

Лабораторне завдання №4: Створення проекту «Blog».

Створення компонентів.

Мета: Навчитися використовувати інтерфейс командного рядка Angular CLI при створенні проекту та компонентів.

Завдання:

A) Створити проект “Blog” та два компоненти `post` та `post-form`. Компонент `post-form` – для створення нового поста. Компонент `post` – для відображення існуючих постів.

B) Зробити звіт по роботі;

C) Angular-додаток `Blog` розгорнути на платформі `Firebase` у проекті з ім'ям «ПрізвищеГрупаLaba4», наприклад «KovalenkoIP01Laba4».

A) Створення проекту “Blog” та двох компонентів `post` та `post-form`. Компонент `post-form` – для створення нового поста. Компонент `post` – для відображення існуючих постів.

I. Створення проекту та каркасу додатку.

1) Створіть проект “Blog” при допомозі команди:

```
ng new Blog
```

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

```
<div class="container">
  <h1>Angular Components</h1>
</div>
```

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

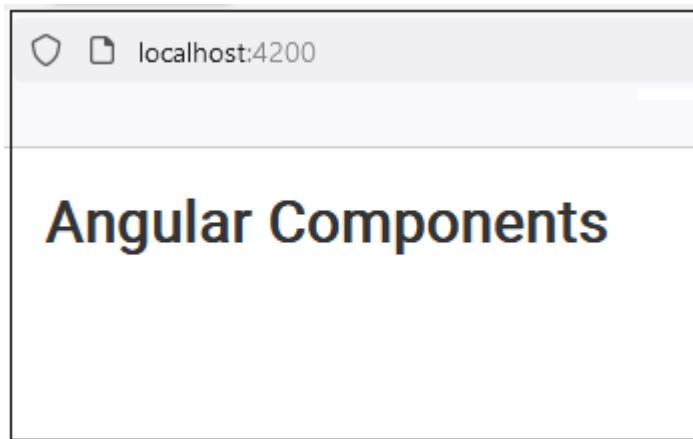
```
@import url('https://fonts.googleapis.com/css?family=Roboto');
* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}
body {
  font-family: 'Roboto', sans-serif;
  font-size: 1rem;
  line-height: 1.6;
  background-color: #fff;
  color: #333;
}
.container {
  max-width: 1000px;
  margin: 0 auto;
  padding-top: 1rem;
}
a {
  text-decoration: none;
}
a:hover {
  color: #666;
```

```

}
ul {
  list-style: none;
}
img {
  width: 100%;
}
.btn {
  display: inline-block;
  background: #333333;
  color: #fff;
  padding: 0.4rem 1.3rem;
  font-size: 1rem;
  border: none;
  cursor: pointer;
  margin-right: 0.5rem;
  transition: opacity 0.2s ease-in;
  outline: none;
}
.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 «СЦЕНАРИЙ NPM» запустіть проект на виконання. Ви отримаєте наступний результат:



- 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`

- 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`

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:

```
<div class="container">
  <h1>Angular Components</h1>
  <app-post-form></app-post-form>
  <hr/>
  <app-post></app-post>
</div>
```

- 8) В шаблоні, який призначений для створення нового посту блога, створіть два поля для введення даних посту "Title", "Text" та кнопку `<button>` для додавання нового посту. Наприклад так:

```
<div>
  <input type="text" class="form-control" placeholder="Title...">
  <input type="text" class="form-control" placeholder="Text...">
  <button class="btn">Додати пост</button>
</div>
```

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

```

<div class="card">
  <h2>Post title</h2>
  <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Dicta, atque.</p>
</div>

```

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

```

<div class="container">
  <app-post-form></app-post-form>
  <hr/>
  <app-post></app-post>
  <app-post></app-post>
</div>

```

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

The screenshot shows a web application interface. At the top, there is a form with two input fields: "Title..." and "Text...". Below these fields is a dark button labeled "Додати пост" (Add post). A horizontal line separates the form from the list of posts below. The list contains two identical post items. Each item has a title "Post title" in bold red text and a body of text "Lorem ipsum dolor sit amet consectetur adipisicing elit. Dicta, atque." in red text. The entire content is enclosed in a light blue border.

II. 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.

