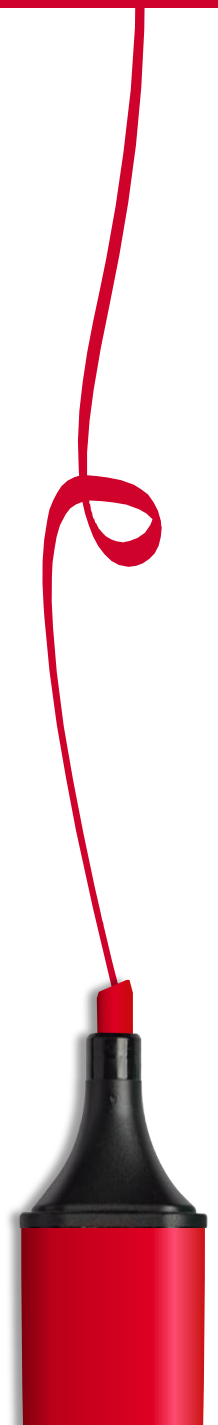
UNE JOURNÉE D'INITIATION À ANGULAR 4

Par sébastien.chassande@soprateria.com

AGENDA

Matin 8h30-11h30

- Javascript
- TypeScript
- Basiques Angular
  - Les composants
  - Les templates
  - Les directives
- Boite à Outils
- TP Premier pas



Après-midi 14h30-18h

- Les observables
  - TP Observable
- Les services
  - TP service
- Le routage
  - TP Routage

JavaScript

JAVASCRIPT : GÉNÉRALITÉS

- JavaScript != Java
- Orienté Objet
- Faiblement typé et dynamique
- Programmation OO, Impérative et fonctionnelle
- ECMAScript 5 (ES5) = JS
- Nouvelle version : ECMAScript 2015/ES6
  - Nouveauté : Classes, Constantes, Arrow functions, Modules
  - Transpileur vers ES5

JAVASCRIPT ES6 : VAR/LET/CONST

- Déclarer une variable avec **var** provoque le hoisting : déclaration de la variable tout en haut de la fonction.
- Nouveau mot clé : **let** : déclare la variable à l'endroit prévu
- Nouveau mot clé : **const** : déclare une constante à l'endroit prévu

```
function addAdminSuffix(user) {
    if (user.isAdmin) {
        var name = user.name + '_ADMIN';
        return name;
    }
    console.log(name); //not defined or undefined ???
    return user.name;
}
console.log(addAdminSuffix({name: 'Jocelyn', isAdmin: true}));
console.log(addAdminSuffix({name: 'Gauthier'}));
```

JAVASCRIPT ES6 : LES CLASSES

- Déclaration de classe
- Héritage
- Méthode static
- Constructeur
- Mot clé this obligatoire pour accèder aux membres
- New pour instancier

➔ Comme en Java !!

```
class Person {
  age() { return 30; }
}
class User extends Person {
  f1;
  login;
  static defaultRole() { return 'ROLE_USER'; }
  constructor(login) {
    super();
    this.login = login;
    this.f1 = 12;
  }
  toString() { return 'Utilisateur : ' + this.login; }
}

let jntakpe = new User('jntakpe');
console.log(jntakpe.toString());
console.log(jntakpe.age());
console.log(User.defaultRole());
```

JAVASCRIPT ES6 : GETTER ET SETTER IMPLICITE

- Déclaration du get et/ou du set

  get monChamp

- ➔ déclaration implicite du champ avec _ en prefix
- Le champ monChamp est accessible de l'extérieur. C'est le getter et le setter qui sont appelés.

```
class User {
  constructor(login) { this._login = login; }
  get login() {
     return this._login.toLowerCase();
  }
  set login(login) {
    if (login) {
       this._login = login.trim();
    }
  }

  toString() {
    return 'Login : \'' + this._login + '\';
  }
}

const jntakpe = new User();
jntakpe.login = ' JNtakpe';
console.log(jntakpe.login);
```

JAVASCRIPT ES6 : ARROW FUNCTION

- Syntaxe raccourci pour déclarer une fonction
- Plusieurs paramètres :
  (p1,p2,p3) => { console.log(p1, p2, p3); }
- this = le parent

```
D
const a = [
   "We're up all night 'til the sun",
  "We're up all night to get some",
   "We're up all night for good fun",
   "We're up all night to get lucky"];

// Sans la syntaxe des fonctions fléchées
const a2 = a.map(  function(s){ return s.length }   );
// [31, 30, 31, 31]

// Avec, on a quelque chose de plus concis
const a3 = a.map( s => s.length );
// [31, 30, 31, 31]
```

JAVASCRIPT ES6 : DESTRUCTURATION

Extraire des valeurs d'un objet

Objet intial

```
const user = {
    username: 'gpeel',
    authority: 'admin',
    address: { city: 'Aix' }
};
```

Création des constantes username et role

```
// renaming field
const { username, authority: role } = user;
console.log(username);
console.log(role);
```

Création des constantes selon un chemin

```
// cascading
const { address: { city } } = user;
console.log(city);
```

Extraire des valeurs d'un tableau

Objet intial

```
const arr = ['Jocelyn', 'Gauthier', 'Benjamin'];
```

```
const [a, b] = arr;
console.log(a);
console.log(b);
```

Création des constantes a et b

JAVASCRIPT ES6 : REST

- Utilisation de … pour indiquer le reste
- Dans les paramètres des fonctions

```
const formateurs = [];

function newPush(...formateursToPush) {
    for (let formateur of formateursToPush) {
        formateurs.push(formateur);
    }
}
newPush('Jocelyn', 'Gauthier', 'Sébastien');
console.log(formateurs);
```

- Dans les tableaux

```
const [firstForm, ...others] = formateurs;
console.log('First : ' + firstForm + ' ||| Rest: ' + others);
```

