

## **Лабораторне заняття №7: Сервіси. Створення локальних та глобальних сервісів.**

**Мета:** Навчитися створювати та використовувати сервіси у Angular.

**Завдання:** Створити два Angular-додатки під назвою Service1 та Service2.

**I) Для Angular-додатку Service1 виконати вправи 1-4;**

**II) Для Angular-додатку Service2 виконати самостійно вправу 5.**

**III) Зробити звіт по роботі.**

**IV) Angular-додаток Service2 розгорнути на платформі Firebase у проєкті з ім'ям «ПрізвищеГрупаLaba7», наприклад «KovalenkoIP01Laba7».**

### **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.

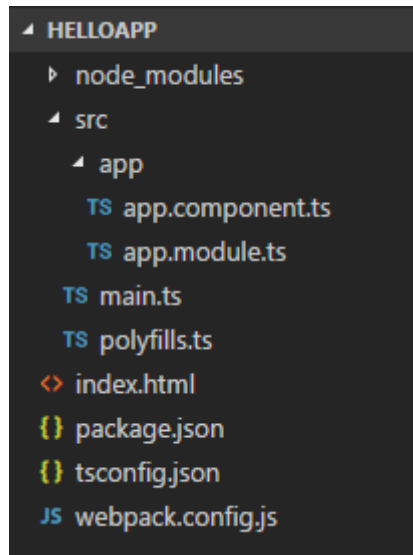
Стандартні завдання сервісів:

- 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.

**I) Для Angular-додатку Service1 виконати вправи 1-4.**

### **Вправа 1:**

Допустимо, у нас є наступна базова структура проєкту:



Додамо до папки `src/app` новий файл, який назвемо `phone.ts`. Визначимо у цьому файлі наступний клас:

```
export class Phone {  
  constructor(public name: string, public price: number) {}  
}
```

Цей клас представлятиме дані, з якими ми працюватимемо.

Далі додамо до папки `src/app` новий файл `data.service.ts`. Цей файл міститиме код сервісу. Згідно з прийнятими в Angular умовами в назві файлу після назви самого сервісу має йти підрядок `.service`. Якщо назва сервісу складається з кількох слів, які починаються з великої літери, то у назві файлу всі ці слова пишуться з маленької літери та поділяються дефісами. Наприклад, якщо сервіс називається `SpecialSuperHeroService`, то файл сервісу буде мати назву `special-super-hero.service.ts`.

Визначимо у цьому файлі наступний сервіс:

```
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))  
  }  
}
```

```
}
```

У сервісі визначено масив даних та методи для роботи з ним. У реальному додатку ці дані можна отримувати з сервера або будь-якого зовнішнього сховища.

Тепер визначимо код компонента:

```
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()
}
}

```

Щоб задіяти сервіс у компоненті, його не тільки треба імпортувати:

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

Але також необхідно його додати до колекції providers компонента:

```
providers: [DataService]
```

Усі сервіси, що використовуються, повинні бути визначені в колекції providers. І після цього ми зможемо задіяти вбудований у Angular механізм dependency injection та отримати об'єкт сервісу в конструкторі компонента і потім використовувати за потребою:

```
constructor(private dataService: DataService){}
```

Для завантаження даних реалізується метод ngOnInit() інтерфейсу OnInit, що викликається після конструктора.

І тому що ми використовуємо директиву ngModel для роботи з елементами форми, нам потрібно підключити модуль FormsModule у файлі app.module.ts:

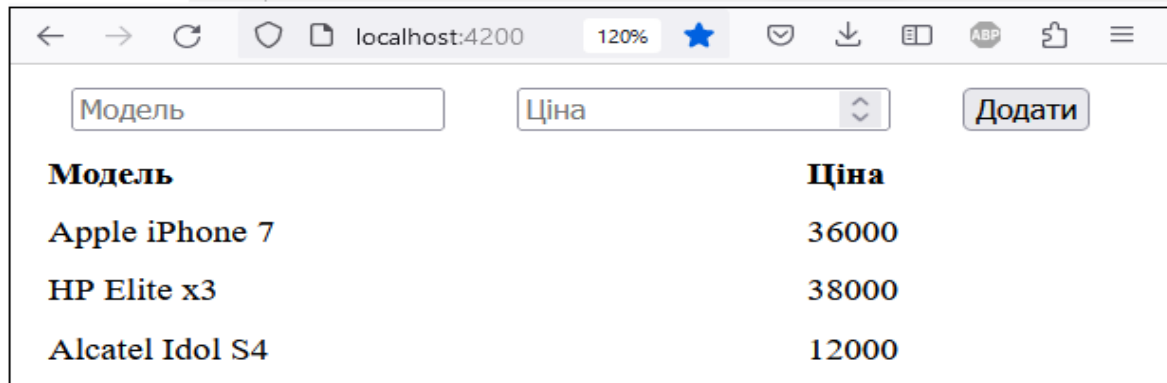
```

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 {}
```

У результаті ми зможемо виводити дані та додавати нові елементи через сервіс:



Модель	Ціна
Apple iPhone 7	36000
HP Elite x3	38000
Alcatel Idol S4	12000

Файл стилів для компонента `app.component.ts` має наступний вміст:

```
.cardinput{
  margin: 0.4rem 1.0rem;
}

td {
  padding: 5px; /* Поля вокруг текста */
  margin: 0.4rem 1.8rem;
}

table {
  width: 100%; /* Ширина таблицы */
  overflow:auto;
  color: rgb(0, 0, 0); /* Цвет текста */
  border-spacing: 1px; /* Расстояние между ячейками */
}

th {
  padding: 5px; /* Поля вокруг текста */
  margin: 0.4rem 1.8rem;
  text-align:left;
}
```

## **Вправа 2:**

### **Впровадження сервісу в інший сервіс**

Цілком ймовірна ситуація, коли ми захочемо використовувати один сервіс в іншому сервісі. Наприклад, у вправі 1 було створено сервіс для роботи з даними. Що якщо нам необхідно логувати всі операції з даними. Для логування визначимо новий сервіс. Для цього додамо до папки `src/app` новий файл `log.service.ts` з наступним вмістом:

```
export class LogService {
  write(logMessage: string) {
    console.log(logMessage)
  }
}
```

Для логування у сервісі визначено метод `write`, який виводить повідомлення на консоль.

Тепер використовуємо цей сервіс. Для цього змінимо код у файлі `data.service.ts`:

```
import { Injectable } from '@angular/core'
import { Phone } from './phone'
import { LogService } from './log.service'
@Injectable()
export class DataService {
  private data: Phone[] = [
    { name: 'Apple iPhone 7', price: 36000 },
    { name: 'HP Elite x3', price: 38000 },
    { name: 'Alcatel Idol S4', price: 12000 },
  ]
  constructor(private logService: LogService) {}

  getData(): Phone[] {
    this.logService.write('операція отримання даних')
    return this.data
  }
  addData(name: string, price: number) {
    this.data.push(new Phone(name, price))
    this.logService.write('операція додавання даних')
  }
}
```

Щоб вказати, що сервіс може використовувати інші сервіси, до класу сервісу застосовується декоратор `@Injectable`. Якщо клас не матиме такого декоратора, то вбудований механізм застосування залежностей не зможе створити об'єкт цього класу і видасть помилку.

Існує загальна рекомендація від розробників Angular застосовувати `@Injectable` до будь-якого класу сервісу, хоча це необов'язково.

Хоча у вправі 1 ми могли використовувати сервіс у компоненті без застосування до компоненту декоратора `@Injectable`. Справа в тому, що декоратор `@Component`, який застосовується до компонента, є підтипом `@Injectable`.

І також у випадку з `DataService` сервіс `LogService` також треба зареєструвати у списку провайдерів `AppComponent`:

```

import { Component, OnInit } from '@angular/core'
import { DataService } from './data.service'
import { Phone } from './phone'
import { LogService } from './log.service'

@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, LogService],
})
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()
}
}

```

І незважаючи на те, що безпосередньо LogService не використовується в компоненті AppComponent, але він використовується в DataService, який викликається в AppComponent.

І при виконанні операцій із даними в консолі браузера ми зможемо побачити роботу сервісу LogService.

The screenshot displays a web application running on localhost:4200. At the top, there is a form with an input field containing 'Samsung Galaxy S7', a price input field with '25000', and a 'Додати' (Add) button. Below the form is a table with two columns: 'Модель' (Model) and 'Ціна' (Price). The table lists five phone models: Apple iPhone 7 (36000), HP Elite x3 (38000), Alcatel Idol S4 (12000), and Samsung Galaxy S7 (25000). The bottom part of the image shows the browser's developer console with the 'Консоль' (Console) tab selected. It displays several log messages from 'webpack-dev-server' and two custom log messages from 'main.js': 'операція отримання даних' (data retrieval operation) and 'операція додавання даних' (data addition operation), both at line 120897.

Модель	Ціна
Apple iPhone 7	36000
HP Elite x3	38000
Alcatel Idol S4	12000
Samsung Galaxy S7	25000

