Лабораторне завдання №5: Робота з директивами. Атрибутивні та структурні директиви.

Мета: Навчитися створювати та використовувати директиви в Angular.

Завдання: Створити чотири Angular-додатки під назвою Directives1, Directives2, Directives3 та Directives4.

I) Для Angular-додатку Directives1 виконати вправу 1;

II) Для Angular-додатку Directives2 виконати вправу 2;

III) Для Angular-додатку Directives3 виконати вправу 3 (виконати самостійне завдання);

IV) для Angular-додатку Directives4 виконати вправу 4 (виконати самостійне завдання).

V) Виконати самостійне завдання зі створенням Angular-додатків Directives5 та Directives6;

VI) Зробити звіт по роботі.

VII) Angular-додатки Directives1 та Directives2 розгорнути на платформі Firebase у проектах з ім'ям «ПрізвищеГрупаLaba5-1» та «ПрізвищеГрупаLaba5-2», наприклад «KovalenkoIP01Laba5-1» та «KovalenkoIP01Laba5-2».

Директиви визначають набір інструкцій, що застосовуються при рендерингу HTML-коду. Директива представляє клас із директивними метаданими. У TypeScript для прикріплення метаданих до класу застосовується декоратор @Directive.

У Angular є три типи директив:

1) Компоненти: компонент по суті є директивою, а декоратор @Component розширює можливості декоратора @Directive за допомогою додавання функціоналу по роботі з шаблонами.
2) Атрибутивні: вони змінюють поведінку існуючого елемента, до якого вони застосовуються. Наприклад, ngModel, ngStyle, ngClass
3) Структурні: вони змінюють структуру DOM за допомогою додавання, зміни чи видалення елементів HTML. Наприклад, це директиви ngFor та ngIf.

I) <u>Вправа №1:</u>

ngClass та ngStyle

ngClass

Директива ngClass дозволяє визначити набір класів, які застосовуватимуться до елемента. В якості значень вона приймає набір класів у такому вигляді:

```
[ngClass]={
   "класс1": true/false,
   "класс2": true/false,
   .....................
}
```

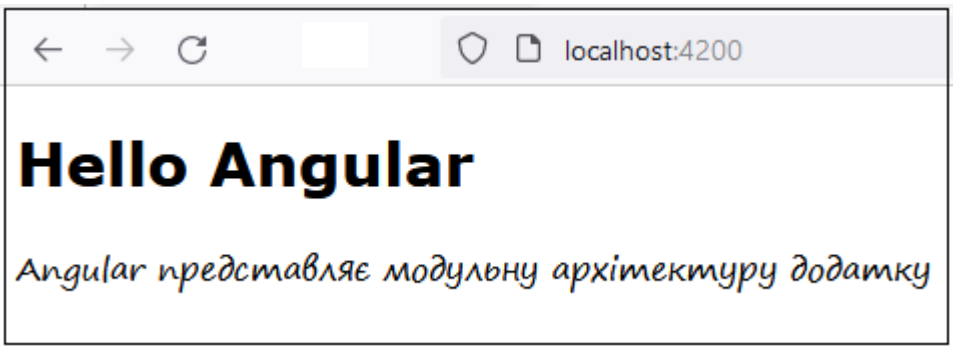
Наприклад, визначимо наступний компонент:

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

@Component({
   selector: 'my-app',
   template: `<div [ngClass]="{verdanaFont:true}">
                <h1>Hello Angular</h1>
                <p [ngClass]="{segoePrintFont:true}">
                   Angular представляет модульную архитектуру приложения
                </p>
             </div>`,
   styles: [
      `.verdanaFont{font-size:15px; font-family:Verdana;}
      .segoePrintFont{font-size:16px; font-family:"Segoe Print";}`
   ]
})
export class AppComponent { }
```

У секції styles у компонента визначено два класи, які встановлюють різні стильові властивості шрифту: verdanaFont та segoePrintFont.

У шаблоні для прив'язки класу до елемента застосовується директива [ngClass]="{verdanaFont:true}". Ця директива приймає js-об'єкт, у якому ключі – це назви класів. Цим назвам надаються булеві значення true (якщо клас застосовується) і false (якщо клас не застосовується). Тобто в даному випадку клас verdanaFont застосовуватиметься до всього блоку div.

Однак у блоці div є параграф, і ми, скажімо, хочемо, щоб до цього параграфа застосовувався інший клас. А за замовчуванням вкладений параграф успадкує стилі від батьківського блоку div і також застосує клас segoePrintFont, в якому можна перевизначити успадковані стилі.



Замість жорстко закодованих значень true/false ми можемо використовувати прив'язку до виразів:

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

@Component({
    selector: 'my-app',
    template: ` <div [ngClass]="{verdanaFont:isVerdana}">
                    <h1>Hello Angular</h1>
                    <p [ngClass]="{segoePrintFont:isSegoe}">
                        Angular представляє модульну архітектуру додатку
                    </p>
                </div>`,
    styles: [
        `.verdanaFont{font-size:13px; font-family:Verdana;}
        .segoePrintFont{font-size:14px; font-family:"Segoe Print";}`
    ]
})
export class AppComponent {
    isVerdana = true;
    isSegoe = true;
}
```

Як альтернативу ми можемо використовувати такі вирази прив'язки:

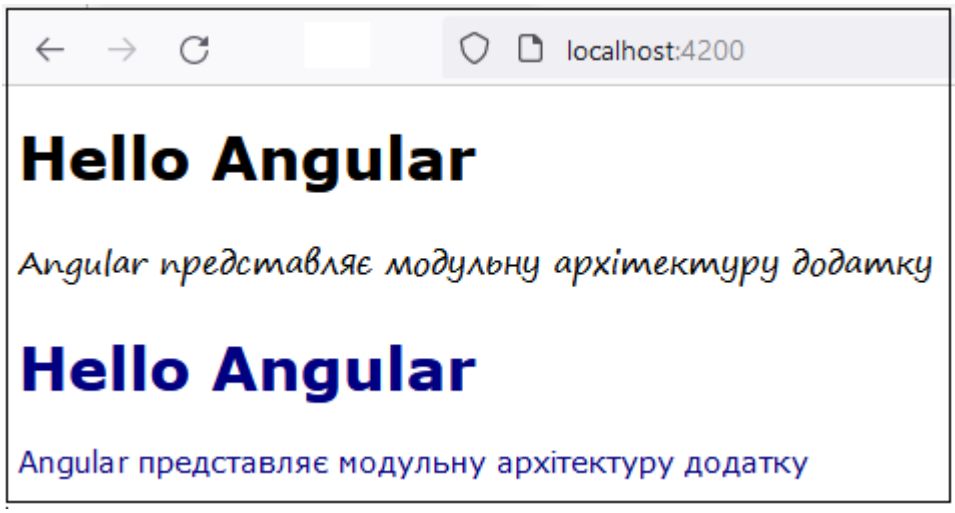
```
<div [class.verdanaFont]="true">
    <h1>Hello Angular</h1>
    <p [class.verdanaFont]="false" [class.segoePrintFont]="true">
        Angular представляє модульну архітектуру додатку
    </p>
</div>
```

Вираз [class.verdanaFont]="true" вказує, що клас verdanaFont буде застосовуватись для цього елемента.

