Angular Kickstart

von 0 auf 100

think tecture

Sascha Lehmann

Developer Consultant @ Thinktecture AG

Spezialisierung: Angular und UI/UX

- @derLehmann_S
- sascha.lehmann@thinktecture.com
- https://www.thinktecture.com/thinktects/sascha-lehmann/

Christian Liebel

Consultant @ Thinktecture AG

Spezialisierung: Angular, PWA, Cross-Platform

- @christianliebel
- christian.liebel@thinktecture.com
- https://www.thinktecture.com/thinktects/christian-liebel/

Timetable

09:00–10:30	Part I
10:30–11:00	Break
11:00–12:30	Part II
12:30–13:30	Lunch Break
13:30–15:00	Part III
15:00–15:30	Break
15:30–17:00	Part IV

Expectations

Setup

LAB #N

Questions: anytime!

Hands-on labs (everyone can participate)

https://bit.ly/ngdkick

Agenda

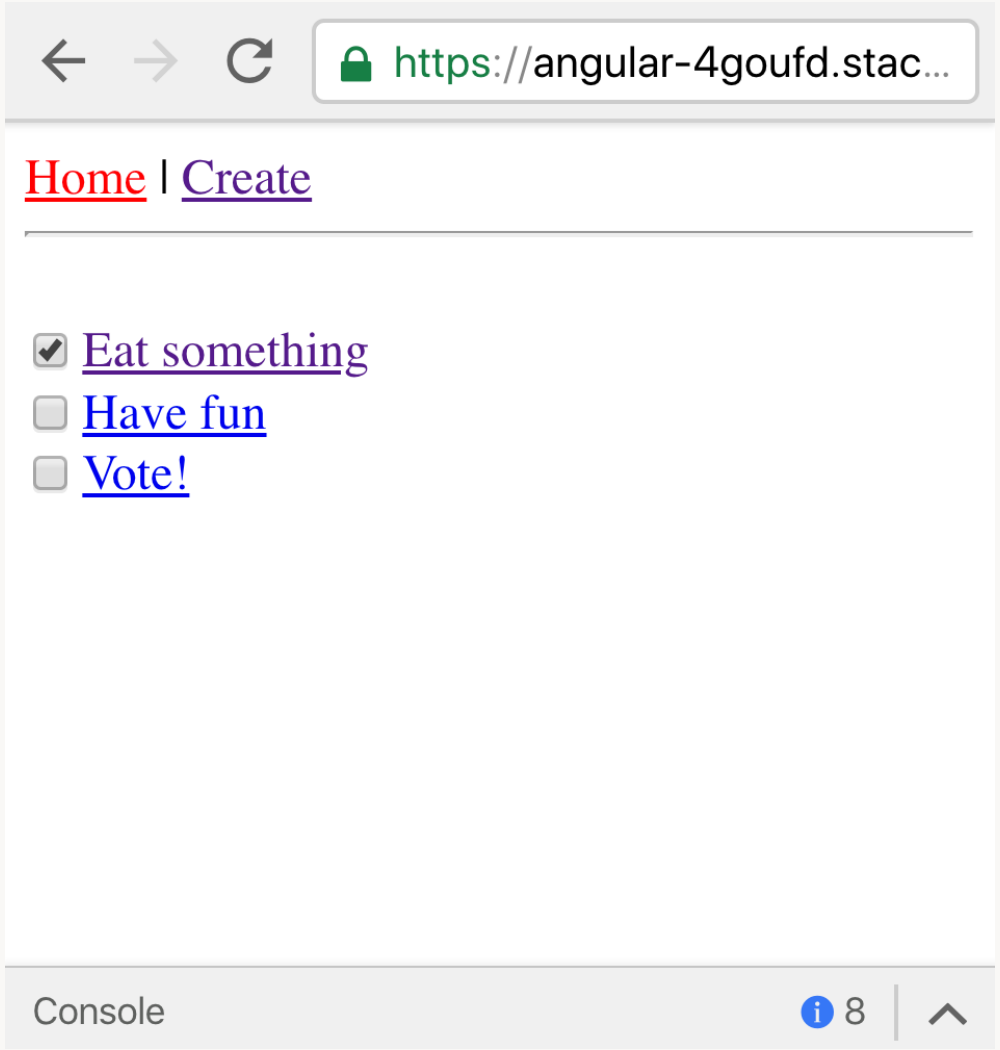
1. What is Angular?
2. Why SPA?
3. Angular CLI
4. Modules
5. Bindings
6. Pipes
7. Components
8. Input/Output
9. Directives
10. Dependency Injection

Agenda

11. Services
12. Structural Directives
13. Observables & RxJS
14. HttpClient
15. Lifecycle Hooks
16. Async Pipe
17. Routing
18. Template-Driven Forms
19. Debugging

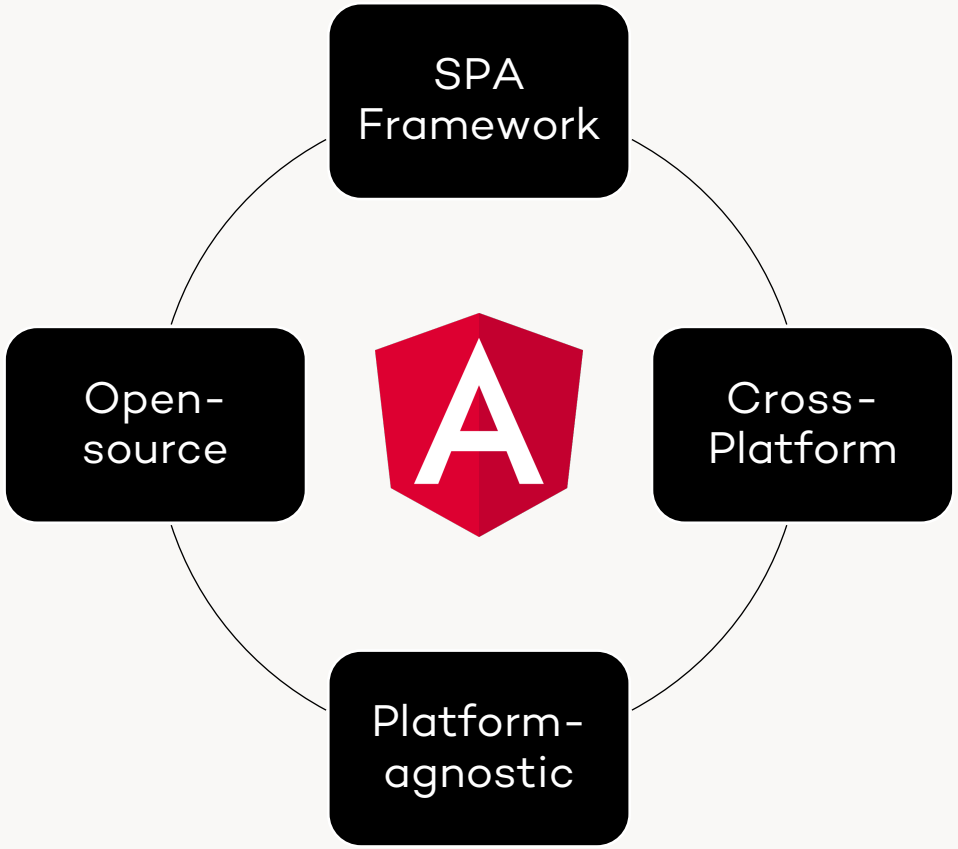
Our Demo Use-Case

The next best todo list app!



1. What is Angular?

What is Angular?



Angular

Technical Basis

Angular

Release Schedule

Time-based release schedule (6 months)

- March/April: even version
- September/October: odd version

Deprecation Policy

- Compatibility to previous major version (1 year)
- Long-Term Supported Version (critical fixes/security patches only)
- Current Version -> 15

2. Why SPA?

Single-Page Web Applications (SPA)

Properties

- Fat clients (i.e., load everything they need to run during bootstrap)
- A change of the view does not lead to a server-side page navigation

Single-Page Web Applications (SPA)

Advantages

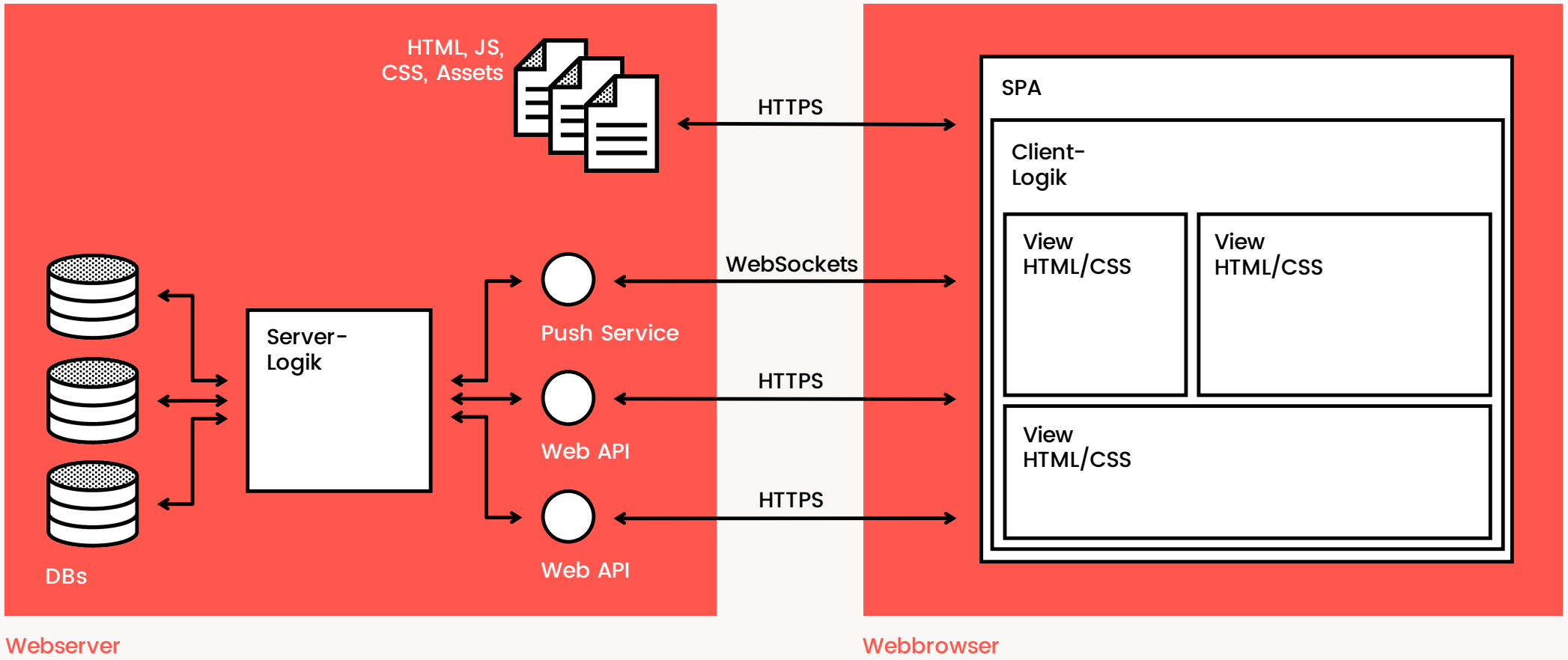
- Very performant
- Works offline
- No special server requirements (i.e., serving static files is sufficient)

Disadvantages

- Some logic (i.e., computation-intensive) can only be run on a server (connection required)
- Logic is transfered to the client (code can't be kept secret)

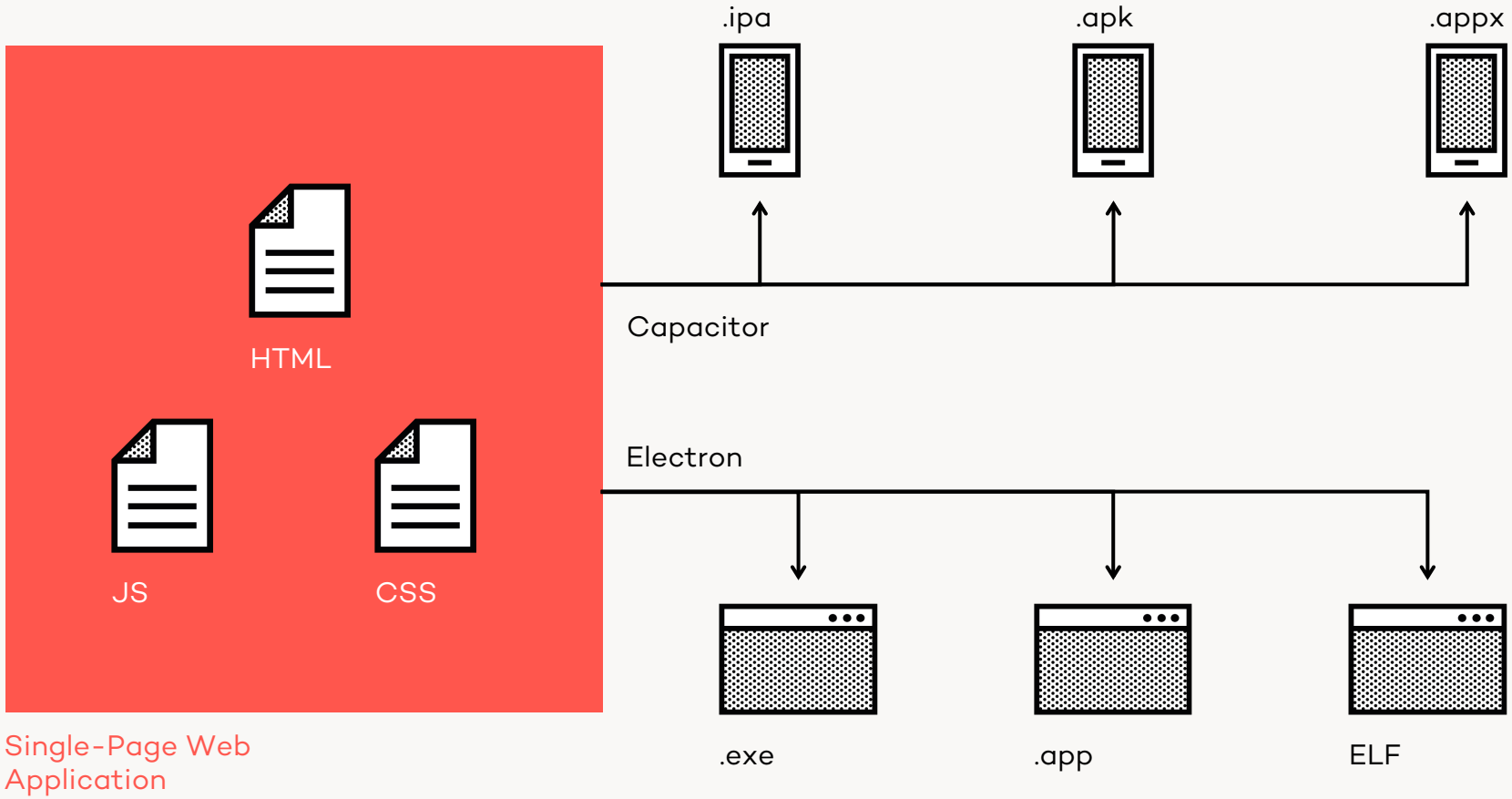
Single-Page Web Applications (SPA)

