



---

# ARCHITECTURE ET DEVELOPPEMENT WEB AVANCE

M. PESSA MASSENE DAVE ARTHUR

PRESENTATION DU CHAPITRE 2

# Approche et développement par composant

## CHAP 2. APPROCHE ET DEVELOPPEMENT PAR COMPOSANT

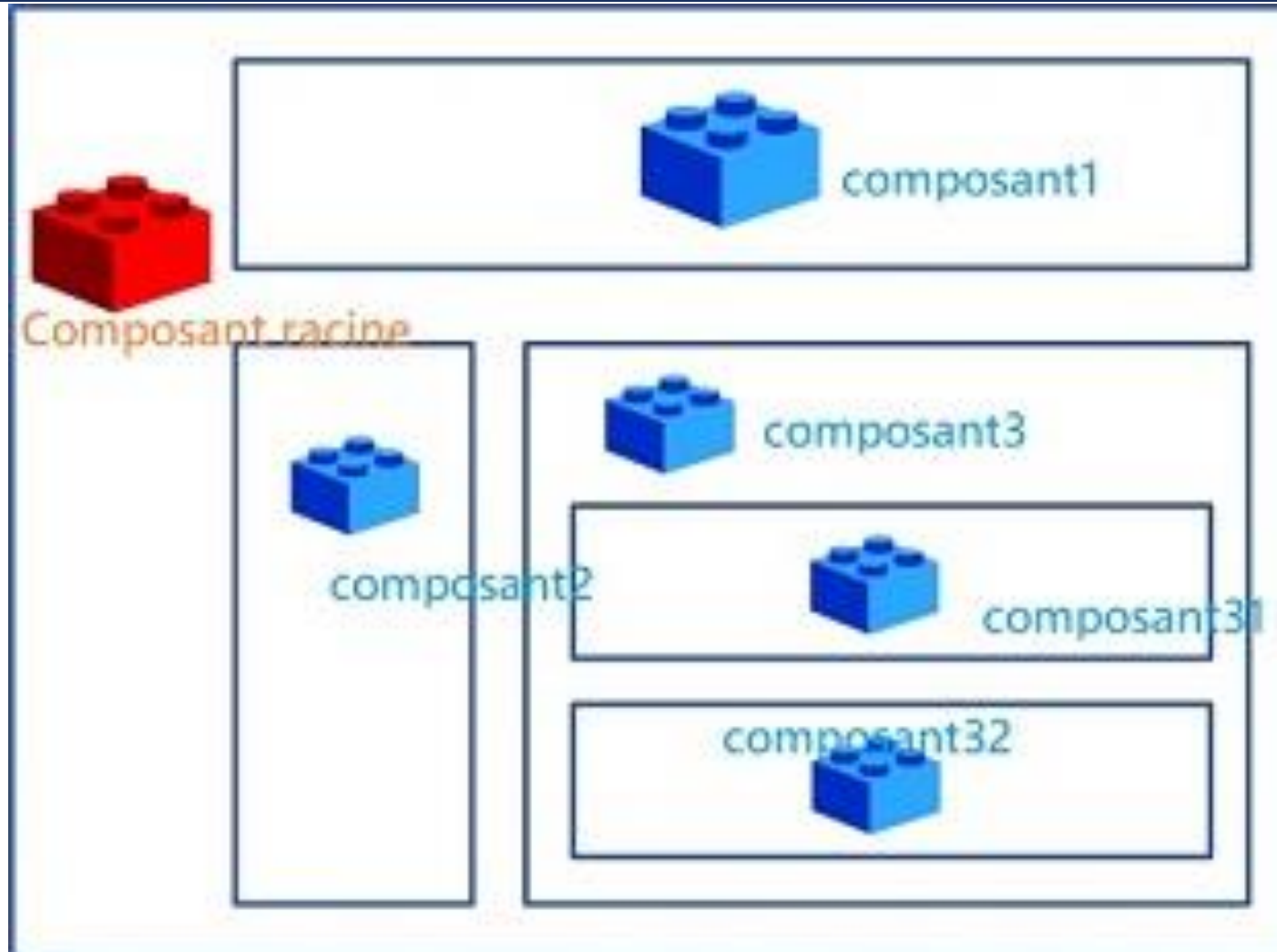


- Modélisation des composants
- Le Framework Angular: utilité et exploitation
- Architecture d'un projet Angular (architecture MVC)
- Les composants multiplateforme web : l'outil Stencil

