Лабораторна робота № 8

Використання сервісів та роутингу.

Сервіси в Angular представляють класи, які виконують деякі специфічні завдання, наприклад, ведення логів, роботу з даними і т.д. На відміну від компонентів і директив сервіси не працюють з представленнями, тобто з розміткою html. Вони виконують строго вузьке спеціалізоване завдання.

Сервіс може інкапсулювати бізнес-логіку, різні обчислювальні завдання, або інші завдання, які краще виносити з компонентів. Завдяки цьому код компонентів може бути зосереджений тільки на роботі з представленням. Крім того, тим самим ми також можемо вирішити проблему повторення коду, якщо нам буде потрібно виконати одну і ту ж задачу в різних компонентах і класах.

Додамо до нашого застосунку сервіс, що візьме на себе роботу із даними списку студентських груп. Для цього виконаємо команду Angular CLI для створення нового сервісу groups-data (рис. 1).

```
ng generate service services/groups-data
```

Рис. 1. Створення сервісу groups-data.

Відкриємо файл groups-data.service.ts та переглянемо його вміст (рис. 2).

```
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class GroupsDataService {
7
8    constructor() { }
9  }
```

Рис. 2. Файл створеного сервісу groups-data.service.ts.

Зауважте, що новий сервіс імпортує Angular Injectable та анотує клас за допомогою декоратора @Injectable(). Це позначає клас як такий, який бере участь у

системі dependency injection. Клас GroupsDataService зможе бути включений до інших класів, а також може мати власні залежності. Декоратор @Injectable() приймає об’єкт метаданих для сервісу, так само, як і декоратор @Component() для наших класів компонентів.

За замовчуванням команда Angular CLI ng generation service реєструє провайдера з root ін’єктором «providedIn: 'root'» в декораторі @Injectable(). Коли ми надаємо сервіс на рівні root, Angular створює єдиний спільний екземпляр GroupsDataService та вводить його у будь-який клас, який цього вимагає. Реєстрація провайдера в метаданих @Injectable також дозволяє Angular оптимізувати застосунок, видаливши службу, якщо виявиться, що вона не використовується.

Реалізуємо функціонал нашого сервісу, перенесши у нього масив із студентськими групами та методи додавання та видалення елементів із класу компоненти AppComponent (рис. 3).

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

@Injectable({
  providedIn: 'root'
})

export class GroupsDataService {
  private groups = [
    {
        number: 201,
        faculty: `факультет комп'ютерних наук`,
        specialty: `комп'ютерні науки`,
        studentsQuantity: 22
    },

    {
        number: 308,
        faculty: `факультет комп'ютерних наук`,
        specialty: `інженерія програмного забезпечення`,
        studentsQuantity: 24
    },
  ];
```

```
21
22      constructor() { }
23
24      getGroups() {
25          return this.groups;
26      }
27
28      addGroup(group) {
29          this.groups.push(group);
30      }

31
32      deleteGroup(index) {
33          this.groups.splice(index,  deleteCount: 1);
34      }
35  }
```

Рис. 3. Код сервісу GroupsDataService.

Із класу компоненти AppComponent прибираємо масив та методи по роботі із ним, які зараз представлені у сервісі GroupsDataService, та у конструкторі підключаємо сервіс і отримуємо дані із нього до локальної змінної (рис. 4).

```
1   import { Component } from '@angular/core';
2   import {GroupsDataService} from './services/groups-data.service';
3
4   @Component({
5     selector: 'app-root',
6     templateUrl: './app.component.html',
7     styleUrls: ['./app.component.scss']
8   })

9   export class AppComponent {
10    groups: any[];
11
12    constructor(private groupService: GroupsDataService) {
13      this.groups = groupService.getGroups();
14    }
15  }
```

Рис. 4. Файл app.component.ts.

Із шаблону App компоненти також прибираємо код, що відповідає за повернення даних із дочірніх компонент, оскільки у них ми також будемо працювати із створеним сервісом (рис. 5).

```
<nav>
    <div class="nav-wrapper">
        <a class="brand-logo">Групи</a>
    </div>
</nav>
<div class="container">
    <app-new-group></app-new-group>
    <app-group
            *ngFor="let group of groups; let i = index"
            [group]="group" [grpIndex]="i">
    </app-group>
</div>
```

Рис. 5. Представлення app.component.html.

Далі виконаємо підключення сервісу у компоненті GroupComponent. Тепер при видаленні групи замість передачі події у батьківський елемент будемо викликати відповідний метод сервісу GroupsDataService (рис. 6).

```
export class GroupComponent implements OnInit {
  @Input() group;
  @Input() grpIndex;
  showInfo = false;

  constructor(private groupService: GroupsDataService) { }

  ngOnInit() {
  }

  delGroup() {
    this.groupService.deleteGroup(this.grpIndex);
  }
}
```

Рис. 6. Виконання змін у GroupComponent.

Аналогічні зміни виконуємо у компоненті NewGroupComponent для реалізації операції додавання студентської групи із використанням створеного сервісу (рис. 7).

