Angular Fundamentals

© Alexander Pajer 2017

Agenda

- Components, Expressions, Templates & Directives
- String Interpolation, Property- & Event-Binding, Two-Way Binding
- Pipes & Internationalization
- Understanding Component Lifecycle
- Custom Directives & Pipes
- Communicating with Nested Components
- Reusable Components using Local References & Content Projection
- Organizing Angular Artifacts using Modules and Barrels

Components, Templates & Directives

Modules

- Every Angular app has at least one NgModule class, the root module named AppModule
- Can have more Modules – eg Features of an Application
- Modules contain:
  - Components
  - Templates
  - Services
  - Directives
  - …

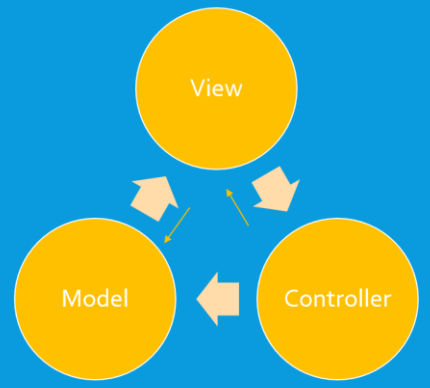
Dependency Injection

- Dependency Injection (DI) is a software design pattern that deals with the matter of providing Object Instances
- Angular provides a Dependency Injection mechanism following core components which can be injected into each other as dependencies instead of "newing" them up
- In TypeScript this is done using the constructor of a class

```
@Component({
selector: 'app-vouchers',
templateUrl: './vouchers.component.html',
styleUrls: ['./vouchers.component.css']
})
export class VouchersComponent implements OnInit {
vouchers: Voucher[];
constructor(private router: Router, private vs: VouchersService) {
}
```

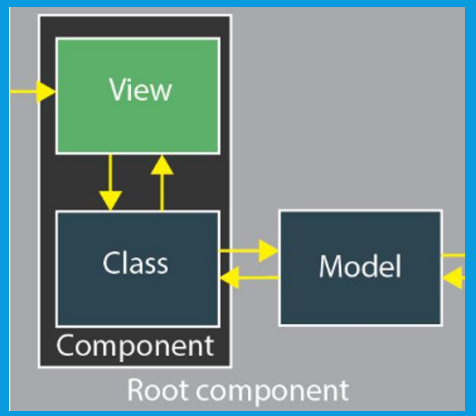
Components

- An Angular App consists of a set of one or more [nested] component
- @Input | @Output are used to exchange data
- It defines:
  - A selector
  - View: HTML | Inline
  - Directives
  - CSS
  - …



home.component.ts

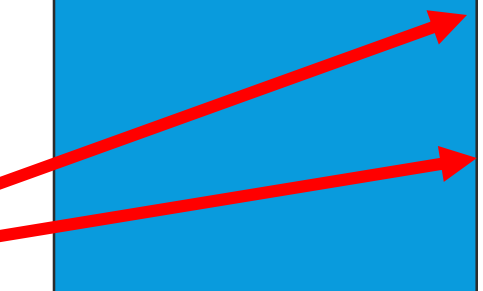
```ts
import {Component} from '@angular/core';

@Component({
  selector: 'home',
  styleUrls: ['./home.component.css'],
  templateUrl: './home.component.html'
})
export class HomeComponent {
}
```



Component Basics

- You define a component's application logic inside a class.
- The class interacts with the view through an API of properties and methods
- Metadata is provided using Decorators
- Components, like other artifacts have to be registered in app.module.ts

