HTML/CSS

Codacademy course April/Mai 2018

Freitag, 30. März 2018
08:45

Display / Flexbox

The **display** property allows you control how an element flows vertically and horizontally a document.
- inline elements take up as little space as possible, and they cannot have manually-adjusted width or height.
- block elements take up the width of their container and can have manually-adjusted heights.
- inline-block elements can have set width and height, but they can also appear next to each other and do not take up their entire container width.
- The ***float*** property can move elements as far left or as far right as possible on a web page
- ***display: flex***; => aktiviert flexbox features für die enthaltenen Items (Default: inline horizontal, vertikat mit stretching etc.)

width: 100% => use entire space of container

container zentriert mit Rand links / rechts:
```
container {
        max-width: 960px;
        margin: auto;       => geht nur wenn width gesetzt ist
}
```
vergleichbar mit **flexbox, horizontal zentriert)**:
div mit: *max-width: ….px; margin: 0 auto;*

border: 1px solid red; => zeigt Umriss des DIV an

flex-grow: 1; => use space in container if available (larger number takes more space) => gut für Menüs/Header

Alignment:
- in x-Richtung (standard "cross axis") => ***justify-content*** (flex-start, center, flex-end, space-around, space-between)
- in y-Richtung (standard "main axis") => ***align-items*** (flex-start, center, flex-end)
- Wenn mit "**flex-direction**: column" andere cross-axis => Bedeutung der beiden properties wird vertauscht

flex-wrap: wrap => statt Skalierung wird falls zu wenig Platz für die flex items eine neue Zeile begonnen.
Für die Anordnung mehrerer Zeilen dann ***align-content*** verwenden (values wie bei justify-content)

flex-flow: shortcut zum Setzen von flex-direction und flex-wrap

IMG / Skalierung

1) via img html-tag => use if image belongs to main content of page
- scaling: width: xx% (oder px oder rem) => skaliert IMG ohne Verzerrung, auf x% des Containers
- Achtung falls IMG in flexbox, da diese standardmässig die flex items vertikal skaliert!! (Abschalten mit z.B. align-items = center oder height: … setzen)
- nur width ODER height setzen, ansonsten Verzerrung

Scale media/img... with container:
```
.container img {
        max-width: 100%;
        height: auto;
        display: block;
}
```

2) via background image of div => use for backgrounds, with text on top etc.
```
imgcontainer: { background-image: url("./xxx/xxx.jpg");  }
```
- background-size: cover; => anwenden auf div, skaliert eine Seite auf Container-Größe, ohne stretchen
  oder: contain; => so groß wie möglich, aber ohne zu stretchen
  oder: fill => füllt div aus ohne Skalierung falls Bild groß genug ist
- background-repeat: no-repeat;
- background-position: center;

Media Queries:

```
@media only screen and (max-width: 480px) {
        body {
        font-size: 12px;
        }
}
```

Typische Screen Sizes
- für Desktop: min-width: 1600px;
- für Tablets: max-width: **768px**; (portrait) bzw. max-width: 1024px; (Landscape)
- für Mobile: max-width: **480px**;

Colors

- 141 feste Farben
- RGB: color: rgb(x, y, z); bzw. rgba(x,y,z,a) mit a = alpha/opacity 0…1
- HSL: color: hsl(h, s, l); mit hue 0…360 auf color wheel, saturation 0…100%, lightness 0..100% (hsla für alpha..)
- Website für color Auswahl auf Site: http://paletton.com

Typography

- font-weight: bold; oder Zahl 100…900 (default = 400, bold=700, light=300)

- font-style: italic;
- word-spacing: 0.3em; (default=0.25em)
- letter-spacing: 0.3em; ("kerning")
- text-transform: uppercase;
- text-align: right; (left, center)
- Line-height:

- if unitless number: means line height is a ratio of font size, e.g. 1.2 (or use px, em, rem, percent)
- non-user fonts: fonts from centralized directories, e.g. fonts.google.com
- embed via link offered by Google font page
- or embed directly: enter link in browser => css rules for font. Copy "latin" section to very top of style.css (@font-face…)

```
@font-face {
  font-family: "Roboto";
  src: url(fonts/Roboto.woff2) format('woff2'),
       url(fonts/Roboto.woff) format('woff'),
       url(fonts/Roboto.tff) format('truetype');
}
```

- https://www.fontsquirrel.com/
- https://fonts.google.com
- https://www.dafont.com/de/
- https://www.typewolf.com/

Pseudo-Classes

Animations

Icons

- via Fontawsome:
  - https://fontawesome.com/
  - https://www.bootstrapcdn.com/fontawesome/

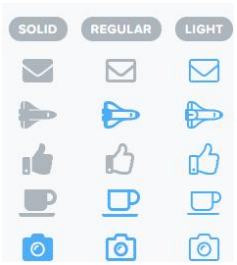


Image Editing

- Make favicon using http://favicon-generator.org
- To add the favicon to your web page: <link rel="icon" href="./<favicon-name>.ico" type="image/x-icon">
- Editing Images (crop, resize…): https://pixlr.com/editor/ (requires flash)
- Convert to SVG: https://image.online-convert.com/convert-to-svg
- use jpg for high-detail pictures, use svg for low-detail, e.g. icons (scales well)

Accessibility

- use semantic html tags where possible (e.g. header, nav, footer)
- ….

VS Code keyboard shortcuts:

- ALT + arrow => **move** current line
- Shift ALT + arrow => **clone** current line
- Shift CTRL K => **delete** current line
- CRTL # => comment in/out

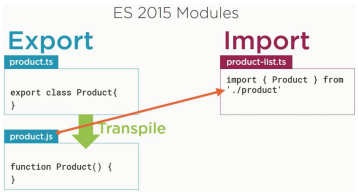
Angular (Getting Started, D. Kurata, 6hrs.)

Samstag, 23. Dezember 2017
21:13

WEB SUPPORT: google mdn ..... => for seach within Mozilla Dev Network

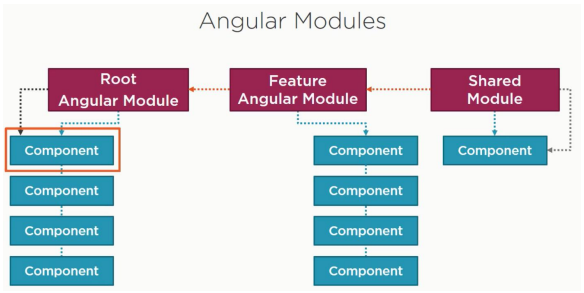
PascalCasing (for e.g. clases) vs. **camelCasing** (for e.g. variables)

Angular Documentation
- Angular.io



Modules (ES 2015)
- create:
  export class Product {
  }
- use: import { Product } from
  './product'

Project setup:
- make sure required dependencies are defined in package.json
- run npm install from project directory
- will installe node-modules (add to git excludes)

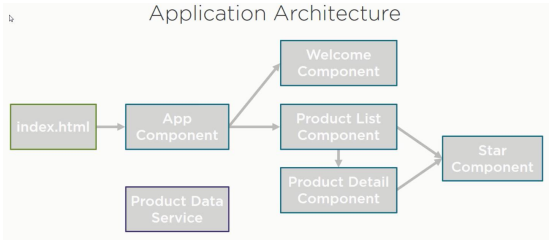


TODO's:
- learn about Twitter bootstrap
- learn about promises and observables
- write backend service to retriebve json data from file
- deploy service to webserver
- learn css

Component decorator is js function with @prefix:

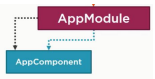
app.component.ts
```
import { Component } from '@angular/core';

@Component({
    selector: 'pm-root',
    template: `
    <div><h1>{{pageTitle}}</h1>
        <div>My First Component</div>
    </div>
    `
})
export class AppComponent {
 pageTitle: string = 'Acme Product Management';
}
```



app.module.ts
```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent }  from './app.component';

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



To use your component within your app:
- (A) create component and 1) export YourComponent 2) add a selector in the component metadata
- (B) in app.module.ts: declare YourComponent and import YourComponent
- (C) in app.component.html (template): reference your selector

(A) product-list.component.ts
```
import {Component} from '@angular/core';
@Component({
selector: 'pm-products',
templateUrl: './product-
list.component.html'
})
export class ProductListComponent {
pageTitle: string = 'Product List';
}
```

(C) app.component.html
```
<div style="text-align:center">
<h1>
Welcome to {{title}}!!
</h1>
<pm-products></pm-products>
</div>
```

(B) app.module.ts
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { ProductListComponent } from './products/product-list.component';
@NgModule({
declarations: [
AppComponent,
ProductListComponent
],
imports: [
BrowserModule
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

Data Binding

a) Interpolation: {{ class-property-name }} => one-way binding from class to template
can also be a template expression (means method call, calculation, etc.)
```
<img src={{product.imageUrl}}>
```

b) property binding

c) Event binding
```
<button (click)='toggleImage()'>
```

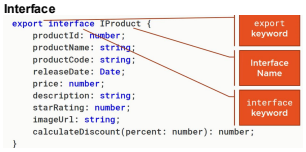
d) 2-way binding
```
<input [(ngModel)]='listFilter'>
```
to use: this directive: in app.module.ts:
```
import { FormsModule } from '@angular/forms';
...
imports: [
BrowserModule,
FormsModule
],
```

Structural Directives
- ***ngIf**
  `<table class="table" *ngIf = 'products.length'>`
  => add or remove element from DOM
- ***ngFor**

- **ngModel** => see above

Transform data with pipes:
- example:
  {{ product.productCode | lowercase }}
- also available: date, currency, uppercase, ...

Interface



Class
```
export class Product implements IProduct {

    constructor(public productId: number,
                public productName: string,
                public productCode: string,
                public releaseDate: string,
                public price: number,
                public description: string,
                public starRating: number,
                public imageUrl: string) {
    }
```

Encapsulate styles in components:

styles
```
@Component({
    selector: 'pm-products',
    templateUrl: './product-list.component.html',
    styles: ['thead {color: #337AB7;}']})
```

styleUrls
```
@Component({
    selector: 'pm-products',
    templateUrl: './product-list.component.html',
    styleUrls: ['./product-list.component.css']})
```

Lifecycle hooks:
- onInit
- onChanges
- onDestroy

to use:
```
import { Component, OnInit } from '@angular/core';
export class ProductListComponent
                implements OnInit {
  pageTitle: string = 'Product List';
  showImage: boolean = false;
  listFilter: string = 'cart';
  products: IProduct[] = [...];

  ngOnInit(): void {
        console.log('In OnInit');
  }
}
```

Getter / Setter for property:
(if logic needs to be executed at value change)
```
_listFilter: string;
get listFilter(): string {
    return this._listFilter;
}
set listFilter(value:string) {
    this._listFilter = value;
}
```

Create input properties:
use **@Input()** decorator in the nested component

star.component.ts
```
@Component({
  selector: 'pm-star',
  templateURL: './star.component.html'
})
export class StarComponent {
  @Input() rating: number;
  starWidth: number;
}
```

Create output => event:

product-list.component.ts
```
@Component({
  selector: 'pm-products',
  templateURL: './product-list.component.html'
})
export class ProductListComponent {
  onNotify(message: string): void { }
}
```

star.component.ts
```
@Component({
  selector: 'pm-star',
  templateURL: './star.component.html'
})
export class StarComponent {
 @Input() rating: number;
 starWidth: number;
 @Output() notify: EventEmitter<string> =
                new EventEmitter<string>();
```

```
ngOnInit(): void {
    console.log('In OnInit');
}
```

ES2015 backtick notation:

```
lick(): void {
 console.log(`The rating ${this.rating} was clicked!`)
```

access element property value

```
@Component({
  selector: 'pm-star',
  templateURL: './star.component.html'
})
export class StarComponent {
  @Input() rating: number;
  starWidth: number;
}
```

pass input at parameter with the directive:

product-list.component.html

