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.

з дисципліни: «Реактивне програмування»

Студент: Черпак Андрій Вадимович

Група: ІП-01

Дата захисту роботи: 06.10.2023

Викладач: доц. Полупан Юлія Вікторівна

Захищено з оцінкою: _____

Київ, 2023

Зміст

Мета: Навчитися працювати з компонентами в Angular.
Завдання: Створити п’ять Angular-додатків під назвою:

1. **Components1 (вправи 1-6). Виконати відповідні вправи;**
2. **Components2 (вправи 7-8). Виконати відповідні вправи;**
3. **Components3 (вправа 9). Виконати відповідні вправи;**
4. **Components4 (вправа 10). Виконати відповідні вправи;**
5. **Components5 (вправа 11). Виконати відповідні вправи;**
6. **Зробити звіт по роботі.**
7. **Angular-додатки Components1 та Components5 розгорнути на платформі**
8. **Firebase у проектах з ім’ям «ПрізвищеГрупаLaba2-1» та «ПрізвищеГрупаLaba2-5», наприклад «KovalenkoIP01Laba2-1» та «KovalenkoIP01Laba2-5»**

Вправа 1: Створення додатку з назвою «Components1»

Із минулого комп’ютерного практикуму запозичимо проект HelloApp:

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

@Component({
    selector: 'my-app',
    template: `<label>Введіть назву:</label>
                <input [(ngModel)]="name"
placeholder="name">
                <h1>Ласкаво просимо {{name}}!</h1>`
})
export class AppComponent {
    name = ' ';
}
```

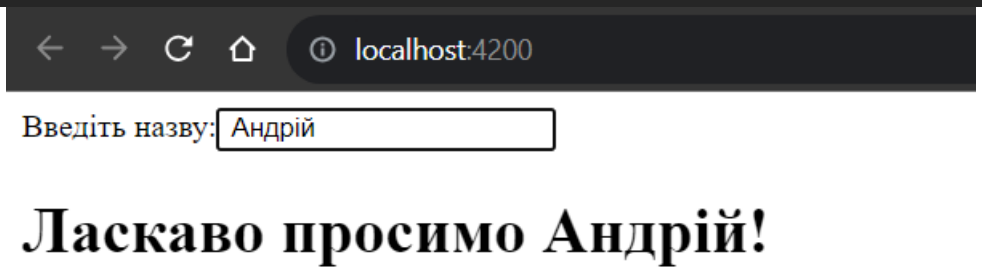


Рисунок 1.1. Застосунок, запозичений з попереднього практикуму.

Вправа 2: Стилі та шаблони компонента.

Для стилізації компонента додамо йому атрибут styles:

```
import { Component } from '@angular/core';
@Component({
    selector: 'my-app',
    template: `<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 { }
```

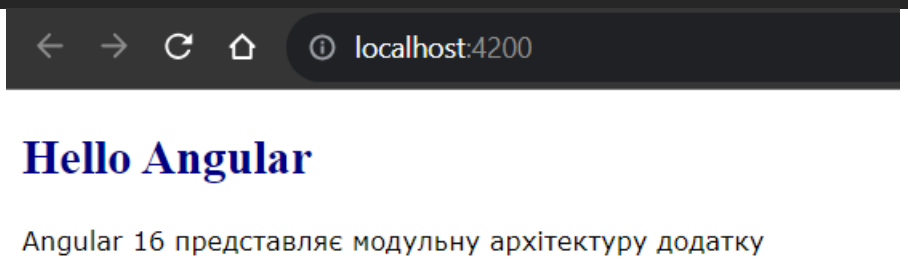


Рисунок 1.2. Приклад додавання стилів до компоненти.

Вправа 3: Підключення зовнішніх файлів.

Якщо стилі або шаблон займають доволі багато місця, є сенс їх винести у окремі файли. Таким чином код компоненту стане чистішим і зрозумілішим.