```

import { Component } from '@angular/core';
export interface Post {
  title:string;
  text:string;

```

```

    id?:number;
  }
  @Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
  })
  export class AppComponent {
    title = 'BlogComponents';
    posts: Post[]=[{title:'Вивчаю компоненти', text:'Створюю проект "Блог"', id:1},
      {title:'Вивчаю директиви', text:'Все ще створюю "Блог"', id:2}]
  }

```

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

```

<div class="container">
<app-post-form></app-post-form>
<hr/>
<app-post
  *ngFor="let p of posts"
  [myPost]="p"
></app-post>
</div>

```

При чому, через змінну myPost в шаблоні app.component.html компонента app.component.ts ми будемо передавати дані в компонент post.component.ts.

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

```

import { Component, Input, OnInit } from '@angular/core';
import { Post } from '../app.component';
@Component({
  selector: 'app-post',
  templateUrl: './post.component.html',
  styleUrls: ['./post.component.css']
})
export class PostComponent implements OnInit {
  @Input() myPost!:Post;
  constructor() { }
  ngOnInit(): void {
  }
}

```

TypeScript видає помилки, якщо не ініціалізувати всі властивості класів під час побудови. Якщо неможливо ініціалізувати дані безпосередньо, але Ви впевнені, що властивість буде призначено під час виконання, можна використовувати оператор

затвердження певного присвоєння ! (Оператор затвердження ненульового значення (non-null assertion operator)), щоб попросити TypeScript ігнорувати цю властивість.

Тобто TypeScript надає спеціальний синтаксис для видалення null і undefined із типу без необхідності виконання явної перевірки. Вказівка ! після виразу означає, що цей вираз не може бути нульовим, тобто мати значення null або undefined.

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

```
<div class="container">
  <app-post-form></app-post-form>
  <hr/>
  <app-post
    *ngFor="let p of posts"
    [toPost]="p"
  ></app-post>
</div>
```

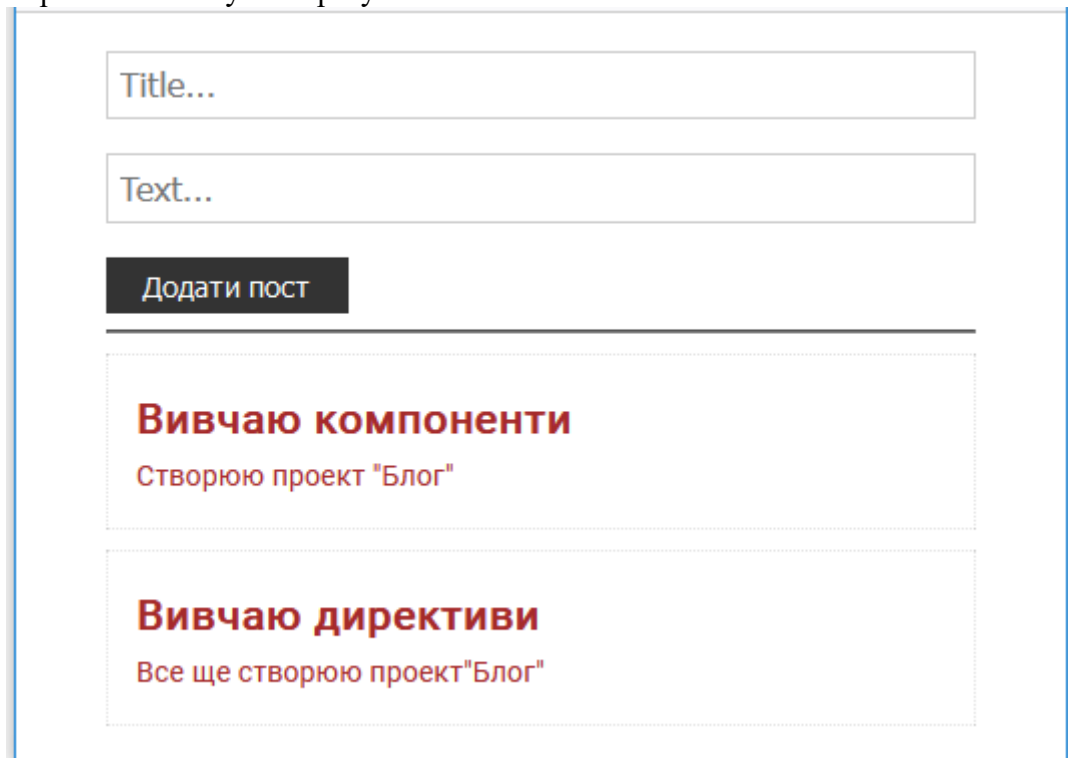
а в компоненті post.component.ts ми хочемо залишити змінну myPost, то в декораторі ми повинні записати так:

```
Input('toPost') myPost!:Post;
```

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

```
<div class="card">
  <h2>{{myPost.title}}</h2>
  <p>{{myPost.text}}</p>
</div>
```

Отримаємо наступний результат:



The screenshot displays a web interface within a blue border. At the top, there is a form with two input fields: 'Title...' and 'Text...'. Below these fields is a dark button labeled 'Додати пост'. A horizontal line separates the form from the content area below. The content area contains two cards, each with a red heading and a line of text. The first card has the heading 'Вивчаю компоненти' and the text 'Створюю проект "Блог"'. The second card has the heading 'Вивчаю директиви' and the text 'Все ще створюю проект "Блог"'. The cards are separated by a dashed line.

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

```
<div class="container">
  <app-post-form></app-post-form>
  <hr/>
  <app-post
    *ngFor="let p of posts"
    [toPost]="p"
  ></app-post>
</div>
```

а в компоненті `post.component.ts` ми хочемо залишити змінну `myPost`, то в декораторі ми повинні записати так:

```
Input('toPost') myPost!:Post;
```

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 переконайтесь, що в модулі `app.module.ts` у вас імпортований модуль `FormsModule` для реалізації двосторонньої прив'язки (`two-way binding`).

```
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { PostFormComponent } from './post-form/post-form.component';
import { PostComponent } from './post/post.component';
```

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

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

```
<div>
  <input
    type="text"
    class="form-control"
    placeholder="Title..."
    [(ngModel)]="title">
  <input
    type="text"
    class="form-control"
    placeholder="Text..."
    [(ngModel)]="text">
  <button class="btn">Додати пост</button>
</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»:

```
import { Component, OnInit } from '@angular/core';
import { Post } from '../app.component';
@Component({
  selector: 'app-post-form',
  templateUrl: './post-form.component.html',
  styleUrls: ['./post-form.component.css']
})
export class PostFormComponent implements OnInit {
  title="";
  text="";

  constructor() { }
  ngOnInit(): void {
  }

  addPost(){
    if (this.title.trim() && this.text.trim()){
      const post: Post={
        title:this.title,
        text:this.text
      }
    }
  }
}
```



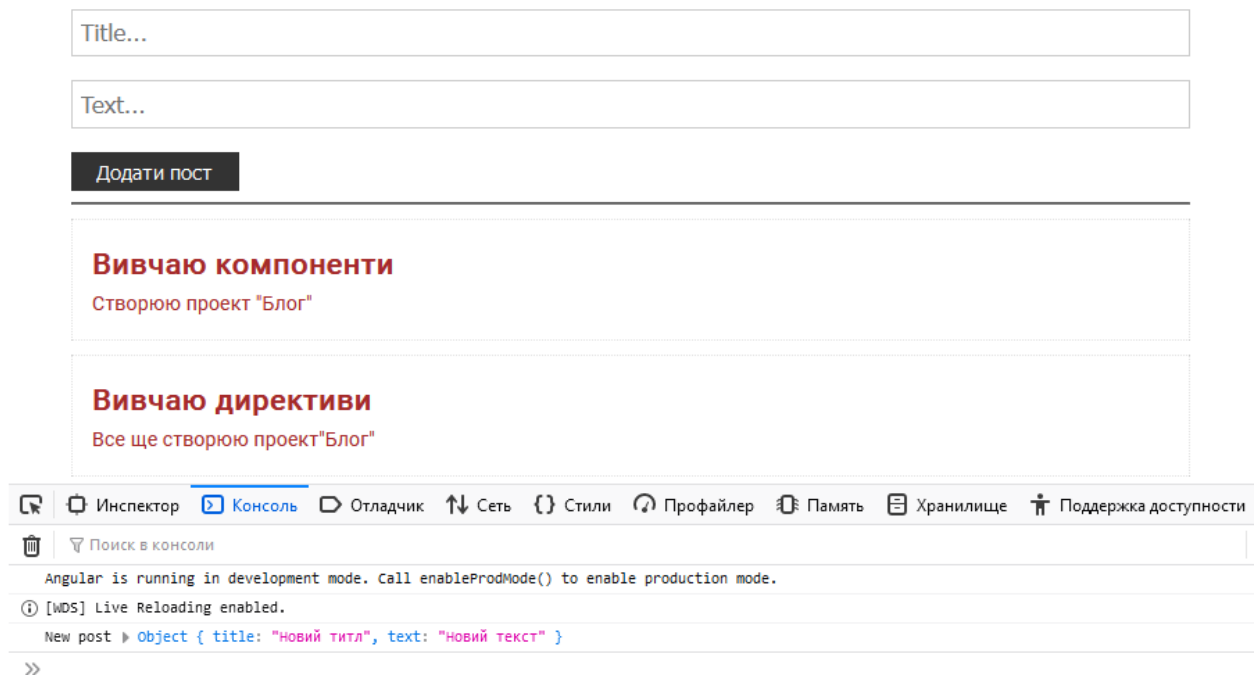
```

console.log('New post',post);
this.title=this.text="";          // очищення полів
}
}
}

