Лабораторне занаття №6: Pipes. Створення та робота з pipes.

Мета: Навчитися створювати та використовувати pipes у Angular.

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

I) Для Angular-додатку Pipes1 виконати вправу 1 (разом зі самостійним завданням);

II) Для Angular-додатку Pipes2 виконати вправу 2;

III) Для Angular-додатку Pipes3 виконати вправу 3.

VI) Самостійно виконати вправу 4 (не створювати новий додаток, а у існуючому додатку Blog створити pipe для фільтрації постів).

VI) Зробити звіт по роботі.

VII) Angular-додатки Pipes1 та Blog розгорнути на платформі Firebase у проектах з ім'ям «ПрізвищеГрупаLaba6-1» та «ПрізвищеГрупаLaba6-4», наприклад «KovalenkoIP01Laba6-1» та «KovalenkoIP01Laba6-4».

I) Для Angular-додатку Pipes1 виконати вправу 1 (разом зі самостійним завданням);

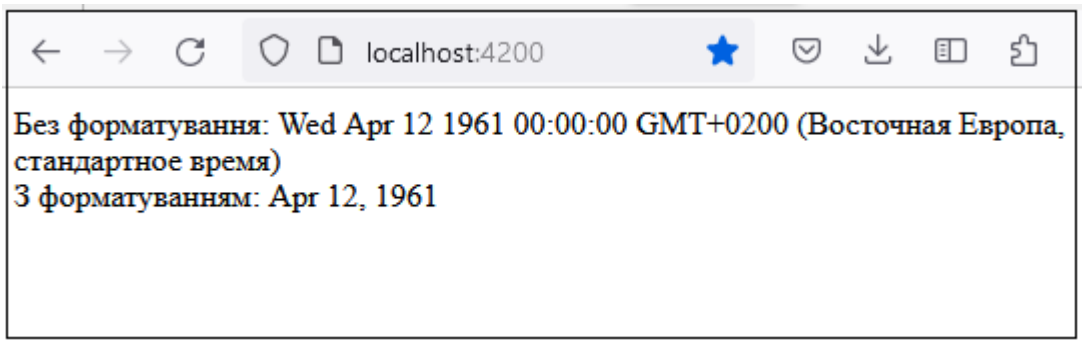
Вправа 1:

Робота з pipes

Pipes представляють спеціальні інструменти, які дозволяють форматувати значення, що відображаються. Наприклад, нам треба вивести певну дату:

```
import { Component} from '@angular/core';
@Component({
   selector: 'my-app',
   template: `<div>Без форматування: {{myDate}}</div>
            <div>З форматуванням: {{myDate | date}}</div>`
})
export class AppComponent {
   myDate = new Date(1961, 3, 12);
}
```

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



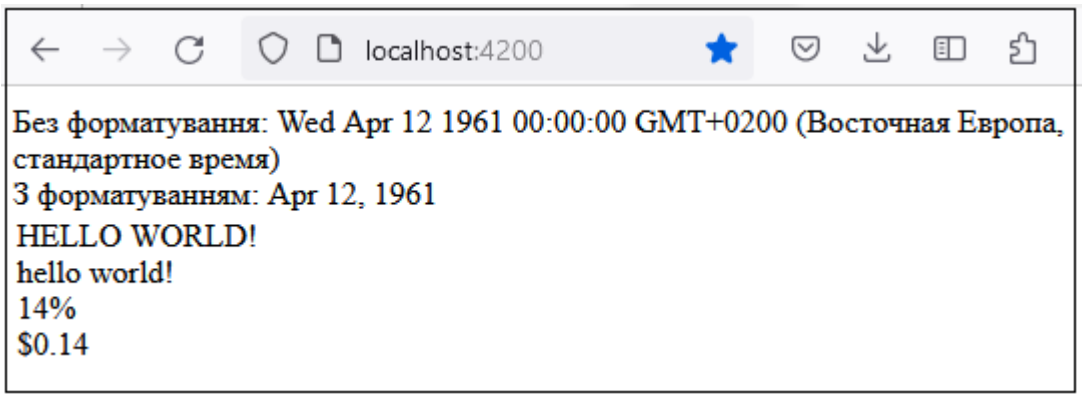
Вбудовані pipes

У Angular є ряд вбудованих pipes. Основні з них:

- CurrencyPipe: форматує валюту
- PercentPipe: форматує відсотки
- UpperCasePipe: переводить рядок у верхній регістр
- LowerCasePipe: переводить рядок у нижній регістр
- DatePipe: форматує дату
- DecimalPipe: задає формат числа
- SlicePipe: обрізає рядок

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

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

@Component({
   selector: 'my-app',
   template: `
            <div>Без форматування: {{myDate}}</div>
            <div>З форматуванням: {{myDate | date}}</div>
            <div>{{welcome | uppercase}}</div>
            <div>{{welcome | lowercase}}</div>
            <div>{{persentage | percent}}</div>
            <div>{{persentage | currency}}</div>`
})
export class AppComponent {
   myDate = new Date(1961, 3, 12);
   welcome: string = "Hello World!";
   persentage: number = 0.14;
}
```

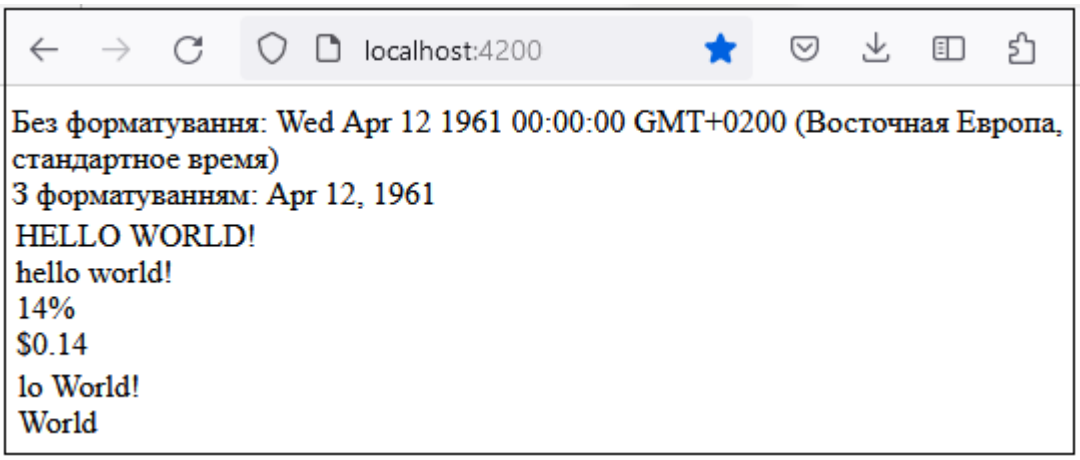


Параметри в pipes

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

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

@Component({
   selector: 'my-app',
   template: `
            <div>Без форматування: {{myDate}}</div>
            <div>З форматуванням: {{myDate | date}}</div>
            <div>{{welcome | uppercase}}</div>
            <div>{{welcome | lowercase}}</div>
            <div>{{persentage | percent}}</div>
            <div>{{persentage | currency}}</div>
            <div>{{welcome | slice:3}}</div>
            <div>{{welcome | slice:6:11}}</div>`
})
export class AppComponent {
   myDate = new Date(1961, 3, 12);
   welcome: string = "Hello World!";
   persentage: number = 0.14;
}
```

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.