```
import { Component } from '@angular/core';
@Component({
    selector: 'my-app',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {
```

```
    name: " ";
}
```

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

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

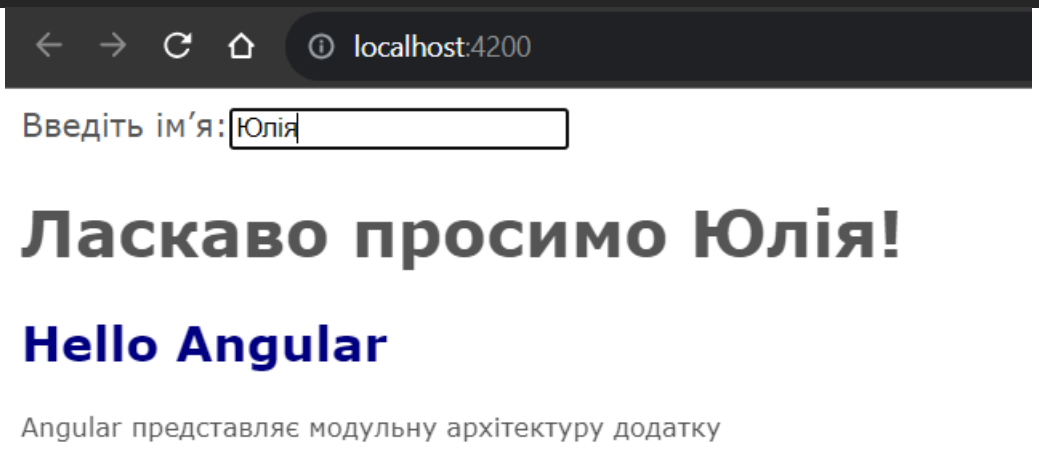


Рисунок 1.3.Результат винесення стилів та шаблону в окремі файли.

Вправа 4: Робота з компонентами.

Окрім основної компоненти, можна також визначити допоміжні. Вони міститимуться всередині основної компоненти, але матимуть власні стилі.

Змінимо код компоненту AppComponent:

```
<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>
```

```
h2, p {
  color:#333;
}
```

```
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 { }
```

Створимо дочірній компонент ChildComponent:

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

@Component({
    selector: 'child-comp',
    template: `<h2>Ласкаво просимо {{name}}!</h2>`,
```

```
    styles: [`h2, p {color:red;}`]
})
export class ChildComponent {
    name= "Тарас";
}
```

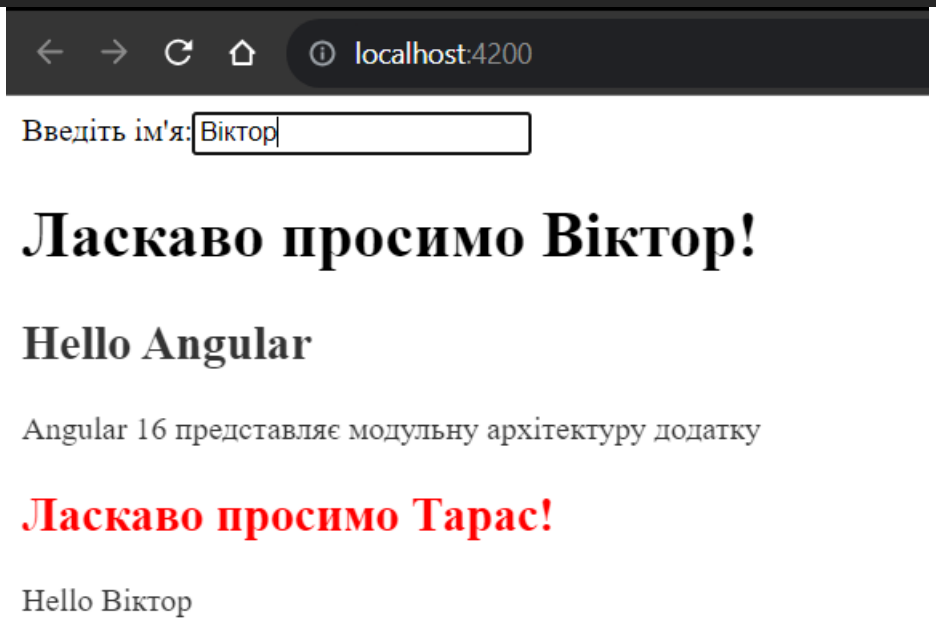


Рисунок 1.4. Використання допоміжних компонент.

Як бачимо, дочірній компонент має власні стилі та простір імен: зовнішня і внутрішня змінні name мають різні значення.

Вправа 5: ng-content.

Також є можливість передавати код HTML всередину дочірнього компоненту. Для цього існує елемент ng-content, куди підставляються передані значення.

