Création de votre premier composant

Table des matières

I. Contexte 3

II. Qu'est-ce qu'un composant ? 3

III. Exercice : Quiz 5

IV. Génération avec le CLI Angular 6

V. Exercice : Appliquer la notion 8

VI. Interactions avec les composants enfants 8

VII. Exercice : Quiz 11

VIII. Essentiel 11

IX. Auto-évaluation 11

A. Exercice 11

B. Test 12

Solutions des exercices 13

I. Contexte

Durée : 1 h

Environnement de travail : IDE Visual Studio Code

Pré-requis : savoir lire et écrire en Typescript

Contexte

Les composants permettent de gérer l'affichage et les interactions avec l'utilisateur. Ils sont au centre de tout développement Angular. Nous allons dans ce module développer la notion de composant, découvrir la structure d'un composant et apprendre à en créer. Nous découvrirons aussi comment faire communiquer les composants entre eux.

II. Qu'est-ce qu'un composant ?

Objectifs de la partie

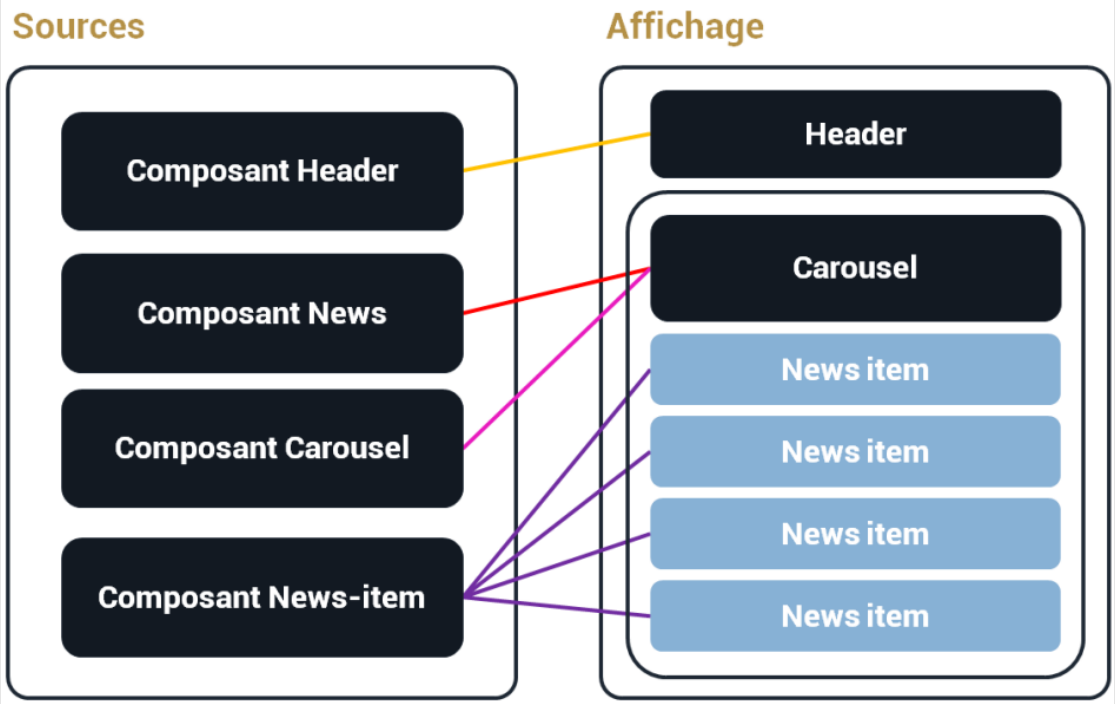
- Comprendre ce que sont les composants et découvrir leur structure

Contexte

En tant que développeur on est amené à créer ou modifier des composants dès lors que des modifications sont à apporter à l'affichage. Il faut créer une nouvelle page dans notre application ? Composant. On veut créer une vue représentant une liste de produits qu'on pourra intégrer dans une page web ? Composant. Les composants sont partout !

