Angular Start

https://ide.c9.io/laczor/angular

Session_1_Basic
6. First App- Basic setup for c9.io (Session_1_Basic/first_app)
10. Adding Bootrsap(Session_1_Basic/project_template)
15. *****Lecture 15-18 Components ****
18.***Creating components from CLI *** (Much easiery than to do it manually)
20.*** Styling component****
21.*** Component Selector****
23***String Interpolation***
24. Property Binding
25. EventBinding
30x. ngIf,ngStyle,ngFor
Session_3_Project (Session_3_Project /prj-basics-final**)**
46. Creating Data Model
57. You can uses *Augury* to debug Angular (https://augury.angular.io/)
Session_5_Databinding (Session_5_Databinding/session_5_start)
75. Lifcycle hooks + ngContent
60. Communicating between components
session_6_directives-start()
87. Creating a basic directive
88. Using @HostBinding to change css properties
86. Example for *ngSwitch, *ngStyle
91. Structural Directive
(recipe-app-1-routing)
92. Creating dropdown with custom directive, by toggling class

6. First App- Basic setup for c9.io

- - in order to make it work, you should include the following line in the project's package.json file
- **("start": "ng serve --host 0.0.0.0 --port 8080 --public $C9_HOSTNAME",)**
- - "ng server" should always run, since it will look for any updates in the sourcefiles
- ###Prior **npm start** use- **nvm install 6.10.2** / nvm alias default 6.10.2.u

First.App project Setup

- - need to import "FormsModul"e at app.module.ts, so angular will be able to know that this module will be used

app.module.ts

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

import { NgModule } from '@angular/core';                                    //Used for binding

import { FormsModule } from '@angular/forms';                                //Used for forms
```

```
import { HttpModule } from '@angular/http';


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


@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],
  providers: [],
  bootstrap: [AppComponent]                                    //Imported bootrsap package
})
export class AppModule { }
```

app.component.ts

1. Name is a basic property of the AppComponent

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


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

app.component.html

1. use [(ngModel)] to two way bind the input property's "name", so it can be used as a variable in other html element

```
<input type="text" [(ngModel)]="name">                          //Two way databinding

<p>{{ name }}</p>                                              //String interpolation
```

10. Adding bootrsap

- - install npm install --save bootstrap
- - include the installed bootrsap css into the
- - no comment should be included in .angular-cli.json
- - .angular-cli.json. is a hidden file, this is where all of the depencies, rooting files are mapped, adding gloabl css files
- - include the bootrsap css file there under styles ("../node_modules/bootstrap/dist/css/bootstrap.min.css",)

.angular-cli.json

```
 "styles": [
        "../node_modules/bootstrap/dist/css/bootstrap.min.css",
        "styles.css"
      ],
```

*****Lecture 15-18 Components ****

- - src/app contains all of the component elements
- - component element name = folder name, server.component.ts = server
- - creating component typescript (importing decoration - defining a selector, giving a template with src, exporting the class)
- - adding the component to the app.module.ts so it knows the component, it's location, it's data

app.component.ts

```
import { Component } from '@angular/core';                  // It is telling angula that this is
 a component


@Component({
  selector: 'app-root',                                    //this it the html selector of the compo
nent
  templateUrl: './app.component.html',                    //Where the html is stored
```

```
  styleUrls: ['./app.component.css']                    //Where the css is stored

})

export class AppComponent {

  name = '';

}
```

index.html
-This is where the component's selector is stored

```
<body>

  <app-root></app-root>

</body>
```

app.module.ts
- Component has to be imported in the app.module

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

- **"module.ts"**

- //bundle all of the information for angular, tells it, what can it use, where it can be imported
- // you have to include the created components in the module.ts file, so angular will know what components it can use,
- // and where they are stored, referencing file, where the typescript can know what information does angular app need.
- - module.ts -imports: importing basic, prewritten modules, that can be used like bootrsap
- - register at module.ts at "ngModule"
- - Including the component id html, in a component's template html file

18.***Creating components from CLI ***

- ng generate component component's name -> ng g c "component's name"
- ng g c component's name --spec false --> will generate a component without including a test file
- ng g c /foldername/header--spec false
- - Will make the typescript component file, htmltemplate, register it at module.ts, ready to use.

20.*** Styling component****

- Css styling can be applied by using simple bootrsap, or using the component's css file
- stlyes can be manually applied, by using stlyes : array, + css commands in the app.component.ts file

```
@Component({
  selector: 'app-root',                                  //this it the html selector of the compo
nent
  templateUrl: './app.component.html',                 //Where the html is stored
 // styleUrls: ['./app.component.css']                   //Where the css is stored
 styles:[`
 h3{color:blue};
 `]
})
```

21.*** Component Selector****