JAVASCRIPT ES6 : DESTRUCTURATION

- Les string template
- Evaluation du contenu des ${ … }
- Eviter de faire des +
- Multi ligne

```
const gpeel = {
        firstname: 'Gauthier',
        lastname: 'PEEL'
};

const fullname = 'Monsieur ' + gpeel.firstname + ' ' + gpeel.lastname;
console.log(fullname);

const tplFullname = `Monsieur ${gpeel.firstname} ${gpeel.lastname}`;
console.log(tplFullname);

const multiLine = `<div>
        <h1>Multi</h1>
        </div>`;
console.log(multiLine);
```

TypeScript

TYPESCRIPT : GÉNÉRALITÉS

- Typage statique et optionnel
- Superset de Javascript
- Injection par type
- Linter : analyse statique de code

TYPESCRIPT : LES FONCTIONS

- Typage du retour d'un fonction

```
function isWeekEnd(day: Days): boolean {
    return day === Days.Saturday || day === Days.Sunday;
}
```

- Paramètre optionnel

```
function setAge(age:number?): void {
    this.user.age = age;
}
```

- Paramètre par défaut

```
function setAge(age:number=20): void {
    this.user.age = age;
}
```

TYPESCRIPT : INTERFACE

- Une interface classique

```
interface Writer {
    write(message: string): void;
}
```

- Une interface d'un objet

```
interface User {
    name: string;
    age?: number;
}
```

- Une interface anonyme

```
function updateAge(user: {birthdate: Date, age: number} ):void {
    if (moment().dayOfYear() === moment(user.birthdate).dayOfYear()) {
        user.age++;
     }
}
```

TYPESCRIPT : LES DÉCORATEURS

- Ajoutés à TypeScript pour Angular
- Permettent de modifier la cible
- Utilisés en Angular pour les metadonnées
- Proposés dans ES7

```
@Component({selector: 'cpm-hello'])
export class HelloComponent {

   @Input() name: string;

   constructor() {
      console.log(`Hello ${name}`);
   }

}
```

QUELQUES OUTILS POUR ANGULAR

LES OUTILS À INSTALLER

- Node : https://nodejs.org/fr/ ➔ LTS
- Yarn : https://yarnpkg.com/lang/fr/ (gestionnaire de paquet meilleur que npm)
- IDE : Visual Studio Code https://code.visualstudio.com
- npm install -g typescript

LISTE D'EXTENSIONS CODE CONSEILLÉES

Visual Studio Code

- Auto Import
- Auto Rename Tag
- Autoend
- Beautify
- TsLint
- Bookmarks
- HTML CSS Support
- JS-CSS-HTML Formatter
- Sass
- Terminal
- Terminal Here
- Terminal Tabs
- Windows Explorer Context Menu
- Eclipse Keymap

ANGULAR CLI

- Boite à outils pour les projets Angular
- Github : https://github.com/angular/angular-cli
- npm install @angular/cli –g
- Génération d'un template de projet

  ng new PROJECT-NAME
  cd PROJECT-NAME
- Lancement du serveur / simulateur

  ng serve
  ➔ http://localhost:4200/

- Génération de composant

- Génération de composant

  ng g c path/MyComponent

Scaffold	Usage
Component	ng g component my-new-component
Directive	ng g directive my-new-directive
Pipe	ng g pipe my-new-pipe
Service	ng g service my-new-service
Class	ng g class my-new-class
Guard	ng g guard my-new-guard
Interface	ng g interface my-new-interface
Enum	ng g enum my-new-enum
Module	ng g module my-module

ANGULAR CLI TOOL

- Une autre CLI
- GitHub project : https://github.com/littleuniversestudios/angular-cli-tools
- npm install angular-cli-tools –g
- Template de projet :

Basic	https://github.com/littleuniversestudios/ng2-basic-seed
Bootstrap	https://github.com/littleuniversestudios/ng2-bootstrap-seed
Material Design	https://github.com/littleuniversestudios/ng2-material-seed
Firebase	https://github.com/littleuniversestudios/ng4-firebase-seed

- Custom projet template

PROJET TODOLIST : CRÉATION DU SQUELETTE DU PROJET

PROJET TODOLIST

1. Installer NodeJS
2. Ré-ouvrer un terminal pour la suite
3. Vérifier votre version de NPM: npm --version
4. Installer les modules suivants :
   - npm install -g @angular/cli
   - npm install -g typescript
5. Créer votre projet todolist : ng new todolist
6. Se placer dans le répertoire : cd todolist
7. Lancer le simulateur : npm start
8. Visualiser l'application créer dans votre navigateur: http://localhost:4200

LES BASIQUES

ANGULAR

- Framework web Javascript
- Open source
- Développé par Google
- Stable depuis septembre 2016
- Multi plateformes
- Rapide
- Généralement en TypeScript
  - Transpileur vers ES5 (JS supporté par les navigateurs)
  - Simulateur/serveur web

Google

ANGULAR BASIQUES

- Orienté composant (comme ReactJS, Aurelia, Vue.js)
- Application = assemblage de composants
- Intégration des web components
- Mais aussi des services, des directives, des guards …

COMPOSANT

- Composant =
  - Une classe (fichier .ts)
  - + Un template html
  - [+ des css]
- Template = fichier html séparé | inliné
- CSS = fichiers css séparés | inlinés
- selector : nom pour utiliser le composant depuis le template d'un autre composant

  ➔ Utilisation d'un composant par un autre template
- Les composants doivent être déclarés dans le module. (Auto via la CLI)

Fichier de la classe struct-component.ts

```
@Component({
        selector: 'app-struct',
        templateUrl: './struct.component.html'
        styleUrls: ['./struct.component.css']
})
export class StructComponent {
  colors: string[] = ['red', 'blue', 'green'];
}
```

Fichier template struct.component.html

```
<div *ngIf="colors.length > 0" class="too">
  <h2>Colors</h2>
  <ul>
     <li *ngFor="let color of colors; let i = index;">{{i}}-{{color}}</li>
  </ul>
</div>
```

Fichier css struct.component.css

```
.too {
  text-align: center;
}
```

COMPOSANT : CYCLE DE VIE

- Le composant implémente une interface pour être prévenu de son cycle de vie.
- Ci-contre un tableau des plus importants
- Initialisation du composant : Pas dans le constructor Mais dans ngOnInit() { … }

Interface	Méthode	Description
OnChange	ngOnChange()	Un des paramètres du composant a changé. Peut être appelé plusieurs fois.
OnInit	ngOnInit()	Le composant est initialisé. Est appelé une seule fois.
AfterViewInit	ngAfterViewInit()	Les vues du composant et de ses enfants sont initialisées.
OnDestroy	ngOnDestroy()	Le composant va être détruit par Angular.