## CHAP 2: MODÉLISATION DES COMPOSANTS

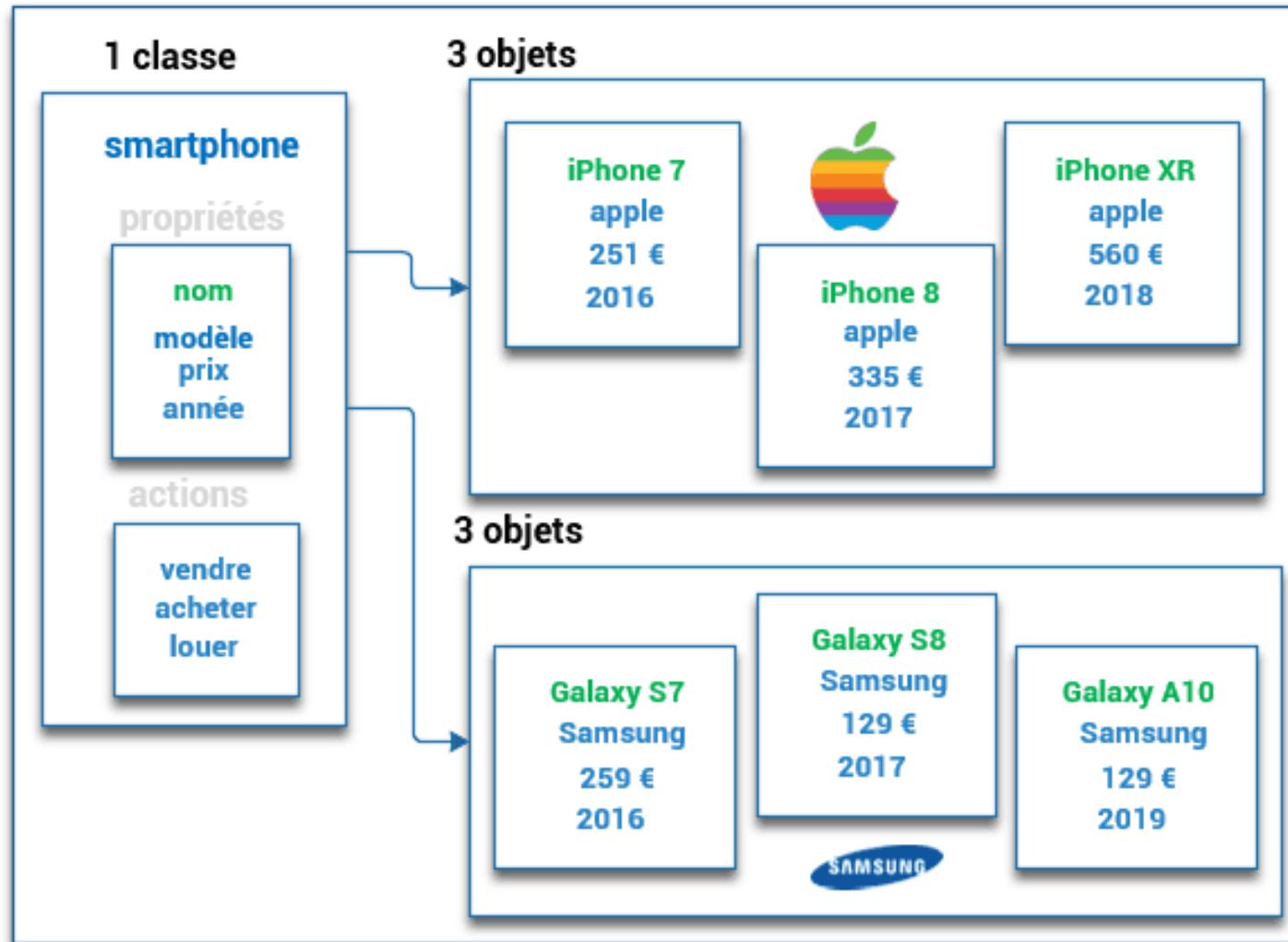
- Un composant est unité indépendante des autres composants;
- Un composant peut avoir une ou plusieurs fonctionnalités;
- Définir un composant revient à identifier son but afin de le rendre cohérent;

## CHAP 2: MODÉLISATION DES COMPOSANTS





## CHAP 2: MODÉLISATION DES COMPOSANTS



## CHAP 2: MODÉLISATION DES COMPOSANTS

- Un composant **ne doit pas être complexe**;
- Un ensemble de composants peuvent constituer un module;
- Un composant **est réutilisable** pour gagner en temps dans la phase de programmation;
- Un composant est paramétrable : des **propriétés** et des **méthodes** peuvent être définis.

## CHAP 2: MODÉLISATION DES COMPOSANTS

- Pour modéliser une architecture de composant, vous pouvez utiliser des outils standard (UML, Merise, etc.) ou alors utiliser votre propre méthode;
- Le but étant de rassembler tous les composants pour en faire un logiciel



## CHAP 2: LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION



- Angular est un Framework utilisé pour réaliser des applications web à la base;
- Il est également possible de réaliser des composants pouvant être réutilisable dans une application web « Angular »

# LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION

- Ce framework est également utilisé pour développer des applications hybrides (Mobile et Desktop)
- Angular s'appuie fortement sur le langage de programmation TypeScript.
- La 1<sup>er</sup> version du Framework est publiée en 2016 sous le nom **AngularJS** (le langage utilisé était JavaScript)

# LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION

- **Angular** est développé et maintenu par Google;
- La dernière version du Framework est 16.0 – *mai 2023*;
- Les applications Angular s'appuient sur une architecture MVC (Models – Views – Controller)

# LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION

- La création d'un projet Angular nécessite certains prérequis, à savoir :  
l'installation Node.js et NPM;
- Il existe plusieurs façons de créer un projet Angular :
  - Via l'outil [@angular/cli](#) (`npm install -g @angular/cli`)
  - Ou directement via la commande `npm`

# LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION

- En utilisant npm

```
$ npm init @angular myFirstApp
```

- En utilisant l'outil @angular/cli

```
$ ng new myFirstApp
```

# LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION

- Via npm (*à partir de la racine du projet créé*)

