- JS / TS
- Histoire
- Le framework
- Ionic

Définition : Le JavaScript est un langage de scripting.

Il était d'abord utilisé pour écrire des scripts de quelques lignes attachés au navigateur.

Aujourd'hui il fonctionne même en dehors du navigateur (Node.js).

```
if ("" == 0) {
// It is! But why??
}
if (1 < x < 3) {
// True for *any* value of x!
```

Le compilateur de JS traduit deux valeurs de type différent au même type avant de les comparer. Il est donc permissif avec ce genre d’erreurs.
Ces “surprises” peuvent être embêtantes sur des programmes très grands.

Définition : TypeScript est un sur-ensemble typé de JavaScript.

Il détecte les erreurs de type.

Le code TypeScript est transcompilé en JavaScript.

JS

```
console.log(4 / []);
```

TS

```
console.log(4 / []);
```

```
The right-hand side of an arithmetic operation must be of type 'any',
'number', 'bigint' or an enum type.
```

Propriété : Tout code JavaScript est compris par un fichier TypeScript.

Définition : Un framework est un ensemble cohérent de composants logiciels. Il se distingue d'une bibliothèque par le cadre de travail qu'il impose au développeur.

Définition : Angular possède un système de **modules**.

Un module Angular est un mécanisme permettant de :
regrouper des composants (mais aussi des services, directives, pipes etc...)
- définir leurs dépendances
- définir leur visibilité.

Les inclusions de Services et modules se font dans le app.module.ts .

```
@NgModule({
  declarations: [
    AppComponent,
    ListImagesComponent,
    ListImagesItemComponent,
    MyPhotosGalleryComponent,
    PerDayPhotosComponent,
    ApolloVideosComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    FormsModule,
    RouterModule.forRoot(appRoutes)
  ],
  providers: [
    getImagesService,
    myGalleryService,
    getApolloImagesService
  ],
  bootstrap: [AppComponent]
})
export class AppModule {

}
```

Propriété : Angular est un framework côté client (Front-end) open source, basé sur le langage TypeScript. Il sert à créer des applications web monopage (single-page applications).

Avantages :

- Géré par Google
- Ionic
- TypeScript

Définition : Angular a une architecture basée sur des **composants**.

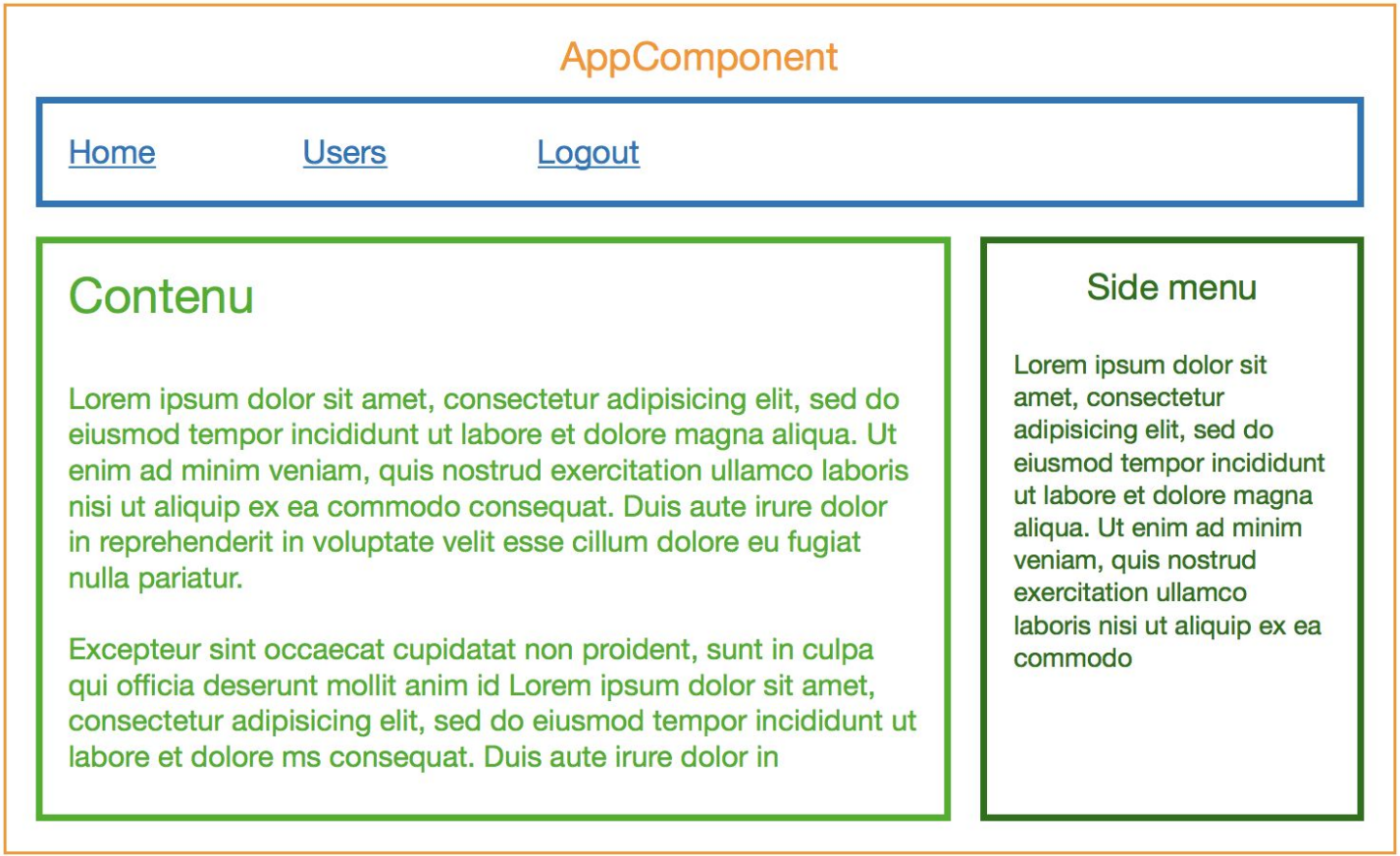
Un composant Angular est :

