

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

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Звіт по лабораторній роботі № 3
Pipes. Створення та робота з pipes.
з дисципліни: «Реактивне програмування»

Студент: Головня Олександр Ростиславович
Група: ІП-11
Дата захисту роботи: _____
Викладач: доц. Полупан Юлія Вікторівна
Захищено з оцінкою: _____

Київ, 2024

Зміст

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»	1
Вправа 1: Робота з pipes	3
Вбудовані pipes	4
Параметри в pipes	5
Форматування дат	5
Форматування чисел	6
Форматування валюти	7
Ланцюжки pipes	8
Створення своїх pipes	8
Передача параметрів	9
Завдання для самостійного виконання	11
Вправа 2: Pure та Impure Pipes.....	12
AsyncPipe	16
Вправа 3: Використання pipes для отримання даних з серверу.....	18
Створення проекту “Blog”	21
Передача параметрів у компонент. Створення постів.	25
Завдання для самостійного виконання	37
Висновок:.....	40
Список використаних джерел:	41

Мета: Навчитися створювати та використовувати pipes у Angular.

Завдання: Створити чотири Angular-додатки під назвою Pipes1, Pipes2, Pipes3 та Blog.

1. Для Angular-додатку Pipes1 виконати вправу 1 (разом зі самостійним завданням);
2. Для Angular-додатку Pipes2 виконати вправу 2;
3. Для Angular-додатку Pipes3 виконати вправу 3.
4. Створити проект “Blog”, в який додати два компоненти post та post-form.

Компонент post-form – для створення нового поста. Компонент post – для відображення існуючих постів. Створити pipe для фільтрації постів.

5. Зробити звіт по роботі. Звіт повинен включати: титульний лист, зміст, основна частина, список використаних джерел.
6. Angular-додатки Pipes1 та Blog розгорнути на платформі Firebase у проектах з ім'ям «ПрізвищеГрупаLaba3-1» та «ПрізвищеГрупаLaba3-4», наприклад «KovalenkoIP01Laba3-1» та «KovalenkoIP01Laba3-4».

Вправа 1: Робота з pipes

Із минулого комп'ютерного практикуму я запозичив частину проекту. Pipes представляють спеціальні інструменти, які дозволяють форматовувати значення, що відображаються. Наприклад, нам треба вивести певну дату:

```
HolovniatIP11Lab3 > pipes1 > src > app > TS app.component.ts > AppComponent
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    template: `
6      <div>Без форматування: {{myDate}}</div>
7      <div>З форматуванням: {{myDate | date}}</div>`
8  })
9
10 export class AppComponent {
11   myDate = new Date(1945, 3, 3);
12 }
13 }
```

Тут створюється дата, яка двічі виводиться у шаблоні (див. рис. 5.1). У другому випадку до дати застосовується форматування за допомогою класу DatePipe.

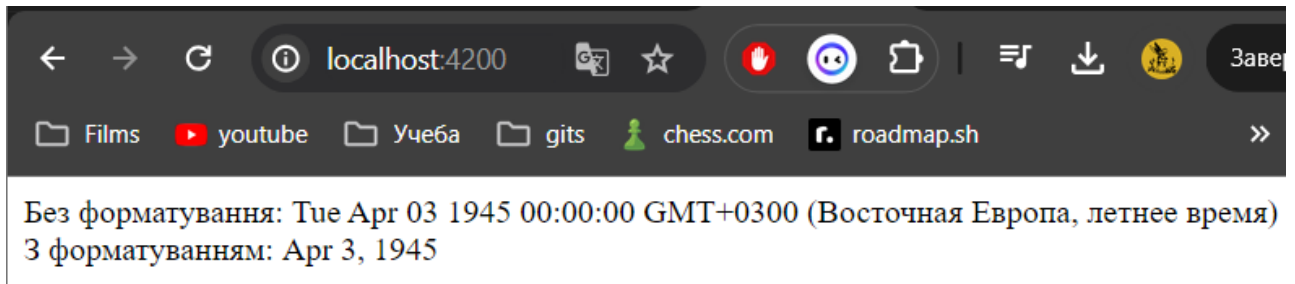


Рисунок 1.1. Результат створення та форматування дат.

Вбудовані pipes

При застосуванні класів суфікс Pipe відкидається (за винятком DecimalPipe - для застосування використовується назва "number"):

```
HolovniaIP11Lab3 > pipes1 > src > app > TS app.component.ts > AppComponent
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    template: `
6      <div>Без форматування: {{myDate}}</div>
7      <div>З форматуванням: {{myDate | date}}</div>
8
9      <div>{{welcome | uppercase}}</div>
10     <div>{{welcome | lowercase}}</div>
11     <div>{{percentange | percent}}</div>
12     <div>{{percentange | currency}}</div>
13   `
14 })
15
16 export class AppComponent {
17   myDate = new Date(1945, 3, 3);
18
19   welcome: string = "Hello World!";
20   percentange: number = 0.14;
21 }
```

Без форматування: Tue Apr 03 1945 00:00:00 GMT+0300 (Восточная Европа, летнее время)
З форматуванням: Apr 3, 1945
HELLO WORLD!
hello world!
14%
\$0.14

Рисунок 1.2. Результат використання вбудованих pipes

Параметри в pipes

Pipes можуть одержувати параметри. Наприклад, пайп SlicePipe, який обрізає рядок, може отримувати як параметр початковий і кінцевий індекси підрядка, який треба вирізати:

```
<div>{{welcome | slice:3}}</div>  
<div>{{welcome | slice:6:11}}</div>
```

Всі параметри в пайп передаються через двокрапку. У даному випадку slice:6:11 вирізає підрядок, починаючи з 6 до 11 індексу. При цьому якщо початок вирізу рядка обов'язково передавати, то кінцевий індекс необов'язковий. В цьому випадку як кінцевий індекс виступає кінець рядка.

Без форматування: Tue Apr 03 1945 00:00:00 GMT+0300 (Восточная Европа, летнее время) З форматуванням: Apr 3, 1945 HELLO WORLD! hello world! 14% \$0.14 lo World! World

Рисунок 1.3. Результат використання параметрів для pipe slice.

Форматування дат

DatePipe як параметр може приймати шаблон дати:

```
<div>{{myNewDate | date:"dd/MM/yyyy"}}</div>
```

```
myNewDate = Date.now();
```

```
Без форматування: Tue Apr 03 1945 00:00:00 GMT+0300 (Восточная Европа, летнее время)
З форматуванням: Apr 3, 1945
HELLO WORLD!
hello world!
14%
$0.14
lo World!
World
04/10/2024
```

Рисунок 1.4. Результат використання DatePipe з шаблоном.

Форматування чисел

DecimalPipe як параметр приймає формат числа у вигляді шаблону:

`{{ value | number [: digitsInfo [: locale]] }}`

value: саме значення, що виводиться

digitsInfo: рядок у форматі "minIntegerDigits.minFractionDigits-maxFractionDigits", де

- minIntegerDigits - мінімальна кількість цифр у цілій частині
- minFractionDigits - мінімальна кількість цифр у дробовій частині
- maxFractionDigits - максимальна кількість цифр у дробовій частині

locale: код застосовуваної культури

```
<div>{{pi | number:'2.1-2'}}</div>
<div>{{pi | number:'3.5-5'}}</div>
```

```
pi: number = 3.1415;
```

```
Без форматування: Tue Apr 03 1945 00:00:00 GMT+0300 (Восточная Европа, летнее время)
З форматуванням: Apr 3, 1945
HELLO WORLD!
hello world!
14%
$0.14
lo World!
World
04/10/2024
03.14
003.14150
```

Рисунок 1.5. Результат форматування чисел.