### Вправа 3:

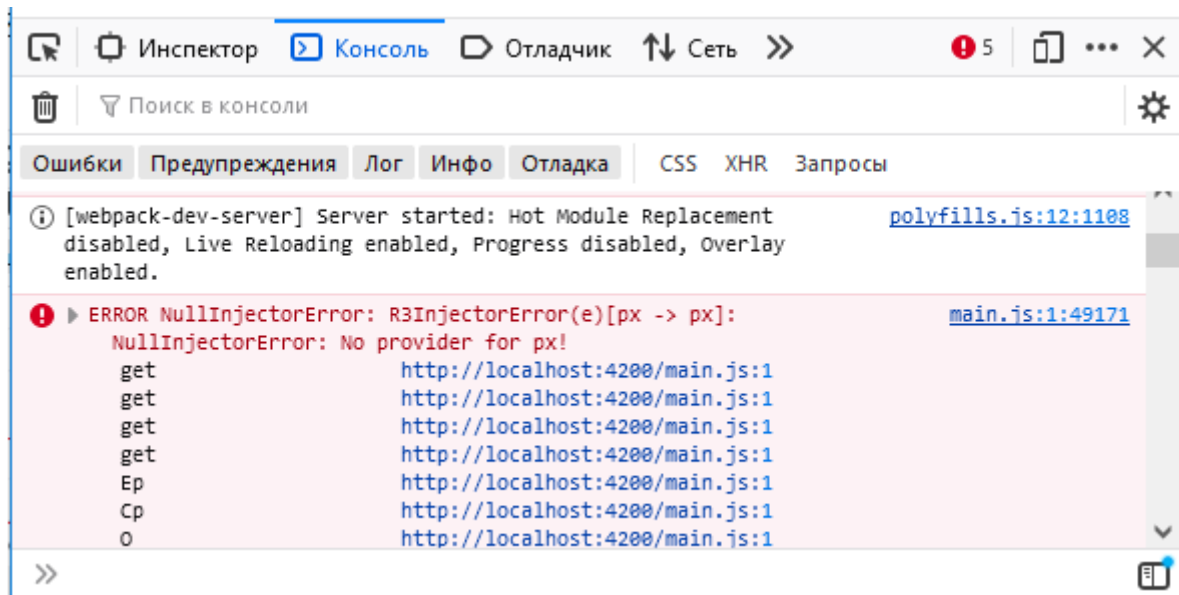


## Опціональні сервіси

Припустимо, з якоїсь причини сервіс LogService не доступний для інжектування, наприклад ми не додали в провайдери компонента AppComponent. Наприклад, буде зазначено так:

```
providers: [DataService],
```

Якщо ми запустимо програму, то в цьому випадку ми отримаємо помилку. Оскільки для сервісу LogService не визначено провайдера.



І тут можна визначити сервіс LogService як опціональний, застосовуючи декоратор Optional. Для цього змінимо код DataService:

```
import { Injectable, Optional } від '@angular/core'
import { Phone } from './phone'
import { LogService } from './log.service'
@Injectable()
export class DataService {
  private data: Phone[] = [
    { name: 'Apple iPhone 7', price: 36000 },
    { name: 'HP Elite x3', price: 38000 },
    { name: 'Alcatel Idol S4', price: 12000},
  ]

  constructor(@Optional() private logService: LogService) {}

  getData(): Phone[] {
    // this.logService.write('операція отримання даних')
    if (this.logService)
```

```

    this.logService.write('операція отримання даних')
    return this.data
  }
  addData(name: string, price: number) {
    this.data.push(new Phone(name, price))
    this.logService.write('операція додавання даних')
  }
}

```

Далі при зверненні до сервісу тепер потрібно перевірити, чи він встановлений, і якщо він встановлений, використовувати його:

```

if (this.logService)
  this.logService.write('операція отримання даних')

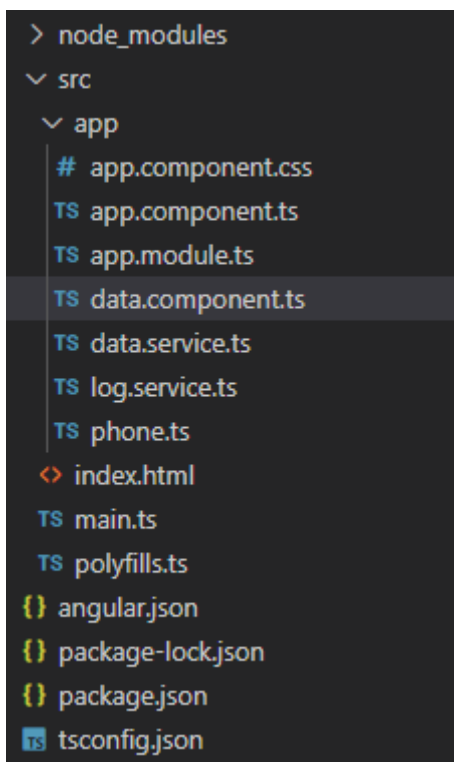
```

При цьому помилки вже не буде.

#### **Вправа 4:**

##### **Один сервіс для всіх компонентів**

Цілком можливо, що у нашому додатку буде кілька різних компонентів, які використовують сервіси. При цьому компоненти можуть використовувати ті самі сервіси. Наприклад, у попередніх вправах було створено сервіс DataService, який керує даними. Визначимо спеціальний компонент для роботи з даними. Для цього візьмемо проект із минулої вправи та додамо до папки src/app новий файл data.component.ts:



Визначимо у цьому файлі наступний код:

```
import{Component,OnInit}from '@angular/core'
import{DataService}from './data.service'
import{LogService}from './log.service'
import{Phone}from './phone'

@Component({
  selector: 'data-comp',
  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,LogService]
})
export class DataComponent 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()
}
}

```

DataComponent завантажує та додає дані. Для роботи із сервісами декоратор Component визначає секцію providers:

```
providers: [DataService, LogService]
```

Використовуємо цей компонент DataComponent у головному компоненті AppComponent:

```

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

@Component({
  selector: 'my-app',
  template: `<data-comp></data-comp>
<data-comp></data-comp>`
})
export class AppComponent {}

```

Цей компонент через елемент data-comp викликає компонент DataComponent. Причому викликає двічі. Тобто для обробки кожного елемента створюватиметься свій об'єкт DataComponent.

