Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили

Дворецький М. Л., Журавська І. М.,
Дворецька С. В., Боровльова С. Ю.

РОЗРОБКА ГІБРИДНИХ ЗАСТОСУНКІВ ДЛЯ ЦИФРОВОЇ ТРАНСФОРМАЦІЇ БІЗНЕСУ

Навчальний посібник

Для підготовки здобувачів вищої освіти
в галузі знань 12 «Інформаційні технології»

ЧНУ
ВИДАВНИЦТВО
Миколаїв – 2022

УДК 004.4:378
Р 62

Рекомендовано до друку вченою радою Чорноморського національного університету імені Петра Могили (протокол № 9 від 13 жовтня 2021 р.).

Рецензенти:
Субботін С. О., д-р техн. наук, професор, завідувач кафедри програмних засобів Національного університету «Запорізька політехніка».
Котлик С. В., канд. техн. наук, доцент, директор Навчально-наукового інституту комп'ютерних систем і технологій «Індустрія 4.0» Одеського національного технологічного університету.

Р 62 Розробка гібридних застосунків для цифрової трансформації бізнесу : навч. посібник / М. Л. Дворецький, І. М. Журавська, С. В. Дворецька, С. Ю. Боровльова. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2022. – 160 с.

ISBN 978-966-336-435-3

Навчальний посібник присвячений розробці гібридних односторінкових вебзастосунків для цифрової трансформації бізнесу у різних галузях. Використання адаптивних вебінтерфейсів допомагатиме організаціям тісніше об'єднувати дані, пристрої та персонал, сприятиме створенню нових способів обслуговування клієнтів і розширення співпраці ключових сегментів бізнесу: B2B, B2C, B2G. Розглянуто створення застосунків, що функціонують на мобільних платформах Android та Apple iOS. Наведені завдання для індивідуального виконання. Посібник призначений для підготовки здобувачів вищої освіти в галузі знань 12 «Інформаційні технології», а також може бути корисний студентам споріднених спеціальностей.

УДК 004.4:378

ISBN 978-966-336-435-3

ЗМІСТ

ВСТУП

Одне із питань, що виникають на початку розробки мобільного застосунку, – це створювати мобільний (адаптивний) вебсайт, нативний або гібридний застосунок. Від прийнятого рішення буде залежати подальший процес розробки.

Адаптивний вебдизайн – дизайн вебсторінок, що забезпечує правильне відображення сайту на різних пристроях, підключених до інтернету, і динамічно підлаштовується під задані розміри вікна браузера. Метою адаптивного вебдизайну є універсальність відображення вмісту вебсайту для різних пристроїв. Для того, щоб вебсайт було зручно переглядати з різних пристроїв і з екранами різної роздільювальної здатності, за технологією адаптивного вебдизайну не потрібно створювати окремі версії вебсайту для окремих видів пристроїв.

Нативні (рідні) застосунки написані мовою програмування, специфічною для платформи, задля якої вони розробляються. Зазвичай це Objective-C або Swift для iOS та Java або Kotlin для Android. Нативні застосунки зазвичай мають кращу продуктивність під час рендерінгу й анімації, ніж гібридні програми. Але якщо вам необхідна реалізація як для Android, так і для iOS – необхідно буде створити два застосунки.

Гібридний застосунок – це мобільний застосунок, який містить вебпредставлення (по суті, ізольований екземпляр браузера) для запуску вебзастосунку із використанням вбудованої оболонки, яка може взаємодіяти із платформою пристрою та вебпредставленням. Це означає, що вебзастосунки можуть працювати на мобільному пристрої, маючи доступ до, наприклад, камери або функцій GPS.

Гібридні застосунки можливі завдяки створеним інструментам, які полегшують зв'язок між вебпредставленням і платформою. Ці інструменти не є частиною офіційних платформ iOS або Android, а є сторонніми інструментами, такими як Apache Cordova. Коли гібридний застосунок буде створено, він буде скомпільований, перетворивши вебзастосунок у нативний застосунок.

Вебзастосунки дозволяють інтернет-користувачам отримати доступ до функціоналу наданого розробниками сервісу або інструменту, використовуючи тільки браузер. Це значно економить час, оскільки програму не потрібно завантажувати і встановлювати, а отже, нею скористається набагато більше число користувачів.

Усі вебзастосунки поділяються на односторінкові (SPA) і багатосторінкові (MPA). SPA-застосунок, Single Page Application, або «застосунок однієї сторінки», – це тип вебзастосунків, в яких завантаження необхідного коду відбувається на одну сторінку, що дозволяє заощадити час на повторне завантаження одних і тих же елементів. Взаємодія з користувачем організована, як правило, через динамічно підвантажені HTML, CSS, JavaScript, зазвичай за допомогою AJAX. SPA нагадують нативні (native) застосунки, з тією лише різницею, що виконуються в рамках браузера, а не у власному процесі операційної системи Angular – це фреймворк від компанії Google для створення клієнтських застосунків, та націлений перш за все на розробку SPA-рішень. У цьому плані Angular є спадкоємцем іншого фреймворку AngularJS. У той же час, Angular – це не нова версія AngularJS, а принципово новий фреймворк.

Angular надає таку функціональність, як двостороннє зв'язування, що дозволяє динамічно змінювати дані в одному місці інтерфейсу при зміні даних моделі в іншому, шаблони, маршрутизація та багато іншого. Однією з ключових особливостей Angular є те, що він використовує як мову програмування TypeScript. Використовувати мову TypeScript необов'язково. За бажання можливо писати програми на Angular за допомогою таких мов, як Dart або JavaScript. Однак TypeScript все-таки є основною мовою для Angular.

TypeScript представляє мову програмування на основі JavaScript, TypeScript зародився в компанії Microsoft, його творцем вважається програміст Андерс Хейлсберг, так само відомий як творець мови C#. Спробуємо розібратися, навіщо потрібна ще одна мова програмування для клієнтської сторони у вебсередовищі, якщо можна використати традиційний JavaScript, яким володіє багато розробників і підтримка якого у співтоваристві програмістів досить висока.

По-перше, TypeScript – це строго типізована і компільована мова, чим, можливо, буде ближче до програмістів Java, C#. Строга типізація зменшує кількість потенційних помилок, які могли б виникнути у розробці на JavaScript.

По-друге, TypeScript реалізує багато концепції, які властиві об'єктно-орієнтованим мовам, наприклад, наслідування, поліморфізм, інкапсуляція, модифікатори доступу та ін.

По-третє, потенціал TypeScript дозволяє швидше і простіше писати великі складні комплексні програми, відповідно їх легше підтримувати, розвивати, масштабувати і тестувати, ніж на стандартному JavaScript.

У той же час, TypeScript є надмножиною над JavaScript, а це значить, що будь-яка програма на JS є програмою на TypeScript. У TS можна використовувати всі ті конструкції, які застосовуються в JS – ті ж оператори, умовні та циклічні конструкції. Більш того, код TS компілюється в JavaScript. Зрештою, TS – це всього лише інструмент, який покликаний полегшити розробку застосунків, а згенерований компілятором TypeScript код JS підтримується переважною більшістю браузерів.

Посібник складається зі вступу, чотирьох розділів та переліку індивідуальних завдань.

У першому розділі розглянуто створення адаптивних вебінтерфейсів на основі таких популярних CSS та JavaScript бібліотек, як twitter bootstrap, bulma та materialize. Також наводиться приклад використання компонент фреймворку ionic.

Другий розділ присвячений розробці односторінкових вебзастосунків із використанням JS фреймворку Angular. Розглядаються робота із компонентами, створення власних компонент, події та директиви, передача та повернення даних, поняття двостороннього зв'язування, а також використання сервісів та роутінгу.

Третій розділ розглядає використання Ionic Framework для створення гібридних застосунків, у тому числі для використання на мобільному пристрої.

У четвертому розділі виконано реалізацію back-end складової, організацію збереження даних на віддаленому сервері та взаємодію із ним.

РОЗДІЛ 1.
РЕАЛІЗАЦІЯ АДАПТИВНИХ ВЕБІНТЕРФЙСІВ

1.1. Використання twitter bootstrap

Розглянемо приклад реалізації мобільного вебінтерфейсу для застосунку роботи із списком студентських груп на базі twitter bootstrap. Створимо index.html із стандартною html5 розміткою (рис. 1.1).

index.html > ...

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <title>Document</title>
6   </head>
7   <body>
8
9   </body>
10  </html>
```

Рисунок 1.1 – Створення index.html

Потім виконаємо підключення бібліотеки bootstrap. Зробити це можна декількома способами, один із яких полягає у завантаженні необхідних файлів бібліотек локально у директорію assets (рис. 1.2). Нагадаємо, що для коректної роботи скриптів bootstrap.js необхідно попередньо підключити також бібліотеки jquery.js та popper.js. Необхідні файли можуть бути завантажені на офіційних сайтах jquery, popper та bootstrap.

```
3   <head>
4       <meta charset="UTF-8">
5       <title>Document</title>
6       <link rel="stylesheet" href="assets/bootstrap.min.css">
7       <script src="assets/jquery-3.4.1.min.js"></script>
8       <script src="assets/popper.min.js"></script>
9       <script src="assets/bootstrap.min.js"></script>
```

Рисунок 1.2 – Підключення завантажених бібліотек

Також можна скористатись cdn-посиланнями (рис. 1.3), але зазначимо, що цей підхід вимагатиме наявності постійного інтернет-з'єднання для роботи із бібліотекою. Звернімо увагу, що cdn-посилання можуть бути іншими.

