Angular

Angular

Attention

- Angular 2.. est très différent d'AngularJs (version 1)

Documentation

- https://angular.io/

Single Page Application - SPA

L'utilisateur reçoit une application pas une page

Pas de rafraichissement de la page

Pas de latence

Echange allégé en Json

Autres concepts :

- Routage
- Templating

Pré requis

NPM Node Package Manager

- fait partie de NodeJs (moteur JS intégrant un serveur web)
- https://nodejs.org/en/

2 langages : TypeScript et ES6

ECMASCRIPT 6, ES6 ou ECMASCRIPT 2015

- une nouvelle version de JS
- pas facile à maintenir pour les gros projets

L'équipe Angular2 a choisi TypeScript comme langage de référence

TypeScript (.ts)

- langage open source de Microsoft
- ajoute des fonctionnalités à ES dont le typage

Transpiler

- transformation des scripts ts en js

Installation - Anglular CLI

Installer le CLI
- npm install -g @angular/cli

Créer un projet (prend du temps...)
- ng new myproject

Créer un composant
- ng generate component my_component

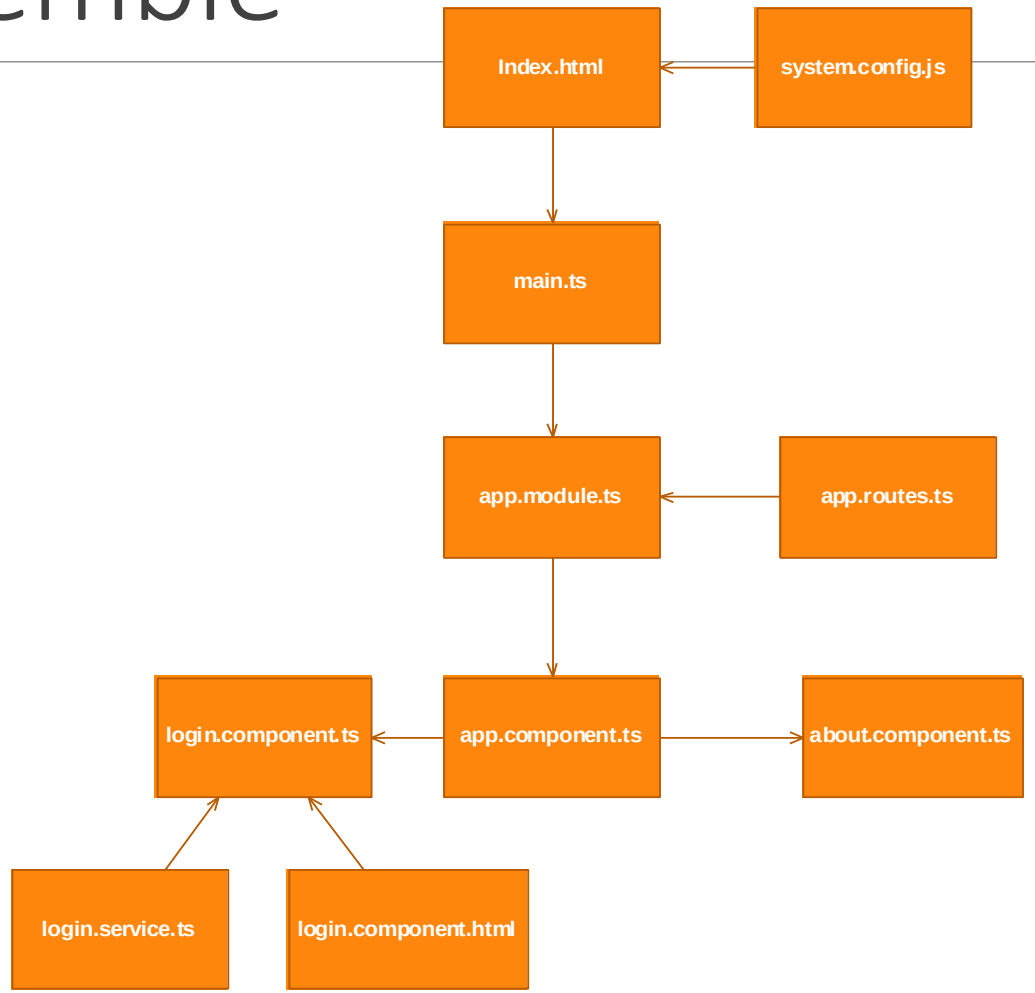
Lancer le serveur
- ng serve

Créer une version de production
- ng build

Créer une class
- ng generate class myClass

En cas de problème, vérifier la version de node et de npm

Vue d'ensemble



La page index.html

Page qui sera appelée par le serveur web

src/index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Dema</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Démarrage principal