Méthode

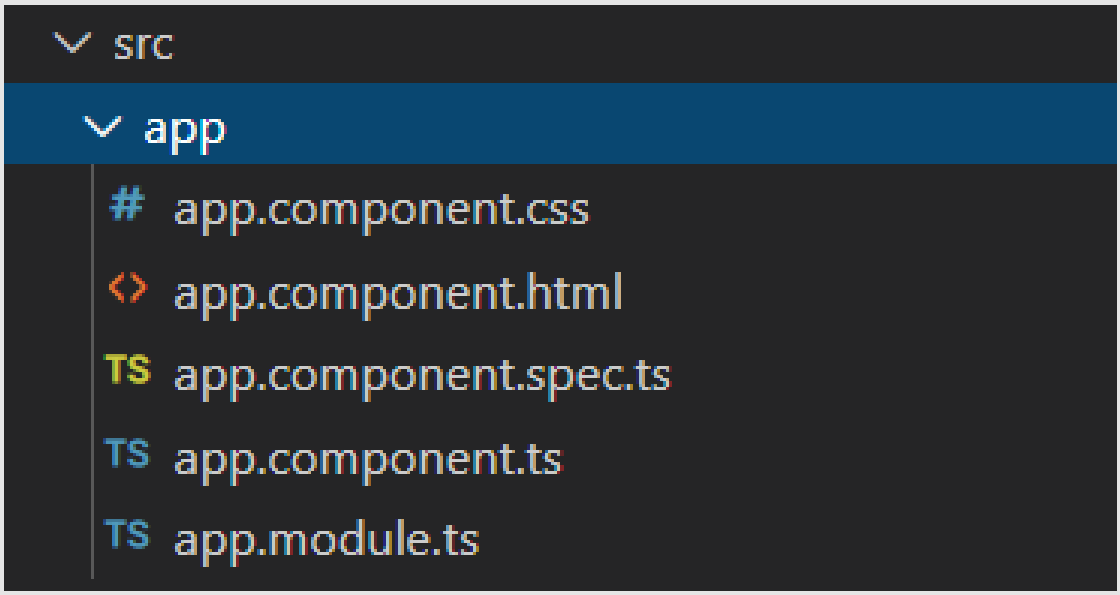
Ils contrôlent la partie graphique et sont les éléments de base d'une application Angular. Un composant permet de définir une page entière ou un sous-ensemble graphique et peut être composé d'autres composants.



Un composant est défini par plusieurs fichiers, gérant chacun un aspect de l'interface graphique. On retrouve ainsi :

- Le contrôleur, une classe Typescript qui porte la déclaration du composant et gère les données et les interactions avec l'utilisateur,
- Le *template* HTML, la partie graphique à proprement parler,
- Un ou des fichiers de style permettant de déclarer les styles propres au composant.

Seul le fichier du contrôleur est obligatoire, le *template* et les styles peuvent être déclarés d'une autre manière. Mais concentrons-nous tout de même sur cette structure, c'est celle que vous rencontrerez dans 99 % des cas. Vous pouvez d'ailleurs la retrouver en observant le composant généré par la commande `ng new`.



Intéressons-nous dans un premier temps au fichier ./src/app/app.component.ts :

```
1 import { Component } from '@angular/core';
2
3 @Component({
4 selector: 'sty-root',
5   templateUrl: './app.component.html',
6 styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9 title = 'sty-modules';
10 }
```

Dans ce fichier est déclarée une classe `AppComponent` enrichie du décorateur `@Component`. C'est lui qui nous permet de déclarer nos composants. À l'instar de `@Module`, il est possible d'ajouter des métadonnées à ce décorateur :

- `selector` : définit le nom de la balise utilisée dans le HTML pour faire appel au composant. Vous remarquerez que le `selector` est préfixé de `sty`, c'est le préfixe que nous avons défini à la création du projet.
- `templateUrl` correspond au chemin relatif vers le fichier HTML servant de *template* du composant (il est aussi possible de renseigner le *template* directement dans le fichier TS grâce à la métadonnée *template*).
- `styleUrls` contient un tableau de chemins relatifs vers les fichiers de styles du composant (de la même manière qu'avec *template*, il est possible d'utiliser la métadonnée `style`).

Au-delà de la déclaration du composant, la classe `AppComponent` porte les données affichées à l'écran et les méthodes permettant à l'utilisateur d'interagir avec l'application.

Complément

Angular - Component[1].

Exercice : Quiz

[solution n°1 p.15]

Exercice

Les composants permettent de définir des pages entières ou des sous-ensembles graphiques :

- ○ Vrai
- ○ Faux

Exercice

Pour définir un composant, on a besoin au minimum d'un fichier Typescript et d'un fichier HTML :

- ○ Vrai
- ○ Faux

Exercice

Le sélecteur d'un composant représente le nom à utiliser pour faire appel à ce composant :

- ○ Vrai
- ○ Faux

1 https://angular.io/api/core/Component

IV. Génération avec le CLI Angular

Objectifs de la partie

- Apprendre à créer un composant avec le CLI Angular

Contexte

A utiliser tout le temps !