Форматування дат

DatePipe як параметр може приймати шаблон дати:

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

@Component({
    selector: 'my-app',
    template: `
            <div>Без форматування: {{myDate}}</div>
            <div>3 форматуванням: {{myDate | date}}</div>
            <div>{{welcome | uppercase}}</div>
            <div>{{welcome | lowercase}}</div>
            <div>{{persentage | percent}}</div>
            <div>{{persentage | currency}}</div>
            <div>{{welcome | slice:3}}</div>
            <div>{{welcome | slice:6:11}}</div>
            <div>{{myNewDate | date:"dd/MM/yyyy"}}</div>
`
})
export class AppComponent {
    myDate = new Date(1961, 3, 12);
    welcome: string = "Hello World!";
    persentage: number = 0.14;
    myNewDate = Date.now();
}
```

Форматування чисел

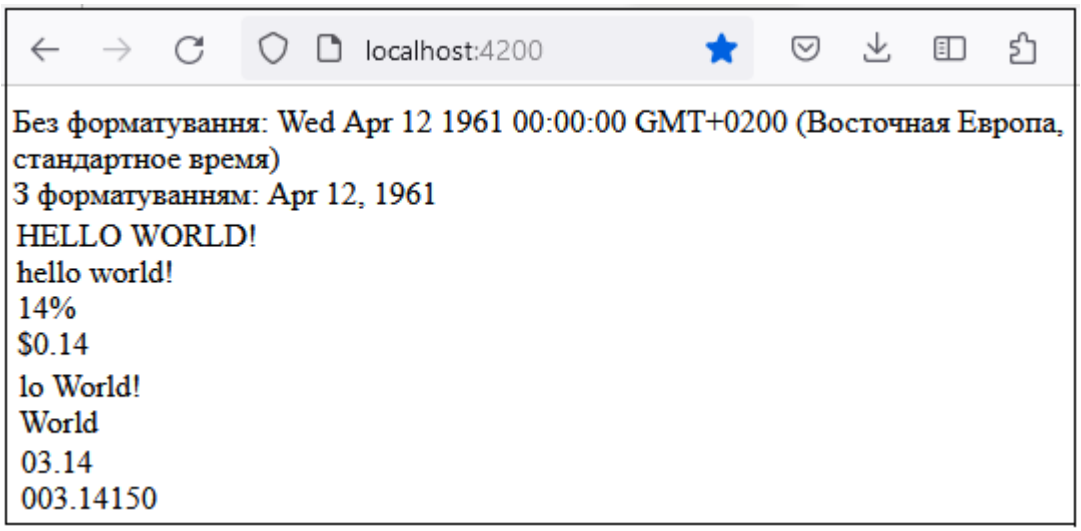
DecimalPipe як параметр приймає формат числа у вигляді шаблону:

```
{{ 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

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

@Component({
   selector: 'my-app',
   template: `
            <div>Без форматування: {{myDate}}</div>
            <div>З форматуванням: {{myDate | date}}</div>
            <div>{{welcome | uppercase}}</div>
            <div>{{welcome | lowercase}}</div>
            <div>{{persentage | percent}}</div>
            <div>{{persentage | currency}}</div>
            <div>{{welcome | slice:3}}</div>
            <div>{{welcome | slice:6:11}}</div>
            <div>{{myNewDate | date:"dd/MM/yyyy"}}</div>

   <div>{{pi | number:'2.1-2'}}</div>
   <div>{{pi | number:'3.5-5'}}</div>`
})
export class AppComponent {
   myDate = new Date(1961, 3, 12);
   welcome: string = "Hello World!";
   persentage: number = 0.14;
   myNewDate = Date.now();
   pi: number = 3.1415;
}
```



Форматування валюти

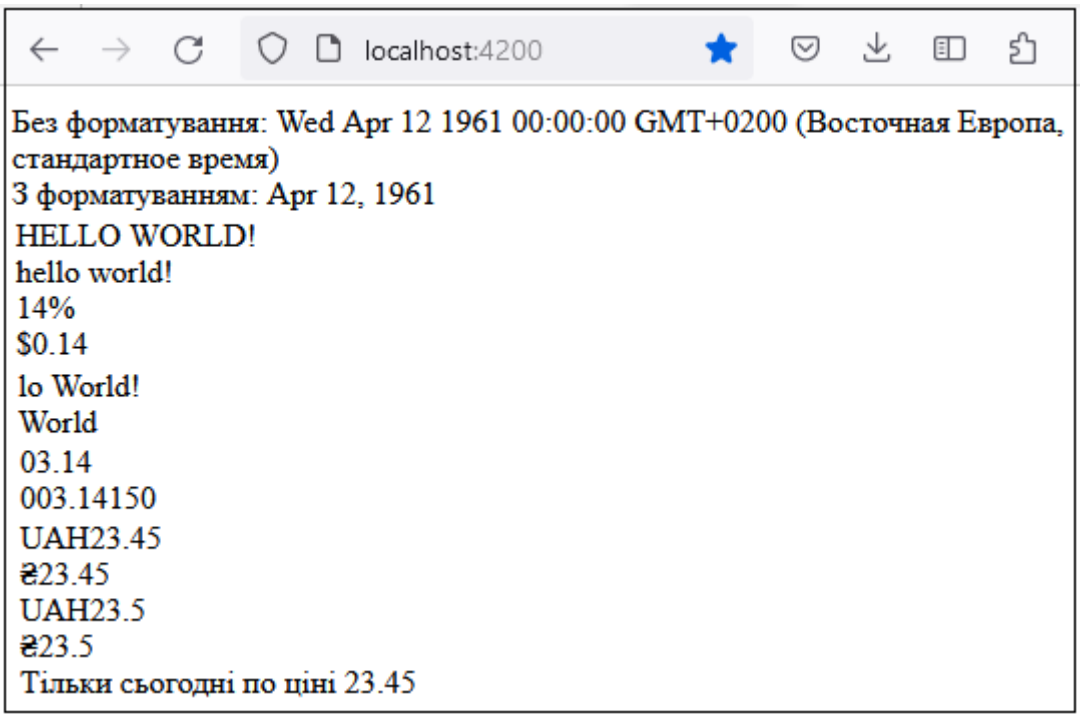
CurrencyPipe може приймати низку параметрів:

```
{{ 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

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

@Component({
  selector: 'my-app',
  template: `

      <div>Без форматування: {{myDate}}</div>
      <div>З форматуванням: {{myDate | date}}</div>
      <div>{{welcome | uppercase}}</div>
```

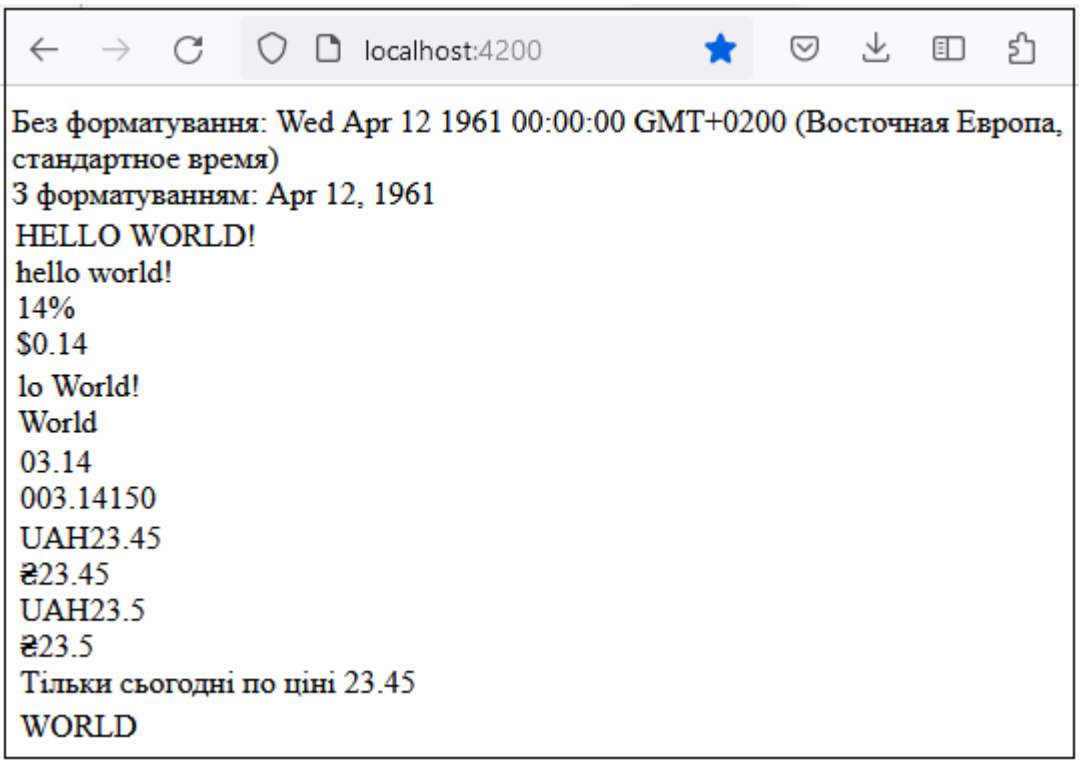
```
            <div>{{welcome | lowercase}}</div>
            <div>{{persentage | percent}}</div>
            <div>{{persentage | currency}}</div>
            <div>{{welcome | slice:3}}</div>
            <div>{{welcome | slice:6:11}}</div>
            <div>{{myNewDate | date:"dd/MM/yyyy"}}</div>
            <div>{{pi | number:'2.1-2'}}</div>
            <div>{{pi | number:'3.5-5'}}</div>

        <div>{{money | currency:'RUB':'code'}}</div>
        <div>{{money | currency:'RUB':'symbol-narrow'}}</div>
        <div>{{money | currency:'RUB':'symbol':'1.1-1'}}</div>
        <div>{{money | currency:'RUB':'symbol-narrow':'1.1-1'}}</div>
        <div>{{money | currency:'RUB':'тока седня по цене '}}</div>`
    })
    export class AppComponent {
        myDate = new Date(1961, 3, 12);
        welcome: string = "Hello World!";
        persentage: number = 0.14;
        myNewDate = Date.now();
        pi: number = 3.1415;

        money: number = 23.45;
    }
```



Ланцюжки pipes

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

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

@Component({
   selector: 'my-app',
   template: `
            <div>Без форматування: {{myDate}}</div>
            <div>З форматуванням: {{myDate | date}}</div>
            <div>{{welcome | uppercase}}</div>
            <div>{{welcome | lowercase}}</div>
            <div>{{persentage | percent}}</div>
            <div>{{persentage | currency}}</div>
            <div>{{welcome | slice:3}}</div>
            <div>{{welcome | slice:6:11}}</div>
            <div>{{myNewDate | date:"dd/MM/yyyy"}}</div>
            <div>{{pi | number:'2.1-2'}}</div>
            <div>{{pi | number:'3.5-5'}}</div>
   <div>{{money | currency:'RUB':'code'}}</div>
   <div>{{money | currency:'RUB':'symbol-narrow'}}</div>
   <div>{{money | currency:'RUB':'symbol':'1.1-1'}}</div>
   <div>{{money | currency:'RUB':'symbol-narrow':'1.1-1'}}</div>
   <div>{{money | currency:'RUB':'тока седня по цене '}}</div>
   <div>{{message | slice:6:11 | uppercase}}</div>`
})
export class AppComponent {
   myDate = new Date(1961, 3, 12);
   welcome: string = "Hello World!";
   persentage: number = 0.14;
   myNewDate = Date.now();
   pi: number = 3.1415;
   money: number = 23.45;

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



Створення своїх pipes

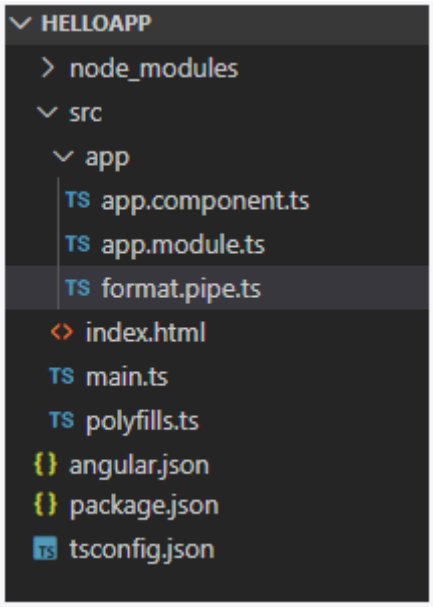
Якщо нам знадобиться деяка передобробка при виведенні даних, додаткове форматування, то ми можемо для цієї мети написати свої власні pipes.

Класи pipes мають реалізувати інтерфейс PipeTransform

```
interface PipeTransform {
transform(value: any, ...args: any[]): any
}
```

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:

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
    name: 'format'
})
export class FormatPipe implements PipeTransform {
  transform(value: number, args?: any): string {

    return value.toString().replace(".", ",");
  }
}
```

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

```
@Pipe({
    name: 'format'
})
```