\$ npm start

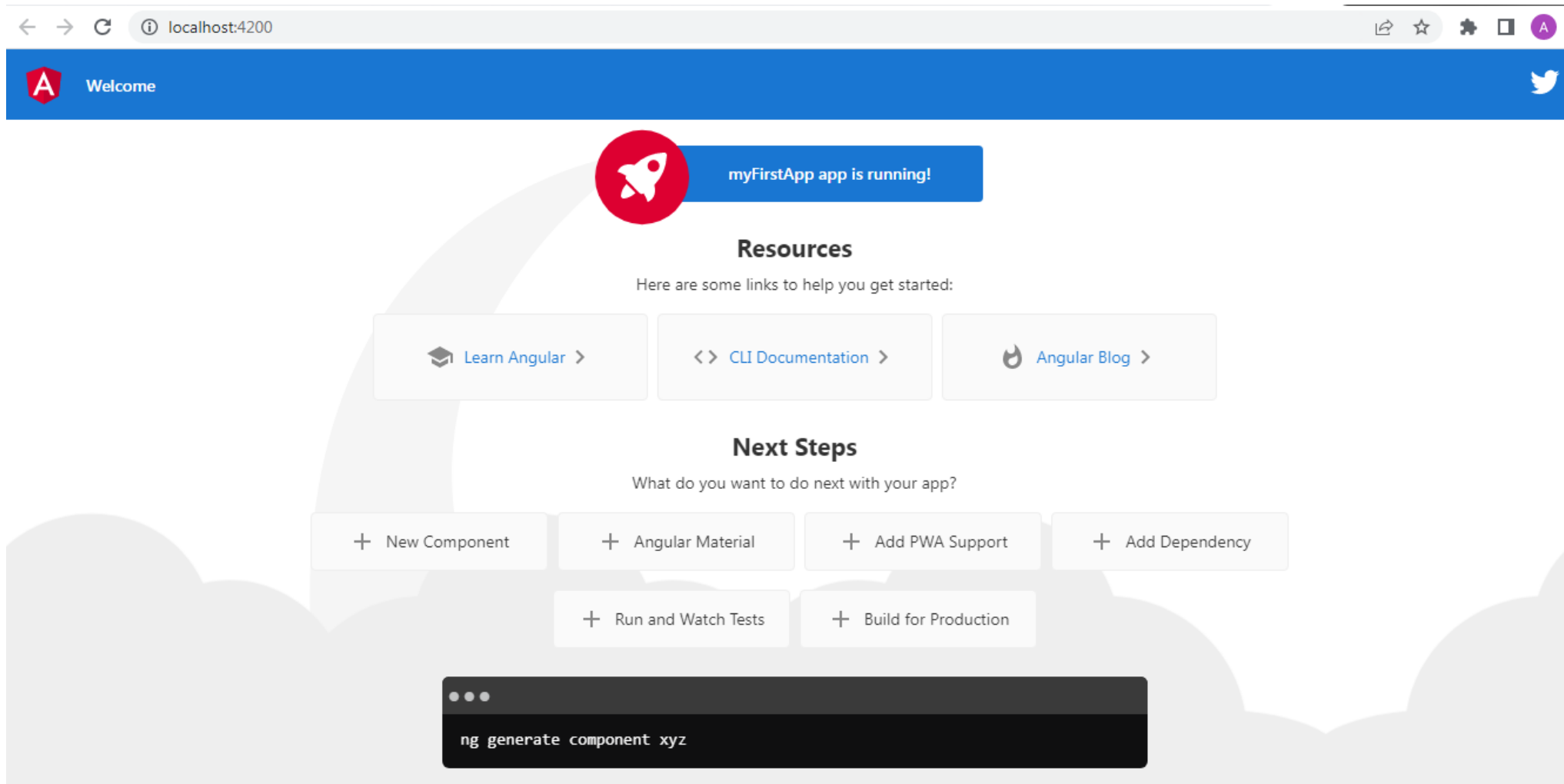
- Via l'outil angular/cli (*à partir de la racine du projet créé*)

\$ ng serve -o

- La commande pour générer l'app pour la production

\$ ng build --prod

# LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION





# LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION

- **Angular** s'appuie fortement sur un schéma de composants;
- Les composants peuvent être un bloc d'éléments, une bannière, un pied de page, un espace vidéo, un chat, en-tête d'une application, etc;
- Il est fortement recommandé d'utiliser **une implémentation par composants**

## LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION

- La commande **ng** est aussi utilisé pour générer le squelette des composants, services, directives et bien d'autres.
- `$ ng generate (ou ng g)`
- Cette commande permet de générer une **classe**, un **composant**, un **module**, une **directive**, un **service**, un **interceptor**;

# LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION

- La liste exhaustive de toutes ces commandes :  
<https://angular.io/cli/doc> .
- Architecture d'un projet Angular est idéal pour les projets complexe;

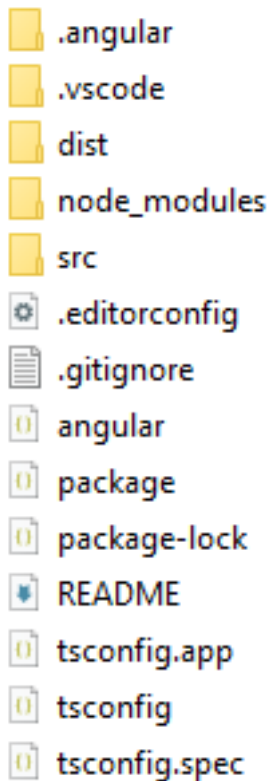
## LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION

- La commande ng est aussi utilisé pour générer le squelette des composants, services, directives et bien d'autres.
- \$ ng generate (ou ng g)
- Cette commande permet de générer une **classe**, un **composant**, un **module**, une **directive**, un **service**, un **interceptor**;

## LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION

- Pour exécuter un projet angular, saisissez la commande :  
`$ ng serve (ou ng serve -o)`
- La commande suivante est utilisé pour déployer un projet angular chez un hébergeur ou via un serveur  
`$ ng build (ou ng build --prod)`

# LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION



- **package.json** est le fichier de configuration des dépendances d'un projet Angular
- **tsconfig.json** est le fichier de configuration relatif à la compilation d'un projet angular (TypeScript)

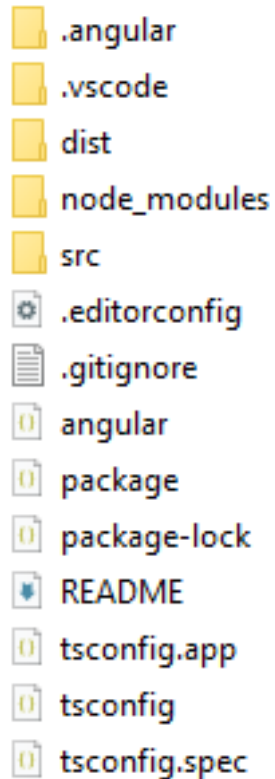
# LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION

- .angular
- .vscode
- dist
- node\_modules
- src
- .editorconfig
- .gitignore
- angular
- package
- package-lock
- README
- tsconfig.app
- tsconfig
- tsconfig.spec

**angular.json** est le fichier de configuration utilisé pour le démarrage de l'App ainsi que la définition des préférences (*configuration de la procédure de compilation*)



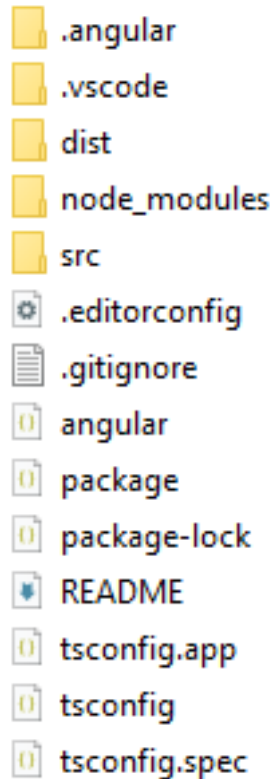
# LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION



.angular  
.vscode  
dist  
node\_modules  
src  
.editorconfig  
.gitignore  
angular  
package  
package-lock  
README  
tsconfig.app  
tsconfig  
tsconfig.spec

- **no\_modules/** : ce dossier est généré une fois que toutes les dépendances ont été installées via la commande *npm install*.
- **dist/** est généré lorsque la commande *ng build* a été exécutée.

# LE FRAMEWORK ANGULAR: UTILITÉ ET EXPLOITATION



.angular  
.vscode  
dist  
node\_modules  
src  
.editorconfig  
.gitignore  
angular  
package  
package-lock  
README  
tsconfig.app  
tsconfig  
tsconfig.spec

- **src/** : c'est dans ce dossier où il faudra implémenter toute la logique métier, les interfaces graphiques, les services, etc. d'un logiciel angular.

# PROCESSUS DE DEMARRAGE D'UN PROJET ANGULAR



- **index.html** : lors de l'exécution d'un projet angular, le fichier *index.html* est le premier fichier à être exécuté.
- **main.ts**: le compilateur va ensuite exécuter le fichier *main.ts*. Ce dernier va exécuter le module de démarrage de l'application angular.

# PROCESSUS DE DEMARRAGE D'UN PROJET ANGULAR



app.component



app.component



app.component.spe



app.component



app.module



app-routing.module

- **app.module.ts** : il définit la configuration du composant « *angular* » qui doit être démarré. Dans ce cas, il s'agit du fichier app.component.ts
- **app.component.ts**: contient la définition d'une classe « particulière » qui sera exécuté.

# PROCESSUS DE DEMARRAGE D'UN PROJET ANGULAR



app.component



app.component



app.component.spec



app.component



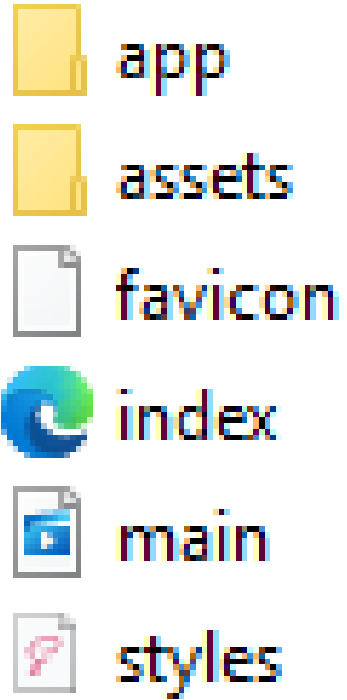
app.module



app-routing.module

- **app.component.ts**: contient également un décorateur **@Component** utilisé configurer la vue html et le style (mise en forme) de la vue.

## AUTRES RESSOURCES



- **app/** : tous les composants, services, les vues html, les feuilles de styles qui constituent le logiciel y seront stockés.
- **assets/**: c'est ici que l'ensemble des fichiers multimédia (*images, vidéos, police d'écriture, etc.*) seront stockés.

# NGMODULES

- `@NgModules` est une classe (**décorateur**).
- Il décrit le procédé sur comment compiler l'ensemble des composants;
- Il est utilisé pour identifier les *composants*, les *pipes*, les *directives*. Grâce à son attribut `exports`, il permet de rendre publique ces éléments;
- Il joue le rôle de « **chef d'orchestre** » dans l'exécution et la compilation d'un projet Angular



## NAVIGATION

- Angular dispose d'un service nommé « Router », il est utilisé pour faciliter navigation entre les vues d'une application.
- Il est aussi utilisé pour passer des partager des données d'une vue à une autre.
- Le fichier principale pour configurer les routes « chemin » se nomme « **app-routing.module.ts** ».

# NAVIGATION

- `import { RouterModule, Routes } from '@angular/router';`
- « **RouterModule** » est utilisé pour ajouter des services pour faciliter la navigation entre les vues.
- La méthode **forRoot()** du module « *RouterModule* » permet de configurer et d'ajouter un module comprenant un schéma des routes définies.

## NAVIGATION

- `import { RouterModule, Routes } from '@angular/router';`
- « **Routes** » est un service Angular permettant de configurer les routes d'une application.
- La méthode **forRoot()** du module « *RouterModule* » permet de configurer et d'ajouter un module comprenant un schéma des routes définies.