```
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">
    <script
            src="https://code.jquery.com/jquery-3.4.1.min.js"
            integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo="
            crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.15.0/esm/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"></script>
```

Рисунок 1.3 – Підключення бібліотек через cdn-посилання

Далі виконаємо реалізацію обробки натискання на кнопку для перевірки правильності роботи бібліотеки. Додаємо на сторінку кнопку, із натисканням на яку буде виводитись повідомлення alert (рис. 1.4). Обробку коду натискання на кнопку розмістимо у файлі code/my.js (рис. 1.5). Також у файлі index.html виконаємо підключення цього скрипта (рис. 1.4).

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <link rel="stylesheet" href="assets/bootstrap.min.css">
    <script src="assets/jquery-3.4.1.min.js"></script>
    <script src="assets/popper.min.js"></script>
    <script src="assets/bootstrap.min.js"></script>
```

```
    <script src="code/my.js"></script>
</head>
<body>
    <div class="text-center mt-5" >
        <button type="button" class="btn btn-primary center"
        id="click-me">Натисни мене !</button>
    </div>
    <div class="m-5" id="alert">
    </div>
</body>
</html>
```

Рисунок 1.4 – Додавання кнопки та підключення my.js

```
$("document").ready(function() {
    $('#click-me').click(function() {
        var html = `<div class="text-left alert alert-success" role="alert">
            <h4 class="alert-heading">Привіт !
                <button type="button" class="close" data-dismiss="alert"
                aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </h4>
            <p>Вітаємо із успішним підключенням twitter bootstrap !</p>
            <hr>
            <p>Тепер можна переходити до реалізації проекту.</p>
        </div>`;
        $('#alert').html(html);
    });
});
```

Рисунок 1.5 – Файл my.js

Перевіримо результат, завантажимо файл у браузері, для перегляду зовнішнього вигляду сторінки так, як вона виглядатиме на мобільному пристрої, запустимо режим розробника та перейдемо у режим адаптивного дизайну (рис. 1.6–1.7).

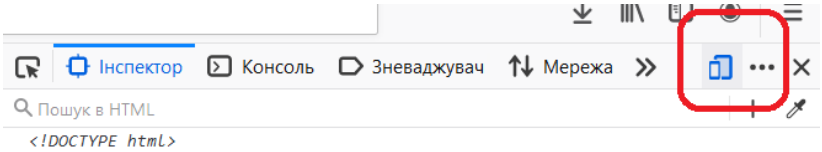

Рисунок 1.6 – Перехід до режиму адаптивного дизайну

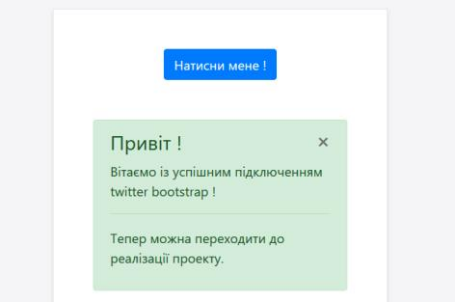

Рисунок 1.7 – Перегляд зовнішнього вигляду сторінки

Реалізуємо сторінку для виведення даних щодо списку студентських груп. Сам список будемо зберігати у LocalStorage. Почнімо зі скрипту, що буде отримувати із LocalStorage списки груп, але якщо там нічого немає – записувати туди 2 групи. Для цього у папці code створимо data.js (рис. 1.8).

```
var groups = localStorage.getItem("groups_data");
if (localStorage.getItem("groups_data") === null) {
    groups = [
        {
            number: 301,
            faculty: 'Computer science'
        },
        {
            number: 302,
            faculty: 'Computer science'
        }
    ];
    localStorage.setItem("groups_data", JSON.stringify(groups));
} else {
    groups = JSON.parse(groups);
}
```

Рисунок 1.8 – Файл data.js

Тепер підключимо цей скрипт до index.html та розмістимо у елементі body nav-панель із заголовком вікна, кнопку для додавання нової групи та таблицю для виведення існуючих груп (рис. 1.9). Також підключимо font-awesome для можливості виведенні іконок на кнопках.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <link rel="stylesheet" href="assets/bootstrap.min.css">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/
    4.7.0/css/font-awesome.min.css">
     <script src="assets/jquery-3.4.1.min.js"></script>
     <script src="assets/popper.min.js"></script>
     <script src="assets/bootstrap.min.js"></script>
     <script src="code/data.js"></script>
     <script src="code/my.js"></script>
 </head>
 <body>
    <nav class="navbar navbar-dark bg-dark text-white">
        <h3>Студентські групи</h3>
    </nav>
    <div class="container mt-3">
        <div class="row">
            <div class="col">
                <a class="btn btn-outline-success" href="#">
                    <i class="fa fa-fw fa-plus"></i>
                </a>
            </div>
        </div>
        <div class="row mt-2">
            <div class="col">
                <table class="table">
                    <thead>
                        <tr>
                            <th scope="col" style="width: 60%">Група</th>
                            <th scope="col" style="width: 40%"></th>
                        </tr>
                    </thead>
                     <tbody id="groups">
                     </tbody>
                 </table>
             </div>
         </div>
    </div>
</body>
</html>
```

Рисунок 1.9 – Зміни файлу index.html

У файлі code/my.js реалізуємо отримання даних списку студентських груп із LocalStorage та виведення їх до таблиці (рис. 1.10).

```
$('document').ready(function() {
  var rowText;
  var content = $('#groups');
  for(var row of groups) {
      rowText =
      `<tr>
          <td>${row.number}</td>
          <td class="text-right">
              <a class="btn btn-outline-secondary" href="group.html?
              number=${row.number}">
                  <i class="fa fa-fw fa-edit"></i>
              </a>
              <button type="button" class="btn btn-outline-danger rem-row"
              rowid="${row.number}">
                  <i class="fa fa-fw fa-trash"></i>
              </button>
          </td>
      </tr>`;
      content.append(rowText);
  }
});
```

Рисунок 1.10 – Скрипт my.js

Завантажимо відкориговану сторінку та переглянемо результат (рис. 1.11).

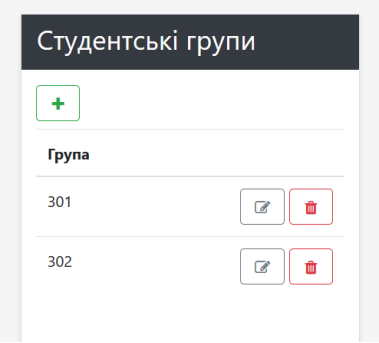

Рисунок 1.11 – Сторінка виведення студентських груп

Реалізуємо функціонал редагування цього списку. Для початку трохи змінимо файл code/data.js – виділимо функцію saveGroups для її подальшого використання під час додавання, зміни та видалення елементів списку (рис. 1.12).

```
function saveGroups(groups) {
    localStorage.setItem("groups_data", JSON.stringify(groups));
}
```

Рисунок 1.12 – Функція saveGroups в data.js

Тепер реалізуємо сторінку для додавання та редагування елементу списку group.html (рис. 1.13). Тут розмістимо html-форму для виведення елементів студентської групи та підключимо скрипт code/group.js, вміст якого буде наведено пізніше.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Студентські групи</title>
    <link rel="stylesheet" href="assets/bootstrap.min.css">
    <script src="assets/jquery-3.4.1.min.js"></script>
    <script src="assets/popper.min.js"></script>
    <script src="assets/bootstrap.min.js"></script>
    <script src="code/data.js"></script>
    <script src="code/group.js"></script>
</head>
<body>
    <nav class="navbar navbar-dark bg-dark text-white">
        <h3>Студентська група</h3>
    </nav>
    <div class="container mt-3">
        <div class="row">
            <div class="col">
                <form>
                    <div class="form-group">
                        <label for="number">Номер групи</label>
                        <input type="text" class="form-control" id="number"
                        placeholder="Номер групи">
                    </div>
                    <div class="form-group">
                        <label for="faculty">Факультет</label>
                        <input type="text" class="form-control" id="faculty"
                        placeholder="Факультет">
                    </div>
                    <button type="button" id="save" class="btn
                    btn-secondary">Зберегти</button>
                </form>
            </div>
        </div>
    </div>
</body>
</html>
```

Рисунок 1.13 – Сторінка редагування даних group.html

Тепер виконаємо невеликі зміни у index.html та code/my.js – додаємо посилання на кнопки «Додати» та «Змінити» і атрибут rowid для кнопки «Видалити» (рис. 1.14–1.15). Також у code/my.js реалізуємо обробку натискання на кнопку «Видалити».

```
        <div class="row">
            <div class="col">
                <a class="btn btn-outline-success" href="group.html">
                    <i class="fa fa-fw fa-plus"></i>
                </a>
            </div>
```

Рисунок 1.14 – Зміни у index.html

```
$('document').ready(function() {
    var rowText;
    var content = $('#groups');
    for(var row of groups) {
        rowText =
        `<tr>
            <td>${row.number}</td>
            <td class="text-right">
                <a class="btn btn-outline-secondary" href="group.html?number=${row.number}">
                    <i class="fa fa-fw fa-edit"></i>
                </a>
                <button type="button" class="btn btn-outline-danger rem-row" rowid="${row.number}">
                    <i class="fa fa-fw fa-trash"></i>
                </button>
            </td>
        </tr>`;
        content.append(rowText);
    }
    $('.rem-row').click(function() {
        let number = $(this).attr('rowid');
        saveGroups(groups.filter((g)=>g.number !== number));
        location.reload();
    })
});
```

Рисунок 1.15 – Зміни у my.js

На останок реалізуємо скрипт data/group.js, що виконуватиме додавання та редагування елементу списку (рис. 1.16). Під час редагування елементу у get-запиті передаватимемо значення number. У group.js виконуємо перевірку, якщо number передано, то виконуємо зміну елементу масиву, інакше додаємо новий елемент.

```
$('document').ready(function() {
    let searchParams = new URLSearchParams(window.location.search);
    let number = 0;
    if (searchParams.has('number')) {
        number = searchParams.get('number');
        let group = groups.find((g)=>g.number == number);
        $('#number').val(group.number);
        $('#faculty').val(group.faculty);
    }

    $('#save').click(function() {
        if (number === 0) {
            groups.push({
                number: $('#number').val(),
                faculty: $('#faculty').val()
            });
        } else {
            let group = groups.find((g)=>g.number == number);
            group.number = $('#number').val();
            group.faculty = $('#faculty').val();
        }
        saveGroups(groups);
        $(location).attr('href','index.html');
    });
});
```

Рисунок 1.16 – Файл data/group.js

Перевіряємо роботу коду, пробуємо додати змінити та видалити елемент списку (рис. 1.17).

