МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Лабораторна робота № 2

з дисципліни «Реактивне програмування»

на тему:

«Робота з компонентами. Взаємодія між компонентами. Прив’язка до подій дочірнього компоненту. Життєвий цикл компоненту»

Перевірила

доцент

Полупан Ю. В.

Виконав

студент ФІОТ

групи ІС-01

Адамов Денис

Київ 2023

Лабораторна робота №2

Тема: Навчитися працювати з компонентами в Angular.
Завдання: Створити п’ять Angular-додатків під назвою**:**
I) Components1 (вправи 1-6). Виконати відповідні вправи;
II) Components2 (вправи 7-8). Виконати відповідні вправи;
III) Components3 (вправа 9). Виконати відповідні вправи;
IV) Components4 (вправа 10). Виконати відповідні вправи;
V) Components5 (вправа 11). Виконати відповідні вправи;
VI) Зробити звіт по роботі;
VII) Angular-додатки Components1 та Components5 розгорнути на платформі Firebase у проектах з ім’ям «ПрізвищеГрупаLaba2-1» та «ПрізвищеГрупаLaba2-5», наприклад «KovalenkoIP01Laba2-1» та «KovalenkoIP01Laba2-5».

Вправи:
1) Створення застосунку з назвою «Components1»
2) Стилі та шаблони компонента.
3) Селектор: host. Призначення та використання.
4) Підключення зовнішніх файлів стилів та шаблонів.
5) ng-content: призначення та використання.
6) Взаємодія між компонентами. Передача даних у дочірній компонент.
7) Прив'язка до сетера.
8) Прив'язка до подій дочірнього компонента.
9) Двостороння прив'язка.
10) Життєвий цикл компоненту.
11) Реалізація всіх методів

Components1

2) Стилі та шаблони компонента

Параметр styles містить набір стилів, які використовуватимуться компонентом.

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

@Component({
  selector: 'app-root',
  template: `<label>Введіть ім’я:</label>
    <input [(ngModel)]="name" placeholder="name" />
    <h1>Ласкаво просимо {{ name }}!</h1>

    <h2>Hello Angular</h2>
    <p>Angular 16 представляє модульну архітектуру додатку</p>`,

  styles: [
    `
      h2,
      h3 {
        color: navy;
      }
      p {
        font-size: 13px;
        font-family: Verdana;
      }
    `,
  ],
})
export class AppComponent {
  name: '';
}
```

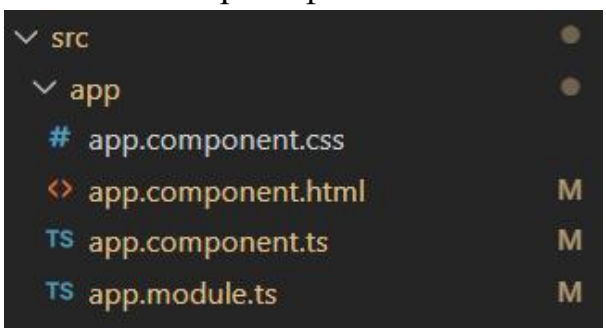
Селектор :host

Селектор :host посилається на елемент, у якому хоститься компонент. І селектор :host дає змогу застосувати стилі до цього елементу.

