Angular 2

Alexandre.Demeure@univ-grenoble-alpes.fr

2017

Angular 2 : Le contexte

AngularJS (v1) développé chez Google en 2009

- Projet annexe à l'origine (20% temps libre)
  - Pour les designers web (pas vraiment les devs)
  - Interagir facilement avec le back et front-end
- Projet Google Feedback (dev en GWT)
  - 6 mois de développements
  - 17.000 lignes de codes
  - Difficile à tester, à faire évoluer
- Un des développeur fait un pari
  - Google Feedback en 2 semaines avec Angular
  - Pari perdu : 3 semaines. 17.000 => 1500 lignes
  - Plus clair, testable, plus facile à faire évoluer
  - Utilisation en interne puis en externe

Angular 2 : Le contexte

AngularJS (v1)

- Philosophie : Que serait HTML si il avait été conçu pour décrire des applications ?
- Cible les Single Page Application
- Double way data binding, testabilité, injection de dépendances, ...
- Communauté très importante pour AngularJS
  - Cf google trends
  - Cf github
- Nombreux projets réalisés
  - https://www.madewithangular.com/

Angular 2 : Le contexte

Angular 2 développé PAR Google

- Annoncé ~2015
- Problèmes de performances, passage à l'échelle
- EcmaScript -> TypeScript
- Standard des web components
- Angular 2.0.0 produit ~15/09/2016

Angular 2

Un framework développé par Google

- Utilisation conseillé avec TypeScript mais possible en ES ou Dart
- Programmation des IHM à l'aide de composants
- Centré sur HTML, étend HTML avec les composants
- Rendu via le client ou sur le serveur
- Utilisable pour des application natives (via IONIC)
- Testablitité

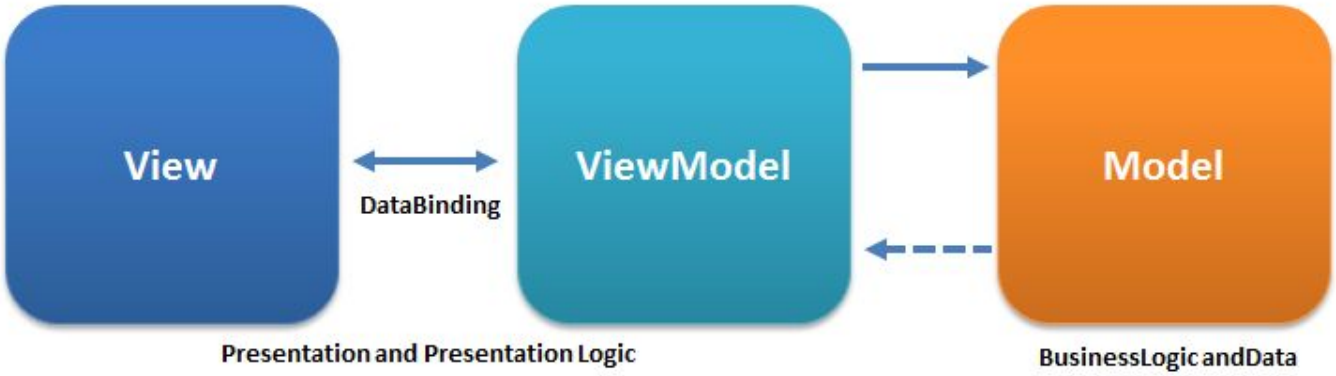
Un framework très riche, mais nous verrons seulement :

- Les composants
- Les directives
- La data-binding
- Les services
- L'injection de dépendances