```
@Component({
  selector: 'app-root',                                  //<app-root></app-root>                -
-> element style selector
  selector: '.app-root',                                 //<div class="app-root"></div>
 --> Class style selector
  selector: '[app-root]',                                 //<div app-root></div>
 --> attribute style selector
  templateUrl: './app.component.html',                 //Where the html is stored
 // styleUrls: ['./app.component.css']                   //Where the css is stored
 styles:[`
 h3{color:blue};
 `]
})
```

23***String Interpolation***

- {{}} - string interpolation Typescript --> HTML
  <!--using string intrapolation, which passes a property from typescript to html
  typescript property will be passed to the variable on {{}}-->
- string interpolation can convert number to string easily,
- {{ function()}} we can use string interpolation to call functions from typescript as well.

24. Property Binding

- [disabled] ="component's variable" --> Variable
- [disabled] ="true" -->Value
- [disabled] ="'true'" -->Passing string

25. EventBinding

- (eventname)="function's name()
- The MDN (Mozilla Developer Network) offers nice lists of all properties and events of the element you're interested in. Googling for `YOUR_ELEMENT` properties or `YOUR_ELEMENT events` should yield nice results.

30. Bindings

- (click)="eventChange()" ---> event biding
- (input)="enableButton()"---> event biding
- [(ngModel)]="name" ---> two way biding with the input
- [disabled]="!enabled" ---> 1 way property binding
-

46. Creating Data Model

recipe.model.ts
- `Recipe` is the defined data Model, which is an exported javascript class with 3 public property of type string
- When it is called with the keyword `new` **Recipe(name,desc,image)** the defined constructor will create a new object.

```
export class Recipe {
  public name: string;
  public description: string;
  public imagePath: string;

  constructor(name: string, desc: string, imagePath: string) {
    this.name = name;
    this.description = desc;
    this.imagePath = imagePath;
  }
}
```

In order to use the model you have to import it !

```
import { Recipe } from '../recipe.model';
```

57. Augury Debug.

Javascript console:

- Chrome developer tool (sources, will include all of the sources, of the webpage)

chrome sources:

- Angular Cll, gives sourcemaps in the main.bundle.js files, which can be seperated, so you can stop the code at a certain line at the code
- You can check the passed variable values at runtime + all of the ojbect properties.
- Sources - > "Webpack", you can access all of your typescript files

Augury chrome extensions:

60. Communicating between components

1.Using local references for `servernameinput`, actually we are passing the whole input element into the **onAddServer() as local reference**

within this html template.

cockpit.component.html

```
    <div class="row">
      <div class="col-xs-12">
        <p>Add new Servers or blueprints!</p>
        <label>Server Name</label>
         <input
        type="text"
        class="form-control"
        #serverNameInput>
        <label>Server Content</label>
        <!--<input type="text" class="form-control" [(ngModel)]="newServerContent">-->
        <input
        type="text"
        class="form-control"
        #serverContentInput>
        <br>
  <!--Assiging event listener to the button click, with built in component's functions-->
        <button
          class="btn btn-primary"
```

```
        (click)="onAddServer(serverNameInput)">Add Server</button>
      <button
        class="btn btn-primary"
        (click)="onAddBlueprint(serverNameInput)">Add Server Blueprint</button>
    </div>
  </div>
```

2. The modules has to be imported,
   - When the onAddServer() method is called, it will call the serverCreated Output to emit the data to the parent component

cockpit.component.ts

```
import { Component, OnInit,
 EventEmitter,                        //This is an object type which will define what data will
 be emiited to the parent component
 Output,                              //This method will pas out the emitter data
 ViewChild,                           //With this you can use the local reference
 ElementRef } from '@angular/core';   //It is for defining the type of the Viewchild


@Component({
  selector: 'app-cockpit',
  templateUrl: './cockpit.component.html',
  styleUrls: ['./cockpit.component.css']
})
export class CockpitComponent implements OnInit {
@Output() serverCreated = new EventEmitter<{serverName: string, serverContent:string}>();
@Output('bpCreated') blueprintCreated = new EventEmitter<{serverName: string, serverContent:str
ing}>();
// ElementRef means it is a reference to an element, ElementRef.nativeElement = the real DOM el
ement with all of it's properties
@ViewChild('serverContentInput') serverContentInput: ElementRef;
  constructor() { }
  ngOnInit() {
```