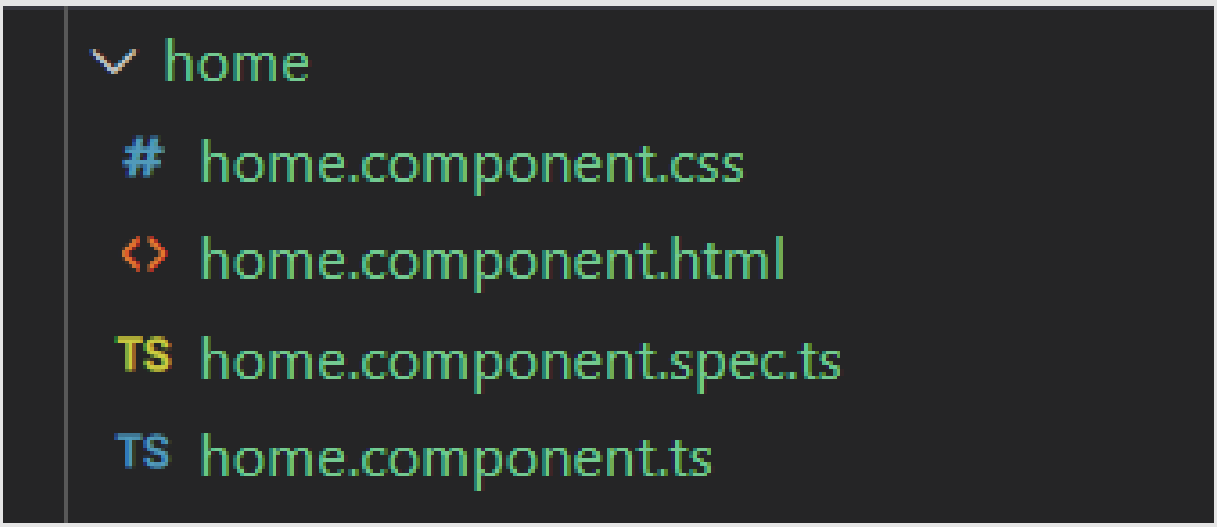
Méthode

Le CLI Angular nous permet de facilement générer la structure de base d'un composant. Pour ce faire, on va se servir de la commande `ng generate component <nom-composant>` (ou encore *`ng g c <nom-composant>`* en version raccourcie).

Dans un terminal, naviguez jusqu'à la racine de votre projet Angular et entrez la commande :

```
ng g c home
```

Cette commande a généré, dans `src`, un répertoire `home` ayant une structure similaire à celle du composant `app` vu précédemment. Ce composant fait donc partie du module principal (*root*) de notre application.



De plus, la commande a déclaré le composant auprès de son module parent. Dans notre cas, le module *root* :