```
<td>
  <pm-star [rating]='product.starRating'>
  </pm-star>
</td>
```

```
@Component({
  selector: 'pm-products',
  templateURL: './product-list.component.html'
})
export class ProductListComponent {
  onNotify(message: string): void { }
}
```

product-list.component.html

```
<td>
  <pm-star [rating]='product.starRating'
           (notify)='onNotify($event)'>
  </pm-star>
</td>
```

```
@Component({
  selector: 'pm-star',
  templateURL: './star.component.html'
})
export class StarComponent {
  @Input() rating: number;
  starWidth: number;
  @Output() notify: EventEmitter<string> =
             new EventEmitter<string>();

  onClick() {
    this.notify.emit('clicked!');
  }
}
```

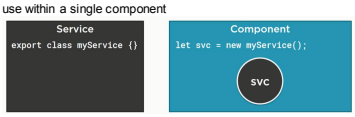
star.component.html

```
<div (click)='onClick()'>
  ... stars ...
</div>
```

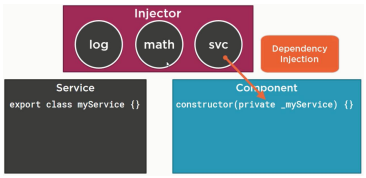
$event = payload of the event

Angluar Services

use within a single component



OR: register with Angular injector, creates a singleton (service instance) for the service:



Dependency Injection

A coding pattern in which a class receives the instances of objects it needs (called dependencies) from an external source rather than creating them itself.

Nested component:

Input decorator
- Attached to a property of any type
- Prefix with @; Suffix with ()

Output decorator
- Attached to a property declared as an EventEmitter
- Use the generic argument to define the event payload type
- Use the new keyword to create an instance of the EventEmitter
- Prefix with @; Suffix with ()

Container:

Use the directive
- Directive name -> nested component's selector

Use property binding to pass data to the nested component

Use event binding to respond to events from the nested component
- Use $event to access the event payload passed from the nested component

create service:

product.service.ts

```
@Injectable()
export class ProductService {

  getProducts(): IProduct[] {
  }

}
```

Service class
- Clear name
- Use PascalCasing
- Append "Service" to the name
- export keyword

Service decorator
- Use Injectable
- Prefix with @; Suffix with ()

Subscribe to observable:

```
x.then(valueFn, errorFn)                    //Promise
x.subscribe(valueFn, errorFn)               //Observable
x.subscribe(valueFn, errorFn, completeFn)   //Observable
let sub = x.subscribe(valueFn, errorFn, completeFn)
```

to use it:
1) **register service** => import it + add to providers
(if done in app.component.ts => available in all app components

```
import { Component } from '@angular/core';
import { ProductService } from './products/product.service';

@Component({
  selector: 'pm-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  providers: [ProductService]
})
export class AppComponent {
  title = 'Angular: Getting Started';
}
```

2) **inject to component**: within the constructor:

```
constructor(private _productService: ProductService) {
}
```

access service e.g. via _productService.getProducts();

Use angular http service:

product-list.component.ts

```
  }

  toggleImage(): void {
    this.showImage = !this.showImage;
  }

  ngOnInit(): void {
    this._productService.getProducts()
        .subscribe(products => {
            this.products = products;
            this.filteredProducts = this.products;
          },
          error => this.errorMessage = <any>error);
  }
```

product.service.ts

```
@Injectable()
export class ProductService {
    private _productUrl = './api/products/products.json';

    constructor(private _http: HttpClient) { }

    getProducts(): Observable<IProduct[]> {
        return this._http.get<IProduct[]>(this._productUrl
            .do(data => console.log('All: ' + JSON.stringi
            .catch(this.handleError);
    }

    private handleError(err: HttpErrorResponse) {
        console.log(err.message);
        return Observable.throw(err.message);
    }
}
```

Pluralsight Courses

- "Angular: Reactive Forms"
  - Http and CRUD
- "Play by Play: Angular 2/RxJS/HTTP and RESTful Services with John Papa and Dan Wahlin"
  - RxJS and Observables

Create component via Angular CLI:

```
ng g c products/product-detail.component --flat
```

g - generate
c - component
--flat no extra folder for component

Don't display "undefined" elements: use "safe navigation operator" (**?**) or use ***ngif**:

```
<div class='panel panel-primary' *ngIf='product'>
    <div class='panel-heading'>
        {{pageTitle + ': ' + product?.productName}}
    </div>
</div>
```

Routing
- Angular apps are SPA's (single page applications)
- the single page is defined in index.html
- view change is done via route activation:

Configure a route for each component

Define options/actions

Tie a route to each option/action

Activate the route based on user action

Activating a route displays the component's view

routerLink is used to tie route to an action

Home | Product List — `<a routerLink="/products">Product List</a>`

`{ path: 'products', component: ProductListComponent }`

- Anglular looks up path for the selected route
- displays component at location defined by **router-outlet**
- import **RouterModule** from @angular/router and call .forRoot([]) to register routes
  => @ngModule: RouterModule.forRoot([])

NOTE:
- webserver must support HTML5 file style urls => ***must support URL rewriting***
- alternative would be # style routing (www.mywebsite.com/#/products)

www.myWebService.com/products/5

```
{ path: 'products', component: ProductListComponent },
{ path: 'products/:id', component: ProductDetailComponent },
```

first example: simple mapping
second example: pass a parameter to the component

Specify **default route**:

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

if no match at all: send to error page

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

order is important: first match wins!

TODO's:

- set base tag in index.html header: <base href="/">

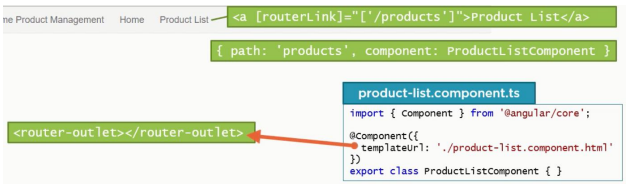
```
index.html
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="ut
5    <title>Acme Prod
6    <base href="/">
```

- link navigation element to routes:

```
<ul class='nav navbar-nav'>
  <li><a [routerLink]="['/welcome']">Home</a></li>
  <li><a [routerLink]="['/products']">Product List</a></li>
</ul>

<router-outlet></router-outlet>
```

router outlet is where the views will be displayed



Nesting vs. Routing:

Nest-able components
- Define a selector
- Nest in another component
- No route

Routed components
- No selector
- Configure routes
- Tie routes to actions

Routing Checklist:

Define the base element

Add RouterModule
- Add each route (RouterModule.forRoot)
- Order matters

path: Url segment for the route
- No leading slash
- '' for default route
- '**' for wildcard route

component
- Not string name; not enclosed in quotes

Angular (Routing from Getting Started, D. Kurata, 6hrs.)

Samstag, 23. Dezember 2017
21:13

Routing

- Angular apps are SPA's (single page applications)
- the single page is defined in index.html
- view change is done via route activation:

Configure a route for each component

Define options/actions

Tie a route to each option/action

Activate the route based on user action

Activating a route displays the component's view

routerLink is used to tie route to an action

- Anglular looks up path for the selected route
- displays component at location defined by **router-outlet**
- import **RouterModule** from @angular/router and call .forRoot([]) to register routes
  => @ngModule: RouterModule.forRoot([])

NOTE:

- webserver must support HTML5 file style urls => ***must support URL rewriting***
- alternative would be # style routing (www.mywebsite.com/#/products)

first example: simple mapping
second example: pass a parameter to the component

Specify **default route**:

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

if no match at all: send to error page

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

order is important: first match wins!

TODO's:

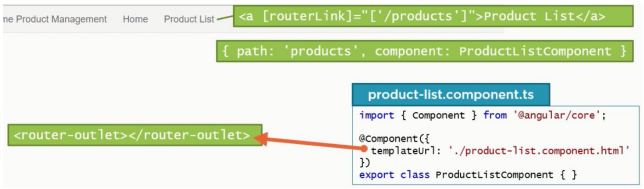
- set base tag in index.html header: <base href="/">

- link navigation element to routes:

```
<ul class='nav navbar-nav'>
  <li><a [routerLink]="['/welcome']">Home</a></li>
  <li><a [routerLink]="['/products']">Product List</a></li>
</ul>
```

```
<router-outlet></router-outlet>
```

router outlet is where the views will be displayed



Nesting vs. Routing:

Nest-able components
- Define a selector
- Nest in another component
- No route

Routed components
- No selector
- Configure routes
- Tie routes to actions

Routing Checklist:

Define the base element

Add RouterModule
- Add each route (RouterModule.forRoot)
- Order matters

path: Url segment for the route
- No leading slash
- '' for default route
- '**' for wildcard route

component
- Not string name; not enclosed in quotes

Tie route to clickable element:

Add the RouterLink directive as an attribute
- Clickable element
- Enclose in square brackets

Bind to a link parameters array
- First element is the path
- All other elements are route parameters

Passing Parameters:

product-list.component.html

```
<td>
  <a [routerLink]="['/products', product.productId]">
    {{product.productName}}
  </a>
</td>
```

app.module.ts

```
{ path: 'products/:id', component: ProductDetailComponent }
```

Reading / receiving parameters:

product-detail.component.ts

```
import { ActivatedRoute } from '@angular/router';

    constructor(private _route: ActivatedRoute) {
       console.log(this._route.snapshot.paramMap.get('id'));
    }
```

app.module.ts

```
{ path: 'products/:id', component: ProductDetailComponent }
```

Protection routes with guards:

CanActivate
- Guard navigation to a route

CanDeactivate
- Guard navigation from a route

Resolve
- Pre-fetch data before activating a route

CanLoad
- Prevent asynchronous routing

NODE

Samstag, 23. Dezember 2017
18:48

node / npm
- version : node -v
- installation of required modules: npm install <modulename>

Proxy settings:
- for node: siehe Confluence. Gespeichert in C:\Users\rippling\.npmrc
- for shell at Siemens:
  `set http_proxy = http://coia.hcvpc.io:9400`
  `set https_proxy = http://coia.hcvpc.io:9400`

  NOTE: test mit curl http://google.com geht nicht über Siemens Proxy !!

setup Lambda with dependency modules:
- from command shell navigate to js source folder
- npm install <modulename>
  => will create 'node_modules' folder, install package and all dependencies
- zip folder content including js source
- upload to lambda, then switch back to "edit inline"

use function from another (helper / lib) file:

1) export the function:

index.js | helper.js

```
module.exports = {
    helpertest: function(i) {
        console.log('helpertest: ' + i);
    }
}
```

2) import and use:

```
var helper = require('./helper.js');

helper.helpertest(66);
```

1.1. “promise” is an object or function with a `then` method whose behavior conforms to this specification.

A promise must provide a `then` method to access its current or eventual value or reason.

A promise's `then` method accepts two arguments:

```
promise.then(onFulfilled, onRejected)
```

2.2.1. Both `onFulfilled` and `onRejected` are optional arguments:
  2.2.1.1. If `onFulfilled` is not a function, it must be ignored.

PROMISES

```
let promise = doSomething();
promise.then(successCallback, failureCallback);

...or simply:
doSomething().then(successCallback, failureCallback);
```

Eingefügt aus <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises>

```
const promiseTimeout = function(ms, promise){
// Create a promise that rejects in <ms> milliseconds
  let timeout = new Promise((resolve, reject) => {
    let id = setTimeout(() => {
      clearTimeout(id);
      reject('Timed out in '+ ms + 'ms.')
    }, ms)
  })
// Returns a race between our timeout and the passed in promise
  return Promise.race([
    promise,
    timeout
  ])
}
```

Eingefügt aus <https://italonascimento.github.io/applying-a-timeout-to-your-promises/>

siehe auch
https://stackoverflow.com/questions/32461271/nodejs-timeout-a-promise-if-failed-to-complete-in-time

