

**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.»
роботи з дисципліни: «Реактивне програмування»

Студент: Мешков Андрій Ігорович

Група: ІІІ-15

Дата захисту роботи: 14 «грудня» 2024

Викладач: доц. Полупан Юлія Вікторівна

Захищено з оцінкою: _____

Київ, 2024

ЗМІСТ

PIPES: ПРИЗНАЧЕННЯ ТА ВИКОРИСТАННЯ	3
ЛАНЦЮЖКИ PIPES	8
СТВОРЕННЯ СВОЇХ PIPES	9
ПЕРЕДАЧА ПАРАМЕТРІВ У PIPES	11
PURE ТА IMPURE PIPES	14
ASYNCPipe	17
ДЕТАЛЬНИЙ ОГЛЯД КОМПОНЕНТУ POST ДОДАТКУ BLOG	20
ДЕТАЛЬНИЙ ОГЛЯД КОМПОНЕНТУ POST-FORM ДОДАТКУ BLOG.	22
ВИСНОВКИ.....	25
СПИСОК ДЖЕРЕЛ	27

Pipes: призначення та використання

Pipes в Angular — це потужний інструмент, який дозволяє формувати дані безпосередньо у шаблонах компонентів. Вони є функціями перетворення, які обробляють вхідні дані й повертають результат у потрібному форматі.

Призначення Pipes

Форматування даних у шаблоні: Pipes дозволяють змінювати відображення даних у HTML без зміни самої бізнес-логіки.

Зручна робота з текстом, датами та числами: Можна легко формувати рядки, числа, валюти, дати та інші типи даних.

Оптимізація коду: Логіка форматування виноситься в pipe, що зменшує навантаження на шаблон і полегшує підтримку коду.

Можливість створення кастомних Pipes: Angular дозволяє розробнику створювати власні Pipes для спеціалізованих завдань.

Використання Pipes

Синтаксис використання

Щоб застосувати pipe у шаблоні, використовується оператор | (pipe):

```
{{ value | pipeName }}
```

- value — значення, яке потрібно відформувати.
- pipeName — ім'я pipe.

Приклад:

```
<p>{{ user.name | uppercase }}</p> <!-- Виведе ім'я у верхньому регістрі -->
<p>{{ 1000 | currency:'USD':'symbol':'1.2-2' }}</p> <!-- Форматування
валюти -->
<p>{{ today | date:'dd/MM/yyyy' }}</p> <!-- Форматування дати -->
```

Вбудовані Pipes в Angular

- uppercase / lowercase - Зміна регістру тексту.

```
<p>{{ 'hello' | uppercase }}</p> <!-- HELLO -->
```

- Date - Форматування дати та часу.

```
<p>{{ today | date:'yyyy-MM-dd' }}</p> <!--  
2024-06-10 -->
```

- Currency - Форматування чисел як валюти.

```
<p>{{ 2500 | currency:'USD':'symbol' }}</p> <!--  
$2,500.00 -->
```

- Percent - Перетворює число у відсоток.

```
<p>{{ 0.25 | percent }}</p> <!-- 25% -->
```

- Decimal - Форматує число у десятковому форматі.

```
<p>{{ 3.14159 | number:'1.2-2' }}</p> <!-- 3.14  
-->
```

- Json - Виводить об'єкт як JSON-рядок.

```
<pre>{{ user | json }}</pre>
```

- Slice - Обрізає масив або рядок.

```
<p>{{ 'Hello World' | slice:0:5 }}</p> <!--  
Hello -->
```

Angular-додаток Pipes1. Вправа 1: Робота з pipes

Створемо додаток Pipes1.