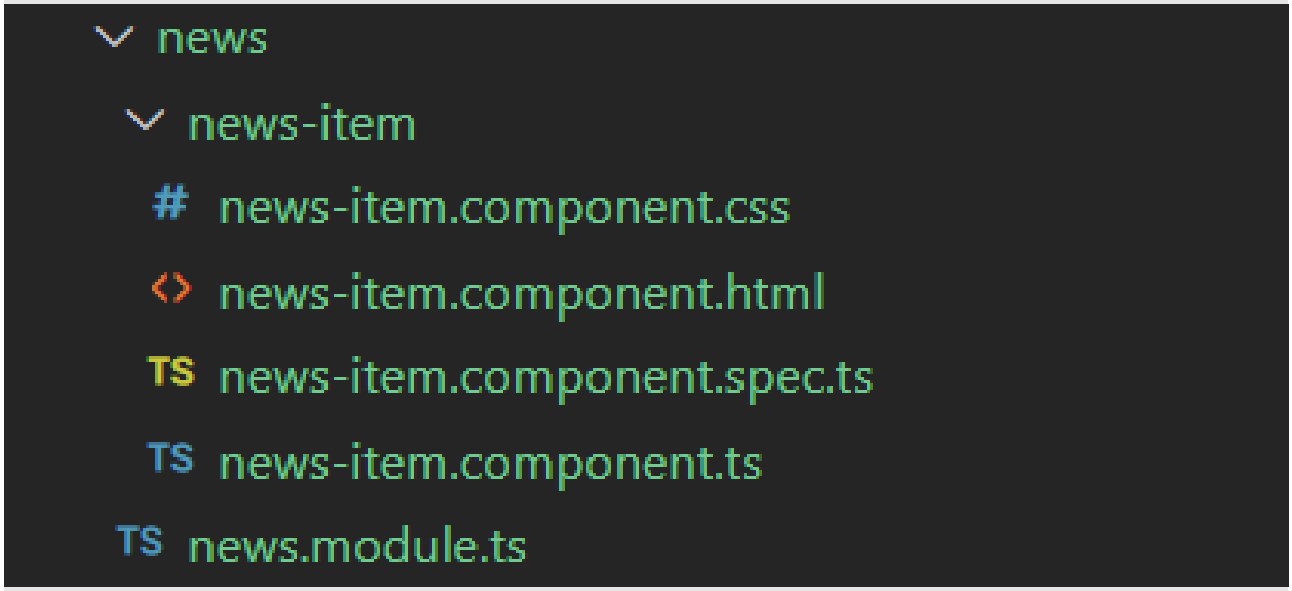
```
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Le plus souvent, par souci d'organisation, on souhaite créer un composant dans un autre module que *root*. Pour ce faire, on fait toujours appel à la commande `ng g c`, mais cette fois-ci, on donne le chemin complet du composant plutôt que son nom : `ng g c <chemin/nom-module>`.

Ainsi, si l'on a créé un module *news* et que l'on veut créer un composant à l'intérieur de ce module, on va utiliser la commande :

```
ng g c news/news-item
```

En regardant l'arborescence de notre projet, on constate que notre composant `news-item` a bien été créé dans le répertoire propre du module *news*. Il fait donc partie du module.

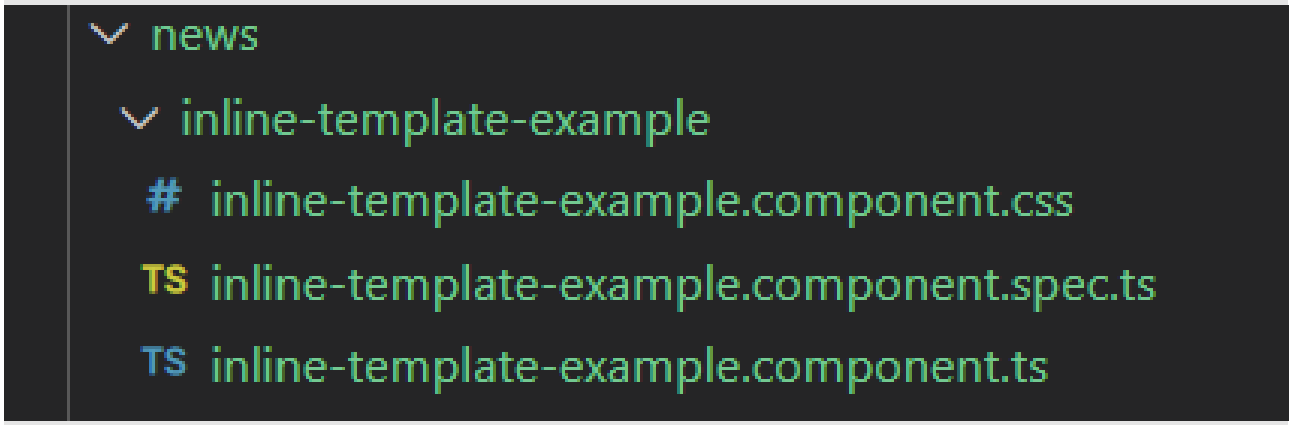


Quelques options disponibles avec la commande `ng g c` :

- `module` : permet de préciser le module parent du composant. Mieux vaut éviter cette option, elle ne fonctionne pas dans 100 % des cas. Préférez renseigner le chemin complet du composant.
- `style` : de la même manière qu'à la création d'un projet, on peut utiliser l'option `style` pour définir le format des fichiers de styles du composant. En son absence, c'est le format défini à la création du projet qui est utilisé.
- `inlineTemplate` : permet de générer un composant dont la déclaration du *template* se fait directement dans la classe TS. Dans ce cas, le fichier <composant>.component.html n'est pas généré et la métadonnée templateUrl du `@Component` est remplacée par *template*.