Architecture



Single-Page Web Applications (SPA)

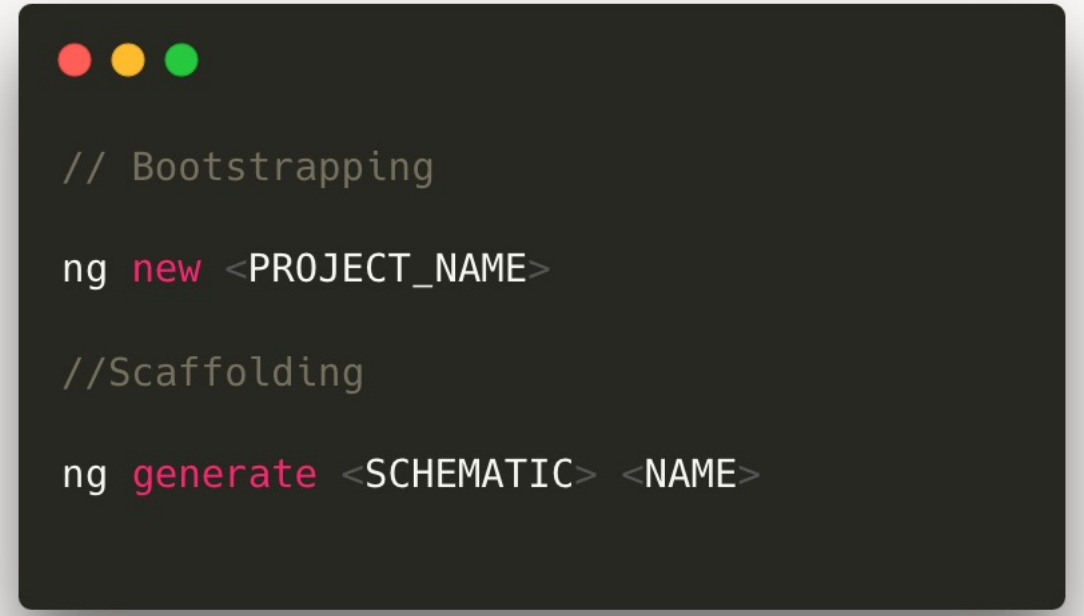
Cross-Platform Support



3. Angular CLI

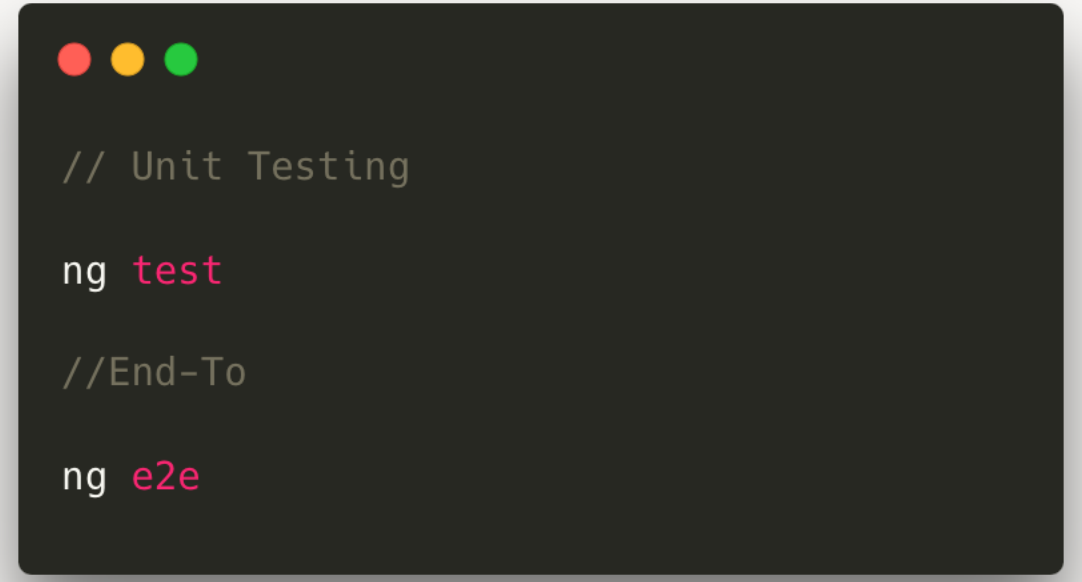
Angular CLI

Features



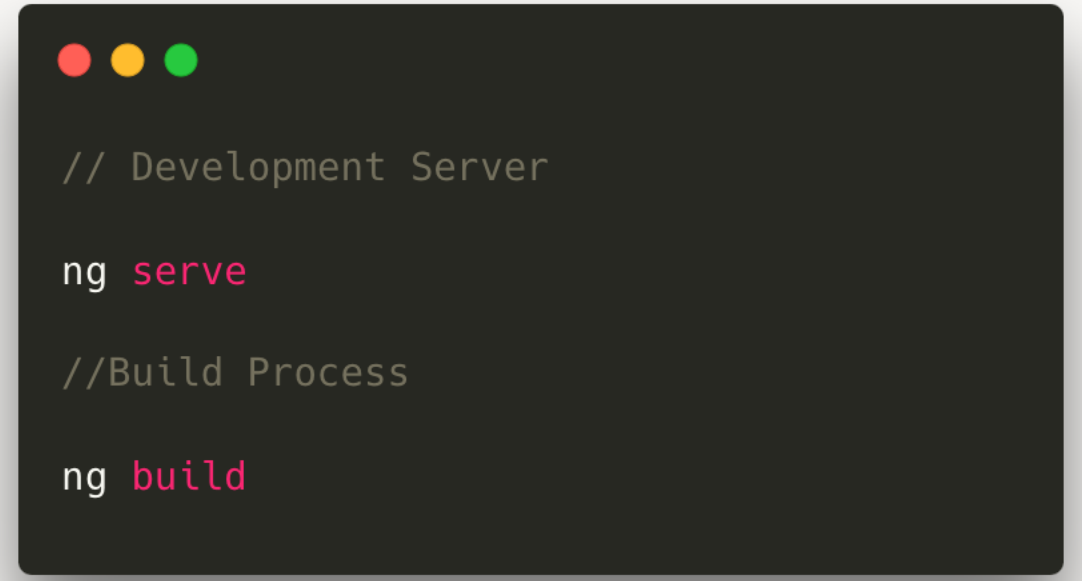
Angular CLI

Features



Angular CLI

Features



Angular CLI

Features

```
// Add Angular Libraries

ng add

//Update App and Dependencies

ng update
```

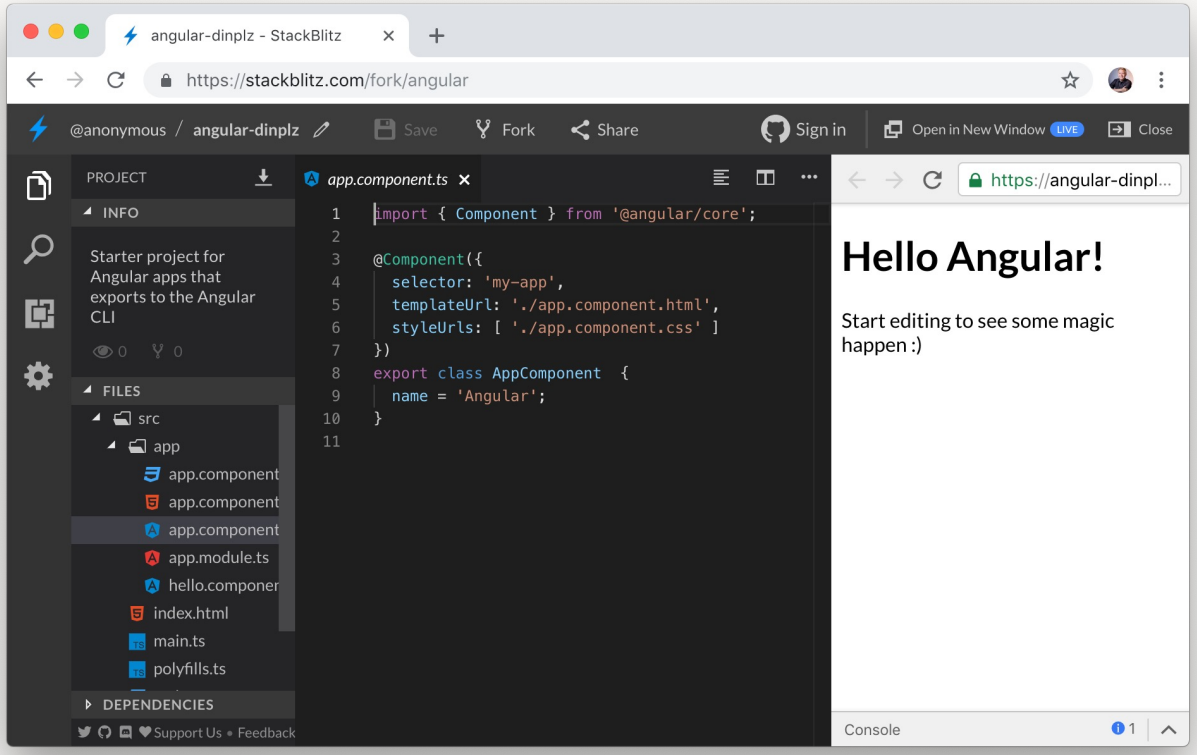
Angular CLI

Features

```
// Linter (Static Code Analysis)

ng lint

//Documentation

ng doc component
```

