НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО

Факультет інформатики та обчислювальної техніки Кафедра інформатики та програмної інженерії

Звіт по лабораторній роботі №2

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

роботи з дисципліни: «Реактивне програмування»

Студент: Мєшков Андрій Ігорович

Група: ІП-15

Дата захисту роботи: 11 «жовтня» 2024

Викладач: доц. Полупан Юлія Вікторівна

Захищено з оцінкою: _____

Київ, 2024

ЗМІСТ

Стилі та шаблони компонента

Було створено новий проєкт на основі програм з минулої лабораторної. Стилізація компонента може виконуватися як за допомогою установки стилів у самому компоненті, так і за допомогою підключення зовнішніх CSS-файлів. Для встановлення стилів у директиві @Component визначено властивість styles. Параметр styles містить набір стилів, які використовуватимуться компонентом.

Було додано нові стилі в «Components1». На рис. 2.1 показано сторінку сайту.

Рис.2.1 Вправа 2

Лістинг коду:

App.component.ts

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

@Component({
    selector: 'my-app',
    templateUrl: `
      <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>
`,
    styleUrls: [`
      h2,h3{color:navy;}
      p{font-size:13px; font-family:Verdana; color:navy;}

      `]
})
export class AppComponent {
    name= 'Andrii';
}
```

Селектор: host. Призначення та використання.

Селектор :host посилається на елемент, у якому хоститься компонент. Тобто в цьому випадку це елемент <my-app></my-app>. І селектор :host дає змогу застосувати стилі до цього елементу.

Було додано нові стилі в «Components1». На рис. 2.2 показано сторінку сайту.

Рис.2.2 Вправа 2(+ :host)

Лістинг коду:

App.component.ts

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

@Component({
    selector: 'my-app',
    templateUrl: `
      <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>
`,
    styleUrls: [`
      h2,h3{color:navy;}
      p{font-size:13px; font-family:Verdana; color:navy;}
      :host {
          font-family: Verdana;
          color: #555;
```

```
      }
      `]
})
export class AppComponent {
    name= 'Andrii';
}
```

Підключення зовнішніх файлів стилів та шаблонів.

Якщо стилів багато, код компонента може бути занадто роздутий, і в цьому випадку їх виносять в окремий файл css. Було додано 2 файли: app.components.html і app.components.css.

Лістинг коду:

App.component.ts

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

@Component({
    selector: 'my-app',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {
    name= 'Andrii';
}
```

App.component.html

```
<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>
```

App.component.css

```
h2,h3{color:navy;}
p{font-size:13px; font-family:Verdana; color:navy;}
:host {
    font-family: Verdana;
    color: #555;
}
```

ng-content: призначення та використання.

Елемент ng-content дозволяє батьківським компонентам впроваджувати код HTML у дочірні компоненти. Створимо дочірній компонент ChildComponent. Замість елемента <ng-content></ ng-content> ззовні можна буде передати будь-який вміст.

Лістинг коду:

App.component.ts

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

@Component({
    selector: 'my-app',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {
    name= 'Andrii';
    name2:string="Mieshkov";
    age:number = 20;
}
```

App.component.html

```
<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>
```

App.component.css

```
h2,h3{color:navy;}
p{font-size:13px; font-family:Verdana; color:navy;}
:host {
    font-family: Verdana;
    color: #555;
}
```

child.component.css

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

@Component({
```

```
    selector: 'child-comp',
    template: `<ng-content></ng-content>
        <p>Привіт {{name}}</p>
    `,
    styles: [`h2, p {color:red;}`]
})
export class ChildComponent {
    name= "Not Andrii";
}
```

App.module.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 { }
```

На рис. 2.3 показано сторінку сайту.

Рис.2.3 Вправи 4-5

Взаємодія між компонентами. Передача даних у дочірній компонент.

У минулій темі було розглянуто, як викликати компонент із головного компонента. Однак за замовчуванням ці компоненти не взаємодіють, вони незалежні. Кожен компонент визначає свої вирази прив'язки. Однак що, якщо ми хочемо прив'язати властивості дочірнього компонента до властивостей головного компонента? Для цього визначимо наступний дочірній компонент.

Ключовим моментом є визначення вхідних властивостей з допомогою декоратора @Input(). І природно, щоб використовувати декоратор, його треба імпортувати.

