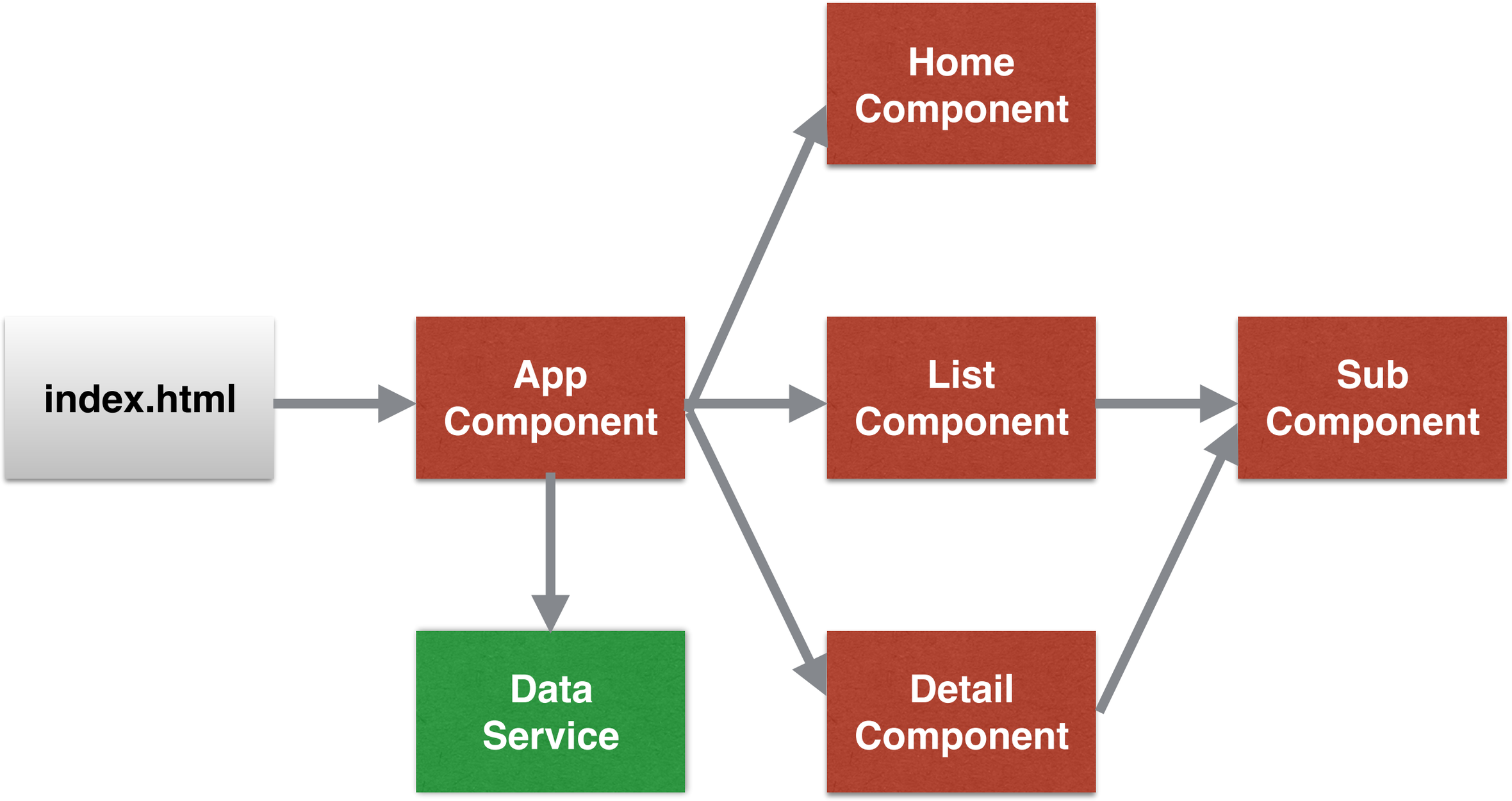
ES6 / ES2015 Module	Angular Module
Organisation des Codes	**Organisation der Anwendung**
Modularization des Codes	**Modularization der Anwendung**
Wiederverwendung von Code	**Wiederverwendung von Anwendungen**

