ng commands

Create new angular project

Creates a new Angular project.

> *ng new project <projectname>*

Start development server

Start Angular live devolopment server.

> *ng serve*

ng generate component

Create a full component in src folder

> *ng generate component <name>*

oder

> *ng g c <name>*

npm commands

Install bootstrap

Install newest version of bootstrap

> *npm install --save bootstrap*

Fix severity vulnerabilities

Fix severity vulnerabilities in project.

> *npm audit fix*

manual project changes

add bootstrap to new project

Add "./node_modules/bootstrap/dist/css/bootstrap.min.css" line in .angular-cli.json file in app.styles array.

> see example 1.1

angular

1 Getting started

Setup Development Environment

```
1. Get newest NodeJs from nodejs.org
2. run _npm install -g npm_
3. run _npm uninstall -g angular/cli_
4. run _npm cache clean_
5. run _npm install -g @angular/cli_
```

2 Basics

Component

Databinding: ngModel

> *<input type="text" [(ngModel)]="name">*

It accepts a domain model as an optional Input. If you have a one-way binding to ngModel with [] syntax, changing the value of the domain model in the component class sets the value in the view. If you have a two-way binding with [()] syntax (also known as 'banana-box syntax'), the value in the UI always syncs back to the domain model in your class.

Databinding: string interpolation

One way databinding from model to view

> *{{propertyName}}*

Directive: ngIf

> _<div *ngIf="condition">
>
> > Content to render when condition is true.
>
> _

Directive: ngFor

Example 1

> *<app-server *ngFor="let server of servers"></app-server>*

Example 2

> <div
>
> > *ngFor="let logItem of log; let i = index"..
> > [ngStyle]="{backgroundColor: i >= 4 ? 'blue' : 'transparent'}"

```
    [ngClass]="{'white-text': i >= 4}">
    {{ logItem }}
</div>
```

Directive: ngClass

```
<p
    [ngClass]="{online: serverStatus === 'online'}">
    {{ 'Server' }} with ID {{ serverId }} is {{ getServerStatus() }}
</p>
```

Directive: ngStyle

```
<p
    [ngStyle]="{backgroundColor: getColor()}"
    {{ 'Server' }} with ID {{ serverId }} is {{ getServerStatus() }}
</p>
```

3 Course Project Basics

4 Debugging

Use Chrome Debugging Tools

Open Chrome debugging tools after by pressing F12.

Use SourceMaps

Angular CLI adds SourceMaps to Javascript files when it sets up bundles for the browser to get an reference between JavaScript files and TypeScript files. Only available in development mode. They are not provided in production mode.

Access TypeScript files:

```
Chrome-> F12 -> Sources -> top -> webpack -> . -> src -> app
```

Here you find your TypeScript file like in your dev environment.

Use Augury

Augury is a chrome extension to debbug your Angular app. You can see your Router, Components and Models. Helps you understand and analyse your Angular app at runtime.

5 Databinding: Components & Databinding Deep Dive

Component life cycle

Event	Description
ngOnChanges	Called after a bound input property changes
ngOnInit	Called once the component is initialized
ngDoCheck	Called during every change detection run
ngAfterContentInit	Called after content (ng-content) has been projected into view
ngAfterContentChecked	Called every time the projected content has been checked
ngAfterViewInit	Called after the component's view (and child views) has been initialized
ngAfterViewChecked	Called every time the view (and child views) have been checked
ngOnDestroy	Called once the component is about to be destroyed

@Input

Decorator that marks a class field as an input property and supplies configuration metadata. The input property is bound to a DOM property in the template. During change detection, Angular automatically updates the data property with the DOM property's value.

@Output()

Decorator that marks a class field as an output property and supplies configuration metadata. The DOM property bound to the output property is automatically updated during change detection.

EventEmitter<type>

Use in components with the @Output directive to emit custom events synchronously or asynchronously, and register handlers for those events by subscribing to an instance.

```
@Output() serverCreated = new EventEmitter<{serverName: string, serverContent: string}>();
```

@ViewChild

Property decorator that configures a view query. The change detector looks for the first element or the directive matching the selector in the view DOM. If the view DOM changes, and a new child matches the selector, the property is updated.

@ContentChild

Use to get the first element or the directive matching the selector from the content DOM. If the content DOM changes, and a new child matches the selector, the property will be updated.

Content queries are set before the ngAfterContentInit callback is called.