Ключовою особливістю таких вхідних властивостей є те, що вони можуть встановлюватися ззовні, тобто ззовні отримувати значення, наприклад, з головного компонента.

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.

Це ж стосується і властивості userAge. В результаті характеристики userAge і userName будуть прив'язані до значень з основного компонента.

На рис. 2.4 показано сторінку сайту.

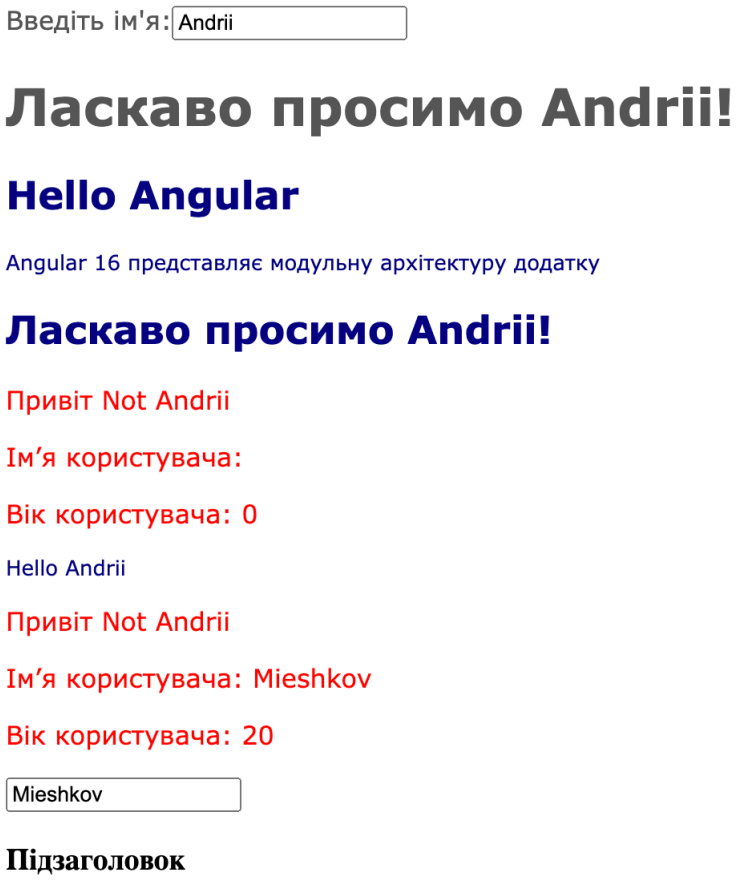

Рис.2.4 Вправа 6

Лістинг коду:

App.component.ts

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

@Component({
    selector: 'my-app',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {
    name= 'Andrii';
    name2:string="Mieshkov";
    age:number = 20;
}
```

App.component.html

```
<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" />
```

App.component.css

```
h2,h3{color:navy;}
p{font-size:13px; font-family:Verdana; color:navy;}
:host {
    font-family: Verdana;
    color: #555;
}
```

child.component.css

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

@Component({
    selector: 'child-comp',
    template: `<ng-content></ng-content>
        <p>Привіт {{name}}</p>
        <p>Ім’я користувача: {{userName}}</p>
        <p>Вік користувача: {{userAge}}</p>
    `,
    styles: [`h2, p {color:red;}`]
})
export class ChildComponent {
    name= "Not Andrii";
    @Input() userName: string = "";
    @Input() userAge: number = 0;
}
```

App.module.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 { }
```

Прив'язка до сетера.

Створено Component2. Крім прив'язки до властивості, ми можемо встановити прив'язку до сеттера дочірнього компонента. Це може бути необхідно, коли у дочірньому компоненті треба здійснювати перевірку або навіть модифікацію значення, що отримується від головного компонента.

У головному компоненті встановлюється вік користувача. А в дочірньому компоненті отримуватимемо переданий вік через сеттер.

У головному компоненті ми можемо запровадити будь-яке значення у полі введення, зокрема і негативні числа. У дочірньому компоненті через сеттер перевіряємо введене значення та за необхідності коригуємо його.

Лістинг коду:

App.component.ts

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

@Component({
    selector: 'my-app',
    template: `
        <child-comp [userName]="name" [userAge]="age" </child-comp>
        <input type="number" [(ngModel)]="age" />

    `

})
export class AppComponent {
    name:string="Andrii";
    age:number = 20;
}
}
```

child.component.css

```
import { Component, EventEmitter, Input, Output} 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; }

}
```