Змінимо компонент ChildComponent:

```
import { 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= "Тарас";
}
```

А також внесемо зміни у файли AppComponent:

```
<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>
```

```
h3{color:navy;}
h2, p {color:navy;}
```

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

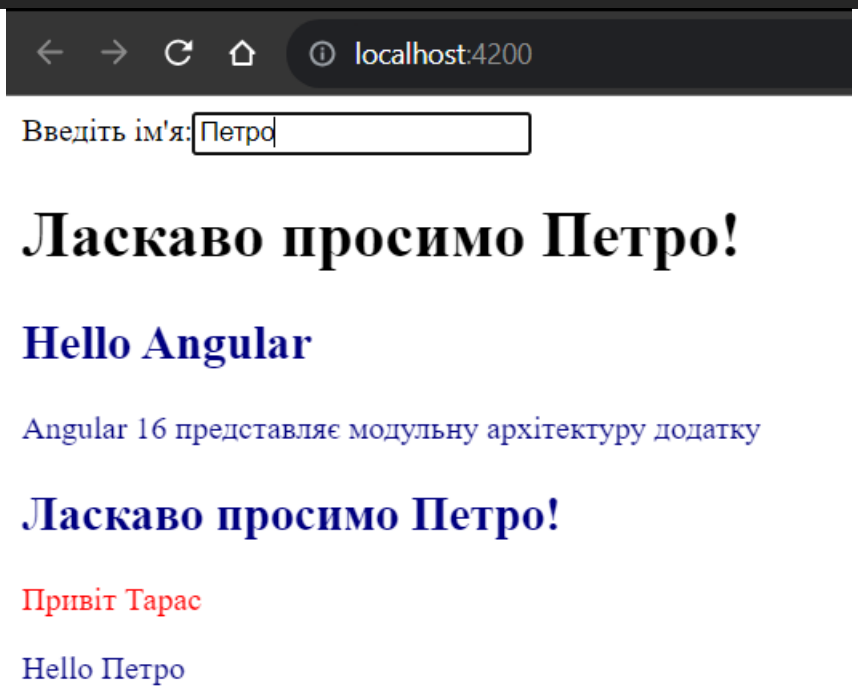


Рисунок 1.5. Приклад передачі у дочірній компонент.

Як бачимо, всередину дочірнього компоненту було передано зовнішнє значення змінної name.

Вправа 6: Взаємодія між компонентами. Передача даних у дочірній компонент через Input.

. У компоненті ChildComponent імпортуємо декоратор Input та змінимо код на наступний:

```
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: [`h3,p{color:red;}`]
})
export class ChildComponent{
    name = "Тарас";
    @Input() userName: string = "";
    @Input() userAge: number = 0;
}
```

Після чого у головній компоненті визначимо такі моделі, як name, name2 та age:

```
import { Component } from '@angular/core';
@Component({
    selector: 'my-app',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {
    name:string = 'Петро';
    name2:string="Том";
    age:number = 24;
}
```

І у шаблоні передамо ці параметри таким чином:

```
<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" />
```