Zusammenfassung

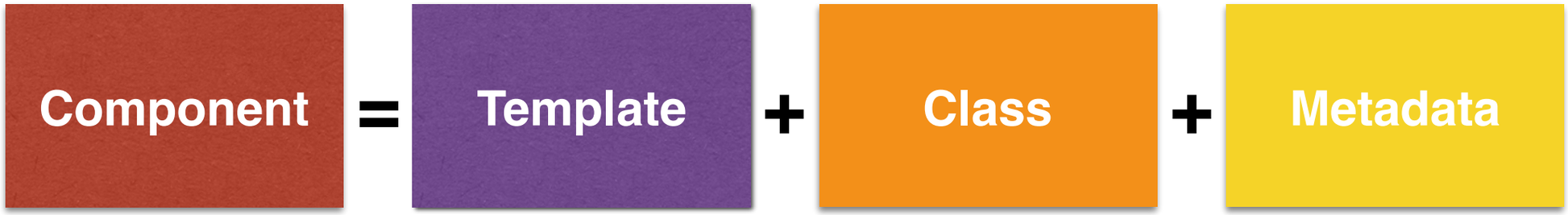
- Angular - Sprachen
- TypeScript im Überblick
- Werkzeuge
- IDE’s und Editoren
- Konventionen: Style Guides
- Eine Angular - Anwendung erstellen
- JavaScript, TypeScript, ES6 / ES2015, Angular und Module

Angular - Components

Seminarbeispiel



Angular - Component



- View
- HTML
- Binding
- Directives

- Code
- TS
- Properties
- Methods

- Angular
- Decorator

Component

- A class with the @Component decorator that associates it with a companion template.
  Together, the component and template define a view.

- A component is a special type of directive.
  The @Component decorator extends the @Directivedecorator with template-oriented features.

- An Angular component class is responsible for exposing data and handling most of the view's display and user-interaction logic through data binding.

https://angular.io/guide/glossary

Component

Import

Metadata

Class

```
app.component.ts

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

@Component({
    selector: 'sas-app',
    template: `
    <div>
        <h1>{{pageTitle}}</h1>
        <div>Hello from AppComponent!</div>
    </div>
    `
})

export class AppComponent {
    pageTitle: string = 'Simple Angular Sample';
}
```

Import

- Angular apps are modular:
  - @angular/core
  - @angular/http
  - @angular/router
  - @angular/animate
- https://www.npmjs.com/~angular

https://angular.io/guide/glossary

Import

- In general, we assemble our application from many modules, both the ones we write ourselves and the ones we acquire from others.

- A typical module is a cohesive block of code dedicated to a single purpose.

- A module exports something of value in that code, typically one thing such as a class. A module that needs that thing, imports it.

https://angular.io/guide/glossary

Metadata

Decorator (@Component())

- A decorator is a function that adds metadata to a class, its members (properties, methods) and function arguments.

- Decorators are a JavaScript language feature, implemented in TypeScript and proposed for ES2016 (AKA ES7).

- To apply a decorator, position it immediately above or to the left of the thing it decorates.

https://angular.io/guide/glossary

Class

- An Angular class responsible for exposing data to a View and handling most of the view's display and user-interaction logic.

https://angular.io/guide/glossary

AppComponent

app.component.ts

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

@Component({
    selector: 'sas-app',
    template: `
    <div>
        <h1>{{pageTitle}}</h1>
        <div>Hello from AppComponent!</div>
    </div>
    `
})