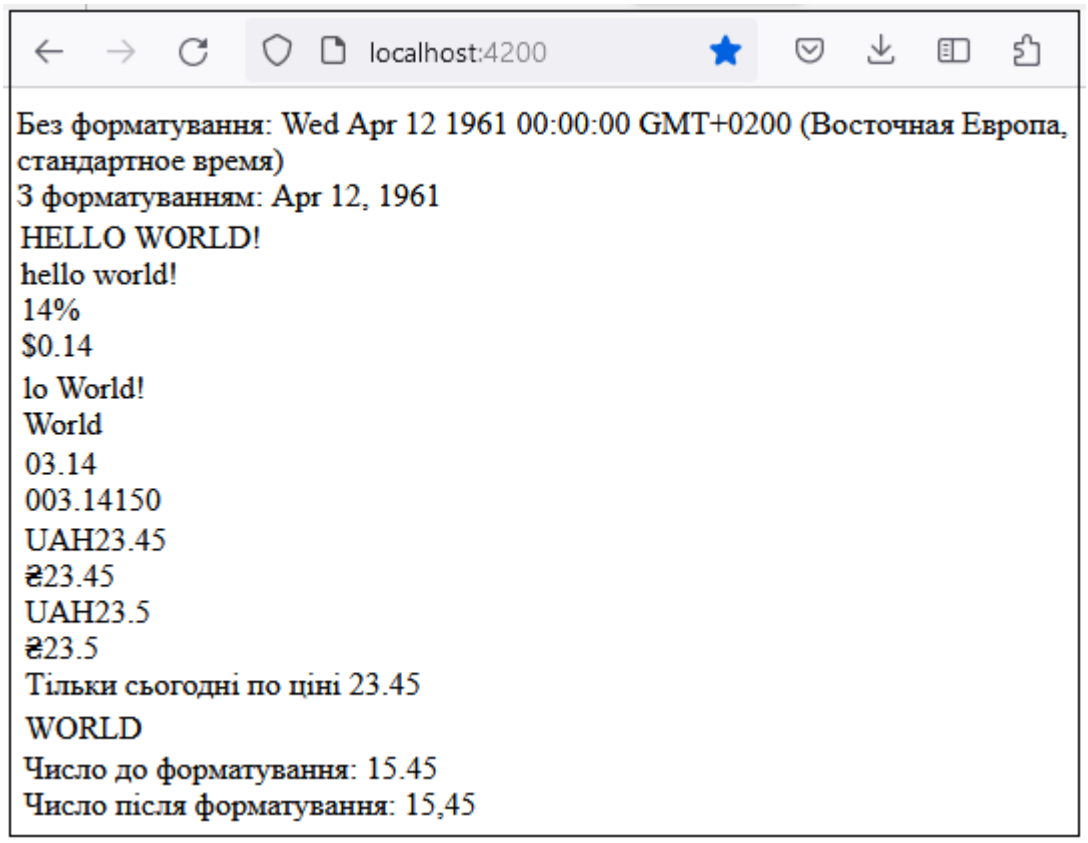
Застосуємо FormatPipe у коді компонента:

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

@Component({
    selector: 'my-app',
    template: `
            <div>Без форматування: {{myDate}}</div>
            <div>З форматуванням: {{myDate | date}}</div>
            <div>{{welcome | uppercase}}</div>
```

```
                <div>{{welcome | lowercase}}</div>
                <div>{{persentage | percent}}</div>
                <div>{{persentage | currency}}</div>
                <div>{{welcome | slice:3}}</div>
                <div>{{welcome | slice:6:11}}</div>
                <div>{{pi | number:'2.1-2'}}</div>
                <div>{{pi | number:'3.5-5'}}</div>
                <div>{{money | currency:'UAH':'code'}}</div>
                <div>{{money | currency:'UAH':'symbol-narrow'}}</div>
                <div>{{money | currency:'UAH':'symbol':'1.1-1'}}</div>
                <div>{{money | currency:'UAH':'symbol-narrow':'1.1-1'}}</div>
                <div>{{money | currency:'UAH':'Тільки сьогодні по ціні '}}</div>
                <div>{{message | slice:6:11 | uppercase}}</div>
        <div>Число до форматування: {{x}}<br>Число після форматування: {{x | format}}</div>`
        })
        export class AppComponent {
          myDate = new Date(1961, 3, 12);
          welcome: string = "Hello World!";
          persentage: number = 0.14;
          pi: number = 3.1415;
          money: number = 23.45;
          message = "Hello World!";
            x: number = 15.45;
        }
```

Але щоб задіяти FormatPipe, його треба додати до головного модуля AppModule:

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent }   from './app.component';
import { FormatPipe} from './format.pipe';

@NgModule({
   imports:      [ BrowserModule],
   declarations: [ AppComponent, FormatPipe],
   bootstrap:    [ AppComponent ]
})
export class AppModule { }
```



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:

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
   name: 'join'
})
export class JoinPipe implements PipeTransform {
  transform(array: any, start?: any, end?: any): any {
    let result = array;
    if(start!==undefined){
        if(end!==undefined){
            result = array.slice(start, end);
        }
        else{
            result = array.slice(start, result.length);
        }
    }
    return result.join('', '');
```

```
  }
}
```

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:

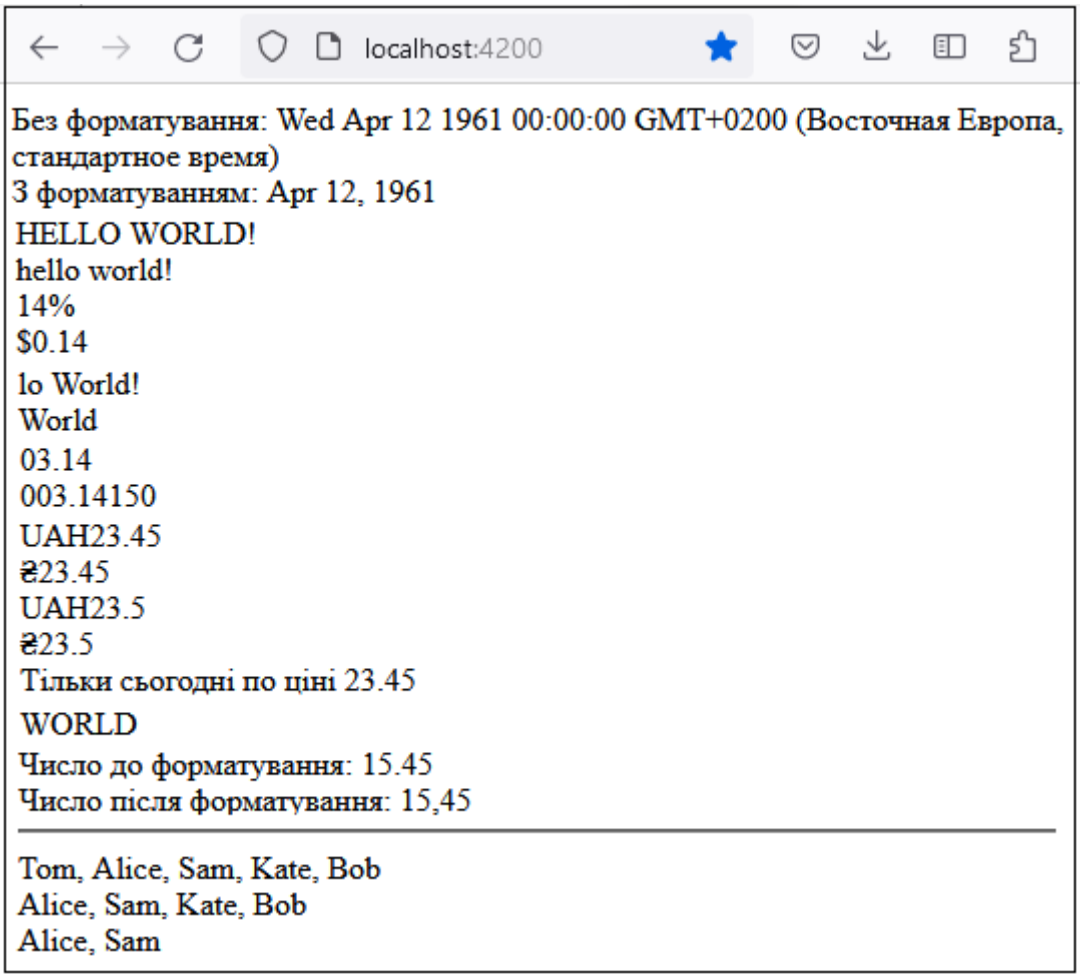
```
import { Component} from '@angular/core';
@Component({
   selector: 'my-app',
   template: `
         <div>Без форматування: {{myDate}}</div>
          <div>З форматуванням: {{myDate | date}}</div>
          <div>{{welcome | uppercase}}</div>
          <div>{{welcome | lowercase}}</div>
          <div>{{persentage | percent}}</div>
          <div>{{persentage | currency}}</div>
          <div>{{welcome | slice:3}}</div>
          <div>{{welcome | slice:6:11}}</div>
          <div>{{pi | number:'2.1-2'}}</div>
          <div>{{pi | number:'3.5-5'}}</div>
          <div>{{money | currency:'UAH':'code'}}</div>
          <div>{{money | currency:'UAH':'symbol-narrow'}}</div>
          <div>{{money | currency:'UAH':'symbol':'1.1-1'}}</div>
          <div>{{money | currency:'UAH':'symbol-narrow':'1.1-1'}}</div>
          <div>{{money | currency:'UAH':'Тільки сьогодні по ціні '}}</div>
          <div>{{message | slice:6:11 | uppercase}}</div>
   <div>Число до форматування: {{x}}<br>Число після форматування: {{x |
format}}</div>
         <hr/>