Форматування валюти

CurrencyPipe може приймати низку параметрів:

```
{{ value | currency[:currencyCode[:display[:digitsInfo[:locale]]]] }}
```

value: сума, що виводиться

currencyCode: код валюти згідно зі специфікацією ISO 4217. Якщо не вказано, то за замовчуванням застосовується USD

display: вказує, як відображати символ валюти. Може приймати такі значення:

- code: відображає код валюти (наприклад, USD)
- symbol (значення за промовчанням): відображає символ

валюти(наприклад, \$)

- symbol-narrow: деякі країни використовують як символ валюти кілька символів, наприклад, канадський долар - CA\$, цей параметр дозволяє отримати власне символ валюти - \$

- string: відображає довільний рядок
- digitsInfo: формат числа, який застосовується в DecimalPipe
- locale: код використовуваної локалі

```
<div>{{money | currency:'UA':'code'}}</div>
<div>{{money | currency:'UA':'symbol-narrow'}}</div>
<div>{{money | currency:'UA':'symbol':'1.1-1'}}</div>
<div>{{money | currency:'UA':'symbol-narrow':'1.1-1'}}</div>
<div>{{money | currency:'UA':'тільки сьогодні по ціні '}}</div>
```

```
money: number = 23.45;
```

Без форматування: Tue Apr 03 1945 00:00:00 GMT+0300 (Восточная Европа, летнее время)

З форматуванням: Apr 3, 1945

HELLO WORLD!

hello world!

14%

\$0.14

lo World!

World

04/10/2024

03.14

003.14150

UA23.45

UA23.45

UA23.5

UA23.5

тільки сьогодні по ціні 23.45

Рисунок 1.6. Результат використання CurrencyPipe з форматування валюти.

Ланцюжки pipes

Цілком можливо, що ми захочемо застосувати відразу кілька pipes до одного значення, тоді ми можемо скласти ланцюжки виразів, розділені вертикальною рисою:

```
<div>{{message | slice:6:11 | uppercase}}</div>
```

```
message = "Hello World!";
```

```
003.14150
UA23.45
UA23.45
UA23.5
UA23.5
тільки сьогодні по ціні 23.45
WORLD
```

Рисунок 1.7. Результат використання ланцюжків Pipe.

Створення своїх pipes

Якщо нам знадобиться деяка передобробка при виведенні даних, додаткове форматування, то ми можемо для цієї мети написати свої власні pipes. Класи pipes мають реалізувати інтерфейс PipeTransform

Метод transform має перетворити вхідне значення. Цей метод як параметр приймає значення, до якого застосовується pipe, а також опціональний набір параметрів. А на виході повертається відформатоване значення. Оскільки перший параметр представляє тип any, а другий параметр - масив типу any, то ми можемо передавати дані будь-яких типів. Також можемо повертати об'єкт будь-якого типу.

Розглянемо найпростіший приклад. Припустимо, нам треба виводити число, в якому роздільником між цілою та дробовою частиною є кома, а не точка. Для цього ми можемо написати маленький pipe. Для цього додамо до проекту до папки src/app новий файл format.pipe.ts:

Визначимо у цьому файлі наступний код:


```
HolovniaIP11Lab3 > pipes1 > src > app > TS format.pipe.ts > ...
1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({
4    name: 'format'
5  })
6  export class FormatPipe implements PipeTransform {
7    transform(value: number, args?: any): string {
8
9      return value.toString().replace(".", ",");
10   }
11 }
```

До кастомного pipe повинен застосовуватися декоратор Pipe. Цей декоратор визначає метадані, зокрема, назву pipe, за якою він використовуватиметься:

```
<div>Число до форматування: {{x}}<br>Число після форматування: {{x | format}}</div>
x: number = 15.45;
```

```
UA23.5
тільки сьогодні по ціні 23.45
WORLD
Число до форматування: 15.45
Число після форматування: 15,45
```

Рисунок 1.8. Результат використання створеного Pipe, який змінює «.» на «,»

Передача параметрів

Додамо ще один pipe, який прийматиме параметри. Нехай це буде клас, який з масиву рядків створюватиме рядок, приймаючи початковий та кінцевий індекси для вибірки даних із масиву. Для цього додамо до проекту новий файл join.pipe.ts, в якому визначимо наступний вміст:

```
HolovniaIP11Lab3 > pipes1 > src > app > TS join.pipe.ts > JoinPipe > transform
1  import { Pipe, PipeTransform } from '@angular/core';
2  @Pipe({
3    name: 'join'
4  })
5  export class JoinPipe implements PipeTransform {
6    transform(array: any, start?: any, end?: any): any {
7      let result = array;
8      if (start !== undefined) {
9        if (end !== undefined) {
10         result = array.slice(start, end);
11       }
12       else {
13         result = array.slice(start, result.length);
14       }
15     }
16     return result.join(", ");
17   }
18 }
```

У метод `transform` класу `JoinPipe` першим параметром передається масив, другий необов'язковий параметр `start` є початковим індексом, з якого проводиться вибірка, а третій параметр `end` - кінцевий індекс.

За допомогою методу `slice()` отримуємо потрібну частину масиву, а за допомогою методу `join()` з'єднуємо масив у рядок. Застосуємо `JoinPipe`:

```
<div>{{users | join}}</div>
<div>{{users | join:1}}</div>
<div>{{users | join:1:3}}</div>
```

```
users = ["Tom", "Alice", "Sam", "Kate", "Bob"];
```

Знову ж таки підключимо `JoinPipe` в модулі програми:

```

HolovniaIP11Lab3 > pipes1 > src > app > TS app.module.ts > AppModule
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { AppComponent } from './app.component';
4  import { FormatPipe } from './format.pipe';
5  import { JoinPipe } from './join.pipe';
6  import { SqrtPipe } from './sqrt.pipe';
7
8  @NgModule({
9    declarations: [
10     AppComponent, FormatPipe, JoinPipe, SqrtPipe
11   ],
12   imports: [
13     BrowserModule
14   ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }

```

Результат роботи:

```

WORLD
Число до форматування: 15.45
Число після форматування: 15,45
Tom, Alice, Sam, Kate, Bob
Alice, Sam, Kate, Bob
Alice, Sam

```

Рисунок 1.9. Результат роботи JoinPipe.

Завдання для самостійного виконання

Розробити ріре, який приймає в якості аргументу число і повертає число після отримання квадратного кореня.

Код створеного Pipe:

```
HolovniaIP11Lab3 > pipes1 > src > app > TS sqrt.pipe.ts > SqrtPipe
1  import { Pipe, PipeTransform } from '@angular/core';
2  @Pipe({
3    name: 'sqrt'
4  })
5  export class SqrtPipe implements PipeTransform {
6    transform(value: number): any {
7      return Math.sqrt(value);
8    }
9  }
```

```
<div>Extra task:</div>
```

```
<div>Square root of {{xSqrt}} = {{xSqrt | sqrt}}</div>
```

```
xSqrt: number = 25;
```

```
Число після форматування
Tom, Alice, Sam, Kate, Bo
Alice, Sam, Kate, Bob
Alice, Sam
Extra task:
Square root of 25 = 5
```

Рисунок 1.10. Результат роботи створеного SqrtPipe.

Вправа 2: Pure та Impure Pipes.

Pipes бувають двох типів: pure (що не допускають змін) та impure (допускають зміни). Відмінність між цими двома типами полягає у реагуванні на зміну значень, що передаються в pipe. За замовчуванням усі pipes є типом "pure". Такі об'єкти відстежують зміни у значеннях примітивних типів (String, Number, Boolean, Symbol). У інших об'єктах - типів Date, Array, Function, Object зміни відстежуються, коли змінюється посилання, але не значення за посиланням.

Тобто, якщо в масив додали елемент, масив змінився, але посилання змінної, яка представляє даний масив, не змінилася. Тому подібну зміну pure pipes не відстежуватиме. Impure pipes відстежують усі зміни. Можливо, постає питання, навіщо тоді потрібні pure pipes? Справа в тому, що відстеження змін позначається на продуктивності, тому pure pipes можуть показувати кращу продуктивність. До того ж не завжди необхідно відслідковувати зміни у складних об'єктах, іноді це не потрібно.