І відповідно змінимо головний модуль програми AppModule:

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { DataComponent } from './data.component';
@NgModule({
  imports: [ BrowserModule, FormsModule ],
  declarations: [ AppComponent, DataComponent],
  bootstrap: [ AppComponent ]
})

```

```
export class AppModule { }
```

Тепер запусимо програму і спробуємо додати новий елемент:

Модель	Ціна
Apple iPhone 7	36000
HP Elite x3	38000
Alcatel Idol S4	12000
Samsung Galaxy S7	20000

Як видно на скріншоті, при додаванні в одному компоненті новий елемент додаватиметься тільки для цього компонента. Тому що у нас два окремі компоненти, і для кожного з них буде створюватись свій набір сервісів DataService та LogService.

Така поведінка не завжди може бути кращою. Можливо, потрібно, щоб компоненти використовували той самий об'єкт сервісу, замість створення різних сервісів кожного компонента. Для цього ми можемо зареєструвати всі сервіси не в компоненті, а в головному модулі AppModule:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { DataComponent } from './data.component';
import { DataService } from './data.service';
import { LogService } from './log.service';
@NgModule({
  imports: [ BrowserModule, FormsModule ],
  declarations: [ AppComponent, DataComponent ],
  providers: [ DataService, LogService ], // реєстрація сервісів
  bootstrap: [ AppComponent ]
})
```

```
export class AppModule { }
```

У цьому випадку ми вже можемо прибрати реєстрацію сервісів з DataComponent:

```
import{Component,OnInit}from '@angular/core'  
import{DataService}from './data.service'  
import{LogService}from './log.service'  
import{Phone}from './phone'
```

```
@Component({  
  selector: 'data-comp',  
  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'],
```

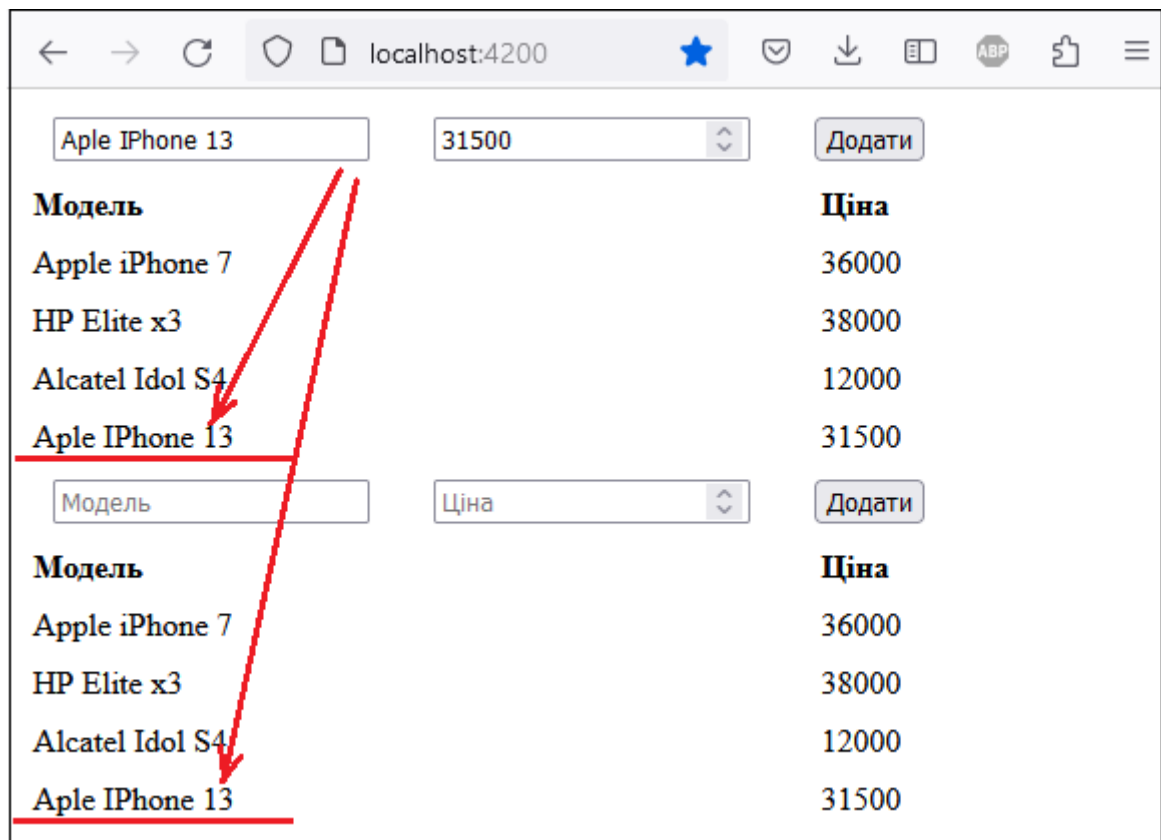
```

})
export class DataComponent 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()
  }
}

```

Після цього обидва об'єкти DataComponent будуть використовувати той самий сервіс DataService. Тому додавання об'єкта в одному компоненті автоматично позначиться на іншому:



### Ієрархія сервісів

Як було показано раніше, ми можемо впроваджувати сервіси у компонентах та модулях. Залежно від цього, де впроваджуються послуги, бувають різні рівні провайдерів

сервісів. Рівень по суті визначає сферу дії та життєвий цикл сервісу. Є три рівні провайдерів сервісів:

- 1) Глобальний чи кореневий рівень
- 2) Рівень модуля
- 3) Рівень компоненту

Для встановлення відповідного рівня сервісу є два способи: додавання сервісу до колекції `providers` у модулі або компоненті та встановлення рівня за допомогою параметра `providedIn` у декораторі `@Injectable`.