src/main.ts

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));
```

Le 1er module

Angular est constitué de modules

Le CLI créée un 1er module

src/app/app.module.ts

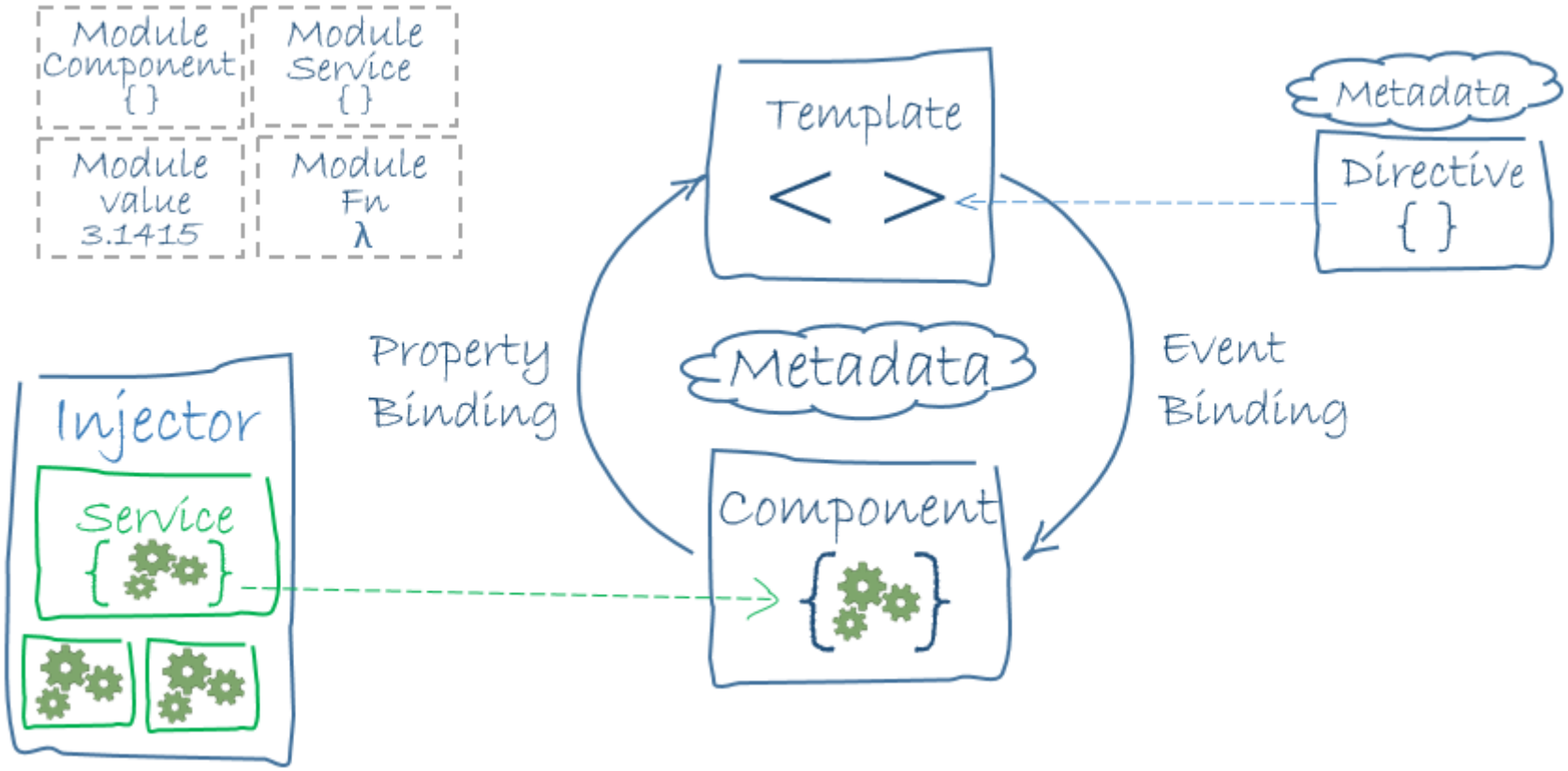
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';


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


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

Tout module contient au moins un composant

Le @ est un décorateur de la classe

L'export permet un utilisation dans un autre script

Les composants



Les composants

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

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

Tout affichage est un composant

Le CLI crée un 1er composant

src/app/app.component.ts

La classe est déclarée comme exportable et est décorée d'un Component

- selector : sélecteur est la balise html qui chargera le module
- template / templateUrl : contenu html (url relative à partir de src)
- style / styleUrl: ajout d'une css (url relative à partir de src)
- directive : permet de consommer une classe importée en utilisant son sélecteur
- provider : permet l'appel de services réutilisables permettant de manager les données
- ...

La variable *title* est initialisée pour être dans le fichier **src/app/app.component.html**