Let's try out StackBlitz

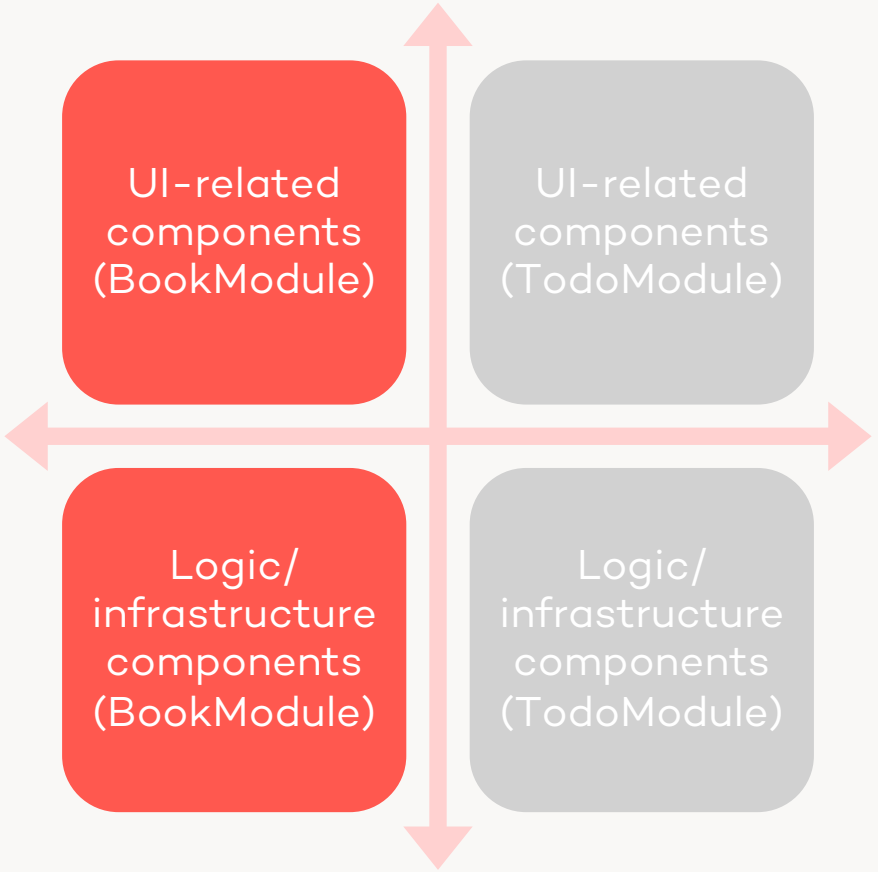
LAB #0



4. Modules

Modules

Application Segmentation

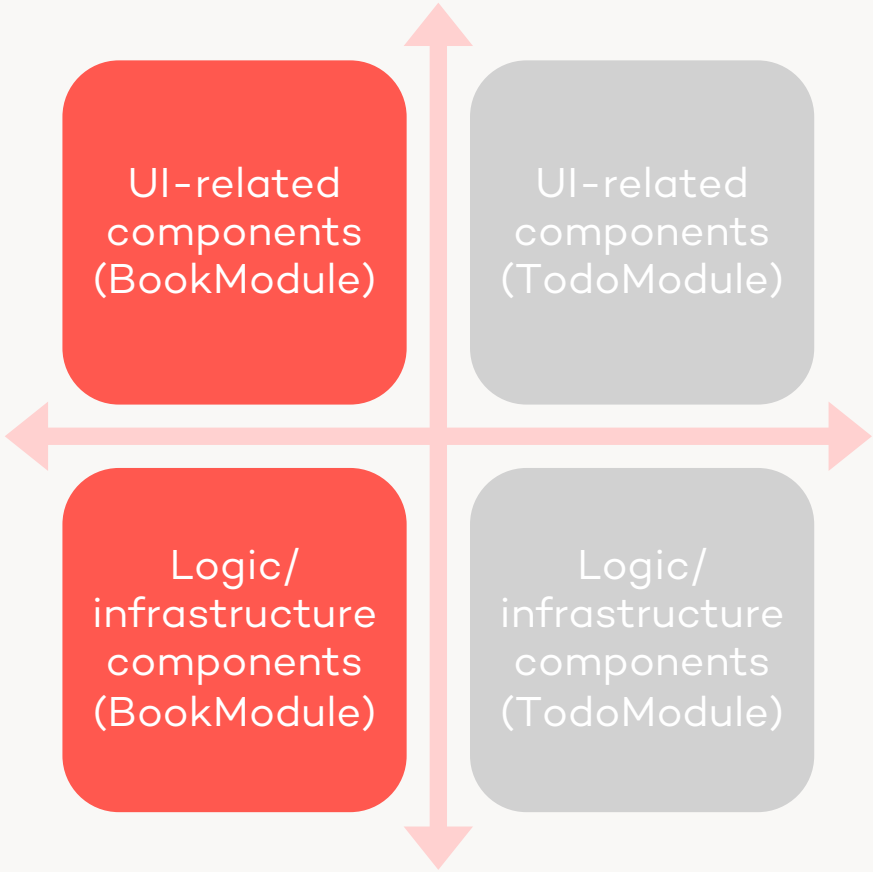


Modules

Angular Building Blocks

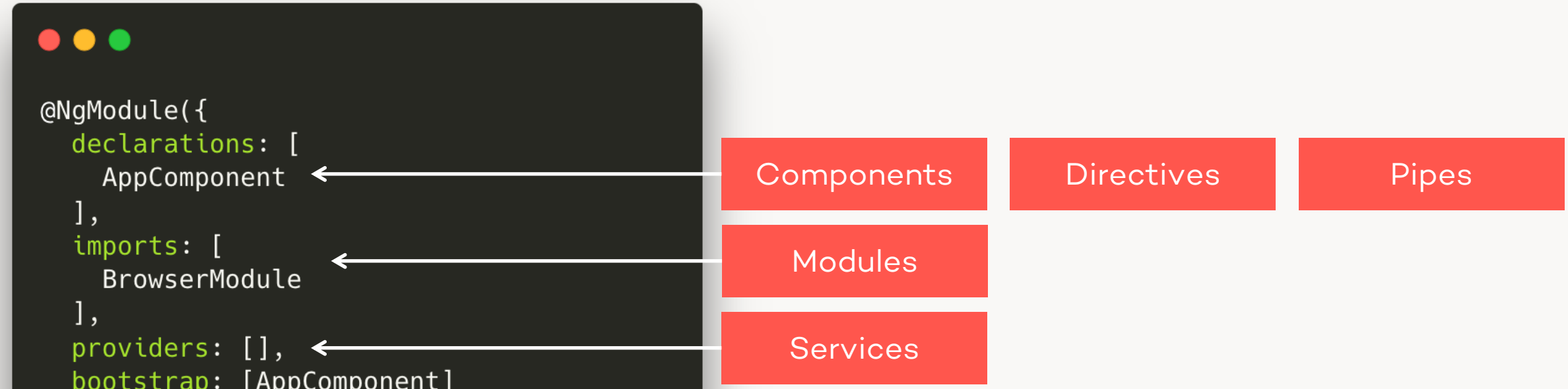
- Components
- Directives
- Pipes

- High-Level Services
- Low-Level Services



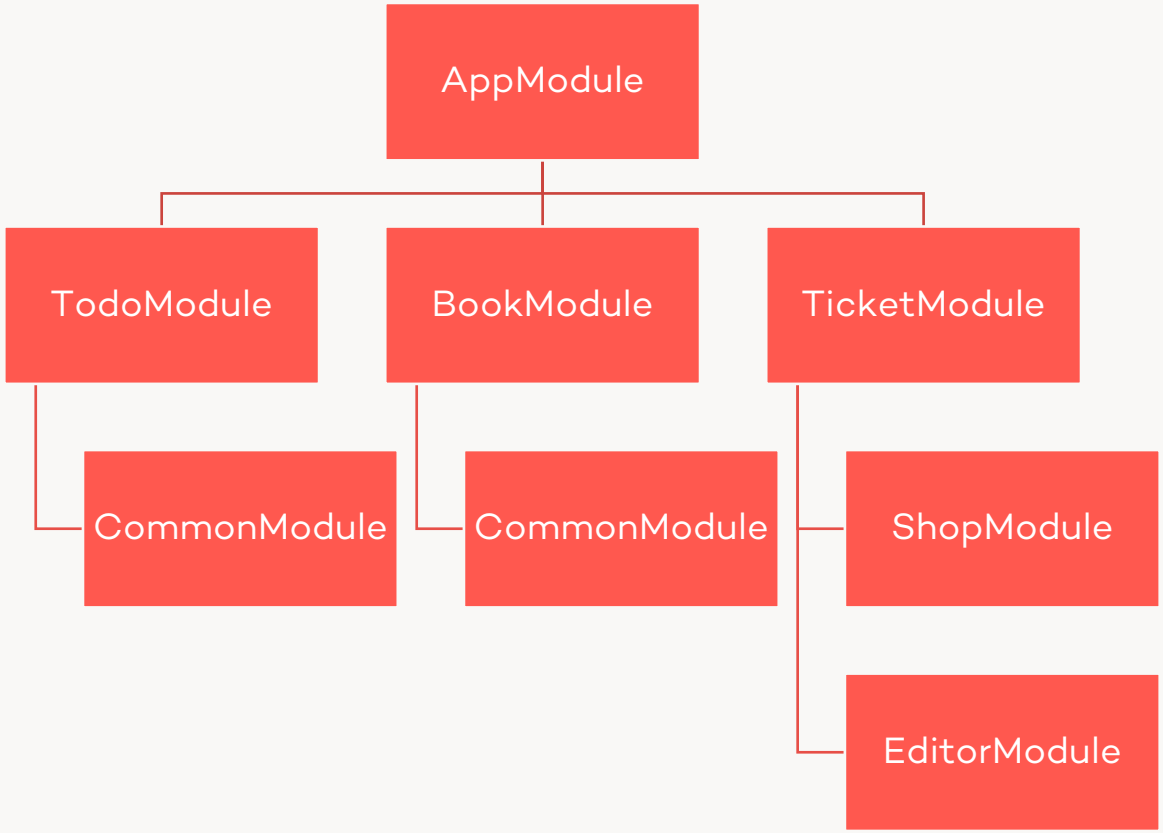
Modules

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



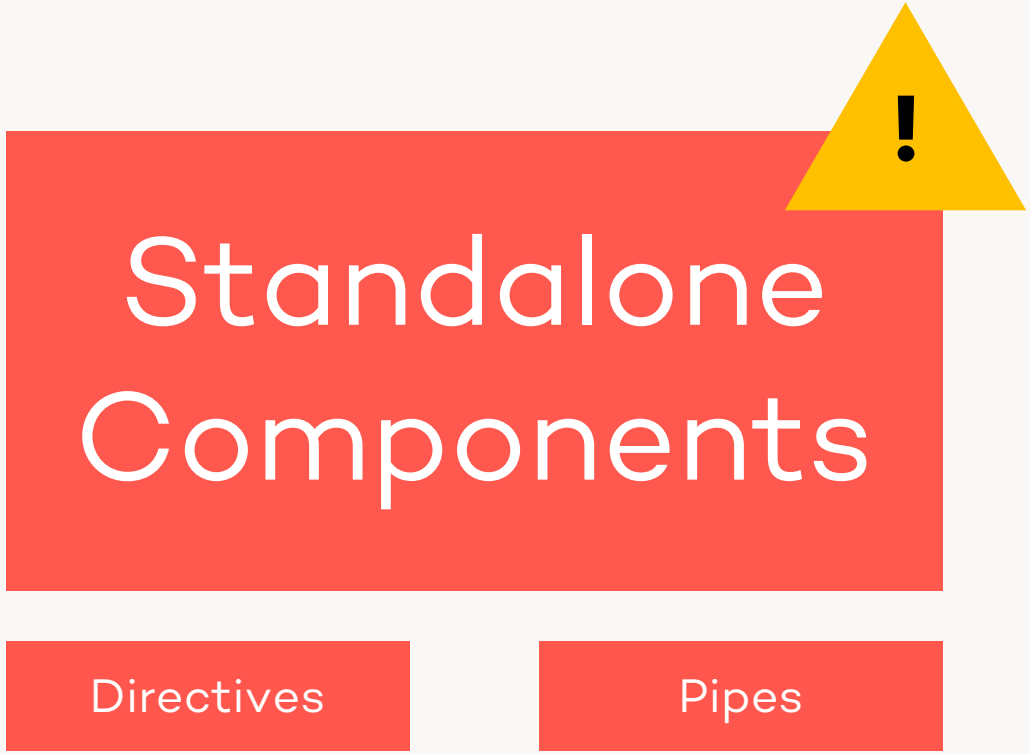
Modules

Dependency Tree



Modules

For the future



5. Bindings

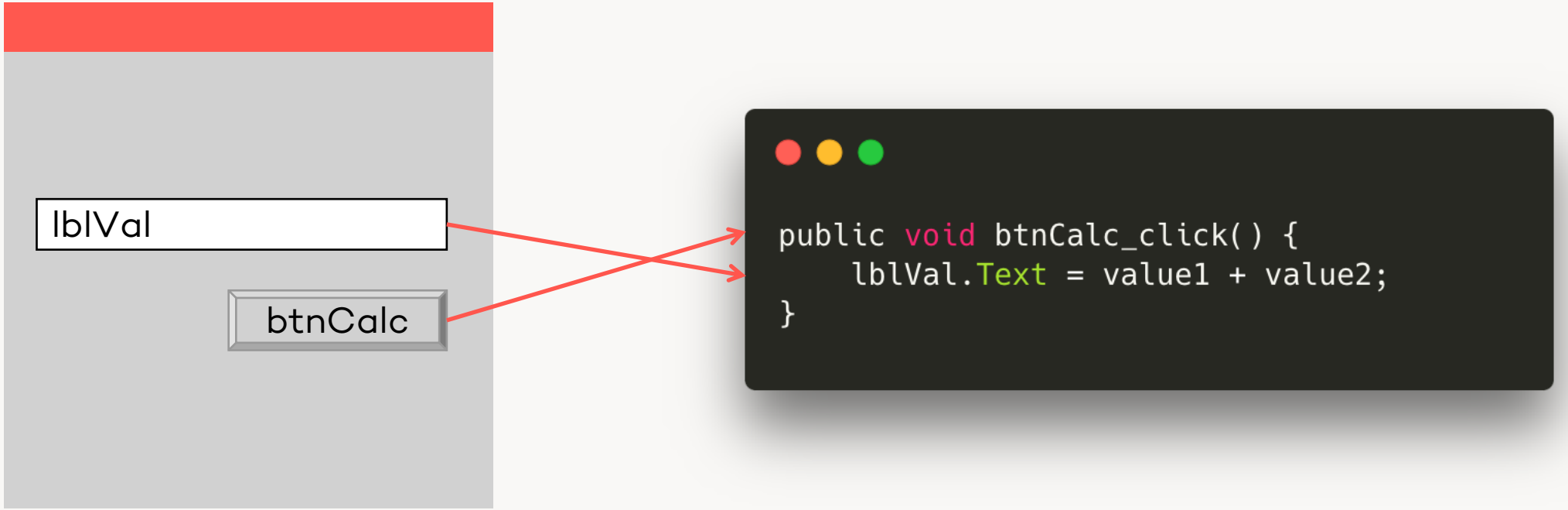
Data Binding

UI references a **property** on the component instance that should be interpolated

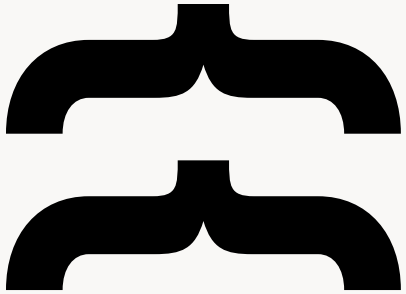
Or: UI references a method on the component instance that should be called on a certain **event**

Automatically updates UI when the model is updated

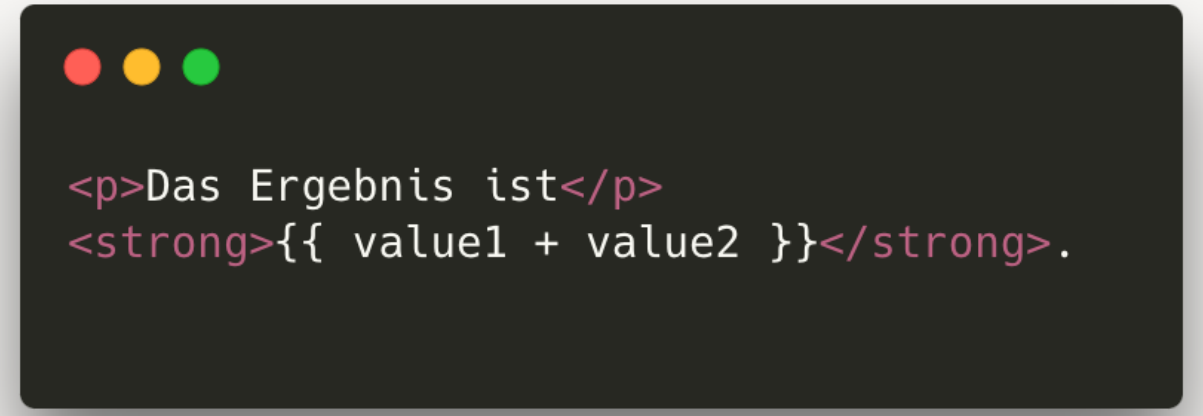
Keeps presentation and model **in sync**


lblVal
btnCalc
public void btnCalc_click() {
lblVal.Text = value1 + value2;
}

Moustache Syntax



Handlebars



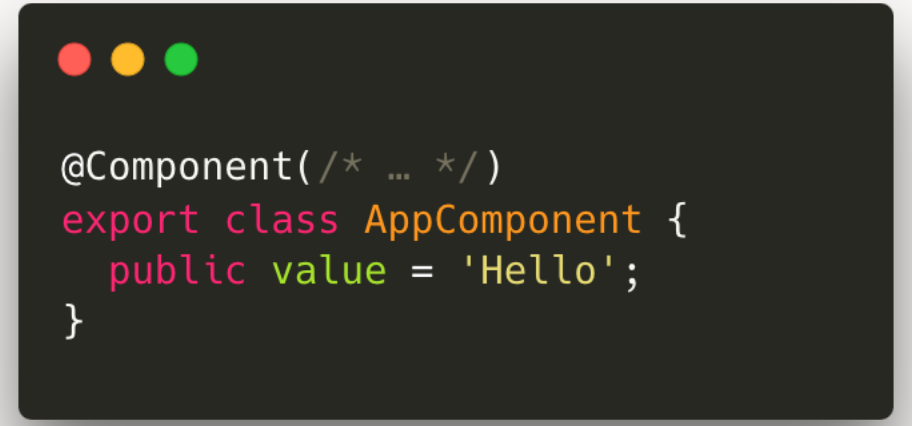
Bindings

Interpolation

Component view (HTML)

```
{{ value }}
```

Component logic (TS)



```
@Component(/* … */)
export class AppComponent {
  public value = 'Hello';
}
```