App.module.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 { }
```

На рис. 2.5 показано сторінку сайту.

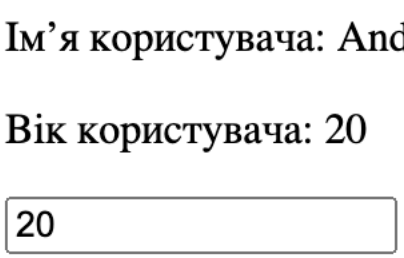

Рис.2.5 Вправа 7

Прив'язка до подій дочірнього компонента.

Ще однією формою взаємодії є прив'язка до подій дочірнього компонента. Так, визначимо дочірній компонент. У цьому компоненті у кнопки використовується подія click, яка викликає метод change, передаючи йому значення true або false. Тут же в дочірньому компоненті ми можемо обробити події. Але якщо ми повинні передавати його батьківському компоненту, то для цього нам треба використовувати властивість типу EventEmitter, яким тут є властивість onChanged. Оскільки ми передаватимемо значення типу true або false, то дана властивість типизується типом boolean. При цьому властивість onChanged повинна бути вихідною, тому вона позначається за допомогою декоратора @Output. Фактично властивість onChanged буде представляти собою подію, яка викликається в методі change() при кліку на кнопку і передається головному компоненту.

За допомогою виразу (onChanged)="onChanged($event)" прив'язуємо метод onChanged до події onChanged(), що викликається в дочірньому компоненті. Параметр $event інкапсулює всі дані, що передаються із дочірнього компонента. У результаті при натисканні на кнопки в дочірньому компоненті подія натискання транслюватиметься головному копоненту, який в залежності від переданого значення збільшуватиме або зменшуватиме лічильник.

Лістинг коду:

App.component.ts

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

@Component({
    selector: 'my-app',
    template: `
        <h2>Кількість кліків: {{clicks}}</h2>
        <input type="number" [(ngModel)]="age" />
```

```
        <child-comp [userName]="name" [userAge]="age"
(onChanged)="onChanged($event)"></child-comp>

    `

})
export class AppComponent {
    name:string="Andrii";
    age:number = 20;
    clicks:number = 0;
    onChanged(increased:any){
    increased==true?this.clicks++:this.clicks--;
 }
}
```

child.component.css

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

@Component({
    selector: 'child-comp',
    template: `
        <p>Ім'я користувача: {{userName}}</p>
        <p>Вік користувача: {{userAge}}</p>
        <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() onChanged = new EventEmitter<boolean>();
    change(increased:any) {
        this.onChanged.emit(increased);
    }
}
```

App.module.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 { }
```

На рис. 2.6 показано сторінку сайту.

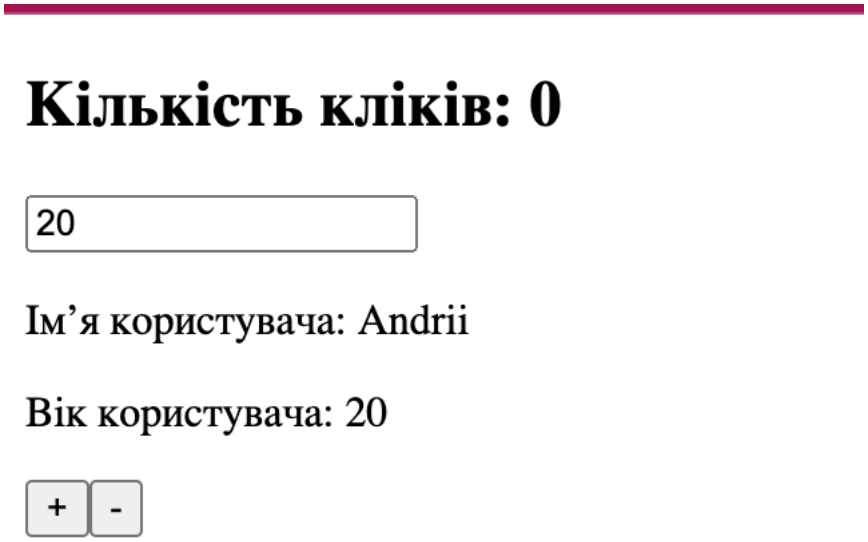

Рис.2.6 Вправа 8

Двостороння прив'язка.