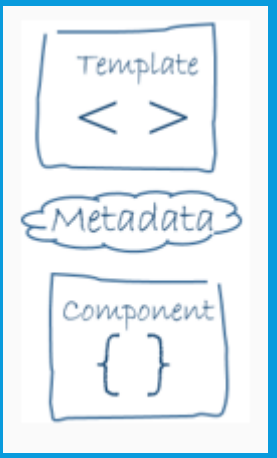
```
import { Component } from '@angular/core';

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



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

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

Decorators / Metadata

- All Angular Artifacts must provide Metadata related to the type of artifact
- For a complete list filter for the component in the API documentation @ https://angular.io/api
- Components have a @Component decorator and the following Metadata:
  - selector
  - template | templateUrl
  - providers
  - …



Template

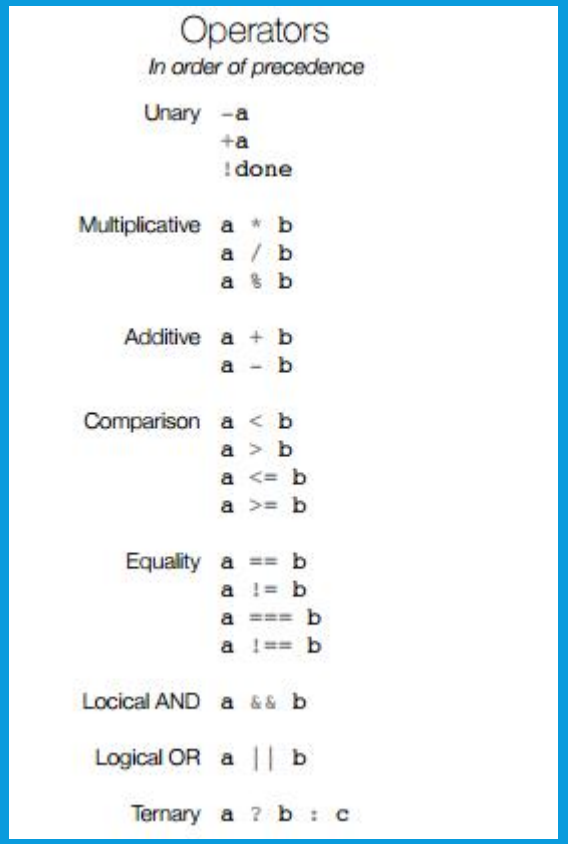
- The component's view can be defined by using the template which tells Angular how to display the component.
- The template describes how the component is rendered on the page.
- Templates do NOT include the <html> tag

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

Expressions

- Expressions are JavaScript-like code snippets that are usually placed in bindings such as {{ expression }}
- Expressions are Typically used in Templates
- Expressions cheat sheet at: http://teropa.info/images/angular_expressions_cheatsheet.pdf

```
<div>
     1+2={{1+2}}
</div>
```



Directives

- Angular directives are used to extend / manipulate DOM
- There are three kinds of directives in Angular:
  - Components- Directives with a template.
  - Attribute Directives - change the appearance or behavior of an element, component, or another directive.
  - Structural Directives - change the DOM layout by adding and removing DOM elements.

Attribute directives

- Change the appearance or behavior of a DOM element
- Other than in Angular JS there are NOT so many built-in Directives in Angular
- Most things can be archieved using property binding
  - Look & Feel: ngStyle, ngClass
  - Databinding: ngModel
  - Forms: ngForm, MinLenghtValidator, ..Validator
  - Routing: RouterLink, RouterOutlet

Structural Directives

- Structural directives shape or reshape the DOM's structure
- Easy to recognize - an asterisk (*) precedes the directive attribute name
  - *ngIf,
  - *ngFor,
  - ngSwitch

ID	Text	
2	BP Tankstelle	Edit
3	Amazon	Edit
5	Inserted by WebApi	Edit
1	Demo AC	Edit

```
<table>
      <thead>
      <tr>
            <th>ID</th>
            <th>Text</th>
            <th></th>
      </tr>
      </thead>
      <tbody>
      <tr *ngFor="let v of vouchers" (click)="showVoucher(v.ID)">
            <td>{{v.ID}}</td>
            <td>{{v.Text}}</td>
            <td><a [routerLink]="[ '/vouchers', v.ID ]">Edit</a></td>
      </tr>
      </tbody>
</table>
```

Databinding

Service

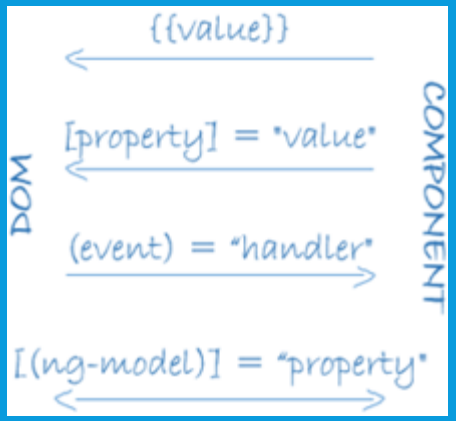
- A service is responsible for data operations
- It typically utilizes the builtin http client
- Can use Promises or Observable pattern