```

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:



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 компонента 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().

```

import { Component, EventEmitter, OnInit, Output } from '@angular/core';
import { Post } from '../app.component';

@Component({
  selector: 'app-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() { }
  ngOnInit(): void {
  }
  addPost(){
    if (this.title.trim()&&this.text.trim()){
      const post: Post={
        title:this.title,
        text:this.text
      }
      this.onAdd.emit(post);
      console.log('New post',post);
      this.title=this.text="";
    }
  }
}

```

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 `<app-post-form>` наступним чином:

```

<app-post-form
  (onAdd)="updatePosts($event)"
></app-post-form>

```

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

```

updatePosts(post:Post){
  this.posts.unshift(post);
}

```

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.

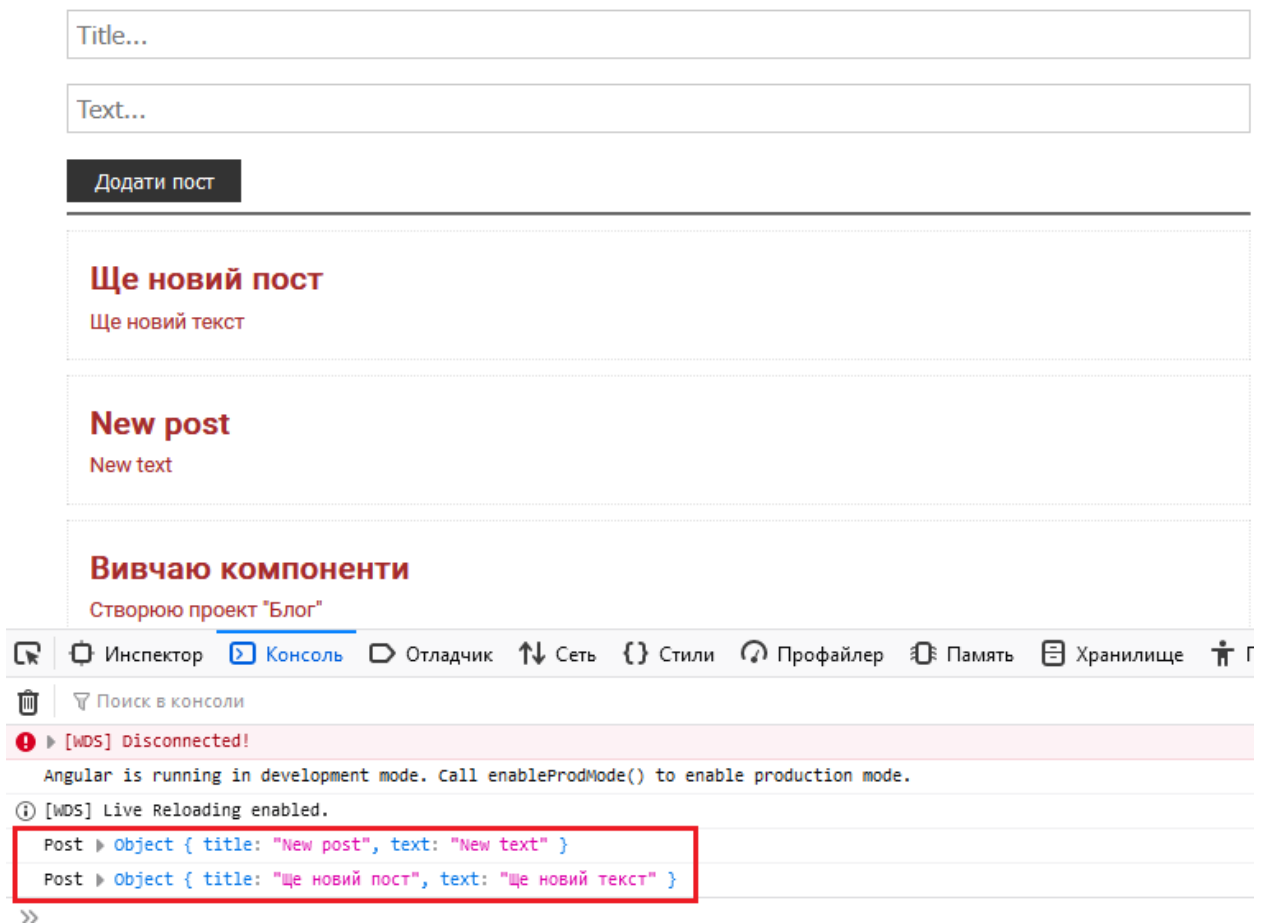
В результаті будемо мати можливість додавати пости, як на рисунку нижче. В консолі розробника, якщо до метода `updatePosts(post:Post)` додати виведення у консоль:

```

console.log('Post',post);