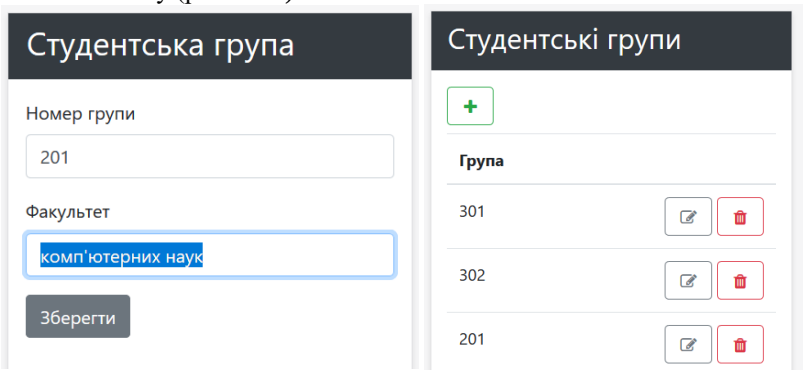

Рисунок 1.17 – Додавання студентської групи

Останній штрих – додавання метатегу initial-scale=1 для відключення масштабування сторінки на мобільному пристрої (рис. 1.18). Вигляд сторінки до та після включення тегу наведено на рис. 1.19.

```
4       <meta charset="UTF-8">
5       <meta name="viewport" content="initial-scale=1">
6       <title>Студентські групи</title>
```

Рисунок 1.18 – Додавання тегу viewport

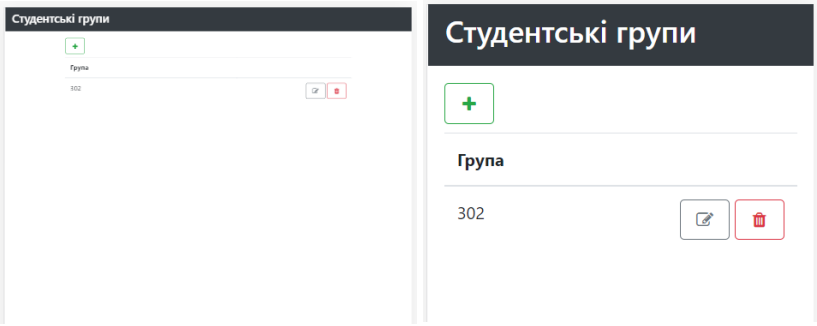

Рисунок 1.19 – Зовнішній вигляд сторінки на мобільному пристрої

1.2. Реалізація адаптивного вебінтерфейсу засобами bulma та materialize

На сьогодні частка мобільного трафіку зростає стрімкими темпами. Адаптивна верстка дозволяє підлаштовуватися основному контейнеру і будь-якому іншому елементу сайту під розділювальну здатність екрану, роблячи можливим змінювати розмір шрифту, розташування об’єктів, колір та ін. Протягом довгого часу Bootstrap був одним з найпопулярніших фреймворків для розробки адаптивних і мобільних вебпроєктів. Але є інші фреймворки, які заслуговують на нашу увагу. Виконаємо реалізацію сторінки списку студентських груп з використанням кількох альтернативних підходів.

Bulma з’явилася лише декілька років тому і відразу стала популярною. Це був один з перших CSS фреймворків, що реалізував повноцінну сітку flexbox. Крім того, в ньому є величезний набір компонентів для розробки будь-якого вебсайту. Єдиним недоліком порівняно з Bootstrap є те, що це тільки CSS-фреймворк, тут немає JavaScript. Тому доведеться написати власний код JavaScript або звернутися до Jquery для реалізації випадних списків чи інших елементів.

Далі наведемо реалізацію попереднього проєкту щодо обліку списку студентських груп із використанням Bulma. Скопіюємо проєкт попередньої роботи, почнімо зі зміни файлу index.html (рис. 1). У блоці head виконуємо підключення bulma.min.css замість bootstrap.min.css та прибираємо popper.js та bootstrap.js. У тілі сторінки body розміщуємо елемент section, у якому також будуть присутні nav та table, але класи елементів будуть трохи іншими відповідно до вимог Bulma (рис. 1.20).

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Hello Bulma!</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.8.0/
    css/bulma.min.css">
    <script defer src="https://use.fontawesome.com/releases/v5.3.1/js/
    all.js"></script>
    <script src="assets/jquery-3.4.1.min.js"></script>
    <script src="code/data.js"></script>
    <script src="code/my.js"></script>
  </head>
  <body>
  <section class="section">
    <div class="container">
        <nav class="navbar is-fixed-top has-background-dark">
            <h3 class="title has-text-white has-text-centered">
                Студентські групи
            </h3>
        </nav>
        <div style="margin: 20px;"></div>
        <a class="button is-primary" href="group.html"><i class="fa fa-fw
        fa-plus"></i></a>
        <div style="margin: 10px;"></div>
        <table class="table is-fullwidth has-margin-top-2">
            <thead>
                <tr>
                    <th>Група</th>
                    <th></th>
                </tr>
            </thead>
            <tbody id="groups">
            </tbody>
        </table>
    </div>
  </section>
  </body>
</html>
```

Рисунок 1.20 – Реалізація index.html

Також змінимо частину my.js, що відповідає за наповнення таблиці даними, а саме відкоригуємо класи елементів відповідно до вимог фреймворку Bulma (рис. 1.21).

```
$('document').ready(function() {
    var rowText;
    var content = $('#groups');
    for(var row of groups) {
        rowText =
        `<tr>
            <td>${row.number}</td>
            <td class="has-text-right">
                <a href="group.html?number=${row.number}" class="button
                is-link">
                    <i class="fa fa-fw fa-edit"></i>
                </a>
                <button type="button" class="button is-danger rem-row"
                rowid="${row.number}">
                    <i class="fa fa-fw fa-trash"></i>
                </button>
            </td>
        </tr>`;
        content.append(rowText);
    }
    $('.rem-row').click(function() {
        let number = $(this).attr('rowid');
        saveGroups(groups.filter((g)=>g.number != number));
        location.reload();
    })
});
```

Рисунок 1.21 – Корегування файлу my.js

Далі наведемо код адаптованого під bulma файлу group.html (рис. 1.22).

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Hello Bulma!</title>
        <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/
        bulma@0.8.0/css/bulma.min.css">
        <script defer src="https://use.fontawesome.com/releases/v5.3.1/js/
        all.js"></script>
```

```
9         <script src="assets/jquery-3.4.1.min.js"></script>
10        <script src="code/data.js"></script>
11        <script src="code/group.js"></script>
12    </head>
13    <body>
14        <section class="section">
15            <div class="container">
16                <nav class="navbar is-fixed-top has-background-dark">
17                    <h3 class="title has-text-white has-text-centered">
18                        Студентська група
19                    </h3>
20                </nav>
21                <div style="margin: 20px;"></div>
22                <form>
23                    <div class="field is-horizontal">
24                        <div class="field-label is-normal">
25                            <label class="label">Номер групи</label>
26                        </div>
27                        <div class="field-body">
28                            <div class="field">
29                            <p class="control">
30                                <input id="number" class="input"
                                  type="email" placeholder="Номер групи">
31                            </p>
32                            </div>
33                        </div>
34                    </div>
35                    <div class="field is-horizontal">
36                        <div class="field-label is-normal">
37                            <label class="label">Факультет</label>
38                        </div>
39                        <div class="field-body">
40                            <div class="field">
41                            <p class="control">
42                                <input id="faculty" class="input"
                                  type="email" placeholder="Факультет">
43                            </p>
44                            </div>
45                        </div>
46                    </div>
47                    <button type="button" class="button is-primary"
                      id="save">Зберегти</button>
48                </form>
49            </div>
50        </section>
51    </body>
52</html>
```

Рисунок. 1.22 – Файл group.html

Збережемо зміни та перевіримо результат (рис. 1.23).

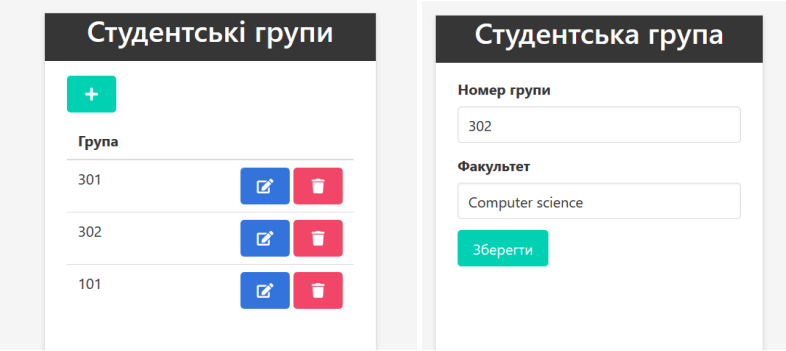

Рисунок 1.23 – Зовнішній вигляд сторінок, реалізованих на базі bulma

Materialize – сучасний адаптивний front-end фреймворк, заснований на принципах Material Design компанії Google. Material design – це мова дизайну, створена Google, яка поєднує в собі традиційні методи дизайну з інноваціями. Ця структура ідеально підходить для розробників, які хочуть долучити Material Design до свого вебсайту без складного коду. Він пропонує card design, Sass mixins, drag-out mobile menu, ripple effect анімації і багато чого ще. Розробники платформи надають приклади коду і детальну документацію, щоб допомогти новим користувачам почати роботу з платформою.