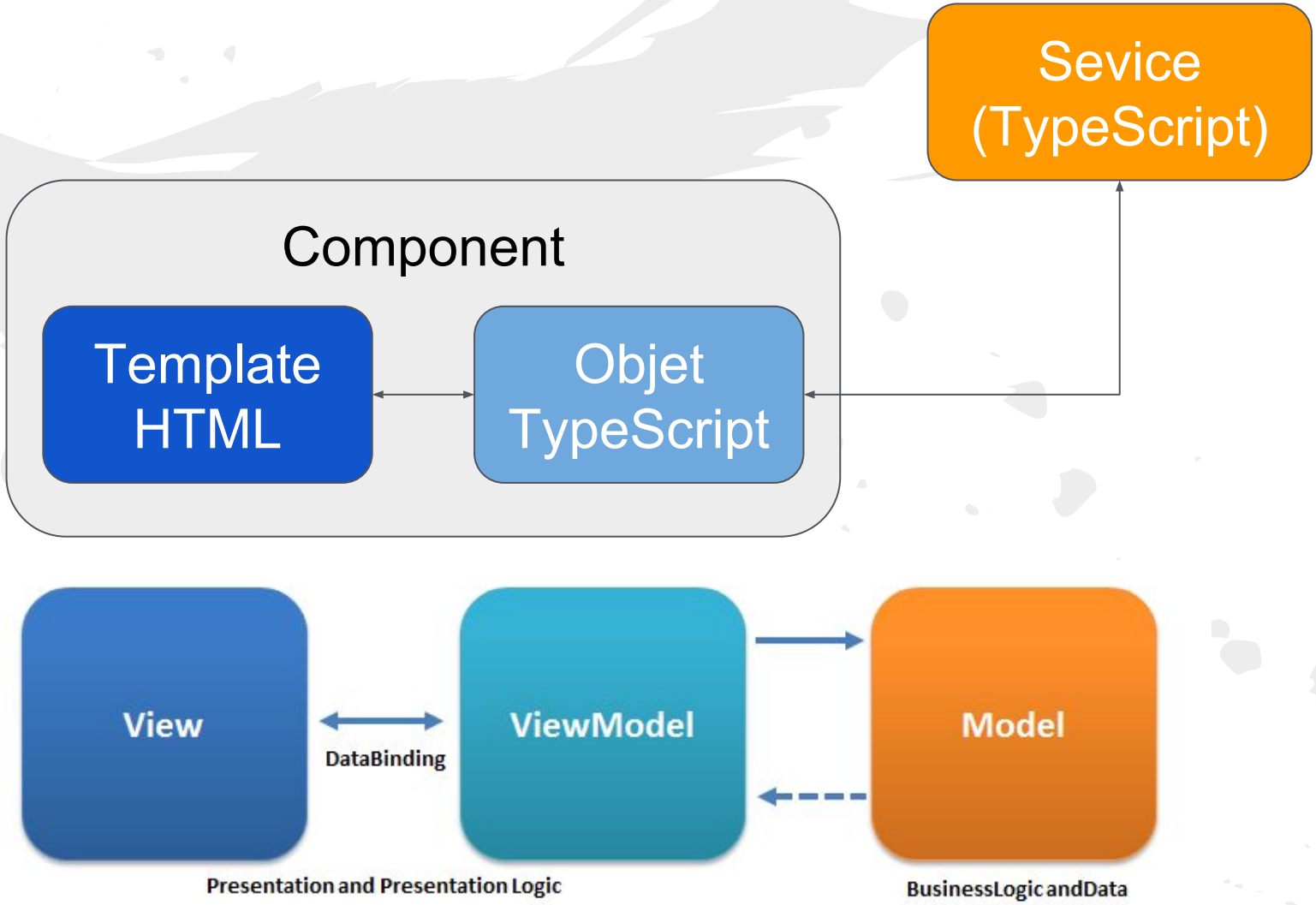
Angular 2

Les constituants principaux d'un composants :

Une chaîne de caractères qui représente le template du composant	Une classe TypeScript qui gère la logique du composant	Parfois des services qui permettent d'agir sur le noyau



Angular 2


Sevice
(TypeScript)
Component
Template
HTML
Objet
TypeScript
View
DataBinding
ViewModel
Model
Presentation and Presentation Logic
BusinessLogic andData

Angular 2 : Module

Exemple de la liste de choses à faire

- Définition d'un module contenant
  - Décoration @ngModule
  - Attention ngModule ≠ module Typescript
  - Les déclarations des composants (ListeChoses, ItemChose)
  - Les déclaration des services (ListeChosesService)

```
@NgModule({
    imports      : [ CommonModule, FormsModule, HttpModule ],
    exports      : [ ListeChoses ],
    declarations: [ ListeChoses, ItemChose ],
    providers    : [ ListeChosesService ]
})
export class ListeChosesModule { }
```

Angular 2 : Module

```
@NgModule({
    imports      : [ CommonModule, FormsModule, HttpModule ],
    exports      : [ ListeChoses ],
    declarations: [ ListeChoses, ItemChose ],
    providers    : [ ListeChosesService ]
})
export class ListeChosesModule { }
```

Sevice (TypeScript)

Angular 2 : Service

Exemple de la liste de choses à faire

- Définition d'un service
  - Lien avec le noyau fonctionnel
  - Une simple classe
  - Décorateur @Injectable
    (pour le cas d'une dépendance à d'autre services
    Ex: service Http)

```
@Injectable()
export class ListeChosesService {
    http     : Http;
    constructor(http : Http) {}
    getData () : Observable<NF.ListeChoses> {...}
}
```

