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 No 6

Pipes. Створення та робота з pipes.

З дисципліни: «Реактивне програмування»

Студент: Трофимов Данило Олегович

Група: ІП-02

Дата захисту роботи:

Викладач: доц. Полупан Юлія Вікторівна

Захищено з оцінкою:

Київ, 2023

ЗМІСТ

ЗМІСТ 2

ЗМІСТ 2 ..... **ПОМИЛКА! ЗАКЛАДКУ НЕ ВИЗНАЧЕНО.**

ЗАГАЛЬНИЙ ХІД РОБОТИ: 4

1) PIPES: ПРИЗНАЧЕННЯ ТА ВИКОРИСТАННЯ 4

2) ЛАНЦЮЖКИ PIPES 6

3) СТВОРЕННЯ СВОЇХ PIPES 7

1. Створити клас для pipe: 8

2. Зареєструвати свій pipe: 8

3. Використання свого pipe в шаблоні: 9

4. Передавання параметрів в pipe (опціонально): 9

Приклад: 9

4) ПЕРЕДАЧА ПАРАМЕТРІВ У PIPES 10

Передача одного параметра: 10

Передача більше одного параметра: 10

Використання параметрів у pipe: 11

Передача динамічних параметрів: 12

Використання ...args в Angular Pipes: 12

Використання у шаблоні: 13

5) PURE TA IMPURE PIPES 13

Pure Pipes: 13

Impure Pipes: 14

Використання у шаблоні: 15

6) ASYNCPIPE 16

Приклад 1: Async Pipe з interval 16

Приклад 2: Async Pipe з HTTP запитом 17

HttpService: 18

7) ПОСИЛАННЯ НА РОЗГОРНУТІ ДОДАТКИ 18

ВИСНОВКИ 19

СПИСОК ЛІТЕРАТУРИ 20

Загальний хід роботи:

1. Згідно з інструкціями ініціалізовано angular проєкти Pipe1-3, скопійовано проєкт Blog;
2. Було опрацьовано завдання згідно з методичними вказівками;
3. Було розгорнуто додаток Blog.

1) Pipes: призначення та використання

У Angular pipes використовуються для перетворення даних перед їх відображенням в шаблоні. Pipes можуть бути вбудованими (наприклад, DatePipe для роботи з датами) або користувацькими, які ви самі створюєте, як у вас у коді.

Основним призначенням pipes є обробка та форматування даних перед їх відображенням в шаблоні. Вони приймають вхідні дані, обробляють їх і повертають нове значення. При цьому вони не змінюють оригінальні дані, а лише повертають трансформоване значення.

Наш код містить у першому завданні три користувацькі pipes: FormatPipe, JoinPipe та SqrtPipe. Давайте розглянемо кожен з них:

FormatPipe:

```
@Pipe({
  name: 'format'
})
export class FormatPipe implements PipeTransform {
  transform(value: number, args?: any): string {
    return value.toString().replace(".", ",");
  }
}
```

Цей pipe перетворює числове значення в рядок і замінює крапку комою. Ми можемо викликати його у шаблоні, передаючи числове значення через | format.

JoinPipe:

```
@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(", ");
  }
}
```

Цей pipe об'єднує елементи масиву в рядок, розділені комами. Ми иожемо викликати його у шаблоні, передаючи масив через | join:start:end.

SqrtPipe:

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

```
export class SqrtPipe implements PipeTransform {
  transform(value: number, args?: any): number {
    return Math.sqrt(value);
  }
}
```

Цей pipe використовується для обчислення квадратного кореня числа. Ви можете викликати його у шаблоні, передаючи числове значення через | sqrt.

Щоб використовувати ці pipes у шаблоні, нам потрібно додати їх до декларації модуля (або використовувати їх глобально). Після цього ми можемо використовувати їх у шаблонах компонентів, наприклад:

```
<!-- Використання FormatPipe -->
{{ someNumber | format }}
<!-- Використання JoinPipe -->
{{ someArray | join:1:3 }}
<!-- Використання SqrtPipe -->
{{ someNumber | sqrt }}
```

Це дозволяє нам зручно трансформувати та форматувати дані, не забруднюючи код компонента.

