Лабораторна робота 15.

Ionic Framework. Власний back-end та post-запити. Частина 2

Продовжимо реалізовувати функціонал роботи із даними, що зберігаються на сервері БД за допомогою взаємодії із back-end API за допомогою http-сервісу angular.

Почнімо із реалізації перегляду та редагування даних групи. До інтерфейсу StudGroup додаємо атрибут id (рис. 1) та ініціалізуємо його в ngOnInit компоненти stud-group (рис. 2).

```
5   export interface StudGroup {
6     id: number;
7     number: number;
8     faculty: string;
9     specialty: string;
10    studentsQuantity: number;
11  }
```

Рис. 1. Зміни у data-getter.service.ts.

```
19    ngOnInit() {
20      if (this.isNew) {
21        this.studentGroup = {
22          id: null,
23          number: null,
24          specialty: '',
25          studentsQuantity: null,
26          faculty: ''
27        };
28        this.title = 'Нова група';
29      }
30    }
```

Рис. 2. ngOnInit класу компоненти stud-group

В back-end API у класі моделі реалізуємо метод зміни студентської групи (рис. 3) та обробимо відповідну подію у класі контролеру. Також додаємо новий маршрут до масиву захищених (рис. 4).

```
11      public static function editGroup($group) {
12          return (new DB())->runQuery(
13              sql: "update groups set number = '".$group['number']."',
14                  faculty = '" . str_replace( search: "'", replace: "\'", $group['faculty'])."',
15                  specialty = '" . str_replace( search: "'", replace: "\'", $group['specialty'])."',
16                  studentsQuantity = ".$group['studentsQuantity']."
17                  where id = " . $group['id']);
18      }
```

Рис. 3. Метод editGroup класу Model.

```
5       private $protectedActions = ['get-groups', 'edit_group'];
```

```
25          case 'edit_group':
26              if (Model::editGroup($this->data)) {
27                  $res = ['update' => 'success'];
28              } else {
29                  $res = ['error' => 'groups update error !'];
30              }
31              break;
```

Рис. 4. Доповнення методу run() класу контролера API.

До сервісу data-getter.service.ts застосунку додаємо метод editGroup, що взаємодіятиме із API (рис. 5).

```
47
48      editGroup(group) {
49        return this.http.post<any>(
50            url: this.baseUrl + '?action=edit_group&token=' + this.token,
51            group
52        );
53      }
```

Рис. 5. Додавання методу до сервісу data-getter.service.ts.

У компоненті stud-group також реалізуємо метод збереження змін (рис. 6).

```
16
17      constructor(private dataGetter: DataGetterService) {}
```

```
43
44      saveGroup() {
45        this.dataGetter.editGroup(this.studentGroup).subscribe(
46            next: data => console.log(data)
47        );
48      }
```

Рис. 6. Додавання методу до компоненти stud-group.

І нарешті додаємо до представлення компоненти stud-group кнопку, за якою будемо виконувати збереження даних (рис. 7).

```
28        </ion-button>
29        <ion-button *ngIf="!isNew" color="primary" (click)="saveGroup()">
30          <ion-icon slot="start" name="save"></ion-icon> Зберегти
31        </ion-button>
32      </ion-card-content>
33    </ion-card>
```

Рис. 7. Кнопка збереження даних у stud-group.component.html.

Перевіряємо роботу перегляду та редагування даних студентської групи. Паралельно пересвідчуємося у тому, що дані змінилися у БД MySQL.

Далі переходимо до реалізації функціоналу додавання та видалення даних. Спочатку реалізуємо відповідний функціонал на back-end (рис. 8-9).

```
19
20      public static function addGroup($group) {
21          return (new DB())->runQuery(
22              sql: "insert into groups(number, faculty, specialty,
    studentsQuantity)
23                      values('".$group['number']."', '" . str_replace( search: "'",
     replace: "\'", $group['faculty'])."',
24                      '" . str_replace( search: "'", replace: "\'", $group['specialty'])
    ."',
25                      ".$group['studentsQuantity'].")");
26      }
27
28      public static function removeGroup($group) {
29          return (new DB())->runQuery(
30              sql: "delete from groups
31                  where id = " . $group['id']);
32      }
```

Рис. 8. Доповнення класу моделі API.

```
5       private $protectedActions = ['get-groups', 'edit_group', 'add_group',
    'del_group'];
6
32              case 'add_group':
33                  if (Model::addGroup($this->data)) {
34                      $res = ['insert' => 'success'];
35                  } else {
36                      $res = ['error' => 'groups insert error !'];
37                  }
38                  break;
39              case 'del_group':
40                  if (Model::removeGroup($this->data)) {
41                      $res = ['delete' => 'success'];
42                  } else {
43                      $res = ['error' => 'group delete error !'];
44                  }
45                  break;
```

Рис. 9. Доповнення контролеру API.