          <div>{{users | join}}</div>
          <div>{{users | join:1}}</div>
          <div>{{users | join:1:3}}</div>`
})
export class AppComponent {
  myDate = new Date(1961, 3, 12);
  welcome: string = "Hello World!";
  persentage: number = 0.14;
  pi: number = 3.1415;
  money: number = 23.45;
  message = "Hello World!";
```

```
  x: number = 15.45;
    users = ["Tom", "Alice", "Sam", "Kate", "Bob"];
}
```

Знову ж таки підключимо JoinPipe в модулі програми:

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent }   from './app.component';
import { FormatPipe} from './format.pipe';
import { JoinPipe} from './join.pipe';
@NgModule({
    imports:      [ BrowserModule],
    declarations: [ AppComponent, FormatPipe, JoinPipe ],
    bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Результат роботи:



Виконати самостійно: Розробити pipe, який приймає в якості аргументу число і повертає число після отримання квадратного кореня.

II) Для Angular-додатку Pipes2 виконати вправу 2;

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:

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
   name: 'format'
})
export class FormatPipe implements PipeTransform {
  transform(value: number, args?: any): string {

    return value.toString().replace(".", ",");
  }
}
```

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:

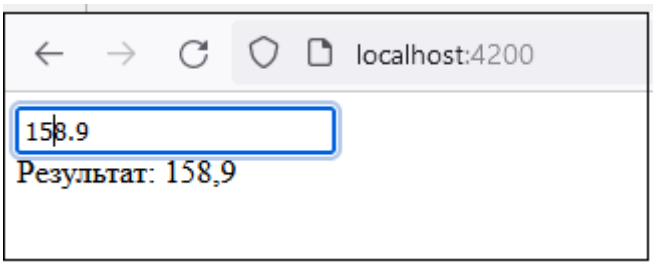
```
import { Component} from '@angular/core';

@Component({
   selector: 'my-app',
   template: `<input [(ngModel)]="num" name="fact">
```

```
    <div>Результат: {{num | format}}</div>`
})
export class AppComponent {

    num: number = 15.45;
}
```

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

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule }   from '@angular/forms';
import { AppComponent }   from './app.component';
import { FormatPipe} from './format.pipe';
import { JoinPipe } from './join.pipe';
@NgModule({
    imports:      [ BrowserModule, FormsModule ],
    declarations: [ AppComponent, FormatPipe, JoinPipe],
    bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

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



Але в минулій темі був також створений інший pipe:

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
    name: 'join'
})
export class JoinPipe implements PipeTransform {
  transform(array: any, start?: any, end?: any): any {
    return array.join(", ");
  }
}
```

Цей pipe 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:

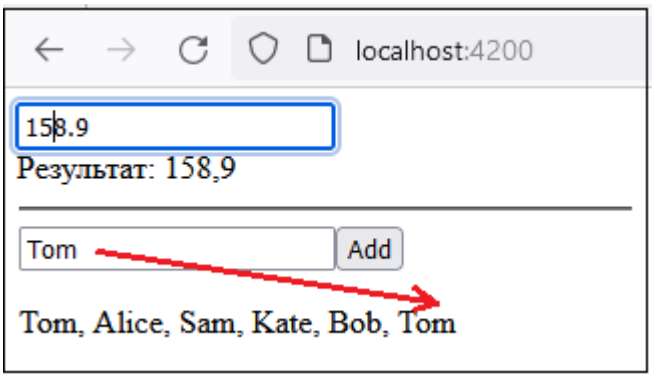
```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
    name: 'join',
    pure: false
})
export class JoinPipe implements PipeTransform {
  transform(array: any, start?: any, end?: any): any {
    return array.join(", ");
  }
}
```

За замовчуванням параметр pure дорівнює true.

Тепер ми можемо додавати в компонент нові елементи в цей масив:

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

@Component({
    selector: 'my-app',
    template: `
    <input [(ngModel)]="num" name="fact">
    <div>Результат: {{num | format}}</div>
    <hr/>
            <input #user name="user" class="form-control">
            <button class="btn" (click)="users.push(user.value)">Add</button>
            <p>{{users | join}}</p>`
})
export class AppComponent {
    num: number = 15.45;
    users = ["Tom", "Alice", "Sam", "Kate", "Bob"];
}
```

І до всіх доданих елементів також застосовуватиметься JoinPipe:



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.