Реалізуємо наш інтерфейс на базі Materialize. Почнімо зі змін у головному файлі – index.html. Завантажимо (https://materializecss.com/getting-started.html) та підключимо необхідні бібліотеки і виконаємо стилізацію елементів відповідно до стандартів matezialize.css (рис. 1.24).

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="initial-scale=1">
    <title>Студентські групи</title>
    <link rel="stylesheet" href="assets/materialize.min.css">
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    <script src="assets/jquery-3.4.1.min.js"></script>
    <script src="assets/materialize.min.js"></script>
    <script src="code/data.js"></script>
    <script src="code/my.js"></script>
</head>
<body>
    <nav>
        <div class="nav-wrapper">
```

```
            <a class="brand-logo">Групи</a>
        </div>
      </nav>
    <div class="container">
        <div class="row" style="margin-top: 20px;">
            <div class="col m6 offset-m3 s12">
                <a href="group.html" class="btn-floating btn-large waves-effect waves-light
                green">
                    <i class="material-icons">add</i>
                </a>
            </div>
        </div>
        <div class="row">
            <div class="col m6 offset-m3 s12 no-padding">
                <table class="striped">
                    <thead>
                        <tr>
                            <th style="width: 60%">Група</th>
                            <th style="width: 40%"></th>
                        </tr>
                    </thead>
                    <tbody id="groups">
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</body>
</html>
```

Рисунок 1.24 – Файл index.html

Також необхідно змінити стилізацію рядків, що заповнюються до таблиці, за допомогою javascript у файлі my.js (рис. 1.25).

```
$('document').ready(function() {
    var rowText;
    var content = $('#groups');
    for(var row of groups) {
        rowText =
        `<tr>
            <td><h5>${row.number}</h5></td>
            <td class="right-align">
            <a href="group.html?number=${row.number}" class="btn-floating waves-effect
            waves-light blue" style="margin-right:10px;">
                <i class="material-icons">edit</i>
            </a>
            <a rowid="${row.number}" class="rem-row btn-floating waves-effect waves-light red">
                <i class="material-icons">delete</i>
            </a>
            </td>
        </tr>`;
        content.append(rowText);
    }
    $('.rem-row').click(function() {
        let number = $(this).attr('rowid');
        saveGroups(groups.filter((g)=>g.number != number));
        location.reload();
    })
});
```

Рисунок 1.25 – Файл my.js

Аналогічно змінимо файли group.html та group.js для відображення та заповнення даних другої сторінки застосунку (рис. 1.26–1.27).

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="initial-scale=1">
    <title>Студентські групи</title>
    <link rel="stylesheet" href="assets/materialize.min.css">
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    <script src="assets/jquery-3.4.1.min.js"></script>
    <script src="assets/materialize.min.js"></script>
    <script src="code/data.js"></script>
    <script src="code/group.js"></script>
</head>
<body>
    <nav>
        <div class="nav-wrapper">
            <a class="brand-logo">Група</a>
        </div>
    </nav>
    <div class="container">
        <div class="row">
            <form>
                <div class="input-field col s12">
                    <input id="number" type="text" class="validate">
                    <label for="number">Номер групи</label>
                </div>
                <div class="input-field col s12">
                    <input id="faculty" type="text" class="validate">
                    <label for="faculty">Факультет</label>
                </div>
                <div class="input-field col s12 right-align">
                    <a id="save" class="btn-floating btn-large waves-effect waves-light green">
                        <i class="material-icons">done</i>
                    </a>
                </div>

            </form>
        </div>
    </div>
</body>
</html>
```

Рисунок 1.26 – Файл group.html

```javascript
$('document').ready(function() {
    let searchParams = new URLSearchParams(window.location.search);
    let number = 0;
    if (searchParams.has('number')) {
        number = searchParams.get('number');
        let group = groups.find((g)=>g.number == number);
        $('#number').val(group.number);
        $('#faculty').val(group.faculty);
```

```
        $('label').addClass('active');
    }

    $('#save').click(function() {
        if (number === 0) {
            groups.push({
                number: $('#number').val(),
                faculty: $('#faculty').val()
            });
        } else {
            let group = groups.find((g)=>g.number == number);
            group.number = $('#number').val();
            group.faculty = $('#faculty').val();
        }
        saveGroups(groups);
        $(location).attr('href','index.html');
    });
});
```

Рисунок 1.27 – Файл group.js

Переглянемо зовнішній вигляд застосунку (рис. 1.28).

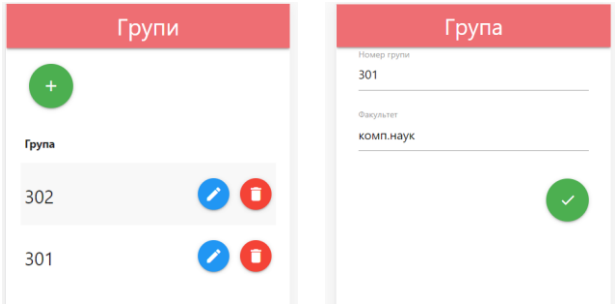

Рисунок 1.28 – Вигляд застосунку із використанням materialize.css

Завантажимо наш застосунок в інтернет та переглянемо його зовнішній вигляд на емуляторі мобільного пристрою (рисунок 1.29).

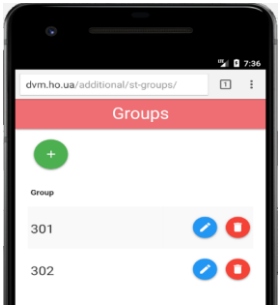

Рисунок 1.29 – Зовнішній вигляд застосунку на емуляторі

1.3. Знайомство з Ionic Framework

Ionic – це завершений SDK з відкритим кодом для розробки гібридних мобільних застосунків. Оригінальна версія була побудована на вершині AngularJS та Apache Cordova. Однак остання версія була випущена як набір вебкомпонентів, що дозволяють користувачеві вибирати будь-який фреймворк розробки інтерфейсу користувача, наприклад Angular, React або Vue.js. Також можливо використовувати компоненти Ionic окремо, без використання жодного фреймворку.

Ionic надає інструменти та сервіси для розробки гібридних мобільних, настільних та прогресивних вебзастосунків на основі сучасних технологій веброзробки, використовуючи такі вебтехнології, як CSS, HTML5 та SASS. Зокрема, мобільні застосунки можуть бути побудовані за допомогою даних вебтехнологій, а потім встановлені на пристрої, використовуючи Cordova або Capacitor.

У цій роботі розглянемо використання компонентів Ionic для створення застосунку обліку студентських груп. Застосунок буде представлений у вигляді вебсторінки, тому встановлення фреймворку поки не обов'язкове. Виконаємо підключення необхідних компонент через cdn-посилання (рис. 1.30).

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <script type="module" src="https://cdn.jsdelivr.net/npm/@ionic/core/dist/ionic/
       ionic.esm.js"></script>
8      <script nomodule src="https://cdn.jsdelivr.net/npm/@ionic/core/dist/ionic/ionic.js"></
       script>
9      <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/@ionic/core/css/
       ionic.bundle.css"/>
10     <title>Students groups</title>
11 </head>
```

Рисунок 1.30 – Підключення ionic через cdn

Далі реалізуємо зовнішній вигляд головної сторінки, додавши у тіло header та сітку із елементом «Список» (рис. 1.31).

```
<body>
    <ion-header>
        <ion-toolbar>
            <ion-title>Groups</ion-title>
        </ion-toolbar>
    </ion-header>
    <section id="main">
        <ion-grid>
            <ion-row>
              <ion-col size="12" size-md="6" offset-md="3">
                <ion-button color="success" size="large">
                    <ion-icon name="add-circle-outline"></ion-icon>
                </ion-button>
              </ion-col>
            </ion-row>
            <ion-row>
                <ion-col size="12" size-md="6" offset-md="3">
                    <ion-list>
                        <ion-list-header>
                            <ion-label>Group</ion-label>
                        </ion-list-header>
                        <ion-item>
                            <ion-label>301</ion-label>
                            <ion-button color="primary" size="medium">
                                <ion-icon name="create"></ion-icon>
                            </ion-button>
                            <ion-button color="danger" size="medium">
                                <ion-icon name="trash"></ion-icon>
                            </ion-button>
                        </ion-item>
                    </ion-list>
                </ion-col>
              </ion-row>
          </ion-grid>
    </section>
</body>
</html>
```

Рисунок 1.31 – Шаблон головної сторінки

Завантажимо застосунок у браузері та переглянемо зовнішній вигляд сторінки (рис. 1.32).

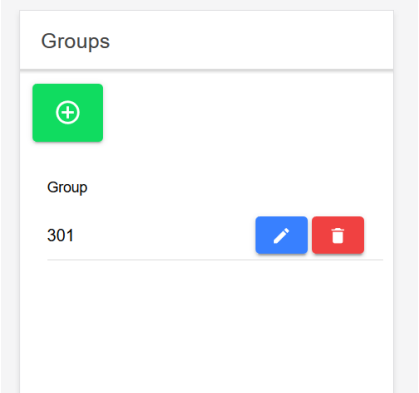

Рисунок 1.32 – Зовнішній вигляд головної сторінки

Цього разу реалізуємо наш застосунок за принципом односторінкового застосунку (SPA), тому форму додавання та редагування елементу реалізуємо в окремій секції файлу index.html (рис. 1.33).