```
export class NewGroupComponent implements OnInit {

  showForm = false;

  constructor(private groupService: GroupsDataService) { }

  ngOnInit() {
  }

  onSubmit(myForm) {
    const fields = myForm.form.controls;
    this.showForm = false;
    this.groupService.addGroup( group: {
      number: fields.number.value,
      faculty: fields.faculty.value,
      specialty: fields.specialty.value,
      studentsQuantity: fields.studentsQuantity.value
    });
  }
}
```

Рис. 7. Виконання змін у NewGroupComponent.

Перевіримо роботу застосунку – все має працювати, як і раніше.

В реальності процес отримання даних може займати деякий час (читання файлу, даних БД, отримання відповіді на http-запит і т.д.). Тому більш правильно у даному випадку робити це асинхронно. Для реалізації асинхронного отримання даних використаємо бібліотеку RxJS, що дозволяє управляти асинхронними операціями і подіями в застосунку в стилі реактивного програмування. Вона побудована на основі патерну проектування Observer і передбачає цілий ряд операторів для маніпуляції асинхронними подіями і обробки переданих ними даних.

Для початку створюється інстанція observable, що генеруватиме подію (у нашому випадку отримання даних). Далі на подію підписується observer (відображення отриманих даних у компоненті App), який виконує вказану дію при виникненні події у observable. Для створення observable використаємо функцію бібліотеки RxJS of(), яка створює інстанцію observable, яка видає елементи масиву, переданого як параметр функції один за одним. Виконаємо необхідні зміни у сервісі groups-data (рис. 8).

```
24
25      getGroups(): Observable<any[]> {
26          return of(this.groups);
27      }
```

Рис. 8. Повернення observable із методу getGroups класу GroupsData.

Інстанція observable також може бути створена із використанням конструкції new (рис. 9).

```
25      getGroups(): Observable<any[]> {
26          return new Observable<any[]>(
27              subscribe: subscriber => {
28                  subscriber.next(this.groups);
29                  subscriber.complete();
30              }
31          );
32      }
```

Рис. 9. Інший спосіб створення observable.

Далі у конструкторі класу AppComponent підпишимося на подію отримання даних від сервісу та передамо отримані дані у змінну groups (рис. 10).

```
12      constructor(private groupService: GroupsDataService) {
13        groupService.getGroups().subscribe(
14              next: (groups) => this.groups = groups
15        );
16      }
```

Рис. 10. Створення observer.

Перевіримо роботу застосунку – все має працювати, як і раніше.

У браузері є звична модель навігації по застосунку: при введенні URL-адреси браузер перейде на відповідну сторінку; при переході за посиланням браузер відкриває нову сторінку; при натисканні на кнопку назад і вперед браузера виконується переміщення назад і вперед згідно історії відвідування сторінок.

Маршрутизація Angular дозволяє здійснювати навігацію від одного представлення до іншого. Вона може інтерпретувати URL-адресу браузера як команду для переходу до іншого представлення. Вона також може передавати

необов'язкові параметри до компоненти, що робить можливим динамічно визначати його вміст. Маршрутизатор записує активність у журнал історії веб-переглядача, що дає можливість працювали із кнопками назад та вперед.

Реалізуємо маршрутизацію у нашому застосунку на прикладі додавання ще одного представлення – списку студентів, що входять до студентської групи. Додамо до застосунку 2 компонента: список студентських груп та список студентів (рис. 11).

```
>ng g c group-list --skipTests

>ng g c student-list --skipTests
```

Рис. 11. Додавання двох нових компонентів.

У компонент group-list перенесемо функціонал із AppComponent (рис. 12-13).

```
1  import { Component, OnInit } from '@angular/core';
2  import {GroupsDataService} from '../services/groups-data.service';
3
4  @Component({
5    selector: 'app-group-list',
6    templateUrl: './group-list.component.html',
7    styleUrls: ['./group-list.component.scss']
8  })
9  export class GroupListComponent implements OnInit {
10
11   groups: any[];
12
13   constructor(private groupService: GroupsDataService) {
14     groupService.getGroups().subscribe(
15       next: (groups) => this.groups = groups
16     );
17   }
18
19   ngOnInit() {
20   }
21
22 }
```

Рис. 12. group-list.component.ts.

