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

Факультет інформатики та обчислювальної техніки Кафедра інформатики та програмної інженерії

Звіт по лабораторній роботі №4

«Сервіси. Створення локальних та глобальних сервісів.»

роботи з дисципліни: «Реактивне програмування»

Студент: Мєшков Андрій Ігорович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Група: ІП-15\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата захисту роботи: 14 «грудня» 2024\_\_\_\_\_\_\_\_\_\_\_

Викладач: доц. Полупан Юлія Вікторівна\_\_\_\_\_\_\_\_\_\_

Захищено з оцінкою: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ, 2024

ЗМІСТ

[Сервіси: призначення та приклади використання 3](#_Toc185782568)

[Впровадження сервісу в інший сервіс 9](#_Toc185782569)

[Опціональні сервіси 11](#_Toc185782570)

[Один сервіс для всіх компонентів 13](#_Toc185782571)

[Ієрархія сервісів 16](#_Toc185782572)

[Детальний огляд та призначення всіх структурних блоків Angular-додатку Service2 20](#_Toc185782573)

[Детальний огляд та призначення всіх структурних блоків Angular-додатку Service3 25](#_Toc185782574)

[ВИСНОВКИ 29](#_Toc185782575)

[СПИСОК ДЖЕРЕЛ 31](#_Toc185782576)

# Сервіси: призначення та приклади використання

**Сервіси**¶

Сервіси в Angular представляють досить широкий спектр класів, які виконують деякі специфічні завдання, наприклад логування, роботу з даними і т.д.

На відміну від компонентів і директив, сервіси не працюють з уявленнями, тобто з розміткою html, не надають на неї прямого впливу. Вони виконують строго певне і досить вузьке завдання.

Стандартні завдання сервісів:

1) Надання даних додатку. Сервіс може сам зберігати дані в пам'яті або для отримання даних може звертатися до якогось джерела даних, наприклад, до сервера.

2) Сервіс може представляти канал взаємодії між окремими компонентами програми

3) Сервіс може інкапсулювати бізнес-логіку, різноманітні обчислювальні завдання, завдання з логування, які краще виносити з компонентів. Таким чином, код компонентів буде зосереджений безпосередньо на роботі з поданням. Крім того, ми також можемо вирішити проблему повторення коду, якщо нам потрібно виконати одну і ту ж задачу в різних компонентах і класах.

Створимо Angular додаток Service.

**Вправа 1**

1. **Створення класу для даних:**
   * У файлі phone.ts був створений клас Phone, який представляє модель даних з двома властивостями: name (назва моделі телефона) і price (ціна телефона).
   * Цей клас слугує основою для роботи з даними у проєкті.
2. **Розробка сервісу:**
   * У файлі data.service.ts був створений сервіс DataService.
   * У сервісі визначено:
     + Приватний масив data із початковими даними про телефони.
     + Метод getData() для отримання даних.
     + Метод addData(name: string, price: number) для додавання нових телефонів до масиву.
3. **Створення компонента:**
   * У компоненті AppComponent реалізовано:
     + Властивості для збереження введених даних (name і price).
     + Масив items для відображення даних.
     + Метод addItem(name: string, price: number) для передачі введених даних у сервіс.
     + Використання ngOnInit для ініціалізації даних із сервісу.
4. **Використання механізму dependency injection:**
   * Сервіс DataService було імпортовано та додано до колекції providers компонента.
   * У конструкторі компонента об'єкт сервісу передано як залежність.
5. **Шаблон компонента:**
   * Створено форму для введення даних із використанням директиви ngModel.
   * Виведено таблицю для відображення наявних телефонів із застосуванням директиви \*ngFor.
6. **Модуль FormsModule:**
   * Для роботи з формами та директивою ngModel у файлі app.module.ts підключено модуль FormsModule.
7. **Стилізація:**
   * У файлі стилів app.component.css налаштовано базові стилі для полів введення, таблиці та її елементів.

У вправі створено повноцінний приклад використання сервісу в Angular, який включає створення сервісу для управління даними, ін'єкцію залежностей у компонент, роботу з формами через директиву ngModel та базову стилізацію.

App.component.ts

*import* { Component, OnInit } *from* '@angular/core'

*import* { DataService } *from* './data.service'

*import* { Phone } *from* './phone'

@Component({

selector: 'my-app',

template: `

<div *class*="row">

<input

*class*="form-control cardinput"

*[(ngModel)]*="name"

*placeholder*="Модель"

/>

<input

*type*="number"

*class*="form-control cardinput"

*[(ngModel)]*="price"

*placeholder*="Ціна"

/>

<button

*class*="btn btn-default cardinput"

*(click)*="addItem(name, price)"

>

Додати

</button>

</div>

<table>

<thead>

<tr>

<th *class*="cardinput">Модель</th>

<th *class*="cardinput">Ціна</th>

</tr>

</thead>

<tbody>

<tr *\*ngFor*="let item of items">

<td *class*="cardinput">{{item.name}}</td>

<td *class*="cardinput">{{item.price}}</td>

</tr>

</tbody>

</table>

`,

styleUrls: ['app.component.css'],

providers: [DataService],

})

*export* class AppComponent implements OnInit {

name: string='';

price: number;

items: Phone[] = []

constructor(private dataService: DataService) {}

addItem(name: string, price: number) {

*this*.dataService.addData(name, price)