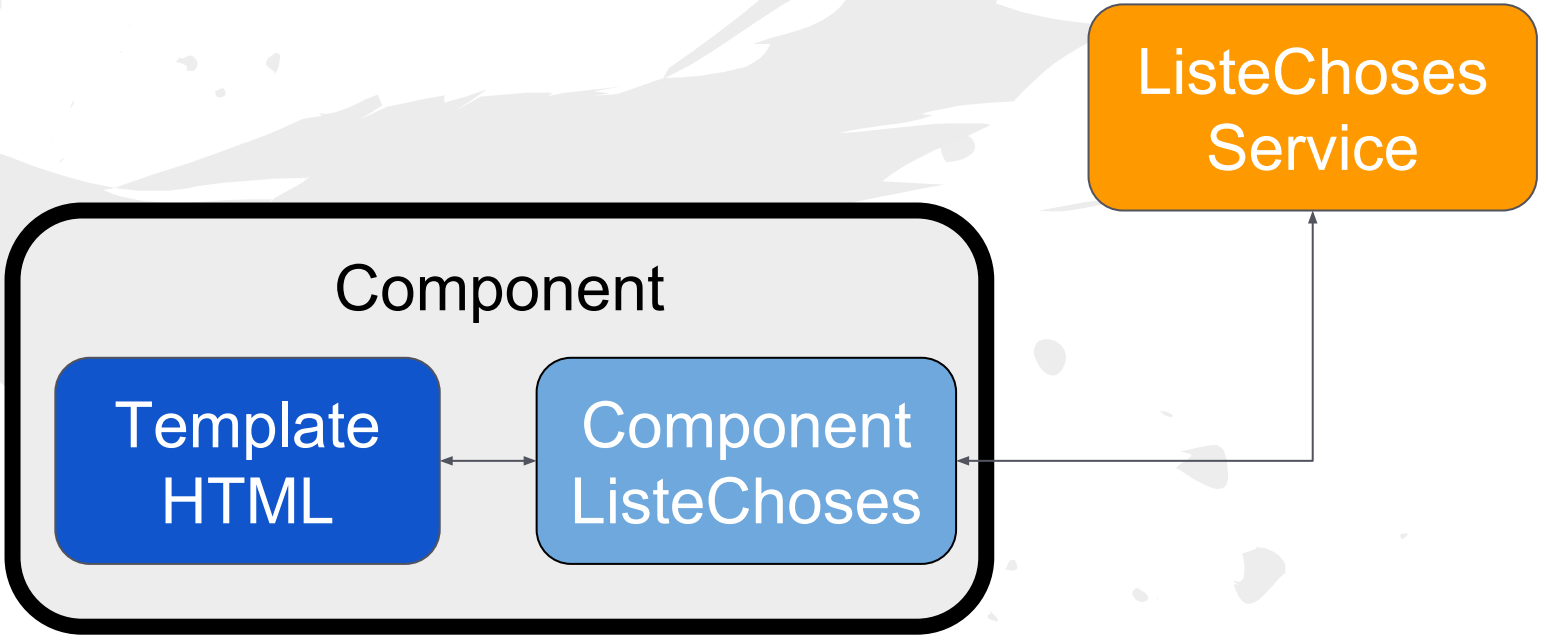
Sevice (TypeScript)

Angular 2 : Service

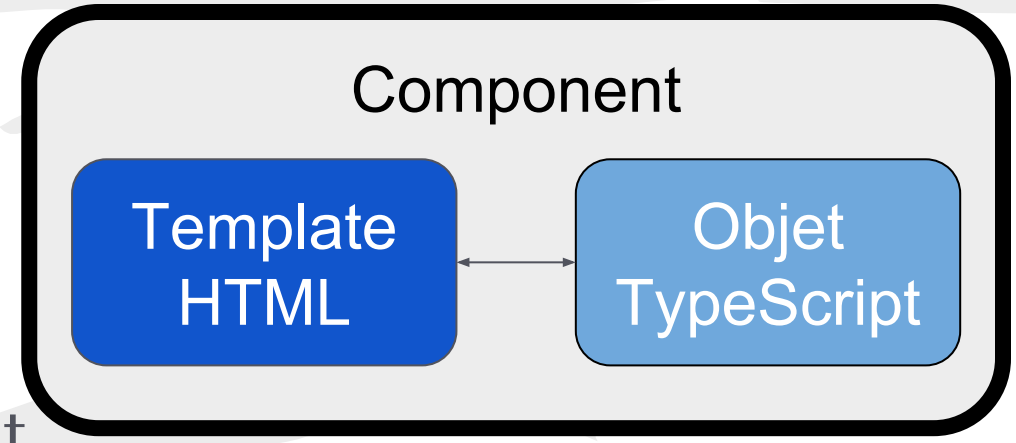
```
@Injectable()
export class ListeChosesService {
    http     : Http;
    constructor(http : Http) {}
    getData () : Observable<NF.ListeChoses> {...}
}
```

Angular 2

```
@Injectable()
export class ListeChosesService {
    http     : Http;
    constructor(http : Http) {}
    getData () : Observable<NF.ListeChoses> {...}
}
```