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

@Component({
        selector: 'app-struct',
        templateUrl: './struct.component.html'
        styleUrls: ['./struct.component.css']
})
export class StructComponent implements OnInit {
  colors: string[] =[];

  public constructor(private myService: MyService) {}

  public ngOnInit() {
    this.colors = this.myService.getColors();
  }
}
```

TEMPLATE D'UN COMPOSANT

- Les doubles moustaches: {{ ..... }}
  - Expression évaluée
- Les champs/méthodes private de la classe ne sont pas visibles par le template
- Utilisation de **?** pour l'accès aux champs non encore chargés (lazy loading) ou optionnel

Exemple avec template et css inlinés

```
import {Component} from '@angular/core';
@Component({
 selector: 'app-interpolation',
 template: `<h1>{{titre}}</h1>
    <h3>Bonjour {{user.firstName + ' ' + user.lastName}}</h3>
    <p>Votre numéro de téléphone : {{user.contact?.phone}}</p>`
 styles: ['h1 { font-weight: normal; }']
})
export class InterpolationComponent {
 titre: string = 'Interpolation';
 user: any = {
  firstName: 'Sebastien',
  lastName: 'Chassande'
 };
}
```

Tout sur les styles: https://angular.io/guide/component-styles

DIRECTIVE : NGFOR

- Dans un template de composant
- Directive pour dupliquer un composant (html ou fabriqué)
- Le bloc contenant la directive sera dupliqué autant de fois qu'il y a d'élément dans le tableau
- Syntaxte :

  "let myVar of myArrayVar"

- Possibilité de connaitre l'index en ajoutant

  ;let myIndexVar = index;

```
<ul>
  <li *ngFor="let color of colors; let i = index;">{{i}}-{{color}}</li>
</ul>
```

DIRECTIVE : NGIF

- Dans un template de composant
- Directive pour afficher ou non un composant (html ou fabriqué)
- Le bloc contenant la directive sera créer ou pas selon la valeur de l'expression

```
<div *ngIf="colors">
  bla bla
</div>
```

- Depuis Angular 4.x : else

```
<div *ngIf="show; else myElseBlock">
    Text to show
</div>
 <ng-template #myElseBlock>
    Alternate text while primary text is hidden
</ng-template>
```

ORGANISATION DU CODE

- Réutilisation
  - Composant = unité de réutilisation dans une application
  - Module = unité de réutilisation entre des applications
- Organisation du code source
  - Un sous module par grand partie de l'application
  - Si pas inliné alors Un répertoire par composant (html, .ts, .css) (Comportement de la CLI)
  - Un répertoire pour tous les composants
  - Un répertoire pour tous les services
  - Un répertoire pour les classes de model

UTILISATION D'UN COMPOSANT PAR UN AUTRE COMPOSANT

```
import {Component} from '@angular/core';
@Component({
  selector: 'app-interpolation',
  template: `<h1>{{titre}}</h1>  <app-user-component [myUser]="user"></ app-user-component>`})
export class ParentComponent {
  titre: string = 'Interpolation';
  user: User;
  ngOnInit() { this.user = { firstName: 'Sebastien', lastName: 'Chassande' }; }
}
```

```
import {Component} from '@angular/core';
@Component({
  selector: ' app-user-component',
  template: `<h3>Bonjour {{myUser.firstName}} {{myUser.lastName}}</h3>`})
export class UserComponent {
  @Input() myUser: User;
}
```

PROJET TODOLIST : MES PREMIERS COMPOSANTS

DÉMARRAGE DE L'APPLICATION

1. Créer une interface Todo comme modèle: ng g class model/Todo
    1. Dans l'interface ajouter les champs:

    id?: number; // le ? Rend optionnel le champ.

    name: string;

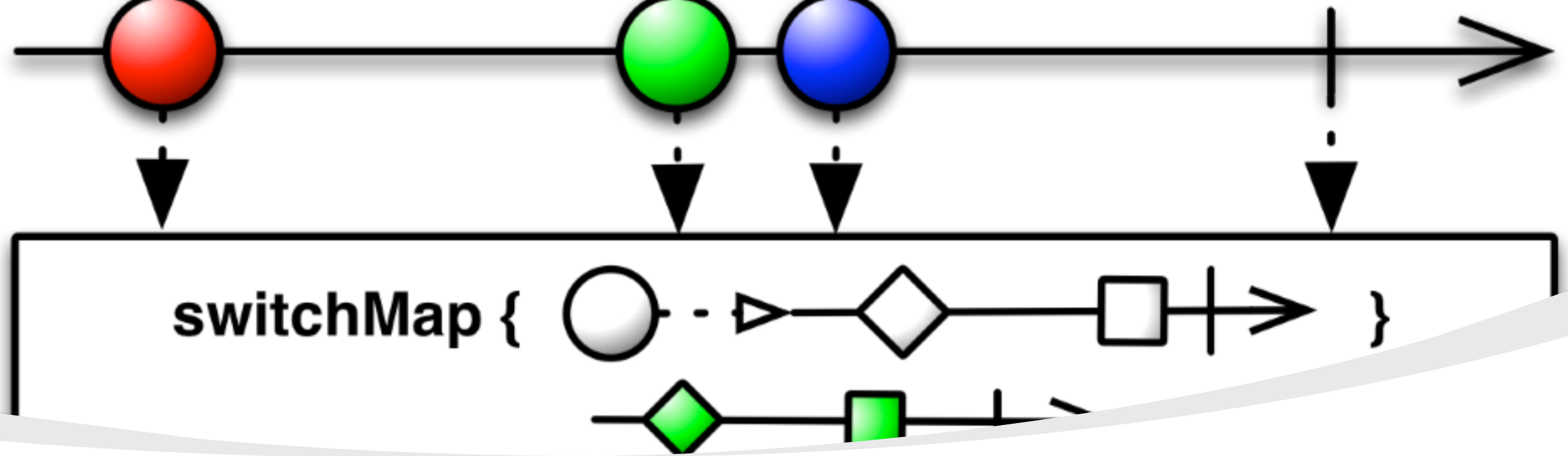
    completed:boolean

2. Créer un composant Todo: ng g c component/Todo

    Ce composant Affiche un objet Todo : Affichage du nom et d'une checkbox pour changer le flag completed

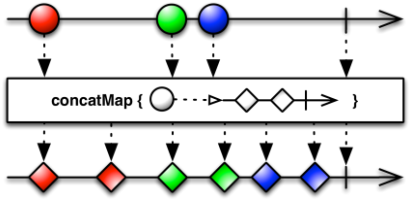
3. Créer un composant TodoList: ng g c component/TodoList
    1. Créer une variables todos: Todo[] = []
    2. Initialiser la variable todos dans un ngOnInit avec un tableau de 3 Todo de votre choix
    3. Dans le template afficher la liste des todos en appliquant un ngFor sur un composant Todo

4. Dans le template principal de l'application (app.component.html) ajouter l'utilisation du composant 'app-todo-list' : <app-app-todo-list></app-app-todo-list>



LES OBSERVABLES

RXJS



- Reactive extensions - Programmation réactive
- Flux événements
- Un événement peut être
  - une valeur,
  - une erreur ou
  - une terminaison
- Observables != Promises
- Asynchrone
- Existent dans de nombreux langage (JS, Java, .NET …)