```
styles: [
  `
    h1,
    h2 {
      color: navy;
    }
    p {
      font-size: 13px;
    }
    :host {
      font-family: Verdana;
      color: #555;
    }
  `,
],
```

3) Підключення зовнішніх файлів стилів та шаблонів

Код самого компонента стає простіше і чистіше за рахунок винесення коду css і html в окремі файли:



lab2 > components1 > src > app > app.component.html > ...

Go to component

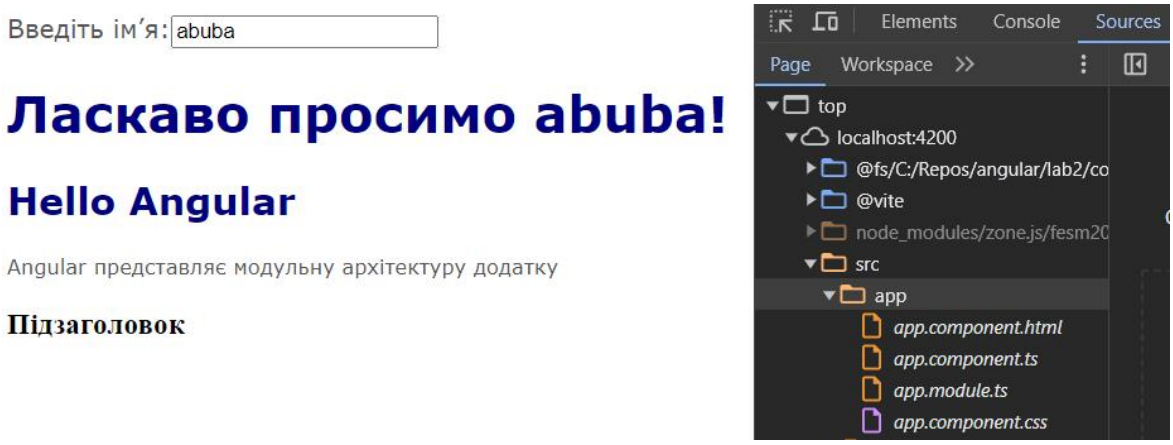
```
1  <label>Введіть ім’я:</label>
2  <input [(ngModel)]="name" placeholder="name" />
3  <h1>Ласкаво просимо {{ name }}!</h1>
4  <h2>Hello Angular</h2>
5  <p>Angular представляє модульну архітектуру додатку</p>
6
```

lab2 > components1 > src > app > # app.component.css

```
1  h1,
2  h2 {
3    color: navy;
4  }
5  p {
6    font-size: 13px;
7  }
8  :host {
9    font-family: Verdana;
10   color: #555;
11 }
```

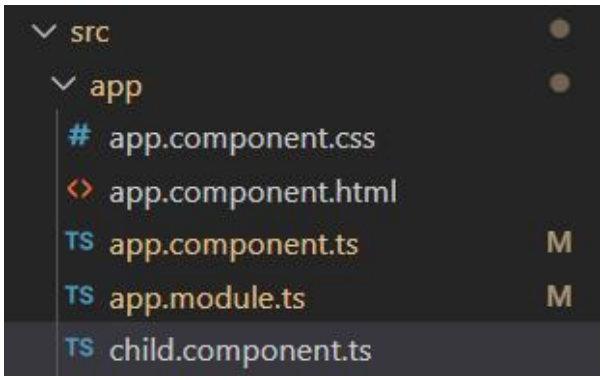
lab2 > components1 > src > app > TS app.component.ts > ...

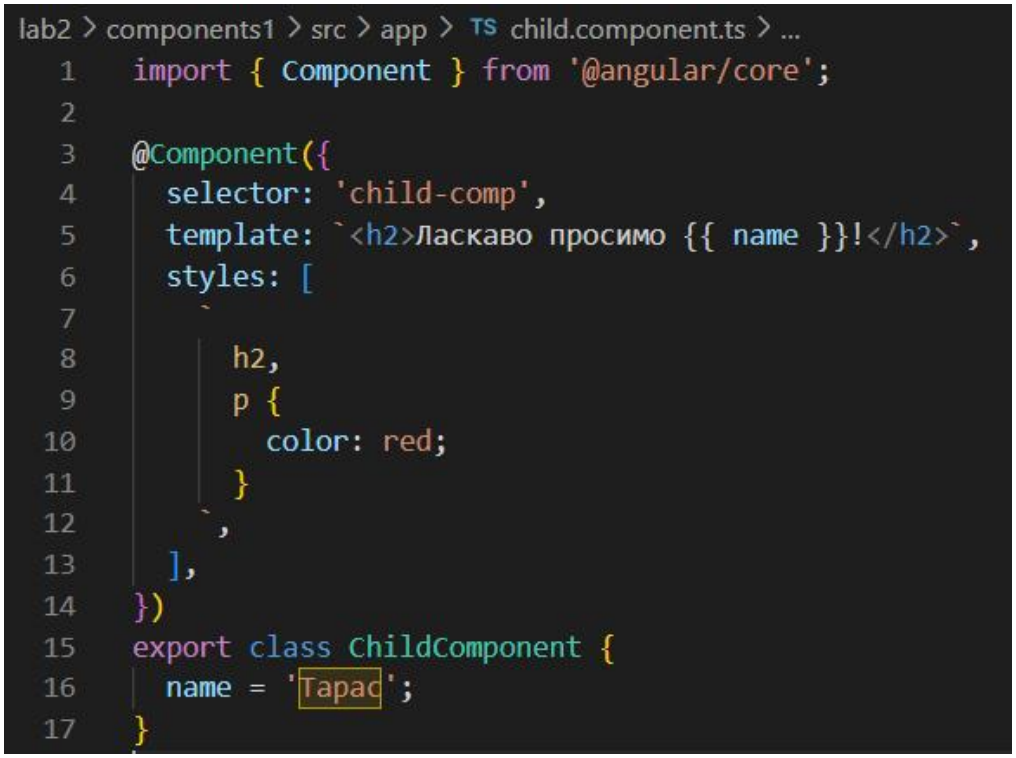
```
1  import { Component } from '@angular/core';
2  @Component({
3    selector: 'app-root',
4    templateUrl: './app.component.html',
5    styleUrls: ['./app.component.css'],
6  })
7  export class AppComponent {
8    name: '';
9  }
```



4) Робота з компонентами

Крім основних компонентів у додатку, ми також можемо визначати якісь допоміжні компоненти, які керують якоюсь ділянкою розмітки html. Більше того, у додатку на сторінці може бути ряд різних блоків з певним завданням. І для кожного такого блоку можна створити окремий компонент, щоб спростити керування блоками на сторінці





lab2 > components1 > src > app > TS app.module.ts > AppModule

```ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { ChildComponent } from './child.component';
@NgModule({
  imports: [BrowserModule, FormsModule],
  declarations: [AppComponent, ChildComponent],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

lab2 > components1 > src > app > TS app.component.ts > AppComponent

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

@Component({
  selector: 'app-root',
  template: ` <label>Введіть ім'я:</label>
    <input [(ngModel)]="name" placeholder="name" />
    <h1>Ласкаво просимо {{ name }}!</h1>
    <h2>Hello Angular</h2>
    <p>Angular 16 представляє модульну архітектуру додатку</p>
    <child-comp></child-comp>
    <p>Hello {{ name }}</p>`,
  styles: [
    `
      h2,
      p {
        color: #333;
      }
    `,
  ],
})
export class AppComponent {
  name = 'Петро';
}
```

Введіть ім'я: Петро

Ласкаво просимо Петро!

Hello Angular

Angular 16 представляє модульну архітектуру додатку

Ласкаво просимо Тарас!

Hello Петро

Підзаголовок

5) ng-content

Елемент ng-content дозволяє батьківським компонентам впроваджувати код HTML у дочірні компоненти

lab2 > components1 > src > app > TS app.component.ts > ...

```ts

@Component({
  selector: 'app-root',
  template: ` <label>Введіть ім'я:</label>
    <input [(ngModel)]="name" placeholder="name" />
    <h1>Ласкаво просимо {{ name }}!</h1>
    <h2>Hello Angular</h2>
    <p>Angular 16 представляє модульну архітектуру додатку</p>
    <child-comp
      ><h2>Ласкаво просимо {{ name }}!</h2></child-comp>
    <p>Hello {{ name }}</p>`,
  styles: [
    `
      h2,
      p {
        color: #333;
      }
    `,
  ],
})
export class AppComponent {
  name = 'Abuba';
}
```

Введіть ім'я: Abuba

Ласкаво просимо Abuba!

Hello Angular

Angular 16 представляє модульну архітектуру додатку

Ласкаво просимо Abuba!

Привіт AbubaSon

Hello Abuba

Підзаголовок

6) Взаємодія між компонентами. Передача даних у дочірній компонент

Оскільки властивість userName у дочірньому компоненті визначена як вхідна з декоратором Input, то в головному компоненті ми можемо її використовувати як атрибут і фактично застосувати прив'язку властивостей.

<child-comp [userName]="name2" [userAge]="age"></child-comp>

lab2 > components1 > src > app > TS app.component.ts > ...

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

@Component({
  selector: 'app-root',
  template: `
    <label>Введіть ім'я:</label>
    <input [(ngModel)]="name" placeholder="name" />
    <h1>Ласкаво просимо {{ name }}!</h1>
    <h2>Hello Angular</h2>
    <p>Angular 16 представляє модульну архітектуру додатку</p>
    <child-comp
      ><h2>Ласкаво просимо {{ name }}!</h2></child-comp
    >
    <p>Hello {{ name }}</p>
    <child-comp [userName]="name2" [userAge]="age"></child-comp>
    <input type="text" [(ngModel)]="name2" />
  `,
  styles: [
    `
      h3 {
        color: navy;
      }
      h2,
      p {
        color: navy;
      }
    `,
  ],
})
export class AppComponent {
  name = 'Abuba';
  name2: string = 'Abuba2';
  age: number = 99;
}
```

Введіть ім'я: Abuba

Ласкаво просимо Abuba!