## NAVIGATION: PREMIER COMPOSANT

- Créons un composant nommé « étudiants ».

```
$ ng g component components/etudiants
```

- Créer la route associé au composant généré

```
const routes: Routes = [  
  { path: '', component: EtudiantsComponent },  
];
```

## NAVIGATION: PREMIER COMPOSANT

- Les **propriétés** et les **méthodes** (*publique*) d'un composant peuvent être appelés sur la vue html ou par un autre composant (classe).
- Nous utilisons `{{ }}` pour afficher leur valeur
- Ex: `{{nom_etudiant}}`

## NAVIGATION: PREMIER COMPOSANT (DIRECTIVE)

- Une **directive** est utilisé pour changer le comportement d'un élément (*balise HTML, composant*) du DOM.
- Sur la vue html, la directive **\*ngIf** permet définir une condition sur un élément html.
- La directive **\*ngFor** quand elle permet d'appliquer une boucle sur élément html

## NAVIGATION: PREMIER COMPOSANT (DIRECTIVE)

### Exemples :

- `<h3 *ngIf="is_ing4">Je suis un étudiant  
ING4 ISI</h3>`
- `<h3 *ngIf="is_ing1 || is_ing2">Je suis un  
étudiant en tronc commun</h3>`



## NAVIGATION: PREMIER COMPOSANT (DIRECTIVE)

Exemple :

```
<ul>
```

```
  <li *ngFor="let item of etudiants">
```

```
    {{item}}
```

```
  </li>
```

```
</ul>
```

## NAVIGATION

- Pour naviguer d'une vue à une autre nous utiliserons le paramètre **routerLink**. Via une vue html;
- Ou encore le service **Router** Angular depuis un fichier TypeScript (TS);
- De plus, il faudra insérer un nouveau chemin qui mène vers le composant appelé;

# NAVIGATION

- Le fichier de définition des routes :

```
const routes: Routes = [  
  { path: '', component: EtudiantsComponent },  
  { path: 'etudiant/:matricule', component:  
DetailEtudiantComponent },  
];
```

## NAVIGATION

### ■ Vue (etudiants)

```
<li *ngFor="let item of etudiants"  
[routerLink]="['/etudiant', '2021i0200']" >
```

### ■ Vue (détails étudiant)

```
ma = this.router.snapshot.paramMap.get("matricule");
```

## ARCHITECTURE MVC

- L'approche **MVC (Models Views and Controller)** est utilisé par Angular dans le but de bien organiser l'architecture logiciel ;
- Cette approche est idéal pour les projets robustes
- Il est facile à prendre à main si et seulement si cette approche est **bien implémentée**;

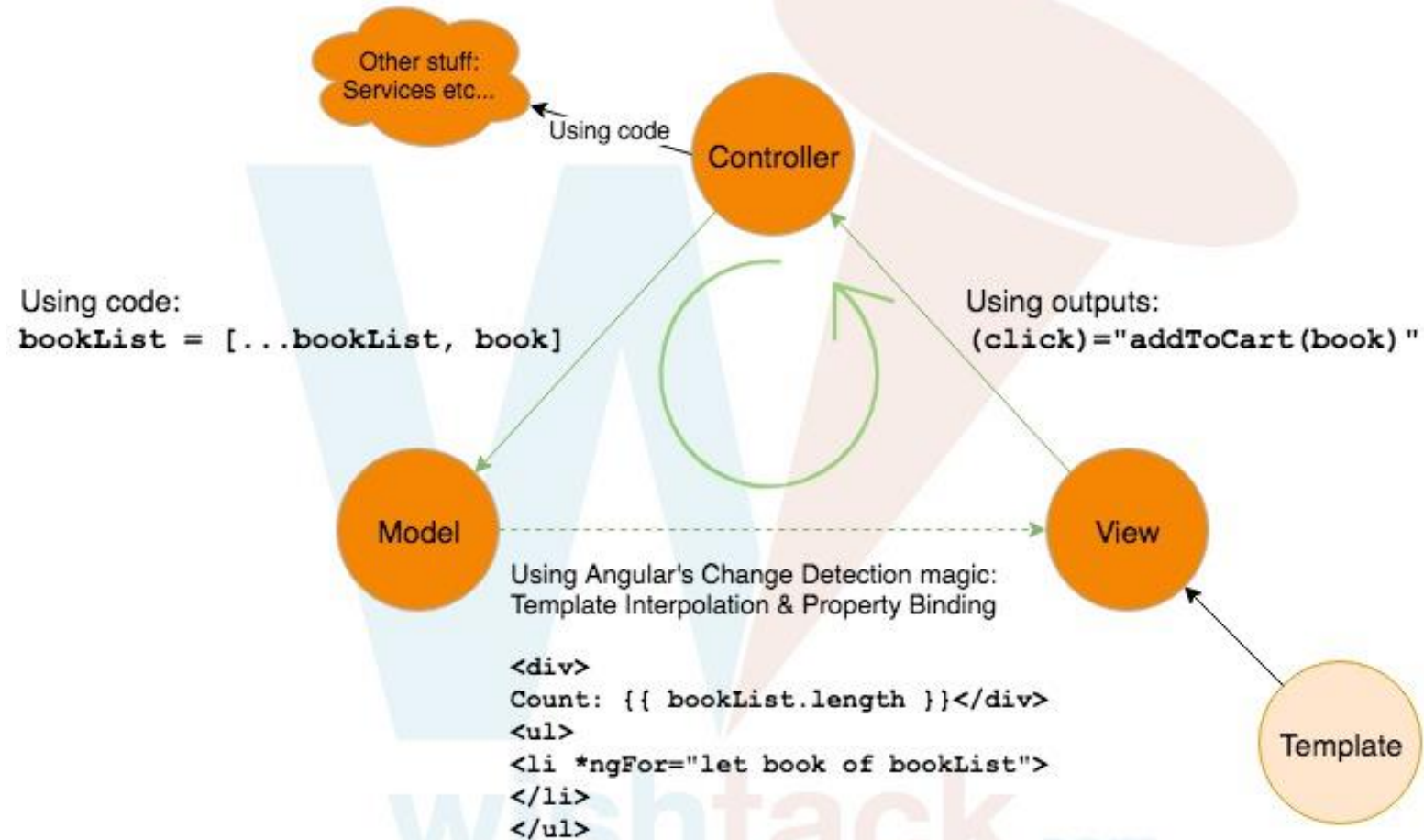
# ARCHITECTURE MVC

- Le « **Controller** » et le « **Model** » sont représentés par l'instance de la classe TypeScript;
- La « **View** » quant à elle est générée par le DOM sur la base d'un template;
- Les actions définies dans le « **Controller** » peuvent être déclenchés par la « **View** »