Composant



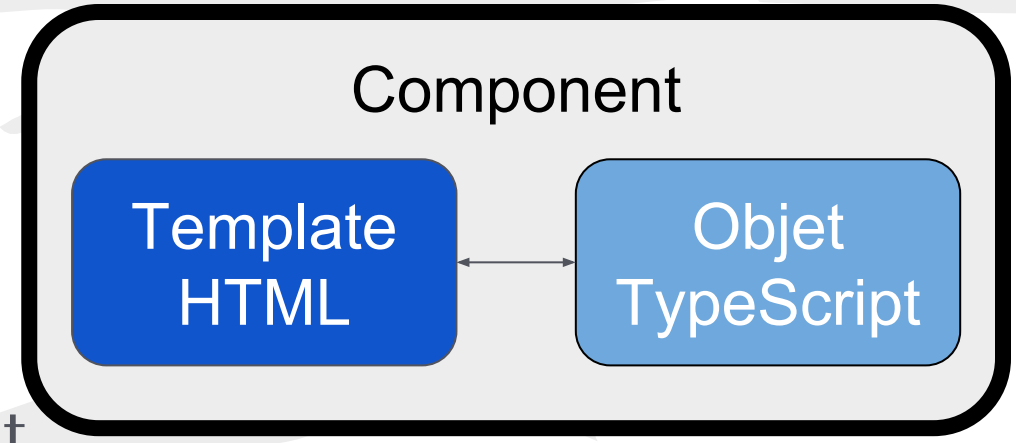
Déclaration d'un composant

- Un composant va correspondre à une balise <liste-choses></liste-choses>
- Décorateur **@Component**
- Sélecteur CSS exprimant les balises représentants le composant
- Template HTML à insérer dans ces balises

```
const htmlTemplate : string = `
...
`;
```

```
@Component({
        selector  : "liste-choses",
        template : htmlTemplate
})
export class CompListeChoses {

}
```

Composant



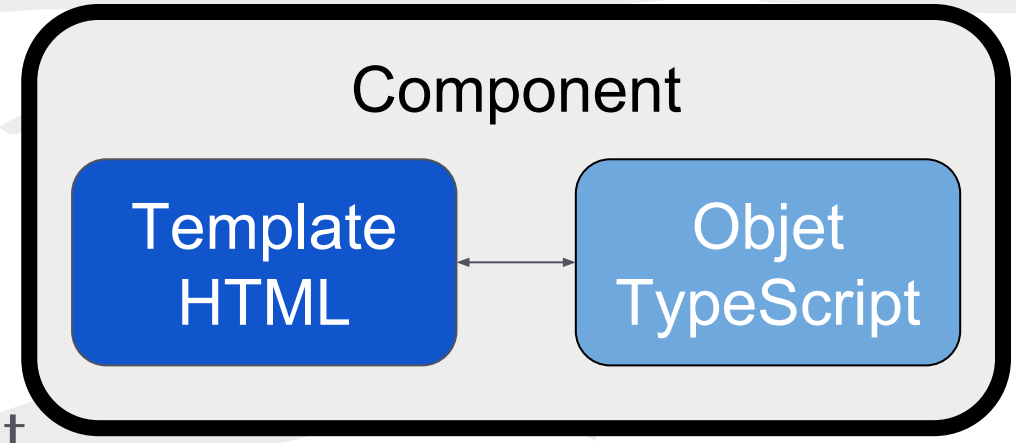
Déclaration d'un composant

- Un composant va correspondre à une balise <liste-choses></liste-choses>
- Décorateur **@Component**
- Sélecteur CSS exprimant les balises représentants le composant
- Template HTML à insérer dans ces balises

```
const htmlTemplate : string = `
...
`;
```

```
@Component({
        selector  : "liste-choses",
        template : htmlTemplate
})
export class CompListeChoses {
        nf : ListeChoses;
        constructor (S: ListeChosesService) {
            S.getData().then(
                nf => this.nf = nf  );
        }
        Ajouter(texte: string) {
                this.nf.Ajouter(texte); }
}
```