```
function run() {
    logger.info('DoNothingController working on process id {0}...'.format(process.pid));
myPromise(4000)
        .then(function() {
            logger.info('Successful!');
        })
        .catch(function(error) {
            logger.error('Failed! ' + error);
        });
}
function myPromise(ms) {
    return new Promise(function(resolve, reject) {
        var hasValueReturned;
        var promiseTimeout = setTimeout(function() {
            if (!hasValueReturned) {
                reject('Promise timed out after ' + ms + ' ms');
            }
        }, ms);
// Do something, for example for testing purposes
        setTimeout(function() {
            resolve();
            clearTimeout(promiseTimeout);
        }, ms - 2000);
    });
}
```

Eingefügt aus <https://stackoverflow.com/questions/32461271/nodejs-timeout-a-promise-if-failed-to-complete-in-time>

Chaining and using .catch

```
new Promise((resolve, reject) => {
      console.log('1) Initial');
      resolve();
})
      .then(() => {
            console.log('2) Resolved...');
            throw new Error('Something failed');
            console.log('but now an error happened!');
      })
      .catch(() => {
            console.log('3) ok, handling error...');
      })
      .then(() => {
            console.log('4) Cleaning up...');
      });
```

Eingefügt aus <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises>

Links & Tools

Dienstag, 26. Dezember 2017
07:50

Angular

- Angular Documentation: www.angular.io
- Angular packages for import: https://www.npmjs.com/~angular

Spring

- http://start.spring.io ==> create spring java project

Swagger

Donnerstag, 4. Januar 2018
11:08

Swagger Editor:

- https://api.siemens.com/editor/

add:

- host: 'starterservicesqa-vfc.eu-central-rc.mindsphere.io'
- basePath: /

CICD Training Chengdu

Dienstag, 20. März 2018
09:56

Create Gitlab project

- new project in code.siemens
- add file "**.gitlab-ci.yml**" with e.g.

```
hello:
    script: echo "Hello Chengdu"
```

- this will be executed whenever the pipeline runs

Set Up Runner:

- Gitlab / Settings / CICD
- Runner settings
- (disable shared runner)
- Enable "mindsphere-mainline runner"
  => this runner is in AWS global infra, peered with all other VPC's

go to **CICD / pipelines**

- go to Jobs
- Refresh
  ==> will execute script in own docker instance

Docker Hub:

- http://hub.docker.com/

.gitlab-ci.yml comands:

- stages => sequential jobs
- before_script: is executed before each single stage
- .jobname => job will NOT run

.gitlab-ci.yml

```
hello-backend:
   image: openjdk:8-slim
   script:
    - echo "Hello Chengdu Backend"
    - javac -version

hello-frontend:
   image: node:9.8.0-alpine
   script:
     - echo "Hello Chengdu Frontend"
     - node --version
```

==> each job executed in parallel, on separate Docker instance

```
stages:
  - backend
  - frontend

hello-backend:
  stage: backend
  image: openjdk:8-slim
  script:
   - echo "Hello Chengdu Backend"
   - javac -version

hello-frontend:
  stage: frontend
  image: node:9.8.0-alpine
  script:
    - echo "Hello Chengdu Frontend"
    - node --version
```

==> execute stages sequentially

```
stages:
- backend
- frontend
before_script:
- echo "Hello start script"
hello-backend:
stage: backend
image: openjdk:8-slim
script:
- echo "Hello Chengdu Backend"
- javac -version
hello-frontend:
stage: frontend
image: node:9.8.0-alpine
script:
- echo "Hello Chengdu Frontend"
- node –version
```

GITLAB

Dienstag, 20. März 2018
10:58

Set up SSH key for GIT

- in bash: ssh-keygen.exe
- add key to git (user profile / settings)

Workflow:

- edit file
- git -am "message…."
- git push

Git Documentation

- https://docs.gitlab.com
- https://docs.gitlab.com/ee/ci/yaml/