З одного боку, може здатися, що немає сенсу в такому розподілі - чому б для всіх сервісів не зробити один кореневий рівень, щоб один об'єкт сервісу був доступний для всього додатку на кшталт синглтона. Однак нерідко виникає необхідність розмежувати функціонал між окремими функціональними частинами додатка. Наприклад, коли два компоненти працюють із різним набором даних – в цьому випадку вони можуть використовувати один клас сервісу, але різні його об'єкти.

### **Кореневий рівень**

Кореневий рівень (root level) передбачає дію сервісу для всієї програми. Створюється та використовується один об'єкт сервісу для всіх частин програми.

Подібний рівень встановлюється, якщо сервіс додається до колекції `providers` головного модуля, який зазвичай називається `AppModule` і з якого починається робота програми. Наприклад:

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { AppComponent } from './app.component'
import { DataService } from './data.service'
@NgModule({
  imports: [BrowserModule, FormsModule],
  declarations: [AppComponent],
  providers: [DataService], // регистрация сервисов
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Якщо `AppModule` є головним модулем, то для сервісу `DataService` визначено кореневий рівень. Тобто під час роботи програми буде створюватись один об'єкт даного сервісу для всієї програми.



Значення root для параметра providedIn у декораторі Injectable також дозволяє встановити кореневий рівень дії:

```
import { Injectable } from '@angular/core'
@Injectable({ providedIn: 'root' })
export class DataService {
  private data: string[] = [
    'Apple iPhone XR',
    'Samsung Galaxy S9',
    'Nokia 9',
  ]
  constructor() {}

  getData(): string[] {
    return this.data
  }
  addData(name: string) {
    this.data.push(name)
  }
}
```

У цьому випадку ми можемо не додавати цей сервіс до колекції providers у головному модулі.

### Рівень модуля

Сервіси рівня модуля діють лише для поточного модуля і його класів - компонентів, директив, сервісів. Це все ті сервіси, які додаються до колекції providers у всіх інших модулях, крім головного модуля. Наприклад:

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { DataComponent } from './data.component'
import { DataService } from './data.service'
@NgModule({
  imports: [BrowserModule, FormsModule],
  declarations: [DataComponent],
  exports: [DataComponent],
  providers: [DataService], // регистрация сервисов
})
export class DataModule {}
```

Або ми можемо задати рівень модуля безпосередньо у сервісі:

```

import { Injectable } from '@angular/core'
import { DataModule } from './data.module'
@Injectable({ providedIn: DataModule }) // тепер сервіс доступний в модуль DataModule
export class DataService {
  private data: string[] = [
    'Apple iPhone XR',
    'Samsung Galaxy S9',
    'Nokia 9',
  ]
  constructor() {}

  getData(): string[] {
    return this.data
  }
  addData(name: string) {
    this.data.push(name)
  }
}

```

### Рівень компоненту

І тут дія сервісу обмежена поточним компонентом. Для кожного об'єкта компонента створюється об'єкт сервісу. Сам сервіс додається також до колекції providers компонента:

```

import { Component } from '@angular/core'
import { DataService } from './data.service'

@Component({
  selector: 'data-comp',
  template: `
    <div>
      <div>
        <input [(ngModel)]="name" placeholder="Модель" />
        <button (click)="addItem(name, price)">
          Додати
        </button>
      </div>
      <table>
        <tr *ngFor="let item of items">
          <td>{{ item }}</td>
        </tr>
      </table>
    </div>
  `,
  providers: [DataService], // додавання модуля DataService
})

```

```

export class DataComponent {
  items: string[] = []
  constructor(private dataService: DataService) {}