```
<section id="edit">
    <ion-grid>
        <ion-row>
          <ion-col size="12" size-md="6" offset-md="3">
                <ion-item>
                    <ion-label position="floating">Group number</ion-label>
                    <ion-input id="number"></ion-input>
                </ion-item>
                <ion-item>
                    <ion-label position="floating">Faculty</ion-label>
                    <ion-input id="faculty"></ion-input>
                </ion-item>
                <ion-button id="save" color="primary" size="medium" float-right
                style="margin-top: 20px;">
                    <ion-icon name="checkmark"></ion-icon>
                </ion-button>
          </ion-col>
        </ion-row>
    </ion-grid>
</section>
```

Рисунок 1.33 – Шаблон сторінки редагування елементу

Зовнішній вигляд форми редагування наведено на рис. 1.34.

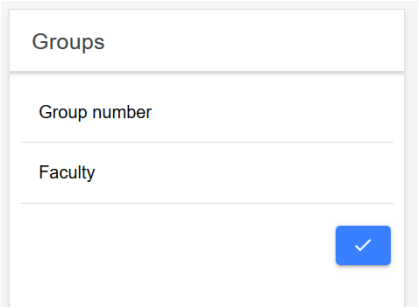

Рисунок 1.34 – Зовнішній вигляд форми редагування

Тепер реалізуємо js-код для представлення даних та обробки дій користувача. Файл data.js порівняно з попередніми роботами фактично залишається без змін (рис. 1.35).

```
function saveGroups(groups) {
    localStorage.setItem("groups_data", JSON.stringify(groups));
}

var groups = localStorage.getItem("groups_data");
if (localStorage.getItem("groups_data") === null) {
    groups = [
        {
            number: 301,
            faculty: 'Computer science'
        },
        {
            number: 302,
            faculty: 'Computer science'
        }
    ];
    saveGroups(groups);
} else {
    groups = JSON.parse(groups);
}
```

Рисунок 1.35 – Файл data.js

Далі наведемо реалізацію обробки виводу даних та дій користувача у файлі my.js. Під час реалізації функціоналу не будемо використовувати додаткових js бібліотек. На початку у процесі завантаження документа будемо виводити список груп до елементу ion-list. Також сховаємо

секцію редагування даних. Додаємо події для кнопок додавання, зміни та видалення даних (рис. 1.36).

```
let editType = '';
let number = '';
document.addEventListener("DOMContentLoaded", function(event) {
    var rowText;
    var list = document.getElementById('list');

    for(var row of groups) {
        var rowText = document.createElement('ion-item');
        rowText.innerHTML = `<ion-label>${row.number}</ion-label>
                    <ion-button class="edit" color="primary" size="medium" rowid="${row.number}
                    ">
                        <ion-icon name="create"></ion-icon>
                    </ion-button>
                    <ion-button class="delete" color="danger" size="medium" rowid="$
                    {row.number}">
                        <ion-icon name="trash"></ion-icon>
                    </ion-button>`;
        list.appendChild(rowText);
    }

    document.querySelector('#main').style.display = "";
    document.querySelector('#edit').style.display = 'none';

    document.querySelectorAll('.edit')
      .forEach(input => input.addEventListener('click', ({target}) => {
        editType = "edit";
        number = target.getAttribute('rowid');
        let group = groups.find((g)=>g.number == number);

        document.getElementById('number').value = group.number;
        document.getElementById('faculty').value = group.faculty;
        document.querySelector('#main').style.display = "none";
        document.querySelector('#edit').style.display = "";
    }));

    document.querySelectorAll('.delete')
      .forEach(input => input.addEventListener('click', ({target}) => {
        number = target.getAttribute('rowid');
        saveGroups(groups.filter((g)=>g.number != number));
        location.reload();
    }));

    document.querySelector('#save').addEventListener('click', () => {
        if (editType == "add") {
            groups.push({
                number: document.getElementById('number').value,
                faculty: document.getElementById('faculty').value
            });
        } else {
            let group = groups.find((g)=>g.number == number);
            group.number = document.getElementById('number').value;
            group.faculty = document.getElementById('faculty').value;
        }
        saveGroups(groups);
        location.reload();
    });

```

```
    document.querySelector('#add').addEventListener('click', () => {
        document.querySelector('#main').style.display = "none";
        document.querySelector('#edit').style.display = "";
        document.getElementById('number').value = "";
        document.getElementById('faculty').value = "";
        editType = "add";
    });
});
```

Рисунок 1.36 – Файл my.js

Повернімось до файлу index.html та виконаємо невеликі зміни. Підключимо створені data.js та my.js, видалимо тестовий рядок із групою 301, додамо ідентифікатори та класи для декількох елементів для можливості подальшого звернення до них із коду js. Повний код index.html після вказаних змін наведено на рис. 1.37.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <script type="module" src="https://cdn.jsdelivr.net/npm/@ionic/core/dist/ionic/
    ionic.esm.js"></script>
    <script nomodule src="https://cdn.jsdelivr.net/npm/@ionic/core/dist/ionic/ionic.js"></
    script>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/@ionic/core/css/
    ionic.bundle.css"/>
    <script src="data.js"></script>
    <script src="my.js"></script>
    <title>Students groups</title>
</head>
<body>
    <ion-header>
        <ion-toolbar>
            <ion-title>Groups</ion-title>
        </ion-toolbar>
    </ion-header>
    <section id="main">
        <ion-grid>
            <ion-row>
              <ion-col size="12" size-md="6" offset-md="3">
                <ion-button id="add" color="success" size="large">
                    <ion-icon name="add-circle-outline"></ion-icon>
                </ion-button>
              </ion-col>
            </ion-row>
            <ion-row>
                <ion-col size="12" size-md="6" offset-md="3">
                    <ion-list id="list">
                        <ion-list-header>
                            <ion-label>Group</ion-label>
                        </ion-list-header>
                    </ion-list>
                </ion-col>
            </ion-row>
        </ion-grid>
    </section>
```

```
    <section id="edit">
        <ion-grid>
            <ion-row>
              <ion-col size="12" size-md="6" offset-md="3">
                    <ion-item>
                        <ion-label position="floating">Group number</ion-label>
                        <ion-input id="number"></ion-input>
                    </ion-item>
                    <ion-item>
                        <ion-label position="floating">Faculty</ion-label>
                        <ion-input id="faculty"></ion-input>
                    </ion-item>
                    <ion-button id="save" color="primary" size="medium" float-right
                    style="margin-top: 20px;">
                        <ion-icon name="checkmark"></ion-icon>
                    </ion-button>
              </ion-col>
            </ion-row>
        </ion-grid>
    </section>
</body>
</html>
```

Рисунок 1.37 – Index.html після виконання змін

На рис. 1.38 наведемо вигляд користувацького інтерфейсу отриманого застосунку.

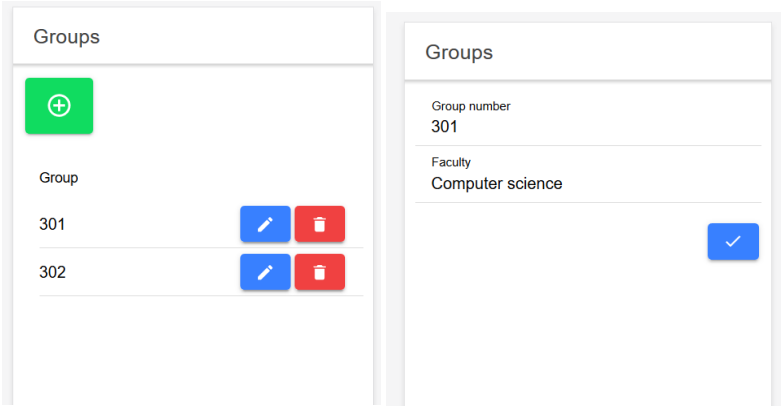

Рисунок 1.38 – Головна сторінка та форма редагування елементу

Як бачимо, тепер, з утратою активністю видимості, таймер зупиняється, а продовжує свою роботу з поверненням активності на екран користувача.

РОЗДІЛ 2.
РОЗРОБКА ОДНОСТОРІНКОВИХ ВЕБЗАСТОСУНКІВ НА БАЗІ ANGULAR FRAMEWORK

2.1. Поняття SPA та знайомство з Angular

Для того, щоб почати роботу із Angular, необхідно попередньо виконати деякі налаштування середовища розробки. Для цього встановимо node.js та менеджер пакетів npm.

Node або Node.js – програмна платформа, заснована на движку V8 (здійснює трансляцію JavaScript у машинний код), що перетворює JavaScript з вузькоспеціалізованої мови в мову загального призначення. Node.js додає можливість JavaScript взаємодіяти з пристроями вводу-виводу, підключати інші зовнішні бібліотеки, написані різними мовами, забезпечуючи їх виклики з JavaScript-коду. Node.js застосовується переважно на сервері, виконуючи роль вебсервера, але є можливість розробляти на Node.js і десктопні віконні застосунки. В основі Node.js лежить подієво-орієнтоване і асинхронне (або реактивне) програмування з неблокуючим введенням. Для установки node.js перейдемо на офіційний сайт https://nodejs.org/uk/ (рис. 2.1), завантажимо дистрибутив та виконаємо встановлення, користуючись підказками інсталятора.

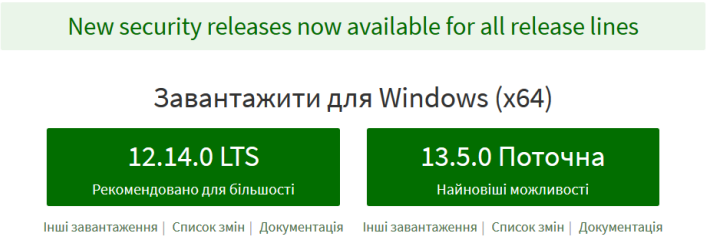

Рисунок 2.1 – Сторінка для завантаження node.js

Щоб використовувати інструменти (або пакети) у Node.js, нам потрібна можливість встановлювати і управляти ними. Для цього створено npm, пакетний менеджер Node.js. Він встановлює потрібні вам пакети і надає зручний інтерфейс для роботи з ними.