Does not retrieve elements or directives that are in other components' templates, since a component's template is always a black box to its ancestors.

6 Databinding: Course Project - Components & Databinding

7 Directives Deep Dive

An Attribute directive changes the appearance or behavior of a DOM element.

There are three kinds of directives in Angular:

```
1. Components—directives with a template.
2. Structural directives—change the DOM layout by adding and removing DOM
elements.
    - ngif
    - ngfor
3. Attribute directives—change the appearance or behavior of an element,
component, or another directive.
    - ngclass
    - ngstyle
```

Generate a directive by CLI

> ng generate directive <directiveName>
>
> oder
>
> ng g d <directiveName>

Basic directive

Uses the ***ElementRef*** to change DOM element.

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

@Directive({
  selector: '[appBasicHighlight]'
})
export class BasicHighlightDirective implements OnInit {
  constructor(private elementRef: ElementRef) {
  }

  ngOnInit() {
    this.elementRef.nativeElement.style.backgroundColor = 'green';
  }
}
```

Better directive with renderer

Uses the ***Renderer*** to change DOM element. This works with service workers too! see https://angular.io/guide/service-worker-intro

```
import {
  Directive,
  Renderer2,
  OnInit,
  ElementRef,
  HostListener,
  HostBinding,
  Input
} from '@angular/core';

@Directive({
  selector: '[appBetterHighlight]'
})
export class BetterHighlightDirective implements OnInit {
  @Input() defaultColor: string = 'transparent';
  @Input('appBetterHighlight') highlightColor: string = 'blue';
  @HostBinding('style.backgroundColor') backgroundColor: string;

  constructor(private elRef: ElementRef, private renderer: Renderer2) { }

  ngOnInit() {
    this.backgroundColor = this.defaultColor;
    // this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'blue');
  }

  @HostListener('mouseenter') mouseover(eventData: Event) {
    // this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'blue');
    this.backgroundColor = this.highlightColor;
  }

  @HostListener('mouseleave') mouseleave(eventData: Event) {
    // this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'transparent');
    this.backgroundColor = this.defaultColor;
  }

}
```

It uses HostListener to execute code after some event occurred, like mouse or keyboard events. It changes the DOM element by using the renderer. See commented lines in the picture above.

Better directive using HostBinding

Decorator that marks a DOM property as a host-binding property and supplies configuration metadata. Angular automatically checks host property bindings during change detection, and if a binding changes it updates the host element of the directive.

```
import {
  Directive,
  Renderer2,
  OnInit,
  ElementRef,
  HostListener,
  HostBinding,
  Input
} from '@angular/core';

@Directive({
  selector: '[appBetterHighlight]'
})
export class BetterHighlightDirective implements OnInit {
  @Input() defaultColor: string = 'transparent';
  @Input('appBetterHighlight') highlightColor: string = 'blue';
  @HostBinding('style.backgroundColor') backgroundColor: string;

  constructor(private elRef: ElementRef, private renderer: Renderer2) { }

  ngOnInit() {
    this.backgroundColor = this.defaultColor;
    // this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'blue');
  }

  @HostListener('mouseenter') mouseover(eventData: Event) {
    // this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'blue');
    this.backgroundColor = this.highlightColor;
  }

  @HostListener('mouseleave') mouseleave(eventData: Event) {
    // this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'transparent');
    this.backgroundColor = this.defaultColor;
  }

}
```

This example uses HostBinding to change an attribute of a DOM element.

8 Course Project - Directives

9 Using Services & Dependency Injection

@Injectable

Decorator that marks a class as available to be provided and injected as a dependency.

10 Course Project - Services & Dependency Injection

11 Changing Pages with Routing

12 Course Project - Routing

13 Understanding Observables

Always unsubscribe a subscription!

14 Course Project Observables

15 Handling Forms in Angular Apps

16 Course Project - Forms

17 Using Pipes to Transform Output

A pipe takes in data as input and transforms it to a desired output.

see https://angular.io/guide/pipes

Pipe operator like in unix

{{ server.instanceType | uppercase }}

Chaining pipes

You can add the output from one pipe to the input to another pipe. The order is from left to right.

{{ server.started | date:'fullDate' | uppercase }}

Convert a text to uppercase

{{ server.instanceType | uppercase }}

Convert a datetime

{{ server.started | date:'fullDate' | uppercase }}

Sunday, August 8, 1920

If a pipe allows parameters it will be added an seperated by :

Build in pipes

Build in pipes are documented at angular.io website. Under documentation -> API reference and filter for pipes

Create a new pipe with CLI