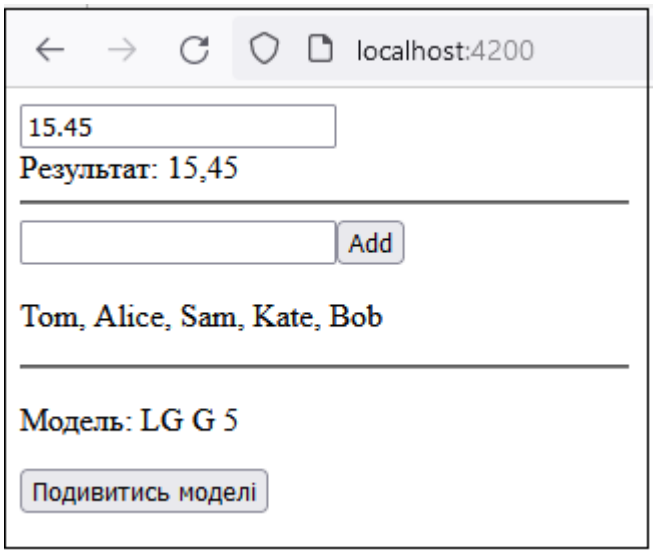
AsyncPipe відстежує об'єкти Observable та Promise та повертає отримане з цих об'єктів значення. При отриманні значення AsyncPipe сигналізує компонент про те, що треба перевірити зміни. Якщо компонент знищується, AsyncPipe автоматично відписується від об'єктів Observable і Promise, що унеможливлює можливі витоки пам'яті.

Використовуємо AsyncPipe:

```
import { Component} from '@angular/core';
import { Observable, interval } from 'rxjs';
import { map } from 'rxjs/operators';

@Component({
   selector: 'my-app',
   template: `
<input [(ngModel)]="num" name="fact">
   <div>Результат: {{num | format}}</div>
  <hr/>
   <input #user name="user" class="form-control">
   <button class="btn" (click)="users.push(user.value)">Add</button>
   <p>{{users | join}}</p>
          <p>Модель: {{ phone| async }}</p>
          <button (click)="showPhones()">Посмотреть модели</button>`
})
export class AppComponent {
  num: number = 15.45;
  users = ["Tom", "Alice", "Sam", "Kate", "Bob"];
   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]));
    }
}
```

Тут з періодичністю 500 мілісекунд у шаблон компонента передається черговий елемент з масиву phones.

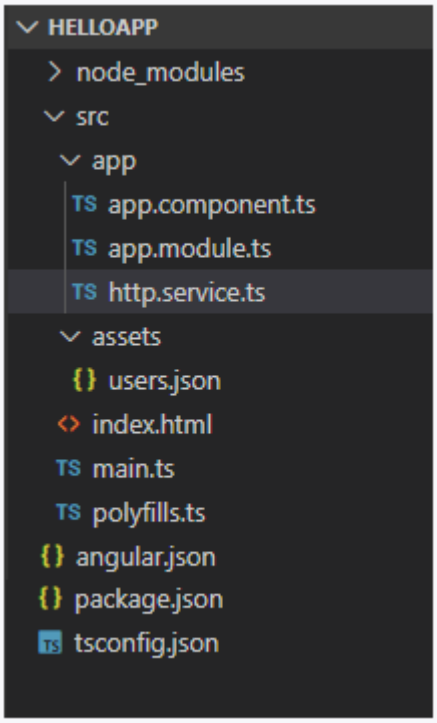


Компонент не повинен підписуватись на асинхронне отримання даних, обробляти їх, а при знищенні відписуватись від отримання даних. Всю цю роботу робить AsyncPipe.

III) Для Angular-додатку Pipes3 виконати вправу 3.

Вправа 3:

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



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

```
import {Injectable} from '@angular/core';
import {HttpClient} from '@angular/common/http';
@Injectable()
export class HttpService{
    constructor(private http: HttpClient){ }
    getUsers(){
        return this.http.get('assets/users.json');
    }
}
```

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

```
[{
    "name": "Bob",
    "age": 28
},{
    "name": "Tom",
    "age": 45
},{
    "name": "Alice",
    "age": 32
}]
```

У файлі app.component.ts використовує сервіс:

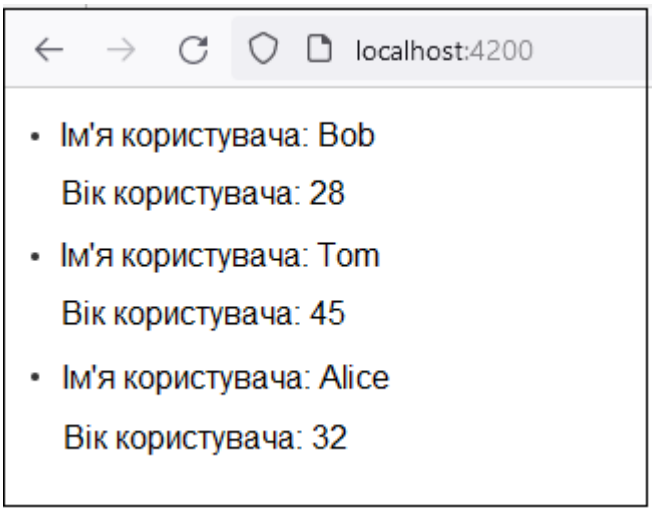
```
import { Component, OnInit} from '@angular/core';
```

```
import { HttpService} from './http.service';
import {Observable} from 'rxjs';

@Component({
   selector: 'my-app',
   template: `<ul>
                <li *ngFor="let user of users | async">
                <p>Ім’я користувача: {{user.name}}</p>
                <p>Вік користувача: {{user.age}}</p>
                </li>
             </ul>`,
   providers: [HttpService]
})
export class AppComponent implements OnInit {

   users: Observable<Object>|undefined;;
   constructor(private httpService: HttpService){}
   ngOnInit(){

       this.users = this.httpService.getUsers();
   }
}
```

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:



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

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent }   from './app.component';

import { HttpClientModule }   from '@angular/common/http';
@NgModule({
   imports:      [ BrowserModule, HttpClientModule ],
   declarations: [ AppComponent],
   bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