Зпершу виведемо 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(2004, 5, 21);
}
```

Без форматування: Fri May 21 2004 00:00:00 GMT+0200 (за центральноєвропейським літнім часом)
З форматуванням: May 21, 2004

Рис.3.1 Результат

Вбудовані pipes. Параметри в pipes

Використаємо різні pipes. Використаємо з деякі з параметрами.

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

@Component({
  selector: 'my-app',
  template: `<div>Date Без форматування: {{myDate}}</div>
    <div>Date З форматуванням: {{myDate | date}}</div>
    <div>Uppercase – {{welcome | uppercase}}</div>
    <div>Lowercase – {{welcome | lowercase}}</div>
    <div>Slice:0:5 – {{welcome | slice:0:5}}</div>
    <div>Number – {{percentage}}</div>
    <div>Percent – {{percentage | percent}}</div>
    <div>Currency – {{percentage | currency:'EUR':'symbol'}}</div>
    <div>Number:'1.2-2' – {{percentage | number:'1.2-2' }}</div>
    `
})
export class AppComponent {
```

```

myDate = new Date(2004, 4, 21);
welcome: string = "Hello World!";
percentage: number = 0.1514;
}

```

Date Без форматування: Fri May 21 2004 00:00:00 GMT+0200 (за центральноєвропейським літнім часом)
 Date З форматуванням: May 21, 2004
 Uppercase - HELLO WORLD!
 Lowercase - hello world!
 Slice:0:5 - Hello
 Number - 0.1514
 Percent - 15%
 Currency - €0.15
 Number:'1.2-2' - 0.15

Рис.3.2 Результат

Форматування дат

Використаємо форматування дати.

```

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

@Component({
  selector: 'my-app',
  template: `<div>Date Без форматування: {{myDate}}</div>
    <div>Date З форматуванням: {{myDate | date}}</div>
    <div>Uppercase - {{welcome | uppercase}}</div>
    <div>Lowercase - {{welcome | lowercase}}</div>
    <div>Slice:0:5 - {{welcome | slice:0:5}}</div>
    <div>Number - {{percentage}}</div>
    <div>Percent - {{percentage | percent}}</div>
    <div>Currency - {{percentage | currency:'EUR':'symbol'}}</div>
    <div>Number:'1.2-2' - {{percentage | number:'1.2-2' }}</div>
    <div>Date - форматування - {{myDate | date:'dd/MM/yyyy'}}</div>
  `
})
export class AppComponent {
  myDate = new Date(2004, 4, 21);
  welcome: string = "Hello World!";
  percentage: number = 0.1514;
}

```

Число: 1.2-2 - 0.15

Date - форматування - 21/05/2004

Рис.3.3 Результат

Форматування валюти

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

@Component({
  selector: 'my-app',
  template: `<div>Date Без форматування: {{myDate}}</div>
    <div>Date 3 форматуванням: {{myDate | date}}</div>
    <div>Uppercase - {{welcome | uppercase}}</div>
    <div>Lowercase - {{welcome | lowercase}}</div>
    <div>Slice:0:5 - {{welcome | slice:0:5}}</div>
    <div>Number - {{percentage}}</div>
    <div>Percent - {{percentage | percent}}</div>
    <div>Currency - {{percentage | currency:'EUR':'symbol'}}</div>
    <div>Number:'1.2-2' - {{percentage | number:'1.2-2' }}</div>
    <div>Date - форматування - {{myDate | date:'dd/MM/yyyy'}}</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>
  `
})
export class AppComponent {
  myDate = new Date(2004, 4, 21);
  welcome: string = "Hello World!";
  percentage: number = 0.1514;
  money: number = 23.45;
}
```

UAH23.45

€23.45

UAH23.5

€23.5

тільки сьогодні по ціні 23.45

Рис.3.3 Результат

Ланцюжки pipes

Використаємо одразу декілька pipes.

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

@Component({
  selector: 'my-app',
  template: `
    <div>Message - {{message}}</div>
    <div>Декілька pipes - {{message | slice:8:18 | uppercase}}</div>
  `
})
export class AppComponent {
  message = "Writing Laboratory Work #3!";
}
```

Message - Writing Laboratory Work #3!
Декілька pipes - LABORATORY

Рис.3.4 Результат

Створення своїх pipes

Створемо власний pipe з заміною крапки на кому та додаванням зірочок і знаку.