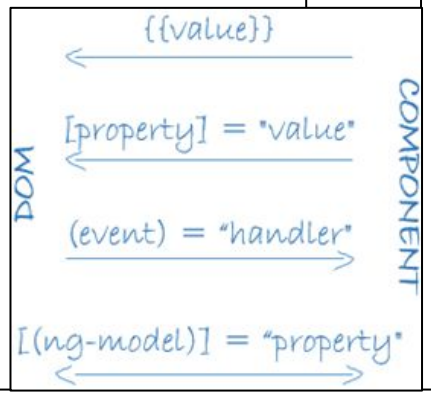
Composant



Déclaration d'un composant

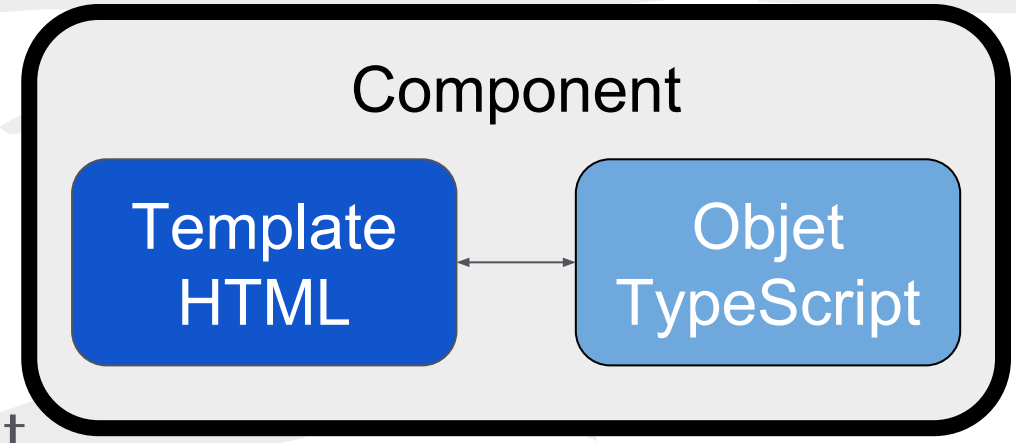
- Un composant va correspondre à une balise <liste-choses></liste-choses>
- Décorateur **@Component**
- Sélecteur CSS exprimant les balises représentants le composant
- Template HTML à insérer dans ces balises

```
const htmlTemplate : string = `
...
`;
```




```
@Component({
        selector  : "liste-choses",
        template : htmlTemplate
})
export class CompListeChoses {
        nf : ListeChoses;
        constructor (S: ListeChosesService) {
            S.getData().then(
                nf => this.nf = nf  );
        }
        Ajouter(texte: string) {
                this.nf.Ajouter(texte); }
}
```

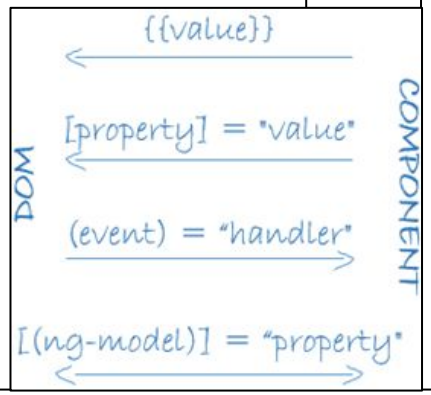
Composant



Déclaration d'un composant

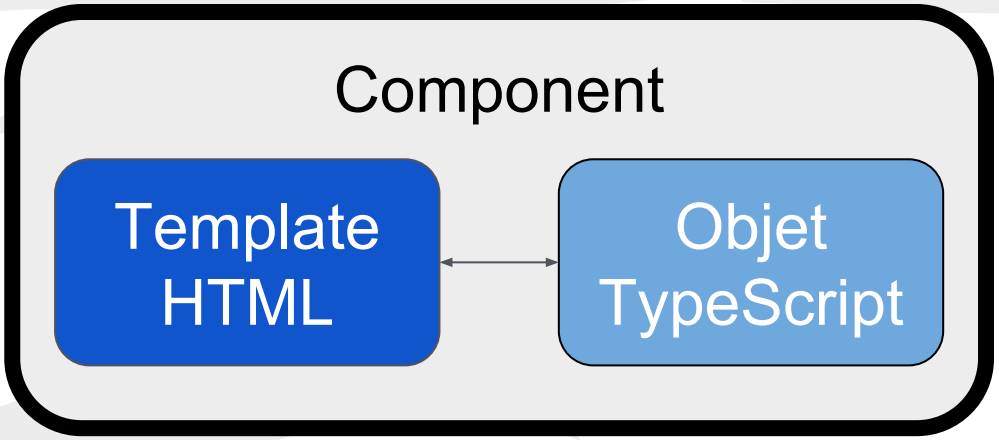
- Un composant va correspondre à une balise <liste-choses></liste-choses>
- Décorateur **@Component**
- Sélecteur CSS exprimant les balises représentants le composant
- Template HTML à insérer dans ces balises

```
const htmlTemplate : string = `
<form (ngSubmit)="Ajouter(newText.value)">
        <input type="text"  #newText  />
</form>
`;
```

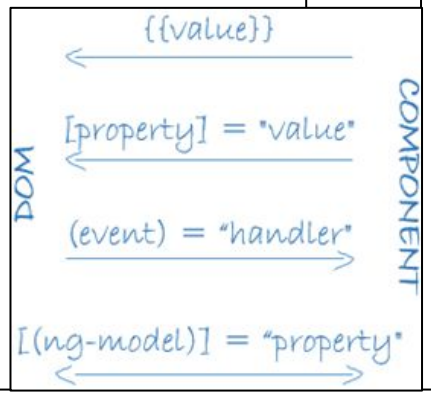



```
@Component({
        selector  : "liste-choses",
        template : htmlTemplate
})
export class CompListeChoses {
        nf : ListeChoses;
        constructor (S: ListeChosesService) {
            S.getData().then(
                nf => this.nf = nf  );
        }
        Ajouter(texte: string) {
                this.nf.Ajouter(texte); }
}
```