2) Ланцюжки pipes

В Angular ви можете об'єднувати кілька pipes в ланцюжок (pipe chain) для трансформації даних в шаблоні. Ланцюжок pipe створюється за допомогою символу каналу (|), який вказує, що ви хочете застосувати декілька pipes один за одним. Кожен pipe в ланцюжку обробляє вхідні дані та передає трансформоване значення наступному pipe.

Ось приклад ланцюжка pipe у шаблоні:

{{ someValue | firstPipe | secondPipe | thirdPipe }}

У цьому прикладі:

- someValue - вхідні дані, які будуть оброблені ланцюжком pipe.
- firstPipe - перший pipe в ланцюжку, який обробляє someValue і повертає трансформоване значення.
- secondPipe - другий pipe в ланцюжку, який обробляє трансформоване значення від firstPipe.
- thirdPipe - третій pipe в ланцюжку, який обробляє трансформоване значення від secondPipe.

Це дозволяє вам послідовно застосовувати різні pipes для отримання необхідного результату. Важливо враховувати порядок pipes, оскільки вони застосовуються в тому порядку, в якому вони вказані у ланцюжку.

Приклад ланцюжка pipe з pipes:

{{ someNumber | sqrt | format }}

У цьому випадку someNumber спочатку обробляється pipe sqrt, а потім результат передається в pipe format. Перший pipe (sqrt) обчислює квадратний корінь цього рядка-числа?, а другий (format) трансформує число в рядок заміною крапки комою.

3) Створення своїх pipes

Створення власних pipes у Angular дозволяє вам розширити можливості трансформації даних в шаблонах. Давайте розглянемо кроки створення власних pipes:

1. Створити клас для pipe:

Треба створити клас, який реалізує інтерфейс PipeTransform. Цей інтерфейс вимагає наявності методу transform, який приймає вхідні дані та параметри і повертає трансформоване значення.

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

@Pipe({
  name: 'custom'
})
export class CustomPipe implements PipeTransform {
  transform(value: any, args?: any): any {
    // Логіка трансформації значення
    return transformedValue;
  }
}
```

2. Зареєструвати свій pipe:

Необхідно додати свій pipe до декларації(declarations) модуля або використовувати глобально в AppModule, якщо ви хочете використовувати його в усьому додатку.

```
import { NgModule } from '@angular/core';
import { CustomPipe } from './path-to-your-pipe/custom.pipe';

@NgModule({
  declarations: [
    // інші компоненти, директиви, pipes
    CustomPipe
  ],
  // інші конфігураційні параметри модуля
})
```

export class YourModule { }

3. Використання свого pipe в шаблоні:

Використовуйте ім'я створеного pipe в шаблоні разом з оператором каналу (|).

{{ someValue | custom:arg1:arg2 }}

4. Передавання параметрів в pipe (опціонально):

Якщо нашому pipe потрібні параметри, ми можемо передавати їх через двокрапку.

```
//параметри в методі transform
transform(value: any, arg1: any, arg2: any): any {
  // Логіка трансформації з використанням параметрів
  return transformedValue;
}
```

І використовуйте їх у шаблоні:

{{ someValue | custom:param1:param2 }}

Приклад:

Нехай у нас є pipe для конвертації часу з секунд у години, хвилини і секунди:

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

@Pipe({
  name: 'timeFormat'
})
export class TimeFormatPipe implements PipeTransform {
  transform(seconds: number): string {
    const hours = Math.floor(seconds / 3600);
    const minutes = Math.floor((seconds % 3600) / 60);
    const remainingSeconds = seconds % 60;
```

```
    return `${hours}h ${minutes}m ${remainingSeconds}s`;
  }
}
```

В шаблоні компонента ми можемо використовувати цей pipe:

```
{{ totalTimeInSeconds | timeFormat }}
```

4) Передача параметрів у pipes

При передачі параметрів у Angular pipe, ми можемо використовувати додаткові аргументи в шаблоні. Параметри дозволяють вам змінювати поведінку pipe в залежності від конкретних потреб. Ось кілька прикладів передачі параметрів у pipe:

Передача одного параметра:

```
// pipe з одним параметром
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'custom'
})
export class CustomPipe implements PipeTransform {
  transform(value: any, multiplier: number): any {
    return value * multiplier;
  }
}
```

В шаблоні ми передаємо параметр через двокрапку:

```
{{ someValue | custom:2 }}
```

Передача більше одного параметра:

```
// pipe з більше ніж одним параметром
```

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

@Pipe({
  name: 'custom'
})
export class CustomPipe implements PipeTransform {
  transform(value: any, arg1: any, arg2: any): any {
    // Логіка трансформації з використанням параметрів
    return transformedValue;
  }
}
```

В шаблоні ви передаєте більше параметрів через двокрапку:

```
{{ someValue | custom:param1:param2 }}
```

Використання параметрів у pipe:

У тілі методу transform ми можемо використовувати передані параметри для зміни логіки трансформації. Наприклад:

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

@Pipe({
  name: 'append'
})
export class AppendPipe implements PipeTransform {
  transform(value: string, appendString: string): string {
    return value + appendString;
  }
}
```

В шаблоні:

```
{{ "Hello" | append:" World" }}
```

У цьому прикладі, pipe append додає рядок " World" до рядка "Hello", і результатом буде "Hello World".

Передача динамічних параметрів:

Ми також можемо передавати параметри, які змінюються динамічно, наприклад, використовуючи значення змінної з компонента:

{{ someValue | custom:dynamicParam }}

де dynamicParam - це змінна, яку ви визначаєте в компоненті.

В TypeScript та JavaScript ...args представляє рест-оператор, який дозволяє функції приймати змінну кількість аргументів у вигляді масиву. У випадку Angular pipes, вони також можуть використовувати ...args для обробки змінної кількості аргументів.

Використання ...args в Angular Pipes:

При створенні Angular pipe ви можете використовувати ...args для отримання доступу до змінної кількості аргументів, переданих у шаблоні.

Наприклад, розглянемо pipe, який додає до рядка додаткові значення (аргументи):

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


@Pipe({
  name: 'appendValues'
})
export class AppendValuesPipe implements PipeTransform {
  transform(value: string, ...args: string[]): string {
    // Додаємо всі аргументи до вихідного значення
    return value + args.join(', ');
  }
```

}

У цьому прикладі, ...args: string[] означає, що в args буде зберігатися масив усіх аргументів, переданих у шаблоні. Ми потім можемо долучити ці значення до вихідного рядка.

Використання у шаблоні:
{{ "Hello" | appendValues:" World":" Angular":"Labs" }}

У цьому випадку pipe appendValues отримує "Hello" як основне значення та додає до нього всі подальші аргументи, розділені комами. Результат буде "Hello World, Angular, Labs".

Користуючись ...args, ви отримуєте гнучкість обробки будь-якої кількості аргументів у нашому pipe. Це особливо корисно, коли вам потрібно передати різні кількості параметрів для різних випадків використання.

Ці приклади демонструють, як ви можете ефективно використовувати параметри для налаштування роботи нашого pipe в шаблоні Angular

5) Pure та Impure Pipes

У Angular розрізняють два типи pipes: "pure" (чистий) та "impure" (не чистий). Це визначає, коли Angular викликає метод transform нашого pipe для трансформації даних. Давайте розглянемо це на прикладі нашого pipe.

Pure Pipes:

Чисті pipes викликаються тільки тоді, коли Angular визначає, що вхідні дані, передані в pipe, були змінені. Це означає, що якщо вхідні дані не змінилися, Angular використовуватиме збережене трансформоване значення і не викликатиме метод transform.

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

```
@Pipe({
  name: 'pureJoin',
  pure: true // or empty value
})
export class PureJoinPipe 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 цьому випадку pure: true позначає, що цей pipe є чистим. Це означає, що Angular викличе метод transform тільки тоді, коли значення array, start або end зміниться.

Impure Pipes:

Не чисті pipes викликаються при кожному циклі зміни детектора змін (change detection cycle), незалежно від того, чи відбулися зміни у вхідних даних чи ні.

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

@Pipe({
  name: 'impureJoin',
  pure: false
```

```
})
export class ImpureJoinPipe 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 цьому випадку pure: false позначає, що цей pipe є не чистим. Це означає, що Angular викликає метод transform кожен раз під час циклу зміни детектора, незалежно від того, чи відбулися зміни в вхідних даних.