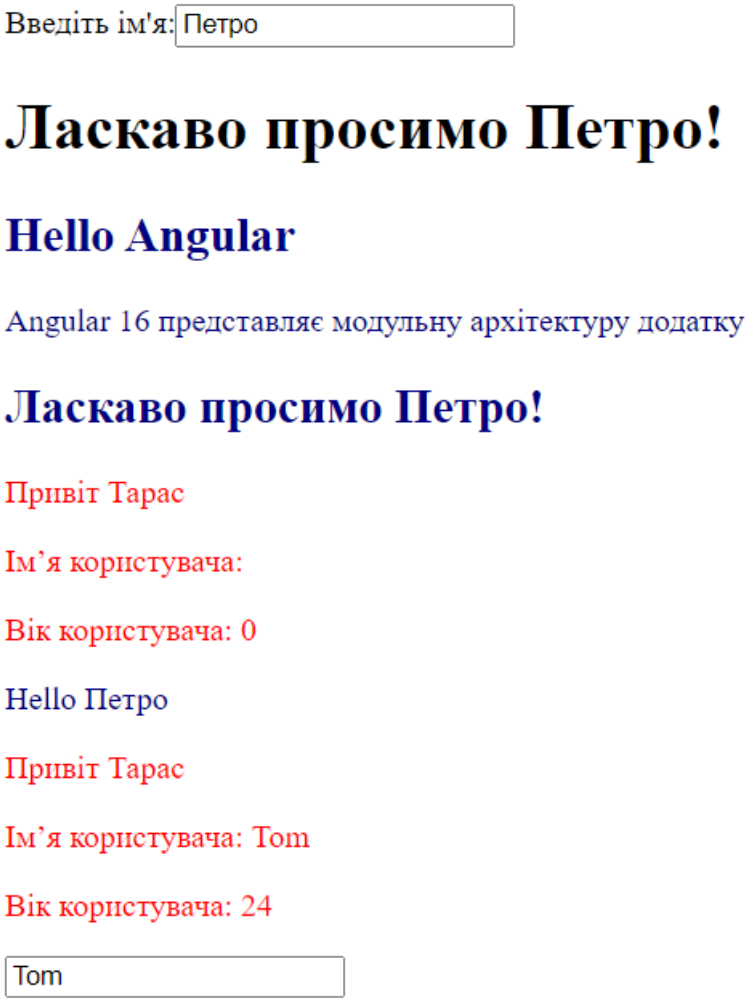


Рисунок 1.6. Результат передачі параметрів через Input.

Вправа 7: Прив'язка до сетера.

Іноді нам необхідно якось опрацьовувати передані значення. Наприклад, валідувати їх, або якимось чином змінювати. У такому разі можна скористатися прив’язкою до сетера.

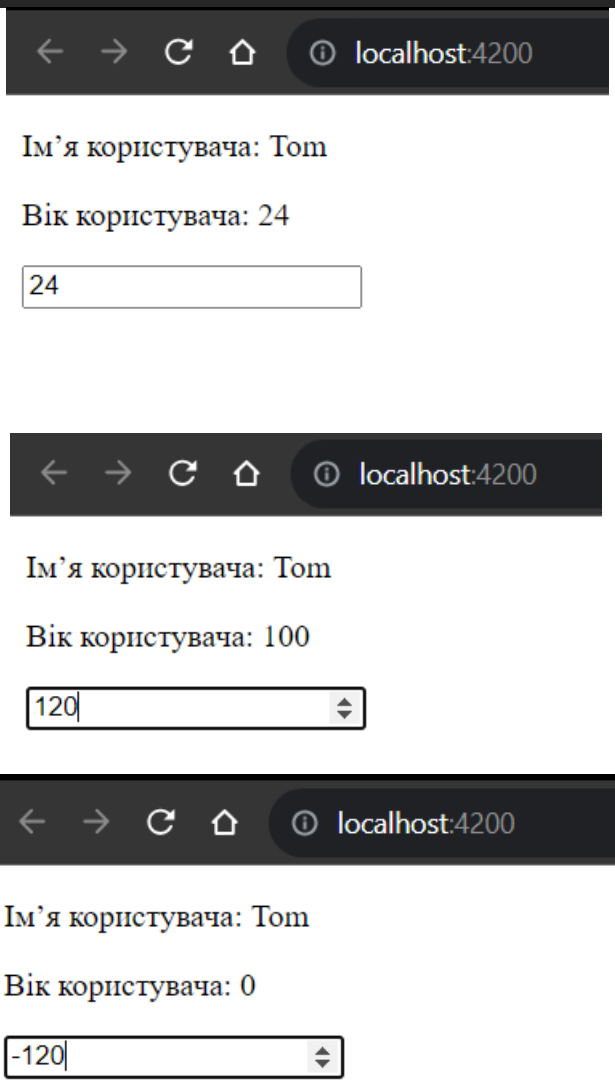
Наведемо приклад такої прив’язки у child.component:

```
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; }
}
```

Тоді app.component матиме такий вигляд:

```
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="Tom";
    age:number = 24;
}
```



Рисунки 1.7-1.9. Приклади прив'язки до сетера.

Вправа 8: Прив'язка до подій дочірнього компонента.