Створимо додаток Component 3. У прикладі вище ми визначали прив'язку до події дочірнього компонента: у разі виникнення події у дочірньому компоненті ми обробляли її у головному компоненті за допомогою методу. Але ми можемо використовувати двосторонню прив'язку між властивостями головного і дочірнього компонента.

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.

Для відстеження зміни моделі цього поля за допомогою атрибуту (ngModelChange) прив'язуємо метод, який спрацьовує при зміні значення. Тобто ngModelChange – це фактично подія зміни введеного значення, тому тут діє прив'язка до події.

Так як у нас тут одностороння прив'язка, то в методі-обробнику отримуємо введене значення та змінюємо властивість userName і генеруємо подію userNameChange, яку визначено як вихідний параметр.

Тобто тут ззовні ми отримуємо значення властивості userName і встановлюємо його для текстового поля. При введенні користувача в це поле генеруємо зовні подію userNameChange.

Тут встановлюється двостороння прив'язка властивостей userName дочірнього компонента та властивості name головного компонента. При цьому не потрібно вже вручну обробляти подію userNameChange, все робитиметься автоматично.

Лістинг коду:

App.component.ts

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

@Component({
    selector: 'my-app',
    template: `
        <child-comp [(userName)]="name"></child-comp>
        <div>Обране ім'я: {{name}}</div>
    `

})
export class AppComponent {
    name:string="Andrii";
}
```

child.component.css

```
import { Component, EventEmitter, Input, Output} 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);
    }
}
```

App.module.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 { }
```

На рис. 2.7 показано сторінку сайту.

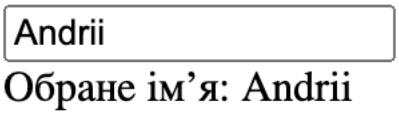

Рис.2.7 Вправа 9

Життєвий цикл компоненту.

Створимо додаток Component 4. Після створення компонента фреймворк Angular викликає у цього компоненту ряд методів, які представляють різні етапи життєвого циклу:

- 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.

Кожен такий метод визначено в окремому інтерфейсі, який називається ім'ям методу без префікса "ng". Наприклад, метод ngOnInit визначено в інтерфейсі OnInit. Тому, якщо ми хочемо відстежувати якісь етапи життєвого циклу компонента, то клас компонента має застосовувати відповідні інтерфейси.

Метод ngOnInit() застосовується для комплексної ініціалізації компонента. Тут можна виконувати завантаження даних із сервера або інших джерел даних. ngOnInit() не аналогічний конструктору. Конструктор також може виконувати деяку ініціалізацію об'єкта, водночас щось складне у конструкторі робити не рекомендується. Конструктор має бути по можливості простим та виконувати саму базову ініціалізацію. Щось складніше, наприклад, завантаження даних із сервера, яке може зайняти тривалий час, краще робити в методі ngOnInit.

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. ngOnChanges 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.

Тобто 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 ngOnChange.

У той самий час слід зазначити, що цей метод викликається лише при зміні вхідних властивостей з декоратором @Input. Тому зміна властивості age у AppComponent тут не буде відстежуватися.

Лістинг коду:

App.component.ts

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

@Component({
    selector: 'my-app',
    template: `
        <child-comp [name]="name"></child-comp>
        <input type="text" [(ngModel)]="name" />
        <input type="number" [(ngModel)]="age" />
    `

})
export class AppComponent implements OnChanges {
    name:string="Andrii";
    age:number = 20;

    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);
    }
}
```

child.component.css

```
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);
    }
}
```

App.module.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 { }
```

На рис. 2.8 показано сторінку сайту.