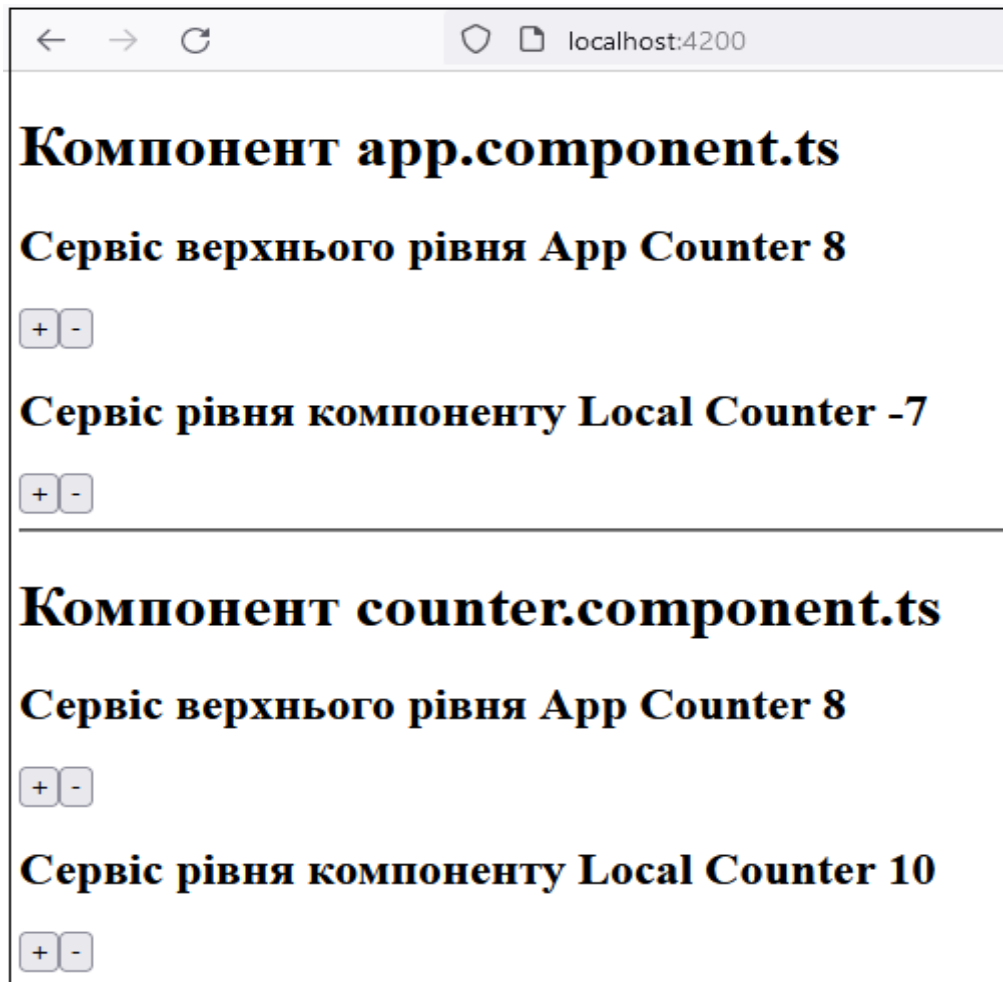
  addItem(name: string) {
    this.dataService.addData(name)
  }
  ngOnInit() {
    this.items = this.dataService.getData()
  }
}

```

II) Для Angular-додатку Service2 виконати самостійно вправу 5.

#### **Вправа 5: Виконати самостійно:**

Створіть новий проект Service2. Створіть папку app/services. У ній створіть два сервіси з різними областями видимості. Перший сервіс app-counter.service.ts глобального рівня. Інший сервіс local-counter.service.ts локального рівня. Використайте глобальний сервіс app-counter.service.ts та локальний сервіс local-counter.service.ts для двох компонентів app.component.ts та counter.component.ts як показано нижче (компонент counter.component.ts створіть самостійно):



## Виконання

- 1) Перший сервіс app-counter.services з класом AppCounterService кореневого рівня, який зареєстрований глобально та інжектований у компоненті app.component.ts має наступний вміст:

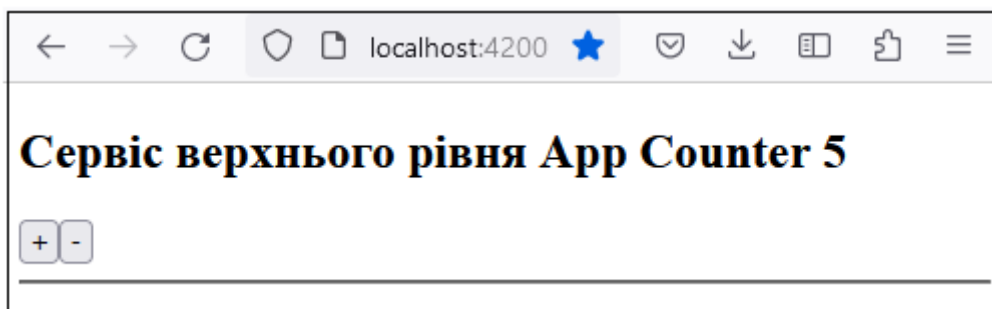
```
import{Injectable}from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class AppCounterService{
  counter=0;
  increase() {
    this.counter++;
  }
  decrease() {
    this.counter--;
  }
}
```

Шаблон app.component.html при цьому має наступний вміст:

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

<button class="btn" (click)="appCounterService.increase()">+</button>
<button class="btn" (click)="appCounterService.decrease()">-</button>
<hr/>
```



При цьому кнопки "+" та "-" працюють зменшуючи або збільшуючи значення counter.

- 2) Інший сервіс local-counter.services з класом LocalCounterService зареєстрований локально на рівні компоненту app.component.ts та інжектований в нього. Сервіс має наступний вміст:

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

```

@Injectable()
export class LocalCounterService{

  counter=0;
  increase() {
    this.counter++
  }
  decrease() {
    this.counter-
  }
}