# ARCHITECTURE MVC

- Angular détecte les changements et met à jour la « **View** »;
- Le « **Controller** » met à jour l'état du « **Model** »;
- Les actions définies dans le « **Controller** » peuvent être déclenchés par la « **View** »

# ARCHITECTURE MVC





## ARCHITECTURE MVC (VIOLATION)

- L'architecture MVC **n'est plus respecté** lorsqu'une « **Vue** » **dépend fortement** de son « **Controller** » ;
- Ceci nous ramène à une approche traditionnelle où le « **Controller** » et la « **Vue** » sont liées
- Toutefois, il n'est pas exclu que cette violation soit actée

# ARCHITECTURE MVC

- L'objectif d'une architecture MVC est aussi de séparer les rôles des composants:
  - Les conteneurs de composants
  - Les composants de présentation

# ARCHITECTURE MVC

- **Les conteneurs de composants** ont pour but de fournir les données à présenter
- **Les composants de présentation** se chargent de définir un rendu des données à visualiser

## ARCHITECTURE MVC

- Exemple : voici le contenu d'un composant.

```
<app-book-preview *ngFor="let student of studentsList">  
</app-book-preview>
```

# ARCHITECTURE MVC: COMMUNICATION ENTRE LES COMPOSANTS

- Pour étendre l'indépendance d'un composant, il existe des paramètres et événements qui peuvent être définis
- Grâce à ces événements, la communication entre composants devient pertinente sans briser l'architecture de votre logiciel

## ARCHITECTURE MVC: COMMUNICATION ENTRE LES COMPOSANTS

- Pour transmettre des données à un composant nous utiliserons le concept des « **attributs** » à appliquer sur une balise

```
<app-etudiant [etudiant]="etudiantList[0]">
```

```
</app-etudiant>
```

## ARCHITECTURE MVC: COMMUNICATION ENTRE LES COMPOSANTS

- D'où la notion de **composant parent** et **composant enfant**;
- Le composant parent joue le rôle de **conteneur** des composants enfants.
- Ceci peut produire un schéma sous forme d'arbre où un composant peut être à la fois parent et/ou enfant.

## ARCHITECTURE MVC: COMMUNICATION ENTRE LES COMPOSANTS

- Lors de la définition d'un composant, il s'agira de définir une propriété spécial à qui il faudra associer un décorateur;
- Le décorateur **@Input()** est utilisé pour insérer des données à un composant;
- Le **composant parent** envoie des données au **composant enfant**



## ARCHITECTURE MVC: COMMUNICATION ENTRE LES COMPOSANTS

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

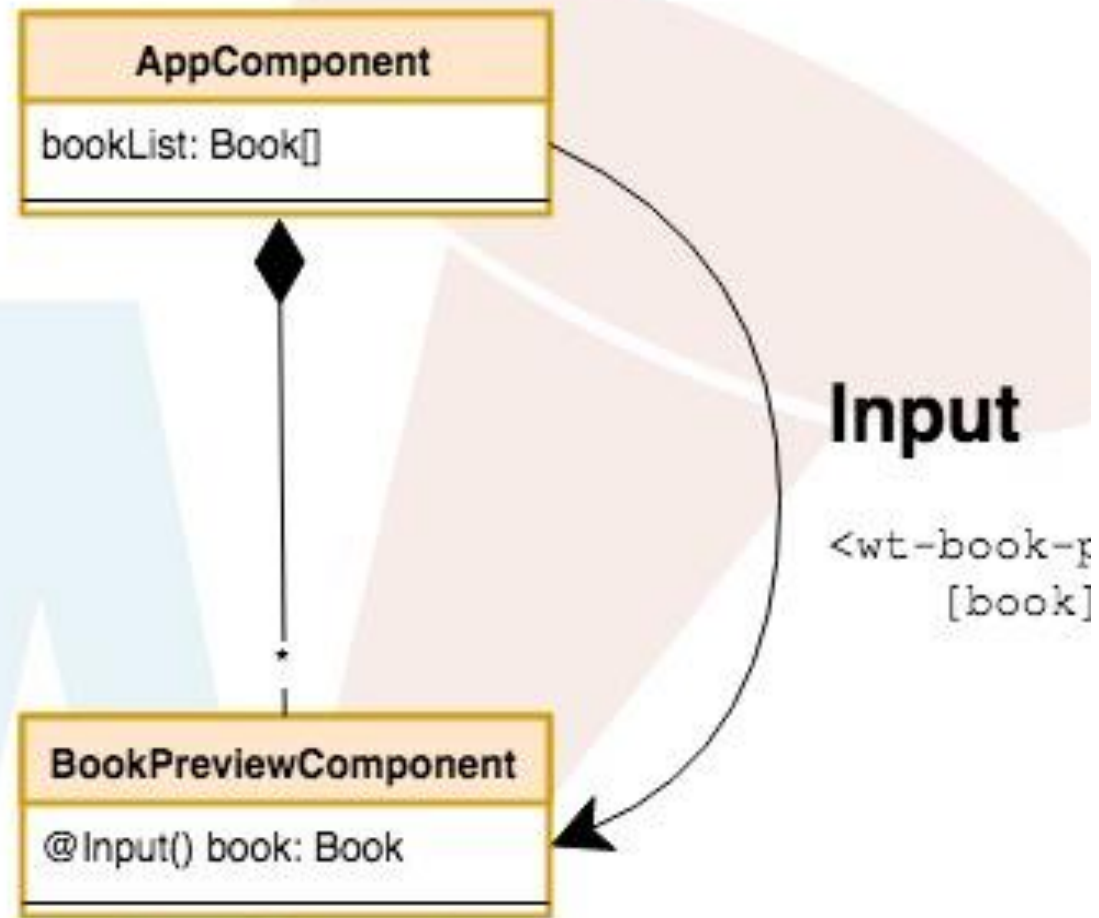
```
...
```

```
export class EtudiantComponent {
```

```
    @Input() etudiant: Etudiant;
```

```
}
```