```
  }
  onAddServer(nameInput: HTMLInputElement) {
    this.serverCreated.emit({
      serverName: nameInput.value,
      serverContent: this.serverContentInput.nativeElement.value});
  }
  onAddBlueprint(nameInput: HTMLInputElement) {
    this.blueprintCreated.emit({
      serverName: nameInput.value,
      serverContent: this.serverContentInput.nativeElement.value});
  }
}
```

3. We can listen if a children has emitted any output, with the **@OutPut(emitter)**
app.component.html

```
<app-cockpit
  (serverCreated)="onServerAdded($event)"
  (bpCreated)="onBlueprintAdded($event)"
  ></app-cockpit>
```

4.- If yes we can use the recieved object the way it has been defined in the emitter.
app.component.ts

```
//@Output('bpCreated') blueprintCreated = new EventEmitter<{serverName: string, serverContent:s
tring}>();
onBlueprintAdded(blueprintData: {serverName: string, serverContent:string}) {
    this.serverElements.push({
      type: 'blueprint',
      name: blueprintData.serverName,
      content: blueprintData.serverContent
    });
```

75. Lifcycle hooks + ngContent

1. The app component (parent) has an array of serverelements
app.component.ts

```
export class AppComponent {

  serverElements = [{type:'server',name:'Testserver',content:'test'}];

}
```

2. With ***ngFor** we are looping through this array to create `<app-server-element>` component's.
- They have a property of **srvElement**, and we are passing the element values via propertybinding + adding contecnt to the child components
server-element.component-ts

```
 @Input('srvElement') element: {type: string, name: string, content: string};
```

app.component.html

```
<div class="container">
  <hr>
    <div class="row">
    <div class="col-xs-12">
      <app-server-element
      *ngFor="let serverElement of serverElements"
      [srvElement]="serverElement"
      [name]="serverElement.name">
        <p #contentParagraph></p>
          <p>
            <strong *ngIf="serverElement.type === 'server'" style="color: red">{{
serverElement.content }}</strong>
            <em *ngIf="serverElement.type === 'blueprint'">{{ serverElement.content }}</em>
          </p>
      </app-server-element>
    </div>
    </div>
```

```
</div>
```

server-element.component.html
- In the main component's html we are creating content, which will be deleted by default if we do not sign it with `<ng-content>` tag.

```
      <div
        class="panel panel-default">
        <div class="panel-heading" #heading>{{ name }}</div>
        <div class="panel-body">
<!--we are using ng-content directive, to tell angular, that this component will have some html
 conten
between it's opening and closing tag-->
          <ng-content></ng-content>
        </div>
      </div>
```

Server-element.component.ts
Viewencapsulation: (parent component's css will be applied to the child)

- **ViewEncapsulation.Emulated** ---> Default, will apply different selectors to each components (so a general "p" selector in the app.component.css won't be applied to the componenets)
- **ViewEncapsulation.None** ---> The general css in the app.component.css will be applied to every component (class selection won't work), since the specific component selectors are removed
- **ViewEncapsulation.Native** ---> Support shadow DOM / which is actually, not supported by every browser.

LifeCycle Hooks

- // 0. Each directive/interface has to be imported from @angular/core + included in the class implementations
- Basically you can execute different codes depending on the actual state of the component

ContentChild
`@ContentChild` Basically with this you can reference the insterted html element from the app.component.html, with the local reference of **#contentParagraph**

Thus you can reference the whole html : **this.paragraph.nativeElement**
or just the content of it **this.paragraph.nativeElement.textContent**

```
import { Component, OnInit,Input,ViewEncapsulation,OnChanges,SimpleChanges,DoCheck,AfterContent
Init,AfterContentChecked,AfterViewInit,AfterViewChecked,OnDestroy,ViewChild,ElementRef,ContentC
```

```
hild} from '@angular/core';


@Component({
  selector: 'app-server-element',
  templateUrl: './server-element.component.html',
  styleUrls: ['./server-element.component.css'],
  encapsulation: ViewEncapsulation.Emulated // None, Native
})
export class ServerElementComponent implements OnInit, OnChanges,DoCheck, AfterContentInit, Aft
erContentChecked, AfterViewInit, AfterViewChecked,
  OnDestroy{
    @Input('srvElement') element: {type: string, name: string, content: string};
    @Input() name: string;
    @ViewChild('heading') header: ElementRef;
    @ContentChild('contentParagraph') paragraph: ElementRef;
  constructor() {
     console.log('constructor called!');
  }
// 2. ngOnChanges, runs every time when the component has been changed.
// Has 1 argument: changes, with type of SimpleChanges(it has to be imported as well from @angu
lar/core)
// change argument will tell us the current value of the component, the previous value before t
he change
  ngOnChanges(changes: SimpleChanges){
     console.log('ngOnChange called!');
     console.log(changes);
  }
// 1. Runs every time a component has been created constructor/ngOnInit
  ngOnInit() {
     console.log('ngOnInit called!');
  }
// (will be triggered, almost to any changes took place in the app)
  ngDoCheck() {
```