Для перевірки наявності node.js та npm запустимо командний рядок та виконаємо команди node --version та npm --version (рис. 2.2).

```
C:\>node --version
v10.15.3

C:\>npm --version
6.4.1
```

Рисунок 2.2 – Перевірка версії node та npm

Для встановлення Angular та розгортання нового проєкту завантажимо сторінку з інструкціями установки на офіційному сайті https://angular.io/ guide/setup-local (рис. 2.3).

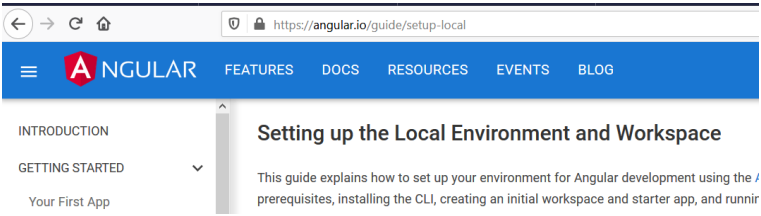

Рисунок 2.3 – Офіційний сайт проєкту Angular

Почнімо зі встановлення утиліти командного рядка Angular CLI за допомогою менеджера пакетів npm. Для цього виконаємо команду npm install -g @angular/cli (рис. 2.4).

```
C:\>npm install -g @angular/cli
C:\Users\michael\AppData\Roaming\npm\ng -> C:\Users\michael\AppData\Roamir

> @angular/cli@8.3.21 postinstall C:\Users\michael\AppData\Roaming\npm\noc
> node ./bin/postinstall/script.js

? Would you like to share anonymous usage data with the Angular Team at Go
Google's Privacy Policy at https://policies.google.com/privacy? For more d
how to change this setting, see http://angular.io/analytics. No
+ @angular/cli@8.3.21
added 251 packages from 186 contributors in 58.426s
```

Рисунок 2.4 – Встановлення Angular CLI

Для перевірки версії Angular CLI у командному рядку можемо виконати ng --version (рис. 2.5).

Рисунок 2.5 – Перевірка версії Angular CLI

Після встановлення Angular CLI можна нарешті приступати до створення нового проєкту Angular. Для цього у директорії, де плануємо створити проєкт, виконуємо команду ng new project-name (рис. 2.6). Відмовляємось від додавання роутінгу та обираємо SCSS як формат стилів.

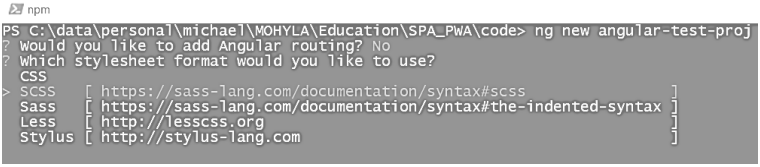

Рисунок 2.6 – Створення нового проєкту Angular

Після завершення установки відкриваємо директорію angular-test-proj у редакторі коду. У командному рядку виконаємо команду ng serve (рис. 2.7).

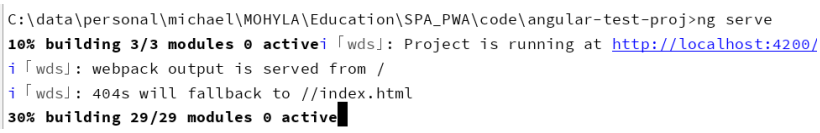

Рисунок 2.7 – Виконання проєкту

Після завершення build-у переходимо у браузері за посиланням http://localhost:4200/ та переглядаємо стандартний зовнішній вигляд сторінки після створення проєкту (рис. 2.8).

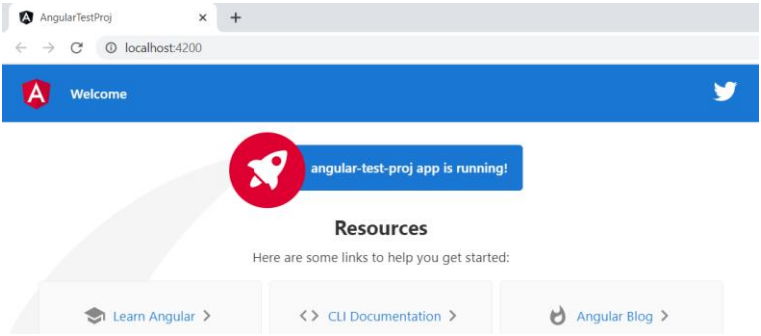

Рисунок 2.8 – Зовнішній вигляд застосунку

Відкривши файл angular.json, можемо побачити, що відправною точкою для завантаження сторінки застосунку є файл src/index.html, а точкою входу під час виконання коду – src/main.ts (рис. 2.9).

```
16        "architect": {
17          "build": {
18            "builder": "@angular-devkit/build-angular:browser",
19            "options": {
20              "outputPath": "dist/angular-test-proj",
21              "index": "src/index.html",
22              "main": "src/main.ts",
23              "polyfills": "src/polyfills.ts",
```

Рисунок 2.9 – Фрагмент файлу angular.json

Переглянемо вміст файлу src/index.html (рис. 2.10).

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>AngularTestProj</title>
6    <base href="/">
7    <meta name="viewport" content="width=device-width, initial-scale=1">
8    <link rel="icon" type="image/x-icon" href="favicon.ico">
9  </head>
10 <body>
11   <app-root></app-root>
12 </body>
13 </html>
```

Рисунок 2.10 – Файл src/index.html

Як бачимо, маємо стандартну html5-розмітку, за винятком тегу app-root. Для того, щоб зрозуміти, що він означає і звідки береться вміст

сторінки, наведеної на рис. 2.8, продовжимо інспектувати файли проєкту та переглянемо src/main.ts (рис. 2.11).

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch( onrejected: err => console.error(err));
```

Рисунок 2.11 – Файл src/main.ts

Тут нас передусім цікавить рядок bootstrapModule (AppModule), який вказує, що стартовим модулем буде AppModule. Також, із import бачимо, що він розташований у ./app/app.module. Отже, відкриємо та переглянемо файл src/app/app.module.ts (рис. 2.12).

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

import { AppComponent } from './app.component';

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

Рисунок 2.12 – Файл app.module.ts

Поки що тут нас цікавить масив declarations, у якому міститься один елемент – компонент AppComponent, що підключений із файлу ./app.component. Відкриваємо файл app.component.ts (рис. 2.13).

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  title = 'angular-test-proj';
}
```

Рисунок 2.13 – Файл app.component.ts

Бачимо стандартний клас-компонент Angular. У декораторі @Component вказано ім'я тегу, що може бути використано в html-коді, у нашому випадку (саме такий ми бачили у index.html на рис. 2.10) та ім'я файлу-шаблону, html-код якого буде виводитись у результаті вказування нашого тегу (<app-root></app-root>). Переглянемо нарешті вміст файлу app.component.html. Ми не наводимо його вміст, оскільки він досить великий і в цьому немає значного сенсу. Замість цього видалимо з нього усе і замінимо його таким кодом (рис. 2.14).

app.component.html

```
<h1>Hello world !</h1>
```

Рисунок 2.14 – Виконання змін у app.component.html

Перейдемо у браузер та переглянемо результат (рис. 2.15). Як бачимо, зовнішній вигляд сторінки змінився згідно з виконаними нами змінами у файлі app.component.html.

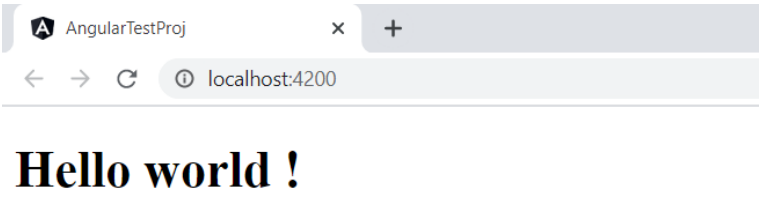

Рисунок 2.15 – Зовнішній вигляд сторінки після виконання змін

2.2. Робота із компонентами

Додаємо до проєкту власну компоненту. Це зручніше роботи, використовуючи утиліту командного рядка Angular CLI, однак перший раз зробимо це вручну. Для цього в директорії src/app створимо папку my, а у ній – файл my.component.ts (рис. 2.16).

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

@Component({
    selector: 'app-my',
    template: `<div>
        <p>This is my component</p>
    </div>`,
    styles: [`div {
        border: dashed 1px black;
    }
    p {
        margin: 15px;
    }`]
})
export class MyComponent {

}
```

Рисунок 2.16 – Створення папки та TypeScript файлу компоненти my

У файлі створюємо експортний клас MyComponent (експортний для можливості його використання за межами файлу). Далі за допомогою import підключаємо декоратор Component та визначаємо з його допомогою selector – ім'я html-тегу нашого компонента, template – шаблон для його відображення, та styles – css для стилізації його елементів. Однак цього поки що недостатньо – у результаті наведення мишею на назву класу MyComponent можемо побачити повідомлення, що цей компонент не підключено у жодному з модулів (рис. 2.17).

```
15  export class MyComponent {
16
17
```

MyComponent is not declared in any Angular module more... (Ctrl+F1)

Рисунок 2.17 – Помилка у my.component.ts

Виправимо це. Відкриємо модуль app.module.ts, додамо туди рядок імпорту MyComponent та додаємо MyComponent до масиву declarations (рис. 2.18).

```
4   import { AppComponent } from './app.component';
5   import {MyComponent} from './my/my.component';
6
7   @NgModule({
8     declarations: [
9       AppComponent, MyComponent
10    ],
11    imports: [
```

Рисунок 2.18 – Додавання MyComponent
до declarations у app.module.ts

Тепер виведемо наш компонент на сторінку. Для цього відкриємо файл app.component.html та додамо до нього рядок <app-my></app-my> (рис. 2.19).

```
1   <h1>Hello world !!</h1>
2   <app-my></app-my>
```

Рисунок 2.19 – Змінений app.component.html

Переглянемо результат додавання нового компонента у браузері (рис. 2.20).