Bindings

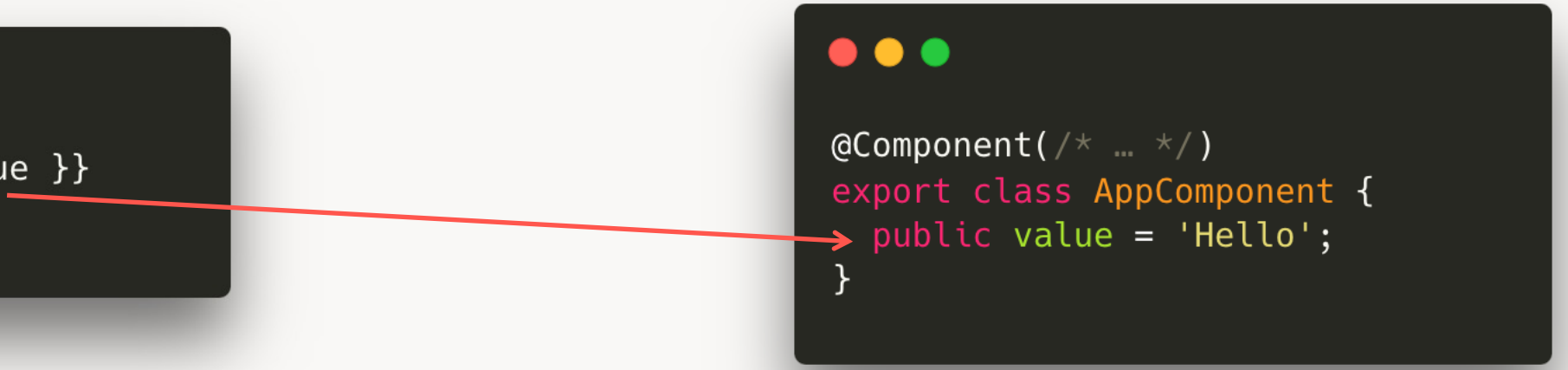
Interpolation

Component view (HTML)

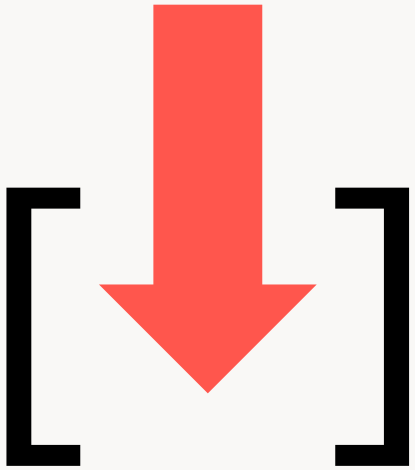
```
{{ value }}
```

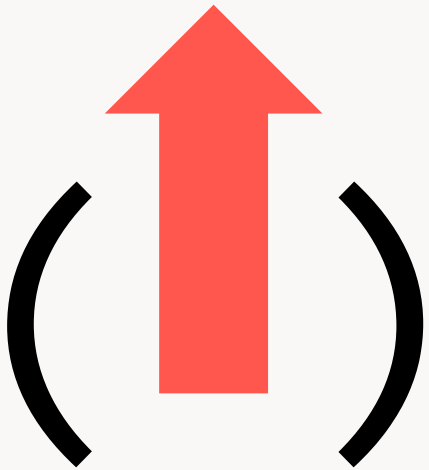
Component logic (TS)

```
@Component(/* … */)
export class AppComponent {
  public value = 'Hello';
}
```

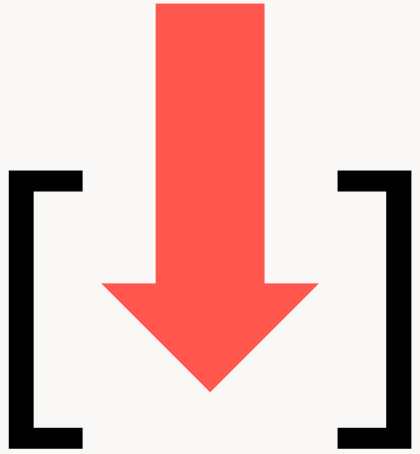


More Bindings





Property Binding



Pass data **in**

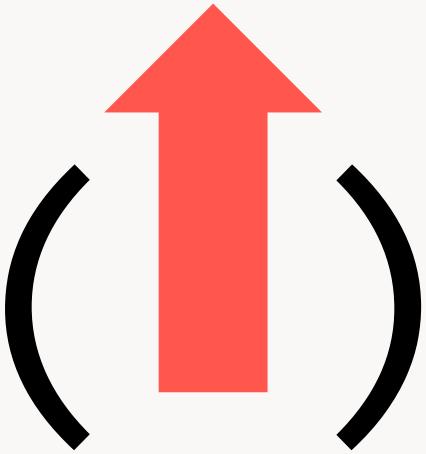
Bind to a certain **property** of a DOM node or component/directive

Property Bindings

More options

```
<!-- Define attributes on a DOM element and bind value -->
<button [attr.aria-label]="'Help'">Help</button>

<!-- Bind class existance to a certain value -->
<div [class.done]="done">Todo</div>

<!-- Bind a style property to a certain value -->
<button [style.width.px]="300">I’m 300px wide!</button>
```

Event Binding



Get data **out**

Bind to a certain **event** of a DOM node or component/directive

Event Binding

```ts
// app.component.ts
public onClick(): void {
  alert('Hello!');
}
```

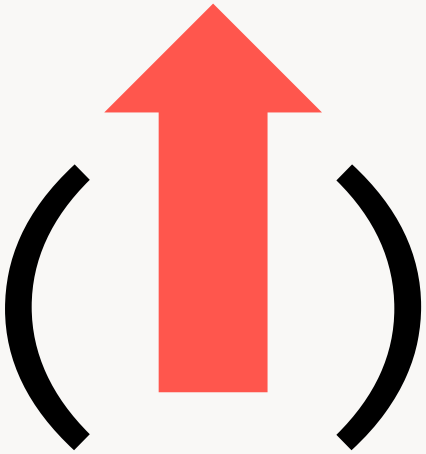
```html
<!-- app.component.html -->
<button (click)="onClick()">Click me.</button>
```

Bindings

LAB #1

- Interpolations
- Property Bindings
- Event Bindings

Event Binding



Magic value: $event

Contains the event arguments

```
<div (click)="onClick($event)"></div>
```

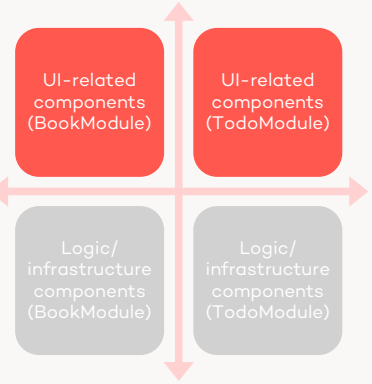
Event Binding

LAB #2

```
public onClick(event: MouseEvent): void {
  alert(event.clientX);
}

public onMouseMove(event: MouseEvent): void {
  console.log(event.clientX);
}
```

```
<button (click)="onClick($event)" (mousemove)="onMouseMove($event)">Click me.</button>
```

6. Pipes



UI-related

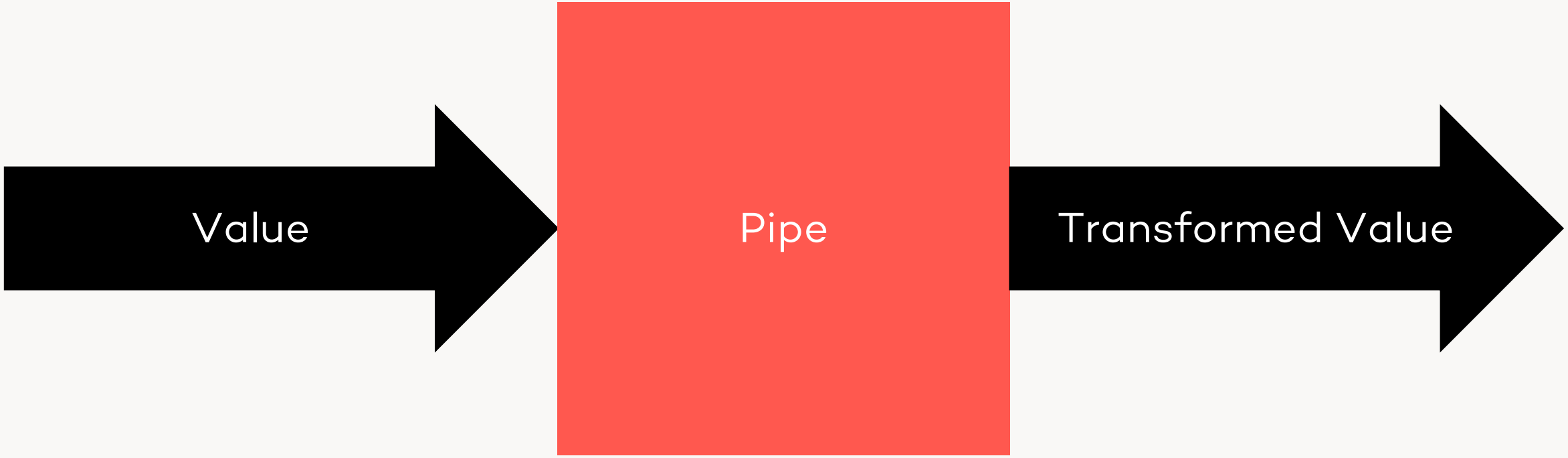
Re-usable

Manipulate binding values for the view
without changing the underlying value (one-way)

```
{{ value | pipe }}
```

Pipes

Principle



Pipes

Principle

Pipes

Built-in Pipes

- uppercase
- lowercase
- date
- number
- percent
- currency
- json

Pipes

Parameters

Pipes can also have parameters delimited by a **colon**

```
{{ value | number:'0.3' }}
```

Pipes

Built-in Pipes

```
{{ value | uppercase }}          // --> HELLO

{{ number | percent }}           // --> 314%

{{ number | currency }}          // --> $ 3.14

{{ number | number:'0.5' }}      // --> 3.14159
```

Pipes

Custom Pipes

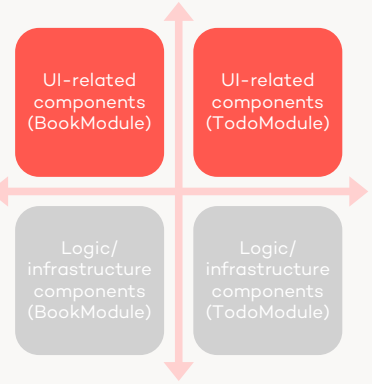
```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({ name: 'yell' })
export class YellPipe implements PipeTransform {

  transform(value: any, args?: any): any {
    return null;
  }

}
```

Pipes

LAB #3

- Interpolation
- Built-in pipes
- Create a new pipe

7. Components

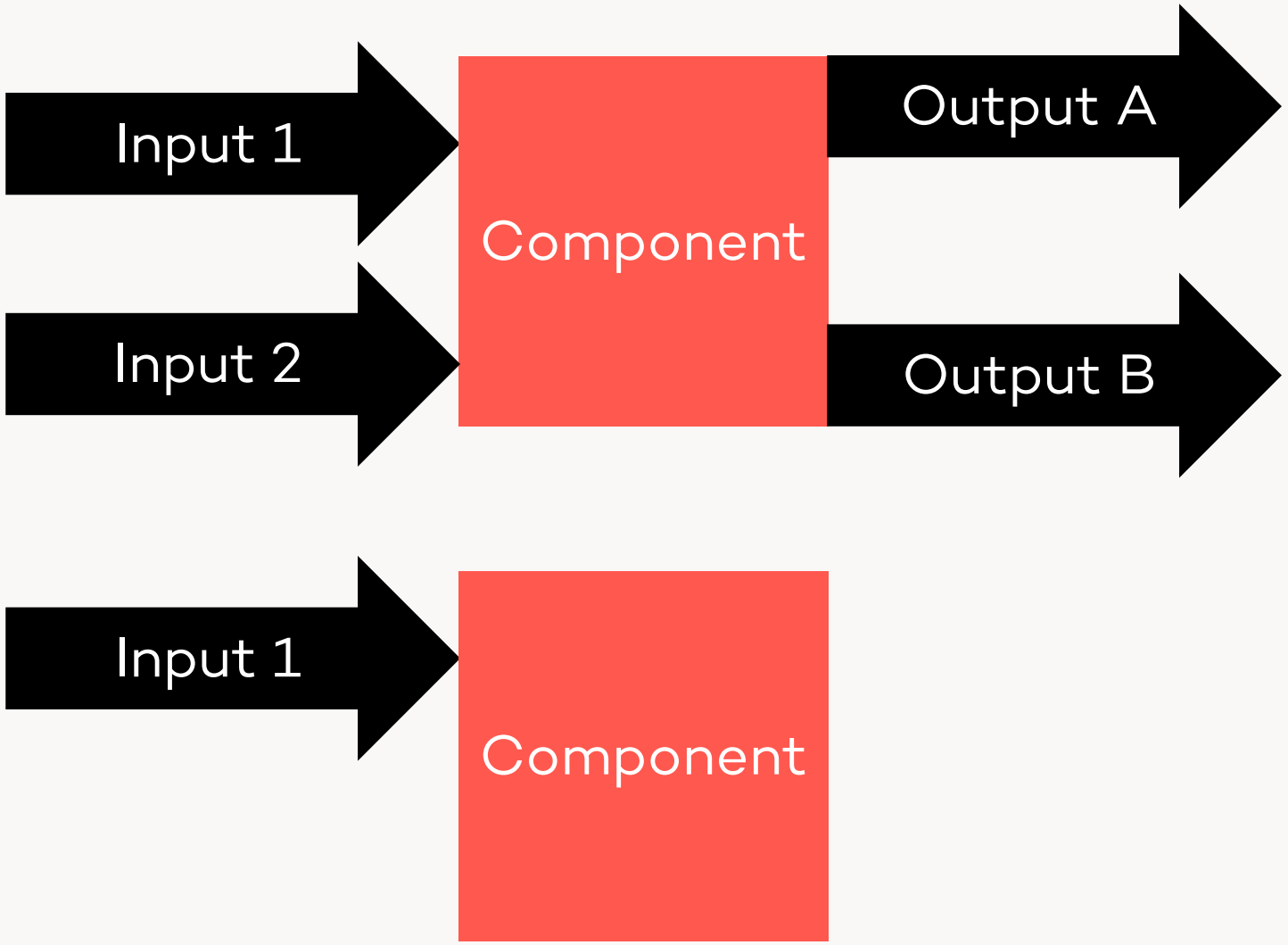


UI-related
Re-usable
Custom DOM element
HTML Template

```
<app-todo></app-todo>
```