Однак, за допомогою ngClass ми можемо задати цілий набір класів, які застосовуються до елемента:

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

@Component({
   selector: 'my-app',
   template: ` <div [ngClass]="{verdanaFont:isVerdana}">
   <h1>Hello Angular</h1>
   <p [ngClass]="{segoePrintFont:isSegoe}">
      Angular представляє модульну архітектуру додатку
   </p>
</div>
                  <div [ngClass]="currentClasses">
                  <h1>Hello Angular</h1>
                  <p>
                     Angular представляє модульну архітектуру додатку
                  </p>
               </div>`,
   styles: [
      `.verdanaFont{font-size:13px; font-family:Verdana;}
       .segoePrintFont{font-size:16px; font-family:"Segoe Print";}
      .navyColor{color:navy;}`
   ]
})
export class AppComponent {

   isVerdana = true;
   isSegoe = true;
   isNavy = true;

   currentClasses={
      verdanaFont: this.isVerdana,
      navyColor: this.isNavy
   }
}
```

В даному випадку для елемента встановлюються два класи verdanaFont і navyColor.



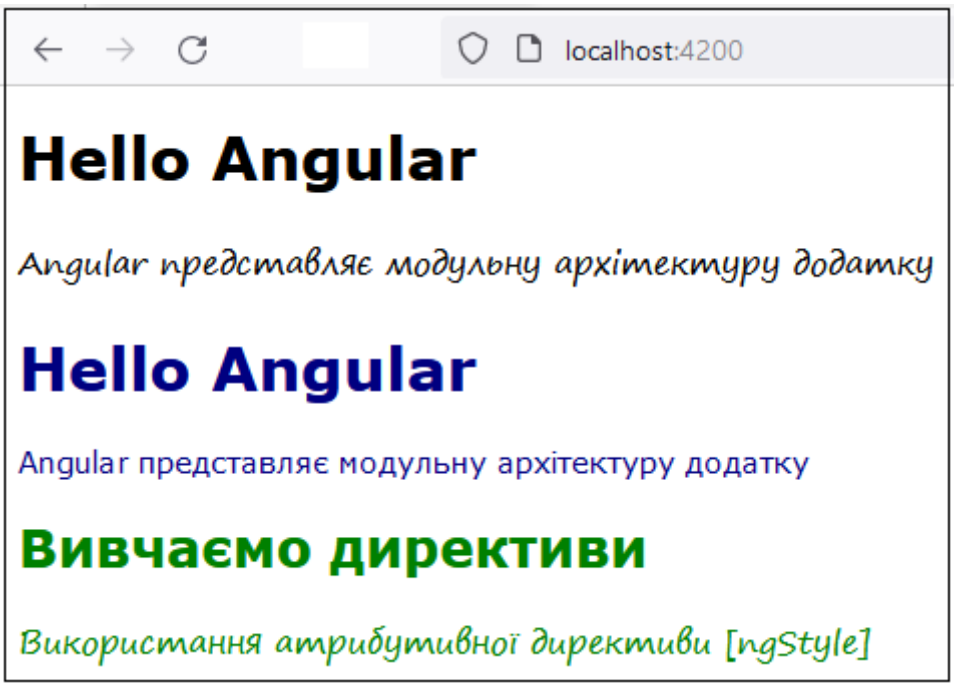
ngStyle

Директива ngStyle дозволяє встановити набір стилів, які застосовуються до елемента. Як значення директива приймає js-об'єкт, у якому ключі - назви властивостей CSS:

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

@Component({
   selector: 'my-app',
   template: `
            <div [ngClass]="{verdanaFont:isVerdana}">
            <h1>Hello Angular</h1>
            <p [ngClass]="{segoePrintFont:isSegoe}">
            Angular представляє модульну архітектуру додатку
            </p>
            </div>
              <div [ngClass]="currentClasses">
              <h1>Hello Angular</h1>
              <p>
                 Angular представляє модульну архітектуру додатку
              </p>
            </div>

             <div [ngStyle]="{'font-size':'13px', 'font-family':'Verdana', 'color':'green'}">
          <h1>Вивчаємо директиви</h1>
          <p [ngStyle]="{'font-size':'16px', 'font-family':'Segoe Print'}">
          Використання атрибутивної директиви [ngStyle]
          </p>
          </div>
   `,
      styles: [
         `.verdanaFont{font-size:13px; font-family:Verdana;}
          .segoePrintFont{font-size:16px; font-family:"Segoe Print";}
```

```
        .navyColor{color:navy;}`
    ]
})
export class AppComponent {
  isVerdana = true;
   isSegoe = true;
   isNavy = true;
   currentClasses={
      verdanaFont: this.isVerdana,
      navyColor: this.isNavy
    }
}
```

Отримаємо:



Аналогічно для встановлення стилів можна використовувати властивості об'єкта style:

```
<div [style.fontSize]="'13px'" [style.fontFamily]="'Verdana'">
   <h1>Hello Angular 16</h1>
   <p [style.fontSize]="'14px'" [style.fontFamily]="'Segoe Print'">
      Angular 16 представляет модульную архитектуру приложения
   </p>
</div>
```

Також ми можемо встановити прив'язку.

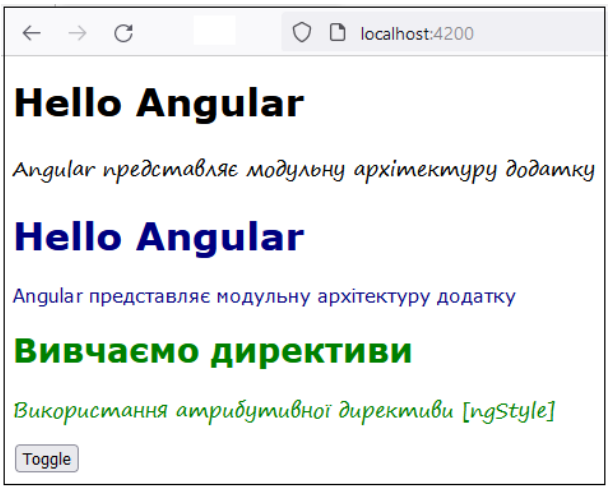
Динамічна зміна стилів

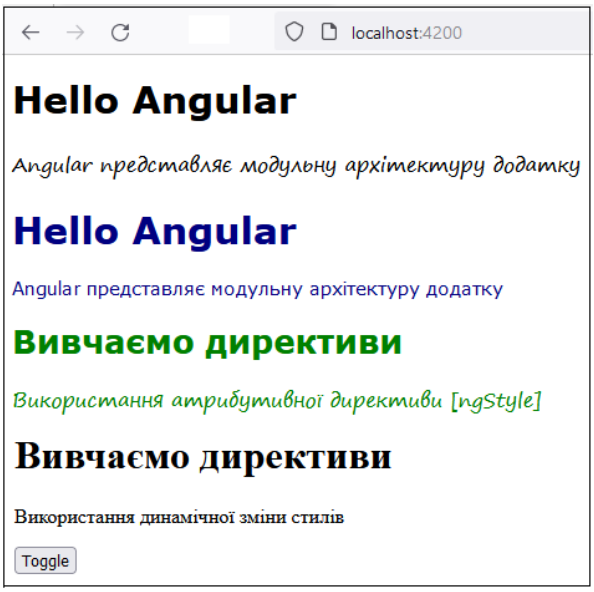
Директиви ngClass та ngStyle дозволяють встановлювати прив'язку до виразів, завдяки чому ми можемо динамічно змінювати стилі чи класи. Наприклад:

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

@Component({
   selector: 'my-app',
   template: `
   <div [ngClass]="{verdanaFont:isVerdana}">
   <h1>Hello Angular</h1>
   <p [ngClass]="{segoePrintFont:isSegoe}">
      Angular представляє модульну архітектуру додатку
   </p>
</div>

               <div [ngClass]="currentClasses">
               <h1>Hello Angular</h1>
               <p>
                  Angular представляє модульну архітектуру додатку
               </p>
            </div>

            <div [ngStyle]="{'font-size':'13px', 'font-family':'Verdana', 'color':'green'}">
               <h1>Вивчаємо директиви</h1>
               <p [ngStyle]="{'font-size':'16px', 'font-family':'Segoe Print'}">
                  Використання атрибутивної директиви [ngStyle]
               </p>
            </div>