Hello Angular

Angular 16 представляє модульну архітектуру додатку

Ласкаво просимо Abuba!

Привіт AbubaSon

Ім’я користувача:

Вік користувача: 0

Hello Abuba

Привіт AbubaSon

Ім’я користувача: Abuba2

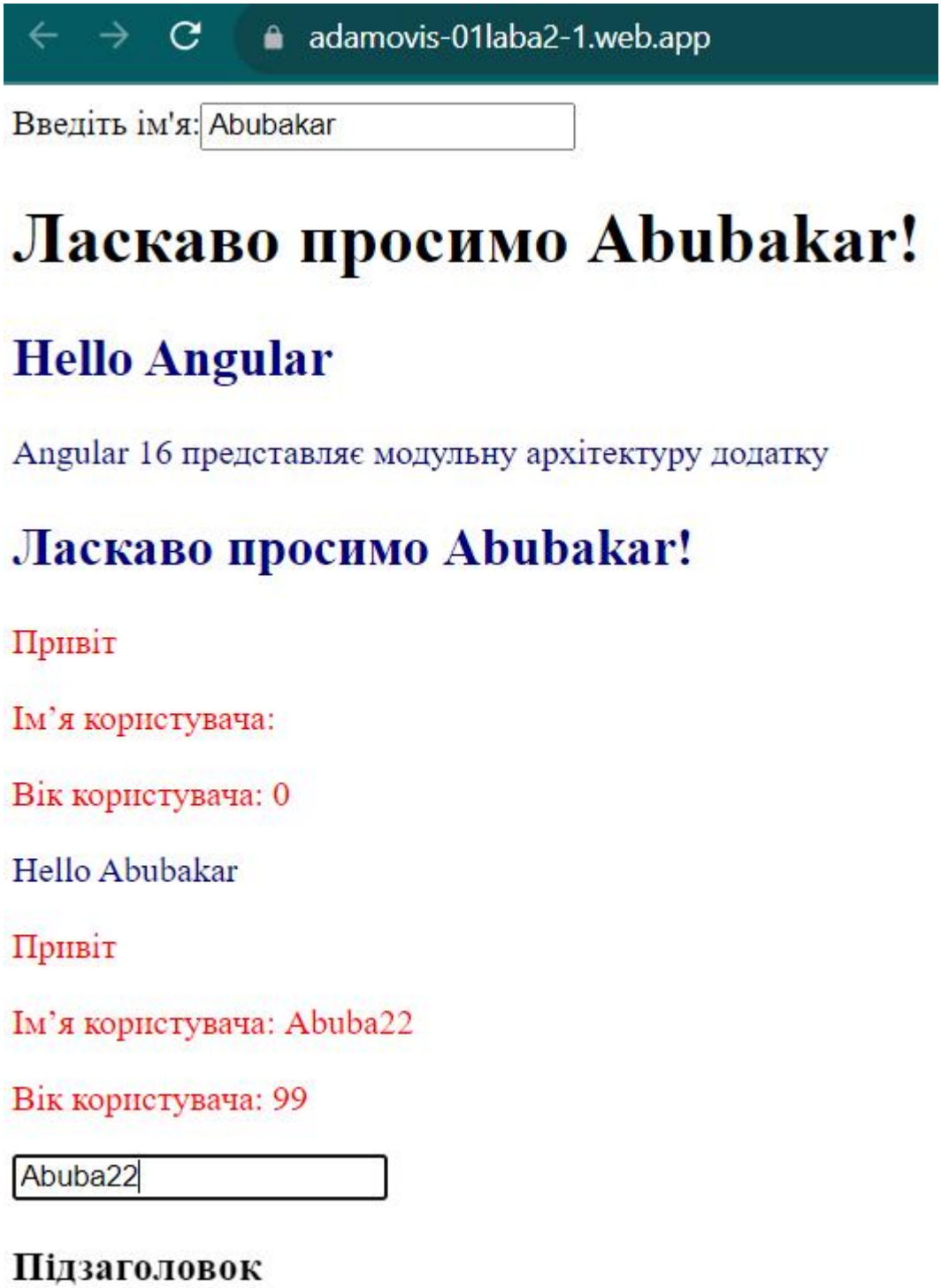
Вік користувача: 99

Abuba2

Підзаголовок

Розгорнутий на Firebase застосунок Components1

https://adamovis-01laba2-1.web.app/



Components2

7) Прив'язка до сетера

Крім прив'язки до властивості, ми можемо встановити прив'язку до сетера дочірнього компонента. Це може бути необхідно, коли у дочірньому компоненті треба здійснювати перевірку або навіть модифікацію значення, що отримується від головного компонента.

lab2 > components2 > src > app > TS app.component.ts > ...