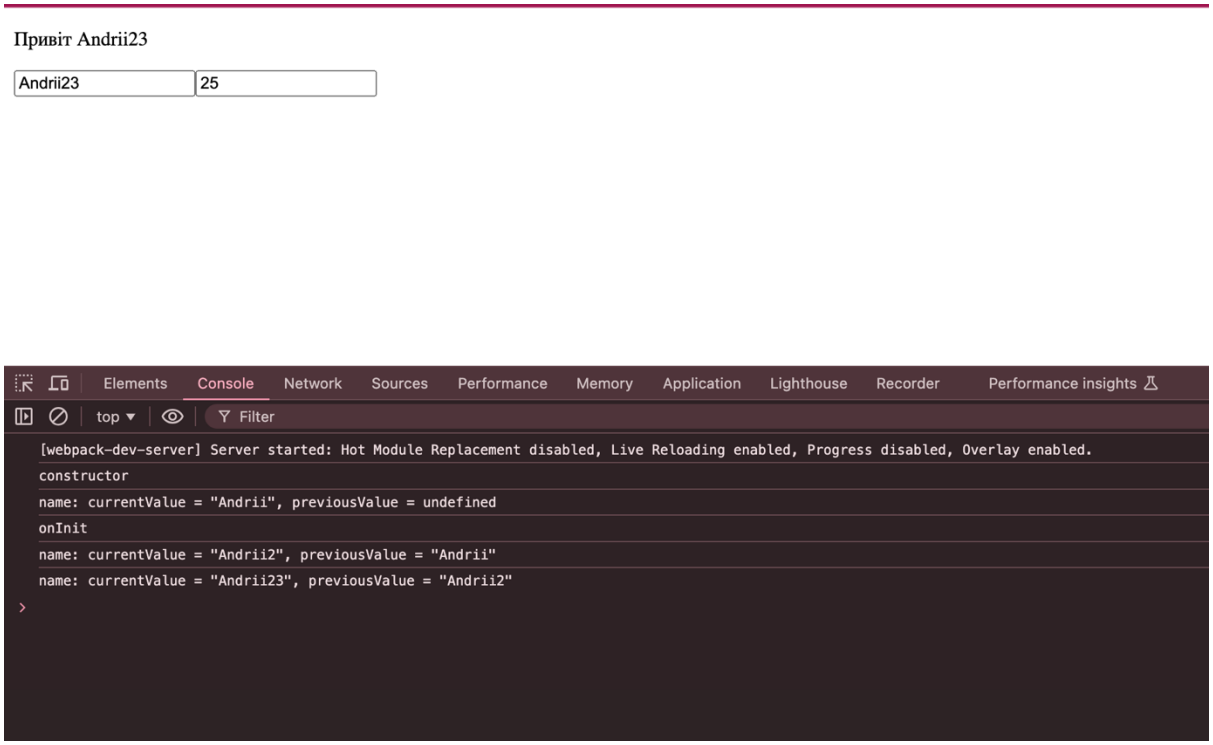

Рис.2.8 Вправа 10

Створимо додаток Component 5 з реалізацією всіх методів.

Лістинг коду:

App.component.ts

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

@Component({
    selector: 'my-app',
    template: `
        <child-comp [name]="name"></child-comp>
        <input type="text" [(ngModel)]="name" >
    `

})
export class AppComponent {
    name:string="Andrii";
}
```

child.component.css

```
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,Afte
rViewInit {

    @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++;
    }
}
```

App.module.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 { }
```

На рис. 2.9 показано сторінку сайту.