```

можна також побачити дані 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 `<app-post>` на наступне:

```
<app-post
  *ngFor="let p of posts"
  [myPost]="p"
>
<small *ngIf="p.text.length>20; else short">Пост довгий</small>
<ng-template #short>
  <small>Пост короткий</small>
</ng-template>
</app-post>
```

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></ng-content>`, який ще називають проекцією контенту, таким чином:

```

<div class="card">
  <h2>{{myPost.title}}</h2>
  <p>{{myPost.text}}</p>
  <ng-content></ng-content>
</div>

```

`<ng-content>` дає можливість імпортувати HTML контент ззовні компонента та вставити його в шаблон іншого компонента в певне місце.

В результаті будемо мати:

The screenshot shows a web interface for managing posts. At the top, there is a form with two input fields: 'Title...' and 'Text...'. Below these fields is a dark button labeled 'Додати пост' (Add post). Underneath the button is a horizontal line, followed by a list of three posts, each enclosed in a dashed border. The first post has the title 'еанкен' (eanken), the text 'аенр' (aenr), and the status 'Пост короткий' (Short post). The second post has the title 'Вивчаю компоненти' (Learning components), the text 'Створюю проект "Блог"' (Creating project "Blog"), and the status 'Пост довгий' (Long post). The third post has the title 'Вивчаю директиви' (Learning directives), the text 'Все ще створюю проект "Блог"' (Still creating project "Blog"), and the status 'Пост довгий' (Long post).

25) Останнім кроком у нас буде додавання можливості для видалення будь-якого поста і демонстрація роботи метода `ngOnDestroy()` (<https://angular.io/guide/lifecycle-hooks>). Спочатку додамо в компонент, який відповідає за відображення постів `post.component.ts` метод `ngOnDestroy()` та імплементуємо відповідний інтерфейс `OnDestroy`.

```

import { Component, Input, OnInit, OnDestroy } from '@angular/core';
import { Post } from '../app.component';

```

```

@Component({
  selector: 'app-post',
  templateUrl: './post.component.html',
  styleUrls: ['./post.component.css']
})
export class PostComponent implements OnInit, OnDestroy {

```

```

@Input() myPost!: Post;
constructor() { }
ngOnInit(): void {
}
ngOnDestroy(){
  console.log('метод ngOnDestroy');
}
}