Тепер подивимося на прикладі. У вправі 1 було створено клас FormatPipe:

```
HolovniaIP11Lab3 > pipes1 > src > app > TS format.pipe.ts > ...
1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({
4    name: 'format'
5  })
6  export class FormatPipe implements PipeTransform {
7    transform(value: number, args?: any): string {
8
9      return value.toString().replace(".", ",");
10   }
11 }
```

За замовчуванням це pure pipe. А це означає, що він може відстежувати зміну значення, яке йому передається, оскільки воно є типом number. У компоненті ми могли динамічно змінювати значення, для якого виконується форматування:

```
HolovniaIP11Lab3 > pipes2 > src > app > TS app.component.ts > AppComponent > num
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    template: `
6      <input [(ngModel)]="num" name="fact">
7      <div>Результат: {{num | format}}</div>
8    `
9  })
10
11
12  export class AppComponent {
13    num: number = 15.45;
14  }
```

У файлі app.module.ts підключимо FormsModule, щоб використовувати двосторонню прив'язку:

```

HolovniaIP11Lab3 > pipes2 > src > app > TS app.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { FormsModule } from '@angular/forms';
4  import { AppComponent } from './app.component';
5  import { FormatPipe } from './format.pipe';
6  import { JoinPipe } from './join.pipe';
7
8  @NgModule({
9    imports: [ BrowserModule, FormsModule ],
10   declarations: [ AppComponent, FormatPipe, JoinPipe ],
11   bootstrap: [ AppComponent ]
12 })
13 export class AppModule { }

```

Тут жодних проблем із введенням би не виникло - змінюємо число в текстовому полі, і відразу змінюється форматований результат

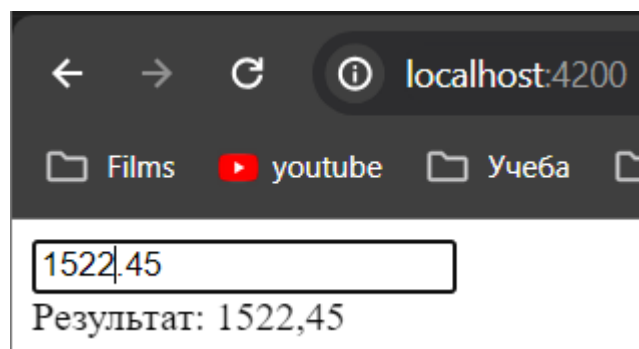


Рисунок 2.1. Демонстрація роботи по зміну формату.

Але в минулій темі був також створений інший pipe:

```
HolovniaIP11Lab3 > pipes2 > src > app > TS join.pipe.ts > ...
1  import { Pipe, PipeTransform } from '@angular/core';
2  @Pipe({
3    name: 'join'
4  })
5  export class JoinPipe implements PipeTransform {
6    transform(array: any, start?: any, end?: any): any {
7      let result = array;
8      if (start !== undefined) {
9        if (end !== undefined) {
10         result = array.slice(start, end);
11       }
12       else {
13         result = array.slice(start, result.length);
14       }
15     }
16     return result.join(", ");
17   }
18 }
```

Цей піп виконує операції над масивом. Відповідно, якщо в компоненті динамічно додавати нові елементи в масив, до якого застосовується JoinPipe, то ми не побачимо змін. Так як JoinPipe не відстежуватиме зміни над масивом. Тепер зробимо його impure pipe. Для цього додамо до декоратора Pipe параметр pure: false:

```
HolovniaIP11Lab3 > pipes2 > src > app > TS join.pipe.ts > JoinPipe > transform
1  import { Pipe, PipeTransform } from '@angular/core';
2  @Pipe({
3    name: 'join',
4    pure: false
5  })
6  export class JoinPipe implements PipeTransform {
7    transform(array: any, start?: any, end?: any): any {
8      return array.join(", ");
9    }
10 }
```

За замовчуванням параметр pure дорівнює true. Тепер ми можемо додавати в компонент нові елементи в цей масив:

```

HolovniaIP11Lab3 > pipes2 > src > app > TS app.component.ts > AppComponent > num
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    template: `
6      <input [(ngModel)]="num" name="fact">
7      <div>Результат: {{num | format}}</div>
8
9      <hr/>
10     <input #user name="user" class="form-control">
11     <button class="btn" (click)="users.push(user.value)">Add</button>
12     <p>{{users | join}}</p>
13
14   `
15 })
16
17 export class AppComponent {
18   num: number = 15.45;
19
20   users = ["Tom", "Alice", "Sam", "Kate", "Bob"];
21
22 }

```

І до всіх доданих елементів також застосовуватиметься JoinPipe:

15.45

Результат: 15,45

Sasha Add

Tom, Alice, Sam, Kate, Bob, Sasha

Рисунок 2.2. Результат роботи створеного impure Pipe.

Коли додається новий елемент, клас JoinPipe знову починає обробляти масив. Тому pipe застосовується до всіх елементів.

AsyncPipe

Одним із вбудованих класів, який на відміну від інших pipes вже за замовчуванням є тип impure. AsyncPipe дозволяє отримати результат асинхронної операції. AsyncPipe відстежує об'єкти Observable та Promise та

повертає отримане з цих об'єктів значення. При отриманні значення AsyncPipe сигналізує компонент про те, що треба перевірити зміни. Якщо компонент знищується, AsyncPipe автоматично відписується від об'єктів Observable і Promise, що унеможливорює можливі витoki пам'яті. Використовуємо AsyncPipe:

```
import { Observable, interval } from 'rxjs';  
import { map } from 'rxjs/operators';
```

```
<p>Модель: {{ phone | async }}</p>  
<button (click)="showPhones()">Переглянути моделі</button>
```

```
phones = ["iPhone 7", "LG G 5", "Honor 9", "Idol S4", "Nexus 6P"];  
phone: Observable<string>|undefined;  
constructor() { this.showPhones(); }  
showPhones() {  
  this.phone = interval(500).pipe(map((i:number)=> this.phones[i]));  
}
```

15.45

Результат: 15,45

Add

Tom, Alice, Sam, Kate, Bob

Модель: Honor 9


Рисунок 2.3. Результат роботи AsyncPipe.

Компонент не повинен підписуватись на асинхронне отримання даних, обробляти їх, а при знищенні відписуватись від отримання даних. Всю цю роботу робить AsyncPipe.

Вправа 3: Використання pipes для отримання даних з серверу

Оскільки AsyncPipe дозволяє легко витягувати дані з результату асинхронних операцій, його дуже зручно застосовувати, наприклад, при завантаженні даних з мережі.

У файлі http.service.ts визначимо сервіс, який отримує дані із сервера:

```
HolovniaIP11Lab3 > pipes3 > src > app > TS http.service.ts > 
1  import {Injectable} from '@angular/core';
2  import {HttpClient} from '@angular/common/http';
3  @Injectable()
4  export class HttpService{
5      constructor(private http: HttpClient){ }
6      getUsers(){
7          return this.http.get('assets/users.json');
8      }
9  }
10
```

Для зберігання даних у папці src/assets визначимо файл users.json:

```
HolovniaIP11Lab3 > pipes3 > src > assets > {} users.json > {} 2
1  [
2      {
3          "name": "Bob",
4          "age": 28
5      },
6      {
7          "name": "Tom",
8          "age": 45
9      },
10     {
11         "name": "Alice",
12         "age": 32
13     }
14 ]
```

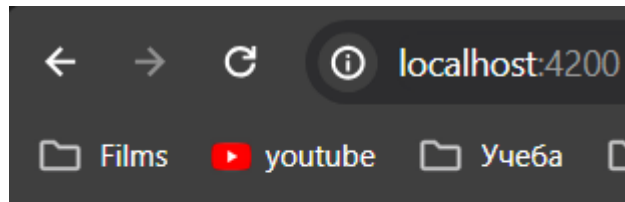
У файлі app.component.ts використовує сервіс:

```
HolovniaIP11Lab3 > pipes3 > src > app > TS app.component.ts > AppComponent >
1 import { Component, OnInit } from '@angular/core';
2 import { HttpService } from '../http.service';
3 import { Observable } from 'rxjs';
4
5 @Component({
6   selector: 'my-app',
7   template: `<ul>
8     <li *ngFor="let user of users | async">
9       <p>Ім'я користувача: {{user.name}}</p>
10      <p>Вік користувача: {{user.age}}</p>
11    </li>
12  </ul>`,
13   providers: [HttpService]
14 })
15 export class AppComponent implements OnInit {
16   users: Observable<Object>|undefined;
17   constructor(private httpService: HttpService){}
18   ngOnInit(){
19     this.users = this.httpService.getUsers();
20   }
21 }
```

Знову ж таки завантаження даних запускається в методі `ngOnInit()`. У шаблоні компонента до отриманих даних застосовується `AsyncPipe`:

```
<li *ngFor="let user of users | async">
```

І коли дані будуть отримані, вони відразу будуть відображені на веб-сторінці:



- Ім'я користувача: Bob
Вік користувача: 28
- Ім'я користувача: Tom
Вік користувача: 45
- Ім'я користувача: Alice
Вік користувача: 32

Рисунок 3.1. Відображення даних на веб-сторінці.