```
interface Observable<T> {
    Subscription subscribe(Observer s);
}
```

```
interface Observer<T> {
   void onNext(T t);
   void onError(Throwable t);
   void onCompleted();
}
```

SYNC VS ASYNC

	Single	Multiple
Syncrhone	T	Iterable<T>
Asynchrone	Future<T>	Observable<T>

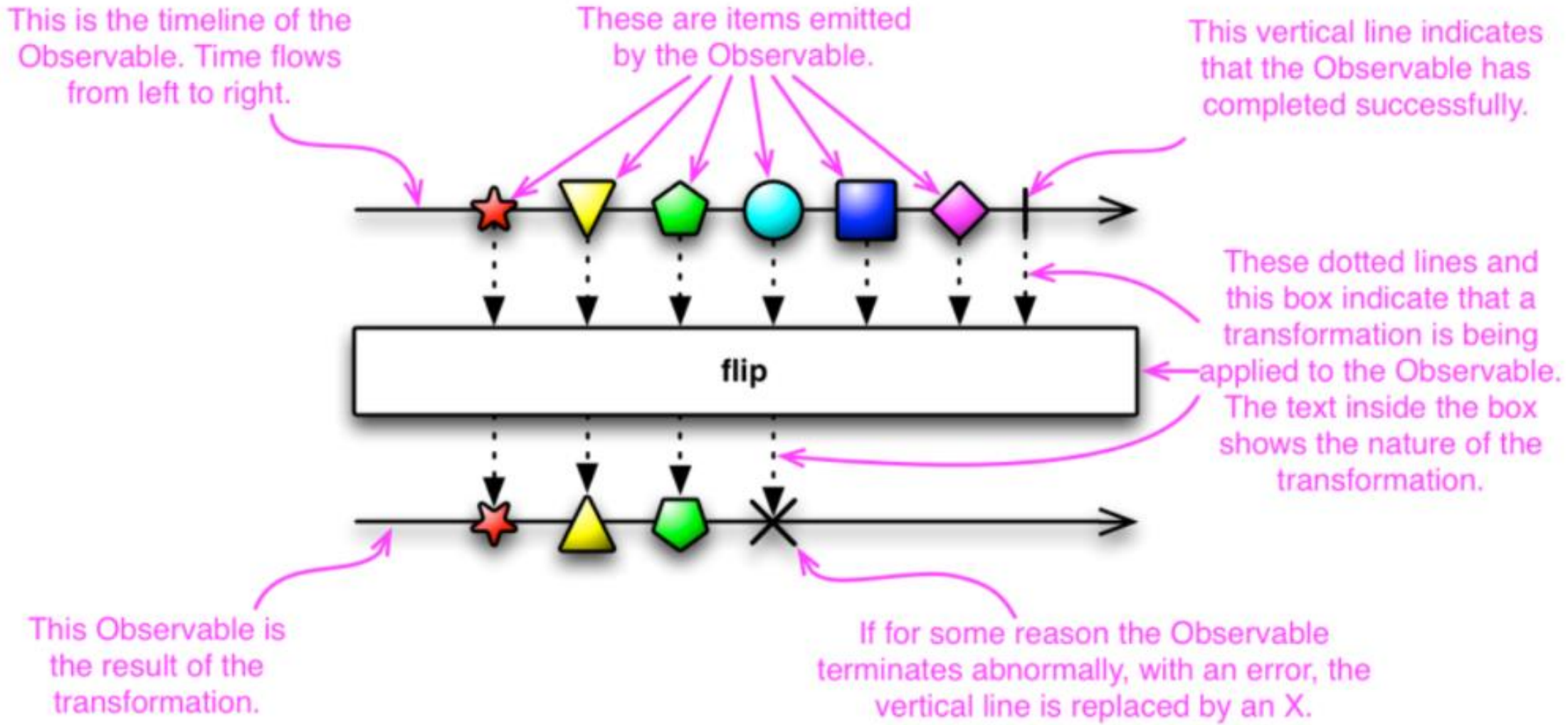
PULL VS PUSH

Pull	Push
Iterable	Observable
T next()	onNext(T)
throws Exception	onError(Exception)
returns	onCompleted()

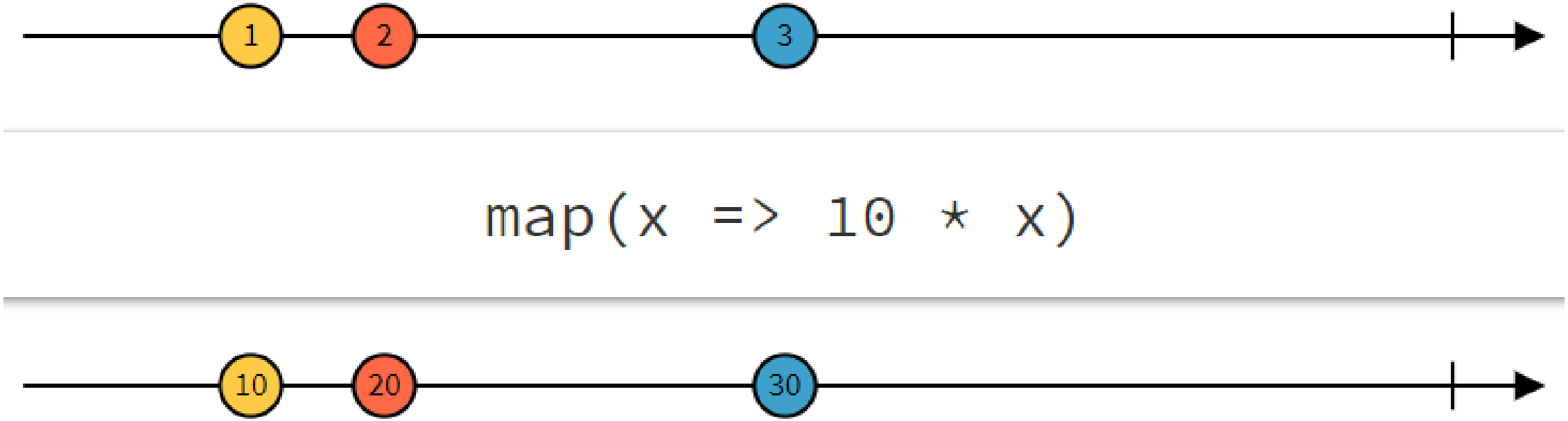
LES MARBLES

- Marbles : ensemble d'opérateurs pour manipuler le flux d'évenements
- Des exemples d'opérateur pour manipuler les observables
  - Création d'un observable: From, Of, interval, timer
  - Combinaison : concat, merge, combineLatest
