AUJOURD'HUI: ANGULAR 20

- A peu près une nouvelle version majeure tous les 6 mois qui peut introduire des problèmes de rétrocompatibilités mais permet une amélioration continue du framework.
    - Angular 19: les signaux, ...
    - Angular 18: nouvelle syntaxe pour le contrôle de flux dans les templates, ...
    - Angular 17: la CLI (Command Line Interface) Angular utilise Vite, ...
    - Angular 15: composants standalone, ...

LES GUIDES ANGULAR

- Pour mettre à jour une application, l'équipe d'Angular fournit des guides
- Un guide de bonnes pratiques de codage est également disponible et a été mis à jour en 2025

ANGULAR CLI

- Outil en ligne de commande pour rapidement démarrer un projet (et bien plus)

Documentation Angular CLI

- Installation

```
npm install -g @angular/cli@20.2.0
```

```
ng version
```

`npx ng version` sur les PC de l'IUT

PREMIÈRE APPLICATION ANGULAR

- Création d'une application Angular

```
ng new my-first project --prefix iut
```

```
--defaults --skip-install
```

> Remplacez "my" par votre nom dans cette commande

- Visualisez l'arborescence créée
- Ajoutez/remplacez par les fichiers fournis dans le zip
- Déplacez-vous dans le répertoire `my-first-project` et installez les dépendances `npm install`
- Ajoutez ESLint

```
ng add angular-eslint@20.1.1 --skip-confirmation
```

PREMIÈRE APPLICATION ANGULAR

- Utilisez le "linter"

```
ng lint --fix
```

- Utilisez `prettier`

```
npm run format
```

- Lancez les tests unitaires

```
ng test
```

- Servir l'application

```
ng serve
```

COMPOSANT ANGULAR (1/3)

- Un composant est une classe exposant une vue et définissant la manière dont l'utilisateur interagit avec cette vue.
- Un composant est composé de trois parties:
  - la partie logique = une classe TypeScript
  - la partie sémantique = le template HTML
  - la partie présentation = les styles CSS

COMPOSANT ANGULAR (2/3)

```ts
// extrait de app.component.ts
import { Component } from '@Angular/core';

@Component({
  selector: 'iut-root',
  imports: [],
  templateUrl: './app.html',
  styleUrl: './app.css'
})
export class AppComponent {
  protected readonly title = signal('my-first-project');
}
```

- le décorateur @Component identifie la classe qui le suit immédiatement comme un composant et spécifie des méta-données sur celui-ci.
- le modificateur d'accès `private` limiterait l'accès à la classe; `protected` autorise en plus l'accès à partir du template du composant.

COMPOSANT ANGULAR (3/3)

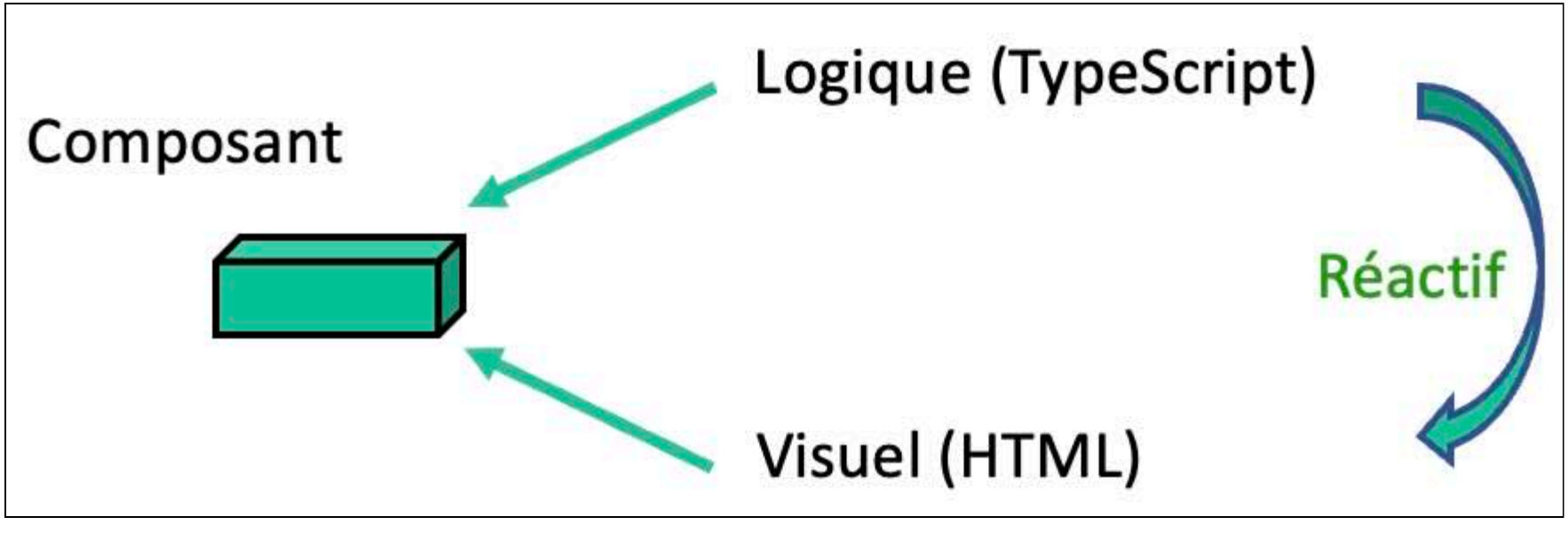
- une application Angular est composée d'une hiérarchie de composants (comme avec Vue.js).
- une instance du composant racine `app` (`iut-root`) est utilisée par la page HTML de l'application.

```
//extrait de index.html
<body>
  <iut-root></iut-root>
</body>
```