```
@Injectable()
export class VouchersService {
constructor(private http: HttpClient) { }

getVouchers() : Promise<any> {
return this.http.get('/assets/vouchers.json').toPromise();
}
}
```

```
export class VouchersComponent implements OnInit {

vouchers: Voucher[];

constructor(private router: Router,
            private vs: VouchersService) {}

ngOnInit() {
this.vs.getVouchers().then(data => this.vouchers = data)
}
```

Data Binding

- Angular provides several databinding patterns
  - String Interpolation
    - <h1>{{ title }}</h1>
  - Property binding
    -
  - Event binding
    - <button (click)="onClick(param)">
  - Two-way binding
    - <input type="text" [(ngModel)]="firstName">

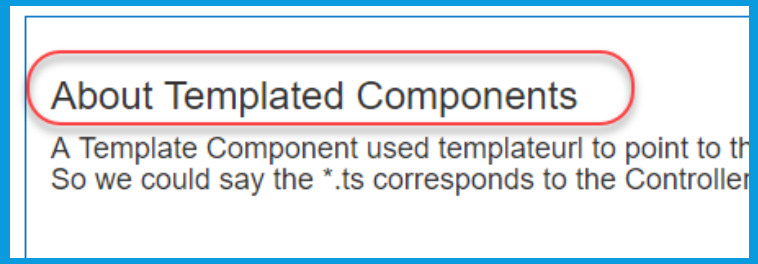


String Interpolation

- Just like Expressions it uses double curly brackets{{ variable }}
- „variable“ must be defined in your.component.ts as prop

```
export class TemplateComponent implements OnInit {

title: string = "About Templated Components";
```

```
<h3>{{title}}</h3>
```



Property Binding

- Properties of the vm can be bound to any HTML attribute
- Just warp the HTML attribute in square brackets [prop]
- Replace most of Angular 1.x Directives like ng-hide -> [hidden]="hiddenprop"

```
<div>
            <img [src]="person.imgUrl" />
</div>
```

Common DOM Events

- Mouse events:
  - onclick, onmousedown, onmouseup
  - onmouseover, onmouseout, onmousemove
- Key events:
  - onkeypress, onkeydown, onkeyup
  - Only for input fields
- Interface events:
  - onblur, onfocus
  - onscroll
- Form events
  - onchange – for input fields
  - onsubmit
    - Allows you to cancel a form submission
    - Useful for form validation

Event Binding

- Used to bind to DOM events without the "on": onlick -> (click)
- Can pass parameters

```
<h4>Show and hide</h4>
<a (click)="toggleDisplay()">Toggle</a>
<button (click)="toggleDisplay()">Toggle</button>

<div class="wrapper">
<span [style.visibility]="hide ? 'hidden': 'visible'">Msg 1</span>
<span [style.visibility]="!hide ? 'hidden': 'visible'">Msg 2</span>
</div>
```

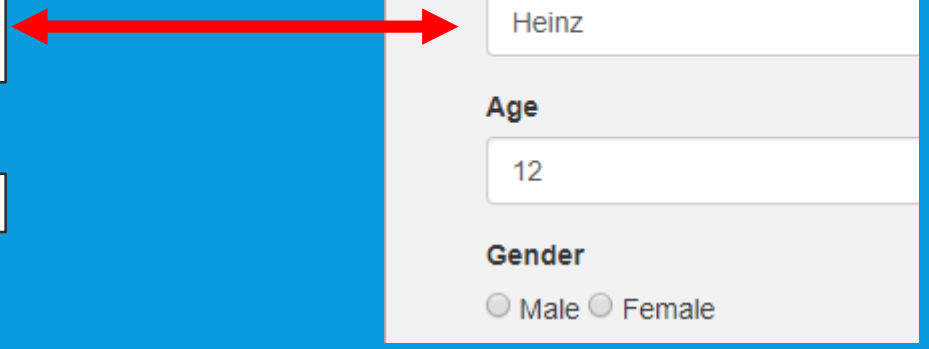
```
toggleDisplay(){
this.hide = !this.hide;
}
```

Two Way Binding

- Archieved using [(ngModel)]='prop'
- Requires FormsModule from '@angular/forms'