}

ngOnInit() {

*this*.items = *this*.dataService.getData()

}

}

App.component.css

*.row*{

width: 53%;

padding: 2em 27% 0 20%;

display: flex;

flex-direction: row;

justify-content: space-between;

}

*.form-control* {

width: 35%;

}

*.cardinput*{

margin: 0.4rem 1.0rem;

}

td {

padding: 5px;

margin: 0.4rem 1.8rem;

}

table {

width: 100%;

overflow:auto;

color: rgb(0, 0, 0);

border-spacing: 1px;

padding: 0 20%;

}

th {

padding: 5px;

margin: 0.4rem 1.8rem;

text-align:left;

}

Phone.ts

*export* class Phone {

constructor(public name: string, public price: number) {}

}

Data.service.ts

*import* { Phone } *from* './phone'

*export* class DataService {

private data: Phone[] = [

{ name: 'Apple iPhone 7', price: 36000 },

{ name: 'HP Elite x3', price: 38000 },

{ name: 'Alcatel Idol S4', price: 12000},

]

getData(): Phone[] {

*return* *this*.data

}

addData(name: string, price: number) {

*this*.data.push(new Phone(name, price))

}

}

App.module.ts

*import* { NgModule } *from* '@angular/core';

*import* { BrowserModule } *from* '@angular/platform-browser';

*import* { FormsModule } *from* '@angular/forms';

*import* { AppComponent } *from* './app.component';

@NgModule({

imports: [ BrowserModule, FormsModule ],

declarations: [ AppComponent],

bootstrap: [ AppComponent ]

})

*export* class AppModule { }

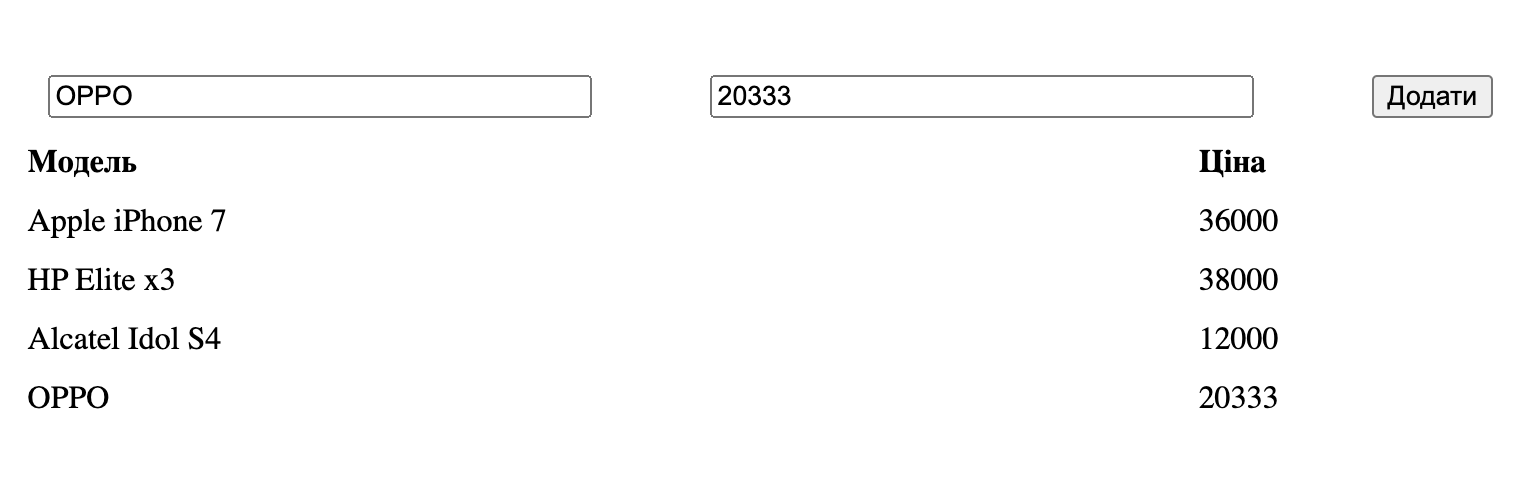


Рисунок 4.1. Результат

# Впровадження сервісу в інший сервіс

**Вправа 2**

***Нові файли:***

1. **log.service.ts:**
   * Створено новий сервіс LogService для логування.
   * Містить метод write(logMessage: string), який виводить повідомлення до консолі.

***Змінені файли:***

1. **data.service.ts:**
   * Додано імпорт нового сервісу:
   * import { LogService } from './log.service';
   * Додано декоратор @Injectable до класу DataService.
   * У конструкторі DataService додано залежність від LogService:
   * constructor(private logService: LogService) {}
   * Логування операцій додано до методів:
     + У методі getData() викликається this.logService.write('операція отримання даних').
     + У методі addData() викликається this.logService.write('операція додавання даних').
2. **app.component.ts:**
   * Додано імпорт нового сервісу:
   * import { LogService } from './log.service';
   * У секцію providers додано LogService:
   * providers: [DataService, LogService],

***Основні дії:***

1. Реалізовано механізм **використання одного сервісу в іншому**.
   * LogService використовується в DataService для логування операцій із даними.
2. Використання декоратора @Injectable для DataService, що забезпечує можливість ін'єкції залежностей.
3. Логіка роботи:
   * Усі операції додавання та отримання даних тепер супроводжуються логуванням у консоль.