```
ng g c news/inline-template-example --inlineTemplate
```

L'exécution de cette commande donne le résultat suivant :



```
@Component({
  selector: 'sty-inline-template-example',
  template: `
    <p>
      inline-template-example works!
    </p>
  `,
  styleUrls: ['./inline-template-example.component.css']
})
```

La commande `ng g c` possède de nombreuses options permettant de paramétrer la création du composant.

Syntaxe **À retenir**

```
ng g c
```

Complément

Angular-ng generate[1].

V. Exercice : Appliquer la notion

Question

[solution n°2 p.15]

Générez un composant `article` qui utilise un fichier de style SASS et dont la déclaration du *template* est contenue dans la classe TS. Le sélecteur du composant doit être `app-custom-article`.

VI. Interactions avec les composants enfants

Objectifs de la partie

- Apprendre à faire passer des données entre les composants

Contexte

Bien souvent, nous avons besoin de faire passer des informations entre un composant et ses enfants.

Méthode

Nous avons vu qu'un composant Angular peut être composé d'autre composants. Cette approche est intéressante par nature mais le devient encore plus dès lors que l'on peut échanger des informations entre un parent et son enfant.

Passage d'informations à l'enfant `@Input`.

Le décorateur `@Input` permet de déclarer qu'un composant attend des informations en entrée. Il se place sur une propriété du composant et indique que celle-ci est *bindée* sur une propriété de son parent. Le parent doit renseigner cette propriété sous forme d'attribut du sélecteur. Une modification de la propriété d'origine dans le parent déclenche une modification de la propriété *bindée* de l'enfant.

Admettons que l'on désire afficher des news sur l'accueil de notre application. Nous créons donc un composant `news-item` représentant l'affichage d'une news. Le composant `app` va donc utiliser ce composant news-item (créant ainsi une relation parent-enfant). C'est bien le composant `app` qui va récupérer et contenir la liste de news et leur détail. Il va donc falloir que celui-ci passe les informations sur chacune des news au composant `news-item`.

1 https://angular.io/cli/generate#component

```typescript
@Component({
  selector: 'sty-news-item',
  templateUrl: './news-item.component.html'
})
export class NewsItemComponent {
  @Input()
  public title: string;
  @Input()
  public content: string;
}
```

Le composant `news-item` possède deux propriétés : `title` et `content`. Ces propriétés sont annotées `@Input`, cela indique qu'elles sont attendues en entrée du composant. Le *template* d'une *news* est relativement simple et contient juste l'affichage du titre et du contenu :

```html
<h3>{{ title }}</h3>
<p>{{ content }}</p>
```

Pour passer les paramètres `title` et `news`, le composant `app` utilise la syntaxe suivante :

```html
<p>Flash info, dernières nouvelles :</p>
<div *ngFor="let news of newsList">
  {{ news.title }}
  <sty-news-item title="{{ news.title }}" content="{{ news.content }}"></sty-news-item>
</div>
```

Il parcourt la liste de *news* qu'il contient grâce à sa propriété `newsList`. Et pour chaque news, il crée une instance du composant `news-item` pour afficher ses informations.

Il est aussi tout à fait possible de passer des valeurs fixes en paramètre :

```html
<sty-news-item title="Hello" content="World"></sty-news-item>
```

Passage d'informations au parent `@Output`.

Un composant enfant peut aussi avoir besoin de passer des informations ou un événement à son parent afin qu'il effectue des actions. Cette organisation du code permet de rendre les composants plus génériques et indépendants. Cette transmission d'informations se fait grâce au décorateur `@Output`.

Reprenons l'exemple de notre affichage de *news*. Nous voulons maintenant pouvoir notifier le composant `app` au clic sur un bouton afin qu'il ajoute un smiley au titre.

```typescript
@Component({
  selector: 'sty-news-item',
  templateUrl: './news-item.component.html'
})
export class NewsItemComponent {
  @Input()
  public title: string;
  @Input()
  public content: string;

  @Output()
  private smileyClick = new EventEmitter<any>();

  appendSmiley() {
    this.smileyClick.emit(this.title);
  }
}
```

Vous remarquerez que `@Output` s'utilise sur une propriété de type `EventEmitter` et qu'il faut appeler la méthode `.emit()` pour déclencher l'événement.

```
<h3>{{ title }}</h3>
<p>{{ content }}</p>

<button (click)="appendSmiley()">Ajouter smiley</button>
```

Le clic sur le bouton « *Ajouter smiley* » provoque l'appel à la méthode du news-item, celle-là même qui utilise le `.emit()` et déclenche l'événement.