```
<h5 class="center-align">Список груп</h5>
<app-new-group></app-new-group>
<app-group
        *ngFor="let group of groups; let i = index"
        [group]="group" [grpIndex]="i">
</app-group>
```

Рис. 13. group-list.component.html.

В модулі застосунку app.module.ts пропишимо додатковий функціонал, що дозволить використовувати маршрутизацію Angular та створить необхідні нам маршрути (рис. 14).

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

import { AppComponent } from './app.component';
import {MyComponent} from './my/my.component';
import { GroupComponent } from './group/group.component';

import {FormsModule} from '@angular/forms';
import { NewGroupComponent } from './new-group/new-group.component';
import {RouterModule, Routes} from '@angular/router';
import { GroupListComponent } from './group-list/group-list.component';
import { StudentListComponent } from './student-list/student-list.component';

const routes: Routes = [
  {path: 'groups', component: GroupListComponent},
  {path: 'students/:grpId', component: StudentListComponent},
  {path: '', redirectTo: 'groups', pathMatch: 'full'}
];

@NgModule({
  declarations: [
    AppComponent, MyComponent, GroupComponent, NewGroupComponent,
    GroupListComponent, StudentListComponent
  ],

  imports: [
    BrowserModule,
    FormsModule,
    RouterModule.forRoot(routes)
  ],

  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Рис. 14. Створення та підключення маршрутів.

Змінимо кореневий компонент – AppComponent, приберемо з нього все зайве та додаємо область для виведення компонента відповідно до обраного маршруту (рис. 15-16).

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {

  constructor() { }
}
```

Рис. 15. App.component.ts.

```
<nav>
    <div class="nav-wrapper">
        <a class="brand-logo">Групи</a>
    </div>
</nav>
<div class="container">
    <router-outlet></router-outlet>
</div>
```

Рис. 16. App.component.html.

Після цього застосунок маю працювати, як і раніше. Перевіримо це, та перейдемо до реалізації відображення списку студентів групи.

Виконаємо розширення сервісу group-data, додаємо до нього масив студенів students та метод getStudents, що повертає список студентів певної групи, загорнутий у observable (рис. 17).

```
  private students = [
    {name: 'Іванов Василь', groupNumb: 201},
    {name: 'Дмитренко Петро', groupNumb: 201},
    {name: 'Петренко Дарина', groupNumb: 201},
    {name: 'Васильєва Марина', groupNumb: 308},
    {name: 'Павлов Микита', groupNumb: 308},
    {name: 'Васін Андрій', groupNumb: 308},
  ];
```

```
42
43    getStudents(groupNumber: number): Observable<any[]> {
44      return of(this.students.filter( callbackfn: elem => {
45        return elem.groupNumb === groupNumber;
46      }));
47    }
```

Рис. 17. Додавання студентів у сервіс group-data.

У шаблоні group.component.html додаємо посилання для переходу до списку студентів групи (рис. 18).

```
12                  keyboard_arrow_down
13                </i>
14            </a>
15            <a [routerLink] = "'/students/'+group.number"
16               class="btn-floating waves-effect waves-light blue-grey">
17                <i class="material-icons">
18                  view_headline
19                </i>
20            </a>
21        </div>
```

Рис. 18. Додавання посилання у group.component.html.

Переглядаємо результат (рис. 19).

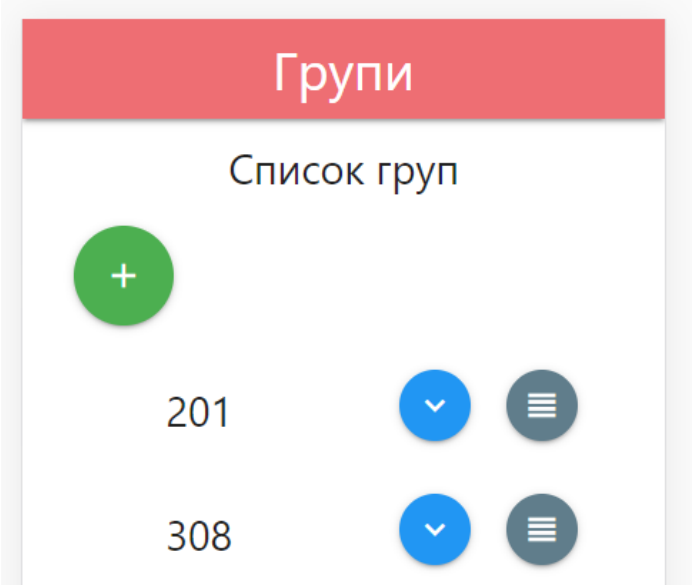

Рис. 19. Посилання на список студентів.

У класі StudentsListComponent реалізуємо отримання значення переданого параметру grpId та метод, який отримає дані щодо списку студентів із сервісу group-data (рис. 20).

```
import { Component, OnInit } from '@angular/core';
import {GroupsDataService} from '../services/groups-data.service';
import {ActivatedRoute} from '@angular/router';

@Component({
  selector: 'app-student-list',
  templateUrl: './student-list.component.html',
  styleUrls: ['./student-list.component.scss']
})
export class StudentListComponent implements OnInit {

  students: any[];

  constructor(private groupService: GroupsDataService,
              private activatedRoute: ActivatedRoute) {
  }

  ngOnInit() {
    this.activatedRoute.params.subscribe(
        next: params => this.getStudents(params.grpId)
    );
  }

  getStudents(numb: string) {
    const n = +numb;
    this.groupService.getStudents(n).subscribe(
        next: (students) => {
          this.students = students;
        }
    );
  }
}
```

Рис. 20. Клас StudentsListComponent.