***Результат:***

* При виконанні дій із даними в консоль виводяться повідомлення, що демонструють інтеграцію та роботу сервісу LogService всередині DataService.

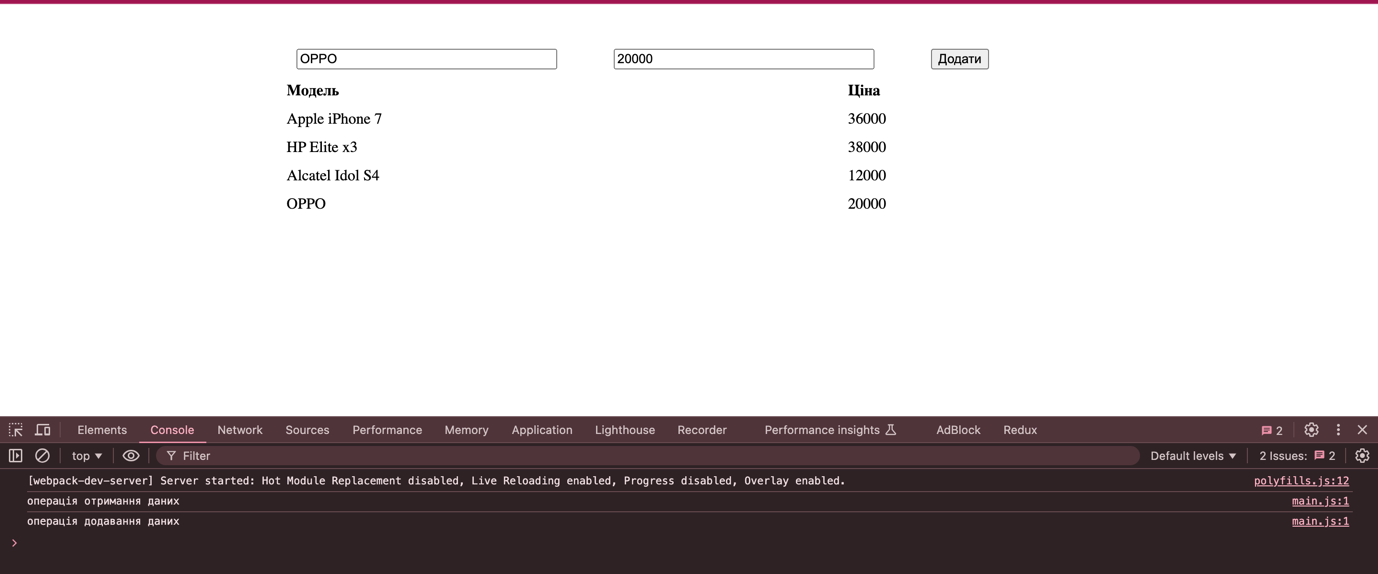


Рисунок 4.2. Результат

# Опціональні сервіси

**Вправа 3**

**Що таке опціональні сервіси:**

* Опціональні сервіси дозволяють визначити залежності, які можуть бути недоступними під час виконання програми.
* Якщо сервіс не наданий у провайдерах, програма не видасть помилку, а замість цього перевірить наявність залежності і виконає дію лише за її наявності.

**Як працює механізм:**

* Для цього застосовується декоратор @Optional із бібліотеки Angular.
* Коли залежність визначена як опціональна, Angular при її відсутності просто надає null замість об'єкта.

**Змінені файли:**

1. **data.service.ts:**
   * Імпортовано декоратор Optional:
   * import { Injectable, Optional } from '@angular/core';
   * Внесено зміни у конструктор для вказання LogService як опціонального:
   * constructor(@Optional() private logService: LogService) {}
   * Змінено методи для перевірки наявності LogService перед використанням:
     + У методі getData():
     + if (this.logService) this.logService.write('операція отримання даних');
     + У методі addData() додано аналогічну перевірку (для виклику write).
2. **app.component.ts:**
   * Видалено LogService із секції providers, залишено лише:
   * providers: [DataService],

**Результат вправи:**

1. **Без помилок при відсутності сервісу:**
   * Якщо LogService не зареєстровано в провайдерах, програма не видасть помилку. Замість цього logService матиме значення null.
2. **Гнучкість у використанні:**
   * Сервіс DataService тепер працює незалежно від того, чи доступний LogService. Логування операцій відбувається лише за умови, якщо LogService присутній.
3. **Практичність:**
   * Цей підхід корисний для сервісів, які можуть бути не обов’язковими (наприклад, логування, аналітика чи тимчасові інтеграції).

У вправі було використано декоратор @Optional, щоб зробити сервіс LogService опціональним. Це дозволяє уникнути помилок у випадках, коли сервіс відсутній у провайдерах. Тепер програма може гнучко працювати без прив’язки до обов’язкової наявності залежності.

# Один сервіс для всіх компонентів

**Вправа 4**

**Проблема:** Коли компоненти реєструють свої власні провайдери сервісів, для кожного компонента створюється окремий екземпляр сервісу. Це призводить до того, що дані між компонентами не синхронізуються, навіть якщо компоненти використовують однакові сервіси.

**Рішення:** Щоб забезпечити використання одного й того самого екземпляра сервісу між усіма компонентами, сервіси потрібно реєструвати в головному модулі програми (AppModule), а не в окремих компонентах.