export class AppComponent {
    pageTitle: string = 'Simple Angular Sample';
}
```

Bootstrapping/Startup

- Angular sind **Single Page Application** (SPA)
- **index.html** ist der **Startpunkt** der Anwendung
- **index.html** enthält die **„Root“-Component**
- **Start** der Anwendung heißt **Bootstrapping**

Bootstrapping/Startup

1. **index.html** *systemjs.config.js /<my-app>*
2. **systemjs.config.js** *main.ts*
3. **main.ts** *AppModule*
4. **app.module.ts** *AppComponent*
5. **app.component.ts** *Code*

Component-Konventionen

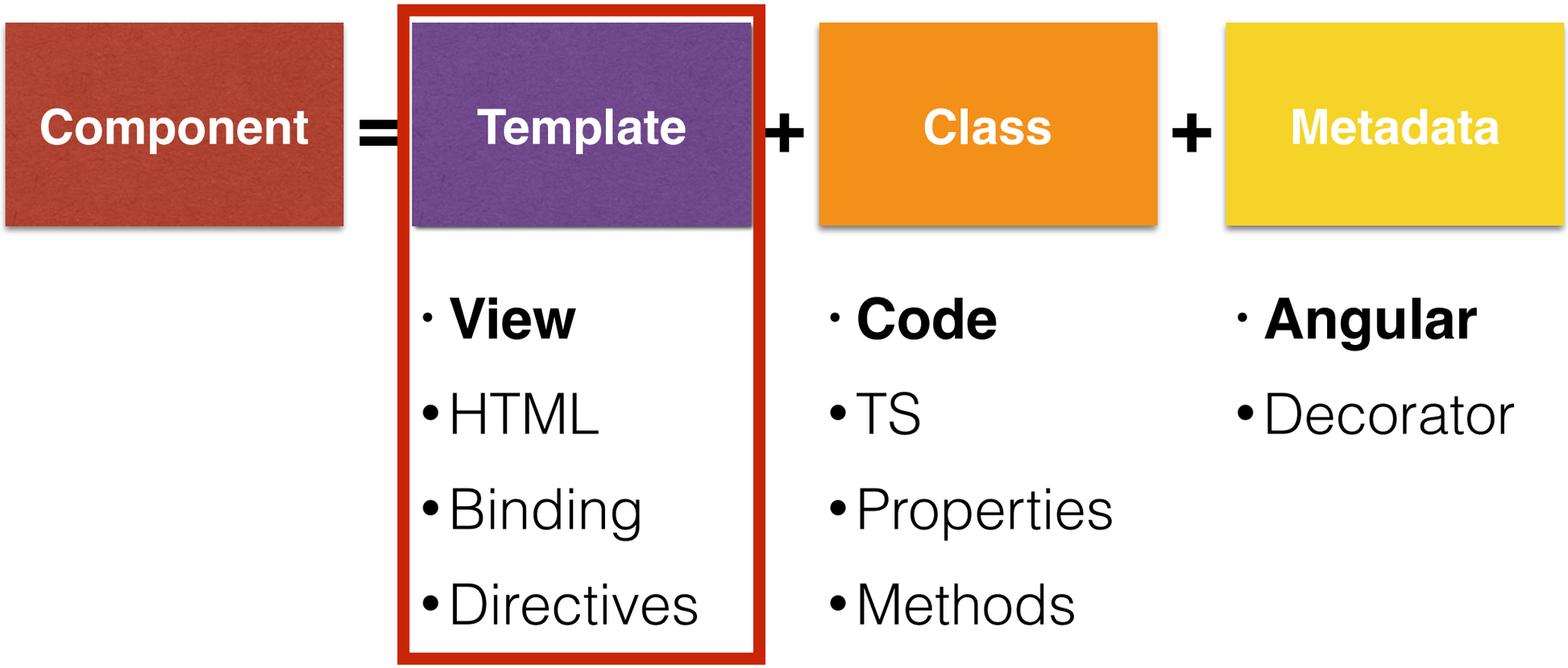
- Filename
  - **kebab-case**, *.component*-Suffix
- Class
  - **PascalCase**, *Component*-Suffix
- Properties/Methods
  - **camelCase**

Zusammenfassung

- Component
- Import
- Decorator
- Class
- Bootstrapping
- Konventionen

Angular - Templating

Angular - Component



Templates

Template

- A template is a chunk of HTML that Angular uses to render a view with the support and continuing guidance of an Angular directive, most notably a component.

- We write templates in a special Template Syntax.

- An Template expression in a JavaScript-like syntax that Angular evaluates within a data binding.

https://angular.io/guide/glossary

Template-Varianten