- le nom de la balise à intégrer au template pour utiliser un composant est son sélecteur.
- en production, `index.html` contient en plus le chargement de scripts pour lancer l'application Angular et hydrater le composant racine. Le fichier `main.ts` contient l'instruction pour lancer l'application (`bootstrapApplication`).

LA RÉACTIVITÉ

Rappel: Le visuel d'un composant peut utiliser la valeur d'une propriété de celui-ci. Si celle-ci est réactive, son changement est détecté et le visuel est actualisé. C'est la **réactivité**.



LA RÉACTIVITÉ ET ANGULAR (1/5)

- Jusqu'à récemment, avec Angular, tout était réactif:
  - une bibliothèque ZoneJS détectait tout code exécuté par le navigateur suite à une arrivée de promesse, un événement écouté, … et déclenchait une détection de changements
  - une détection de changement était effectuée par Angular qui parcourait tout l'arbre des composants et ré-évaluait chaque expression pour voir ce qui devait être mis à jour dans le DOM

LA RÉACTIVITÉ ET ANGULAR (2/5)

- Depuis Angular 19, il est recommandé d'utiliser **les signaux** pour gérer la réactivité.
- C'est ce qui est utilisé dans Vue.js avec sa fonction `ref` par exemple.
- Un signal c'est une valeur (nombre, chaîne de caratères, objet, tableau) enveloppée dans un objet.
- On la modifie avec sa méthode `set` et on accède à sa valeur avec l'appel d'une fonction `getter`.

```
protected readonly title = signal('my-first-project');

constructor(){
  setTimeOut(() => {
    this.title.set('new-title')
  }, 1000);
}
```

```
<h1>Hello, {{ title() }}</h1>
```

INTERPOLATION DANS UN TEMPLATE

Elle permet de lier l'affichage aux données (one-way data binding).

```ts
// extrait de iut-root.ts
export class App {
  protected readonly title = signal('my-first-project');
}
```

```html
<!-- extrait de app.component.html -->
<h1>Hello, {{ title() }}</h1>
```

L'expression entre {{ }} est évaluée.

LA RÉACTIVITÉ ET ANGULAR (3/5)

- Avec les signaux, on peut construire une application Angular sans ZoneJS: une 'zoneless' application.
- Seuls quelques événements précis déclencheront une détection de changements dont: événements écoutés du DOM, modifications d'un `input` de composant ou d'un signal.

La synchronisation zoneless était expérimentale en v18 et v19. Elle est devenue stable avec Angular 20. Autant l'adopter pour les nouveaux projets. C'est le futur d'Angular.

LA RÉACTIVITÉ ET ANGULAR (4/5)

- Plutôt que de faire ré-évaluer toutes les expressions de tous les composants à chaque déclenchement d'un détection de changements, il est recommandé d'utiliser la stratégie **OnPush**.
- Un composant ne sera vérifié que dans quelques cas dont:
    - modification d'un signal du composant. Dans le cas d'un objet ou d'un tableau, il faut que la référence change, pas seulement le contenu.
    - changement d'un de ses `input`
    - exécution d'un de ses gestionnaires d'événements

permet de modifier la stratégie avec cette option dans `angular.json`

LA RÉACTIVITÉ ET ANGULAR (5/5)