Також реалізуємо представлення students-list.component.html для виведення списку студентів (рис. 21).

```
<div class="row valign-wrapper">
    <div class="col m6 offset-m3 s12">
        <a [routerLink] = "'/groups'"
           class="btn-floating waves-effect waves-light blue-grey">
            <i class="material-icons">
                keyboard_arrow_left
            </i>
        </a>
    </div>
</div>

<div class="row valign-wrapper">
    <div class="col m6 offset-m3 s12">
        <h5 class="center-align">Список студентів</h5>
    </div>
</div>
<div *ngFor="let student of students" class="row valign-wrapper">
    <div class="col m6 offset-m3 s12">
        <h6>{{student.name}}</h6>
    </div>
</div>
```

Рис. 21. Представлення students-list.component.html.

Переглянемо отриманий результат (рис. 22).

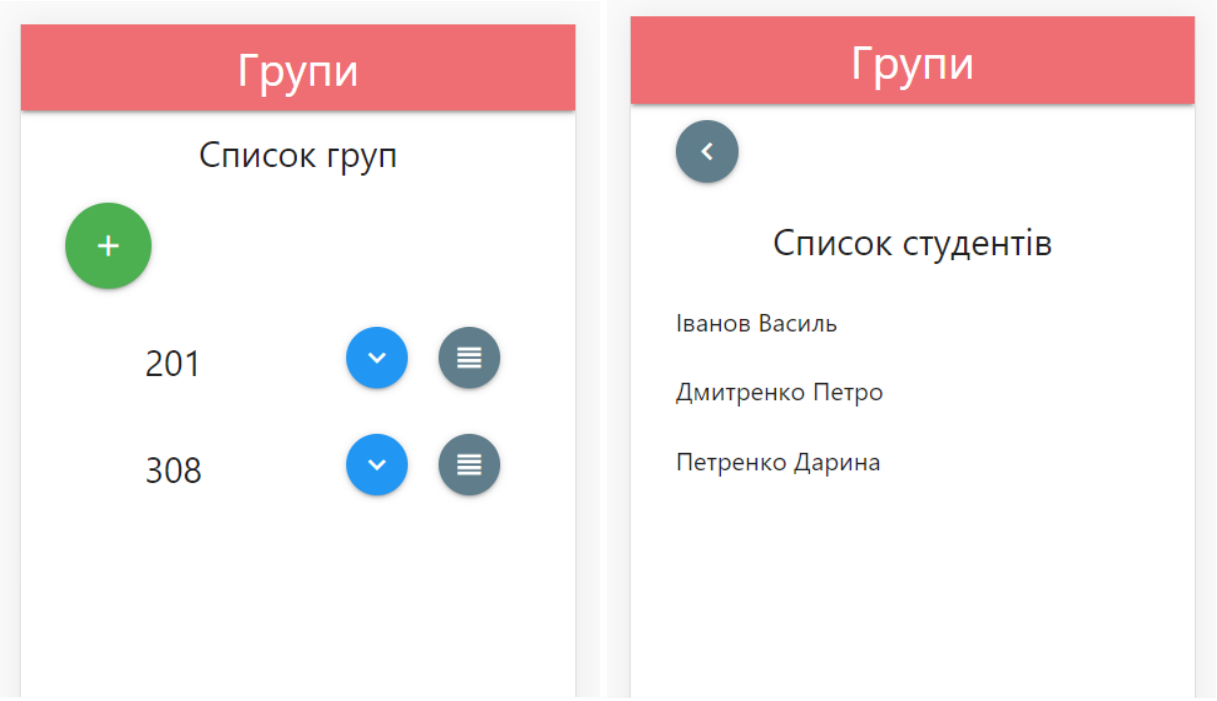

Рис. 22. Список студентів.

Індивідуальне завдання.

На базі Angular Framework реалізувати застосунок із адаптивним інтерфейсом, що забезпечує перегляд, та редагування (включаючи додавання та видалення) даних списку елементів згідно свого варіанту індивідуального завдання. Реалізувати перегляд елементів підпорядкованого списку елементів (згідно варіанту).

Додатково, реалізувати редагування списку підпорядкованих елементів (включаючи додавання та видалення).

При виконанні завдання створити мінімум 3 власні компоненти, реалізувати передачу даних у дочірній компонент та повернення події у батьківський, роботу сервісами та маршрутизацію.