```
<p>Flash info, dernières nouvelles :</p>
<div *ngFor="let news of newsList">
  {{ news.title }}
  <sty-news-item title="{{ news.title }}" content="{{ news.content }}"
(smileyClick)="appendSmiley($event)"></sty-news-item>
</div>
```

On utilise ensuite `(smileyClick)=”<`, méthode du parent permettant de traiter l'événement `>”` pour récupérer et traiter l'événement émis par l'enfant.

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

  public newsList = [
    new News('Météo - Demain', 'Il fait beau demain.'),
    new News('Météo - Apres demain', 'Il fait beau après demain !'),
    new News('Attention', 'Problème de contrôle')
  ];

  appendSmiley($title: string) {
    this.newsList.find(news => news.title === $title).title += ' :)';
  }
}
```

Enfin, la méthode `appendSmiley` du parent a la charge d'ajouter un smiley à la fin du titre de la *news* concernée, retrouvée ici grâce à son titre passé par l'enfant dans l'événement.

Syntaxe **À retenir**

- `@Input`,
- `@Output`,
- `<childComponent attributInput=”valeur” (attributOutput)=”méthode de traitement de l'événement ”>`.

Complément

- Angular - Input[1]
- Angular - Output[2]

1 https://angular.io/api/core/Input
2 https://angular.io/api/core/Output

Exercice : Quiz

[solution n°3 p.15]

Exercice

Le décorateur `@Input` permet de *binder* une propriété de l'enfant avec un élément fourni par son parent :

- ○ Vrai
- ○ Faux

Exercice

Si une propriété `@Input` est modifiée dans l'enfant, sa valeur est automatiquement mise à jour dans le parent :

- ○ Vrai
- ○ Faux

Exercice

Le décorateur `@Output` se place sur une propriété de type `EventEmitter` :

- ○ Vrai
- ○ Faux

VIII. Essentiel

IX. Auto-évaluation

A. Exercice

Question

[solution n°4 p.16]

Créer un clavier de téléphone avec l'affichage du numéro. Utiliser des décorateurs appris pendant l'unité pédagogique pour permettre de transférer des données entre composants.

Objectif de l'exercice :

Créer un composant téléphone avec 3 sous-composants :

- Clavier
- Boutons de suppression
- Bouton d'appel

Le composant du téléphone doit permettre d'afficher un numéro et un message spécifiant si l'appel est en cours ou s'il y a une erreur dans le numéro.

Le composant clavier est composé de 10 boutons allant de 0 à 9 et à chaque clic sur un bouton on doit ajouter le numéro au composant téléphone.

Le composant « *boutons de suppression* » est composé de 2 boutons :

- Lorsque l'on clique sur le premier bouton le téléphone doit supprimer 1 numéro.
- Lorsque l'on clique sur le deuxième bouton le téléphone doit supprimer tous les numéros.

Le composant « *bouton d'appel* » est composé de 1 bouton. Au clic le bouton prend le numéro actuel et vérifie qu'il commence bien par 0 et qu'il contient 10 numéros.

Indice :

Pour interagir avec le clic de l'utilisateur il suffit d'ajouter une directive `click`, voici un exemple :

```
1 <button (click)=”maMethod()”> mon bouton</button>
```

- Un clic sur le bouton déclenchera l'appel à la méthode "`maMethod()`" de votre composant.
- Le numéro de téléphone ne doit pas être de type `number` car il doit commencer par 0.
- La fonction `substr` permet de récupérer des sous-parties d'une chaîne de caractères.
- Il n'y a qu'un seul *Input* pour tout l'exercice. Éviter de traiter les valeurs dans les sous-composants.

B. Test

Exercice 1 : Quiz

[solution n°5 p.16]

Question 1

Quel est l'utilité des composants Angular ?

- ☐ Ils permettent de gérer l'aspect graphique de l'application
- ☐ Ils permettent de gérer les interactions avec les utilisateurs
- ☐ Ils représentent une page entière ou un sous-ensemble graphique
- ☐ Les composants n'ont pas de relation entre eux

Question 2

Un composant est obligatoirement constitué au minimum de 3 fichiers :

- ○ Vrai
- ○ Faux

Question 3

Que représente la métadonnée templateUrl du décorateur `@Component` ?

- ○ Il s'agit du chemin relatif vers le fichier *template*
- ○ Il s'agit du *template* HTML inline
- ○ Il s'agit du fichier de style spécifique au composant

Question 4

Quelles sont les propositions vraies à propos de la commande `ng g c` ?