Щоб завантаження даних з мережі спрацювало, треба додати в AppModule модуль HttpClientModule:

```
HolovniaIP11Lab3 > pipes3 > src > app > TS app.module.ts > AppModule
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { HttpClientModule } from '@angular/common/http';
4  import { AppComponent } from './app.component';
5
6  @NgModule({
7    declarations: [AppComponent],
8    imports: [BrowserModule, HttpClientModule],
9    bootstrap: [AppComponent]
10 })
11 export class AppModule { }
```

Створення проекту “Blog”

1) Створіть проект “Blog” при допомозі команди: `ng new Blog`

```
PS C:\Users\Саша Головня\Desktop\Reactive programming\HolovniaIP11Lab3> ng new blog
? Which stylesheet format would you like to use? CSS [ https://developer
]
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/
CREATE blog/angular.json (2678 bytes)
CREATE blog/package.json (1074 bytes)
CREATE blog/README.md (1092 bytes)
CREATE blog/tsconfig.json (1045 bytes)
CREATE blog/.editorconfig (290 bytes)
CREATE blog/.gitignore (629 bytes)
CREATE blog/tsconfig.app.json (439 bytes)
CREATE blog/tsconfig.spec.json (449 bytes)
CREATE blog/.vscode/extensions.json (134 bytes)
CREATE blog/.vscode/launch.json (490 bytes)
CREATE blog/.vscode/tasks.json (980 bytes)
CREATE blog/src/main.ts (256 bytes)
CREATE blog/src/index.html (303 bytes)
CREATE blog/src/styles.css (81 bytes)
CREATE blog/src/app/app.component.html (20239 bytes)
CREATE blog/src/app/app.component.spec.ts (939 bytes)
CREATE blog/src/app/app.component.ts (313 bytes)
CREATE blog/src/app/app.component.css (0 bytes)
CREATE blog/src/app/app.config.ts (318 bytes)
CREATE blog/src/app/app.routes.ts (80 bytes)
CREATE blog/public/favicon.ico (15086 bytes)
✓ Packages installed successfully.
  Successfully initialized git.
PS C:\Users\Саша Головня\Desktop\Reactive programming\HolovniaIP11Lab3> |
```

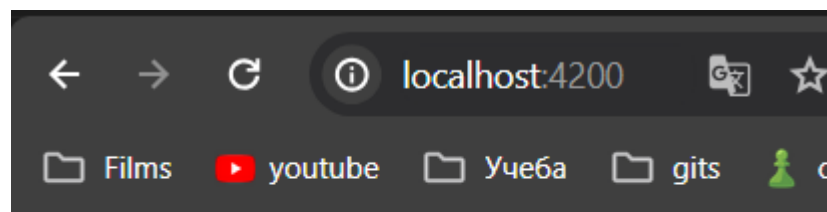
2) В файлі `app.component.html` створіть наступний вміст:

```
HolovniaIP11Lab3 > blog > src > app > <> app.component.html > div.container
1 <div class="container">
2   <h1>Angular Components</h1>
3 </div>
```

3) В файлі `src/styles.css` створіть наступний зміст:

```
HolovniaIP11Lab3 > Blog > src > # styles.css > input
1  @import url('https://fonts.googleapis.com/css?family=Roboto');
2  * {
3    box-sizing: border-box;
4    margin: 0;
5    padding: 0;
6  }
7  body {
8    font-family: 'Roboto', sans-serif;
9    font-size: 1rem;
10   line-height: 1.6;
11   background-color: #fff;
12   color: #333;
13 }
14 .container {
15   max-width: 1000px;
16   margin: 0 auto;
17   padding-top: 1rem;
18 }
19 a {
20   text-decoration: none;
21 }
22 a:hover {
23   color: #666;
24 }
25 ul {
26   list-style: none;
27 }
28 img {
29   width: 100%;
30 }
31 .btn {
32   display: inline-block;
33   background: #333333;
34   color: #fff;
35   padding: 0.4rem 1.3rem;
36   font-size: 1rem;
37   border: none;
38   cursor: pointer;
39   margin-right: 0.5rem;
40   transition: opacity 0.2s ease-in;
41   outline: none;
42 }
    .btn:hover {
    opacity: 0.8;
    }
    .form-control {
    display: block;
    margin-top: 0.3rem;
    }
    .card {
    padding: 1rem;
    border: #ccc 1px dotted;
    margin: 0.7rem 0;
    }
    input,
    select,
    textarea {
    display: block;
    width: 100%;
    padding: 0.4rem;
    font-size: 1.2rem;
    border: 1px solid #ccc;
    margin: 1.2rem 0;
    }
    hr {
    margin: .5rem 0;
    }
```

4) При допомозі вкладки «СЦЕНАРІЙ NPM» в Visual Studio Code запустіть проект на виконання. Ви отримаєте наступний результат (див. рис. 5.15):



Angular Components

Рисунок 4.1. Відображення веб-сторінки.

5) При допомозі Angular CLI створіть компонент post-form для створення нового поста. Для цього в терміналі в папці проекту введіть наступну команду:

Ng g c post-form --skip-tests

```
PS C:\Users\Саша Головня\Desktop\Reactive programming\HolovniaIP11Lab3\blog> Ng g c post-form
CREATE src/app/post-form/post-form.component.html (25 bytes)
CREATE src/app/post-form/post-form.component.spec.ts (630 bytes)
CREATE src/app/post-form/post-form.component.ts (257 bytes)
CREATE src/app/post-form/post-form.component.css (0 bytes)
```

6) При допомозі Angular CLI створіть компонент post для відображення існуючих постів. Для цього в терміналі в папці проекту введіть наступну команду:

Ng g c post --skip-tests

```
PS C:\Users\Саша Головня\Desktop\Reactive programming\HolovniaIP11Lab3\blog> Ng g c post
CREATE src/app/post/post.component.html (20 bytes)
CREATE src/app/post/post.component.spec.ts (601 bytes)
CREATE src/app/post/post.component.ts (238 bytes)
CREATE src/app/post/post.component.css (0 bytes)
```

Після цього відповідні класи створених компонентів будуть автоматично додані до модуля app.module.ts. Автоматично створені компоненти будуть мати селектори app-post-form та app-post відповідно.

7) В шаблоні компонента app.component.html виведіть два компонента у наступному виді:

```
HolovniaIP11Lab3 > blog > src > app > <> app.component.html > <div> div.container > <div> app-post
1  <div class="container">
2    <h1>Angular Components</h1>
3    <app-post-form></app-post-form>
4    <hr/>
5    <app-post></app-post>
6  </div>
```

8) В шаблоні, який призначений для створення нового посту блога, створіть два поля для введення даних посту “Title”, “Text” та кнопку для додавання нового посту. Наприклад так:

```
HolovniaIP11Lab3 > blog > src > app > post-form > <> post-form.component.html > div
1 <div>
2   <input type="text" class="form-control" placeholder="Title...">
3   <input type="text" class="form-control" placeholder="Text...">
4   <button class="btn">Додати пост</button>
5 </div>
```

9) В шаблоні, який призначений для відображення існуючих постів, виведіть існуючий (статичний) пост, використовуючи із файлу styles.css клас "card". Наприклад, так:

```
HolovniaIP11Lab3 > blog > src > app > post > <> post.component.html > div.card
1 <div class="card">
2   <h2>Post title</h2>
3   <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Dicta, atque.</p>
4 </div>
```

10) В шаблоні app.component.html виведіть дані компонентів post-form та post через їхні селектори app-post-form та app-post відповідно:

```
HolovniaIP11Lab3 > blog > src > app > <> app.component.html > div.container
1 <div class="container">
2   <app-post-form></app-post-form>
3   <hr/>
4   <app-post></app-post>
5   <app-post></app-post>
6
7 </div>
```

Додайте у файл Styles.css відступи padding-left, padding-right та color:brown для класу "container". Результат повинен бути наступний (див. рис. 5.16):

Post title
 Lorem ipsum dolor sit amet consectetur adipisicing elit. Dicta, atque.

Post title
 Lorem ipsum dolor sit amet consectetur adipisicing elit. Dicta, atque.

Рисунок 4.2. Відображення веб-сторінки з створеними компонентами.

Передача параметрів у компонент. Створення постів.