```
Inline-Template / Single Line
template:
    '<h1>{{pageTitle}}</h1>'
```

```
Inline-Template / Multi-Line with Back-Ticks

template: `
<div>
    <h1>{{pageTitle}}</h1>
    <div>
        My First Component
    </div>
</div>
`
```

```
Linked-Template

templateUrl:
    'item-list.component.html'
```

Data Binding

Data Binding

Template / DOM

Interpolation: **{{...}}**

Property Binding: **[...]**

Event Binding: **(...)**

Two-Way Data Binding: **[(...)]**

Class / Component

Data Binding

- Angular has a rich data binding framework with a variety of data binding operations and supporting declaration syntax.
  - Interpolation
  - Property Binding
  - Attribute Binding
  - Class Binding
  - Style Binding
  - Event Binding
  - Two-way data binding with ngModel

https://angular.io/guide/glossary

Interpolation

Template

```
HTML

<h1>{{ pageTitle }}</h1>

{{ 'Title: ' + pageTitle }}

{{ 'Title: ' + getTitle() }}

<h1 innerText="{{getTitle()}}"
></h1>
```

Class

```
Class

export class HomeComponent {
    pageTitle: string = 'Hello';

    getTitle(): string {
        return pageTitle;
    }
}
```

Property Binding

- Write a template property binding to set a property of a view element. The binding sets the property to the value of a template expression.

- The most common property binding sets an element property to a component property value.

https://angular.io/guide/glossary

Property Binding

HTML

```
<h1 [innerText]="getTitle()"></h1>
<img [src]="item.imageUrl">

<img src="{{item.imageUrl}}">
<img src="http://openclipart.org/{{item.imageUrl}}">
```

Attribute Binding

- We can set the value of an attribute directly with an attribute binding.

- This is the only exception to the rule that a binding sets a target property. This is the only binding that creates and sets an attribute.

- We must use attribute binding when there is no element property to bind (ex.: ARIA, SVG, and table span attributes)

https://angular.io/guide/glossary

Attribute Binding

HTML

```
<tr><td [attr.colspan]="1 + 1">One-Two</td></tr>

<button [attr.aria-label]="actionName">{{actionName}} with
Aria</button>
```

Class Binding

- We can add and remove CSS class names from an element's class attribute with a class binding.
- Class binding syntax resembles property binding. Instead of an element property between brackets, we start with the prefix class, optionally followed by a dot (.) and the name of a CSS class: [class.class-name].

https://angular.io/guide/glossary

Class Binding

HTML

```
<div class="bad curly special">Bad curly special</div>

<div class="bad curly special"
     [class]="badCurly">Bad curly</div>

<div [class.special]="isSpecial">The class binding is
special</div>

<div class="special"
     [class.special]="!isSpecial">This one is not so
special</div>
```

Style Binding

- We can set inline styles with a style binding.
- Style binding syntax resembles property binding. Instead of an element property between brackets, we start with the prefix style, followed by a dot (.) and the name of a CSS style property: [style.style-property].

https://angular.io/guide/glossary

Style Binding

HTML

```
<button [style.color]="isSpecial ? 'red': 'green'">Red</
button>
<button [style.background-color]="canSave ? 'cyan': 'grey'"
>Save</button>

<button [style.font-size.em]="isSpecial ? 3 : 1">Big</
button>
<button [style.font-size.%]="!isSpecial ? 150 : 50">Small</
button>
```

Event Binding

- Event binding syntax consists of a target event within parentheses on the left of an equal sign, and a quoted template statement on the right.
    - Target event:
        - example: change, click, mousemove
    - $event:
        - DOM even object
- https://developer.mozilla.org/en-US/docs/Web/Events

https://angular.io/guide/glossary

Event Binding

HTML

```
<button (click)="doSomething($event)">Click here</button>

<a href="#" (mouseover)="showSomething()">Move here</a>

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

Two-Way Binding

- On the element side that takes a combination of setting a specific element property and listening for an element change event.