Components

Principle



Components

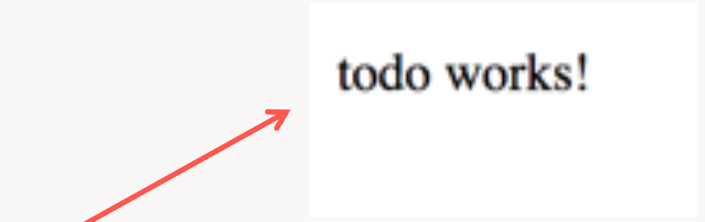
```
@Component({
  selector: 'app-todo',
  templateUrl: './todo.component.html',
  //template: '<strong>inline template</strong>',
  styleUrls: ['./todo.component.css']})
export class AppComponent {}
```

Components

Usage

```
<app-todo></app-todo>
```

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



Components

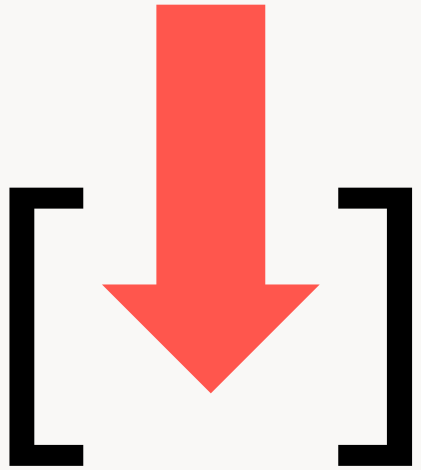
LAB #4

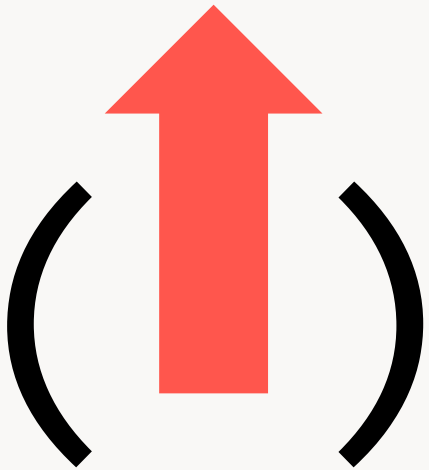
- Create a new component
- Use the new component in your AppComponent's template

8. Input/Output

Input/Output

Bindings for component state/events





Input

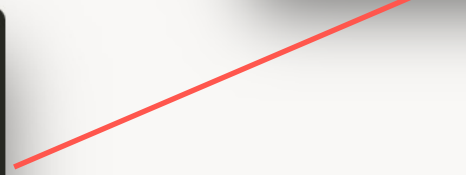
We want to pass arbitrary data to components:

```
<app-todo [todo]="myTodo"></app-todo>
```

Input

Component Perspective

```
<app-todo [todo]="myTodo"></app-todo>
```



```
@Input()
public todo;

@Input('todo')
public todoX;
```

Input

Property Bindings

If you want to bind a static string to a property, you can use a simplified form by leaving out the square brackets.

```
<!-- Instead of -->
<p [title]="'test'"></p>

<!-- Use -->
<p title="test"></p>
```

Input

Property Bindings

If you want to react to changes (i.e. new value or updated reference) of the property binding, use TypeScript field setters:

```
@Input()
public set todo(value) {
    // do something with value
}
```

Output

We want to get informed about custom events:

```
<app-todo (done)="onDone()"></app-todo>
```

Output

Component Perspective

```
<app-todo (done)="onDone($event)"></app-todo>
```

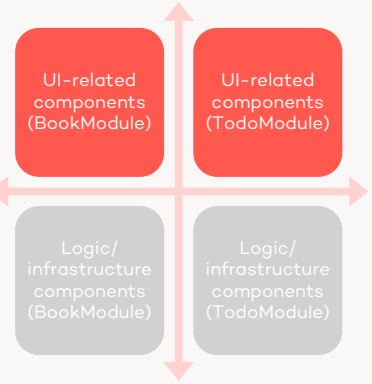
```
@Output()
public done = new EventEmitter<any>(); // done.next(todo);

@Output('done')
public doneEmitter = new EventEmitter<any>();
```

Input/Output

LAB #5

- Input
- Output

9. Directives



UI-related

Re-usable

Manipulate styling or behaviour of a DOM element

Or: Manipulate DOM structure (not covered here)

```
<app-todo myDirective></app-todo>
```

Directives

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

@Directive({ selector: '[appTest]' })
export class TestDirective {
  @Input() public color: string;
}
```

```html
<div appTest [color]="myColor"></div>
<app-todo appTest color="green"></app-todo>
```

HostBinding

```
<app-todo [style.backgroundColor]="color"></app-todo>
```

```
@Component({ selector: 'app-todo', /* … */ })
export class TodoComponent {
  @HostBinding('style.backgroundColor')
  public color = 'hotpink';
}
```

HostListener

```
<app-todo (click)="onClick()"></app-todo>
```

```
@Component({ selector: 'app-todo', /* … */ })
export class TodoComponent {
  @HostListener('click')
  public onClick() {}
}
```

Directives

LAB #6

- Create a color directive
- Create a click directive

10. Dependency Injection

Dependency Injection

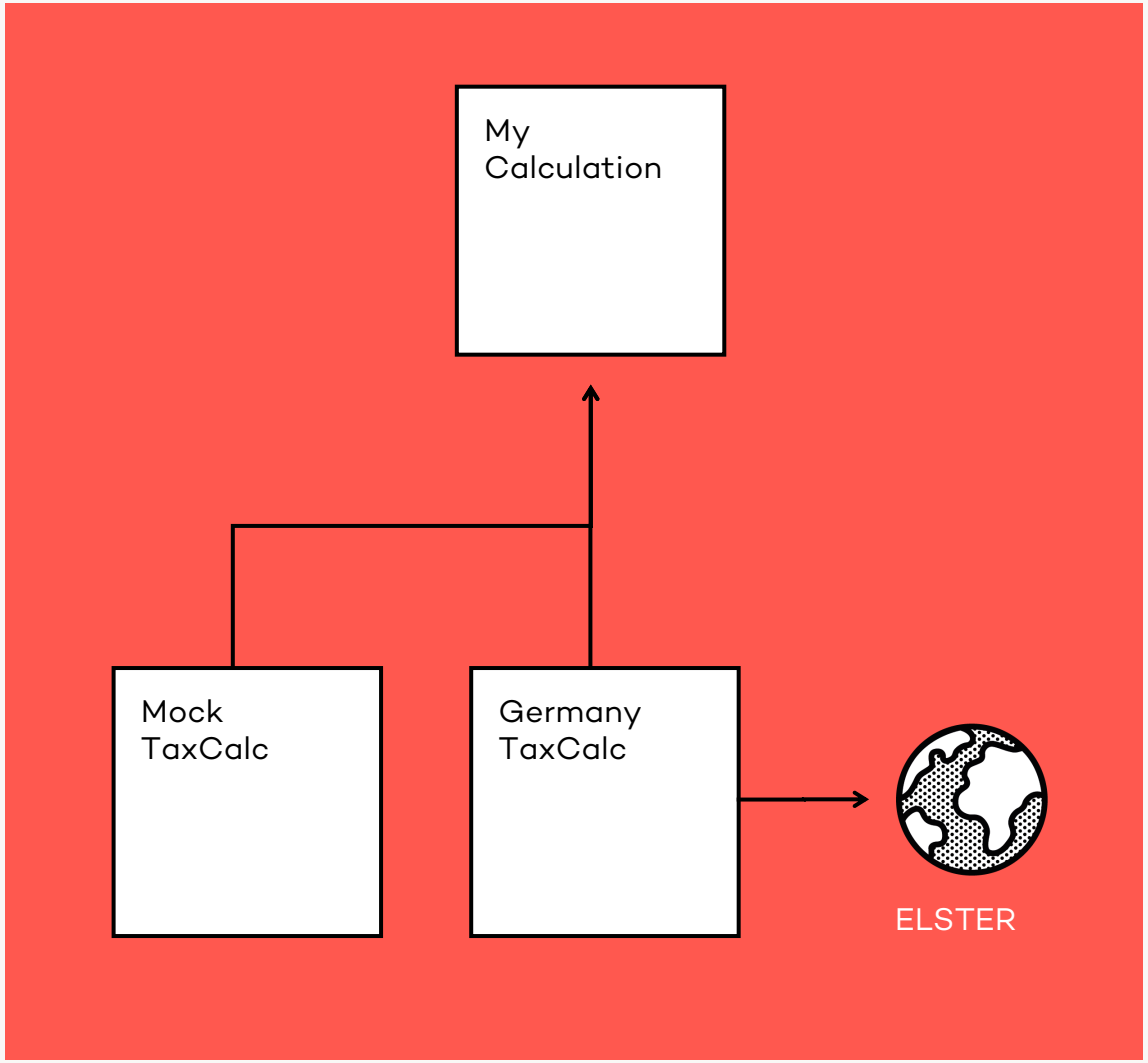
Motivation

- Low Coupling, High Cohesion
- In a unit test, a developer is not interested in arranging a test setup for a complete tax calculation or even performing it.
- A developer might be interested in switching between different strategies (e.g. a different tax calculation for Germany and Austria)
- Thus: A component should not arrange its dependencies on its own (Inversion of Control, IoC) but rely on an external party instead (dependency injection container/IoC container)

Dependency Injection

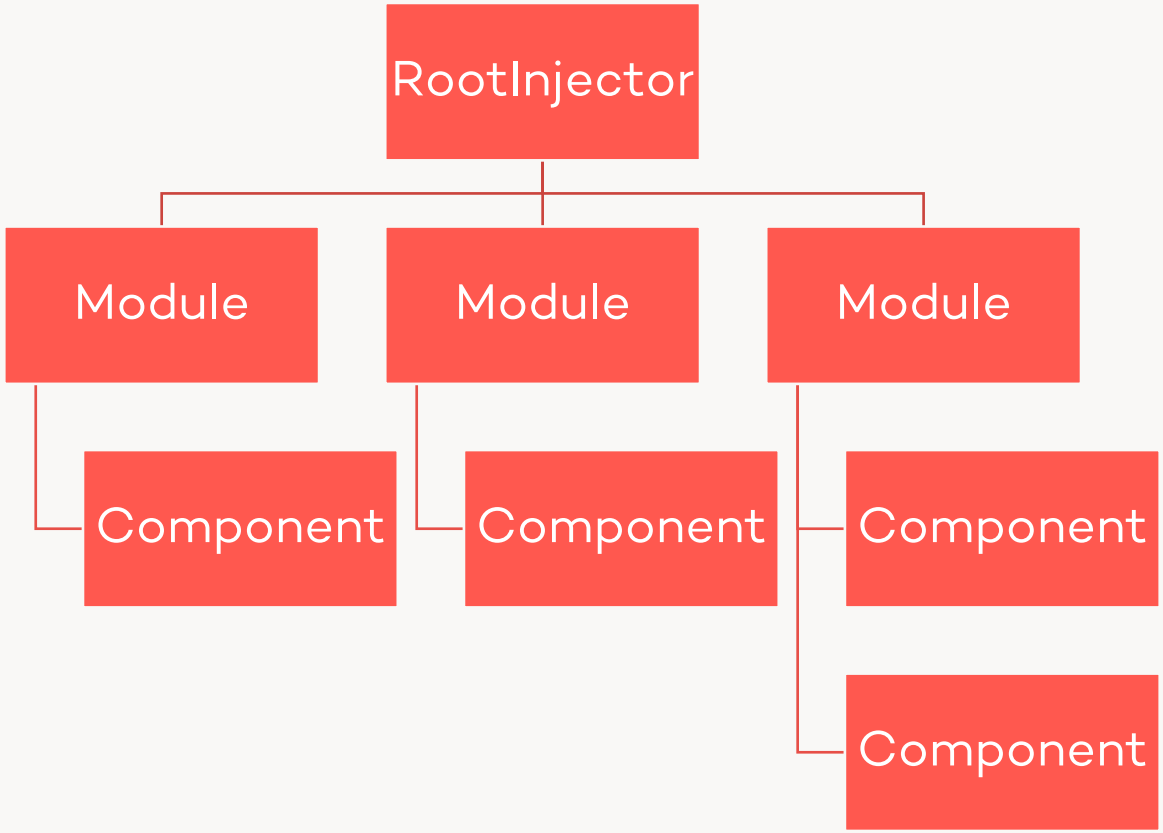
Goal

- DI container is aware of environment
- Sets up dependencies accordingly
- Low Coupling, High Cohesion



Angular DI

Dependency Tree



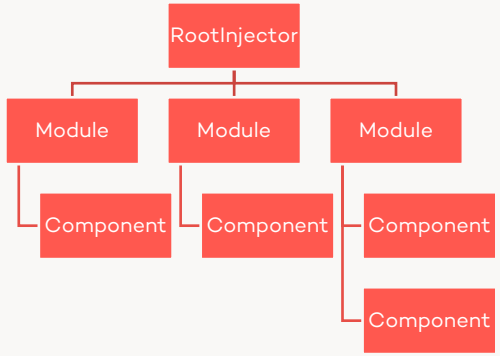
Angular DI

Dependency Tree

- Angular Dependency Injection is type-based
- Only classes can be used as providers and injectables, as interfaces vanish during TypeScript transpilation
- Alternative for non-class dependencies: InjectionTokens
- Classes have to be marked as `@Injectable()` if they want to request dependencies
- Dependencies can be requested by simply using them as a constructor parameter

Angular DI

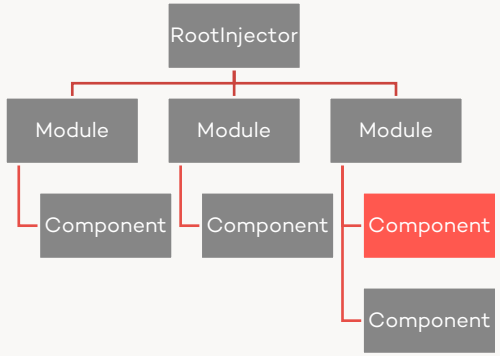
Self-Register as an Application-wide Singleton