Іноді нам необхідно у основному компоненті реагувати на зміні у дочірньому. Цього можна досягти таким чином:

app.component:

```
import { Component } from '@angular/core';
@Component({
    selector: 'my-app',
    template: `
        <h2>Кількість кліків: {{clicks}}</h2>
        <child-comp
(onChanged)="onChanged($event)"></child-comp>
    `
})
export class AppComponent {
    name:string="Tom";
    age:number = 24;
    clicks:number = 0;
    onChanged(increased:any){
        increased==true?this.clicks++:this.clicks--;
    }
}
```

child.component:

```
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);
    }
}
```

У такому разі при натисканні кнопок дочірнього компоненту відбуватиметься зміна значення моделі основного компоненту:

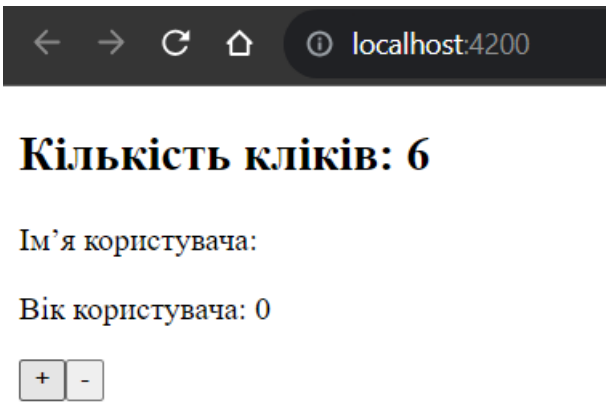


Рисунок 1.10. Приклади прив’язки до подій дочірнього компонента

Вправа 9: Двостороння прив'язка.

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

child.component:

```
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);
    }
}
```

app.component:

```
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 = "Tom";
}
```

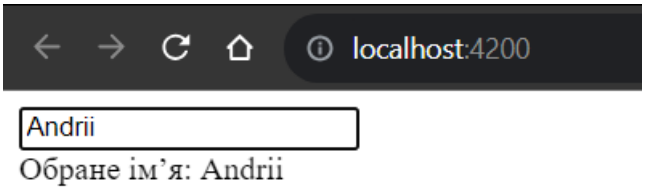


Рисунок 1.11. Приклад двосторонньої прив'язки.

Вправа 10: Життєвий цикл компоненту.

Існує також можливість відслідковувати різні етапи життєвого циклу компоненту за допомогою методів ngOnInit, ngOnChanges і так далі.

Нехай дочірній компонент виглядає так:

```
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);
    }
}
```

Тоді основний компонент:

```
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="Tom";
    age:number = 25;
    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);
```

```
    }
}
```

Після запуску додатку та зміни значень полів у консолі зможемо відслідкувати порядок запуску тих чи інших методів. Також помітимо, що при зміні значення моделі, яка не використовується дочірнім компонентом, його метод OnChange не викликатиметься.

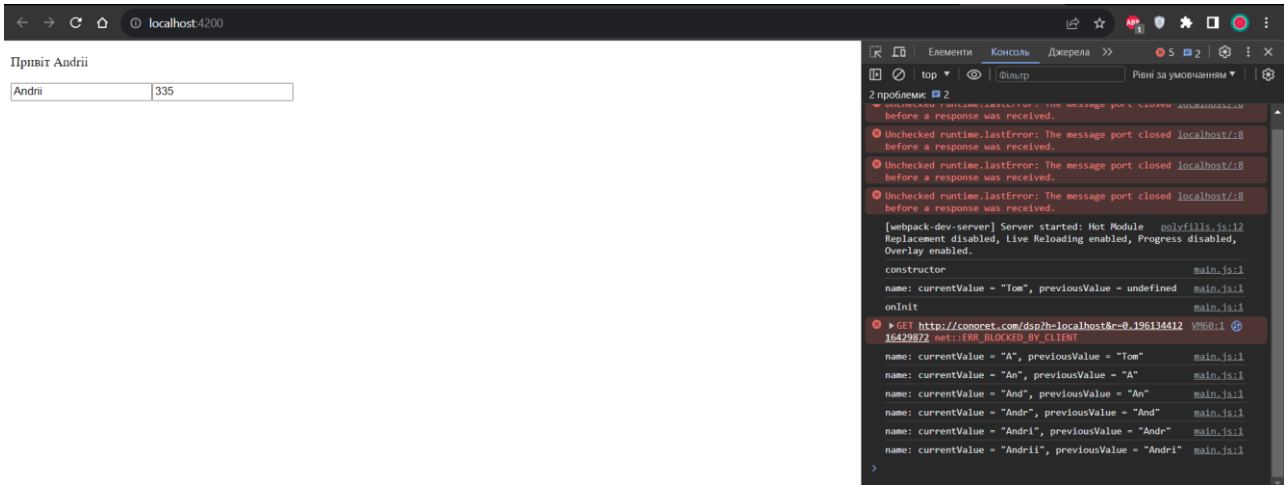


Рисунок 1.12. Життєвий цикл компоненту.