- ☐ Cette commande est fournie par Angular CLI
- ☐ Par défaut, elle génère un composant avec un template inline
- ☐ Cette commande permet de générer la structure d'un composant Angular
- ☐ C'est un raccourci pour `ng generate component`

Question 5

Comment indiquer à `ng g c` de générer un composant dans un module donné ?

- ☐ L'option `--module` permet de préciser le module et il s'agit de la solution recommandée
- ☐ En indiquant le chemin complet du composant plutôt que son seul nom : *`ng g c mon-module/mon-composant`*
- ☐ L'option `--module` permet de préciser le module mais elle est source d'erreur et il vaut mieux l'éviter
- ☐ C'est impossible

Question 6

Les composants sont indépendants les uns des autres :

- ○ Vrai
- ○ Faux

Question 7

Où se placent les décorateurs `@Input` et `@Output` ?

- ○ Sur les méthodes des composants
- ○ Sur les méthodes des services
- ○ Sur les propriétés des composants
- ○ Dans la déclaration des modules

Question 8

Lorsque l'on utilise un composant, comment lui passe-t-on un paramètre `@Input` ?

- ○ Dans le constructeur de l'instance de composant
- ○ Grâce à un attribut portant le nom de l'input
- ○ Dans le corps de la balise sélecteur du composant
- ○ Grâce à un `event emitter`

Question 9

Quel décorateur utilise-t-on pour passer des informations d'un composant enfant à son parent ?

- ○ `@Output`
- ○ `@Input`
- ○ `@Param`
- ○ `@Inbound`

Question 10

Quelles propositions sont vraies à propos du décorateur `@Output` ?

- ☐ Il se place sur une propriété d'un composant
- ☐ Lors de l'utilisation du composant dans un *template* HTML, il faut préciser la méthode à appeler lorsqu'un événement est émis
- ☐ Une propriété `@Output` peut être de n'importe quel type
- ☐ Une propriété `@Output` doit être de type `EventEmitter`

Solutions des exercices

Exercice p. 5 Solution n°1

Exercice

Les composants permettent de définir des pages entières ou des sous-ensembles graphiques :

- (•) Vrai
- ( ) Faux

Les composants gèrent l'aspect graphique de l'application et permettent de définir des pages ou sous-ensembles graphiques.

Exercice

Pour définir un composant, on a besoin au minimum d'un fichier Typescript et d'un fichier HTML :

- ( ) Vrai
- (•) Faux

Seule la classe Typescript est nécessaire. Le template peut être défini inline, dans la métadonné template de `@Component`.

Exercice

Le sélecteur d'un composant représente le nom à utiliser pour faire appel à ce composant :

- (•) Vrai
- ( ) Faux

Pour utiliser un composant dans un fichier HTML, on utilise une balise nommée d'après le sélecteur de celui-ci.

p. 8 Solution n°2

ng g c article --style=sass --inlineTemplate --selector=app-custom-article

```
@Component({
  selector: 'sty-custom-article',
  template: `
    <p>
      article works!
    </p>
  `,
  styleUrls: ['./article.component.sass']
})
export class ArticleComponent
```

Exercice p. 11 Solution n°3

Exercice

Le décorateur `@Input` permet de *binder* une propriété de l'enfant avec un élément fourni par son parent :

- (●) Vrai
- ( ) Faux

`@Input` permet au parent de transmettre une valeur à son enfant. Cette valeur peut être fournie en dur ou bien provenir d'une propriété du parent.

Exercice

Si une propriété `@Input` est modifiée dans l'enfant, sa valeur est automatiquement mise à jour dans le parent :

- ( ) Vrai
- (●) Faux

La mise à jour n'est pas automatique dans le parent. `@Input` sert seulement à passer une information du parent à l'enfant.

Exercice

Le décorateur `@Output` se place sur une propriété de type `EventEmitter` :

- (●) Vrai
- ( ) Faux

Il faut alors ensuite déclencher l'évènement grâce à la méthode « emit » de l'EventEmitter afin de notifier le parent qu'une modification a eu lieu.

p. 11 Solution n°4

- Pensez à utiliser en priorité `@angular/cli`
- Créer votre projet : `ng g new first-component`
- Création des composant :
  - `ng g c phone`
  - `ng g c keyboard`
  - `ng g c delete`
  - `ng g c call`

Exercice p. 12 Solution n°5

Question 1

Quel est l'utilité des composants Angular ?