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

@Component({
  selector: 'app-root',
  template: ` <child-comp [userName]="name" [userAge]="age"></child-comp>
    <input type="number" [(ngModel)]="age" />`,
})
export class AppComponent {
  name: string = 'abuba';
  age: number = 24;
}
```

lab2 > components2 > src > app > TS child.component.ts > ...

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

@Component({
  selector: 'child-comp',
  template: `<p>Ім’я користувача: {{ userName }}</p>
    <p>Вік користувача: {{ userAge }}</p>`,
})
export class ChildComponent {
  @Input() userName: string = '';
  _userAge: number = 0;

  @Input()
  set userAge(age: number) {
    if (age < 0) this._userAge = 0;
    else if (age > 100) this._userAge = 100;
    else this._userAge = age;
  }
  get userAge() {
    return this._userAge;
  }
}
```

Ім’я користувача: abuba

Вік користувача: 0

-3

8) Прив'язка до подій дочірнього компонента

Фактично властивість **changed** буде представляти собою подію, яка викликається в методі **change()** при кліку на кнопку і передається головному компоненту та обробляється там у методі **OnChanged()**.

lab2 > components2 > src > app > TS app.component.ts > ...

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

@Component({
  selector: 'app-root',
  template: `
    <h2>Кількість кліків: {{ clicks }}</h2>
    <child-comp (changed)="onChanged($event)"></child-comp>
  `,
})
export class AppComponent {
  clicks: number = 0;
  onChanged(increased: any) {
    increased == true ? this.clicks++ : this.clicks--;
  }
}
```

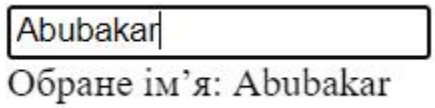
lab2 > components2 > src > app > TS child.component.ts > ...

```ts
import { Component, EventEmitter, Input, Output } from '@angular/core';

@Component({
  selector: 'child-comp',
  template: ` <button (click)="change(true)">+</button>
    <button (click)="change(false)">-</button>`,
})
export class ChildComponent {
  @Input() userName: string = '';
  _userAge: number = 0;
  @Input()
  set userAge(age: number) {
    if (age < 0) this._userAge = 0;
    else if (age > 100) this._userAge = 100;
    else this._userAge = age;
  }
  get userAge() {
    return this._userAge;
  }
  @Output() changed = new EventEmitter<boolean>();
  change(increased: any) {
    this.changed.emit(increased);
  }
}
```

Components3

9) Двостороння прив'язка

Ми можемо використовувати двосторонню прив'язку між властивостями головного і дочірнього компонента.



lab2 > components3 > src > app > TS app.component.ts > ...

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

@Component({
  selector: 'app-root',
  template: `<child-comp [(userName)]="name"></child-comp>
    <div>Обране ім'я: {{ name }}</div>`,
})
export class AppComponent {
  name: string = 'Abuba';
}
```

Тут визначено вхідну властивість userName, до якого прив'язане текстове поле input. Для зв'язку використовується атрибут [ngModel], який пов'язує значення атрибута value у текстового полі з властивістю userName.

lab2 > components3 > src > app > TS child.component.ts > ...

```ts
import { Component, Input, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'child-comp',
  template: ` <input
    [ngModel]="userName"
    (ngModelChange)="onNameChange($event)" />`,
})
export class ChildComponent {
  @Input() userName: string = '';
  @Output() userNameChange = new EventEmitter<string>();
  onNameChange(model: string) {
    this.userName = model;
    this.userNameChange.emit(model);
  }
}
```