```
export class BindingComponent implements OnInit {

person = {id: 1, name: "Heinz", age: 12, imgUrl: ""};
```

```
<input type="text" [(ngModel)]="person.name">
```

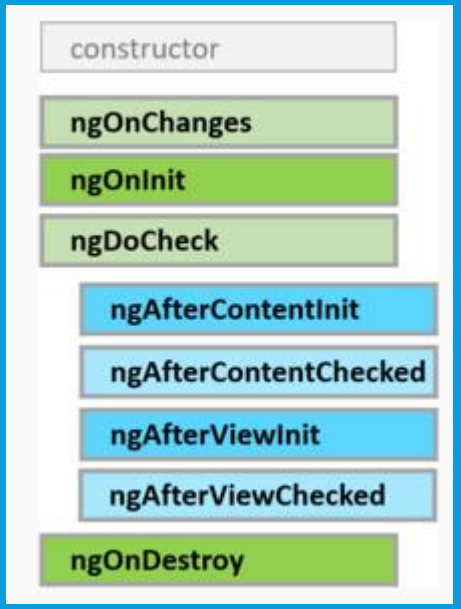


Component Lifecycle

Lifecycle Hooks

- A component has a lifecycle managed by Angular
- 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.
- Hooks are definedusing Interfaces

```
export class VouchersComponent implements OnInit {

  vouchers: Voucher[];

  constructor(private router: Router, private vs: VouchersService) {

  }

  ngOnInit() {
    this.vs.getVouchers().then(data => this.vouchers = data)
  }

  showVoucher(id: number){
    this.router.navigate(['/voucher/' + id]);
  }
}
```

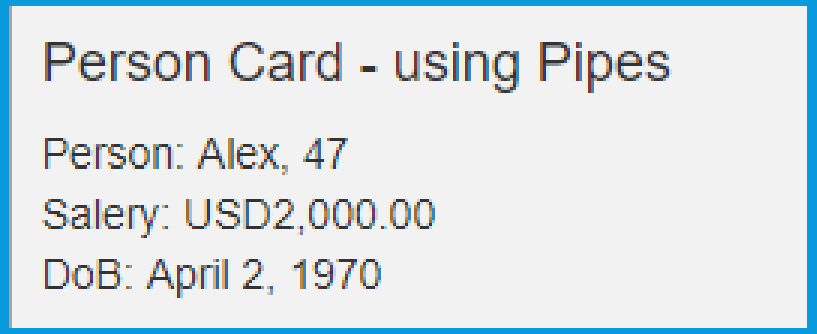


Pipes & Directives

Pipes

- Tranform displayed values with a template – like Formating expressions (currency, date)
- Lots of builtin pipes – custom pipes possible - Support chaining
  - CurrencyPipe, DatePipe, DecimalPipe
  - UpperCasePipe, LowerCasePipe, TitleCasePipe
  - JsonPipe, AsyncPipe, SlicePipe
  - …

```
Salery: {{person.salery | currency}}<br/>
DoB: {{person.dateOfBirth | date:'longDate'}}<br/>
```



Localization

- For i18n add {provide: LOCALE_ID, useValue: "de-DE"} to Providers section of module
- For pipes localization import
  - registerLocaleData
  - register appropriate locale

```
import { registerLocaleData } from '@angular/common';
import localeDe from '@angular/common/locales/de';

registerLocaleData(localeDe)
```

```
providers: [
  VouchersService,
  FirebaseService,
  {provide: LOCALE_ID, useValue: "de-DE"},
  RouteGuard
],
```

Custom Pipes

- Used to implement custom piping functionality ie filtering, custom formatting
- Requires import { Pipe, PipeTransform } from '@angular/core';
- Must implement transform() defined in PipeTransform

```
@Pipe({
name: 'VoucherFilter'
})
export class VoucherFilterPipe implements PipeTransform {

transform(items: Voucher[], filter: string, field: string): Voucher[] {
if(!items || !filter || !field){
return items;
}
return items.filter(item=>item[field].toLowerCase().includes(filter.toLowerCase()))
}
}
```

Custom Directive

- ElementRef provides direct access to DOM
- nativeElement – represents the DOM element that was originally touched
- @Input ATTRIBUTENAME used to attach attribure to DOM-element

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

@Directive({selector: '[highlight]'})
export class HighlightDirective{

@Input() highlight : string;

constructor(el: ElementRef){
el.nativeElement.style.color = '#0072C6';
el.nativeElement.style.fontSize = '20px';
}
}
```