#### **1. Додано новий компонент** DataComponent**:**

* Файл data.component.ts:
  + Реалізовано компонент для додавання та відображення даних із сервісу DataService.
  + У початковій версії сервіс DataService зареєстровано в секції providers самого компонента:
  + providers: [DataService, LogService]

#### **2. Оновлено головний компонент** AppComponent**:**

* Файл app.component.ts:
  + Включено два виклики компонента DataComponent:

template: `

<data-comp></data-comp>

<data-comp></data-comp>

`,

#### **3. Поведінка з локальними провайдерами:**

* Для кожного екземпляра DataComponent створюється окремий екземпляр DataService. Дані між компонентами не синхронізуються.

#### **4. Реєстрація сервісів на рівні модуля:**

* У файлі app.module.ts сервіси DataService і LogService зареєстровано в секції providers модуля:
* providers: [DataService, LogService]
* Це забезпечує створення єдиного екземпляра кожного сервісу для всієї програми.

#### **5. Видалено провайдери з** DataComponent**:**

* Файл data.component.ts:
  + Секцію providers видалено.
  + Тепер компонент отримує єдиний екземпляр DataService, наданий через AppModule.

### **Результат:**

1. **Синхронізація даних між компонентами:**
   * Тепер обидва екземпляри DataComponent використовують той самий об'єкт DataService.
   * Додавання даних в одному екземплярі компонента автоматично відображається в іншому.
2. **Зменшення ресурсів:**
   * Завдяки єдиному екземпляру сервісу зменшується кількість створюваних об'єктів, що покращує продуктивність.
3. **Глобальна доступність сервісів:**
   * Сервіси доступні для будь-якого компонента в програмі, без необхідності дублювання їхньої реєстрації.

### Код для порівняння:

#### Початкова реєстрація сервісів у компоненті:

@Component({

selector: 'data-comp',

providers: [DataService, LogService]

})

#### Реєстрація сервісів на рівні модуля:

@NgModule({

providers: [DataService, LogService]

})

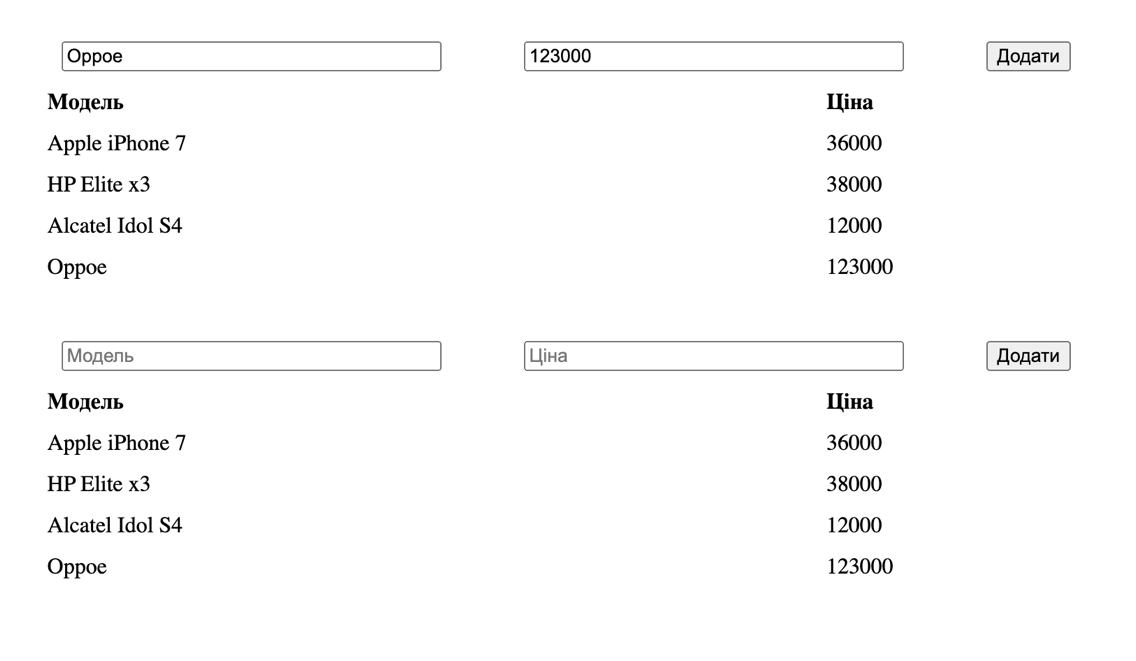


Рисунок 4.3. Результат

# Ієрархія сервісів

В Angular сервіс може бути впроваджений на різних рівнях. Рівень провайдера визначає область дії (scope) та життєвий цикл (lifecycle) сервісу. Залежно від того, де сервіс зареєстрований, його поведінка і доступність змінюються. Розрізняють три рівні провайдерів:

1. **Глобальний або кореневий рівень (Root Level)**
2. **Рівень модуля (Module Level)**
3. **Рівень компонента (Component Level)**

**1. Кореневий рівень (Root Level)**

Сервіс із кореневим рівнем доступний для всієї програми. Його екземпляр створюється один раз і використовується всіма компонентами, модулями та іншими сервісами. Це схоже на патерн Singleton.