```
@Injectable({
    providedIn: 'root'
})
export class TaxCalculation {}
```

Angular DI

Providing Dependencies




```
@Component({
    providers: [GermanTaxCalculation]
})
export class TaxComponent {}
```

Angular DI

Consuming Dependencies

```
@Injectable()
export class MyCalculation {
    constructor(taxCalc: TaxCalculation) {}
}
```

Throws an error if dependency cannot be resolved!

Angular DI

Consuming Dependencies

```
@Injectable()
export class MyCalculation{
  readonly taxCalc = inject(TaxCalculation);
}
```

Dependency Injection

Sample

`ElementRef` allows accessing the native rendering host element of the directive or component

Retrieved via Dependency Injection

```
constructor(elRef: ElementRef) {}
```

Angular DI

Storing Dependencies

```
export class FooComponent {
  private readonly elementRef: ElementRef;

  constructor(elementRef: ElementRef) {
    this.elementRef = elementRef;
  }
}
```

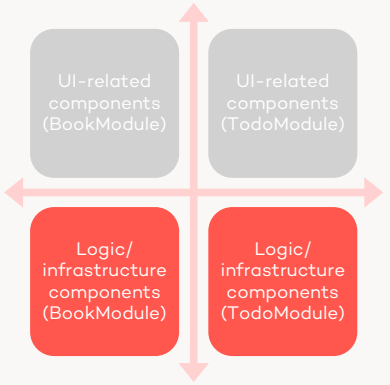
Angular DI

Storing Dependencies

```
export class FooComponent{
  //private readonly elementRef: ElementRef;

  constructor(private readonly elementRef:ElementRef){}
}
```

Useful TypeScript feature: Constructor parameter + access modifier automatically creates a field with the same name and type

11. Services



Not UI-related

Re-usable

Contain common (domain-specific) logic

Contain infrastructure (non-domain-specific) code

Service

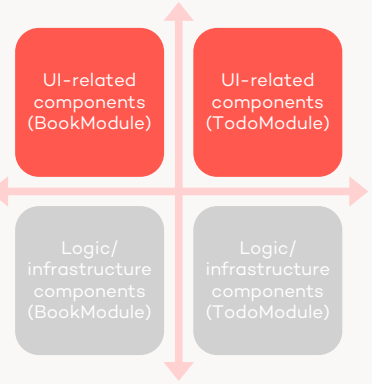
```ts
import { Injectable } from '@angular/core';

@Injectable()
export class TodoService {
  constructor() { }
}
```

Dependency Injection/Services

LAB #7

- Injecting ElementRef
- Injection Tokens
- Create a new service

12. Structural Directives



UI-related
Re-usable
Manipulate DOM structure

```
<app-todo *myDirective></app-todo>
```

*ngIf

Conditionally Include Node in DOM

```
<button (click)="toggle()">Toggle</button>
<div *ngIf="show">
    I'm visible!
</div>
```

*ngSwitchCase

Multiple conditions

```
<div [ngSwitch]="user.language">
    <div *ngSwitchCase="DE">Hallo, {{ user.name }}!</div>
    <div *ngSwitchCase="ES">¡Hola, {{ user.name }}!</div>
    <div *ngSwitchDefault>Hello, {{ user.name }}!</div>
</div
```

```
<div>
  <div>¡Hola, {{ user.name }}!</div>
</div>
```

*ngFor

Repeat DOM Node

```
<ul>
    <li *ngFor="let todo of todos">{{ todo.name }}</li>
</ul>
```

*ngFor

Repeat DOM Node

```
[
        "Wash my clothes"
        "Tidy up the room"
        "Mine bitcoin"
]
```

```
<ul>
    <li>Wash my clothes</li>
    <li>Tidy up the room</li>
    <li>Mine bitcoin</li>
</ul>
```

*ngFor

LAB #8

- *ngIf
- *ngFor

13. Observables & RxJS

Observables

Motivation

Obviously, not all use cases can be solved synchronously

When we are using our to-do API, this will be an asynchronous task (due to network roundtrip)

For a fast and fluid user experience, everything that could potentially take longer than 16ms (=> 60 fps) should be done **asynchronously**!

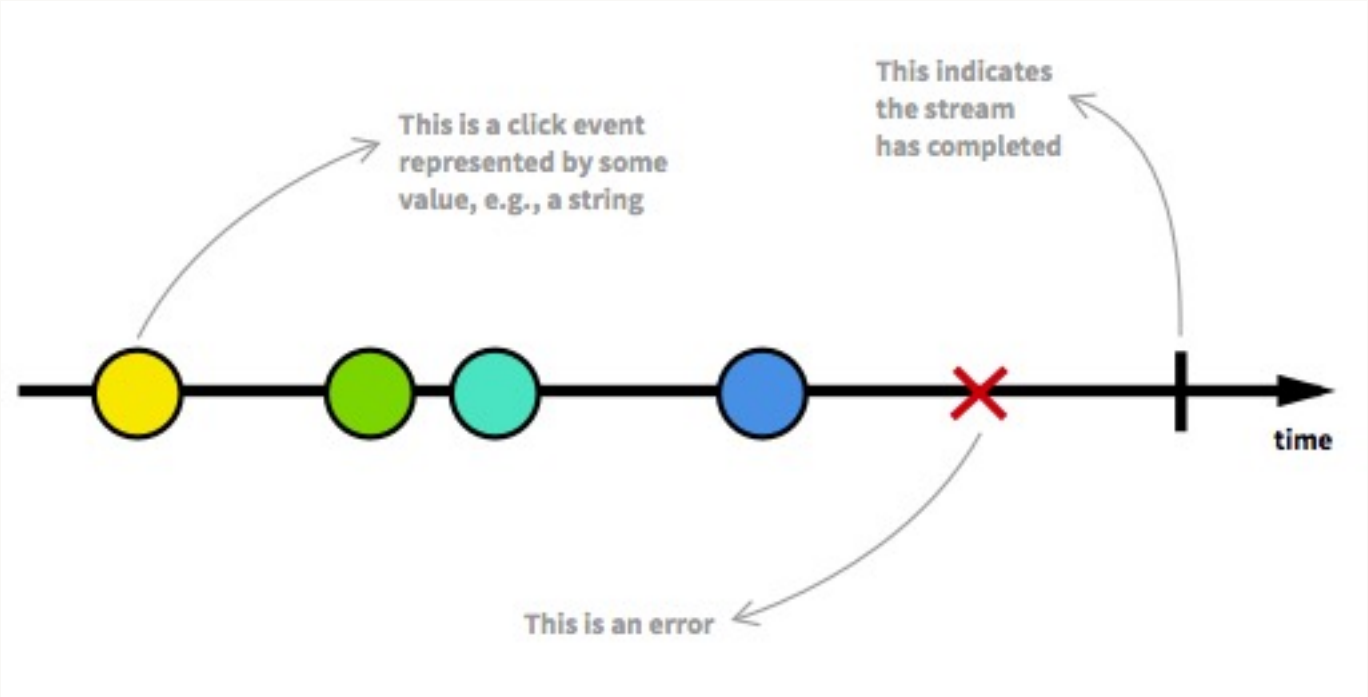
Observables

Motivation

	Callback	Promise	Observable
Execution	Eager	Eager	Lazy
Values	Multiple	Single	Multiple
Cancelable	No	No*	Yes
Composable	No	Yes	Yes

Observables

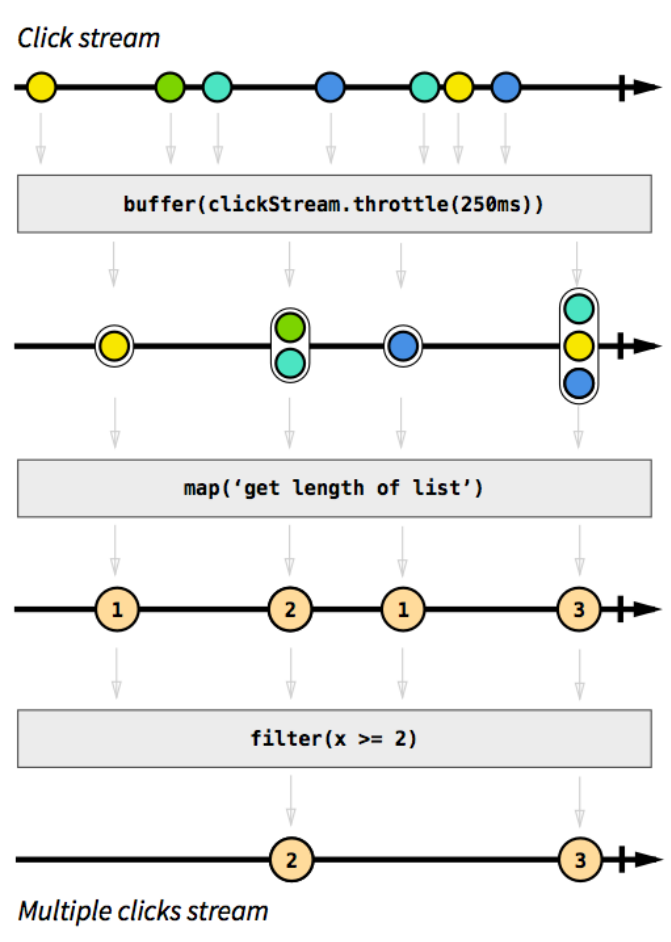
"Data over Time"



https://gist.github.com/staltz/868e7e9bc2a7b8c1f754 (abgerufen am 23.05.2018)

Observables

Reactive



https://gist.github.com/staltz/868e7e9bc2a7b8c1f754 (abgerufen am 23.05.2018)

RxJS

Originally known as Reactive Extensions for .NET

Open-Source

Published for JavaScript, Java, ...

High-Level Flow Composition

Provides an Observable implementation

Provides operators (map, throttle, ...)

Observables

Upgrade Synchronous Values to an Observable

```
of(value)       → Promise.resolve(value)
throwError(err) → Promise.reject(err)
```

Observables

Subscribe

```
public todos: Todo[];

//this.todos = todoService.getAll();

todoService.getAll().subscribe(todos => this.todos = todos);
```

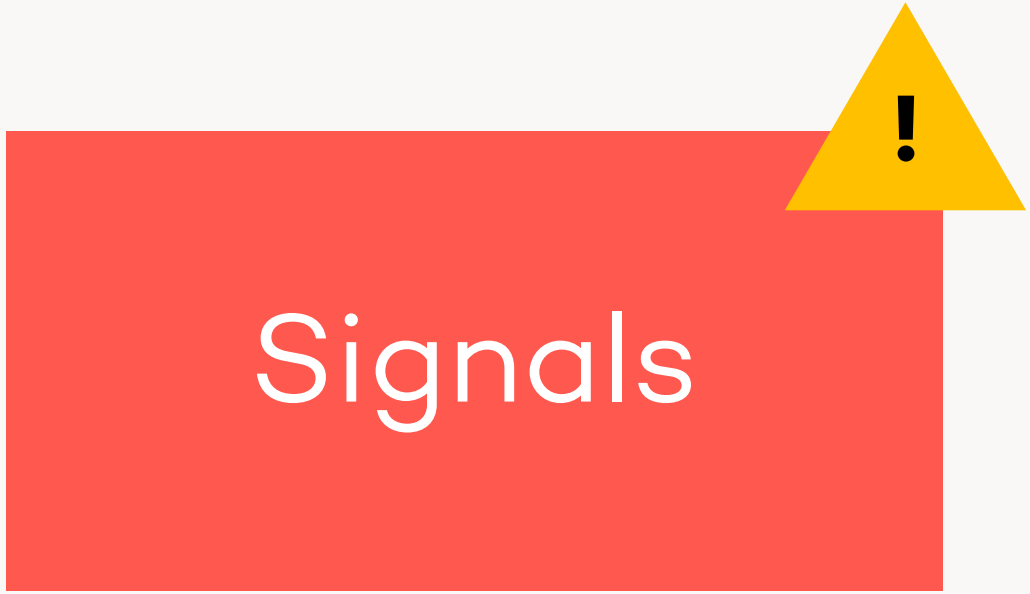
Observables

Unsubscribe