  - Filtrage : distinct, filter, first, last, elementAt, find
  - Opération mathématique : count, max, min, reduce
  - Tranformation : map, repeat,
  - Utilitaire : delay
- http://rxmarbles.com/

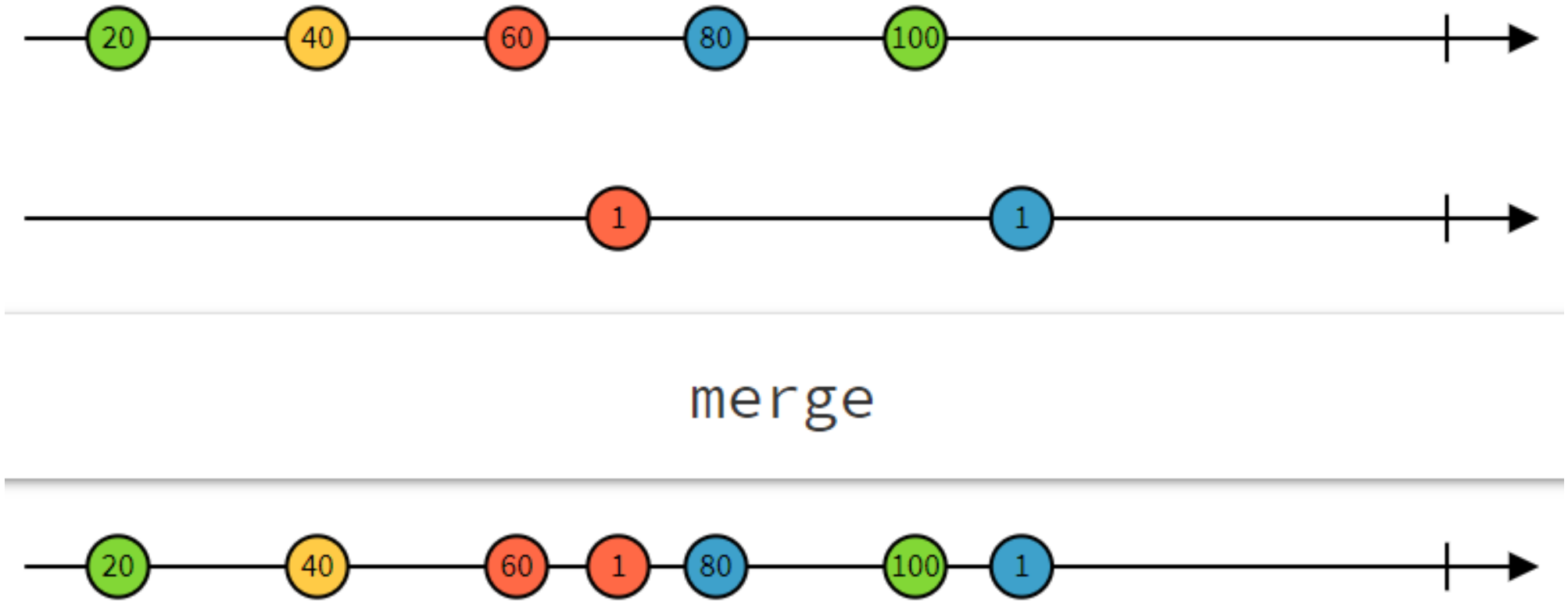
COMPRENDRE UN DIAGRAMME MARBLE



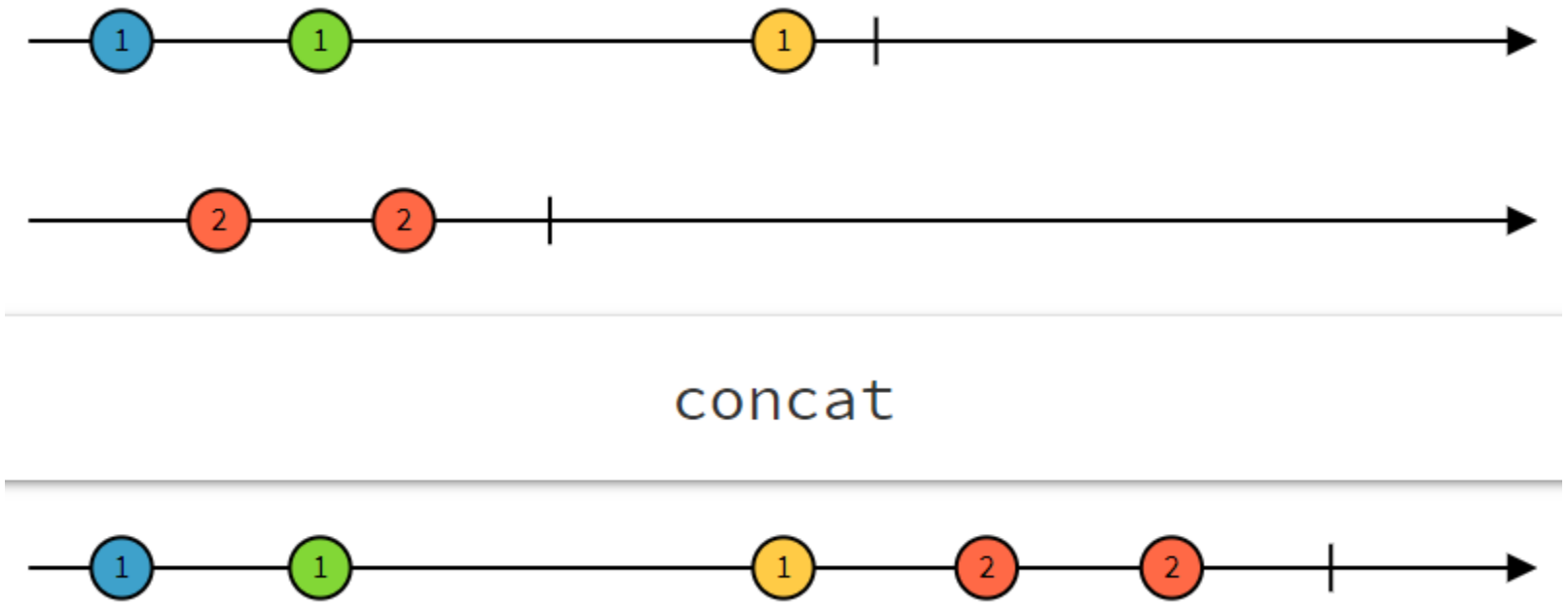
OPERATEUR : MAP



OPERATEUR : MERGE



OPERATEUR : CONCAT



COMMENT FAUT-IL UTILISER UN OBSERVABLE ?

- Il faut souscrire/s'abonner via la méthode subscribe sinon rien de ne se passe !
- Il y a 3 fonctions à donner en paramètres
  1. La fonction pour traiter un événement
  2. La fonction pour traiter le cas d'erreur
  3. La fonction pour traiter la fin de l'observable