               <div [ngClass]="{invisible: visibility}">
               <h1> Вивчаємо директиви </h1>
               <p>
                  Використання динамічної зміни стилів
               </p>
            </div>
            <button (click)="toggle()">Toggle</button>`,
   styles: [ `
      .verdanaFont{font-size:15px; font-family:Verdana;}
       .segoePrintFont{font-size:16px; font-family:"Segoe Print";}
       .navyColor{color:navy;}
      .invisible{display:none;}
`]
})
export class AppComponent {
      isVerdana = true;
   isSegoe = true;
   isNavy = true;
   currentClasses={
      verdanaFont: this.isVerdana,
```

```
    navyColor: this.isNavy
  visibility: boolean = true;
  // переключаем переменную
  toggle(){
    this.visibility=!this.visibility;
  }
}
```

Вираз [ngClass]="{invisible: visibility}" встановлює для класу invisible прив'язку до значення змінної visibility. Після натискання кнопки ми можемо перемикати цю властивість s керувати видимістю блоку.





В якості альтернативи також можна було б використовувати такий вираз:

```
<div [class.invisible]="visibility">
```

Або також можна було б написати так:

```
<div [style.display]="visibility?'block':'none'">
```

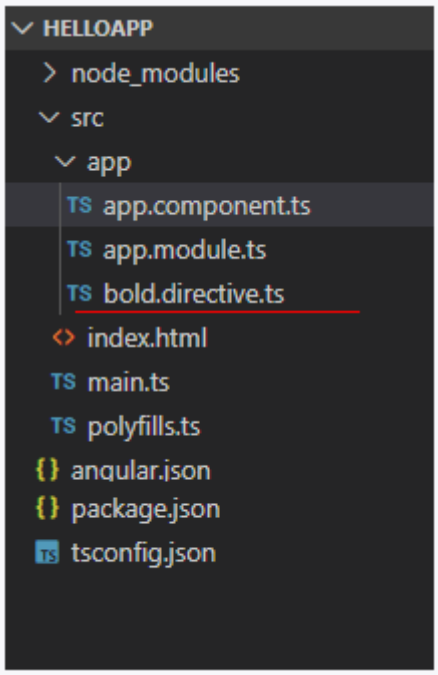
II) <u>Вправа 2:</u>

Створення атрибутивних директив

Атрибутивні директиви змінюють поведінку елемента, якого вони застосовуються. Наприклад, директива ngClass дозволяє встановити для елемента клас CSS. При цьому сама директива застосовується до елемента у вигляді атрибуту:

```
<div [ngClass]="{verdanaFont:true}">
```

І за потреби ми можемо самі створювати якісь свої директиви атрибутів для певних цілей. Отже, створимо свою директиву. Додамо до папки src/app новий файл, який назвемо bold.directive.ts:



Визначимо у файлі bold.directive.ts наступний код:

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

@Directive({
    selector: '[bold]'
})
export class BoldDirective{

    constructor(private elementRef: ElementRef){

        this.elementRef.nativeElement.style.fontWeight = "bold";
    }
}
```

Директива – це звичайний клас на TS, до якого застосовується декоратор Directive, відповідно нам треба імпортувати цю директиву з angular/core. Крім того, тут імпортується клас "ElementRef". Він представляє посилання на елемент, до якого застосовуватиметься директива.

При застосуванні декоратора @Directive необхідно визначити селектор CSS, з яким буде асоційовуватися директива. Селектор CSS для атрибута повинен визначатися у квадратних дужках. В даному випадку як селектор виступає [bold].

Сам декоратор @Directive застосовується до класу, який називається BoldDirective. Це, власне, і є клас директиви, який визначає її логіку.

Для отримання елемента, до якого застосовується ця директива, у класі визначено конструктор, який має один параметр: private elementRef: ElementRef. Через цей параметр Angular передаватиме або інжектуватиме той елемент із шаблону, в якому застосовується директива.

Оскільки параметр визначено з ключовим словом private, то для нього буде створюватися однойменна приватна змінна, через яку ми можемо отримати об'єкт ElementRef і зробити з ним будь-які маніпуляції. Зокрема, тут йде звернення до вкладеної властивості nativeElement, через яку у елемента встановлюється жирний шрифт:

```
this.elementRef.nativeElement.style.fontWeight = "bold";
```

Тепер візьмемо код головного компонента та застосуємо директиву:

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

@Component({
   selector: 'my-app',
   template: ` <div>
               <p bold>Вивчаю директиви</p>
               <p>Створення атрибутивних директив</p>
             </div>`
})
export class AppComponent {}
```

Тут визначено два параграфи, і до першого їх застосовується директива. Оскільки в коді директиви було визначено селектор "[bold]", то щоб її застосувати, в коді елемента застосовується даний селектор.

Але сама собою директива не запрацює. Нам ще треба її підключити в модулі програми - класі AppModule:

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent }   from './app.component';
import { BoldDirective} from './bold.directive';

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

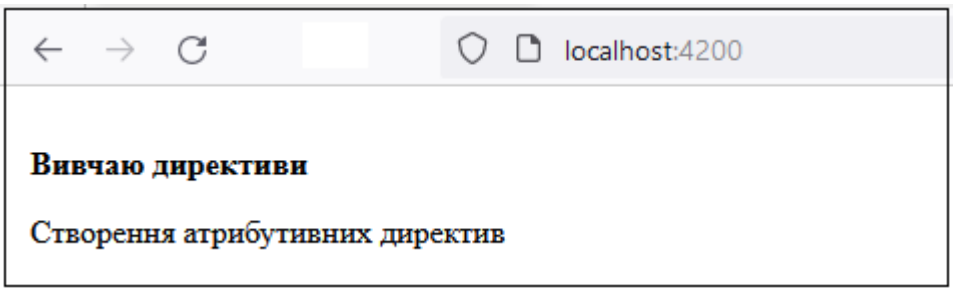
Як і компоненти, директиви також треба спочатку імпортувати з файлу, де вони оголошені:

```
import { BoldDirective } from './bold.directive';
```

Потім вона додається до секції declarations:

```
declarations: [ AppComponent, BoldDirective],
```

І якщо ми запустимо програму, то побачимо застосування директиви до першого параграфу:



Для керування стилізацією елемента вище цей елемент витягувався через об'єкт ElementRef у конструкторі директиви, і для нього встановлювалися стильові властивості. Однак набагато зручніше для керування стилем використовувати рендерер. Так, створимо нову директиву italic.directeve.ts в такий спосіб:

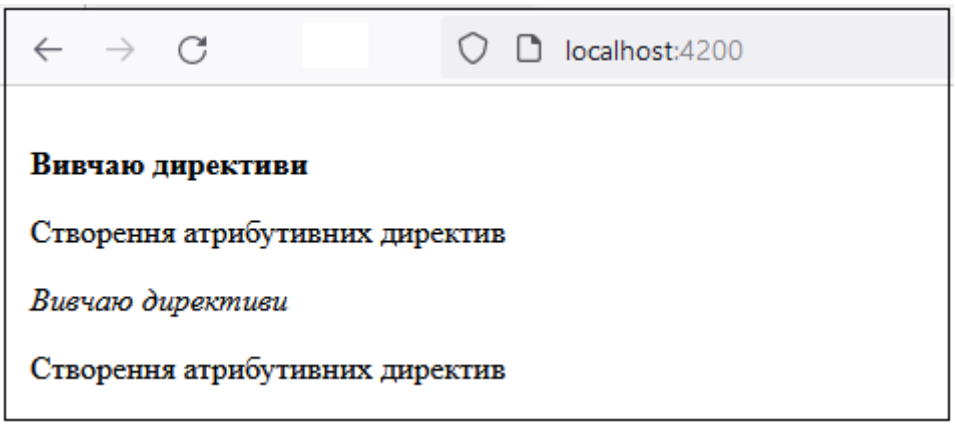
```
import {Directive, ElementRef, Renderer2} from '@angular/core';