Directives with events

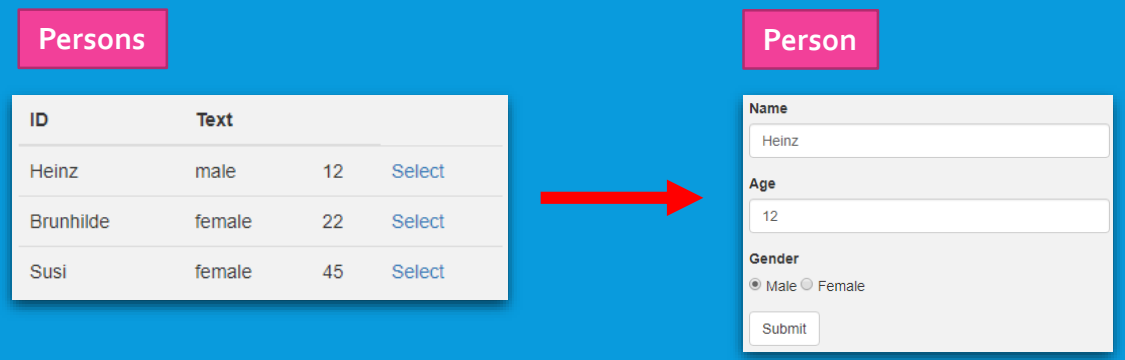
- Directives can respond to DOM Events
- @HostListener decorator used to handle Event within Directive

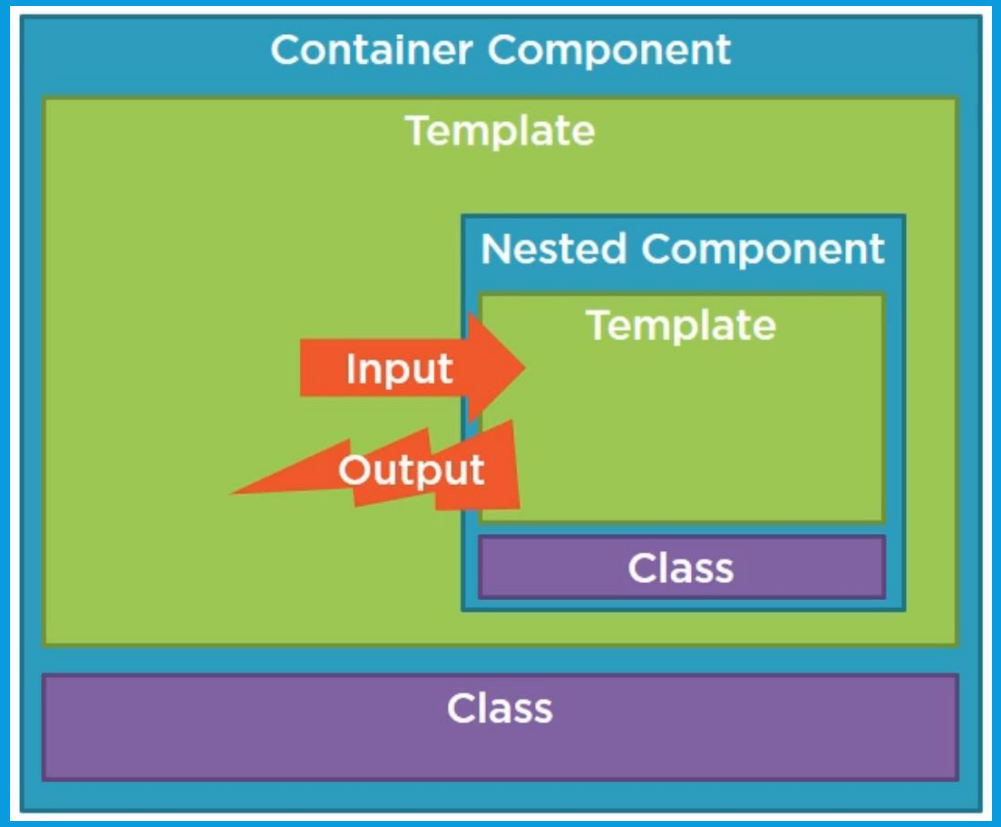
```
@HostListener('mouseenter') onMouseEnter() {
  this.hover(true);
}

@HostListener('mouseleave') onMouseLeave() {
  this.hover(false);
}

hover(underline: boolean){
if(underline){
   this.renderer.setElementStyle(this.el.nativeElement, 'text-decoration', 'underline');
} else {
   this.renderer.setElementStyle(this.el.nativeElement, 'text-decoration', 'none');
}
}
```