App.component.ts

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

@Component({
  selector: 'my-app',
  template: `
    <div>Число до форматування: {{x}}<br>Число після форматування: {{x |
      format}}</div>
  `
})
export class AppComponent {
  x: number = 15.45;
}
```

Format.pipe.ts

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

@Pipe({
  name: 'format'
})

export class FormatPipe implements PipeTransform {
  transform(value: number, args?: any): string {
    return `* ${value.toString().replace(".", ",")}! *`;
  }
}
```

App.module.ts

```
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';
```

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

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

Рис.3.5 Результат

Передача параметрів у pipes

Додамо ще один pipe, який прийматиме параметри. Це буде клас, який з масиву рядків створюватиме рядок, приймаючи початковий та кінцевий індекси для вибірки даних із масиву.

App.component.ts

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

@Component({
  selector: 'my-app',
  template: `
    <div>{{users | join}}</div>
    <div>{{users | join:1}}</div>
    <div>{{users | join:2:4}}</div>
  `
})
export class AppComponent {
  users = ["Andrii", "Alina", "Serhei", "Kateryna", "Bazhan"];
}
```

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(", ");
  }
}
```

App.module.ts

```
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 { }
```

Andrii, Alina, Serhei, Kateryna, Bazhan
Alina, Serhei, Kateryna, Bazhan
Serhei, Kateryna

Рис.3.6 Результат

Самостійна робота

Створемо піре квадратного кореня числа.

App.component.ts

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

@Component({
  selector: 'my-app',
  template: `
    <div>Число: {{ number }}</div>
    <div>Квадратний корінь: {{ number | sqrt }}</div>
  `
})
export class AppComponent {
  number: number = 25;
}
```

sqrt.pipe.ts

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

@Pipe({
  name: 'sqrt'
})
export class SqrtPipe implements PipeTransform {
  transform(value: number): number | null {
    if (typeof value === 'number' && value >= 0) {
      return Math.sqrt(value);
    } else {
      console.warn('SqrtPipe: Значення повинно бути невід'ємним числом.');
```

App.module.ts

```
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';
import { SqrtPipe } from './sqrt.pipe';

@NgModule({
  imports: [ BrowserModule, FormsModule ],
  declarations: [ AppComponent, FormatPipe, JoinPipe, SqrtPipe ],
  bootstrap: [ AppComponent ]
})

export class AppModule { }
```

Число: 25
Квадратний корінь: 5

Рис.3.7 Результат

Pure та Impure Pipes

Angular-додаток Pipes2. Вправа 2: Pure та Impure Pipes

Створемо додаток Pipes2.

Протестуємо Pure pipe (Викликаються лише тоді, коли змінюються вхідні дані. Це робить їх ефективними.) використовуючи pipe з минулої вправи.

App.component.ts

```
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;
}
```

format.pipe.ts

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

@Pipe({
  name: 'format'
})

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

App.module.ts

```
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';

@NgModule({
  imports: [ BrowserModule, FormsModule ],
  declarations: [ AppComponent, FormatPipe ],
  bootstrap: [ AppComponent ]
})

export class AppModule { }
```

1521.42
Результат: 1521,42

Рис.3.8 Результат

Протестуємо Inpure pipe (Викликаються при кожному циклі зміни (change detection). Зазвичай використовуються з масивами, об'єктами.) використовуючи pipe з минулої вправи і додавши параметр pure: false.

App.component.ts

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

@Component({
  selector: 'my-app',
  template: `
    <input #user name="user" class="form-control">
    <button class="btn" (click)="users.push(user.value)">Add</button>
    <p>{{users | join}}</p>
  `
})
export class AppComponent {
  users = ["Andrii", "Alina", "Serhei", "Kateryna", "Bazhan"];
}
```

join.pipe.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 {
    return array.join(", ");
  }
}

```

App.module.ts

```

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 { }

```

Andrii, Alina, Serhei, Kateryna, Bazhan, Alex

Рис.3.9 Результат

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.