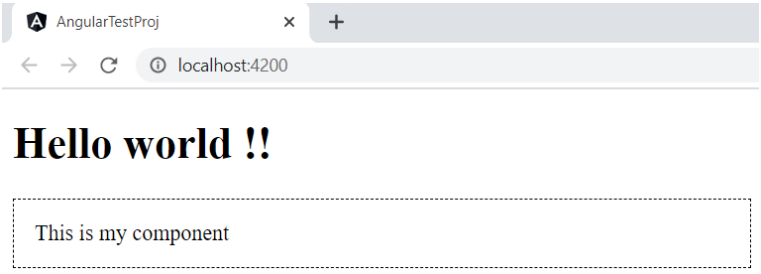

Рисунок 2.20 – Зміни на сторінці застосунку

Повернімося до файлу my.component.ts. Зараз у ньому представлений і клас MyComponent, і html-шаблон компонента, і його таблиця стилів css. Цей варіант підійде лише для дуже простих компонент, html та css-код яких є дуже малим. Але навіть у цьому випадку від виглядає досить непривабливо, оскільки поєднує код трьох різних мов. Розділимо html, css та TypeScript у різні файли. Для цього у директорії scr/app/my створимо файли my.component.html та my.component.scss такого вмісту (рис. 2.21–2.22).

```
<div>
    <h3>This is my component</h3>
    <p>version 0.0.2</p>
</div>
```

Рисунок 2.21 – Файл my.component.html

```
div {
  border: dashed 1px black;
  padding: 15px;
  h3 {
    color: blue;
    font-weight: bold;
  }
  p {
    margin: 0px;
  }
}
```

Рисунок 2.22 – Файл my.component.scss

Також виконаємо зміни у my.component.ts: замість наведення html та css коду виконаємо підключення створених my.component.html та my.component.scss (рис. 2.23).

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

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

}
```

Рисунок 2.23 – Зміни у my.component.ts

Переключимось у браузер та переглянемо результат виконаних змін (рис. 2.24).

Рисунок 2.24 – Перегляд виконаних змін

Наступний компонент group створимо, використавши засоби командного рядка Angular CLI. Для цього виконаємо команду ng generate component group --skipTests або її коротку форму ng g c group --skipTests (рис. 2.25).

```
C:\data\personal\michael\MOHYLA\Education\SPA_PWA\code\angular-test-proj>ng g c group --skipTests
CREATE src/app/group/group.component.html (20 bytes)
CREATE src/app/group/group.component.ts (266 bytes)
CREATE src/app/group/group.component.scss (0 bytes)
UPDATE src/app/app.module.ts (448 bytes)
```

Рисунок 2.25 – Створення компонента засобами Angular CLI

У цей раз Angular CLI автоматично створює файли для html, css та typescript, а також виконує підключення нового компоненту в app.module.ts. Відкриємо group.component.html та змінимо його (рис. 2.26).

```
1  <div class="group">
2      <h3>Група 301</h3>
3      <p class="faculty">факультет комп'ютерних наук</p>
4      <p class="specialty">спеціальність: комп'ютерні науки</p>
5  </div>
```

Рисунок 2.26 – Зміни group.component.html

Також додаємо стилізацію для нового компонента (рис. 2.27).

```scss
div.group {
  padding: 20px;
  background: linen;
  border-radius: 5px;
  h3 {
    margin-top: 0px;
  }
  p.faculty {
    color: blue;
  }
  p.specialty {
    font-size: 0.8em;
    color: brown;
  }
}
```

Рисунок 2.27 – Файл group.component.scss

Додаємо виведення компонента в app.component.html (рис. 2.28) та переглядаємо результат (рис. 2.29).

```html
<h1>Hello world !!</h1>
<app-group></app-group>
```

Рисунок 2.28 – Додавання app-group до app.component.html

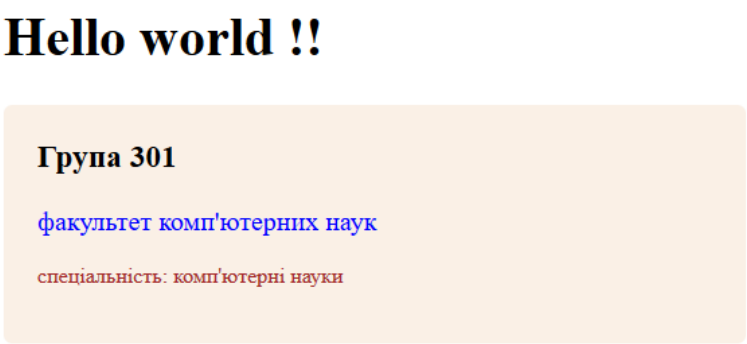

Рисунок 2.29 – Перегляд результату

Додаємо нашому компоненту трохи динаміки. Реалізуємо у класі GroupComponent 3 поля – номер групи, назву факультету та спеціальності (рис. 2.30).

```
export class GroupComponent implements OnInit {

  number = '308';
  faculty = `факультет комп'ютерних наук`;
  specialty = `інженерія програмного забезпечення`;

  constructor() { }
```

Рисунок 2.30 – Додавання змінних до класу GroupComponent

Тепер, використовуючи оператор інтерполяції {{ }}, виконаємо вставку в шаблон змінних класу typescript (рис. 2.31).

```
<div class="group">
    <h3>Група {{ number }}</h3>
    <p class="faculty">{{ faculty }}</p>
    <p class="specialty">спеціальність: {{ specialty }}</p>
</div>
```

Рисунок 2.31 – Зміни у шаблоні group.component.html

Після змін сторінка у браузері матиме такий вигляд (рис. 2.32).

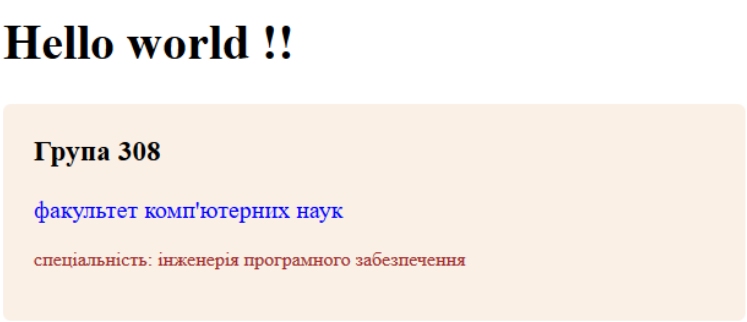

Рисунок 2.32 – Сторінка застосунку після змін

Спробуємо вивести до шаблону значення інших типів. Додаємо до класу значення типу число, масив та об’єкт (рис. 2.33) та виведемо їх на шаблон (рис. 2.34).

```
    number = '308';
    faculty = `факультет комп'ютерних наук`;
    specialty = `інженерія програмного забезпечення`;

    studentsQuantity = 22;
    students = [
        'Іванов Василь',
        'Смірнова Марія',
        'Максимов Іван'
    ];
    starosta = {
        name: 'Іванов Василь',
        age: 20,
        address: 'Mykolaiv, Ukraine'
    }
```

Рисунок 2.33 – Додавання нових полів до класу GroupComponent

```
<div class="group">
    <h3>Група {{ number }}</h3>
    <p class="faculty">{{ faculty }}</p>
    <p class="specialty">спеціальність: {{ specialty }}</p>
    <p>Кількість студентів: {{studentsQuantity}}</p>
    <p>Склад групи: {{students}}</p>
    <p>Староста: {{starosta}}</p>
</div>
```

Рисунок 2.34 – Додавання нових полів
у шаблон group.component.html

Переглядаємо результат (рис. 2.35).

Група 308

факультет комп'ютерних наук

спеціальність: інженерія програмного забезпечення

Кількість студентів: 22

Склад групи: Іванов Василь,Смірнова Марія,Максимов Іван

Староста: [object Object]

Рисунок 2.35 – Результат змін

Як бачимо, з числом жодних проблем, масив виводиться аналогічно до array.join(), а от із об’єктом не все так добре. Для виведення об’єкта у json-форматі можемо застосувати додаткові можливості під час виводу даних, так звані pipe-и (рис. 2.36). Також використаємо тег pre для більш привабливого вигляду виведеного json-у.

```
<p>Склад групи: {{students}}</p>
<p>Староста:</p><pre>{{starosta | json}}</pre>
</div>
```

Рисунок 2.36 – Виведення об’єкту у вигляді json-у

Тепер поле «староста» виглядає таким чином (рис. 2.37).

Староста:

```
{
  "name": "Іванов Василь",
  "age": 20,
  "address": "Mykolaiv, Ukraine"
}
```

Рисунок 2.37 – Вигляд поля у json-форматі

Додаємо трохи динаміки – приховаємо назву факультету та спеціальність через 2 секунди після завантаження сторінки. Для цього у метод ngOnInit розмістимо такий код (рис. 2.38).