- Si vous travaillez sur un projet angular qui utilise toujours `zone.js`, vous pouvez créez de nouveaux composants prêts pour un futur passage en mode Zoneless.

```
@Component({
  selector: 'iut-component',
  imports: [],
  templateUrl: './component.html',
  styleUrl: './component.css',
  changeDetection: ChangeDetectionStrategy.OnPush
})
```

Le mode par défaut de `changeDetection` est `checkAlways`

FLUX DE CONTRÔLE DANS UN TEMPLATE (1/4)

SÉLECTION ET ALTERNATIVES:

- @if(condition) {} / @else if(condition) {}/ @else {},
- @switch(expression) {} / @case(valeur) {} / @default {}

```
@if (isLoggedIn()) {
  <p>Welcome back, Friend!</p>
}
```

Remplacent *ngIf, *ngElse, ...

C'est la syntaxe recommandée.

FLUX DE CONTRÔLE DANS UN TEMPLATE (2/4)

BOUCLE @FOR {} / @EMPTY{}:

```
@for (item of items(); track item.id) {
  <li>{{ item.nom }}</li>
} @empty {
  <li>There are no items.</li>
}
```

track associe un item d'une collection à l'extrait correspondant du DOM (comme l'attribut spécial key dans Vue.js). Il faut donc utiliser une propriété de l'item unique et stable pour assurer de bonnes performances.

FLUX DE CONTRÔLE DANS UN TEMPLATE (3/4)

Dans le for, des variables contextuelles existent.

Variables	Description du contenu
$count	nombre d'items
$index	indice de l'item
$first	est le premier item
$last	est le dernier item
$odd	l'item a un indice impair
$even	l'item a un indice pair

FLUX DE CONTRÔLE DANS UN TEMPLATE (4/4)

```
<ul>
@for (item of ['salade', 'thon', 'maïs']; track $index) {
  <li>item n°{{ $index }}/{{$count}}: {{ item }}</li>
}
</ul>
```

- item n°0/3: salade
- item n°1/3: thon
- item n°2/3: maïs

track peut être utilisé avec l'indice de l'item si la collection est statique.

BINDING DE PROPRIÉTÉ

Elle permet de lier les propriétés des éléments du DOM (balises HTML ou composants) à des données (one-way data binding). C'est le v-bind (:) de Vue.js.

```
<ul>
@for (item of liens; track item.id) {
  <li><a [href]="item.url">{{ item.texte }}</a></li>
}
</ul>
```

GESTION D'ÉVÉNEMENT

```
@Component({
    ...
    template: `<button type="button" (click)="onButtonClick()">`
})
class MyComponent {
    onButtonClick() {
        console.log('Hello 👋');
    }
}
```

Le gestionnaire d'événement est une instruction, ou plusieurs: l'appel d'une méthode du composant est une bonne pratique. On peut lui transmettre $event:

- l'Event déclencheur dans le cas d'un événement classique du DOM.
- l'information portée par l'événement dans le cas d'un événement personnalisé.

C'est le v-on (@) de Vue.js.

TWO-WAY DATA BINDING

L'équivalent du v-model de Vue.js s'obtient avec `[(maPropriété)]`. Pour se rappeler de l'ordre des symboles, on parle de bananes () dans une boîte [].

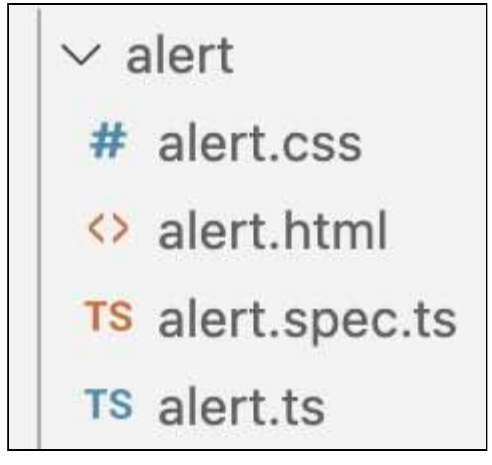
S'utilise surtout avec les éléments de formulaire.

> Son utilisation n'est pas systématique car Angular offre des modules dédiés aux formulaires qui n'en ont pas besoin

CRÉATION D'UN COMPOSANT

`ng generate component alert` OU

`ng g c alert`



Ce composant affiche un paragraphe avec un message qui est une propriété du composant:

Votre alerte est bien intégrée!