- Un bout d'interface de l'application, il peut représenter un bouton, une fenêtre de tchat, une barre de navigation...
- Il peut contenir d'autres composants.
- Il peut être affiché dynamiquement.

Propriété : Le point d’entrée de toute application angular est le composant AppComponent, toujours inclus dans le fichier index.html.

Index.html

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


AppComponent
Home
Users
Logout
Contenu
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore ms consequat. Duis aute irure dolor in
Side menu
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo

Propriété : Le comportement des composants est décrit dans les métadonnées, en utilisant le système d’annotations (ou décorateurs).

Chaque composant possède :

- Une vue
- Une classe typescript
- Une feuille de styles

Métadonnées du AppComponent

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

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

Selecteur du component

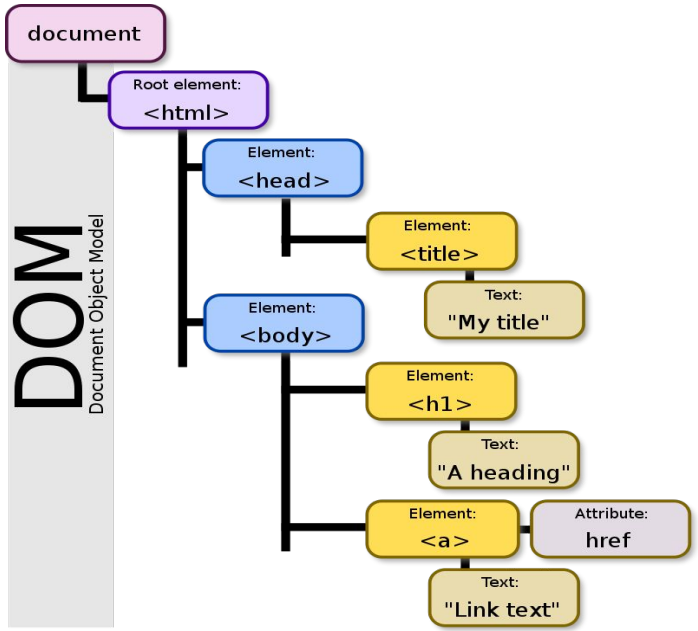
```
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-mon-premier',
5    templateUrl: './mon-premier.component.html',
6    styleUrls: ['./mon-premier.component.scss']
7  })
8  export class MonPremierComponent implements OnInit {
9
10   constructor() { }
11
12   ngOnInit() {
13   }
14
15 }
```

Utiliser le component

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
</div>
<app-mon-premier></app-mon-premier>
```

Le DOM

Définition : Le DOM est une Interface de programmation qui permet à des scripts de modifier le contenu d'une page web. Il possède une structure en arbre.



A quoi sert Angular ?

Angular gère le DOM grâce au **databinding.**

Définition : Le **databinding** est la communication de données entre la vue (fichiers html) et la logique (fichiers js)

Il y a quatre types de data-binding

1. Le string interpolation
2. Le property binding
3. L’event binding
4. Le two-way binding

Du code vers le HTML : le string interpolation

Définition : Le string interpolation sert à afficher dans la vue une donnée définie en typescript.

Du code vers le HTML : le string interpolation

Définition d'une variable dans le typescript.