`App.component.ts`

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

@Component({
  selector: 'my-app',
  template: `
    <p>Модель: {{ phone | async }}</p>
    <button (click)="showPhones()">Посмотреть модели</button>
  `
})
export class AppComponent {
  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]));
  }
}
```

Модель: Nexus 6P

Посмотреть модели

Рис.3.10 Результат

Angular-додаток Pipes3. 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. Визначимо наступний проект

App.component.ts

```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';
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: [HttpClient]
})
export class AppComponent {
  users: Observable<Object> | undefined;
  constructor(private http: HttpClient) {}
  ngOnInit() {
    this.users = this.http.getUsers();
  }
}
```

http.service.ts

```
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');
  }
}
```

App.module.ts

```
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 { }
```

src/assets/users.json

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

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

Рис.3.11 Результат

Детальний огляд компоненту post додатку Blog

Компонент `post` відповідає за відображення окремого посту та надає можливість його видалення. Він є дочірнім компонентом для `app.component`, отримуючи дані про пост через `@Input` і передаючи ідентифікатор посту для видалення через `@Output`.

1. Вхідні дані

- Використовує декоратор `@Input()` для отримання об'єкта посту `myPost` (типу `Post`).

2. Події

- Використовує декоратор `@Output()` для емісії події `onRemove`, яка передає `id` посту, що видаляється.

3. Життєвий цикл

- `ngOnInit()` — метод викликається при ініціалізації компонента.
- `ngOnDestroy()` — викликається перед знищенням компонента; виводить лог повідомлення.

4. Шаблон

- Відображає інформацію про пост: заголовок, текст, дату створення.
- Кнопка `Delete` дозволяє видалити пост, викликаючи метод `removePost()`.

5. Стили

- Шаблон використовує прості стилі через клас `card` для оформлення.

Код

`post.component.ts`

```
import { Component, Input, OnInit, OnDestroy, Output, EventEmitter } from
 '@angular/core';
import { Post } from '../app.component';
@Component({
  selector: 'post',
  templateUrl: './post.component.html',
  styleUrls: ['./post.component.css']
})
export class PostComponent implements OnInit, OnDestroy {
  @Input() myPost!:Post;
  @Output() onRemove=new EventEmitter<number>()
  constructor() { }
  removePost(){
    this.onRemove.emit(this.myPost.id)
  }
  ngOnInit(): void {
  }
  ngOnDestroy(){
    console.log('метод ngOnDestroy');
  }
}
```

post.component.html

```
<div class="card">
  <p style="text-align:right">Дата: {{myPost.date | date:'HH:mm:ss
dd:MM:yyyy'}}</p>
  <h2>{{myPost.title}}</h2>
  <p>{{myPost.text}}</p>
  <button class="btn" (click)="removePost()">Delete</button>
  <ng-content></ng-content>
</div>
```

Дата: 17:58:34 22:12:2024

Рис.3.12 Результат

Детальний огляд компоненту post-form додатку Blog

Компонент post-form дозволяє створювати нові пости. Він є дочірнім компонентом для app.component і надсилає дані нового посту через @Output.

1. Події

- Використовує декоратор @Output() для емісії події onAdd, яка передає об'єкт нового посту (типу Post).

2. Функціонал

- Метод addPost() перевіряє наявність даних у полях title і text. Якщо вони заповнені, створюється новий об'єкт посту з унікальним id та поточною датою, який передається в onAdd.

3. Потік даних

- Властивість myDate\$ — це Observable, який емулює потік часу. Дата оновлюється кожну секунду.

4. Шаблон

- Забезпечує текстові поля для введення заголовку (title) і тексту (text) посту.
- Кнопка Додати пост викликає метод addPost().

Код

post-form.component.ts

```
import { Component, EventEmitter, OnInit, Output } from '@angular/core';
import { Post } from '../app.component';
import { Observable } from 'rxjs';
@Component({
  selector: 'post-form',
  templateUrl: './post-form.component.html',
  styleUrls: ['./post-form.component.css']
})
export class PostFormComponent implements OnInit {
  @Output() onAdd: EventEmitter<Post> = new EventEmitter<Post>()
  title='';
```