# ARCHITECTURE MVC



## ARCHITECTURE MVC: COMMUNICATION ENTRE LES COMPOSANTS

- De la même manière, un composant (**composant enfant**) est capable de fournir des données à un composant qui l'a appelé ( **composant parent**);
- Le décorateur **@Output()** est utilisé pour envoyer des données à un composant parent;

# ARCHITECTURE MVC: COMMUNICATION ENTRE LES COMPOSANTS

## ■ composant parent

```
<app-etudiant (aged)="onBirthdate($event)">  
</app-etudiant>
```

## ARCHITECTURE MVC: COMMUNICATION ENTRE LES COMPOSANTS

- La méthode **onBirthdate()** est défini dans le composant parent.
- « **\$event** » représente la valeur émise par le composant enfant
- L'événement « **aged** » est une propriété (décorateur) définie dans le composant enfant

# ARCHITECTURE MVC: COMMUNICATION ENTRE LES COMPOSANTS

## ■ composant enfant (TS)

```
@Output() aged = new EventEmitter<number>();
```

```
computeAge() {  
    const value: number = 22;  
    this.aged.emit(value);  
}
```

# ARCHITECTURE MVC: COMMUNICATION ENTRE LES COMPOSANTS

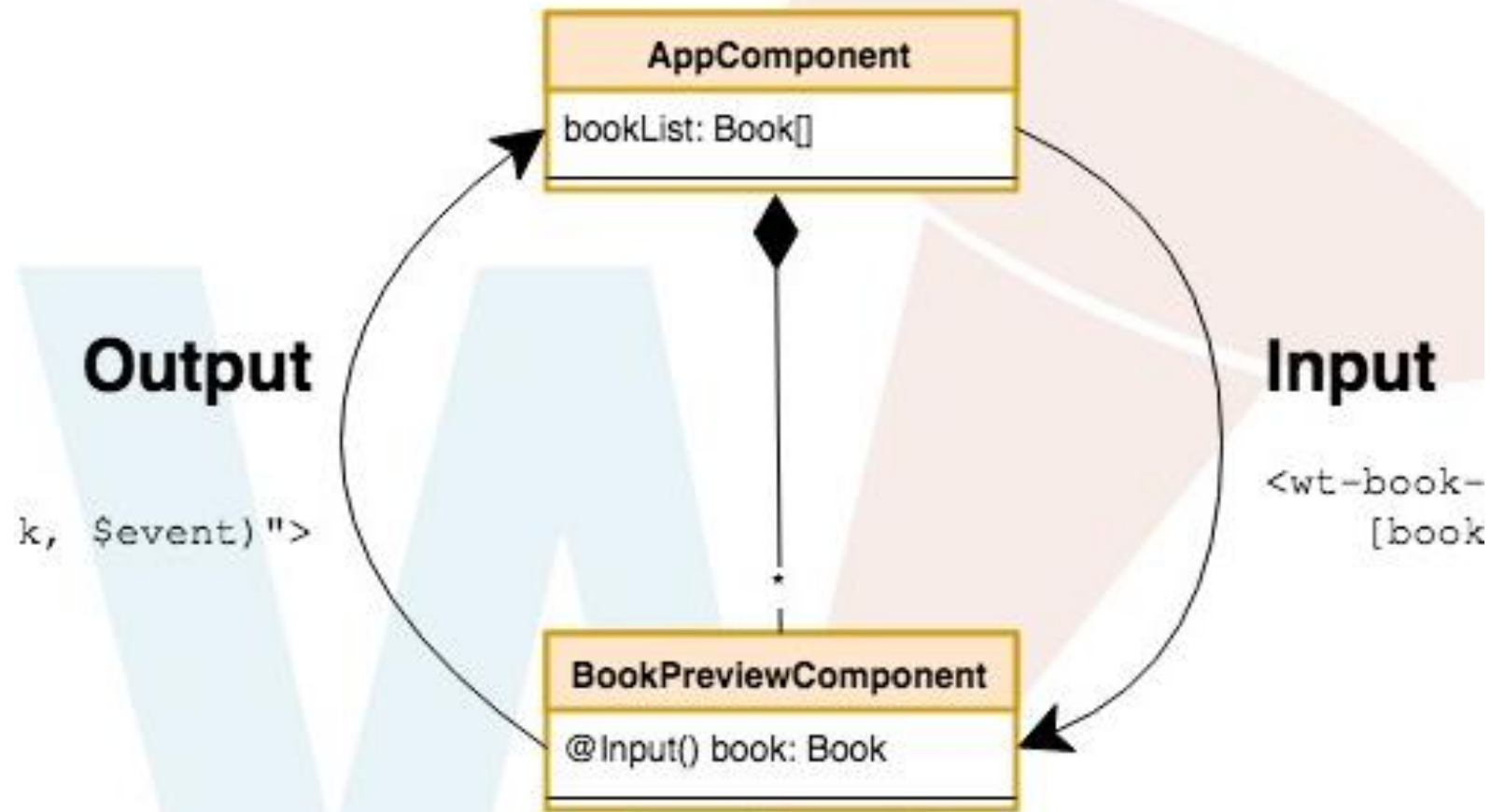
## ■ composant enfant (Html)

...

```
<button  
type="button" (click)="computeAged()">  
    Calcul age  
</button>
```

...