```
// La création d'un observable
this.data = new Observable(observer => {
          setTimeout(() => { observer.next(42); }, 1000);
          setTimeout(() => { observer.next(43); }, 2000);
          setTimeout(() => { observer.complete(); }, 3000);
});

// L'inscription et la consommation d'un observable
let subscription = this.data.subscribe(
          value => this.values.push(value),
          error => this.anyErrors = true,
          () => this.finished = true );
}
```


QUAND FAUT-IL UTILISER UN OBSERVABLE ?

- Dés qu'une opération est asynchrone !
  - Exemple : Appel du serveur
- Nombre de données inconnues au départ

➔ les Services

LES SERVICES

UN SERVICE ANGULAR C'EST QUOI ?

- Service = simple class
- Singleton
- Déclaré dans un module
- Rôle
  - Stockage de données (Panier)
  - Appel du backend
  - Algorithme métier
- Injectable dans les composants ou d'autres services grâce à **@Injectable()**

```
@Injectable()
export class GithubService {

   constructor(private http: Http) {  }

   findReposForUser(username: string): Observable<any> {
     return this.http.get(`http://api.github.com/users/${username}/repos`)
                     .map(res => res.json());
    }
}
```

Injection du service http dans le service **Githubservice**

LE SERVICE HTTP

- Service Angular nommé Http permettant de récupérer des données
- Réalise les requêtes AJAX en utilisant XMLHttpRequest
- Renvoi des observables
- Nécessite l'import du module HttpModule
- Méthode : get, post, put, delete, patch, head
- Transformation du résultat grâce aux opérateur d'Observable

```
import {Injectable} from '@angular/core';
import {Http} from '@angular/http';
import {Observable} from 'rxjs';

@Injectable()
export class GithubService {

  constructor(private http: Http) {}

  findReposForUser(username: string): Observable<any> {
     return this.http.get(
        `http://api.github.com/users/${username}/repos`)
        .map(res => res.json());
  }
}
```

SERVICE HTTP : LES PARAMÈTRES DANS LE HEADER

```
public getRequestOptions(): RequestOptions {
    const headers: any = {
        'Content-Type': 'application/json',
        ‘version': ‘1.0',
        'X-Auth-Token': ‘ZERF43534R4RCZ4TRTZERTZER434‘
    }
    return new RequestOptions({ headers: new Headers(headers) });
}