```
    console.log('ngDoCheck called!');
  }
// (will be triggered, when the content is projected  to the <ng-content>)
  ngAfterContentInit() {
  console.log('ngAfterContentInit called!');
  console.log('Text Content of paragraph: ' + this.paragraph.nativeElement.textContent);
  }
// (will be triggered after each change detection cycle, to check if the content has been modif
ied or not)
  ngAfterContentChecked() {
    console.log('ngAfterContentChecked called!');
  }
// (when the view has been rendered to the component)
  ngAfterViewInit() {
    console.log('ngAfterViewInit called!');
    console.log('Text Content: ' + this.header.nativeElement.textContent);
  }
// (when the view has been rendered to the component and checked to any changes)
// When the element is about to be destroyed
  ngAfterViewChecked() {
    console.log('ngAfterViewChecked called!');
  }


  ngOnDestroy() {
    console.log('ngOnDestroy called!');
  }


}
```

87. Creating a basic directive

1.Creating the directive with an**[attribute]** selector, + injecting **ElementRef**

2. Not a good practice to directly change an element!
3. You can use CLI command to create a directive **--> ng g d directiveName**

basic-highlight.directive.ts

```
import  {
    Directive,          //This is the declaration of this typescript file
    ElementRef,         //To get access to the elementRef deriective we use injectoion
    OnInit
} from '@angular/core';

@Directive({
    selector:'[appBasicHighlight]'          //We are using attribute selector, so if an element
 will have this attribute the directive will be activated
})

// Lecture 84, setting up manually
export class BasicHighlightDirective  implements OnInit {
    //adding a property of elementRef to the componenet with the type of ElenemtRef
    constructor(private elementRef:ElementRef){
    }
    ngOnInit(){
        this.elementRef.nativeElement.style.backgroundColor="yellow";
        console.log('First directive created');
    }
}
```

2. Adding it to the **app.module.ts,** so every component will be able to use it.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';
import { BasicHighlightDirective } from './basic-highlight/basic-highlight.directive';
```

```
import { AppComponent } from './app.component';
import { BetterHighlightDirective } from './better-highlight/better-highlight.directive';
import { DirectiveNameDirective } from './directive-name.directive';


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

3. Using it in the **app.component.html**

```
<p appBasicHighlight >First attribute Directive</p>
```

86. Example for *ngSwitch, *ngStyle

```
        <div *appDirectiveName="onlyOdd">
            <li
            class="list-group-item"
```

```
            *ngFor= 'let number of oddNumbers'
            [ngStyle]="{backgroundColor:'yellow'}">
              <p>{{number}}</p>
            </li>
        </div>
        <div>Creating divs with ngswitch</div>
        <div [ngSwitch]="switchvalue">
          <p *ngSwitchCase="3">The switch values is 3. </p>
          <p *ngSwitchCase="5"> The switch values is 5. </p>
          <p *ngSwitchDefault> The switch value is default </p>
        </div>
```

87.Use the renderer to access the DOM

- Good practice to use rendered for manipulating the DOm
- An attribute selector directive has been created, **ElementRef** + **renderer** have been injected in the constructor
- **Hostlistener** for events has been imported as well
- **https://angular.io/api/core/Renderer2**

better-highlight.directive.ts

```
import {
  Directive,
  OnInit,
  ElementRef,                          //To access the element where the attribute selector has
 been placed
  Renderer2,                           //A method to change the style of an element
  HostListener,                        //To listen for events with the componenet
} from '@angular/core';


@Directive({
  selector: '[appBetterHighlight]'
})
export class BetterHighlightDirective {
  constructor(private elRef: ElementRef, private renderer: Renderer2) { }
```

```
  ngOnInit(){
   this.renderer.setStyle(this.elRef.nativeElement,'backgroundColor','blue');
   //Instead of using direct approach :  this.elementRef.nativeElement.style.backgroundColor="y
ellow";
  }
  @HostListener('mouseenter') mouseover(eventData: Event) {
    this.backgroundColor = this.highlightColor;
  }
  @HostListener('mouseleave') mouseleave(eventData: Event) {
    this.renderer.setStyle(this.elRef.nativeElement,'backgroundColor','yellow');
  }

}
```

app.component.html

```
 <p appBetterHighlight  >First attribute Directive</p>
```

88. Using `@HostBinding` to change css properties

app.component.html