```
export class AppareilComponent implements OnInit {

  appareilName: string = 'Machine à laver';

  constructor() { }
```

Du code vers le HTML : le string interpolation

Affichage de la variable dans la vue

```
<li class="list-group-item">
  <h4>Appareil : {{ appareilName }}</h4>
</li>
```

Les doubles accolades servent à évaluer l'expression.

Du code vers le HTML : le string interpolation avec retour de fonction

On peut également interpréter le retour d'une méthode définie dans le typescript.

```
<li class="list-group-item">
  <h4>Appareil : {{ appareilName }} -- Statut : {{ getStatus() }}</h4>
</li>
```

Du code vers le HTML : le property binding

Définition : Le property binding sert à interpréter une propriété définie dans le typescript dans un attribut html.

```
<button class="btn btn-success" [disabled]="isAuth">Tout allume</button>
```

Les crochets servent à évaluer l'expression qui se trouve dans les guillemets.

```
export class AppComponent {
  isAuth = false;
}
```

Du HTML vers le TypeScript : l'event binding

Définition : L'event binding sert à exécuter un programme au lancement d'un événement.

```
<button class="btn btn-success"
        [disabled]="!isAuth"
        (click)="onAllumer()">Tout allumer</button>
```

Fonction de l’event binding

```
onAllumer() {
    console.log('On allume tout !');
}
```

La liaison à double sens : le two-way Binding

Définition : Le two-way Binding permet de lier une donnée des deux côtés ; la modification côté HTML génère la modification côté TypeScript et inversement.

Le two-way binding utilise le property binding et l'event binding. C'est pourquoi on utilise les crochets du property binding et les parenthèses de l'event binding.

```
<li class="list-group-item">
  <h4>Appareil : {{ appareilName }} -- Statut : {{ getStatus() }}</h4>
  <input type="text" class="form-control" [(ngModel)]="appareilName">
</li>
```

Les propriétés personnalisées

Définition : Les propriétés personnalisées permettent de personnaliser les données d’un composant en lui transmettant des données depuis d’autres composants.

Les propriétés personnalisées

On crée une variable en @Input dans le composant fils.

```
export class AppareilComponent implements OnInit {

  @Input() appareilName: string;

  appareilStatus: string = 'éteint';

  constructor() { }

  ngOnInit() {
  }

  getStatus() {
    return this.appareilStatus;
  }

}
```

Les propriétés personnalisées

On crée des variables dans le composant père.

```
export class AppComponent {
  isAuth = false;

  appareilOne = 'Machine à laver';
  appareilTwo = 'Frigo';
  appareilThree = 'Ordinateur';

  constructor() {
```

Les propriétés personnalisées

On transmet les variables aux composants fils par property binding.

```
<ul class="list-group">
    <app-appareil [appareilName]="appareilOne"></app-appareil>
    <app-appareil [appareilName]="appareilTwo"></app-appareil>
    <app-appareil [appareilName]="appareilThree"></app-appareil>
</ul>
```

Définition : Les directives prédéfinies sont des instructions fournies par Angular pour faciliter le travail de programmation au développeur.

Il y a deux types de directives prédéfinies :

- Les directives structurelles
- Les directives par attribut

Définition : Les directives structurelles modifient **l'affichage** des éléments du DOM (afficher, enlever, afficher plusieurs fois, etc…).

On les utilise avec la notation astérisque *nomDirective (camel case).

ngIf

Définition : Le *ng if* est une directive structurelle qui sert à afficher un élément html selon la valeur d'une condition.

```
<li class="list-group-item">
  <div style="width:20px;height:20px;background-color:red;"
       *ngIf="appareilStatus === 'éteint'"></div>
  <h4>Appareil : {{ appareilName }} -- Statut : {{ getStatus() }}</h4>
  <input type="text" class="form-control" [(ngModel)]="appareilName">
</li>
```

Définition : Le ng for est une directive structurelle qui permet d'itérer sur les éléments d'un tableau.