```
<h1>
  Welcome to {{ title }}!
</h1>
```

Utiliser des directives pour conditionner l'affichage des données

Afficher des données du modèle (interpolation) se fait avec {{ma_var}}

```
template: `
<h1>{{title}}</h1>
<h2>My favorite hero is: {{myHero.name}}</h2>
<p>Heroes:</p>
<ul>
  <li *ngFor="let hero of heroes">
    {{ hero.name }}
  </li>
</ul>
`
```

*ngFor est un itérateur

*ngIf permet de conditionner l'affichage

```
<p *ngIf="heroes.length > 3">There are many heroes!</p>
```

Debbuger votre application

2 sources d'information :

- La transpilation des scripts TS à l'aide de npm start
- La console JS du navigateur

Lier un événement au DOM

Dans l'HTML, l'événement entre parenthèses est lié à une méthode

```
<button (click)="onClickMessage()">Click me!</button>
<p>{{Message}}</p>
```

Dans la classe, on définit la méthode

```
export class AppComponent {
  title = 'demo';
  Message ='';

  onClickMessage() {
      this.Message = 'Message posted !';
  }
}
```

Les pipes

Les pipes permettent de traiter les données directement dans les templates

Par exemple, mettre en majuscules, minuscules, le formatage de dates..

- The hero's birthday is {{ birthday | date:"MM/dd/yy" }}

Il est possible de créer ses propres Pipes

4 formes de data binding

Interpolation

- {{name}}

Utilisation de variables locales, du composant au modèle

- [value] = "name"

Liaison par événement

- (click)="selectPerson(name)"

Liaison directe au modèle -> 2 way data binding

- [(ngModel)] = "name "

http://www.learn-angular.fr/le-data-binding-angular/

Formulaire- validation

Validation

- Le propriété *name* est utilisée pour détecter les changements associée à la déclaration de celle-ci en variable locale
- *Required* rend la case obligatoire
- Exemple
  - **<input** type="text" name="name" required [(ngModel)]="person.name" #name**>**

A ce stade un simple CSS permet de visualiser l'état de la validation

```
// valid and required show green
.ng-valid[required] {
  border-left: 5px solid #42A948; /* green */
}

// invalid
.ng-invalid {
  border-left: 5px solid #a94442; /* red */
}
```

Formulaire - validation

```
<label for="name">Name</label>

<input type="text" id="name" class="form-control"
      required   maxlength="24"
      name="name" [(ngModel)]="hero.name"
      #name="ngModel" >

<div *ngIf="name.errors && (name.dirty || name.touched)"
    class="alert alert-danger">
   <div [hidden]="!name.errors.required">
    Name is required
   </div>
    <div [hidden]="!name.errors.maxlength">
    Name cannot be more than 24 characters long.
   </div>
</div>
```

State	Class if true	Class if false
The control has been visited.	ng-touched	ng-untouched
The control's value has changed.	ng-dirty	ng-pristine
The control's value is valid.	ng-valid	ng-invalid

hidden est une propriété système qui cache l'élément

Communication parent -> enfant

Dans le parent, insertion du sélecteur et des variables transmises

```
<message [msg]="varContent" ></message>
```

Dans l'enfant

- Importer Input
- Déclarer l'attribut Input

