МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Лабораторна робота №6

з дисципліни «Реактивне програмування»

на тему:

«Pipes. Створення та робота з pipes»

Перевірила

доцент

Полупан Ю. В.

Виконав

студент ФІОТ

групи ІС-01

Адамов Денис

Київ 2023

Зміст

Лабораторне заняття №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».

Pipes: призначення та використання

Pipes представляють спеціальні інструменти, які дозволяють форматувати значення, що відображаються.

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").

```
welcome: string = 'Hello World!';
persentage: number = 0.14;
pi: number = 3.1415;
```

```
<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>
```

HELLO WORLD!
hello world!
14%
$0.14
lo World!
World
03.14
003.14150

Ланцюжки pipes

Ми можемо складати ланцюжки виразів, розділені вертикальною рискою для застосування відразу кількох pipes до одного значення.

```
message = 'Hello World!';
```

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

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.

lab6 > 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      return value.toString().replace('.', ',');
9    }
10 }
```

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

Число до форматування: 15.45
Число після форматування: 15,45

Передача параметрів у pipes

У шаблоні параметри у pipes передаються через двокрапку.

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

lab6 > pipes1 > src > app > TS join.pipe.ts > ...

```
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(', ');
  }
}
```

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

Tom, Alice, Sam, Kate, Bob
Alice, Sam, Kate, Bob
Alice, Sam

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.

Для створення impure pipe необхідно додати до декоратора Pipe параметр pure: false.

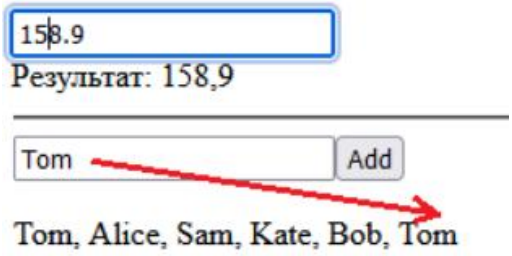
lab6 > pipes2 > src > app > TS join.pipe.ts > JoinPipe

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

@Pipe({
  name: "join",
  pure: false
})
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(", ");
  }
}
```

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

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



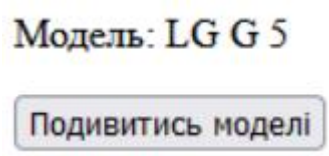
AsyncPipe

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

```
<p>Модель: {{ phone | async }}</p>
```

```
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.