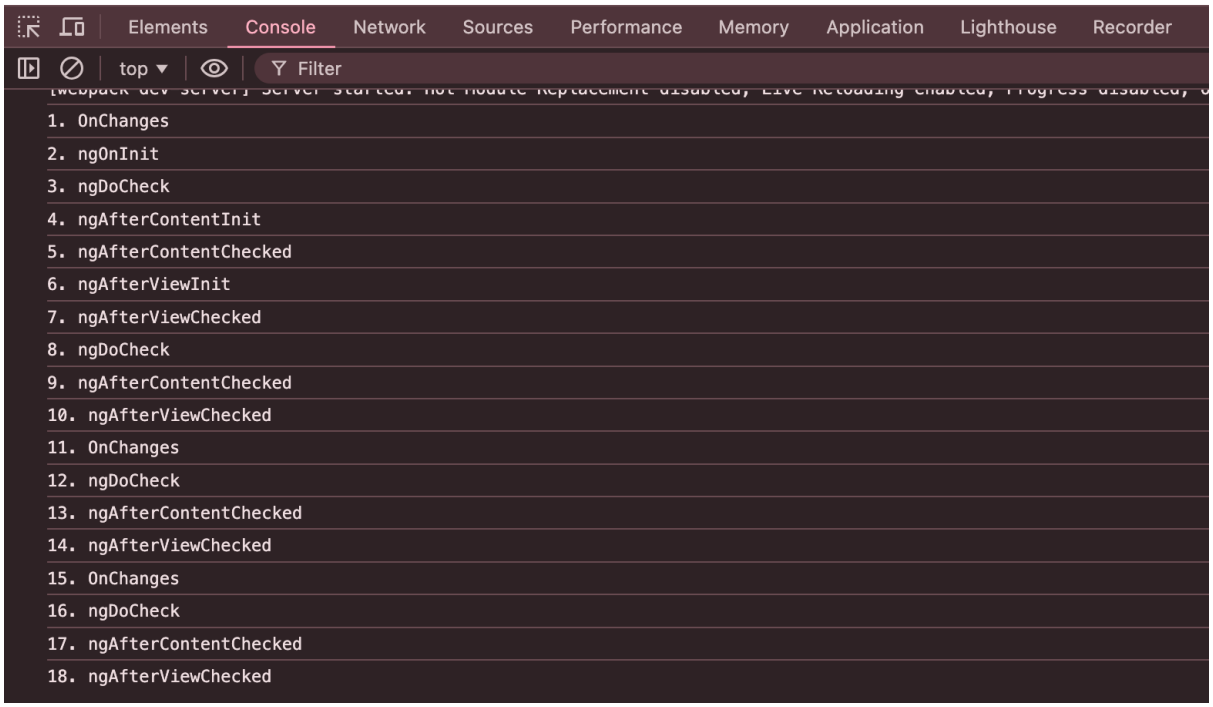

Рис.2.9 Вправа 11

ВИСНОВКИ

У ході виконання лабораторної роботи я засвоїв базові навички роботи з компонентами Angular, взаємодії між ними та прив'язки до подій дочірніх компонентів. Основні результати та висновки:

Я створив п’ять Angular-додатків, які демонструють основні принципи створення та взаємодії компонентів. Вивчив структуру Angular-компонентів, які складаються з шаблону (HTML), стилів (CSS) та логіки (TypeScript).

Я навчився передавати дані між батьківськими та дочірніми компонентами за допомогою властивостей і подій. Розібрав механізм прив'язки до подій дочірнього компоненту та передачі даних в обох напрямках, що є важливим для побудови динамічних та інтерактивних веб-застосунків.

Ознайомився з життєвим циклом компонентів в Angular, включно з основними хукамі: `ngOnInit()`, `ngOnChanges()`, `ngOnDestroy()` та іншими. Це дало можливість зрозуміти, як правильно керувати станом компоненту на різних етапах його існування, що допомагає у розробці стабільних та ефективних додатків.

Виконані вправи дали можливість практично закріпити навички створення багатокомпонентних додатків, їхньої взаємодії та керування подіями. Я створив та розгорнув додатки на платформі Firebase, що допомогло ознайомитися з процесом розгортання Angular-додатків на реальних серверах.

Додатки Components1 та Components5 були успішно розгорнуті на платформі Firebase у відповідних проектах з іменами https://mieshkovip15laba2-1.web.app/ та https://mieshkovip15laba2-5.web.app/ . Це дало мені можливість отримати досвід роботи з інструментами для розгортання веб-додатків та зрозуміти важливість деплою як етапу розробки.

Отже, в ході цієї лабораторної роботи я отримав глибше розуміння основ Angular, зокрема, роботи з компонентами та їхньої взаємодії, а також процесу розгортання застосунків на хмарних платформах, таких як Firebase. Це створило базу для подальшого вивчення та застосування Angular у розробці складніших веб-додатків.

СПИСОК ДЖЕРЕЛ

1. Документація Angular. URL: https://v17.angular.io/docs
2. Документація Nodejs. URL: https://nodejs.org/docs/latest/api/
3. Документація Firebase. URL: https://firebase.google.com/docs?hl=en