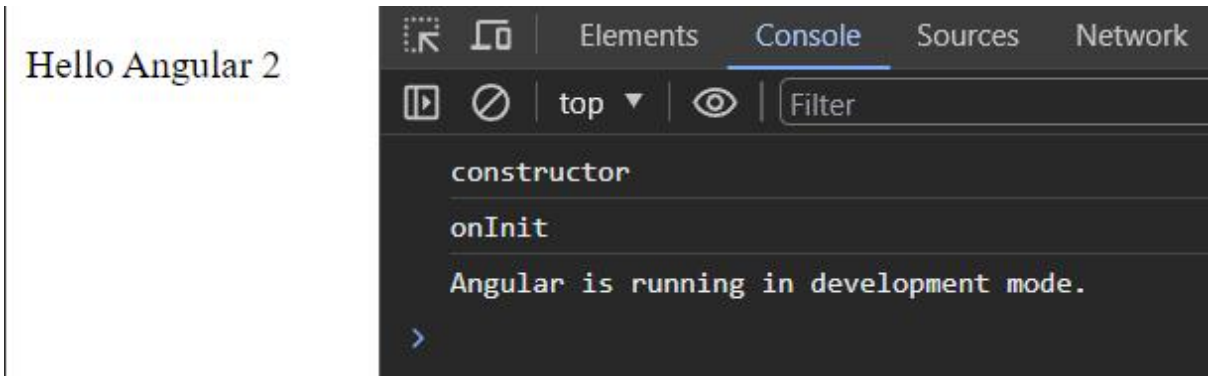
Тут встановлюється двостороння прив'язка властивостей userName дочірнього компонента та властивості name головного компонента.

Components4

10) Життєвий цикл компоненту

lab2 > components4 > src > app > TS app.component.ts > ...

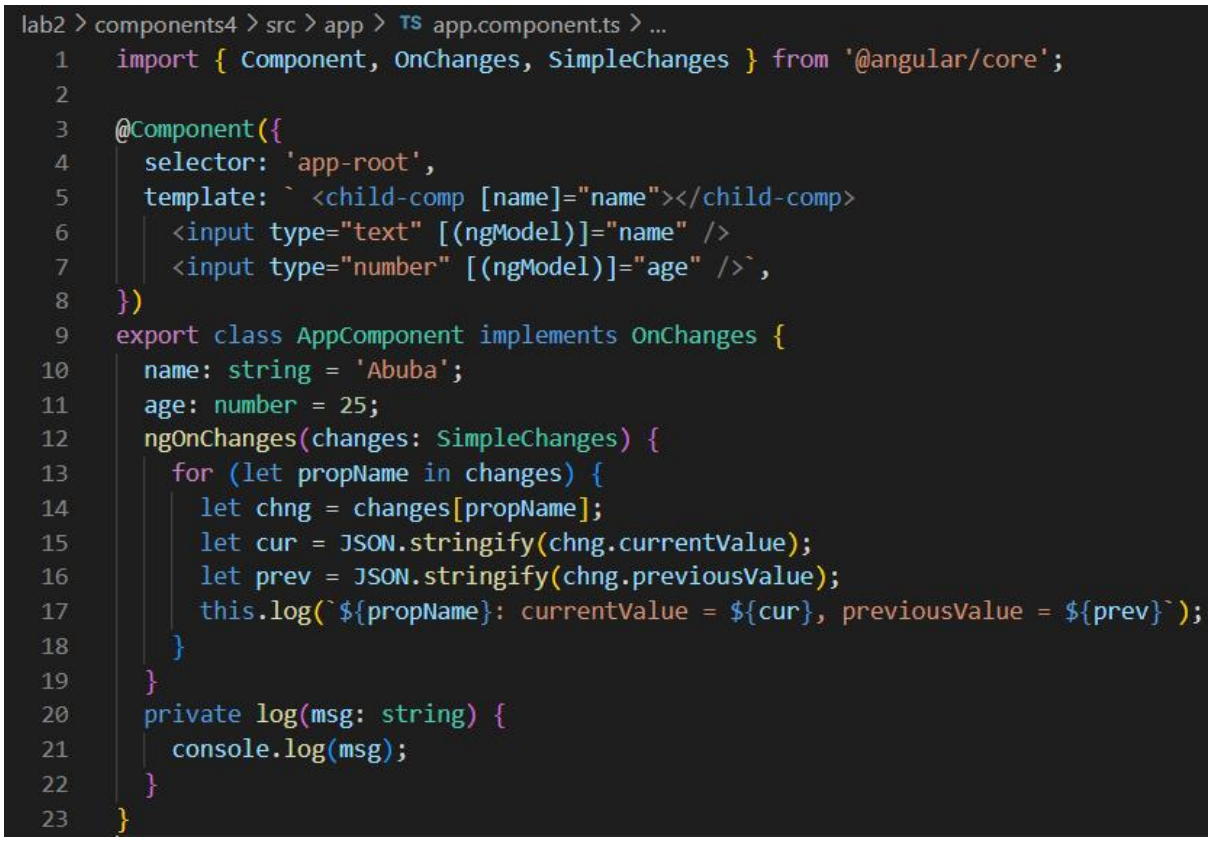
```ts
import { Component, OnInit, OnDestroy } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `<p>Hello Angular 2</p>`,
})
export class AppComponent implements OnInit, OnDestroy {
  name: string = 'Tom';

  constructor() {
    this.log(`constructor`);
  }
  ngOnInit() {
    this.log(`onInit`);
  }
  ngOnDestroy() {
    this.log(`onDestroy`);
  }
  private log(msg: string) {
    console.log(msg);
  }
}
```