- Angular offers a special two-way data binding syntax for this purpose, [(x)]. The [(x)] syntax combines the brackets of Property Binding, [x], with the parentheses of Event Binding, (x).

https://angular.io/guide/glossary

Two-Way Binding

HTML

```
<h2>{{pageTitle}}</h2>

<p>
<input [(ngModel)]="pageTitle">
</p>
```

Class

```
export class HomeComponent {
    pageTitle: string = 'Hello';
}
```

[()] = Banana in a Box

Directives

Directives

- A Directive is almost always associated with an HTML element or attribute.

- Developers can invent custom HTML markup to associate with their custom directives.

- Directives fall into one of three categories:
  - Components
  - Attribute Directives (ex.: ngClass, ngStyle)
  - Structural Directives (ex.: ngIf, ngFor)

https://angular.io/guide/glossary

Structural Directives

- A category of Directive that can shape or re-shape HTML layout, typically by adding, removing, or manipulating elements and their children.

- The ngIf "conditional element" directive and the ngFor "repeater" directive are good examples in this category.

https://angular.io/guide/glossary

ngClass

Template	Class

HTML

```
<div
[ngClass]="setClasses()">This div
is ???</div>
```

Class

```
setClasses() {
  let classes =  {
    red: this.isRed,
    bold: this.isBold,
    italic: this.isItalic,
  };
  return classes;
}
```

ngStyle

Template

HTML

```
<div [ngStyle]="setStyles()">This
div is ???</div>
```

Class

Class

```
setStyles() {
  let styles = {
   'font-style':  this.style,
   'font-weight': this.weight,
   'font-size':   this.size
  };
  return styles;
}
```

ngIf

HTML

```
<table *ngIf="items && items.length">
    <thead>
    </thead>
    <tbody>
    </tbody>
</table>
```

ngFor

HTML

```
<tr *ngFor="let item of items">
    <td>{{ item.id }}</td>
    <td>{{ item.name }}</td>
    <td>{{ item.description }}</td>
</tr>
```

* and <template>

- The * is a bit of syntactic sugar that makes it easier to read and write directives that modify HTML layout with the help of templates.

- NgFor, NgIf, and NgSwitch all add and remove element subtrees that are wrapped in <template> tags.

https://angular.io/guide/glossary

ngIf

HTML

```
<table template="ngIf:items && items.length“>
    <thead>
    </thead>
    <tbody>
    </tbody>
</table>
```

ngFor

HTML

```
<tr template="ngFor #item of items">
    <td>{{ item.id }}</td>
    <td>{{ item.name }}</td>
    <td>{{ item.description }}</td>
</tr>
```

Component Styles

Component Styles

- Angular applications are styled with regular CSS.
- On top of this, Angular has the ability to bundle component styles with our components enabling a more modular design than regular stylesheets.
- One way to do this is to set the styles property in the component metadata. The styles property takes an array of strings that contain CSS code.
- We can load styles from external CSS files by adding a styleUrls attribute into a component's @Component