11) В компоненті `app.component.ts` створимо інтерфейс для визначення типів майбутніх об'єктів проекту і на його основі створимо масив постів.

```
HolovniaIP11Lab3 > blog > src > app > TS app.component.ts > ...
1  import { Component } from '@angular/core';
2  export interface Post {
3    title:string;
4    text:string;
5    id?:number;
6  }
7  @Component({
8    selector: 'app-root',
9    templateUrl: './app.component.html',
10   styleUrls: ['./app.component.css']
11 })
12
13 export class AppComponent {
14   posts: Post[]=[{title:'Вивчаю компоненти', text:'Створюю проект "Блог"', id:1},
15   {title:'Вивчаю директиви', text:'Все ще створюю "Блог"', id:2}]
16 }
```

12) В шаблоні `app.component.html` при допомозі структурної директиви `ngFor` у селекторі компонента, призначеного для виведення постів, виведемо створені статичні пости наступним чином:

```
HolovniaIP11Lab3 > blog > src > app > <> app.component.html > div.container > app-post
1  <div class="container">
2    <app-post-form></app-post-form>
3    <hr/>
4    <app-post>
5      *ngFor="let p of posts"
6      [myPost]="p"
7    </app-post>
8    <app-post></app-post>
9  </div>
```

13) В компоненті, який відповідає за відображення постів `post.component.ts` введемо нову змінну, при допомозі якої будемо приймати дані. Назвемо її `myPost`.

```
HolovniaIP11Lab3 > blog > src > app > post > TS post.component.ts > PostComponent > ngOnInit
1  import { Component, Input, OnInit } from '@angular/core';
2  import { Post } from '../app.component';
3
4  @Component({
5    selector: 'app-post',
6    templateUrl: './post.component.html',
7    styleUrls: ['./post.component.css']
8  })
9  export class PostComponent implements OnInit {
10    @Input() myPost!: Post;
11    constructor() { }
12    ngOnInit(): void {
13    }
14  }
```

```
HolovniaIP11Lab3 > blog > src > app > app.component.html > ...
1  <div class="container">
2    <app-post-form></app-post-form>
3    <hr/>
4    <app-post *ngFor="let p of posts" [myPost]="p">
5
6    </app-post>
7    <app-post></app-post>
8  </div>
```

```
HolovniaIP11Lab3 > blog > src > app > post > TS post.component.ts > PostComponent
1  import { Component, Input, OnInit } from '@angular/core';
2  import { Post } from '../app.component';
3
4  @Component({
5    selector: 'app-post',
6    templateUrl: './post.component.html',
7    styleUrls: ['./post.component.css']
8  })
9  export class PostComponent implements OnInit {
10    @Input('toPost') myPost!: Post;
11    constructor() { }
12    ngOnInit(): void {
13    }
14
15  }
```

14) В шаблоні post.component.html через змінну myPost ми приймаємо дані та виводимо їх на сторінку:

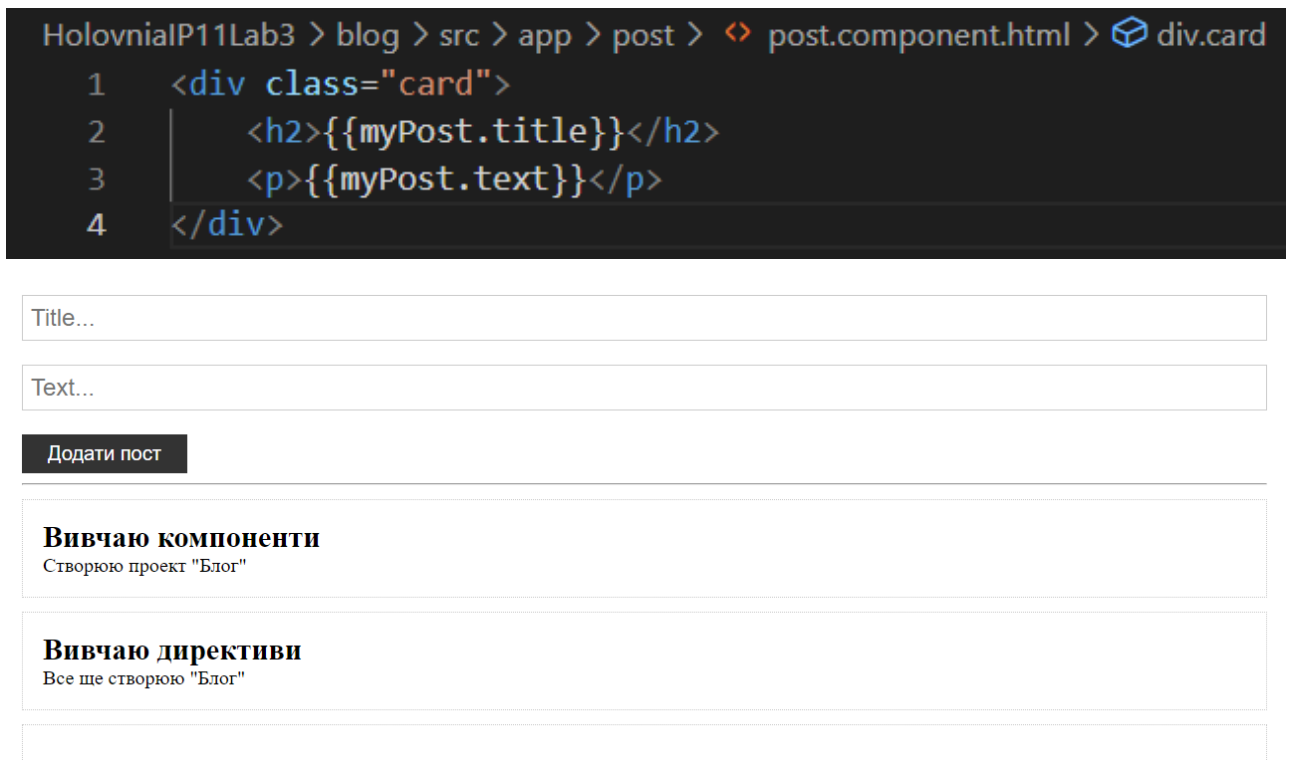
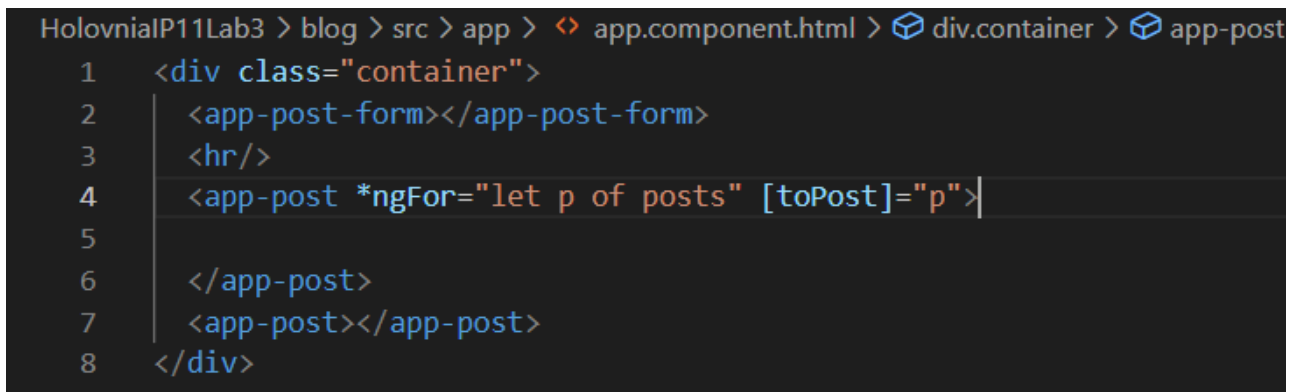


Рисунок 4.3. Відображення веб-сторінки з виводом даних на сторінку.

15) При чому, якщо ми захочемо передавати дані в шаблоні app.component.html не через змінну myPost, а наприклад через змінну toPost, так:



а в шаблоні post.component.html ми нічого не змінюємо, і продовжуємо працювати зі змінною myPost.