```
@Input()  msg: String;
```

- Utiliser l'attribut dans le template

```
{{msg}}
```

Communication enfant -> parent

Dans le parent

- Récupérer l'événement dans le sélecteur

```
<message (close)="onClose($event)" ></message>
```

- Ajouter une méthode *onClose(content)* ayant un paramètre *content* dans ce cas-ci

Dans l'enfant, utiliser un événement pour avertir le parent

- Importer *EventEmitter*
- Déclarer un attribut *Output*

```
@Output() close: EventEmitter<any> = new EventEmitter();
```

- Appeler cet événement dans une méthode

```
onClose( ) {
    this.close.emit(this.content);
}
```

La gestion des routes

Routes

- décrit les routes des applications

RouterOutlet

- est un placeholder qui contiendra le contenu de chaque route

RouterLink

- est utilisé pour lier les routes à leur application

Le composant Routes

Créer un composant app.routes.ts

```
import { Routes } from '@angular/router';
import { LogintComponent }   from './login.component';
import { HomeComponent }   from './home.component';



export const routes: Routes = [
    { path: '', redirectTo: 'home', pathMatch: 'full' },
    { path: 'home', component: HomeComponent },
    { path: 'login', component: LoginComponent }
];
```

Définir la balise Base dans le head de index.html

```
<base href="/" >
```

Importation des routes

Intégration dans le module app.module.ts

- Import du module Router

```
import {RouterModule} from "@angular/router";
```

- Import des routes

```
import { routes} from './app.routes';
```

```
@NgModule({
    imports:      [ BrowserModule, FormsModule, HttpModule, RouterModule.forRoot(routes)]
```

Intégration de la navigation et du placeholder

Dans le composant app.component.ts

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

@Component({
    selector: 'my-app',
    template: `
    <nav>
        <a routerLink="/home" Accueil</a>
        <a routerLink="/login" Connexion</a>
        <a routerLink="/about" About</a>
    </nav>
    <router-outlet></router-outlet>`
})

export class AppComponent {  }
```

Navigation

Ajouter un bouton dans l'HTML

```
<button   (click)="gotoLogin()">Login</button>
```

Ajouter une fonction dans le composant qui redirige vers la route

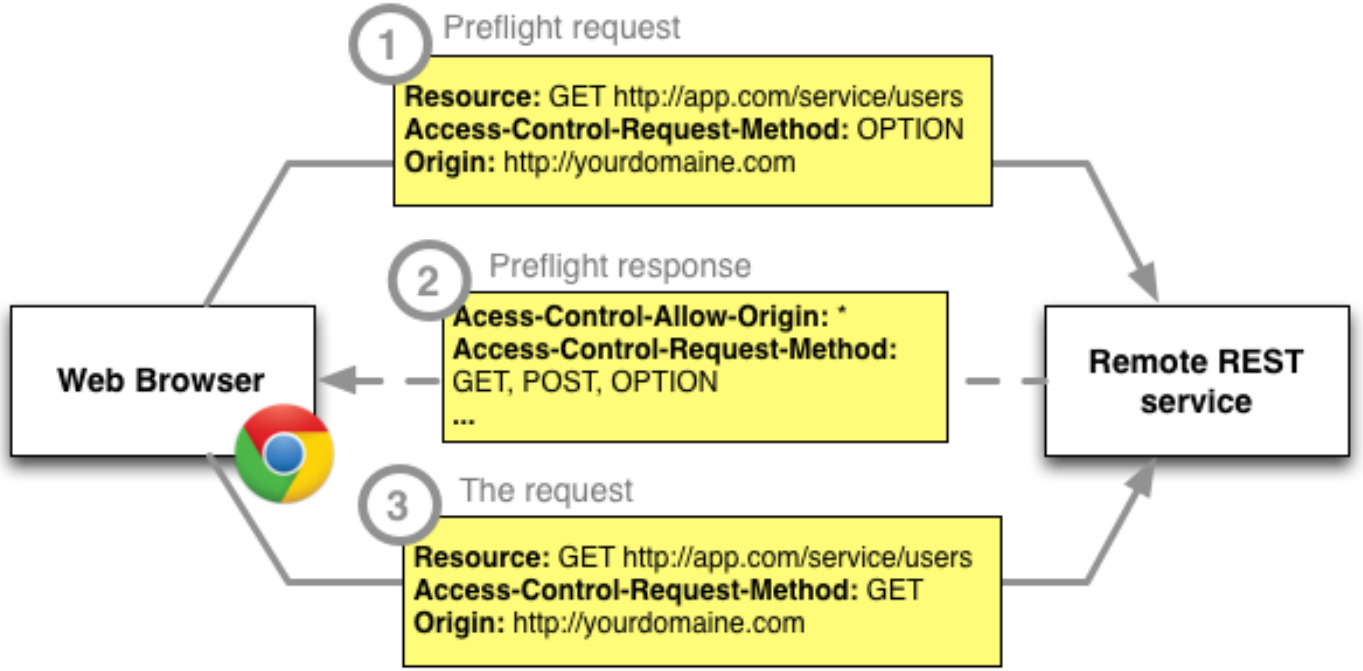
```
gotoLogin() : void {
  this.router.navigate(['/login']);
}
```