Composant

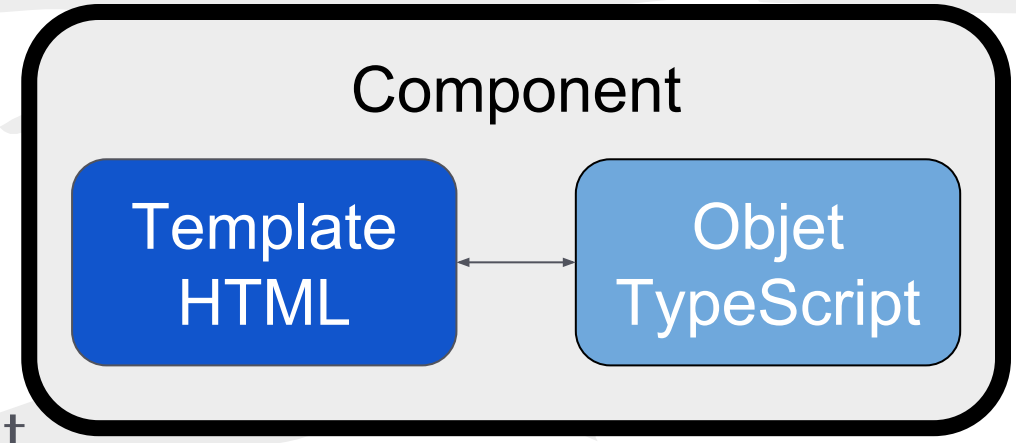



```
const htmlTemplate : string = `
<form (ngSubmit)="Ajouter(newText.value)">
        <input type="text"  #newText  />
</form>
`;
```




```
@Component({
        selector  : "liste-choses",
        template : htmlTemplate
})
export class CompListeChoses {
        nf : ListeChoses;
        constructor (S: ListeChosesService) {
            S.getData().then(
                nf => this.nf = nf  );
        }
        Ajouter(texte: string) {
                this.nf.Ajouter(texte); }
}
```

Composant



Déclaration d'un composant

- Un composant va correspondre à une balise <liste-choses></liste-choses>
- Décorateur **@Component**
- Sélecteur CSS exprimant les balises représentants le composant
- Template HTML à insérer dans ces balises

```
const htmlTemplate : string = `
<form (ngSubmit)="Ajouter(newText.value)">
        <input type="text"  #newText  />
</form>
<ul>
        <li *ngFor="let c of nf.choses">
                <item-chose [nf]="c"></item-chose>
        </li>
</ul>
{{nf.choses.length}} items
`;
```

```
@Component({
        selector  : "liste-choses",
        template : htmlTemplate
})
export class CompListeChoses {
        nf : ListeChoses;
        constructor (S: ListeChosesService) {
            S.getData().then(
                nf => this.nf = nf  );
        }
        Ajouter(texte: string) {
                this.nf.Ajouter(texte); }
}
```

Faisons le point

- Un module Angular ?
- Un service dans Angular ?
- Injection de dépendance ?

Faisons le point

- Un composant dans Angular ?

- Data-binding ?

Exercice: MVP vs MVVM

Angular 2 : Encore les composants

Templates et Composants

- Où sont définis les directives ngSubmit, click, *ngFor, ... ?

```
const htmlTemplate : string = `
<form (ngSubmit)="Ajouter(newText.value)">
        <input type="text"  #newText  />
</form>
<ul>
        <li *ngFor="let c of nf.choses">
                <item-chose [nf]="c"></item-chose>
        </li>
</ul>
{{nf.choses.length}} items
`;
```

Templates et Composants

- Où sont définis les directives ngSubmit, click, *ngFor, ... ?

```
@NgModule({
    imports      : [ CommonModule, FormsModule, HttpModule ],
    exports      : [ ListeChoses ],
    declarations: [ ListeChoses, ItemChose ],
    providers    : [ ListeChosesService ]
})
export class ListeChosesModule { }
```

```
const htmlTemplate : string = `
<form (ngSubmit)="Ajouter(newText.value)">
        <input type="text"  #newText  />
</form>
<ul>
        <li *ngFor="let c of nf.choses">
                <item-chose [nf]="c"></item-chose>
        </li>
</ul>
{{nf.choses.length}} items
`;
```

Templates et Composants

- Dans quel contexte le code TypeScript est il exécuté ?

```
const htmlTemplate : string = `
<form (ngSubmit)="Ajouter(newText.value)">
        <input type="text"  #newText  />
</form>
<ul>
        <li *ngFor="let c of nf.choses">
                <item-chose [nf]="c"></item-chose>
        </li>
</ul>
{{nf.choses.length}} items
`;
```