```

text='';

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

addPost(){
    if (this.title.trim() && this.text.trim()){
        const post: Post={
            id: Date.now(),
            title: this.title,
            text: this.text,
            date: this.date_post
        }
        this.onAdd.emit(post);
        console.log('New post', post);
        this.title=this.text='';
    }
}

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

```

post-form.component.html

```

<div>
    <p style="text-align:right">Дата: {{myDate$ | async | date:'HH:mm:ss
dd:MM:yyyy'}}</p>
    <input
        type="text"
        class="form-control"
        placeholder="Title..."
        [(ngModel)]="title">
    <input
        type="text"
        class="form-control"
        placeholder="Text..."
        [(ngModel)]="text">
    <button class="btn" (click)="addPost()">Додати пост</button>
</div>

```

Search...

Дата: 17:58:20 22:12:2024

Вивчаю компоненти

Створюю проект "Блог"

Delete

Пост довгий

Дата: 17:58:20 22:12:2024

Вивчаю директиви

Все ще створюю проект"Блог"

Delete

Пост довгий

Рис.3.13 Результат

ВИСНОВКИ

У цій лабораторній роботі ми детально ознайомилися з концепцією **Pipes** в Angular, їхнім призначенням, використанням, а також із методами створення власних pipes.

Основні результати:

Pipes в Angular є інструментами для перетворення даних безпосередньо в шаблонах компонентів. Вони дозволяють форматовувати текст, числа, дати, валюти, а також обробляти асинхронні дані.

Ми розглянули широкий набір вбудованих pipes, таких як:

- uppercase / lowercase для зміни регістру тексту.
- date для форматування дат.
- currency для відображення чисел у вигляді валюти.
- percent, number та інші для роботи з числами. Ці pipes значно спрощують роботу з шаблонами та зменшують обсяг коду.

Ми навчилися створювати власні pipes для виконання специфічних завдань. Це дозволяє адаптувати функціонал до потреб конкретного проєкту. Наприклад:

- Pipe для заміни символів у числі (format).
- Pipe для з'єднання масиву рядків (join).
- Pipe для обчислення квадратного кореня (sqrt).

Було розглянуто різницю між чистими та нечистими pipes:

Pure pipes викликаються лише за умови зміни вхідних даних, що робить їх ефективними для використання.

Impure pipes викликаються при кожній перевірці змін, тому вони корисні для роботи з динамічними масивами та об'єктами.

AsyncPipe є вбудованим нечистим pipe, який дозволяє працювати з асинхронними даними, такими як Observable або Promise. Він автоматично відписується від потоків даних після знищення компонента, що підвищує безпеку додатка.

Застосування Pipes для роботи з серверними даними
Pipes можуть бути корисними при обробці даних, отриманих з серверу. Використовуючи **AsyncPipe**, можна спростити роботу з потоками асинхронних даних, наприклад, при завантаженні інформації про користувачів.

Додатки Components1 та Components5 були успішно розгорнуті на платформі Firebase у відповідних проектах з іменами <https://mieshkovip15laba3-1.web.app/> та <https://mieshkovip15laba3-4.web.app/>.

Pipes є невід'ємною частиною Angular, які дозволяють суттєво оптимізувати роботу з даними в шаблонах. Їх застосування зменшує обсяг коду в компонентах, покращує його читабельність та полегшує підтримку проєкту. Вивчення custom pipes і їхнього взаємозв'язку з вбудованими pipes дає розробникам інструменти для гнучкої роботи з даними.

СПИСОК ДЖЕРЕЛ

1. Документація Angular. URL: <https://v17.angular.io/docs>
2. Документація Nodejs. URL: <https://nodejs.org/docs/latest/api/>
3. Документація Firebase. URL: <https://firebase.google.com/docs?hl=en>