https://angular.io/guide/glossary

Component Lifecycle

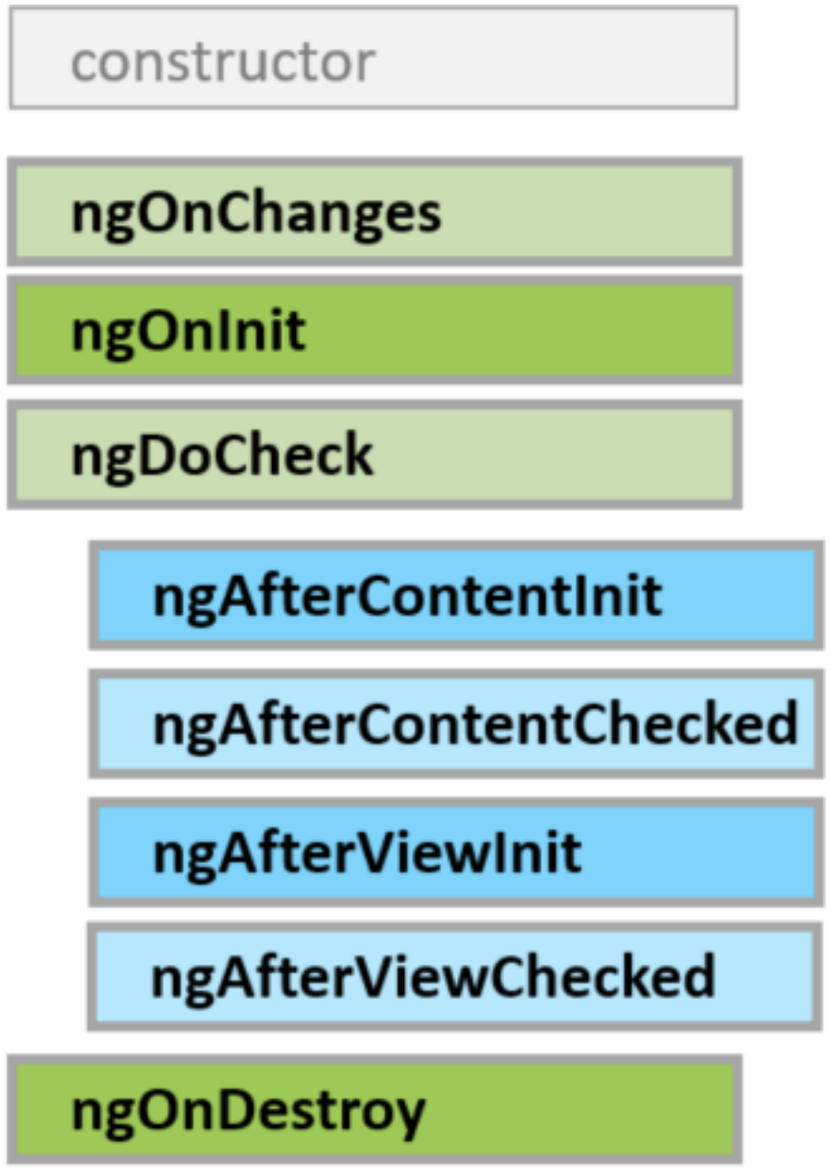
Component Lifecycle

- A component has a lifecycle managed by Angular itself.
- Angular creates it, renders it, creates and renders its children, checks it when its data-bound properties change, and destroys it before removing it from the DOM.
- Angular offers lifecycle hooks that provide visibility into these key life moments and the ability to act when they occur.
- A directive has the same set of lifecycle hooks, minus the hooks that are specific to component content and views.

https://angular.io/guide/glossary

Component Lifecycle

Lifecycle Hooks



Lifecycle Hooks

- **constructor** .
- **ngOnChanges** - before ngOnInit and when a data-bound input property value changes
- **ngOnInit** - after the first ngOnChanges
- **ngDoCheck** - during every Angular change detection cycle
  - **ngAfterContentInit** - after projecting content into the component
  - **ngAfterContentChecked** - after every check of projected component content
  - **ngAfterViewInit** - after initializing the component's views and child view
  - **ngAfterViewChecked** - after every check of the component's views and child view
- **ngOnDestroy** - just before Angular destroys the directive/component

https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html

Zusammenfassung

- Templates
- Data Binding
- Directives
- Component Styles
- Component Lifecycle

Angular - Pipes

Pipes

- An Angular pipe is a function that transforms input values to output values for display in a view.

- We use the @Pipe decorator to associate the pipe function with a name.

- We then can use that name in our HTML to declaratively transform values on screen.

https://angular.io/docs/ts/latest/guide/glossary.html

Build-In Pipes

- @angular/common
- AsyncPipe, CurrencyPipe, DatePipe, DecimalPipe, I18nPluralPipe, I18nSelectPipe, JsonPipe, LowerCasePipe, PercentPipe, SlicePipe, UpperCasePipe

https://angular.io/docs/ts/latest/api/#!?apiFilter=pipe&query=pipe

Build-In Pipes

HTML

```
{{ item.code | lowercase }}

<img [src]="item.imageUrl"
       [title]="item.title | uppercase“>

{{ item.price | currency | lowercase }}

{{ item.price | currency:USD:true:1.2-2 }}
```

Custom Pipes

- @Pipe()-Decorator
- Implements PipeTransform
- transform-Method

https://angular.io/docs/ts/latest/guide/pipes.html

Zusammenfassung

- Pipes
- Build-In Pipes
- Custom Pipes

Angular - DI, Services

Dependency Injection

- Dependency injection is an important application design pattern.