Templates et Composants

- Dans quel contexte le code TypeScript est il exécuté ?

```
const htmlTemplate : string = `
<form (ngSubmit)="Ajouter(newText.value)">
        <input type="text"  #newText  />
</form>
<ul>
        <li *ngFor="let c of nf.choses">
                <item-chose [nf]="c"></item-chose>
        </li>
</ul>
{{nf.choses.length}} items
`;
```

```
@Component({
        selector  : "liste-choses",
        template : htmlTemplate
})
export class CompListeChoses {
        nf : ListeChoses;
        constructor (S: ListeChosesService) {
            S.getData().then(
                nf => this.nf = nf  );
        }
        Ajouter(texte: string) {
                this.nf.Ajouter(texte); }
}
```

Templates et Composants

- Comment Angular connait ListeChosesService ?
- Comment Angular connait la balise item-chose ?
- Que signifie [nf] = "c" ?

```
const htmlTemplate : string = `
<form (ngSubmit)="Ajouter(newText.value)">
        <input type="text"  #newText  />
</form>
<ul>
        <li *ngFor="let c of nf.choses">
                <item-chose [nf]="c"></item-chose>
        </li>
</ul>
{{nf.choses.length}} items
`;
```

```
@Component({
        selector  : "liste-choses",
        template : htmlTemplate
})
export class CompListeChoses {
        nf : ListeChoses;
        constructor (S: ListeChosesService) {
            S.getData().then(
                nf => this.nf = nf  );
        }
        Ajouter(texte: string) {
                this.nf.Ajouter(texte); }
}
```

Templates et Composants

- Que signifie @Input() nf : Chose ?

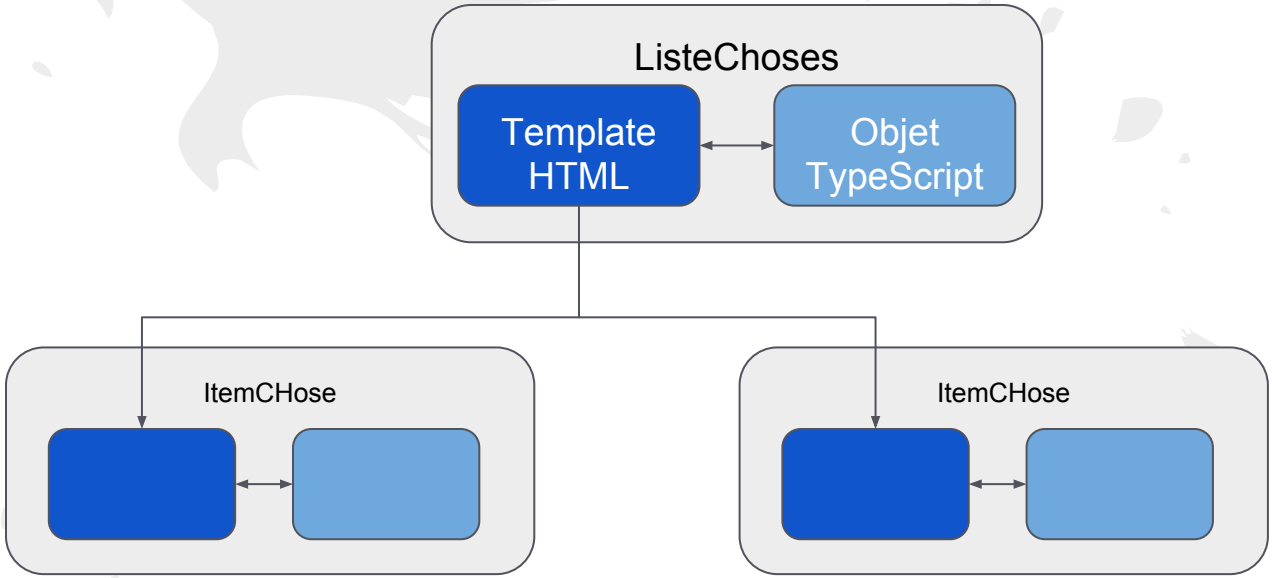
```
const htmlTemplate : string = `
<input type="checkbox"
        [ngModel]="nf.fait"
        (ngModelChange)="setFait($event)"  />

<label>{{nf.texte}}</label>

<button (onclick)="dispose()">X</button>
`;
```

```
@Component({
        selector  : "item-chose",
        template : htmlTemplate
})
export class ItemChose {
        @Input() nf : Chose;
        setFait(f: boolean) {this.nf.Fait(f);}
        dispose() {this.nf.dispose();}
}
```