Реєстрація сервісу на кореневому рівні

1. **Через головний модуль:** У головному модулі (зазвичай AppModule) сервіс додається до колекції providers:

@NgModule({

providers: [DataService]

})

*export* class AppModule {}

1. **Через декоратор @Injectable:** У декораторі @Injectable параметр providedIn: 'root' автоматично додає сервіс до кореневого рівня:

@Injectable({ providedIn: 'root' })

*export* class DataService {}

**Перевага:** У цьому випадку немає потреби додавати сервіс до providers у модулі.

**Область дії кореневого сервісу**

* Сервіс діє на весь додаток.
* Один екземпляр сервісу використовується в усіх компонентах і модулях.

**2. Рівень модуля (Module Level)**

Сервіс із рівнем модуля доступний лише в межах одного модуля і його компонентів, директив або інших сервісів.

**Реєстрація сервісу на рівні модуля**

1. **Додавання до колекції providers:**

@NgModule({

providers: [DataService]

})

*export* class DataModule {}

1. **Через providedIn у декораторі @Injectable:**

@Injectable({ providedIn: DataModule })

*export* class DataService {}

У цьому випадку сервіс буде доступний тільки в модулі DataModule.

**Область дії сервісу рівня модуля**

* Сервіс обмежений лише тим модулем, у якому зареєстрований.
* Навіть якщо два модулі використовують один і той самий сервіс, кожен із них матиме свій окремий екземпляр.

**3. Рівень компонента (Component Level)**

Сервіс із рівнем компонента доступний тільки для поточного компонента. Якщо компонент має кілька екземплярів, кожен із них отримає власний екземпляр сервісу.

**Реєстрація сервісу на рівні компонента**

Сервіс додається до секції providers компонента:

@Component({

providers: [DataService]

})

*export* class DataComponent {}

#### **Область дії сервісу рівня компонента**

* Сервіс обмежений поточним компонентом.
* Для кожного екземпляра компонента створюється власний екземпляр сервісу.

### **Чому потрібна ієрархія сервісів?**

Ієрархія сервісів надає гнучкість у розподілі функціональності між різними частинами програми:

* **Ефективність:** Кореневий сервіс використовується там, де потрібен один екземпляр для всієї програми.
* **Локалізація:** Сервіси модуля або компонента ізолюють функціональність, запобігаючи конфліктам.
* **Модульність:** Дозволяє створювати незалежні функціональні модулі з власними сервісами.

### **Порівняння рівнів**

| **Рівень** | **Сфера дії** | **Кількість екземплярів** | **Приклад використання** |
| --- | --- | --- | --- |
| **Кореневий** | Уся програма | Один | Управління глобальними даними (напр., аутентифікація). |
| **Модульний** | Окремий модуль | Один на модуль | Управління даними модуля (напр., дані про товари). |
| **Компонентний** | Окремий компонент | Один на екземпляр компонента | Ізольовані обчислення або тимчасові дані для компонента. |

Ієрархія сервісів в Angular дозволяє контролювати їхню доступність і життєвий цикл, створюючи масштабовану та модульну архітектуру додатків. Залежно від потреби, можна вибирати відповідний рівень впровадження сервісу, забезпечуючи ефективне використання ресурсів і ізоляцію функціональності.

# Детальний огляд та призначення всіх структурних блоків Angular-додатку Service2

Angular-додаток складається з кількох структурних блоків, кожен з яких виконує конкретну роль. Розглянемо структуру додатку Service2 з прикладом сервісів AppCounterService та LocalCounterService, а також компонентів AppComponent і CounterComponent.

**1. AppModule**

Призначення: Головний модуль програми, який визначає, які компоненти та сервіси входять до програми, а також які модулі Angular використовуються.

*import* { NgModule } *from* '@angular/core';

*import* { BrowserModule } *from* '@angular/platform-browser';

*import* { FormsModule } *from* '@angular/forms';

*import* { AppComponent } *from* './app.component';

*import* { CounterComponent } *from* './counter/counter.component';

@NgModule({

imports: [ BrowserModule, FormsModule ],

declarations: [ AppComponent, CounterComponent],

bootstrap: [ AppComponent ],

providers: [],

})

*export* class AppModule { }

**Ключові блоки**:

* **declarations**: Містить оголошення компонентів, які належать до цього модуля.
* **imports**: Інші Angular-модулі, необхідні для роботи додатку.
* **providers**: Глобальні сервіси (немає необхідності, оскільки AppCounterService зареєстровано на рівні root).
* **bootstrap**: Компонент, з якого починається додаток.

**2. Сервіси**

*2.1. Глобальний сервіс AppCounterService*

Призначення: Надання спільного функціоналу для всього додатку (глобальна область видимості).

*import*{Injectable}*from* '@angular/core';

@Injectable({

providedIn: 'root'

})

*export* class AppCounterService{

counter=0;

increase() {

*this*.counter++

}

decrease() {

*this*.counter--

}

}

**Ключові блоки**:

* **@Injectable**: Декоратор, що вказує, що цей клас може бути інжектований як залежність.
* **providedIn: 'root'**: Доступність сервісу на глобальному рівні (у всьому додатку).
* Методи increase і decrease змінюють значення лічильника.

*2.2. Локальний сервіс LocalCounterService*

Призначення: Надання функціоналу, обмеженого до конкретного компонента (локальна область видимості).