public askResetPassword(email: string): Observable<Response> {
    this.log.d('askResetPassword(', email, ')');
    return this.http.get(`${env.serverUrl}/users/resetPassword/${email}`,
                         this.getRequestOptions())
        .map(res => { return { errorCode: 0}; })
        .catch(err => { return Observable.of({ errorCode: err.json()}); });
}
```


PROJET TODOLIST : MISE EN PLACE D'UN SERVICE

AVEC UN BACKEND DE STOCKAGE DES TODOS

- Le backend REST
  - Disponible en téléchargement sur http://chassande.brontes.feralhosting.com/formation/ecom/
  - Zip du projet Spring boot (3 classes …)
  - Jar exécutable
- Lancer l'exécutable sur votre ordinateur: java -jar backend-0.0.1-SNAPSHOT.jar
  - Base de données H2 mémoire
- CRUD
  - URL : http://localhost:8080/todos
  - Méthode : GET, POST, DELETE
  - Documentation de l'API en Swagger : http://localhost:8080/swagger-ui.html#/Todo32Entity
  - Utiliser Postman (via chrome) pour test l'API (https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop)

PROBLÈME DE ACCESS-CONTROL-ALLOW-ORIGIN ...

Le problème

Lorsque le serveur web et le backend REST ont des origines (URL) différentes le navigateur bloque l'accès au backend :

XMLHttpRequest cannot load http://localhost:8080/todos. No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://localhost:4200' is therefore not allowed access.

Solution en prod

Sur le serveur backend REST, supporter la requête http OPTIONS en spécifiant l'adresse des serveurs web autorisé dans le champ *Access-Control-Allow-Origin*

Solution pour le développement

Fichier: ./package.json, dans le champ scripts :

```
"start": "ng serve --proxy-config proxy.conf.json",
```

Fichier: ./proxy.conf.json

```
{
  "/": {
    "target":"http://localhost:8080",
    "secure": false,
    "pathRewrite": { "^/todos": "/todos" },
    "changeOrigin": true,
    "logLevel": "debug"
  }
}
```

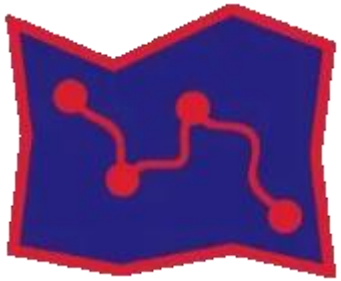
AJOUTER UN SERVICE ANGULAR

- Ajouter un service angular pour gérer les Todo: ng g s service/Todo
- Déclarer ce service dans le app.module.ts dans le tableau des ‘providers’
- Créer une fonction dans le service qui utilise le service http pour appeler le backend

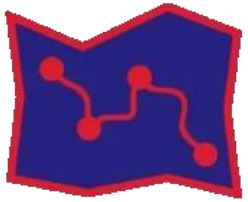
```
public all(): Observable<Todo[]> { … }
```

- Dans le composant de liste, utiliser le service
  - Dans le ngOnInit() : appeler le service pour récupérer les données
- Remplir votre base en effectuant qq requêtes POST /todos via postman

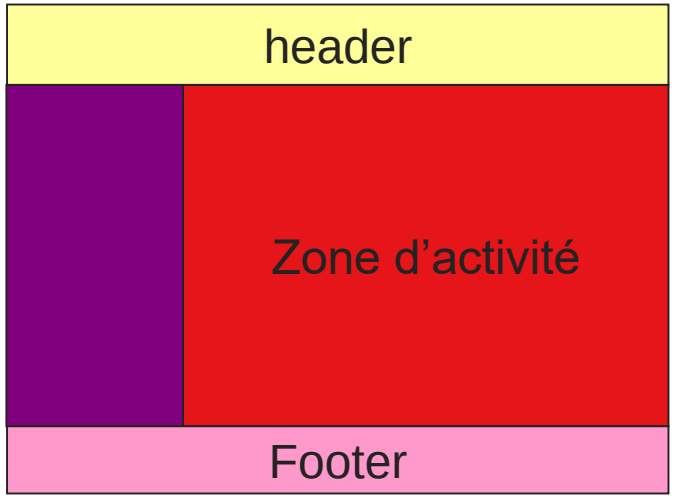
ROUTING



ROUTAGE : LE PRINCIPE



- SPA : Single Page association => Une seule page qui s'auto modifie via du JS
- Une application = { activités }
- une seule activité en même temps
- Routage d'activité en activité
- Utilisation de l'URL apres le # pour identifier l'activité et ses paramètres
- https://angular.io/docs/ts/latest/guide/router.html



CONFIGURATION DES ROUTES

- Dans le app.module.ts
  - Déclaration du module
  - Association des routes avec des composants

- Dans le html

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

```
import { RouterModule, Routes} from '@angular/router';
export const appRoutes: Routes = [
        { path: 'foo',       component: FooComponent },
        { path: 'bar/:id',  component: BarDetailComponent }];

@NgModule({
        declarations: [ AppComponent, ... ],
        imports: [ BrowserModule,
                FormsModule,
                HttpModule,
                RouterModule.forRoot(appRoutes) ],
        bootstrap: [AppComponent]})
export class AppModule { }
```


NAVIGATION

- Dans un template

  *<a **routerLink**="/bar/12">Bar 12*

- Dans du code .ts (composant/service)
  - *Injection du router dans le component*

    *constructor(**private router: Router**) { }*

  - *Puis utilisation dans une méthode*

    *this.router.**navigate**(['/bar', 12]);*

- Controler la navigation avec un Guard

```
import { Injectable } from '@angular/core';
import { CanActivate } from '@angular/router';
@Injectable()
export class AuthGuard implements CanActivate {
  canActivate() {
    console.log('AuthGuard#canActivate called');
    return true;
  }
}
```

```
import { AuthGuard } from '../auth-guard.service';
const adminRoutes: Routes = [
      {       path: 'admin',
              component: AdminComponent,
              canActivate: [AuthGuard]
       } ];
```

NAVIGATION : URL AVEC PARAMÈTRES

- Déclarer la route avec les paramètres

```
const appRoutes: Routes = [
  {path: 'resetPassword/:userId/token/:token', component: MyComponent},
  {path: '**', redirectTo: 'accueil'}
];
```

- Récupérer la valeur des paramètres dans le composant