```
ng generate pipe <name of pipe>
```

or

```
ng g p <name of pipe>
```

Pure pipe

```
@pipe
{
```

```
    name: nameOfPipe,
    pure: false; //default value is true

}
```

When true, the pipe is pure, meaning that the transform() method is invoked only when its input arguments change. Pipes are pure by default.

If the pipe has internal state (that is, the result depends on state other than its arguments), set pure to false. In this case, the pipe is invoked on each change-detection cycle, even if the arguments have not changed.

Create your own pipe

see folder "17 - pipes-final"

async pipe

Use async pipe on promise and oberservables.

18 Making Http Requests

Call backend by using http calls. Used to get data from a server or store data at a server.

HTTP Verbs

```
-POST
-GET
-PUT
-OPTION
Always call by the browser before a POST is called.
-DELETE
```

Create an Firebase backend

```
-go to firebase.google.com
-login with your google account
-go to console at firebase.google.com
-add a new firebase project
-go to database and create a new realtime database
-after creation you see a URL to send HTTP request to the database
```

Setup a HTTP request

```
-goto app.module.ts
-Add HttpClientModule from @angular/common/http
```

```
-go to your component and inject HttpClient from @angular/common/http
-create a post request
```

```
this.http
  .post(
    'https://ng-complete-guide-c56d3.firebaseio.com/posts.json',
    postData
  )
  .subscribe(responseData => {
      console.log(responseData);
  }
);
```

```
You need to subscribe to this request otherwise the request gets not executed.
```

Creating a get request

```
- Get data
- Set return type in get request.
- Transform data
- output our data
```

```
private fetchPosts() {
  this.isFetching = true;
  this.http
    .get<{ [key: string]: Post }>(
      'https://ng-complete-guide-c56d3.firebaseio.com/posts.json'
    )
    .pipe(
      map(responseData => {
        const postsArray: Post[] = [];
        for (const key in responseData) {
          if (responseData.hasOwnProperty(key)) {
            postsArray.push({ ...responseData[key], id: key });
          }
        }
        return postsArray;
      })
    )
    .subscribe(posts => {
      this.isFetching = false;
      this.loadedPosts = posts;
    });
}
```

```
-get:   Create a get request. Get is an generic method which can be extended by
the return type of the request.
-pipe:
    -map: Is used to transform type of responseData to postsArray.
-subscribe:
```

```
export interface Post {
  title: string;
  content: string;
  id?: string;
}
```

Handle errors

Each observable returns as a second parameter a way to access the error if available.

```
this.postsService.fetchPosts().subscribe(
  posts => {
    this.isFetching = false;
    this.loadedPosts = posts;
  },
  error => {
    this.error = error.message;
    console.log(error);
  }
);
```

Http Header and Query params

```
  let searchParams = new HttpParams();
  searchParams = searchParams.append('print', 'pretty');
  searchParams = searchParams.append('custom', 'key');
  return this.http
    .get<{ [key: string]: Post }>(
      'https://ng-complete-guide-c56d3.firebaseio.com/posts.json',
      {
        headers: new HttpHeaders({ 'Custom-Header': 'Hello' }),
        params: searchParams
      }
    )
```

Http response events and response type

```
deletePosts() {
  return this.http
    .delete('https://ng-complete-guide-c56d3.firebaseio.com/posts.json', {
      observe: 'events',
      responseType: 'text'
    })
    .pipe(
      tap(event => {
        console.log(event);
        if (event.type === HttpEventType.Sent) {
          // ...
        }
        if (event.type === HttpEventType.Response) {
          console.log(event.body);
        }
      })
    );
}
```

Event Types can be - Sent - Response - ...

Response Types can be - Text - json - ...

Http Interceptors

Intercepts and handles an HttpRequest or HttpResponse.

19 Course Project - Http

20 Authentication & Route Protection in Angular

21 Dynamic Components

22 Angular Modules & Optimizing Angular Apps

23 Deploying an Angular App

24 Bonus: Working with ngRx in our Project

25 Bonus: Angular Universal

26 Angular Animations

27 Adding Offline Capabilities with Service Workers

28 A Basic Introduction to Unit Testing in Angular Apps

29 Angular Changes & New Features

30 Course Roundup

31 Custom Project & Workflow Setup

32 Bonus: TypeScript Introduction (for Angular 2 Usage)