*import*{Injectable}*from* '@angular/core';

@Injectable()

*export* class LocalCounterService{

counter=0;

increase() {

*this*.counter++

}

decrease() {

*this*.counter--

}

}

**Ключові блоки**:

* Відсутність providedIn: 'root' означає, що сервіс має бути зареєстрований у властивості providers відповідного компонента.

**3. Головний компонент AppComponent**

Призначення: Відображає інтерфейс головної сторінки додатку, інтегрує глобальний та локальний сервіси.

*import* { Component } *from* '@angular/core'

*import* { AppCounterService } *from* './services/app-counter.service';

*import* { LocalCounterService } *from* './services/local-counter.service';

@Component({

selector: 'app-root',

templateUrl: 'app.component.html',

styleUrls: ['app.component.css'],

providers: [LocalCounterService],

})

*export* class AppComponent {

constructor(

public appCounterService: AppCounterService,

public localCounterService: LocalCounterService

) {}

}

Шаблон

<h1>Компонент app.component.ts</h1>

<h2>Сервіс верхнього рівня App Counter {{appCounterService.counter}}</h2>

<button *class*="btn" *(click)*="appCounterService.increase()">+</button>

<button *class*="btn" *(click)*="appCounterService.decrease()">-</button>

<h2>Сервіс рівня компоненту Local Counter {{localCounterService.counter}}</h2>

<button *class*="btn" *(click)*="localCounterService.increase()">+</button>

<button *class*="btn" *(click)*="localCounterService.decrease()">-</button>

<hr/>

<app-counter></app-counter>

**Ключові блоки**:

* Інжекція сервісів AppCounterService (глобальний) та LocalCounterService (локальний).
* Взаємодія з шаблоном через прив’язку даних ({{ appCounterService.counter }}) та обробку подій ((click)).

**4. Додатковий компонент CounterComponent**

Призначення: Демонструє роботу з глобальним і локальним сервісами окремо.

*import* { Component } *from* '@angular/core'

*import* { AppCounterService } *from* '../services/app-counter.service';

*import* { LocalCounterService } *from* '../services/local-counter.service';

@Component({

selector: 'app-counter',

templateUrl: 'counter.component.html',

providers: [LocalCounterService],

})

*export* class CounterComponent {

constructor(

public appCounterService: AppCounterService,

public localCounterService: LocalCounterService

) {}

}

Шаблон

<h1>Компонент counter.component.ts</h1>

<h2>Сервіс верхнього рівня App Counter {{appCounterService.counter}}</h2>

<button *class*="btn" *(click)*="appCounterService.increase()">+</button>

<button *class*="btn" *(click)*="appCounterService.decrease()">-</button>

<h2>Сервіс рівня компоненту Local Counter {{localCounterService.counter}}</h2>

<button *class*="btn" *(click)*="localCounterService.increase()">+</button>

<button *class*="btn" *(click)*="localCounterService.decrease()">-</button>

<hr/>

**Ключові блоки**:

* **Інжекція сервісів**: Аналогічно до AppComponent.
* **Локальна область видимості**: Кожен екземпляр LocalCounterService працює незалежно у різних компонентах.