16) На наступному кроці реалізуємо додавання нового поста через компонент post-form з очисткою полів цієї форми після додавання та відображення нового поста через компонент post. Спочатку переконаємось, що в модулі app.module.ts у нас імпортований модуль FormsModule для реалізації двосторонньої прив'язки (two-way binding).

```
HolovniatP11Lab3 > blog > src > app > TS app.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { FormsModule } from '@angular/forms';
4  import { AppComponent } from './app.component';
5  import { PostFormComponent } from './post-form/post-form.component';
6  import { PostComponent } from './post/post.component';
7
8  @NgModule({
9    declarations: [AppComponent, PostFormComponent, PostComponent],
10   imports: [BrowserModule, FormsModule],
11   providers: [],
12   bootstrap: [AppComponent]
13 })
14 export class AppModule { }
```

17) В шаблоні post-form.component.html введемо директиву [(ngModel)] таким чином:

```
HolovniatP11Lab3 > blog > src > app > post-form > post-form.component.html > div
1  <div>
2    <input type="text" class="form-control" placeholder="Title..." [(ngModel)]="title">
3    <input type="text" class="form-control" placeholder="Text..." [(ngModel)]="text">
4    <button class="btn">Додати пост</button>
5  </div>
```

що буде означати, що зміна значення в цьому шаблоні (в полі title та text) призводить до миттєвої зміни значення в компоненті post-form.component.ts цього шаблону і навпаки. Зміна значення в компоненті post-form.component.ts призводить до миттєвої зміни значення в шаблоні post-form.component.html. Тобто, як тільки буде введено будь-яке значення в текстове поле, то воно одразу ж буде передаватися в компонент.

18) Додамо також до кнопки «Додати пост» обробник кліку з назвою addPost:

```
<button class="btn" (click)="addPost()">Додати пост</button>
```

19) В компоненті post-form.component.ts внесемо наступні зміни, щоб можна було оброблювати клік по кнопці «Додати пост»:

```
HolovniatP11Lab3 > blog > src > app > post-form > TS post-form.component.ts > PostFormComponent > constructor
1  import { Component, EventEmitter, OnInit, Output } from '@angular/core';
2  import { Post } from '../app.component';
3  import { Observable } from 'rxjs';
4
5  @Component({
6    selector: 'app-post-form',
7    templateUrl: './post-form.component.html',
8    styleUrls: ['./post-form.component.css']
9  })
10 export class PostFormComponent implements OnInit {
11   title='';
12   text='';
13   constructor() {}
14   ngOnInit(): void {}
15   addPost(){
16     if (this.title.trim()&&this.text.trim()){
17       const post: Post={
18         title:this.title,
19         text:this.text
20       }
21       console.log('New post',post);
22       this.title=this.text=''; // очищення полів
23     }
24   }
```

Метод trim() тут використовується для видалення пробілів з рядка title та text і якщо в ці змінні були передані не пусті значення, то ми заповнюємо змінну post. Після запуску проекту, внесення в поля title та text значень «Новий титл» і «Новий текст» та активізації кнопки «Додати пост» в консолі ми отримаємо наступний результат

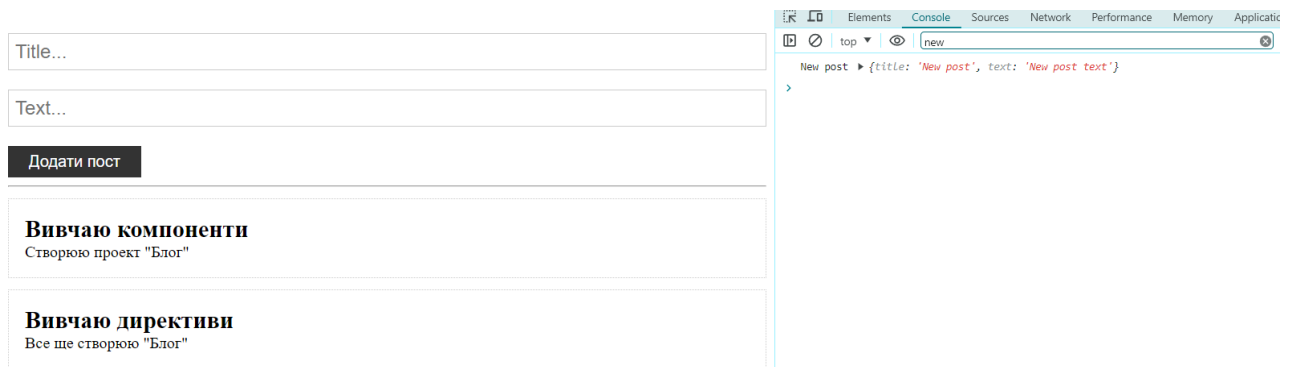


Рисунок 4.4. Відображення веб-сторінки та вивід в консолі.

20) На наступному кроці необхідно передати дані, отримані в змінній `post` дочірнього компонента `post-form.component.ts` в батьківський компонент `app.component.ts` в змінну `posts`. Якщо нам необхідно з одного компоненту щось відправляти назовні, то ми повинні використовувати декоратор `Output`. Завдання декораторів `Input` та `Output` полягає в обміні даними між компонентами. Вони є механізмом отримання/відправки даних від одного компонента до іншого. `Input` використовується для отримання даних, у той час як `Output` для їх надсилання. `Output` відправляє дані, виставляючи їх в якості виробників подій, зазвичай як об'єкти класу `EventEmitter`.

Тому в дочірньому компоненті `post-form.component.ts` із якого треба забрати дані і відправити в `app.component.ts` введемо нову змінну з декоратором `Output()`.

```
HolovniaIP11Lab3 > blog > src > app > post-form > TS post-form.component.ts > PostFormComponent >
4
5  ✓ @Component({
6    selector: 'app-post-form',
7    templateUrl: './post-form.component.html',
8    styleUrls: ['./post-form.component.css']
9  })
10  ✓ export class PostFormComponent implements OnInit {
11    @Output() onAdd: EventEmitter<Post> = new EventEmitter<Post>()
12    title='';
13    text='';
14    constructor() { }
15    ngOnInit(): void { }
16  ✓ addPost(){
17  ✓    if (this.title.trim() && this.text.trim()) {
18  ✓      const post: Post = {
19        title: this.title,
20        text: this.text
21      }
22      this.onAdd.emit(post);
23      console.log('New post', post);
24      this.title = this.text = ''; // очищення полів
25    }
26  }
27 }
```

Метод `eventEmitter.emit()` дозволяє передавати довільний набір аргументів функції слухача. При виклику звичайної функції слухача стандартне ключове слово `this` навмисно встановлюється для посилання на екземпляр `EventEmitter`, до якого прикріплений слухач.

21) Тепер в шаблоні app.component.html ми повинні прийняти дані в селекторі наступним чином:

```
HolovniaIP11Lab3 > blog > src > app > <> app.component.html > div.container > app-p
1 <div class="container">
2   <app-post-form (onAdd)="updatePosts($event)" ></app-post-form>
3   <hr/>
4   <app-post *ngFor="let p of posts" [toPost]="p"> </app-post>
5
6 </div>
```

22) А в компоненті необхідно створити новий метод для прийняття даних і оновлення масиву posts:

```
HolovniaIP11Lab3 > blog > src > app > TS app.component.ts > AppComponent > updatePosts
1 import { Component } from '@angular/core';
2 export interface Post {
3   title:string;
4   text:string;
5   id?:number;
6 }
7 @Component({
8   selector: 'app-root',
9   templateUrl: './app.component.html',
10  styleUrls: ['./app.component.css']
11 })
12 export class AppComponent {
13   posts: Post[]=[
14     {title:'Вивчаю компоненти', text:'Створюю проект "Блог"', id:1},
15     {title:'Вивчаю директиви', text:'Все ще створюю "Блог"', id:2}
16   ]
17   updatePosts(post:Post){
18     this.posts.unshift(post);
19   }
20 }
```

Метод unshift() додає один або більше елементів на початок масиву і повертає нову довжину масиву. Індекси всіх елементів, що спочатку присутні в масиві, збільшуються на одиницю (якщо методу було передано лише один аргумент) або на число, що дорівнює кількості переданих аргументів. Метод unshift() змінює вихідний масив, а чи не створює його модифіковану копію. В результаті будемо

мати можливість додавати пости, як на рисунку нижче. В консолі розробника, якщо до метода `updatePosts(post:Post)` додати виведення у консоль:

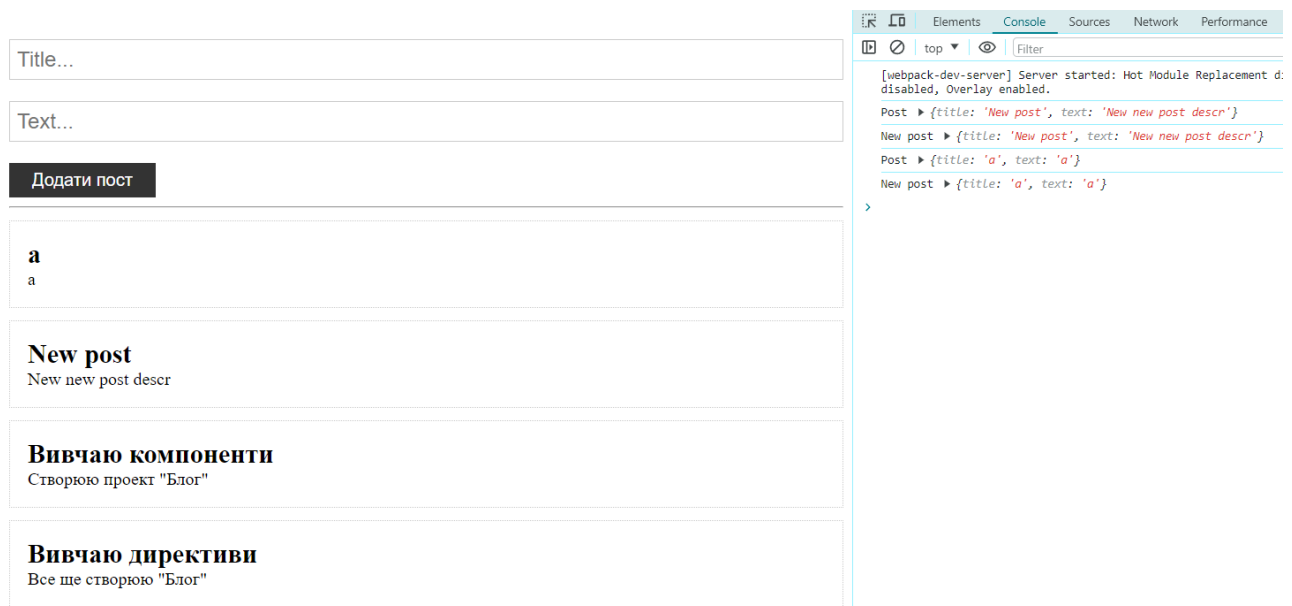


Рисунок 4.5. Відображення веб-сторінки з даними нових постів.

23) На наступному кроці реалізуємо логіку перевірки довжини поста, а саме довжину `Post.text`, і якщо довжина менше 20 символів, то виведемо нижче поста «Пост короткий», інакше «Пост довгий». Для цього в шаблоні `app.component.html` змінюємо вміст тега на наступне:

```
HolovniaIP11Lab3 > blog > src > app > <> app.component.html > div.container > app-post
1  <div class="container">
2    <app-post-form (onAdd)="updatePosts($event)" >/app-post-form>
3    <hr/>
4    <app-post *ngFor="let p of posts" [toPost]="p">
5      <small *ngIf="p.text.length>20; else short">Пост довгий</small>
6      <ng-template #short>
7        <small>Пост короткий</small>
8      </ng-template>
9    </app-post>
10
11 </div>
```

Angular директива `ng-template` 'відрисовує' Angular шаблон: це означає, що вміст цього тега міститиме частину шаблону, яка потім може бути використана разом з іншими шаблонами для формування остаточного шаблону компонента.

Директива `ng-template` використовується спільно з `ngIf`, `ngFor` та `ngSwitch` директивами.

24) Далі, щоб отобразити рядок «Пост короткий» та «Пост довгий» необхідно в шаблоні `post.component.html` використати елемент `<ng-content>`, який ще називають проекцією контенту, таким чином:

```
HolovniaIP11Lab3 > blog > src > app > post > <> post.component.html >
1 <div class="card">
2   <h2>{{myPost.title}}</h2>
3   <p>{{myPost.text}}</p>
4   <ng-content></ng-content>
5 </div>
```

`<ng-content>` дає можливість імпортувати HTML контент ззовні компонента та вставити його в шаблон іншого компонента в певне місце. В результаті будемо мати (див. рис.

5.20):

d
d
Пост короткий

Вивчаю компоненти
Створюю проект "Блог"
Пост довгий

Вивчаю директиви
Все ще створюю "Блог"
Пост довгий

Рисунок 4.6. Відображення веб-сторінки інформацією про пост(короткий/довгий).

25) Останнім кроком у нас буде додавання можливості для видалення будь-якого поста і демонстрація роботи метода ngOnDestroy()

Спочатку додамо в компонент, який відповідає за відображення постів post.component.ts метод ngOnDestroy() та імплементуємо відповідний інтерфейс OnDestroy.

```
HolovniaIP11Lab3 > blog > src > app > post > TS post.component.ts > PostComponent
1  import { Component, Input, OnDestroy, EventEmitter, Output,
2  import { Post } from '../app.component';
3  @Component({
4      selector: 'app-post',
5      templateUrl: './post.component.html',
6      styleUrls: ['./post.component.css']
7  })
8  export class PostComponent implements OnInit, OnDestroy {
9      @Input('toPost') myPost!: Post;
10     constructor() { }
11     ngOnInit(): void {
12     }
13     ngOnDestroy(){
14         console.log('метод ngOnDestroy');
15     }
16 }
17
```

26) Далі реалізуємо логіку видалення поста. Спочатку створимо кнопку в шаблоні post.component.html для видалення поста:

```
HolovniaIP11Lab3 > blog > src > app > post > post.component.html > div.card
1  <div class="card">
2      <h2>{{myPost.title}}</h2>
3      <p>{{myPost.text}}</p>
4      <button class="btn" (click)="removePost()">&times;</button>
5      <ng-content></ng-content>
6  </div>
```

27) Тепер додаємо логіку до компонента post.component.ts. Для прослуховування події вносимо параметр onRemove через декоратор Output, в який будемо передавати id поста:

```
HolovniaIP11Lab3 > blog > src > app > post > TS post.component.ts > PostComponent > removePost
1  import { Component, Input, OnDestroy, EventEmitter, Output, OnInit } from '@angular/core';
2  import { Post } from '../app.component';
3  @Component({
4    selector: 'app-post',
5    templateUrl: './post.component.html',
6    styleUrls: ['./post.component.css']
7  })
8  export class PostComponent implements OnInit, OnDestroy {
9    @Input('toPost') myPost!: Post;
10
11    constructor() { }
12    ngOnInit(): void {
13    }
14    ngOnDestroy() {
15      console.log('метод ngOnDestroy');
16    }
17    @Output() onRemove=new EventEmitter<number>()
18    removePost(){
19      this.onRemove.emit(this.myPost.id)
20    }
21
22  }
```

28) В app.component.html ми можемо прослухати подію onRemove у поста та видалити пост при допомозі метода deletePost, який нам ще треба створити в цьому компоненті:

```
HolovniaIP11Lab3 > blog > src > app > app.component.html > div.container
1  <div class="container">
2    <app-post-form (onAdd)="updatePosts($event)" ></app-post-form>
3    <hr/>
4    <app-post *ngFor="let p of posts" [toPost]="p" (onRemove)="deletePost($event)">
5      <small *ngIf="p.text.length>20; else short">Пост довгий</small>
6      <ng-template #short>
7        <small>Пост короткий</small>
8      </ng-template>
9    </app-post>
10
11  </div>
```

29) В app.component.ts створимо метод deletePost():

```

HolovniaIP11Lab3 > blog > src > app > TS app.component.ts > AppComponent > deletePost
1  import { Component } from '@angular/core';
2  export interface Post {
3      title:string;
4      text:string;
5      id?:number;
6  }
7  @Component({
8      selector: 'app-root',
9      templateUrl: './app.component.html',
10     styleUrls: ['./app.component.css']
11 })
12 export class AppComponent {
13     posts: Post[]=[
14         {title:'Вивчаю компоненти', text:'Створюю проект "Блог"', id:1},
15         {title:'Вивчаю директиви', text:'Все ще створюю "Блог"', id:2}
16     ]
17     updatePosts(post:Post){
18         this.posts.unshift(post);
19         console.log('Post',post);
20     }
21     deletePost(id:number){
22         this.posts=this.posts.filter(p=>p.id!==id)
23     }
24 }

```

В результаті, після видалення поста ми побачимо, що визивається метод `ngOnDestroy`, який часто використовується для того, щоб відписуватись від різних слухачів (див. рис. 5.22)