```

Шаблон app.component.html при цьому має наступний вміст:

```

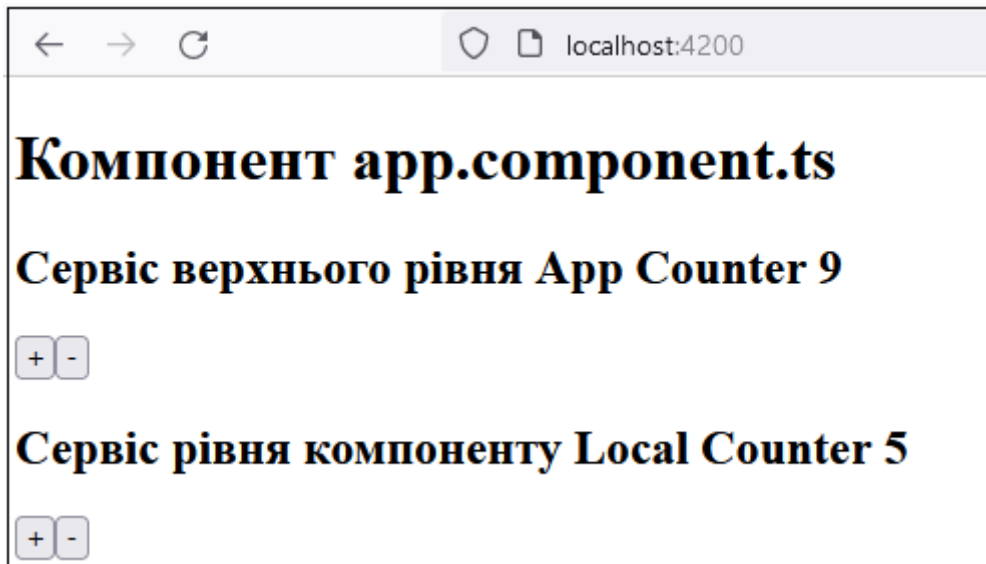
<h1>Компонент app.component.ts</h1>
<h2>Сервіс верхнього рівня App Counter {{appCounterService.counter}}</h2>

<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/>

```



3) Створіть окремий компонент у папці app/counter з ім'ям counter.component.ts та імпортуйте в нього сервіс глобального рівня app-counter.services. У шаблоні цього компонента повторіть логіку app.component.html.

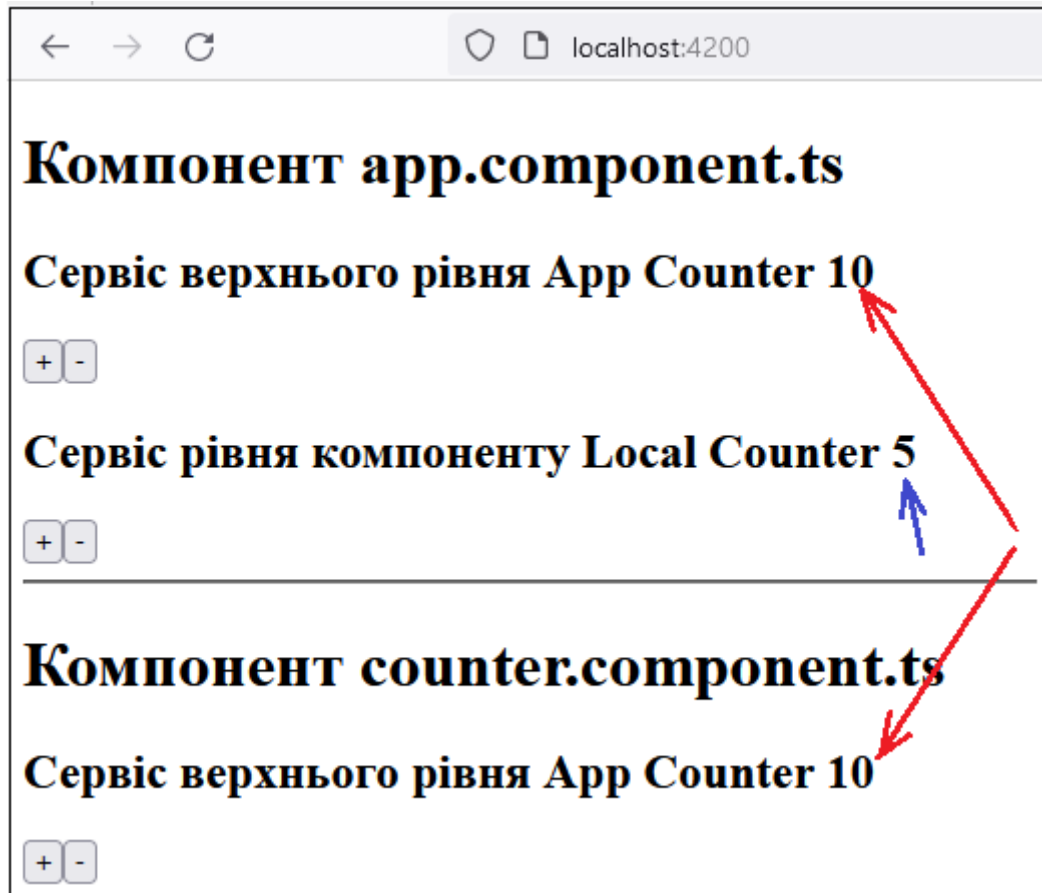
Після цього шаблон компонента counter.component.html матиме такий вигляд:

```

<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>
<hr/>

```



4) Окремо у властивості providers компонента counter.component.ts зареєструйте локальний сервіс LocalCounterService наступним чином:

```
providers:[LocalCounterService]
```