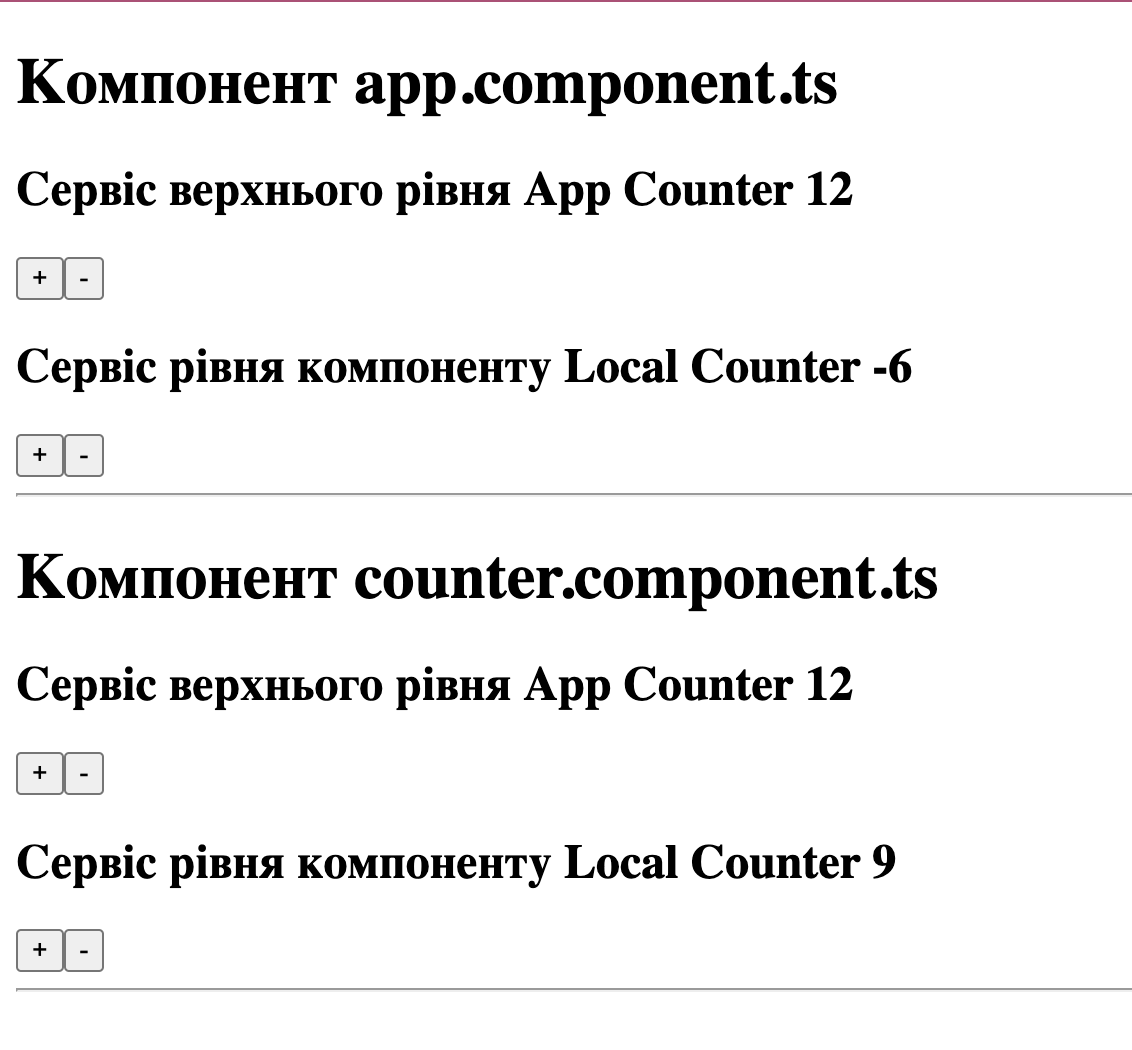
****

Рисунок 4.4. Результат

# Детальний огляд та призначення всіх структурних блоків Angular-додатку Service3

**Основні файли та їх призначення**

**src/app/app.component.ts**

Цей файл містить логіку головного компонента AppComponent, який виконує запит до сервісу для отримання користувачів і відображає їх у вигляді таблиці.

Призначення:

* Визначає властивість users$, яка є Observable, що містить дані користувачів.
* Використовує сервіс UserService для отримання даних.
* Використовує шаблон для відображення таблиці з користувачами.

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

*import* { Observable } *from* 'rxjs';

*import* { UserService } *from* './services/user.service';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

*export* class AppComponent {

users$: Observable<any>;

constructor(private userService: UserService) {

*this*.users$ = *this*.userService.getUsers();

}

}

**src/app/app.component.html**

Цей файл містить шаблон для відображення таблиці з користувачами. Дані виводяться через **async pipe(варіант 1, №94 у списку)**, який підписується на Observable users$ і автоматично отримує та відображає дані:

<div *class*="container mt-5">

<h1 *class*="mb-4">Список користувачів</h1>

<table *class*="table table-bordered">

<thead>

<tr>

<th>ID</th>

<th>Name</th>

<th>Username</th>

<th>Email</th>

<th>Phone</th>

<th>Website</th>

</tr>

</thead>

<tbody>

<tr *\*ngFor*="let user of users$ | async">

<td>{{ user.id }}</td>

<td>{{ user.name }}</td>

<td>{{ user.username }}</td>

<td>{{ user.email }}</td>

<td>{{ user.phone }}</td>

<td><a *[href]*="'http://' + user.website" *target*="\_blank">{{ user.website }}</a></td>

</tr>

</tbody>

</table>

</div>

**src/app/services/user.service.ts**

Цей файл містить логіку для отримання даних користувачів через HTTP-запит. В середині сервісу використовується HttpClient для виконання запиту до API https://jsonplaceholder.typicode.com/users.

Призначення:

* Здійснює запит до сервера за допомогою HTTP GET.
* Повертає Observable, щоб компоненти могли підписатися на потік даних.

*import* { Injectable } *from* '@angular/core';

*import* { HttpClient } *from* '@angular/common/http';

*import* { Observable } *from* 'rxjs';

@Injectable({

providedIn: 'root',

})

*export* class UserService {

private apiUrl = 'https://jsonplaceholder.typicode.com/users';

constructor(private http: HttpClient) {}

getUsers(): Observable<any> {

*return* *this*.http.get<any>(*this*.apiUrl);

}

}

**src/app/app.module.ts**

Цей файл є основним модулем вашого Angular-додатку, де реєструються всі компоненти, сервіси та модулі, які використовуються в додатку.

Призначення:

* Імпортує необхідні модулі, такі як BrowserModule для роботи з браузером і HttpClientModule для роботи з HTTP запитами.
* Декларує компоненти, які будуть використовуватися в додатку. typescript

*import* { NgModule } *from* '@angular/core';

*import* { BrowserModule } *from* '@angular/platform-browser';

*import* { HttpClientModule } *from* '@angular/common/http';

*import* { AppComponent } *from* './app.component';

@NgModule({

imports: [ BrowserModule, HttpClientModule ],

declarations: [AppComponent],

bootstrap: [ AppComponent ],

providers: [],

})

*export* class AppModule { }

**Пояснення основних аспектів**

Async pipe: дозволяє Angular підписуватися на Observable, автоматично отримувати дані та відображати їх, без потреби вручну підписуватись і відписуватись.

HttpClient: використовується для виконання HTTP запитів до API.

UserService: відповідає за отримання даних про користувачів.