```
export class AppComponent {
  isAuth = false;

  appareils = [
    {
      name: 'Machine à laver',
      status: 'éteint'
    },
    {
      name: 'Frigo',
      status: 'allumé'
    },
    {
      name: 'Ordinateur',
      status: 'éteint'
    }
  ];

  constructor() {
```

ngFor

```
<div class="container">
  <div class="row">
    <div class="col-xs-12">
      <h2>Mes appareils</h2>
      <ul class="list-group">
        <app-appareil  *ngFor="let appareil of appareils"
                       [appareilName]="appareil.name"
                       [appareilStatus]="appareil.status"></app-appareil>
      </ul>
      <button class="btn btn-success"
              [disabled]="!isAuth"
              (click)="onAllumer()">Tout allumer</button>
    </div>
  </div>
</div>
```

Les directives par attribut

Définition : Les directives par attribut modifient **l'apparence** d'un objet HTML (balises) ou d'un composant.

Elles utilisent la notation [nomDirective]=”valeur” (property binding et camel case).

Définition : la directive par attribut ng style permet d'ajouter des styles à un élément html de manière dynamique.

```
1 <h4 [ngStyle]="{color: getColor()}">Appareil : {{ appareilName }} -- Statut : {{ getStatus() }}</h4>
```

La directive ng style permet de retourner la valeur d'un attribut CSS avec une fonction.

```
getColor() {
    if(this.appareilStatus === 'allumé') {
      return 'green';
    } else if(this.appareilStatus === 'éteint') {
      return 'red';
    }
}
```

ngClass

Définition : La directive par attribut ng class permet d'ajouter des classes déjà définies à un élément html dynamiquement.

```
<li [ngClass]="{'list-group-item': true,
                'list-group-item-success': appareilStatus === 'allumé',
                'list-group-item-danger': appareilStatus === 'éteint'}">
  <div style="width:20px;height:20px;background-color:red;"
       *ngIf="appareilStatus === 'éteint'"></div>
  <h4 [ngStyle]="{color: getColor()}">Appareil : {{ appareilName }} -- Statut : {{ getStatus() }}</h4>
  <input type="text" class="form-control" [(ngModel)]="appareilName">
</li>
```

Directives custom

Définition : Les directives custom sont des instructions “customisées” par le développeur.

On utilise les directives custom en général pour altérer le comportement des éléments de la vue.

```
<button appExample>Button</button>
```

Directives custom

La création d'une directive custom ressemble beaucoup à celle d'un composant.

```
import { Directive } from '@angular/core'

@Directive({
  selector: '[appExample]',
})
export class ExampleDirective {
  constructor() {}
}
```

Directives custom

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

@Directive({
  selector: '[appExample]',
})
export class ExampleDirective {
  constructor(elementRef: ElementRef) {

elementRef.nativeElement.style.backgroundColor
= '#fff'
  }
}
```

Les pipes

src/app/hero-birthday1.component.ts

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

@Component({
  selector: 'app-hero-birthday',
  template: `<p>The hero's birthday is {{ birthday | date }}</p>`
})
export class HeroBirthdayComponent {
  birthday = new Date(1988, 3, 15); // April 15, 1988
}
```

Les pipes

Définition : Les pipes permettent de transformer le format d'un élément.

src/app/app.component.html

```
<p>The hero's birthday is {{ birthday | date }}</p>
```

src/app/app.component.html

```
<p>The hero's birthday is {{ birthday | date:"MM/dd/yy" }} </p>
```

src/app/app.component.html

```
The chained hero's birthday is
{{ birthday | date | uppercase}}
```

Les custom pipes

On peut également définir des custom pipes. Ils seront utilisés de la même façon qu'un pipe prédéfini.

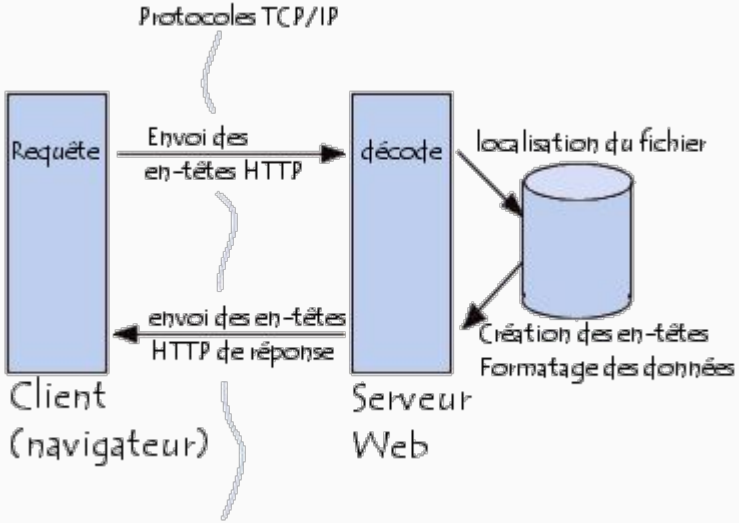
src/app/exponential-strength.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';
/*
 * Raise the value exponentially
 * Takes an exponent argument that defaults to 1.
 * Usage:
 *   value | exponentialStrength:exponent
 * Example:
 *   {{ 2 | exponentialStrength:10 }}
 *   formats to: 1024
*/
@Pipe({name: 'exponentialStrength'})
export class ExponentialStrengthPipe implements PipeTransform {
  transform(value: number, exponent?: number): number {
    return Math.pow(value, isNaN(exponent) ? 1 : exponent);
  }
}
```