INTÉGRATION D'UN COMPOSANT

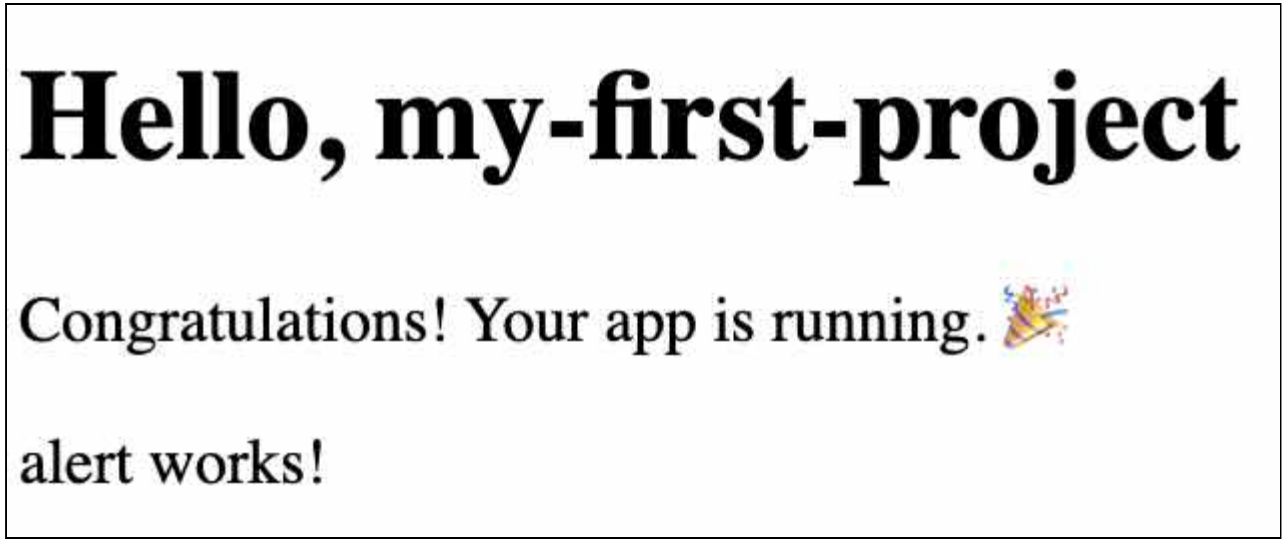
Simplifions le composant App (iut-root) de notre premier projet pour garder dans une balise header le titre de niveau 1 avec le paragraphe de félicitations dans la balise main.

Un nouveau composant est instancié par le fait qu'on l'intègre dans le DOM, qu'il est donc affiché par un template.

Ajoutons donc un composant Alert dans la balise main.

Utilisons l'IDE pour régler le problème des imports.

RÉSULTAT ATTENDU



GESTION DES STYLES

- une feuille de styles globale dans `src/styles.css`
- une feuille de styles par composant pour ses styles spécifiques (ne s'appliquent pas sur les composants enfants par défaut)
- `:host` permet d'appliquer un style à la balise qui correspond au composant (l'élément hôte iut-quelquechose)
- Comment fonctionne cette isolation ? Angular réécrit les feuilles de styles: `p[_ngcontent-ng-cxxx]`

Essayons avec une couleur de police rouge pour les paragraphes de App et un fond jaune pour les paragraphes du composant Alert.

BINDING DE CLASS SIMPLE

`[class.enVente]="aVendre"`

La classe enVente est ajouté à l'élément si aVendre correspond à une valeur truthy. Elle est enlevée si c'est une valeur falsy sauf undefined (0, '', false, …).

BINDING DE CLASSES MULTIPLES (1/2)

- avec une chaîne de caractères

```
[class]="my-class-1 my-class-2"
```

- avec un tableau de chaînes de caractères

```
[class]=['my-class-1', 'my-class-2']
```

- avec un objet littéral avec les classes comme propriétés, affectées quand leur valeur associée est truthy.

```
[class]={'my-class-1': boolean1, 'my-class-2': boolean2}
```

Pour que les classes soient mises à jour avec un objet ou un tableau, il faut que la référence change. Changer la valeur

BINDING DE CLASSES (2/2)

Ajoutons une propriété category au composant Alert. Elle pourra prendre comme valeur:

- 'validation' pour signaler une action réussie
- 'warning' pour signaler une action incorrecte
- 'info' pour les informations

et sera initialisée à 'info'.