Création de l'API du côté Synfony

1. Récupérer les données
2. Renvoyer une réponse Json

```
$em = $this->getDoctrine()->getManager();
$categorys = $em->getRepository('notepadBundle:category')->findAll();
$categorys  = $this->get('serializer')->serialize($categorys , 'json');

$response = new JsonResponse();
$response->headers->set('Content-Type', 'application/json');

$response->setContent($categorys);

return $response;
```

Cross-Origin Resource Sharing (CORS)



Cross-Origin Resource Sharing (CORS)

Preflighted requests

- Le navigateur envoie une 1ère requête pour contrôler si le domaine et si la méthode sont acceptées
- Une requête d'effacement sera exécutée 2 fois et donc plantera à la seconde exécution
- La 1ère requête est envoyée avec la méthode http OPTIONS, on peut donc l'intercepter dans Symfony

```
 /**
     * @Route("/api/category/update/{id}", name = "api_category_update")
     * @Method({"PUT","OPTIONS"})
     */
public function updateAction(Request $request, $id)
{
     if($_SERVER['REQUEST_METHOD'] == 'OPTIONS')
     {
        $response = new Response();
        $response->headers->set('Content-Type', 'application/text');
        $response->headers->set('Access-Control-Allow-Origin', '*');
        $response->headers->set("Access-Control-Allow-Methods", "GET, PUT, POST, DELETE, OPTIONS");
        $response->headers->set('Access-Control-Allow-Headers', 'Content-Type',true);
        return $response;
     }
```

- Vérifier dans Postman que vous recevez bien ce header

Les services

Créer un service en décorant la classe avec @Injectable()

```
import {Injectable} from "@angular/core";
import {User} from "./user";


@Injectable()
export class UserService {   ... }
```

Utiliser un service en l'instanciant dans le constructeur

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

export class UserComponent  {

  constructor(
    private userService : UserService ) {  }
```

Chargement asynchrone

Observables

- permet un travail en temps réel sur les données
- plus intéressant que Promise
  - ne peut être appelé qu'une fois et ne retourne qu'une seule valeur
  - n'est pas annulable
- n'est disponible qu'à partir de ES7 mais également en important RxJS

Importation du module HttpClient

!! Ne pas se fier à la doc qui précède la version 4,3 d'Angular

Import dans src/app.module.ts

```
import { HttpClientModule } from '@angular/common/http';
```

```
imports: [
  BrowserModule,
  HttpClientModule,
],
```

Le composant Http pour l'accès à l'API

```
import {Component, OnInit} from '@angular/core';
import {Category} from './category';
import { HttpClient, HttpHeaders } from '@angular/common/http';

import { Observable } from 'rxjs/Observable';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent implements  OnInit {
  title = 'app';
  categories: Observable<Category[]>;
  private baseUrl = 'http://localhost/notepad_s4/public/index.php/api/category';

  constructor(private http: HttpClient) { }

  ngOnInit(): void {
    this.getCategories();
  }
  getServiceCategories(): Observable<any>  {
    return this.http.get(this.baseUrl + '/list');
  }
  getCategories() {
    this.getServiceCategories().subscribe(
      data => {
        this.categories = data;
        console.log(this.categories);
      }
    );
  }
}
```

Isoler les accès à l'API dans un service

Référence complète

https://angular.io/docs/ts/latest/guide/cheatsheet.html