- Angular has its own dependency injection framework, and we really can't build an Angular application without it.

- It's used so widely that almost everyone just calls it DI.

https://angular.io/docs/ts/latest/guide/dependency-injection.html

Angular - Application

DI - Szenario

Service-Class

```
export class MyService {

}
```

Component-Class

```
import { MyService }
    from './MyService';

export class MyComponent {
    _myService: MyService;

    constructor() {
        _myService =
            new MyService();
    }
}
```

Angular-DI - Szenario

injector
injector.get(MyService)

Service-Class

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

@Injectable()
export class MyService {

}
```

Component-Class

```
import { MyService }
   from './MyService';

export class MyComponent {
   constructor(private
_myService: MyService) {
   }
}
```

Zusammenfassung

- Dependency Injection
- Service erstellen
- Service nutzen

Angular - HTTP, RxJS

HTTP Client

- HTTP is the primary protocol for browser/server communication

- Modern browsers support two HTTP-based APIs: XMLHttpRequest (XHR) and JSONP. A few browsers also support Fetch.

- The Angular HTTP client library simplifies application programming of the XHR and JSONP APIs.

- RxJS ("Reactive Extensions") is a 3rd party library, endorsed by Angular, that implements the asynchronous observable pattern.

https://angular.io/docs/ts/latest/guide/server-communication.html

RxJS (ReactiveX)

- An API for asynchronous programming with observable streams

- ReactiveX is a combination of the best ideas from the Observer pattern, the Iterator pattern, and functional programming

- RxJS is a set of libraries to compose asynchronous and event-based programs using observable collections and Array style composition in JavaScript

http://reactivex.io/

Observable

- An observable as an array whose items arrive asynchronously over time.

- Observables help you manage asynchronous data, such as data coming from a backend service.

- Observables are used within Angular itself, including Angular's event system and its http client service.

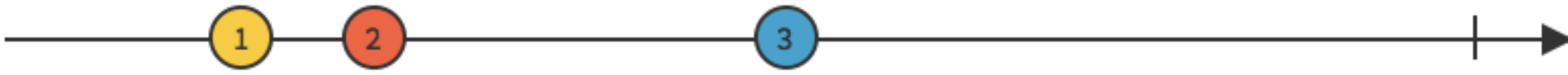
- To use observables, Angular uses a third-party library called Reactive Extensions (RxJS).

- Observables are a proposed feature for ES 2016, the next version of JavaScript.

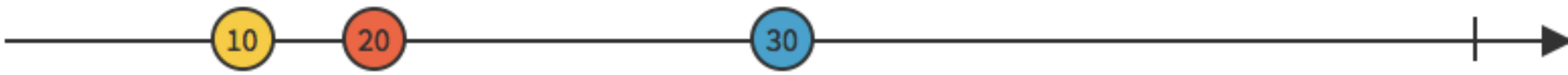
https://angular.io/docs/ts/latest/glossary.html

RxMarbles

Interactive diagrams of Rx Observables