Реалізовуємо необхідні методи у сервісі data-getter.service.ts (рис. 10).

```
54
55      addGroup(group) {
56        return this.http.post<any>(
57            url: this.baseUrl + '?action=add_group&token=' + this.token,
58            group
59        );
60      }
```

```
61
62    delGroup(group) {
63      return this.http.post<any>(
64          url: this.baseUrl + '?action=del_group&token=' + this.token,
65          group
66      );
67    }
```

Рис. 10. Методи addGroup та delGroup сервісу data-getter.

У класі Home також корегуємо методи додавання та видалення елементу (рис. 11).

```
43
44    delete(group) {
45      this.dataGetter.delGroup(group).subscribe(
46          next: res => this.dataGetter.getGroups().subscribe(
47              next: data => this.groups = data
48          )
49      );
50    }
51
52    addGroup(group) {
53      this.dataGetter.addGroup(group).subscribe(
54          next: res => this.dataGetter.getGroups().subscribe(
55              next: data => this.groups = data
56          )
57      );
58      this.showNew = false;
59    }
```

Рис. 11. Методи addGroup та delete у класі home.page.ts.

Також незначних змін потребує представлення home.page.html – у якості параметру методу delete передаємо не індекс масиву, а елемент (рис. 12).

```
37          <ion-item-option color="danger" (click)="delete(group)">
38            <ion-icon name="trash"></ion-icon>
39            Видалити
40          </ion-item-option>
```

Рис. 12. Зміни у home.page.html.

Перевіряємо роботу додавання та видалення даних. Пересвідчитись у зміні даних таблиці groups БД MySQL після виконання відповідних дій.

На остаток виконаємо реалізацію отримання списку студентів певної групи. Для можливості отримання списку студентів за id групи виконаємо передачу ідентифікатору групи при переході за посиланням із сторінки home (рис. 13).

```
<ion-item-options side="end">
  <ion-item-option color="secondary"
                   [routerLink]="['/students/'+group.number, {id: group.id}]"
                   routerDirection="forward">
    <ion-icon name="reorder"></ion-icon>
    Студенти
  </ion-item-option>
</ion-item-options>
```

Рис. 13. Зміна посилання переходу до списку студентів.

На back-end виконаємо доробку моделі – додаємо метод отримання студентів певної групи (рис. 14) та доробку контролеру – обробка відповідного action та виклик доданого до моделі методу (рис. 15).

```
public static function getStudents($groupId) {
    return (new DB())->getArrFromQuery(
        sql: "SELECT id, name, gender, rating
        FROM `students`
        where group_id=$groupId order by name");
}
```

Рис. 15. Back-end API – зміни у model.php.

```
private $protectedActions = ['get-groups', 'edit_group', 'add_group',
'del_group', 'get-students'];
```

```
case 'get-students':
    $res = Model::getStudents($_GET['group']);
    break;
```

Рис. 16. Back-end API – зміни у controller.php.

У сервісі отримання даних від back-end DataGetterService додаємо метод для отримання списку студентських групи (рис. 17).

```
getStudents(id: number) {
  return this.http.get<any>(
      url: this.baseUrl + `?action=get-students&group=${id}&token=$
{this.token}`
  );
}
```

Рис. 17. Метод getStudents сервісу DataGetterService.

У класі сторінки списку студентів отримуємо список студентів відповідно до доданого у DataGetterService методу getStudents (рис. 18).

```
import {ActivatedRoute, Router} from '@angular/router';
```

```
export class StudentsPage implements OnInit {
  grpid: number;
  grpnumb: string;
  students: any[];
```

```
15      constructor(private dataGetter: DataGetterService,
16                  private route: ActivatedRoute,
17                  private router: Router) {
18        this.grpid = +this.route.snapshot.paramMap.get('id');
19      }
20
21      ngOnInit() {
22        this.grpnumb = this.route.snapshot.paramMap.get('grpnumb');
23        this.dataGetter.getStudents(this.grpid).subscribe(
24            next: data => {
25              this.students = data;
26            }
27        );
28      }
29  }
```

Рис. 18. Зміни у students.page.ts.

Перевіряємо роботу списку студентських груп, все має працювати, як і до впровадження back-end API.

Завдання для самостійного виконання.

Реалізувати функціонал для збереження даних власного застосунку на сервері у реляційній БД та механізми взаємодії із ними (має бути реалізовано читання, додавання, зміна та видалення даних мінімум 2-х зв’язаних таблиць). Реалізувати аутентифікацію користувачів (список із паролями мають зберігатись на сервері) та контроль при спробі отримання неавторизованими користувачами захищених даних.

При виведенні даних хоча б однієї таблиці реалізувати механізми оновлення даних та порційного завантаження (refresher та infinite-scroll).

Додатково. Реалізувати декілька ролей користувачів (перегляд, редагування, адміністрування користувачів) та адмін-панель.