```
import { ActivatedRoute, Params } from '@angular/router';
export class ResetPasswordComponent implements OnInit {
  public userId = 0;
  public token: string;

  constructor(private activatedRoute: ActivatedRoute) { }

  ngOnInit() {
    this.activatedRoute.params.subscribe((params: Params) => {
       this.userId = params['userId'];
       this.token = params['token'];
    });
  }
}
```

PROJET TODOLIST : PLUSIEURS PAGE AVEC DU ROUTAGE

METTRE EN PLACE PLUSIEURS PAGES

- Ajouter un composant Home: ng g c component/Home
- Ajouter un composant Help: ng g c component/Help
- Dans la page principale ajouter :
  - Une zone de service
  - Un menu qui permet de naviguer sur 3 pages : home, help ListTodo
- Configurer le routage dans le app.module.ts

LES FORMULAIRES

FORMULAIRE

- Un formulaire est représenté par un *FormGroup*
  - *valid* : si le formulaire est valide
  - *errors* : objet contenant les erreurs du formulaire
  - *dirty* : false jusqu'à ce que l'utilisateur modifie un des champs du formulaire
  - *pristine* : opposé de dirty
  - *touched* : false jusqu'à ce que l'utilisateur soit entré et sorti d'un des champs du formulaire
  - *untouched* : opposé de touched
  - *value* : la valeur du formulaire sous forme de clé/valeur avec le nom du champ en clé
  - *valueChanges* : un observable emettant la valeur du formulaire à chaque changement sur un des champs

CHAMP DE FORMULAIRE

- Un champ de formulaire est un *FormControl* :
  - *valid* : si le champ est valide avec les validations et contraintes qui lui sont appliquées
  - *errors* : objet contenant les erreurs du champ
  - *dirty* : false jusqu'à ce que l'utilisateur modifie le champ
  - *pristine* : opposé de dirty
  - *touched* : false jusqu'à ce que l'utilisateur soit entré et sorti du champ
  - *untouched* : opposé de touched
  - *value* : la valeur du champ
  - *valueChanges* : un observable emettant la valeur du champ à chaque changement

FORMULAIRE DIRIGÉ PAR LE **TEMPLATE** : LES DONNÉES DANS L'OBJET FORM

- Le template contient des directives ngModel
- A la soumission, la méthode analyse l'objet form qui contient les valeurs

```
<h2>Sign up template form</h2>
<form (ngSubmit)="register(loginForm)" #loginForm="ngForm">
  <div class="form-group">
    <label for="username">Nom d'utilisateur</label>
    <input type="text" class="form-control" id="username"
     name="username" ngModel>
  </div>
  <div class="form-group">
    <label for="password">Password</label>
    <input type="password" class="form-control" id="password"
     name="password" placeholder="Mot de passe" ngModel>
  </div>
  <button type="submit">Se connecter</button>
</form>
```

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

@Component({...})
export class SignupFormComponent {
  register(form: NgForm) {
    console.log(form.value);
    // ...
  }
}
```


FORMULAIRE DIRIGÉ PAR LE **TEMPLATE** : TWO WAY BINDING

- Le template fait le lien avec le modèle accessible depuis les champs de la classe
- [(ngModel)]="a.b.monChamp"

```
<h2>Sign up template form</h2>
<form (ngSubmit)="register()">
  <div class="form-group">
    <label for="username">Nom d'utilisateur</label>
    <input type="text" class="form-control" id="username"
     name="username" [(ngModel)]="user.name">
  </div>
  <div class="form-group">
    <label for="password">Password</label>
    <input type="password" class="form-control" id="password"
     name="password" [(ngModel)]="user.password">
  </div>
  <button type="submit">Se connecter</button>
</form>
```

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

@Component({...})
export class SignupFormComponent {
  user: User;
  register() {
    console.log(user.name);
    // ...
  }
}

export interface User {
  name: string;
  paswword: string;
}
```

FORMULAIRE DIRIGÉ PAR LE **TEMPLATE** : LA VALIDATION

- La validation des champs est définie dans le template par des attributs HTML5
- Définition d'une variable pour accéder à l'objet FormControl de chaque champ :

#myfield="ngModel"

```
<input id="name" name="name" required minlength="4" [(ngModel)]="hero.name" #name="ngModel" >

<div *ngIf="name.invalid && (name.dirty || name.touched)" class="alert alert-danger">
  <div *ngIf="name.errors.required">Name is required.</div>
  <div *ngIf="name.errors.minlength">Name must be at least 4 characters long.</div>
</div>
```

FORMULAIRE **RÉACTIF**

- L’association donnée/composant graphique s’effectue dans le code ts
- Plus compliqué et plus lourd
- Mais plus puissant
  - Binding complexe
  - Validation complexe/croisée
- https://angular.io/guide/reactive-forms

➔ Pour les formulaires complexes seulement

PROJET TODOLIST : UNE PAGE D'ÉDITION/CRÉATION D'UN TODO

PROJET TODOLIST : UNE PAGE D'ÉDITION/CRÉATION D'UN TODO

- Ajouter une page d'édition d'un Todo existant ou nouveau contenant formulaire de Todo : le nom et completed

- Consignes
  - Créer un composant dédié qui sait faire l'édition et la création en même temps
  - La page est activable selon 2 urls :
    - Pour la création: /todo
    - Pour l'édition: /todo/:todoId
  - Dans le ngOnInit
    - Récupérer le todoId s'il y en a un
    - S'il y a un todoId alors appeler le backend pour récupérer l'objet: voir swagger du backend
    - Faire une méthode get(todoId) dans le service TodoService
  - Permettre la sauvergarde du todo (nouveau ou existant) en appelant le backend
    - Création ➔ /todo, Méthode http POST
    - Modification ➔ /todo/${todo.id}, Méthode http PUT
    - Faire une seule méthode save(Todo) dans le service TodoService qui s'adapter
  - Dans la page TodoList, ajouter les liens pour éditer un Todo existant et un lien pour ajouter un nouveau todo

CONCLUSION

QU'EST-CE QU'ON A APPRIS ?

- Des évolution JS : des classes
- Ecriture du code TS qui est compilé en JS pour être exécuté par le navigateur
- Ecrit plein de composants
- Utiliser la CLI pour programmer et tester
- L'accès à des services distant en http
- La mise en place de routage entre pages
- Comment faire des formulaires

Delivering Transformation. Together.