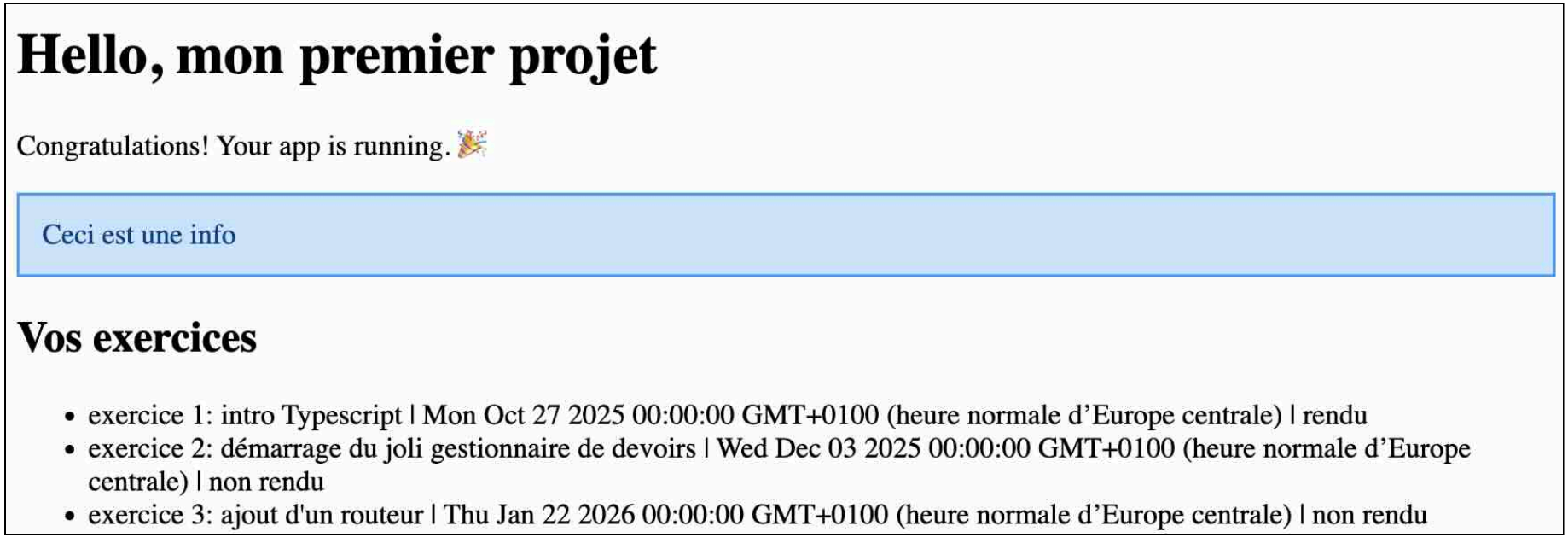
Modifions le composant pour que la catégorie influence l'affichage. Par exemple, l'action réussie sera signalée avec un fond vert, l'action incorrecte avec un fond orange ou rouge et l'information avec un fond bleu.

Utilisons ce nouveau composant pour valider que notre application fonctionne.

RÉSULTAT ATTENDU

EXERCICE PRATIQUE: RÉSULTAT ATTENDU

- Ajouter un composant Exercises à notre premier projet qui permettra à un étudiant ou une étudiante de gérer le rendu de ses devoirs



- Le template du composant contient un titre de niveau 2 et une liste

EXERCICE PRATIQUE: PAS À PAS (1/2)

Un exercice sera stocké dans un objet littéral, comme dans l'exemple ci-dessous. Un exercice a un id, un intitulé, une date limite de rendu et a déjà été rendu ou pas.

```
{
  id: '1',
  title: 'intro Typescript',
  dueDate: new Date(2025, 9, 27),
  submitted: true
}
```

Dans le répertoire `src/app`, créez un répertoire `models`. Dans celui-ci, créez une interface `ExerciseModel`. Utilisez la CLI pour créer le squelette de l'interface:

`ng generate interface models/ExerciceModel` OU

`ng g i models/ExerciceModel`

EXERCICE PRATIQUE: PAS À PAS (2/2)

- Créez un composant `Exercises` en utilisant la CLI, avec une propriété `tabExercises` qui a pour valeur un tableau de trois exercices. En plus de l'exercice précédent, il y aura:

```
{
  id: '2',
  title: 'démarrage du joli gestionnaire de devoirs',
  dueDate: new Date(2025, 11, 3),
  submitted: false
},
{
  id: '3',
  title: 'ajout d\'un routeur',
  dueDate: new Date(2025, 12, 22),
  submitted: false
}
```

- Le template du composant contient un titre de niveau 2 et une liste d'exercices (voir diapo 52 pour l'attendu)
- Intégrez ce composant dans le `template` de App.