- [x] Ils permettent de gérer l'aspect graphique de l'application
- [x] Ils permettent de gérer les interactions avec les utilisateurs
- [x] Ils représentent une page entière ou un sous-ensemble graphique
- [ ] Les composants n'ont pas de relation entre eux

Les composants Angular permettent de gérer l'aspect graphique de l'application ainsi que les interactions avec les utilisateurs. Ils représentent une page entière ou un sous-ensemble graphique.

Question 2

Un composant est obligatoirement constitué au minimum de 3 fichiers :

- ○ Vrai
- ◉ Faux

Un composant doit contenir, a minima, une classe Typescript et un *template* HTML. Le *template* peut cependant être déclaré `inline`, c'est-à-dire directement dans le fichier de la classe Typescript grâce à la métadonnée « *template* » du décorateur `@Component`. Les composants peuvent aussi avoir un fichier de styles propre mais ceci n'est pas une obligation. De fait, seul le fichier `.ts` est obligatoire.

Question 3

Que représente la métadonnée templateUrl du décorateur `@Component` ?

- ◉ Il s'agit du chemin relatif vers le fichier *template*
- ○ Il s'agit du *template* HTML inline
- ○ Il s'agit du fichier de style spécifique au composant

Cette métadonnée peut être remplacée par« *template* » pour déclarer le *template* HTML directement plutôt que dans un autre fichier.

Question 4

Quelles sont les propositions vraies à propos de la commande `ng g c` ?

- ☑ Cette commande est fournie par Angular CLI
- ☐ Par défaut, elle génère un composant avec un template inline
- ☑ Cette commande permet de générer la structure d'un composant Angular
- ☑ C'est un raccourci pour `ng generate component`

Question 5

Comment indiquer à `ng g c` de générer un composant dans un module donné ?

- ☐ L'option `--module` permet de préciser le module et il s'agit de la solution recommandée
- ☑ En indiquant le chemin complet du composant plutôt que son seul nom : *`ng g c mon-module/mon-composant`*
- ☑ L'option `--module` permet de préciser le module mais elle est source d'erreur et il vaut mieux l'éviter
- ☐ C'est impossible

Il faut indiquer le chemin complet du composant. L'option `-module` permet également de préciser le module mais il vaut mieux éviter de l'utiliser.

Question 6

Les composants sont indépendants les uns des autres :

- (●) Vrai
- ( ) Faux

Q C'est le concept même d'un composant. Les composants sont créés pour être réutilisables dans diverses situations. C'est pourquoi il existe des mécanismes permettant de passer des informations entre composants.

Question 7

Où se placent les décorateurs `@Input` et `@Output` ?

- ( ) Sur les méthodes des composants
- ( ) Sur les méthodes des services
- (●) Sur les propriétés des composants
- ( ) Dans la déclaration des modules

Q Les décorateurs `@Input` et `@Output` se placent sur les propriétés des composants.

Question 8

Lorsque l'on utilise un composant, comment lui passe-t-on un paramètre `@Input` ?

- ( ) Dans le constructeur de l'instance de composant
- (●) Grâce à un attribut portant le nom de l'input
- ( ) Dans le corps de la balise sélecteur du composant
- ( ) Grâce à un `event emitter`

Q Quand on utilise le sélecteur d'un composant dans un *template* pour l'afficher, on lui passe un attribut portant le nom de l'input. Cet attribut contient la valeur qui sera utilisée pour définir la propriété `@Input`.

Question 9

Quel décorateur utilise-t-on pour passer des informations d'un composant enfant à son parent ?

- ( ) `@Output`
- (●) `@Input`
- ( ) `@Param`
- ( ) `@Inbound`

Q `@Output` est placé sur une propriété du composant enfant. Cette propriété doit être de type `EventEmitter` et va permettre d'émettre des événements. Le parent a la charge de déclarer l'action à entreprendre lorsque cet événement est levé.

Question 10

Quelles propositions sont vraies à propos du décorateur `@Output` ?

- [x] Il se place sur une propriété d'un composant
- [x] Lors de l'utilisation du composant dans un *template* HTML, il faut préciser la méthode à appeler lorsqu'un événement est émis
- [ ] Une propriété `@Output` peut être de n'importe quel type
- [x] Une propriété `@Output` doit être de type `EventEmitter`

Le décorateur `@Output` se place en effet sur une propriété d'un composant. Il faut également préciser la méthode à appeler. De plus, une propriété `@Output` doit être de type `EventEmitter`.