AppComponent: використовує сервіс для отримання та відображення даних.

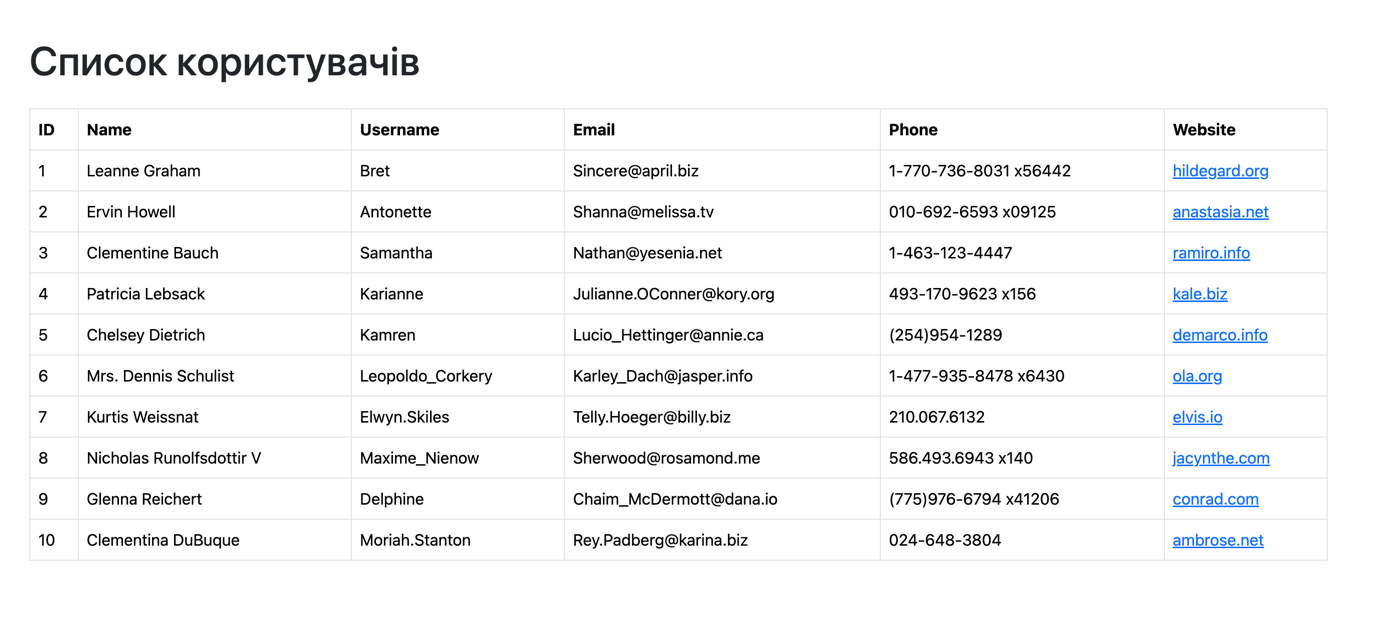


Рисунок 4.5. Результат

# ВИСНОВКИ

У процесі виконання лабораторної роботи було розглянуто механізми створення та впровадження сервісів у додатках Angular, досліджено різні аспекти використання сервісів на глобальному і локальному рівнях та їх роль у забезпеченні зручної і масштабованої архітектури додатків. Основні висновки:

**Функціональність сервісів Angular**. Сервіси надають можливість оптимального розподілу логіки між структурними елементами додатку. Вони використовуються для обробки даних, бізнес-логіки, а також як канали взаємодії між компонентами.

**Інтеграція сервісів**. Сервіси можна інтегрувати в інші сервіси та компоненти завдяки механізму ін’єкції залежностей, що спрощує взаємодію між різними частинами додатка. Логіка роботи з сервісами забезпечує модульність і повторне використання коду.

**Опціональні сервіси**. Використання декоратора @Optional дозволяє створювати сервіси, які не є обов’язковими. Це надає додаткову гнучкість у реалізації додатків, дозволяючи уникати помилок у разі відсутності деяких залежностей.

**Єдиний екземпляр сервісу**. Реєстрація сервісів на рівні головного модуля програми забезпечує створення одного екземпляра сервісу, який доступний для всіх компонентів. Це усуває проблему дублювання даних і знижує використання ресурсів.

**Ієрархія сервісів**. Можливість реєстрації сервісів на рівнях модуля, компонента та кореневого рівня забезпечує масштабованість і розширюваність додатків. Такий підхід дає змогу ізолювати логіку, якщо це необхідно, та зменшити ризик конфліктів.

**Досвід із реальним проєктом**. У виконаних вправах реалізовано практичний приклад роботи Angular-сервісів, включаючи створення, інтеграцію, ієрархічну організацію та використання глобальних і локальних сервісів. Це підтверджує їх ефективність у вирішенні прикладних завдань та демонструє ключову роль у проєктуванні сучасних веб-застосунків.

Ця лабораторна робота дозволила глибше зрозуміти архітектурні принципи Angular-додатків та підтвердила важливість сервісів у створенні модульного, гнучкого й продуктивного коду.

Додатки Service2 та Service3 були успішно розгорнуті на платформі Firebase у відповідних проектах з іменами <https://mieshkovip15laba4-2.web.app/> та <https://mieshkovip15laba4-3.web.app/> .

# 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>