```
 <p appBetterHighlight [defaultColor]="'white'" [highlightColor]='blue'  >First attribute Direc
tive</p>
```

better-highlight.directive.ts

- So here at the element where the attribute selector directive has been placed, we are listening for to 2 `@Input` properties, and using `@HostBinding` to access the component's css property. Basicaly similar to the upperone,, but instead of using renderer, we can use dynamic inputstyles, since we are just using the properties of the element + accessing it's css. So we do not need the **ElementRef** nor the **Renderer2**

```
import {
  Directive,
  OnInit,
  HostListener,                      //To listen for events with the componenet
```

```
  HostBinding,                    //This decorator will help us to access the component's c
ss properties
  Input                           //Listen for property bindings
} from '@angular/core';


@Directive({
  selector: '[appBetterHighlight]'
})
export class BetterHighlightDirective {
@Input() defaultColor : string = 'transparent';
@Input() highlightColor : string = 'blue';
@HostBinding('style.backgroundColor') backgroundColor: string = 'transparent';
  ngOnInit(){
    this.backgroundColor = this.defaultColor;                                    // 89 assig
nin default property
  }
  @HostListener('mouseenter') mouseover(eventData: Event) {
    this.backgroundColor = this.highlightColor;
  }
  @HostListener('mouseleave') mouseleave(eventData: Event) {
    this.backgroundColor = this.defaultColor;
  }
}
```

91. Structural Directive

- So basically we are recreating the *ngIf structural directive
- We are listening to a property input (type of boolean) called `appDirectiveName` which is actually the attribute selector of the directive.
- If the input is true, then with the **ViewContainerRef.createEmbeddedView(<ng-template>)** we are populating the html element, if the input is false, than we are clearing everything from it.
ViewContainerRef.clear()

directive-name.directive.ts

```
import { Directive,Input,
```

```
TemplateRef,                          //Basically it is determining the element where the di
rective is used, if it would be an <ng-template>
ViewContainerRef                      //Tells where to input the "template" html element
} from '@angular/core';
@Directive({
  selector: '[appDirectiveName]'
})
export class DirectiveNameDirective {
  // Lecture 91 appDirectiveName is a property, and with a setter, we can execute a function,
  // condition, is a booleand defined argument, which will come from the property binding.
  @Input() set appDirectiveName (condition:boolean){
    if(!condition){
      // will create the view for the component, with the containing html elements as templates
      this.vcRef.createEmbeddedView(this.templateRef);
    }else{
      this.vcRef.clear();
    }

  }
// What-  Templateref is an injection of the template, we will have the reference for the selec
ted template, <any> due to generic file
// and actually, this is the full html element, which holds the below html elements  where the
 directive has been assigned
// Where - ViewContainerRef - marks the place where we placed tis directive in the document.
  constructor(private templateRef: TemplateRef<any>,private vcRef: ViewContainerRef) { }

}
```

app.component.html

```
 <div *appDirectiveName="onlyOdd">
            <li
            class="list-group-item"
```

```
            *ngFor= 'let number of oddNumbers'
            [ngStyle]="{backgroundColor:'yellow'}">
              <p>{{number}}</p>
            </li>
</div>
//This is actually tha sem, ngIf is an input attribute with the type of boolean and will do the
 same as the appdirectiveName
<ng-template [ngIf]="!onlyOdd">
            <li
            class="list-group-item"
            *ngFor= 'let number of oddNumbers'
            [ngStyle]="{backgroundColor:'yellow'}">
              <p>{{number}}</p>
            </li>
</ng-template>
//Making the same with the *ngIf
<div *ngIf="onlyOdd">
            <li
            class="list-group-item"
            *ngFor= 'let number of oddNumbers'
            [ngStyle]="{backgroundColor:'yellow'}">
              <p>{{number}}</p>
            </li>
</div>
```

92. Creating dropdown with custom directive, by toggling class

- So we update the element's **class property(open)** with **Hostbinding**, listening with **HostListener** and with a boolean, attached to the value of te boolean we add or remove the class.
dropdowndirective.directive.ts

```
import {
  Directive,
```

```
  HostListener,
  HostBinding
} from '@angular/core';


@Directive({
  selector: '[appDropdowndirective]'
})
export class DropdowndirectiveDirective {
  // Using HostBinding we are accessing the class of the listened, html element where the direc
tive has been added
  // by binding it to a boolean variable, it will add/remove the class based on the variable
  @HostBinding ('class.open') isOpen = false;


  @HostListener('click') toggleOpen(){
    this.isOpen = !this.isOpen;
  }
  constructor() { }


}
```

header.component.html

```
        <li class="dropdown"
        appDropdowndirective >
```