Вправа 11: Реалізація всіх методів.

Тепер у новому додатку спробуємо реалізувати усі методи.

child.component:

```
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++;
    }
}
```

app.component:

```
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="Tom";
}
```

Як бачимо з коду, тепер при будь-якій події відбуватиметься логування:

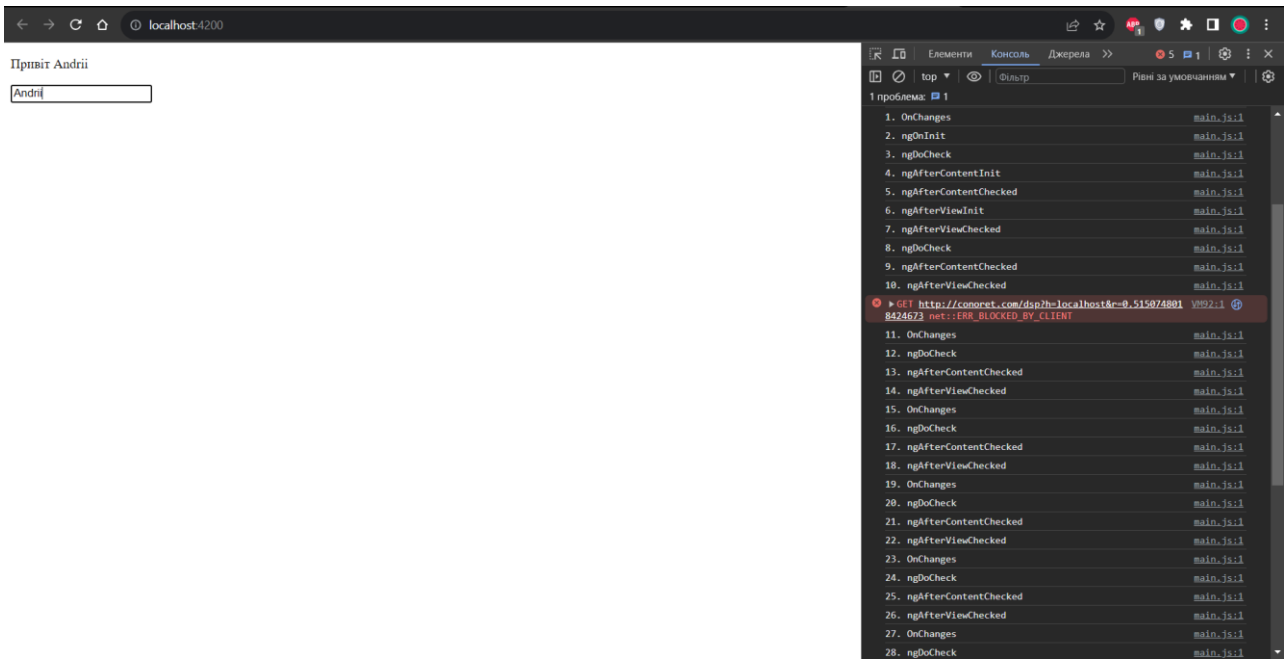


Рисунок 1.13. Результати логування усіх методів.

Посилання на додатки:

1. cherpakip01laba2-1.web.app
2. cherpakip01laba2-5.web.app

Висновок:

Під час виконання комп’ютерного практикуму ми навчилися змінювати стилі компоненту засобами Angular, підключати зовнішні файли, створювати допоміжні компоненти і як передавати у них дані через ng-content тег чи декоратор @Input, так і отримувати дані з допомогою двосторонньої прив’язки через декоратор @Output та EventEmitter. Також ми ознайомилися з життєвим циклом компоненту та навчилися реагувати на його події. Нами було створено 5 додатків, два з яких було розгорнуто на платформі firebase.

Список використаних джерел:

1. Component Lifecycle: https://angular.io/guide/lifecycle-hooks