```

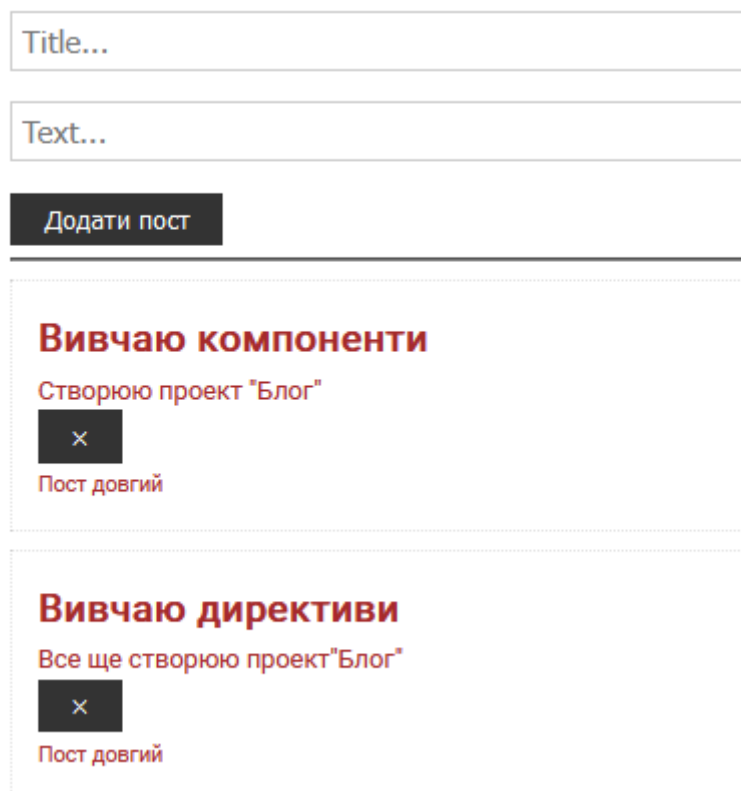
26) Далі реалізуємо логіку видалення поста. Спочатку створимо кнопку в шаблоні post.component.html для видалення поста:

```

<div class="card">
  <h2>{{myPost.title}}</h2>
  <p>{{myPost.text}}</p>
  <button class="btn" (click)="removePost()">&times;</button>
  <ng-content></ng-content>
</div>

```

Отримаємо:



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

```

import { Component, Input, OnInit, OnDestroy, Output, EventEmitter } from
'@angular/core';
import { Post } from '../app.component';

```

```
@Component({
  selector: 'app-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');
  }
}
```

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:

```
<app-post
  *ngFor="let p of posts"
  [myPost]="p"
  (onRemove)="deletePost($event)"
>
```

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

```
deletePost(id:number){
  this.posts=this.posts.filter(p=>p.id!==id)
}
```

В результаті, після видалення поста ми побачимо, що визивається метод ngOnDestroy, який часто використовується для того, щоб відписуватись від різних слухачів.

Title...

Text...

Додати пост

Вивчаю компоненти

Створюю проект "Блог"

×

Пост довгий

Вивчаю директиви

Все ще створюю проект "Блог"

×

Пост довгий

Инспектор Консоль Отладчик Сеть Стили >> Поиск в консоли

Ошибки Предупреждения Лог Инфо Отладка CSS XHR Запросы

Angular is running in development mode. Call enableProdMode() to enable production mode. [core.js:28](#)

[WDS] Live Reloading enabled. [index.js](#)

Title...

Text...

Додати пост

Вивчаю директиви

Все ще створюю проект "Блог"

×

Пост довгий

Инспектор Консоль Отладчик Сеть Стили >> Поиск в консоли

Ошибки Предупреждения Лог Инфо Отладка CSS XHR Запросы

Angular is running in development mode. Call enableProdMode() to enable production mode. [core.js:280](#)

[WDS] Live Reloading enabled. [index.js:!](#)

метод ngOnDestroy [post.component.ts:20:1](#)

В) Зробити звіт по роботі. Звіт повинен бути не менше 5 сторінок без титульного аркуша (шрифт Times New Roman, 14, полуторний інтервал). Титульний аркуш приводиться у додатку. Звіт повинен містити наступні розділи:

- 1) Детальний огляд компоненту post;
- 2) Детальний огляд компоненту post-form.

С) Angular-додаток Blog розгорнути на платформі Firebase у проекті з ім'ям «ПрізвищеГрупаLaba4», наприклад «KovalenkoIP01Laba4».

**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