```
private subscription: Subscription;

this.subscription = someHotObservable.subscribe();

public ngOnDestroy(): void {
  this.subscription.unsubscribe();
}
```

Observables

Future in Angular



14. HttpClient

HttpClient

Infrastructure service provided by Angular's `@angular/common/http` package.

Allows setting up HTTP Requests using a TypeScript-friendly interface:

- get(url)
- post(url, data)
- put(url, data)
- delete(url)

Observables

Service API

LAB #9

- Adjust service
- Use HttpClient

15. Lifecycle Hooks

Lifecycle Hooks

- “Technical” lifecycle hook
- Called on object construction
- Assign fields synchronously
- Called once

1. constructor
2. ngOnChanges
3. ngOnInit
4. ngDoCheck
5. ngAfterContentInit
6. ngAfterContentChecked
7. ngAfterViewInit
8. ngAfterViewChecked
9. ngOnDestroy

Lifecycle Hooks

- Called when bound properties (Input/Output) change
- Event parameters: SimpleChanges (contains previous and current values)
- Purpose: React to changes of bound properties
- Called repeatedly

1. constructor
2. ngOnChanges
3. ngOnInit
4. ngDoCheck
5. ngAfterContentInit
6. ngAfterContentChecked
7. ngAfterViewInit
8. ngAfterViewChecked
9. ngOnDestroy

Lifecycle Hooks

- Called after constructor and first ngOnChanges call
- Purpose: Launch asynchronous tasks/offload complex initialization from constructor
- No parameters
- Called once

1. constructor
2. ngOnChanges
3. ngOnInit
4. ngDoCheck
5. ngAfterContentInit
6. ngAfterContentChecked
7. ngAfterViewInit
8. ngAfterViewChecked
9. ngOnDestroy

Lifecycle Hooks

- Called whenever change detection is executed (“check”)
- Purpose: React to any change in general
- No parameters
- Called repeatedly

1. constructor
2. ngOnChanges
3. ngOnInit
4. ngDoCheck
5. ngAfterContentInit
6. ngAfterContentChecked
7. ngAfterViewInit
8. ngAfterViewChecked
9. ngOnDestroy

Lifecycle Hooks

- Called after the content (i.e. components/directives and subcomponents/subdirectives) has been initialized
- Purpose: Access directives in the component's content
- No parameters
- Called once

1. constructor
2. ngOnChanges
3. ngOnInit
4. ngDoCheck
5. ngAfterContentInit
6. ngAfterContentChecked
7. ngAfterViewInit
8. ngAfterViewChecked
9. ngOnDestroy

Lifecycle Hooks

- Called whenever change detection has been executed on the content
- Purpose: React to any changes in the content
- No parameters
- Called repeatedly

1. constructor
2. ngOnChanges
3. ngOnInit
4. ngDoCheck
5. ngAfterContentInit
6. ngAfterContentChecked
7. ngAfterViewInit
8. ngAfterViewChecked
9. ngOnDestroy

Lifecycle Hooks

- Called after the view (i.e. components/directives and subcomponents/subdirectives) has been initialized
- Purpose: Access directives in the component's view
- No parameters
- Called once

1. constructor
2. ngOnChanges
3. ngOnInit
4. ngDoCheck
5. ngAfterContentInit
6. ngAfterContentChecked
7. ngAfterViewInit
8. ngAfterViewChecked
9. ngOnDestroy

Lifecycle Hooks

- Called whenever change detection has been executed on the view
- Purpose: React to any changes in the view
- No parameters
- Called repeatedly

1. constructor
2. ngOnChanges
3. ngOnInit
4. ngDoCheck
5. ngAfterContentInit
6. ngAfterContentChecked
7. ngAfterViewInit
8. ngAfterViewChecked
9. ngOnDestroy

Lifecycle Hooks

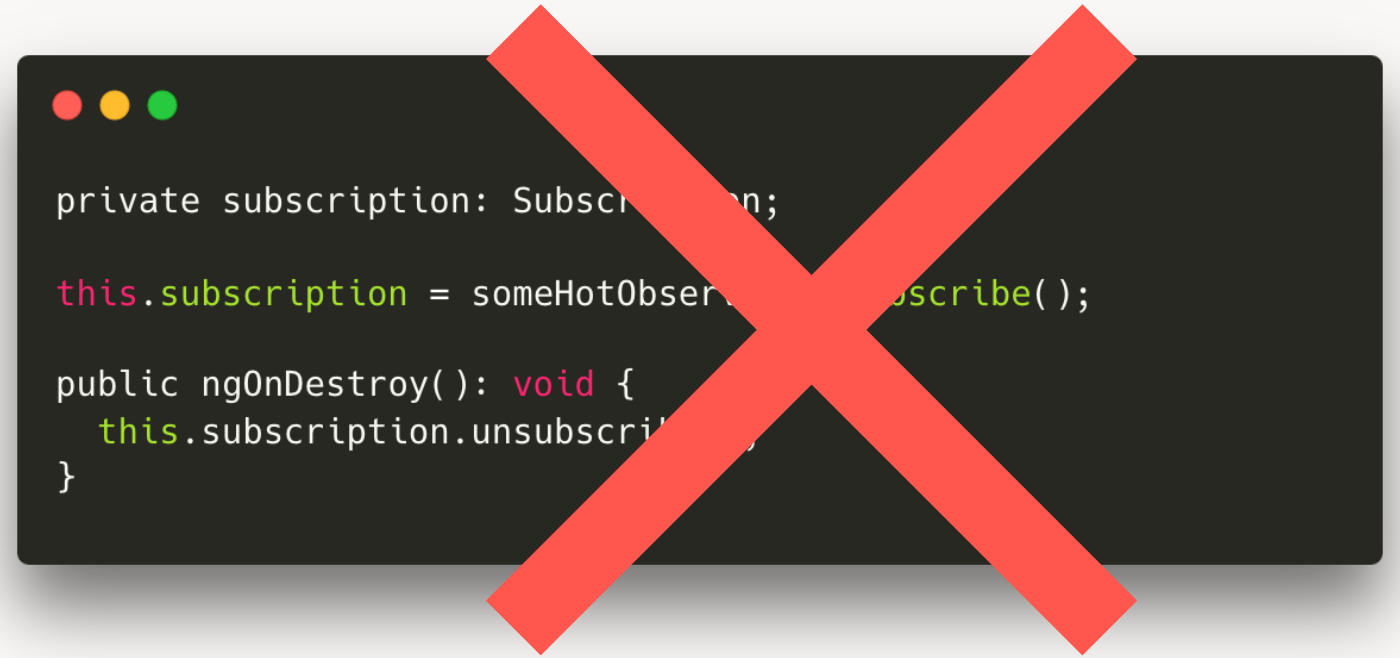
- Called before the component/directive is destroyed
- Purpose: Clean-up (unsubscribe from observables, unregister from events, ...)
- No parameters
- Called once

1. constructor
2. ngOnChanges
3. ngOnInit
4. ngDoCheck
5. ngAfterContentInit
6. ngAfterContentChecked
7. ngAfterViewInit
8. ngAfterViewChecked
9. ngOnDestroy

16. Async Pipe

Observables

Unsubscribe



Async Pipe

Let Angular handle subscribing and unsubscribing for you!

Makes handling observables as easy as handling synchronous values.

Async Pipe

Step 1: Use Observable fields

```
//public todos: Todo[];
public todos$: Observable<Todo[]>;

//todoService.getAll().subscribe(todos => this.todos = todos);
//this.todos = todoService.getAll();
this.todos$ = todoService.getAll();
```

Async Pipe

Step 2: Adjust binding

```
<div *ngIf="todos$ | async as todos">
    You have {{ todos.length }} todos!
</div>

<app-todo *ngFor="let todo of todos$ | async" [todo]="todo">
</app-todo
```

Async Pipe

LAB #10

- Using the async pipe

17. Routing

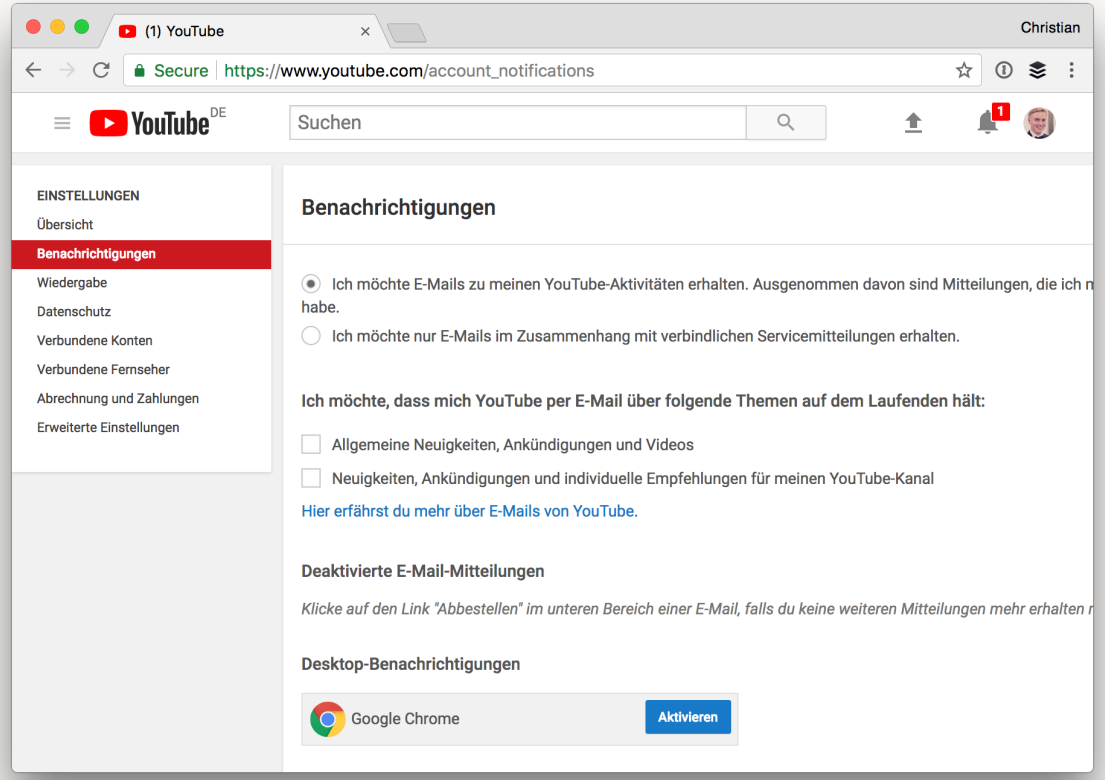
Routing

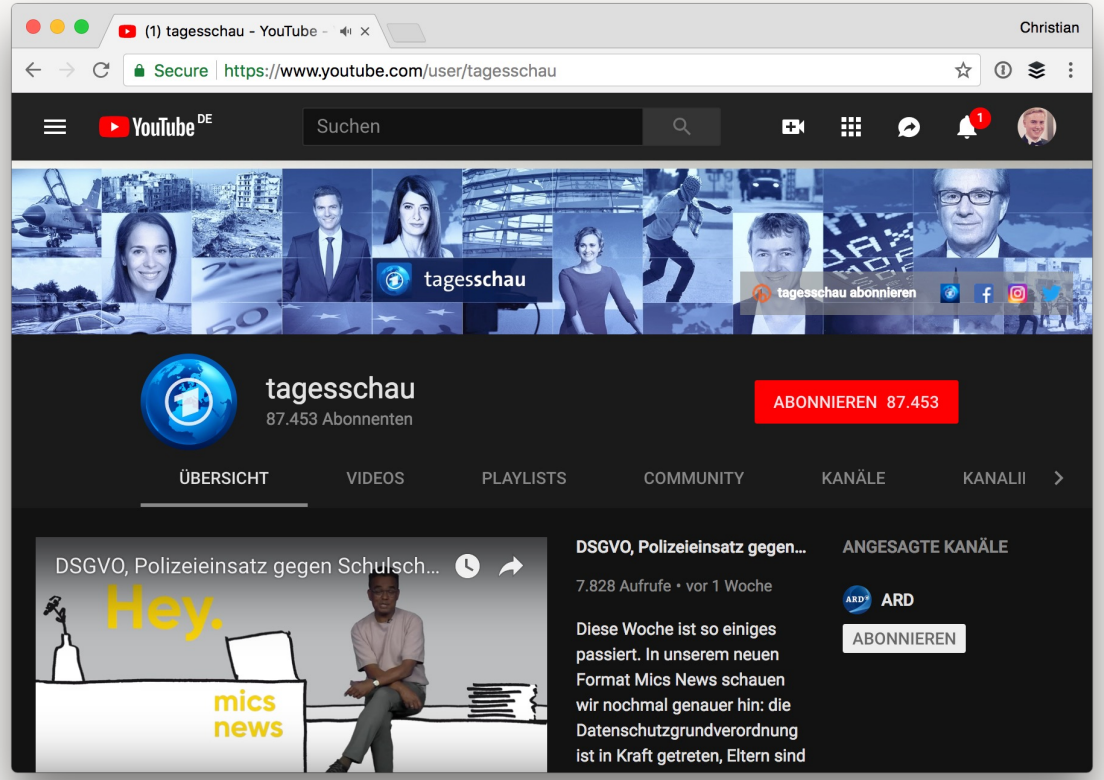
We want to map an application state to a certain URL

Problem: There's no server-side roundtrip

So we need a different method to map app states to a URL

Routing


(1) YouTube
Christian
Secure https://www.youtube.com/account_notifications
YouTube DE
Suchen
EINSTELLUNGEN
Übersicht
Benachrichtigungen
Wiedergabe
Datenschutz
Verbundene Konten
Verbundene Fernseher
Abrechnung und Zahlungen
Erweiterte Einstellungen
Benachrichtigungen
Ich möchte E-Mails zu meinen YouTube-Aktivitäten erhalten. Ausgenommen davon sind Mitteilungen, die ich n
habe.
Ich möchte nur E-Mails im Zusammenhang mit verbindlichen Servicemitteilungen erhalten.
Ich möchte, dass mich YouTube per E-Mail über folgende Themen auf dem Laufenden hält:
Allgemeine Neuigkeiten, Ankündigungen und Videos
Neuigkeiten, Ankündigungen und individuelle Empfehlungen für meinen YouTube-Kanal
Hier erfährst du mehr über E-Mails von YouTube.
Deaktivierte E-Mail-Mitteilungen
Klicke auf den Link "Abbestellen" im unteren Bereich einer E-Mail, falls du keine weiteren Mitteilungen mehr erhalten
Desktop-Benachrichtigungen
Google Chrome
Aktivieren


(1) tagesschau - YouTube
Christian
Secure https://www.youtube.com/user/tagesschau
YouTube DE
Suchen
tagesschau
tagesschau abonnieren
tagesschau
87.453 Abonnenten
ABONNIEREN 87.453
ÜBERSICHT
VIDEOS
PLAYLISTS
COMMUNITY
KANÄLE
KANALII
DSGVO, Polizeieinsatz gegen Schulsch...
Hey.
mics
news
DSGVO, Polizeieinsatz gegen...
7.828 Aufrufe • vor 1 Woche
Diese Woche ist so einiges passiert. In unserem neuen Format Mics News schauen wir nochmal genauer hin: die Datenschutzgrundverordnung ist in Kraft getreten, Eltern sind
ANGESAGTE KANÄLE
ARD
ABONNIEREN

Routing

Strategies

PathLocationStrategy: Use HTML5-based history routing (default)

`https://localhost:4200/home/users/peter`

Note: Requires server-side rewriting!

HashLocationStrategy: Use classic hash-based routing

`https://localhost:4200/#/home/users/peter`

Typically used for cross-platform builds (Electron, Cordova)

Routing

Root module

```
@NgModule({
  declarations: [
    AppComponent,
    TodoComponent
  ],
  imports: [
    BrowserModule,
    RouterModule.forRoot([ /* Routes */ ], { useHash: false })
  ],
  providers: [],
  bootstrap: [AppComponent]})
export class AppModule { }
```

Optional,
default: false

Routing

Child module