Використання у шаблоні:

```
<!-- Використання чистого pipe -->
{{ someArray | pureJoin:1:3 }}

<!-- Використання нечистого pipe -->
{{ someArray | impureJoin:1:3 }}
```

При використанні чистого pipe, Angular буде викликати transform тільки при зміні someArray. У випадку нечистого pipe, Angular буде викликати transform при кожному циклі зміни детектора, незалежно від того, чи змінилися вхідні дані

6) AsyncPipe

Async pipes у Angular використовуються для спрощення відображення асинхронних даних в шаблонах. Основна перевага використання async pipes полягає в тому, що вони автоматично викликають підписку та відписку від Observable, забезпечуючи відображення оновлень даних у шаблоні.

Приклад 1: Async Pipe з interval

У нашому прикладі ми використовуємо interval(500) для емуляції потоку даних через кожні 500 мілісекунд:

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

@Component({
  selector: 'app-root',
  template: `
    <p>Модель: {{ phone | async }}</p>
  `,
})
export class AppComponent implements OnInit {
  phones = ["iPhone 7", "LG G 5", "Honor 9", "Idol S4", "Nexus 6P"];
  phone: Observable<string> | undefined;

  ngOnInit() {
    this.showPhones();
  }

  showPhones() {
    this.phone = interval(500).pipe(map((i: number) => this.phones[i]));
  }
```

}

У цьому випадку async pipe автоматично відбувається підписку на Observable, створене за допомогою interval(500), та відображає оновлені дані кожні 500 мілісекунд.

Приклад 2: Async Pipe з HTTP запитом

У другому прикладі використовується HTTP сервіс для отримання даних з файлу JSON:

```
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs';
import { HttpService } from './http.service';

@Component({
  selector: 'app-root',
  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<any> | undefined;

  constructor(private httpService: HttpService) { }

  ngOnInit() {
```

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

У цьому випадку async pipe автоматично відбувається підписку на Observable, який повертає this.httpService.getUsers(). Якщо дані отримані з сервера змінюються, async pipe автоматично відображає оновлені дані у шаблоні.

HttpService:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable()
export class HttpService {
  constructor(private http: HttpClient) { }

  getUsers(): Observable<any> {
    return this.http.get('assets/users.json');
  }
}
```

У цьому прикладі, метод getUsers сервісу повертає Observable, який представляє дані, отримані з файлу JSON.

7) Посилання на розгорнуті додатки

Додаток розміщений за адресою: https://trofymovip02laba6.web.app/

Висновки

Pipes у Angular є потужним інструментом для трансформації та форматування даних перед їх відображенням у шаблонах. Вони дозволяють вам легко маніпулювати та відображати дані відповідно до наших потреб. Pipes надають зручний спосіб виконання різних операцій трансформації даних, таких як форматування чисел, рядків, дат, обробка масивів і т.д., без забруднення коду компонента. Ми можемо створювати свої власні pipes для вирішення конкретних завдань або реімплементації існуючих pipes для наших потреб. Pipes, такі як async, полегшують відображення асинхронних даних, таких як результати HTTP запитів, автоматично керуючи підпискою та відпискою. Ми можемо об'єднувати кілька pipes в ланцюжок для послідовної обробки даних, спрощуючи код та роблячи його більш читабельним. Загалом, pipes дозволяють створювати динамічні та декларативні шаблони для відображення даних у відповідності до логіки та вимог нашого додатка. Вони забезпечують гнучкість та повторне використання коду, що робить їх важливою частиною інструментарію Angular.

Список літератури

1. Introduction to the Angular docs [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/docs.
2. Angular. Angular tutorials [Електронний ресурс] / Angular – Режим доступу до ресурсу: https://angular.io/tutorial.
3. The Angular Book [Електронний ресурс] – Режим доступу до ресурсу: https://angular-book.dev/.
4. Daniel Schmitz. Angular 14 from Scratch / Daniel Schmitz., 2022.
5. Denys Vuika. Developing with Angular / Denys Vuika., 2018.