Communicating with Nested Components

Nested Components

- Reduces compexity by having smaller chunks
- Enhances re-usibility
- Can be used for Parent – Child relations



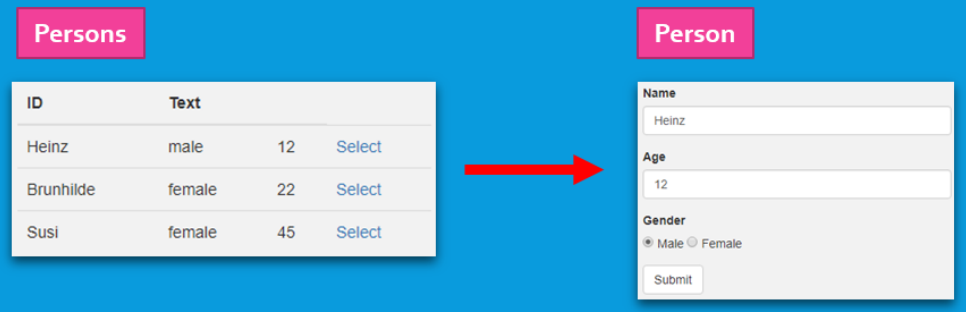


@Input

- Defines an Input from Parent to Child Component
- Decortated using @Input, many @Input possible

```
<div>
<h4>Details</h4>
<app-person [person]="current"></app-person>
</div>
```

```
export class PersonComponent implements OnInit {
@Input() person: Person;
```



@Output

- Defines an Output from Child Component to Parent
- Uses Event Emitter with ONE param, many Events possible

```
export class PersonComponent implements OnInit {
@Input() person: Person;
@Input() edit: boolean;
@Output() savePerson : EventEmitter<Person> = new EventEmitter()

doSave(){
  this.savePerson.emit(this.person)
}
```

```
<app-person (savePerson)="sendtoService($event)">...
```

```
export class PersonsComponent implements OnInit {
...

sendtoService(p:Person){
console.log("saving to service");
console.log(p);
}
```

Local Reference

- Local Reference is a reference to a Child Component using # sign
- Enables alternative communication between parent / child
- Works for:
  - Props
  - Methods

Persons

```
<li *ngFor="let e of employees">
<app-person [person]="e" [edit]="false" #personRef ></app-person>
<a (click)='personRef.doDelete()'>Delete Person</a>
<hr>
</li>
```

Person

```
export class PersonComponent implements OnInit {
@Input() person: Person;
@Input() edit: boolean;
@Output() savePerson : EventEmitter<Person> = new EventEmitter()

doDelete(){
console.log(`deleting ${this.person.name}`);
}
```

Content Projection

- Allows to build re-usable Components
- Content entered in the parent is projected to the Child Component using ng-content
- Use ng-container in parent to straighten resulting html

```
<app-employee>
<div class="heading">Employee {{person.name}}</div>
<div class="body">{{person.name}} is {{person.age}}</div>
</app-employee>
```

```
<div class="panel panel-default">
<div class="pannel-heading">
<ng-content select=".heading"></ng-content>
</div>
<div class="pannel-body">
<ng-content select=".body"></ng-content>
</div>
</div>
```

Organizing Angular Artifacts using Barrels

{barrels}

"A barrel is a way to rollup exports from several modules into a single convenience module.

The barrel itself is a module file that re-exports selected exports of other modules."

Practically a barrel ist just a folder with an index.ts exporting all artifacts

Barrels are used to reduce the number of import statements for

- Models
- Components
- Services
- Directives
- ...