@Directive({
   selector: '[italic]'
})
export class ItalicDirective{

   constructor(private elementRef: ElementRef, private renderer: Renderer2){

      this.renderer.setStyle(this.elementRef.nativeElement, "font-style", "italic");
   }
}
```

Renderer2 представляє сервіс, який також при виклику директиви автоматично передається до її конструктора, і ми можемо використовувати цей сервіс для стилізації елемента. Доповнимо шаблон компонента app.component.ts таким чином:

```
<div>
      <p italic>Вивчаю директиви</p>
```

```
        <p>Створення атрибутивних директив</p>
    </div>
```

Та добавимо нову директиву до модуля app.module.ts. В результаті роботи отримаємо:



Взаємодія з користувачем, HostListener та HostBinding

HostListener - декоратор, який об'являє подію DOM для прослуховування і надає метод обробника для виконання, коли ця подія відбувається.

Крім простої установки значень, атрибутивна директива може взаємодіяти з користувачем. Для цього також може використовуватися декоратор HostListener.

Так, створимо нову директиву mousebold.directive.ts та додамо до неї взаємодію з користувачем:

```
import {Directive, ElementRef, Renderer2, HostListener} from '@angular/core';

@Directive({
   selector: '[mousebold]'
})
export class MouseboldDirective{

   constructor(private element: ElementRef, private renderer: Renderer2){

      this.renderer.setStyle(this.element.nativeElement, "cursor", "pointer");
   }

   @HostListener("mouseenter") onMouseEnter() {
      this.setFontWeight("bold");
   }

   @HostListener("mouseleave") onMouseLeave() {
      this.setFontWeight("normal");
   }

   private setFontWeight(val: string) {
```

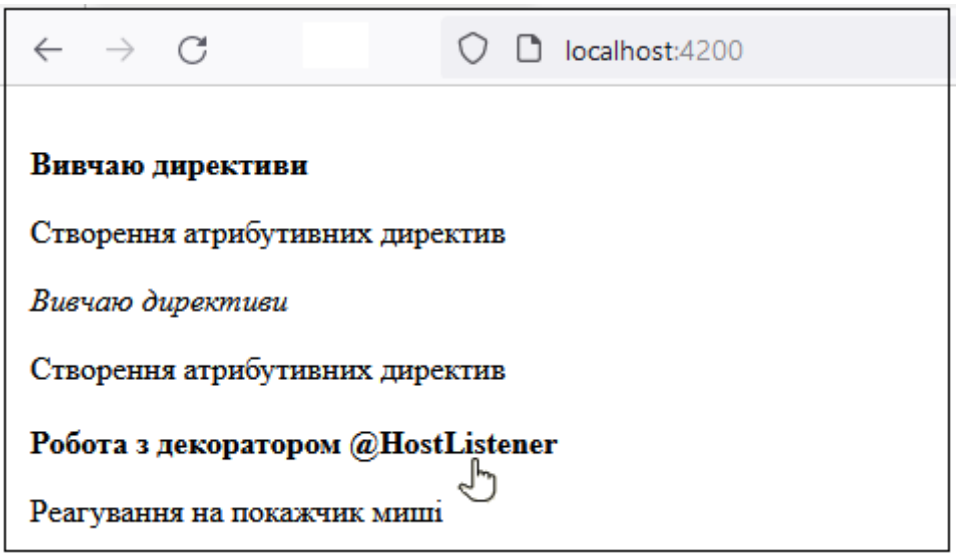
```
        this.renderer.setStyle(this.element.nativeElement, "font-weight", val);
    }
}
```

В шаблоні додамо новий для демонстрації роботи нової директиви:

```
<div>
    <p mousebold>Робота з декоратором @HostListener</p>
    <p>Реагування на покажчик миші</p>
</div>
```

Та добавимо нову директиву до модуля app.module.ts.

Декоратор **@HostListener** дозволяє пов'язати події DOM та методи директиви. Зокрема, в декоратор передається назва події, за якою викликатиметься метод. У даному випадку ми прив'язуємо подію mouseenter (наведення покажчика миші на елемент) і mouseleave (уведення покажчика миші з елемента) до методу setFontWeight(), який встановлює стильову властивість font-weight у елемента. Якщо ми наводимо на елемент, то встановлюється виділення жирним. При відведенні миші виділення скидається.



HostBinding

Ще один декоратор – HostBinding дозволяє пов'язати звичайну властивість класу з властивістю елемента, до якого застосовується директива. Наприклад, створимо нову директиву mouseitalic.directive.ts таким чином:

```
import {Directive, HostListener, HostBinding} from '@angular/core';

@Directive({
  selector: '[mouseitalic]'
```

```
})
export class MouseitalicDirective{

   private fontStyle = "normal";

   @HostBinding("style.font-style") get getFontStyle(){

      return this.fontStyle;
   }

   @HostBinding("style.cursor") get getCursor(){
      return "pointer";
   }

   @HostListener("mouseenter") onMouseEnter() {
      this.fontStyle ="italic";
   }

   @HostListener("mouseleave") onMouseLeave() {
      this.fontStyle = "normal";
   }
}
```

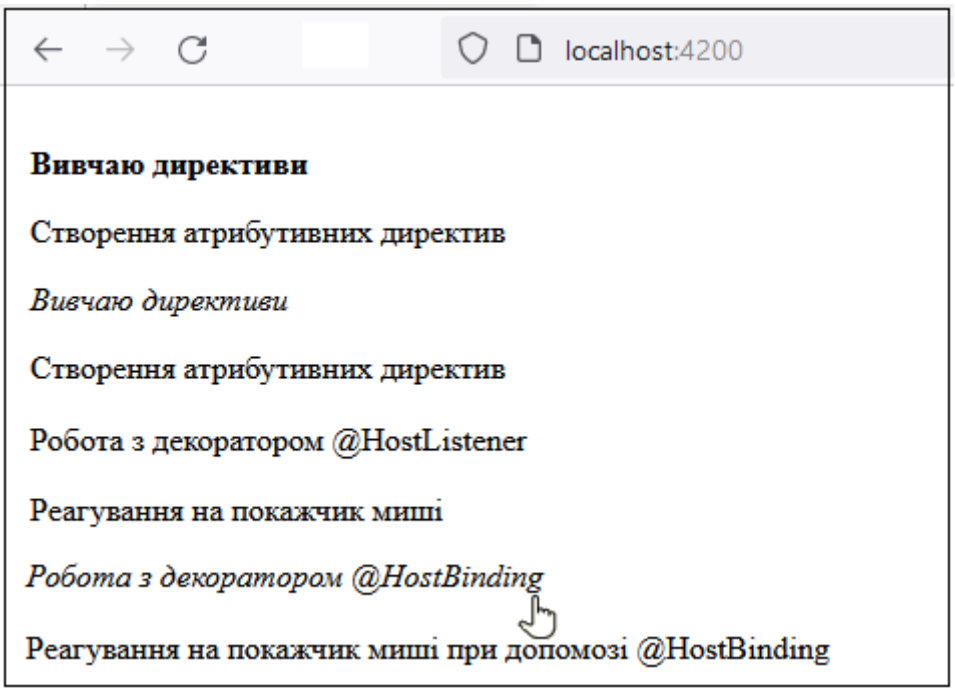
В шаблоні додамо новий для демонстрації роботи нової директиви:

```
 <div>
       <p mouseitalic>Робота з декоратором @HostListener</p>
       <p>Реагування на покажчик миші при допомозі @HostBinding</p>
</div>
```

Та добавимо нову директиву до модуля app.module.ts.

Інструкція @HostBinding("style.font-style") get getFontStyle() пов'язує з властивістю "style.fontStyle" значення, яке повертається цим гетером getFontStyle. А він повертає значення властивості fontStyle, яке також змінюється при наведенні покажчика миші.

В результаті отримаємо:



Властивість host

Замість застосування декораторів HostListener та HostBinding для реагування директиви на дії користувача, ми можемо визначити обробники подій у декораторі Directive за допомогою його властивості host. Так, створимо нову директиву hostmousebold.directive.ts так:

```
import {Directive, ElementRef, Renderer2} from '@angular/core';

@Directive({
    selector: '[hostmousebold]',
    host: {
        '(mouseenter)': 'onMouseEnter()',
        '(mouseleave)': 'onMouseLeave()'
    }
})
export class HostmouseboldDirective{

    constructor(private element: ElementRef, private renderer: Renderer2){

        this.renderer.setStyle(this.element.nativeElement, "cursor", "pointer");
    }

    onMouseEnter(){
        this.setFontWeight("bold");
    }
    onMouseLeave(){
        this.setFontWeight("normal");
    }
    private setFontWeight(val: string) {
```

```
        this.renderer.setStyle(this.element.nativeElement, "font-weight", val);
    }
}
```

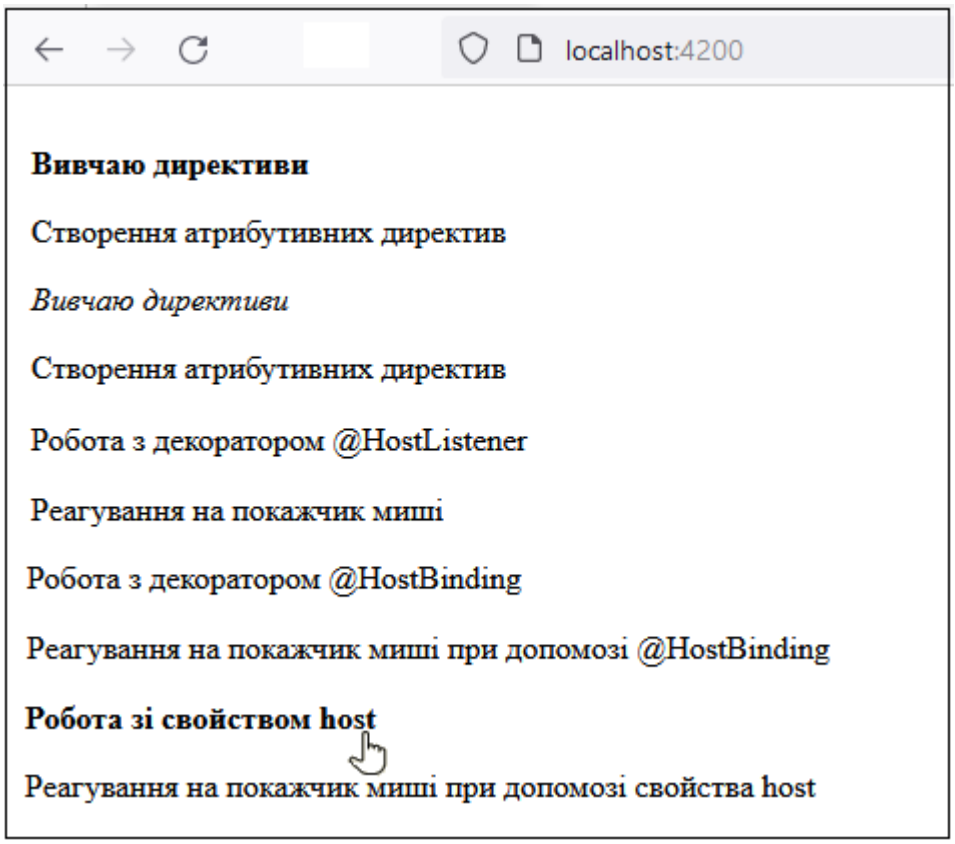
В шаблоні додамо новий для демонстрації роботи нової директиви:

```
<p hostmousebold>Робота з декоратором @HostListener</p>
<p>Реагування на покажчик миші при допомозі @HostBinding</p>
</div>
```

Та добавимо нову директиву до модуля app.module.ts.

Результат роботи директиви в цьому випадку буде аналогічним, тільки тепер всі події та пов'язані з ними обробники визначаються за допомогою параметра host:

```
host: {
   '(mouseenter)': 'onMouseEnter()',
   '(mouseleave)': 'onMouseLeave()'
}
```



III) <u>Вправа 3:</u>

Отримання параметрів у директивах

Директива як компонент може отримувати деякі вхідні параметри ззовні. Для цього також використовують декоратор Input. Отже, створимо директиву ValueDirective і,

припустимо, ми хочемо, щоб у тексті при наведенні також змінювалася висота шрифту. Але при цьому, щоб потрібну висоту шрифту можна було б задати ззовні директиви. Для цього напишемо так:

```
import {Directive, HostListener, Input, HostBinding} from '@angular/core';

@Directive({
   selector: '[valuesize]'
})
export class ValueDirective{

   @Input() selectedSize = "18px";
   @Input() defaultSize = "16px";

   private fontSize : string;
   private fontWeight = "normal";
   constructor(){
      this.fontSize = this.defaultSize;
   }
   @HostBinding("style.fontSize") get getFontSize(){

      return this.fontSize;
   }
   @HostBinding("style.fontWeight") get getFontWeight(){

      return this.fontWeight;
   }
   @HostBinding("style.cursor") get getCursor(){
      return "pointer";
   }
   @HostListener("mouseenter") onMouseEnter() {
      this.fontWeight ="bold";
      this.fontSize = this.selectedSize;
   }
   @HostListener("mouseleave") onMouseLeave() {
      this.fontWeight = "normal";
      this.fontSize = this.defaultSize;
   }
}
```

У цьому випадку визначаються два вхідні параметри:

```
@Input() selectedSize = "18px";
@Input() defaultSize = "16px";
```

Параметр selectedSize відповідає за висоту шрифту при наведенні миші, а параметр defaultSize встановлює висоту шрифту, коли покажчик миші знаходиться поза межами елемента.

Тепер використовуємо ці параметри, змінивши код компонента:

```
import { Component} from '@angular/core';
@Component({
   selector: 'my-app',
   template: `<div>
            <p valuesize selectedSize="28px" [defaultSize]="'14px'">Hello Angular</p>
            <p>Angular представляє модульну архітектуру додатку</p>
            </div>`
})
export class AppComponent {}
```

При застосуванні директиви ми можемо вказати всі вхідні параметри та їх значення:

```
<p valuesize selectedSize="28px" [defaultSize]="'14px'">Hello Angular</p>
```

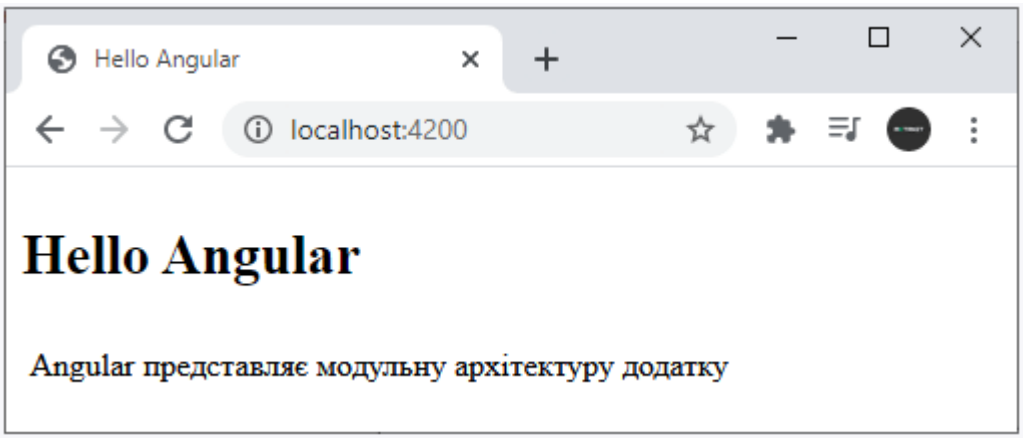
При цьому є дві варіації застосування праметрів. Або назви параметрів беруться у квадратні дужки, а їхні значення додатково беруться у одинарні лапки:

```
[defaultSize]="'14px'"
```

Або назви параметрів передаються без дужок, які значення полягають у подвійні лапки:

```
selectedSize="28px"
```

І при наведенні на елемент автоматично змінюватиметься також і висота шрифту:



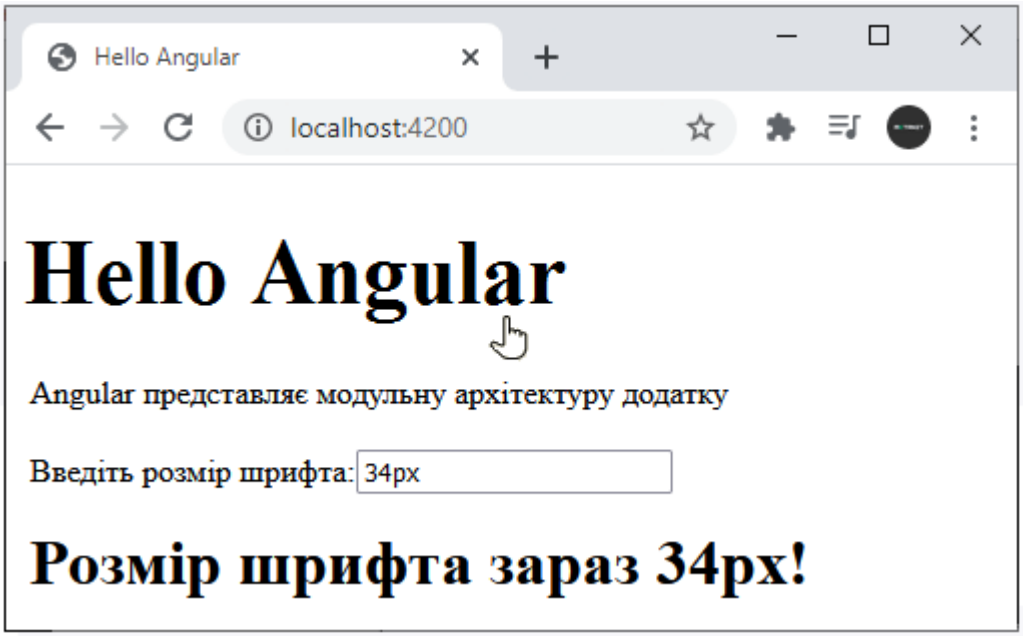
Тепер підемо далі і змінимо перший вхідний параметр:

```
@Input("valuesize") selectedSize = "18px";
```

Тут декоратор Input передається селектор директиви - valuesize. Тому, щоб встановити цей параметр у шаблоні компонента, ми можемо безпосередньо використовувати ім'я директиви:

```
<p [valuesize]="'28px'" [defaultSize]="'14px'">Hello Angular</p>
```

Виконати самостійно: змінити директиву так, щоб можливо було задавати розмір шрифта у полі <input> батьківського компонента.



IV) Вправа 4:

Структурні директиви ngIf, ngFor, ngSwitch

Структурні директиви змінюють структуру DOM за допомогою додавання чи видалення HTML-елементів. Розглянемо три структурні директиви: ngIf, ngSwitch та ngFor.

ngIf

Директива ngIf дозволяє видалити або, навпаки, додати елемент за певної умови. Наприклад, визначимо наступний компонент:

```
import { Component} from '@angular/core';
@Component({
   selector: 'my-app',
   template: `<p *ngIf="condition">
               Привіт світ!
              </p>
              <p *ngIf="!condition">
               Пока світ!
              </p>
```

```
            <button (click)="toggle()">Toggle</button>`
})
export class AppComponent {

   condition: boolean=true;

   toggle(){
      this.condition=!this.condition;
   }
}
```

Залежно від значення властивості condition буде відображатися або перший або другий параграф.

Ми можемо задавати альтернативні вирази за допомогою директиви **ng-template**. Так, попередній приклад буде аналогічним наступному:

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

@Component({
   selector: 'my-app',
   template: `
            <p *ngIf="condition">
              Привіт світ!
            </p>
            <p *ngIf="!condition">
              Пока світ!
            </p>

            <p *ngIf="condition;else unset">
              Привіт Angular!
            </p>
            <ng-template #unset>
              <p>Пока Angular! </p>
            </ng-template>
            <button (click)="toggle()">Toggle</button>`
})
export class AppComponent {

   condition: boolean=true;

   toggle(){
      this.condition=!this.condition;
   }
}
```

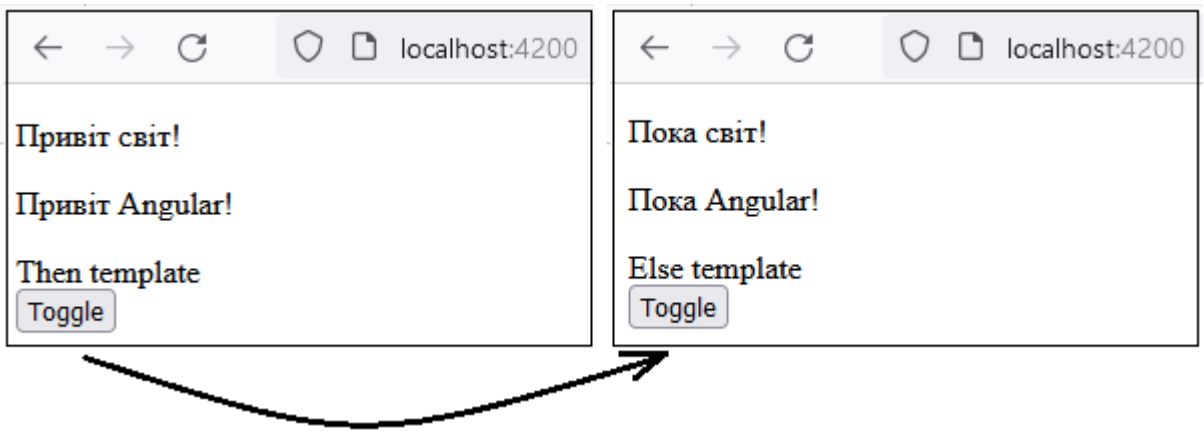
Вираз *ngIf="condition;else unset" вказує, що якщо condition дорівнює false, то спрацьовує блок <ng-template #unset>.

Або можна визначити більш витончену логіку. Так, змінимо шаблон компонента в такий спосіб:

```
template: ` <div *ngIf="condition; then thenBlock else elseBlock"></div>
        <ng-template #thenBlock>Then template</ng-template>
        <ng-template #elseBlock>Else template</ng-template>
        <br/>
        <button (click)="toggle()">Toggle</button>`
```

У разі, якщо умова дорівнює true, то відображається блок thenBlock, інакше відображається блок elseBlock.