Définition : Les Services sont des classes permettant de centraliser des parties du code utilisées par plusieurs parties de l’application.

Requête HTTP

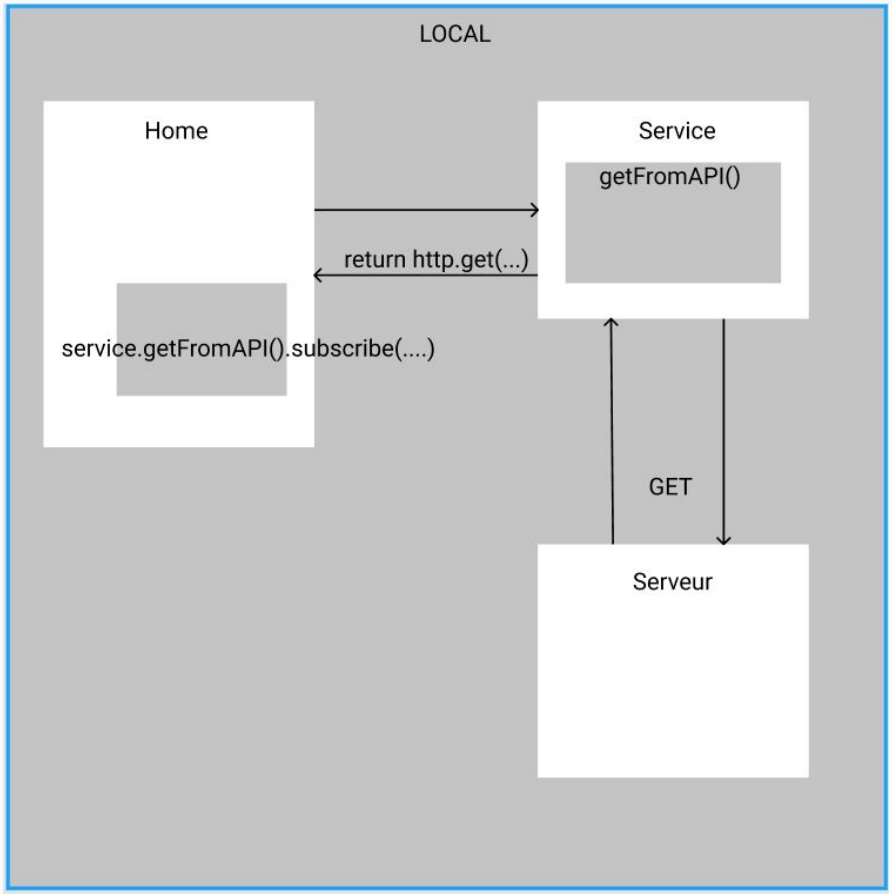
Définition : Une requête HTTP est un ensemble d'informations envoyées à un serveur par le navigateur. Le serveur peut répondre avec des données ou pas.



Observables

Définition : Les objets Observables sont des Promesses++. Ils permettent d'écouter un stream de valeurs reçus d'une requête HTTP.

Http.get et subscribe



Exemple : La méthode get retourne un objet de type Observable. Cet observable ne constitue pas la réponse, mais une Promesse de la réponse.

On type le retour de la réponse avec un any ou avec le type de la réponse si on le connaît.

```
getFromAPI() {
    return this.http
      .get<any>(...)
}
```

Subscribe

La méthode subscribe nous permet d'obtenir la réponse à l'appel HTTP en souscrivant à l'objet observable.

```
getFromApi(){

  this.appareilService.getFromAPI().subscribe(
    (response) => {

    },
    (error) => {

    }
  );
}
```