та інжектуйте його наступним чином:

```
constructor(public appCounterService:AppCounterService,
public localCounterService:LocalCounterService) {}
```

Змініть шаблон counter.component.html наступним чином:

```

<h1>Компонент counter.component.ts</h1>
<h2>Сервіс верхнього рівня App Counter {{appCounterService.counter}}</h2>

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

```

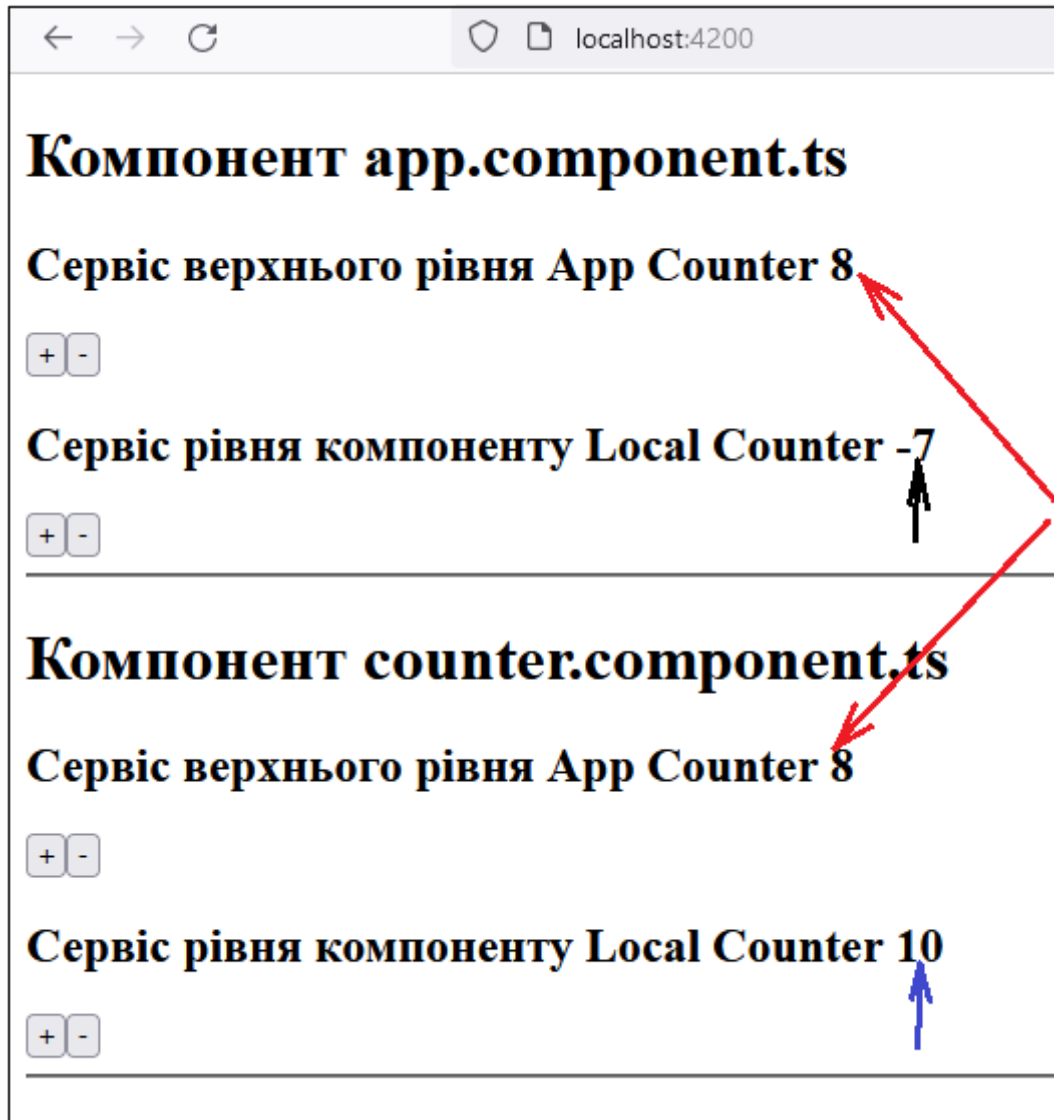
```
<button class="btn" (click)="appCounterService.decrease()">-</button>
```

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

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

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

```
<hr/>
```



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

- а) Сервіси: призначення та приклади використання;
- б) Впровадження сервісу в інший сервіс;
- в) Опціональні сервіси;
- г) Один сервіс для всіх компонентів;
- е) Ієрархія сервісів;

f) Детальний огляд та призначення всіх структурних блоків Angular-додатку Service2.

IV) Angular-додаток Service2 розгорнути на платформі Firebase у проекті з ім'ям «ПрізвищеГрупаLaba7», наприклад «KovalenkoIP01Laba7».



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

Факультет інформатики та обчислювальної техніки  
Кафедра інформатики та програмної інженерії

Звіт по лабораторній роботі №\_\_\_\_\_

---

назва лабораторної роботи

з дисципліни: «Реактивне програмування»

Студент: \_\_\_\_\_

Група: \_\_\_\_\_

Дата захисту роботи: \_\_\_\_\_

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

Захищено з оцінкою: \_\_\_\_\_

Київ, 2023