```
@NgModule({
  declarations: [
    // Some Components / Pipes / Directives
  ],
  imports: [
    RouterModule.forChild([ /* Routes */ ])
  ],
  providers: [],
export class OtherModule { }
```

Routing

Routes

```
{ path: 'home', component: HomeComponent }
```

Maps a static path to a component

http://localhost:4200/home --> HomeComponent

Routing

Routes with Parameters

```
{ path: 'todos/:id', component: DetailComponent }
```

Route parameter matches any string

http://localhost:4200/todos/123 --> DetailComponent

http://localhost:4200/todos/abc --> DetailComponent

Routing

Route Redirects

```
{ path: '', redirectTo:'home', pathMatch: 'full' }
```

Routing

Wildcard Route

```
{ path: '**', component: NotFoundComponent }
```

The wildcard route matches everything and must be the last route.

Routing

RouterOutlet

```
<router-outlet></router-outlet>
<resolved-component></resolved-component>
```

Routing

Mind the route order

Routing

Creating Links

```
<a href="home">My link</a>
```

Problem: How to assign a dynamic route?

Routing

Creating Links

```
<!-- <a href="home">My link</a> -->
<a [href]="'todos' + todo.id">{{ todo.name }}</a>
```

Problem: How to switch between hash and path-based routing?

Routing

RouterLink

```
<!-- Text-based -->
<a [href]="'todos' + todo.id">{{ todo.name }}</a>

<!-- Data-based -->
<a *ngFor="let todo of todos" [routerLink]="['todos', todo.id]">
{{ todo.name }
</a>
```

Routing

Advanced RouterLink

```
<!-- Applies the CSS class “my-active” when the “todos” route is selected -->
<a [href]="'todos' + todo.id">{{ todo.name }}</a>

<!-- Note: This also applies to any child route of “todos”.
If you want an exact match, use: -->
<a routerLink="/todos" routerLinkActive="my-active"
 [routerLinkActiveOptions]="{ exact: true }">Home</a>
```

Routing

Accessing Route Parameters

```
constructor(activatedRoute: ActivatedRoute) {
    activatedRoute.params.subscribe(p => /* … */);
}
```

Note: There's also an activated route snapshot. This snapshot isn't updated when a route parameter changes. Hence, you should avoid using the snapshot.

Routing

Accessing Route Parameters

```
public readonly todo$: Observable<Todo>;

constructor(activatedRoute: ActivatedRoute,
            todoService: TodoService) {

  this.todo$ = this.activatedRoute.params.pipe(
    pluck('id'),
    switchMap(id => todoService.get(+id))
  );

}
```

Routing

Accessing Route Parameters

LAB #11

- Generate components
- Define routes
- Router outlet
- Router links
- Active router links
- Activated route

Routing

Programmatic Routing

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

router.navigate([123], {relativeTo: activatedRoute});
router.navigateByUrl('/todos');
```

Routing

Child Routes

```
{
  path: 'todos',
  component: TodoComponent,
  children: [{
    path: '',
    component: TodoListComponent
  }, {
    path: ':id',
    component: TodoDetailComponent
  }]
}
```

Routing

Child Routes

```
{
  path: 'todos',
  component: TodoComponent,
  children: [{
    path: '',
    component: TodoListComponent
  }, {
    path: ':id',
    component: TodoDetailComponent
  }]
}
```

http://localhost:4200/todos

app.component.ts

<router-outlet></router-outlet>

todo.component.ts

<router-outlet></router-outlet>

todo-list.component.ts

Routing

Not Covered Here

- Lazy Loading
- Preloading Strategies
- Aux Routes
- Guards
- Resolvers
- Router Events

18. Template-Driven Forms

Template-Driven Forms

Module Import

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

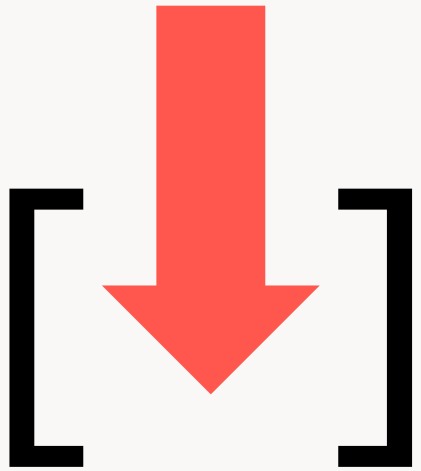
Template-Driven Forms

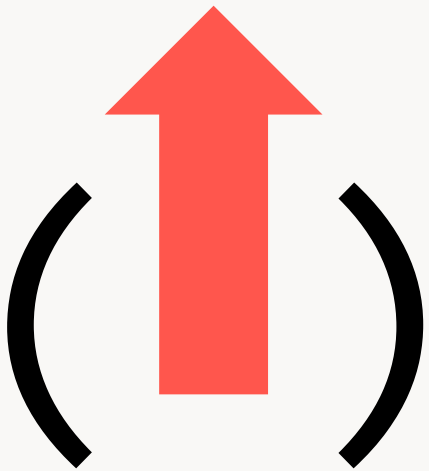
Two-Way Binding

```
<form>
    <input type="checkbox"
    [ngModel]="todo.done"
    (ngModelChange)="todo.done = $event">
</form>
```

Template-Driven Forms

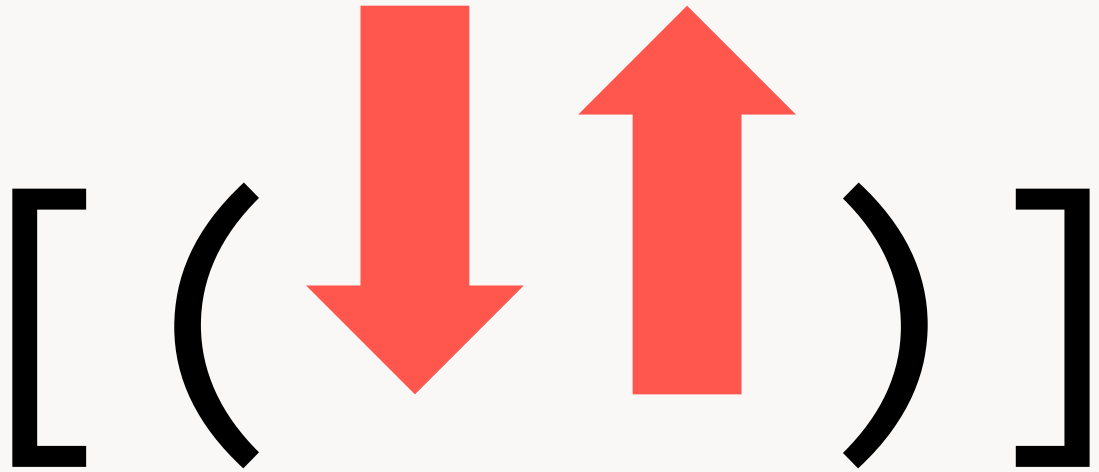
Two-Way Binding





Template-Driven Forms

Two-Way Binding



Template-Driven Forms

Field Definition

```
<form>
  <input type="checkbox" [(ngModel)]="todo.done" name="done">
  <input type="text" [(ngModel)]="todo.name" name="name">
</form>
```

Template-Driven Forms

Validation

```
<form>
  <input type="checkbox" [(ngModel)]="todo.done" name="done">
  <input type="text" [(ngModel)]="todo.name" name="name" required>
</form>
```

Template-Driven Forms

Validators

- required
- maxlength
- minlength
- min
- max
- pattern
- email

Template-Driven Forms

Template Reference Variables

```
<!-- Accessing properties on the native element -->
<div #myDiv>{{ myDiv.nodeName }}</div>

<!-- Accessing directives on an element -->
<form #myForm="ngForm">{{ myForm.valid }}</form>
```

Template-Driven Forms

LAB #12

- Use a form
- Validation

Template-Driven Forms

Form States

State	Opposite	Description
ng-touched	ng-untouched	Control had focus
ng-dirty	ng-pristine	Control value was changed
ng-valid	ng-invalid	Control value is valid
ng-pending		Async validation is pending

Template-Driven Forms

```
<!-- Update bound value when leaving the field -->
<input type="text" [(ngModel)]="todo.name"
       name="name"[ngModelOptions]="{ updateOn: 'blur' }"/>
```

Update bound value on submit

```
[ngModelOptions]="{ updateOn: 'submit' }"
```

Update bound value on change (default)

```
[ngModelOptions]="{ updateOn: 'change' }"
```

Template-Driven Forms

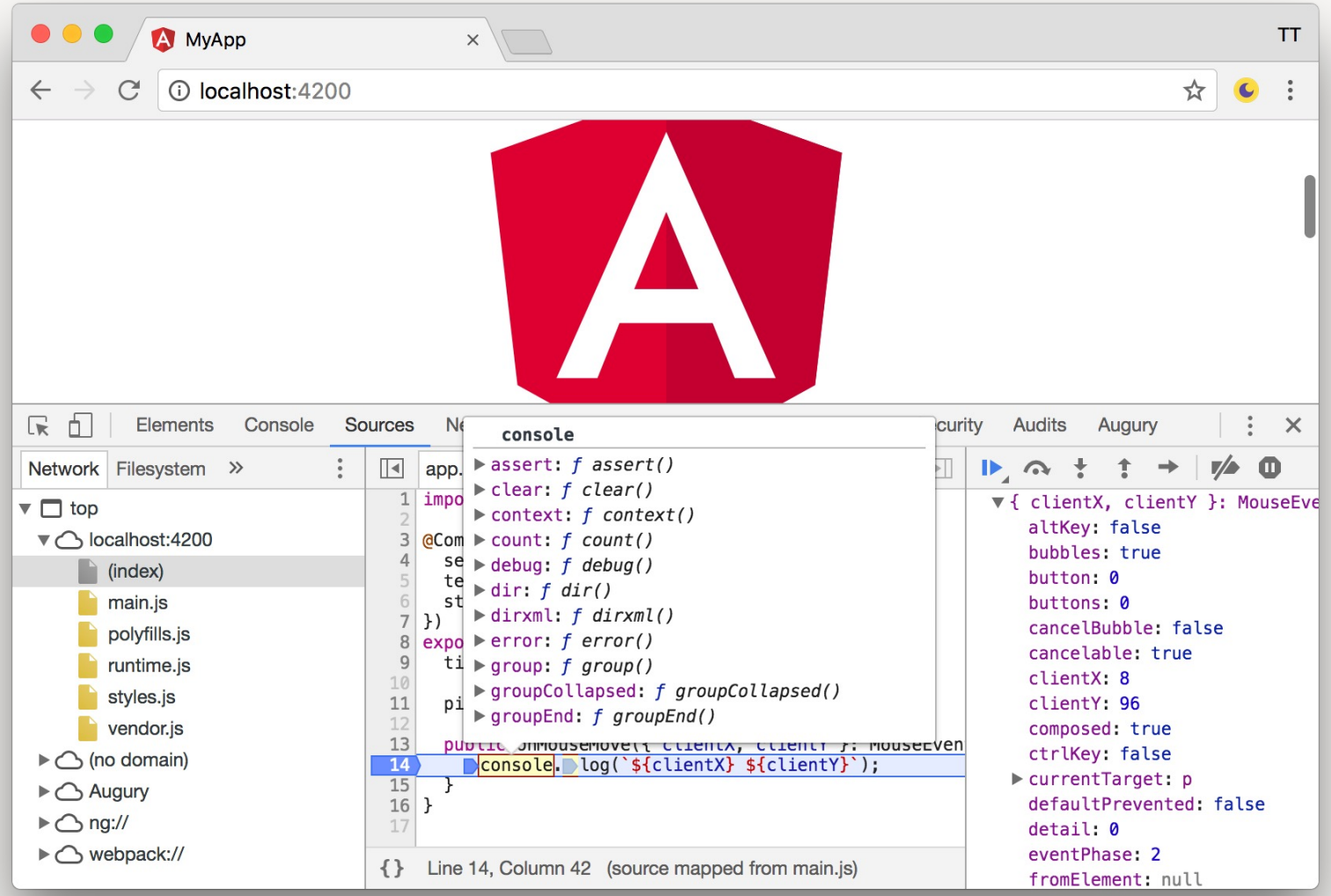
ngFormOptions

Default update method can also be set on form level:

```
<form [ngFormOptions]="{ updateOn: 'blur' }"></form>
```

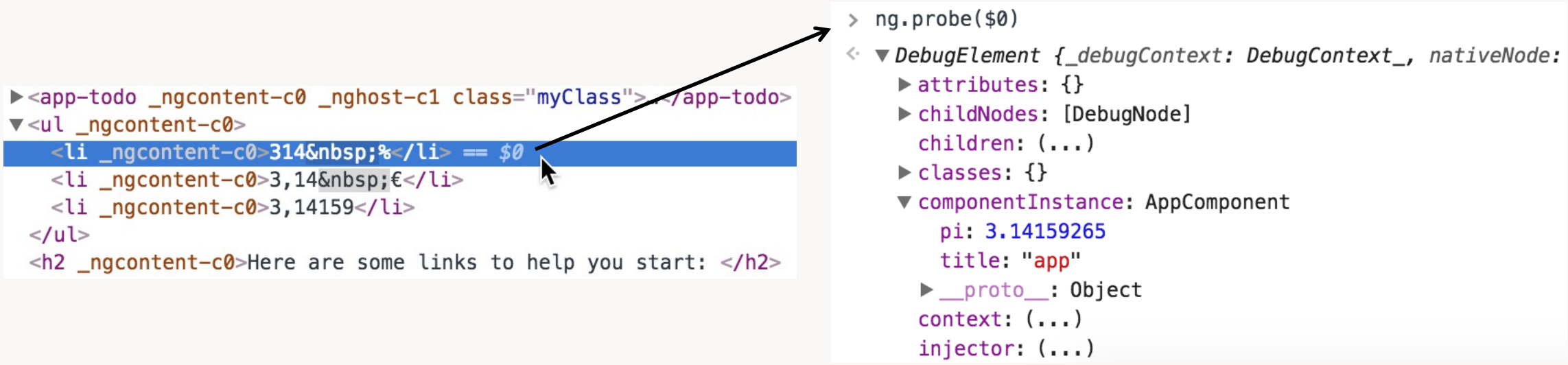
19. Debugging

Chrome DevTools



Debugging

ng.probe



Debugging

Angular DevTools

Chrome DevTools Extension

https://angular.io/guide/devtools

Bonus Material

Tour of Heroes

https://angular.io/tutorial

Thank you

for your kind attention.

think
tecture