Отримаємо наступний результат:



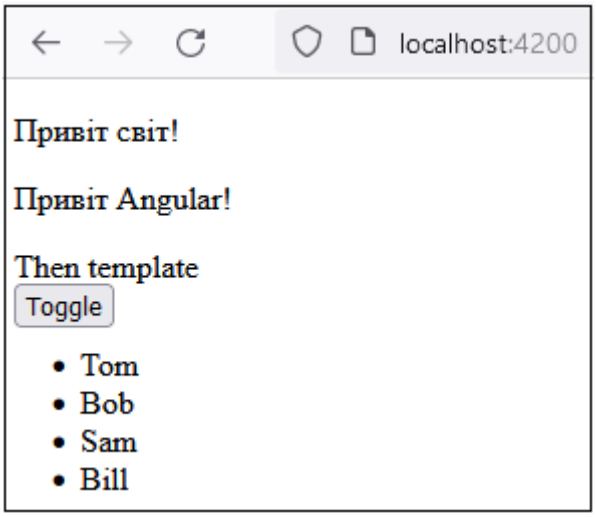
ngFor

Директива ngFor дозволяє перебрати елементи масиву в шаблоні. Наприклад:

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

@Component({
   selector: 'my-app',
   template: `
            <p *ngIf="condition">
              Привіт світ!
            </p>
            <p *ngIf="!condition">
              Пока світ!
            </p>
           <p *ngIf="condition;else unset">
            Привіт Angular!
           </p>
           <ng-template #unset>
            <p>Пока Angular! </p>
```

```
            </ng-template>
          <div *ngIf="condition; then thenBlock else elseBlock"></div>
          <ng-template #thenBlock>Then template</ng-template>
          <ng-template #elseBlock>Else template</ng-template>
<br/>
             <button (click)="toggle()">Toggle</button>

<ul>
              <li *ngFor="let item of items">{{item}}</li>
            </ul>`
})
export class AppComponent {
    condition: boolean=true;

   toggle(){
      this.condition=!this.condition;
   }

   items =["Tom", "Bob", "Sam", "Bill"];
}
```

Як значення директива набуває значення перебору аля-foreach: let item of items. Кожен елемент, що перебирається, поміщається в змінну item, яку ми можемо вивести на сторінку.



При переборі елементів нам доступний поточний індекс елемента через змінну index, яку ми можемо також використовувати. Наприклад:

```
<div>
   <p *ngFor="let item of items; let i = index">{{i+1}}.{{item}}</p>
</div>
```

Треба враховувати, що індексація йде з нуля, тому щоб у даному випадку відлік йшов з одиниці, до змінної і додається одиниця.

Символ зірочки та синтаксичний цукор

Можна помітити, що з використанням директив ngFor і ngIf перед ними ставиться символ зірочка. За фактом це не більше, ніж синтаксичний цукор, який спрощує застосування директиви. Так, визначення ngIf:

```
<p *ngIf="condition">
   Привіт світ
</p>
<p *ngIf="!condition">
   Пока світ
</p>
```

за фактом представлятиме наступний код:

```
<ng-template [ngIf]="condition">
   <p>
      Привіт світ
   </p>
</ng-template>
<ng-template [ngIf]="!condition">
   <p>
      Пока світ
   </p>
</ng-template>
```

У результаті параграф та його текст переміщаються всередину елемента <ng-template>. Сама директива поміщається в тег <ng-template>, у якому застосовується прив'язка властивості. Бульове значення прив'язаної властивості вказує, чи потрібно відображати відповідний контент.

У результаті ми можемо вибирати або перший спосіб із зірочкою, який більш компактний, або другий спосіб із елементами ng-template.

Те саме стосується і директиви ngFor:

```
<ul>
   <li *ngFor="let item of items">{{item}}</li>
</ul>
```

Цей код буде еквівалентний наступному:

```
<ul>
   <ng-template ngFor let-item [ngForOf]="items">
      <li>{{item}}</li>
   </ng-template>
</ul>
```

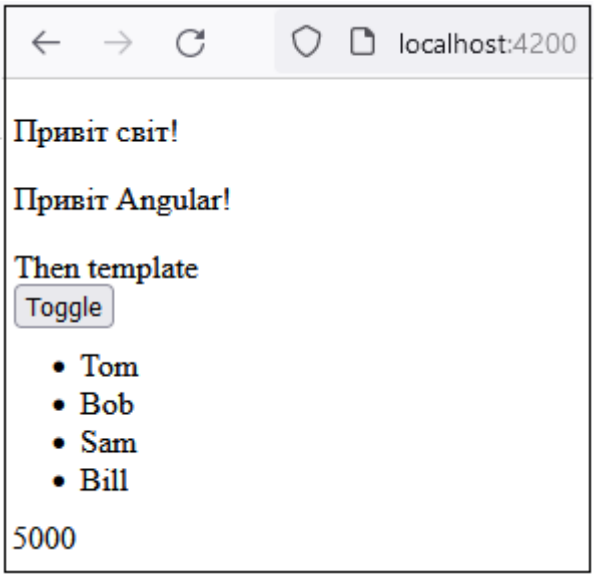
ngSwitch

За допомогою директиви ngSwitch можна вбудувати в шаблон конструкцію switch…case та в залежності від її результату виконання виводити той чи інший блок. Наприклад:

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

@Component({
   selector: 'my-app',
   template: `
             <p *ngIf="condition">
              Привіт світ!
             </p>
             <p *ngIf="!condition">
              Пока світ!
             </p>
            <p *ngIf="condition;else unset">
             Привіт Angular!
            </p>
            <ng-template #unset>
             <p>Пока Angular! </p>
            </ng-template>
          <div *ngIf="condition; then thenBlock else elseBlock"></div>
          <ng-template #thenBlock>Then template</ng-template>
          <ng-template #elseBlock>Else template</ng-template>
<br/>
             <button (click)="toggle()">Toggle</button>
             <ul>
              <li *ngFor="let item of items">{{item}}</li>
             </ul>
              <div [ngSwitch]="count">
              <ng-template ngSwitchCase="1">{{count * 10}}</ng-template>
              <ng-template ngSwitchCase="2">{{count * 100}}</ng-template>
              <ng-template ngSwitchDefault>{{count * 1000}}</ng-template>
             </div>
`
})
export class AppComponent {
    condition: boolean=true;

   toggle(){
      this.condition=!this.condition;
   }
   items =["Tom", "Bob", "Sam", "Bill"];
   count: number = 5;
}
```

Директива ngSwitch в якості значення набуває деякого виразу. В даному випадку це властивість count. В елемент ng-template поміщується інструкція ngSwitchCase, яка порівнює значення виразу із ngSwitch з іншим виразом. Якщо обидва вирази рівні, використовується даний елемент template. Інакше виконання переходить до наступних інструкцій ngSwitchCase. Якщо ж жодна з інструкцій ngSwitchCase не була виконана, викликається інструкція ngSwitchDefault.

В результаті маємо:



Створення структурних директив

Створимо найпростішу структурну директиву. Додамо до папки src/app новий файл while.directive.ts:

Цей файл міститиме директиву. Визначимо у файлі наступний код:

```
import { Directive, Input, TemplateRef, ViewContainerRef } from '@angular/core';

@Directive({ selector: '[while]' })
export class WhileDirective {

   constructor(private templateRef: TemplateRef<any>,
               private viewContainer: ViewContainerRef)
   { }

   @Input() set while(condition: boolean) {
      if (condition) {
        this.viewContainer.createEmbeddedView(this.templateRef);
      } else {
        this.viewContainer.clear();
      }
   }
}
```

Структурна директива має застосовувати декоратор Directive, до якого передається назва селектора директиви у квадратних дужках. У даному випадку селектор - " while ".