Метод ngOnInit() застосовується для комплексної ініціалізації компонента. Тут можна виконувати завантаження даних із сервера або інших джерел даних.

Метод ngOnDestroy() викликається перед видаленням компонента. І в цьому методі можна звільняти ті ресурси, які не видаляються автоматично збирачем сміття. Тут також можна видаляти підписку на якісь події елементів DOM, зупиняти таймери тощо.

Метод ngOnChanges() викликається перед методом ngOnInit() і при зміні властивостей в прив'язці. За допомогою параметра SimpleChanges у методі можна отримати поточне та попереднє значення зміненої властивості.

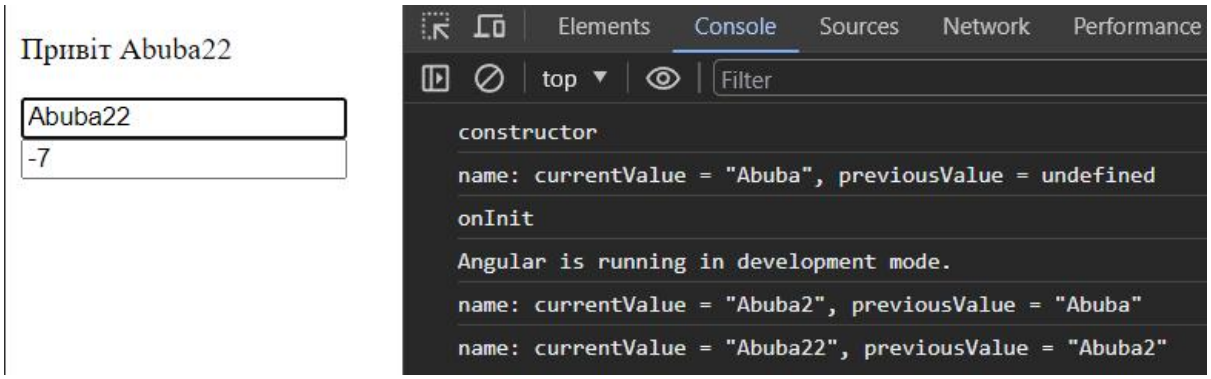


Значення властивості name передається в дочірній компонент ChildComponent з головного - AppComponent. Причому в головному компоненті також реалізований метод ngOnChanges().

І якщо ми запустимо додаток, то зможемо помітити, що при кожній зміні властивості name у головному компоненті викликається метод ngOnChanges.

lab2 > components4 > src > app > TS child.component.ts > ...

```ts
import {
  Component,
  Input,
  OnInit,
  OnChanges,
  SimpleChanges,
} from '@angular/core';

@Component({
  selector: 'child-comp',
  template: `<p>Привіт {{ name }}</p>`,
})
export class ChildComponent implements OnInit, OnChanges {
  @Input() name: string = '';
  constructor() {
    this.log(`constructor`);
  }
  ngOnInit() {
    this.log(`onInit`);
  }

  ngOnChanges(changes: SimpleChanges) {
    for (let propName in changes) {
      let chng = changes[propName];
      let cur = JSON.stringify(chng.currentValue);
      let prev = JSON.stringify(chng.previousValue);
      this.log(`${propName}: currentValue = ${cur}, previousValue = ${prev}`);
    }
  }
  private log(msg: string) {
    console.log(msg);
  }
}
```



Метод викликається лише при зміні вхідних властивостей з декоратором @Input. Тому зміна властивості age у AppComponent тут не буде відстежуватися.

Components5

11) Реалізація всіх методів

lab2 > components5 > src > app > TS app.component.ts > ...