```
map(x => 10 * x)
```



Observables Operators

http://rxmarbles.com/#map

Promise versus Observable

- Promise
  - Rückgabe eines Wertes
  - Kann nicht abgebrochen werden
- Observable
  - Rückgabe von mehreren Werten über Zeitraum
  - Kann abgebrochen werden
  - Unterstützt Operatoren: map, filter, reduce

Zusammenfassung

- HTTP Client
- RxJS
- Observables

Angular - Router, Navigation

Router

- Most applications consist of many screens or views.
- The Angular Component Router is a richly featured mechanism for configuring and managing the entire view navigation process including the creation and destruction of views.
- In most cases, components become attached to a router by means of a *RouterConfig* that defines routes to views.
- A routing component's template has a *RouterOutlet* element where it can display views produced by the router.
- Other views in the application likely have anchor tags or buttons with *RouterLink* directives that users can click to navigate.

https://angular.io/docs/ts/latest/guide/glossary.html

Router Setup

- Setzen der „Base-Href“ in index.html:
  - <base href="/">
- Import des „Router Modules“
  - import { RouterModule } from '@angular/router';
- Konfiguration der „Routen“:
  - RouterModule.forRoot([], { useHash: true })
- Hinzufügen des View-Containers in index.html:
  - <router-outlet></router-outlet>

https://angular.io/docs/ts/latest/guide/router.html

Navigation

- Link:
  - <a [routerLink]="['home']">Home
  - <a [routerLink]="['detail', {id: 1}]">Detail
- Parameter auslesen:
  - constructor(private _router: Router, private _route: ActivatedRoute)
    }
- Navigation per Code:
  - import { Router } from ‚@angular/router';
    constructor(private _router: Router) { }
    onBack(): void {
    this._router.navigate(['Items']);
    }

Zusammenfassung

- Routing
- Router Setup
- Navigation

Angular - Forms, Validation

Forms

- An Angular form coordinates a set of data-bound user controls, tracks changes, validates input, and presents errors.

- Angular offers two types of forms:
  - Template-Driven Forms
  - Reactive Forms

https://angular.io/docs/ts/latest/guide/forms.html

Template-Driven Forms

- In the template-driven approach, you arrange form elements in the component's template.

- You add Angular form directives (mostly directives beginning ng...) to help Angular construct a corresponding internal control model that implements form functionality.

- To validate user input, you add HTML validation attributes to the elements. Angular interprets those as well, adding validator functions to the control model.

- Angular exposes information about the state of the controls including whether the user has "touched" the control or made changes and if the control values are valid

https://angular.io/docs/ts/latest/cookbook/form-validation.html

Template-Driven Forms

- Form:
- <form
  #form="ngForm"
  (ngSubmit)="onSubmit()">

Template-Driven Forms

- Input:
- <input type="text"
  id="name"
  name="name"
  required
  #name="ngModel"
  [(ngModel)]="model.name" >

Template-Driven Forms

- Validation:
  - <div [hidden]="**name.valid || name.pristine**"
    class="alert alert-danger">Name is required
- ngModel - Properties
  - ng-touched / ng-untouched
  - ng-dirty / ng-pristine
  - ng-valid / ng-invalid

Reactive Forms

- Reactive Forms takes a different approach. You create the form control model in code.

- You write the template with form elements and form... directives from the Angular ReactiveFormsModule.

- At runtime, Angular binds the template elements to your control model based on your instructions.

- This approach requires a bit more effort. You have to write the control model and manage it.

https://angular.io/docs/ts/latest/cookbook/form-validation.html

Reactive Forms

- FormBuilder
- FormGroup
- FormControl
- Validators

Angular - Testing

Testing in JavaScript

- Test-Frameworks
- Test-Runner
- Mobile Testing
- Testing in the Cloud

Test-Frameworks

- https://qunitjs.com/
- http://mochajs.org/
- http://jasmine.github.io/
- https://angular.github.io/protractor/

Test-Runner

- http://karma-runner.github.io/
- http://www.phantomjs.org/

Mobile Testing

- http://appium.io/
- http://calaba.sh/
- http://selendroid.io/

Testing in the Cloud

- https://www.browserstack.com/
- https://www.saucelabs.com/
- http://www.telerik.com/mobile-testing
- http://xamarin.com/test-cloud

Testing in Angular

- Unit-Tests:
  - Jasmine - Behavior-Driven JavaScript
    - http://jasmine.github.io/
- E2E-Tests
  - Protractor
  - http://www.protractortest.org/

https://angular.io/docs/ts/latest/guide/testing.html

Das war’s!

peter@gfu.net

Schreiben Sie mir in zwei Wochen eine E-Mail!