Для отримання доступу до шаблону директиви використовується об'єкт **TemplateRef**. Цей об'єкт автоматично передається у конструктор через механізм впровадження залежностей. Крім цього об'єкта у конструктор також передається об'єкт рендерера - **ViewContainerRef**. Ну і за допомогою модифікатора private для обох цих параметрів автоматично будуть створюватися локальні змінні, до яких ми потім зможемо звернутися.

```
constructor(private templateRef: TemplateRef<any>,
            private viewContainer: ViewContainerRef)
{ }
```

За допомогою вхідної властивості-сеттера, до якого застосовується декоратор Input, ми будемо отримувати ззовні деякі значення, які можуть використовуватися при створенні розмітки html. В даному випадку ми отримуємо ззовні деяке булеве значення:

```
@Input() set while(condition: boolean) {
   if (condition) {
      this.viewContainer.createEmbeddedView(this.templateRef);
   } else {
      this.viewContainer.clear();
   }
}
```

Якщо в даному випадку condition дорівнює true, то робимо рендеринг шаблону через виклик this.viewContainer.createEmbeddedView(this.templateRef);. У результаті на веб-сторінці з'явиться елемент, до якого застосовується дана директива.

В іншому випадку, якщо condition дорівнює false, то, навпаки, видаляємо елемент з розмітки за допомогою this.viewContainer.clear().

Тобто, за фактом ми отримали аналог директиви ngIf.

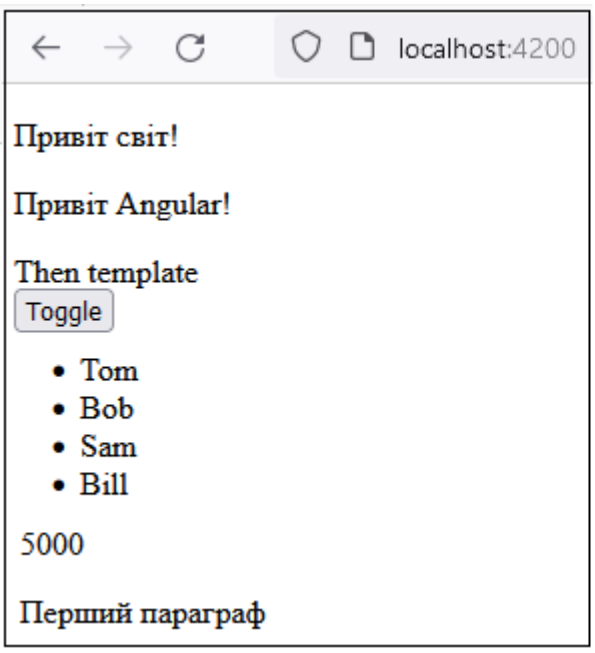
Далі застосуємо директиву в головному компоненті AppComponent:

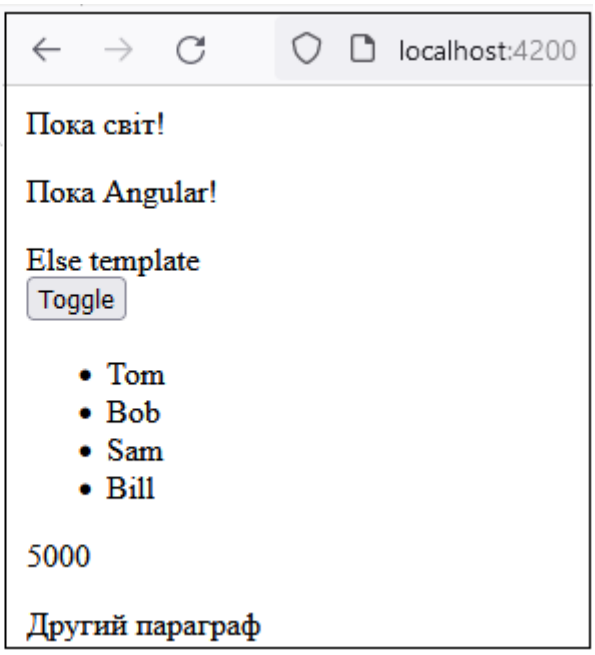
```
import { Component} from '@angular/core';

@Component({
   selector: 'my-app',
   template:`
            <p *ngIf="condition">
              Привіт світ!
            </p>
            <p *ngIf="!condition">
```

```
            Пока світ!
          </p>
          <p *ngIf="condition;else unset">
          Привіт Angular!
          </p>
          <ng-template #unset>
            <p>Пока Angular! </p>
          </ng-template>
        <div *ngIf="condition; then thenBlock else elseBlock"></div>
        <ng-template #thenBlock>Then template</ng-template>
        <ng-template #elseBlock>Else template</ng-template>
<br/>
            <button (click)="toggle()">Toggle</button>

            <ul>
              <li *ngFor="let item of items">{{item}}</li>
            </ul>
            <div [ngSwitch]="count">
            <ng-template ngSwitchCase="1">{{count * 10}}</ng-template>
            <ng-template ngSwitchCase="2">{{count * 100}}</ng-template>
            <ng-template ngSwitchDefault>{{count * 1000}}</ng-template>
          </div>
    <p *while="condition">
      Перший параграф
    </p>
    <p *while="!condition">
      Другий параграф
    </p>
`
})
export class AppComponent {
condition: boolean=true;

    toggle(){
        this.condition=!this.condition;
    }
    items =["Tom", "Bob", "Sam", "Bill"];
    count: number = 5;
}
```

Далі необхідно задіяти дану директиву в головному модулі AppModule. :

І після запуску програми за допомогою цієї директиви ми зможемо регулювати додавання або видалення певного блоку веб-сторінки:





V) **Виконати самостійно:**

1) Розробити Angular-додаток Directives5, в якому створти директиву SumDirective для отримання суми двох доданків з таким шаблоном:

```
<div *sum="let result from 20 and 30" >Сума = {{ result }}</div>
```

2) Розробити Angular-додаток Directives6, в якому створити директиву OtherIfDirective. Директива OtherIfDirective робить протилежне NgIf. NgIf відображає вміст шаблона, коли умова дорівнює true. OtherIfDirective повинна відображати вміст, коли умова дорівнює false. Також в шаблоні встановити кнопку <button>, при активізації якої змінюється стан умови condition з false на true і навпаки.

```
<p *appOtherIf="condition" class="otherif a">
  (A) Condition is false.
</p>

<p *appOtherIf="!condition" class=" otherif b">
  (B) Although the condition is true, this paragraph is displayed.
</p>
```

VI) Зробити звіт по роботі. Звіт повинен бути не менше 8 сторінок без титульного аркуша (шрифт Times New Roman, 14, полуторний інтервал). Титульний аркуш приводиться у додатку. Звіт повинен містити наступні розділи:

a) Директиви: призначення, приклади використання;

b) Огляд атрибутивних директив;

c) Огляд структурних директив;
d) Огляд всіх структурних блоків Angular-додатку Directives5. Детальний огляд директиви SumDirective;
e) Огляд всіх структурних блоків Angular-додатку Directives6. Детальний огляд директиви OtherIfDirective;

VII) Angular-додатки Directives1 та Directives2 розгорнути на платформі Firebase у проектах з ім’ям «ПрізвищеГрупаLaba5-1» та «ПрізвищеГрупаLaba5-2», наприклад «KovalenkoIP01Laba5-1» та «KovalenkoIP01Laba5-2».

ДОДАТОК

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт по лабораторній роботі №____

_____

назва лабораторної роботи

з дисципліни: «Реактивне програмування»

Студент: _____

Група: _____

Дата захисту роботи: _____

Викладач: доц. Полупан Юлія Вікторівна

Захищено з оцінкою: _____

Київ, 2023