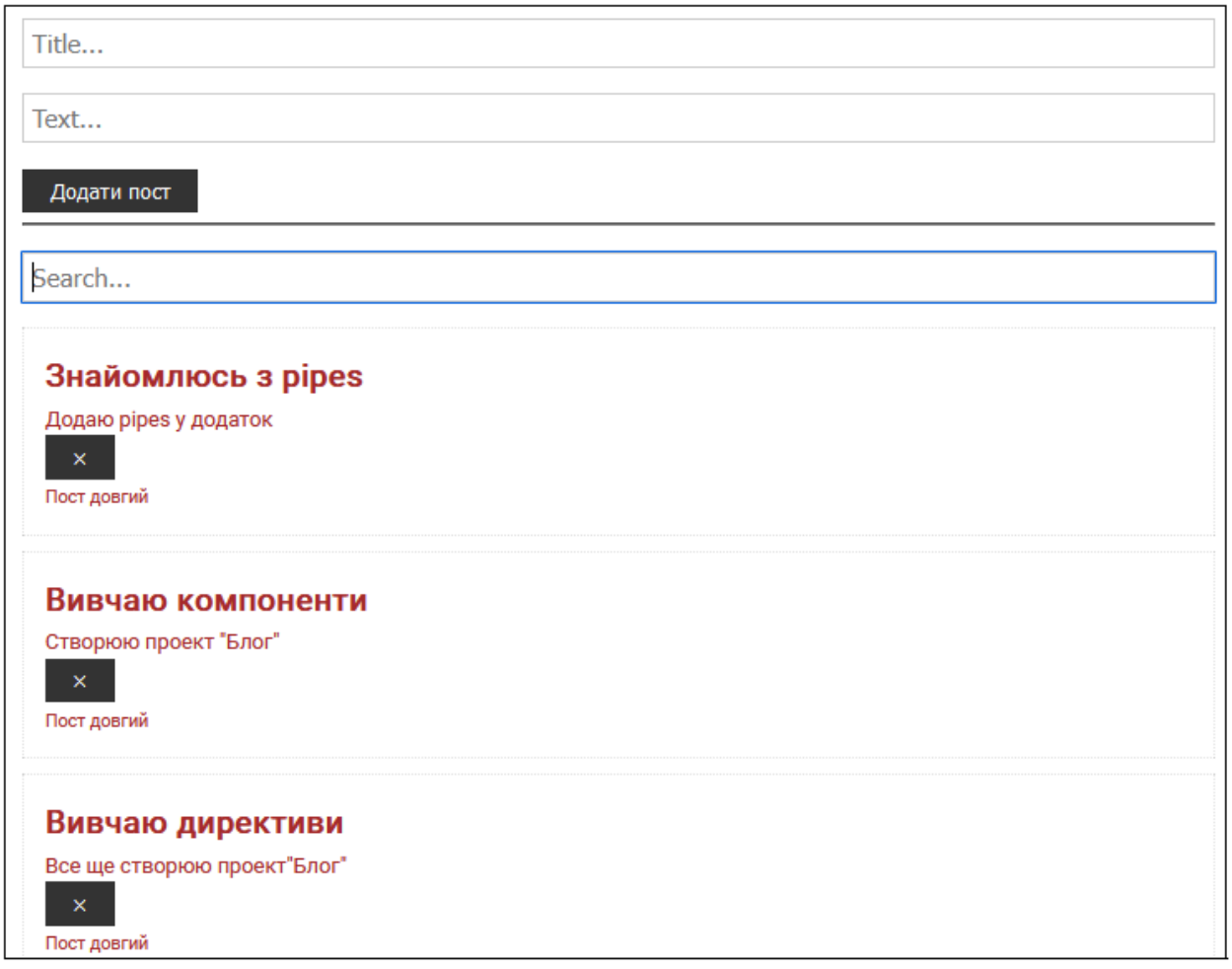
VI) Самостійно виконати вправу 4

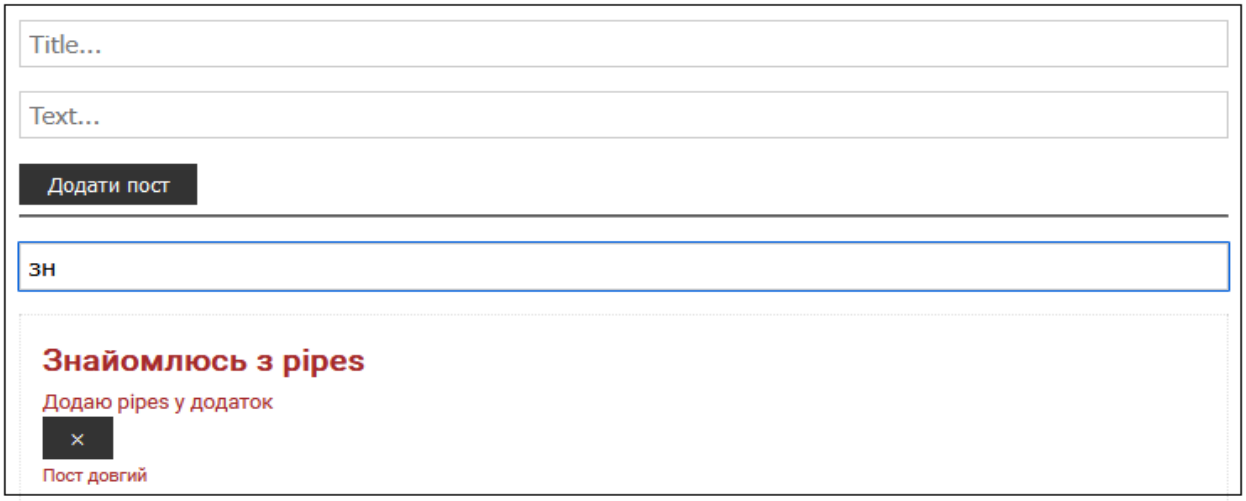
Вправа 4: Виконати самостійно

1) У додаток Blog додати поле:

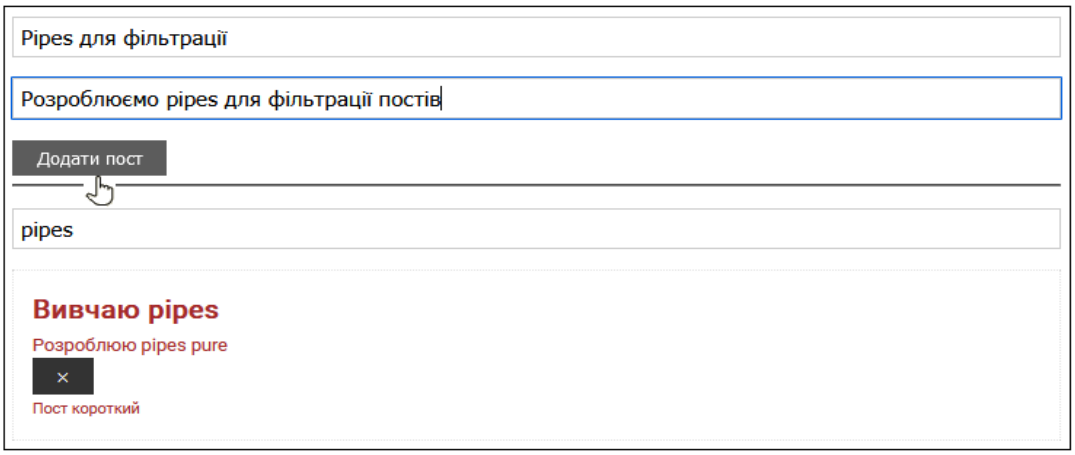
```
<input type="text" [(ngModel)]="search" placeholder="Search...">
```