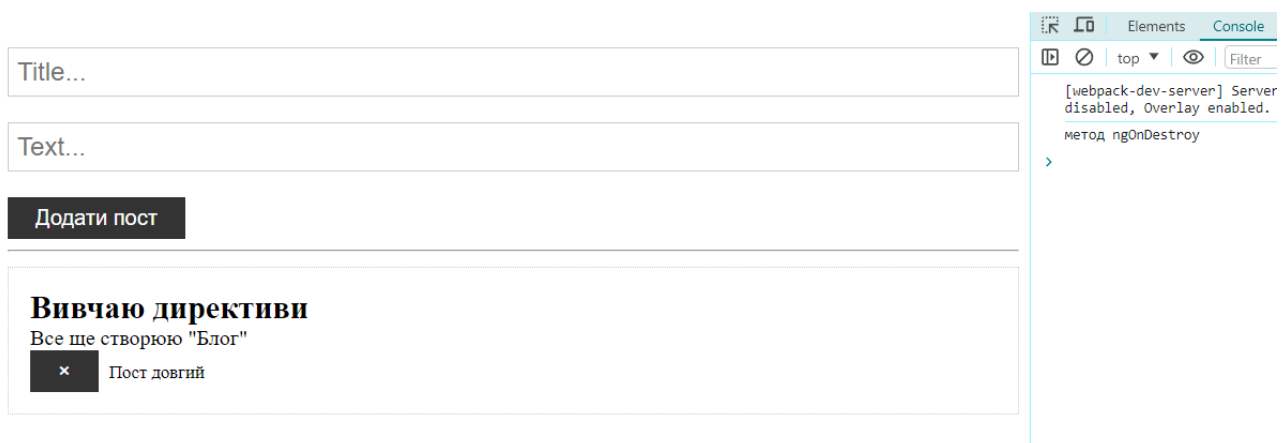


Рисунок 4.7. Демонстрація роботи методу `ngOnDestroy`, при натисканні на кнопку.

Завдання для самостійного виконання

Зараз, якщо додати декілька постів при допомозі кнопки «Додати пост», то при видаленні одного з них можна побачити, що будуть видалені всі додані пости. Для видалення конкретного посту, самостійно створіть додаткову логіку в веб-додатку для видалення конкретного посту

Створимо додаткову змінну в `app.component.ts`

```
counter = this.posts.length;
```

Далі присвоюємо Id поста цю змінну збільшену на 1:

```
updatePosts(post:Post){  
  post.id = this.counter++; // task  
  this.posts.unshift(post);  
  console.log('Post',post);  
}
```

Таким чином ID поста буде унікальним і оброблятиметься коректно, тобто видалятиметься лише один пост за раз.

Створення pipes для фільтрації постів у додатку

30) У додаток Blog додайте поле:

```
<input type="text" [(ngModel)]="search" placeholder="Search...">
```

При чому, якщо в полі Search введено рядок для пошуку і користувач паралельно додає новий пост, title якого включає наявний рядок для пошуку, то новий рядок повинен бути відображений в шаблоні. Наприклад при фільтрації постів по полю title (шукаємо вміст «pipes»), додаємо новий пост з title «Pipes для фільтрації», то новий пост повинен одразу відобразитися в шаблоні:

Додати пост

Вивчаю компоненти

Створюю проект "Блог"

×

Пост довгий

Рисунок 4.8. Результат роботи фільтрації постів через пошук.

Метод transform з двома параметрами можна оформити наступним чином:

```
HolovniaIP11Lab3 > blog > src > app > TS search.pipe.ts > SearchPipe
1 import { Pipe, PipeTransform } from '@angular/core';
2 import { Post } from './app.component';
3 @Pipe({
4   name: 'search',
5   pure: false
6 })
7 export class SearchPipe implements PipeTransform {
8   transform(posts: Post[], search: string = ''): Post[] {
9     if (!search.trim()) {
10       return posts;
11     }
12     return posts.filter(post => {
13       return post.title.toLowerCase().includes(search.toLowerCase()) || post.text.toLowerCase().includes(search.toLowerCase())
14     })
15   }
16 }
```

31) На сторінці при допомозі pipes виведемо поточні час та дату. При додаванні нового посту вказувати час та дату його створення.

В шаблоні компоненту post-form.component.html можна вивести час наступним чином:

```
HolovniaIP11Lab3 > blog > src > app > post-form > post-form.component.html > div > p
1 <div>
2   <p style="text-align:right">Дата: {{myDate$ | async | date:'HH:mm:ss dd:MM:yyyy'}}</p>
3   <input
4     type="text"
5     class="form-control"
6     placeholder="Title..."
7     [(ngModel)]="title">
8   <input
9     type="text"
10    class="form-control"
11    placeholder="Text..."
12    [(ngModel)]="text">
13   <button class="btn" (click)="addPost()">Додати пост</button>
14 </div>
```

Задавши при цьому параметр myDate\$ наступним чином:

```
HolovniatP11Lab3 > blog > src > app > post-form > TS post-form.component.ts > PostFormComponent > myDate$
1  import { Component, EventEmitter, OnInit, Output } from '@angular/core';
2  import { Post } from '../app.component';
3  import { Observable } from 'rxjs';
4
5  @Component({
6    selector: 'app-post-form',
7    templateUrl: './post-form.component.html',
8    styleUrls: ['./post-form.component.css']
9  })
10 export class PostFormComponent implements OnInit {
11   @Output() onAdd: EventEmitter<Post> = new EventEmitter<Post>()
12   title='';
13   text='';
14   myDate$:Observable<Date>=new Observable(obs=>{
15     setInterval(()=>{
16       obs.next(new Date())
17     },1000)
18   })
19 }
```

В шаблоні компоненту post.component.ts можна вивести час наступним чином:

```
HolovniatP11Lab3 > blog > src > app > post > post.component.html > div.card > p
1  <div class="card">
2    <p style="text-align:right">Дата: {{myPost.date | date:'HH:mm:ss dd:MM:yyyy'}}</p>
3    <h2>{{myPost.title}}</h2>
4    <p>{{myPost.text}}</p>
5    <button class="btn" (click)="removePost()">&times;</button>
6    <ng-content></ng-content>
7  </div>
```

Задавши при цьому параметр myPost.date так:

```
HolovniatP11Lab3 > blog > src > app > post-form > TS post-form.component.ts > PostFormComponent
8    styleUrls: ['./post-form.component.css']
9  })
10 export class PostFormComponent implements OnInit {
11   @Output() onAdd: EventEmitter<Post> = new EventEmitter<Post>()
12   title = '';
13   text = '';
14   date1!: Date;
15   constructor() { }
16   ngOnInit(): void {
17     this.myDate$.subscribe(date => {
18       this.date1 = date
19     })
20   }
21   addPost() {
22     if (this.title.trim() && this.text.trim()) {
23       const post: Post = {
24         title: this.title,
25         text: this.text,
26         date: this.date1
27       }
28       this.onAdd.emit(post);
29       console.log('New post', post);
30       this.title = this.text = ''; // очищення полів
31     }
32   }
33   myDate$: Observable<Date> = new Observable(obs => {
34     setInterval(() => {
35       obs.next(new Date())
36     }, 1000)
37   })
38 }
```

Search...

Дата: 21:02:41 04:10:2024

Title...

Text...

Додати пост

Дата: 21:02:36 04:10:2024

Вивчаю компоненти
Створюю проект "Блог"

×

Пост довгий

Дата: 21:02:36 04:10:2024

Вивчаю директиви
Все ще створюю "Блог"

×

Пост довгий

Рисунок 4.9. Результат виводу поточної дати та дат постів.

Посилання на додатки:

<https://holovniaip11lab3blog.web.app/>

<https://holovniaip11lab3pipes1.web.app/>

Висновок:

Під час виконання комп'ютерного практикуму я дізнався про pipes, їх призначення та використання, про ланцюжки pipes, про створення своїх власних pipes, про передачу параметрів у pipes, що таке pure та Impure pipes, а також asyncPipe. Мною було створено 4 додатка, три з яких демонструють особливості використання pipes, а інший додаток – це додаток Blog. Згідно завдання, розгорнув Angular-додатки «Pipes1» та «Blog» на платформі FireBase.

Список використаних джерел:

1. Understanding Pipes: <https://v17.angular.io/guide/pipes-overview>