Templates et Composants

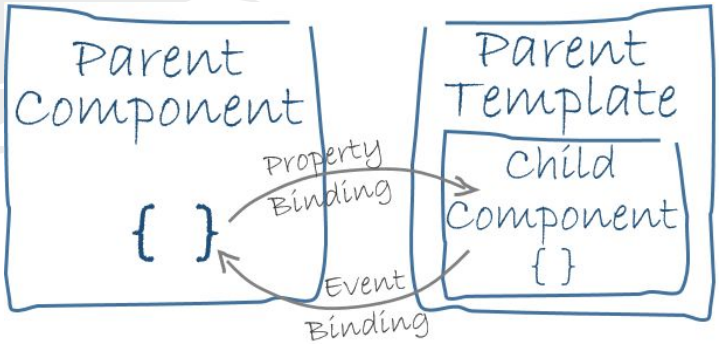
- Hiérarchie de composants

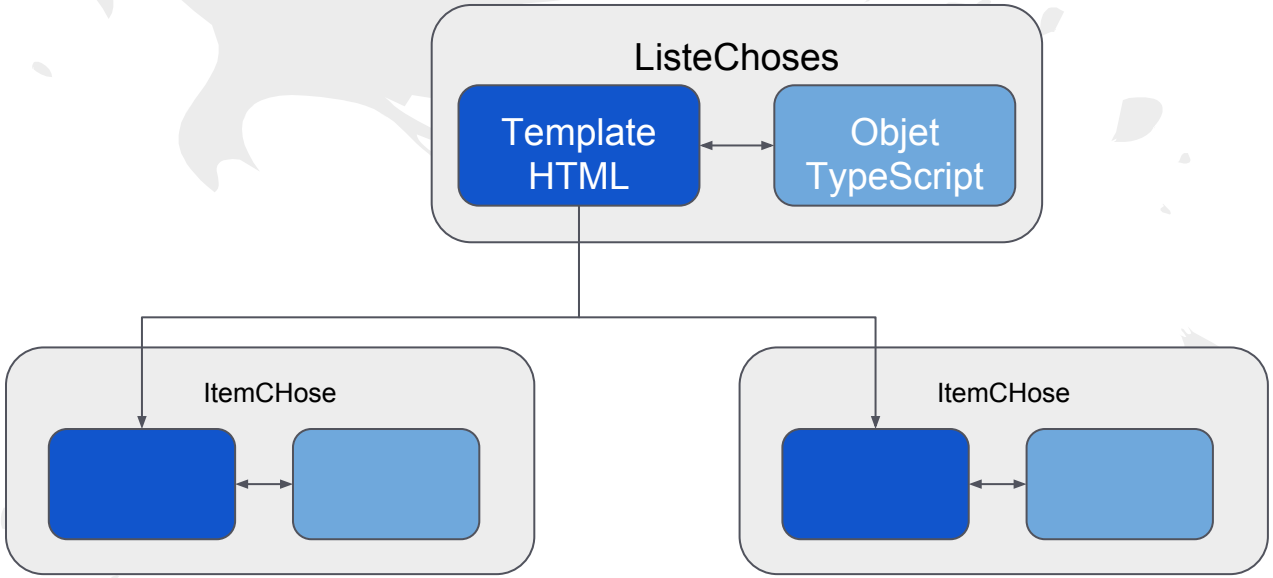
```
...
<li *ngFor="let c of nf.choses">
        <item-chose [nf]="c"></item-chose>
</li>
...
```



Templates et Composants

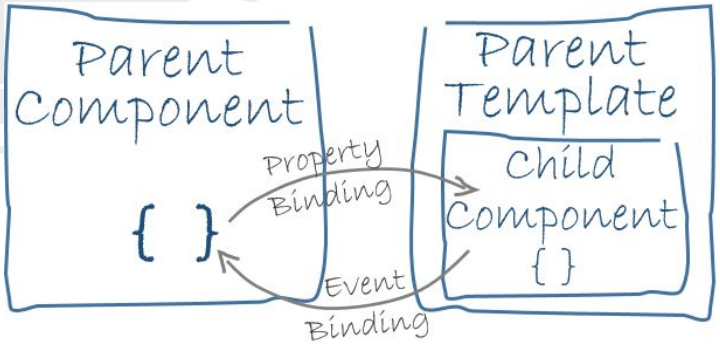
- Hiérarchie de composants
- Communication entre composants via la hiérarchie





Templates et Composants

- Hiérarchie de composants
- Communication entre composants via la hiérarchie
- Exemple ngModel et ngModelChange




```
const htmlTemplate : string = `
<input type="checkbox"
        [ngModel]="nf.fait"
        (ngModelChange)="setFait($event)"  />

<label>{{nf.texte}}</label>

<button (onclick)="dispose()">X</button>
`;
```

```
@Component({
        selector  : "item-chose",
        template : htmlTemplate
})
export class ItemChose {
        @Input() nf : Chose;
        setFait(f: boolean) {this.nf.Fait(f);}
        dispose() {this.nf.dispose();}
}
```

Templates et Composants

- Définissons nous aussi un émetteur d'événements : @Output() onBanco

Angular 2 : Template

Les directives (voir documentation)

- Créer des fragments de code HTML
- Injecter de l'information dans le HTML
- Transmettre de l'information vers le code TypeScript

Trois types de directives

- Directives structurelles (*ngIf, *ngFor, *ngSwitch, ...)
- Directives attributs (titre, [nf], (on-banco), ...)
- Directives composants (item-chose, liste-choses, ...)

Questions ?