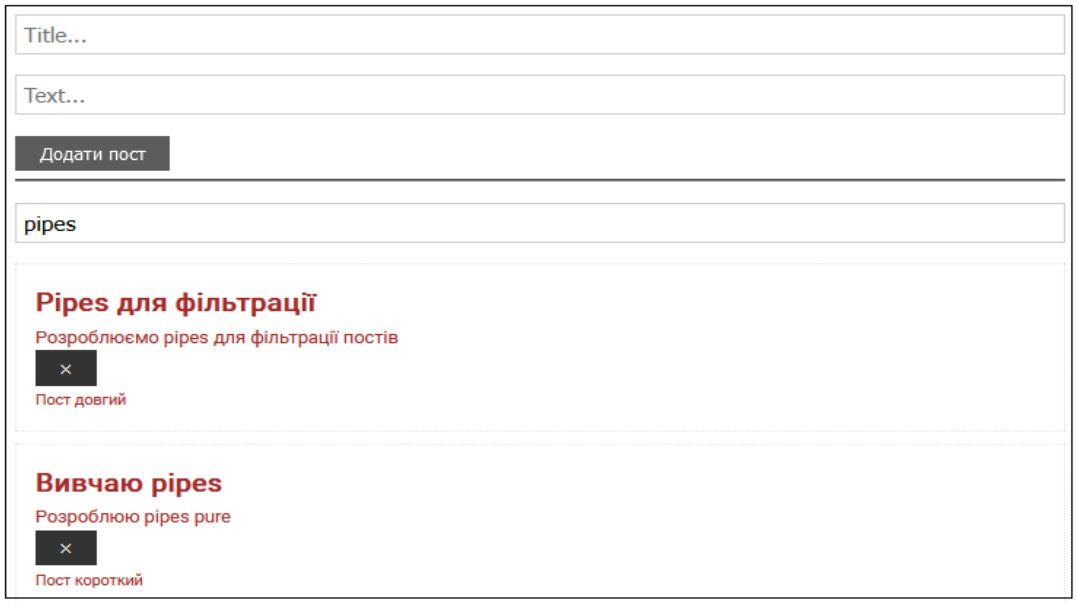
для фільтрації постів по полю Title. У поле Search користувач вводить дані поста (Title) і в компоненті виводяться тільки ті пости, які включають шуканий рядок без врахування регістра.





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:

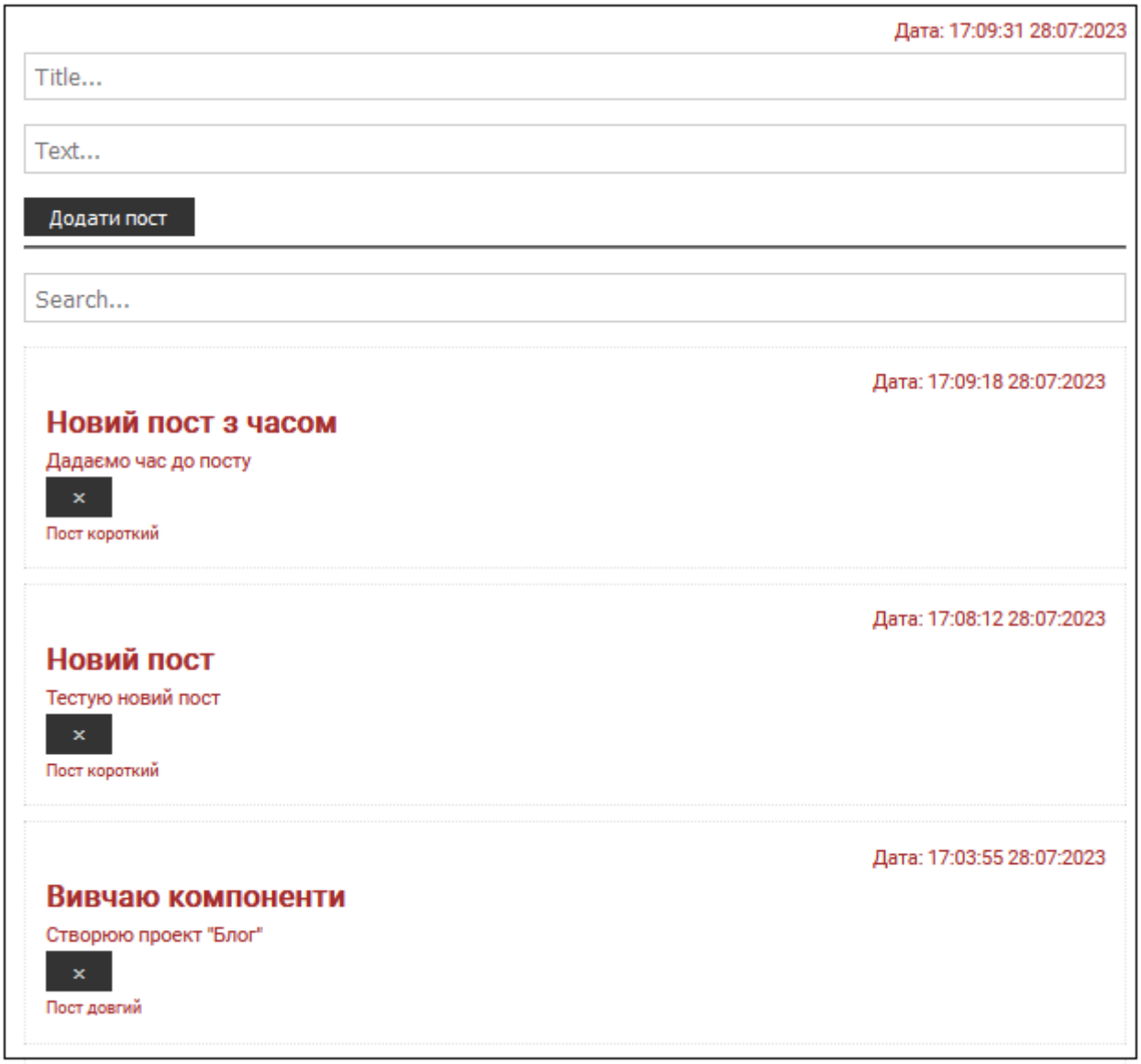




Метод transform з двома параметрами можно оформити наступним чином:

```
transform(posts: Post[], search:string=''): Post[] {
    if (!search.trim()){
   return posts;
    }
    return posts.filter(post=>
      {return post.title.toLowerCase().includes(search.toLowerCase())})
  }
```

2) На сторінці при допомозі pipes виводити 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:

```
<p style="text-align:right">Дата: {{myDate$ | async | date:'HH:mm:ss dd:MM:yyyy'}}</p>
```

Задавши при цьому параметр myDate$ наступним чином:

```
myDate$:Observable<Date>=new Observable(obs=>
   {setInterval(()=>{
     obs.next(new Date())
 },1000)})
```

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

```
<p style="text-align:right">Дата: {{myPost.date | date:'HH:mm:ss dd:MM:yyyy'}}</p>
```

Задавши при цьому параметр myPost.date наступним чином:

```
date1!:Date
  ngOnInit():void {
this.myDate$.subscribe(date=>{
  this.date1=date
})
  }

  addPost(){
if (this.title.trim()&&this.text.trim()){
  const post: Post={
    title:this.title,
    text:this.text,
    date:this.date1
  }
  this.onAdd.emit(post);
  this.title=this.text='';
}
  }
```

VI) 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:

a) Pipes: призначення та використання;
b) Ланцюжки pipes;
c) Створення своїх pipes;
d) Передача параметрів у pipes;
e) Pure та Impure Pipes;
f) AsyncPipe.

VII) Angular-додатки Pipes1 та Blog розгорнути на платформі Firebase у проектах з ім’ям «ПрізвищеГрупаLaba6-1» та «ПрізвищеГрупаLaba6-4», наприклад «KovalenkoIP01Laba6-1» та «KovalenkoIP01Laba6-4».

ДОДАТОК

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