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

@Component({
  selector: 'app-root',
  template: `<child-comp [name]="name"></child-comp>
    <input type="text" [(ngModel)]="name" />`,
})
export class AppComponent {
  name: string = 'Abuba';
}
```

lab2 > components5 > src > app > TS child.component.ts > ...

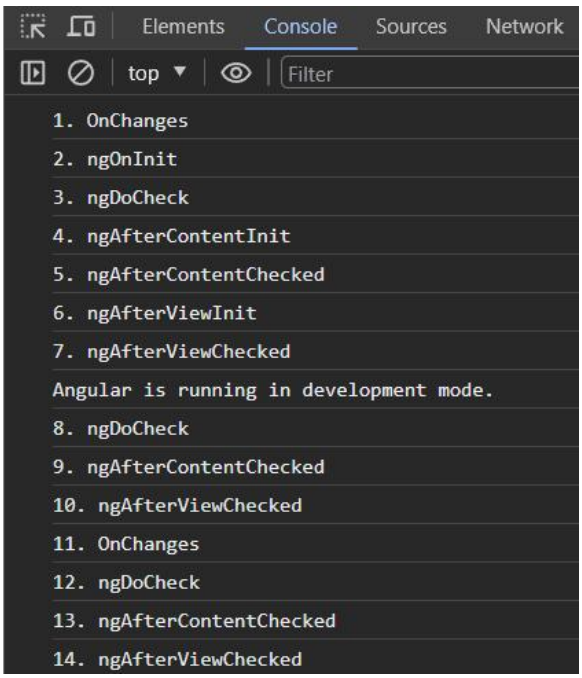
```ts
import {
  Component,
  Input,
  OnInit,
  DoCheck,
  OnChanges,
  AfterContentInit,
  AfterContentChecked,
  AfterViewChecked,
  AfterViewInit,
} from '@angular/core';

@Component({
  selector: 'child-comp',
  template: `<p>Привіт {{ name }}</p>`,
})
export class ChildComponent
  implements
    OnInit,
    DoCheck,
    OnChanges,
    AfterContentInit,
    AfterContentChecked,
    AfterViewChecked,
    AfterViewInit
```

```
{
  @Input() name: string = '';
  count: number = 1;

  ngOnInit() {
    this.log(`ngOnInit`);
  }
  ngOnChanges() {
    this.log(`OnChanges`);
  }
  ngDoCheck() {
    this.log(`ngDoCheck`);
  }
  ngAfterViewInit() {
    this.log(`ngAfterViewInit`);
  }
  ngAfterViewChecked() {
    this.log(`ngAfterViewChecked`);
  }
  ngAfterContentInit() {
    this.log(`ngAfterContentInit`);
  }
  ngAfterContentChecked() {
    this.log(`ngAfterContentChecked`);
  }
  private log(msg: string) {
    console.log(this.count + '. ' + msg);
    this.count++;
  }
}
```

При зверненні до програми ми отримаємо наступний ланцюжок викликів

Привіт Abubaa

Abubaa



• ngOnChanges: викликається до методу ngOnInit() при початковій установці властивостей, які пов'язані механізмом прив'язки, а також при будь-якій їхній переустановці або зміні їх значень. Даний метод як параметр приймає об'єкт класу SimpleChanges, який містить попередні та поточні значення властивості.
• ngOnInit: викликається один раз після встановлення властивостей компонента, що беруть участь у прив'язці. Виконує ініціалізацію компонента.
• ngDoCheck: викликається при кожній перевірці змін властивостей компонента відразу після методів ngOnChanges та ngOnInit.
• ngAfterContentInit: викликається один раз після методу ngDoCheck() після вставки вмісту у представлення компонента.
• ngAfterContentChecked: викликається фреймворком Angular при перевірці змін вмісту, який додається до представлення компонента. Викликається після методу ngAfterContentInit() та після кожного наступного виклику методу ngDoCheck().
• ngAfterViewInit: викликається фреймворком Angular після ініціалізації представлення компонента, а також представленнь дочірніх компонентів. Викликається лише один раз відразу після першого виклику методу ngAfterContentChecked().
• ngAfterViewChecked: викликається фреймворком Angular після перевірки на зміни у представленні компонента, а також перевірки представленнь дочірніх компонентів. Викликається після першого виклику методу ngAfterViewInit() та після кожного наступного виклику ngAfterContentChecked().
• ngOnDestroy: викликається перед тим, як фреймворк Angular видалить компонент.

Розгорнутий на Firebase застосунок Components5

https://adamovis-01laba2-5.web.app/