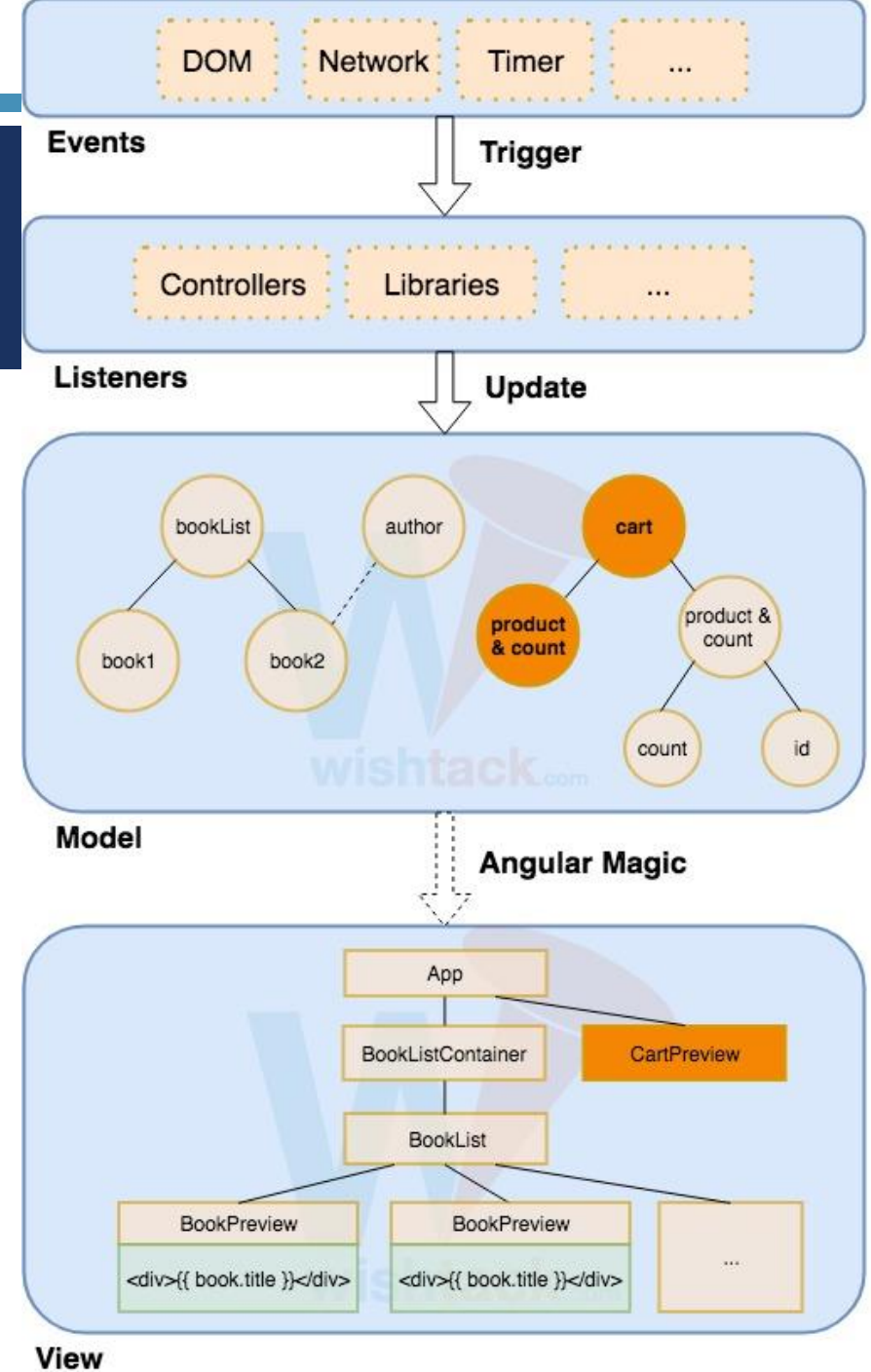
# ARCHITECTURE MVC





# ARCHITECTURE MVC

En somme nous avons une architecture. Nous avons une architecture MVC détaillée qui représente l'architecture profonde d'un logiciel



# STENCIL

- **Stencil (Stencil JS)** est une bibliothèque utilisée pour générer des composants web;
- Un composant web (*pas encore spécifié par W3C*) est un ensemble de technologie web permettant de créer des éléments personnalisés;

Ex: `<alima-login [username]=" [password]="></alima-login>`

# STENCIL

- **Stencil** est capable de construire des composants web réutilisable, robuste pouvant être intégrer sur n'importe qu'elle plateforme Web (Framework ou pas);
- **Stencil** est un outil créé par l'équipe Ionic.
- <https://stenciljs.com/>

# STENCIL

- La commande suivante permet d'initier un projet Stencil;

```
$ npm init stencil
```

- Le script suivant consiste à démarrer le développement des composants:

```
$ cd isi-library
```

```
$ npm install
```

```
$ npm start
```

# STENCIL

- La commande suivante permet générer les composants web réutilisable:

```
$ npm run build
```

- Pour créer un nouveau composant:

```
$ stencil generate my-new-component
```

# STENCIL

Pour intégrer un  
composant web  
dans une  
application web  
(non framework):

```
<html>
  <head>
    <script src="https://.../dist/manage-
user.js"></script>
  </head>
  <body>
    <my-component></my-component>
  </body>
</html>
```



# THANK YOU

ARTHURPESSA@INSTITUTSAIN  
TJEAN.ORG