```
ngOnInit() {
    setTimeout( handler: () => {
        this.faculty = '';
        this.specialty = '';
    }, timeout: 2000);
```

Рисунок 2.38 – Зміна значень полів класу

Переглянемо результат (рис. 2.39).

Група 308

факультет комп'ютерних наук

спеціальність: інженерія програмного забезпечення

Група 308

спеціальність:

Рисунок 2.39 – Результат зміни значень полів

Для передачі змін із класу в компонент та у зворотному напрямку в Angular є спеціальний механізм, що має назву «двостороннє зв’язування». Атрибут src тегу img буде взято у квадратні дужки, що означає передачу значення із typescript коду класу GroupComponent до html-

шаблону. Отже, реалізуємо картинку, що буде змінюватися кожні декілька секунд (рис. 2.40–2.42), та переглянемо отриманий результат.

```
images = [
  'https://image.mel.fm/i/I/IoT9VfRJLP/590.png',
  'https://kartinki-vernisazh.ru/_ph/478/2/550191551.jpg?1577384607',
  'https://i.pinimg.com/474x/7d/14/ba/7d14bac4b8ba2ff32151088ed0c020d6.jpg'
];
curImageIndex = 0;
curImage: string;

constructor() { }

ngOnInit() {
    this.curImage = this.images[this.curImageIndex];
    setInterval( callback: () => {
        this.curImageIndex++;
        if (this.curImageIndex >= this.images.length) {
            this.curImageIndex = 0;
        }
        this.curImage = this.images[this.curImageIndex];
    }, ms: 2000);
```

Рисунок 2.40 – Зміни у group.component.ts

```
<div class="group">
    <img [src]="curImage">
    <h3>Група {{ number }}</h3>
```

Рисунок 2.41 – Зміни у group.component.html

```
img {
  width: 300px;
  margin-bottom: 20px;
}
```

Рисунок 2.42 – Ширина зображення у group.component.scss

2.3. Події та директиви

Виконаємо зміну картинки у шаблоні також за ініціативою користувача. Для цього додаємо метод, що буде виконувати зміну поточного індексу картинки, а також збільшимо інтервал його зміни за таймером (рис. 2.43).

```
ngOnInit() {
    this.curImage = this.images[this.curImageIndex];
    setInterval( callback: () => {
        this.changeCurImage( forward: true);
    }, ms: 10000);
```

```
41      setTimeout( handler: () => {
42          this.faculty = '';
43          this.specialty = '';
44      }, timeout: 2000);
45  }
46
47  changeCurImage(forward: boolean) {
48      if (forward) {
49          this.curImageIndex++;
50      } else {
51          this.curImageIndex--;
52      }
53      if (this.curImageIndex >= this.images.length) {
54          this.curImageIndex = 0;
55      }
56      if (this.curImageIndex < 0) {
57          this.curImageIndex = this.images.length - 1;
58      }
59      this.curImage = this.images[this.curImageIndex];
60  }
```

Рисунок 2.43 – Додавання методу зміни індексу поточної картинки до group.component.ts

Також додаємо у шаблон 2 кнопки із подією click. Подія виділяється круглими дужками, що сигналізує передачу даних (або подій) від шаблону до класу typescript (рис. 2.44).

```
1  <div class="group">
2      <div class="img-container">
3          <button (click)="changeCurImage(false)"><< prev</button>
4          <button (click)="changeCurImage(true)">next >></button>
5          <div>
6              <img [src]="curImage">
7          </div>
8      </div>
```

Рисунок 2.44 – Додавання кнопок у шаблон group.component.html

Також додаємо трохи стилізації у group.component.scss (рис. 2.45).

```
19  .img-container {
20    text-align: center;
21    margin-bottom: 20px;
22  }
```

Рисунок 2.45 – Додавання стилізації у group.component.scss

У результаті отримуємо 2 додаткові кнопки, що надають можливість гортати зображення (рис. 2.46).

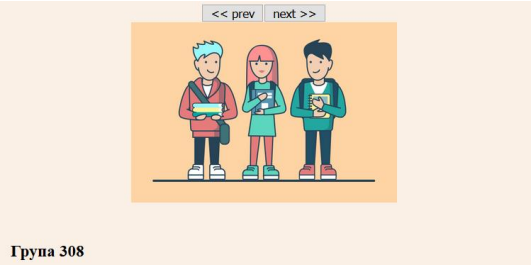

Рисунок 2.46 – Зміни на сторінці застосунку

Реалізуємо також зміну поля номера групи. Для цього поряд із виводом номера групи додаємо поле input, у яке будемо вводити нове значення, та кнопку, із натисканням на яку будемо виконувати зміну значення (рис. 2.47).

```
9       <h3>Група {{ number }}</h3>
10      <div class="edit-number">
11          <input #numb type="text">
12          <button (click)="changeNumber(numb.value)">Змінити</button>
13      </div>
14      <p class="faculty">{{ faculty }}</p>
```

Рисунок 2.47 – Додавання елементів до group.component.html

Також у класі GroupComponent реалізуємо метод, що виконуватиме зміну значення номера групи (рис. 2.48).

```
61
62      changeNumber(numb: string) {
63          this.number = numb;
64      }
```

Рисунок 2.48 – Метод зміни значення поля номера групи у group.component.ts

У результаті маємо функціонал зміни значення номера студентської групи (рис. 2.49).

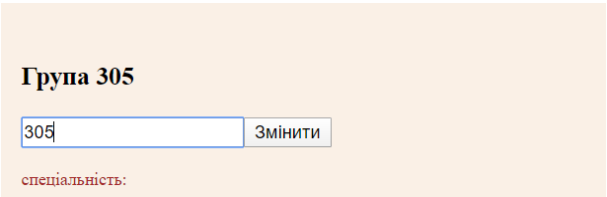

Рисунок 2.49 – Вигляд сторінки після додавання поля вводу

У цьому випадку також можливо обійтися і без кнопки. Змінимо у шаблоні прив'язку до події input та прибираємо кнопку (рис. 2.50). Тепер зміна відбуватиметься з уведенням кожного нового символу у поле вводу.

```
    <div class="edit-number">
        <input #numb type="text" (input)="changeNumber(numb.value)">
<!--        <button (click)="changeNumber(numb.value)">Змінити</button>-->
    </div>
```

Рисунок 2.50 – Зміна події для реагування на зміни значення

Але в Angular можна реалізувати цю задачу ще простіше – за допомогою механізму двостороннього зв'язування, коли зміни у шаблоні автоматично застосовуються для полів класу, і навпаки, зі зміною полів класу автоматично оновляться значення елементів шаблону. Для цього нам, перш за все, необхідно в app.module.ts імпортувати FormsModule (рис. 2.51).

```
import { GroupComponent } from './group/group.component';
import {FormsModule} from '@angular/forms';

  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
```

Рисунок 2.51 – Підключення FormsModule

Далі у шаблоні group.component.html виконаємо двостороннє зв'язування для поля вводу значення номера групи (рис. 2.52).

```
    <div class="edit-number">
        <input type="text" [(ngModel)]="number">
<!--        <input #numb type="text" (input)="changeNumber(numb.value)">-->
<!--        <button (click)="changeNumber(numb.value)">Змінити</button>-->
    </div>
```

Рисунок 2.52 – Використання [(ngModel)]

Тепер навіть можна прибрати із класу GroupComponent метод changeNumber, оскільки Angular візьме на себе всю роботу зі зв'язування елементу input та змінної number.

Тепер трохи познайомимось із директивами. Реалізуємо умовне форматування номера групи, змінимо колір та розмір шрифту для груп 3-го та більш старших курсів. Спочатку додаємо відповідну стилізацію до group.component.scss (рис. 2.53).

```scss
h3 {
  margin-top: 0px;
  &.old {
    color: darkblue;
    font-size: 1.5em;
  }
}
```

Рисунок 2.53 – Зміни у group.component.scss

Далі у класі GroupComponent реалізуємо метод, що визначатиме необхідність стилізації (рис. 2.54).

```ts
isOld() {
    return (+this.number.toString()[0]) > 2;
}
```

Рисунок 2.54 – Метод isOld у group.component.ts

І у шаблоні group.component.html для поля «Номер групи» за допомогою директиви ngClass додаємо клас стилізації для груп, що належать до 3-го курсу та старших (рис. 2.55).

```html
<h3 [ngClass]="{old: isOld()}">Група {{ number }}</h3>
<div class="edit-number">
```

Рисунок 2.55 – Використання директиви ngClass

Далі познайомимося із директивою ngIf, що дозволяє виконувати умовне виведення блоків html-коду у шаблонах. Приховаємо у шаблоні все, окрім номера групи, та додаємо посилання «Детальніше», за яким буде відображатися вся інша інформація. Для цього спочатку додаємо до класу поле-прапорець showInfo (рис. 2.56).

```ts
specialty = 'Інженерія програмного забезпечення';
showInfo = false;
```

Рисунок 2.56 – Додавання поля до класу GroupComponent

Та виконуємо зміни у шаблоні group.component.html (рис. 2.57).

```
<div class="group">
    <div class="img-container" *ngIf="showInfo">
        <button (click)="changeCurImage(false)"><< prev</button>
        <button (click)="changeCurImage(true)">next >></button>
        <div>
            <img [src]="curImage">
        </div>
    </div>

    <h3 [ngClass]="{old: isOld()}">Група {{ number }}</h3>

    <div *ngIf="showInfo; then btnHide else btnShow"></div>
    <ng-template #btnHide>
        <button (click)="showInfo=false">Згорнути</button>
    </ng-template>
    <ng-template #btnShow>
        <button (click)="showInfo=true">Детальніше</button>
    </ng-template>

    <div class="info" *ngIf="showInfo">
        <div class="edit-number">
            <input type="text" [(ngModel)]="number">
            <!--         <input #numb type="text" (input)="chang
            <!--         <button (click)="changeNumber(numb.valu
        </div>
        <p class="faculty">{{ faculty }}</p>
        <p class="specialty">спеціальність: {{ specialty }}</p>
        <p>Кількість студентів: {{studentsQuantity}}</p>
        <p>Склад групи: {{students}}</p>
        <p>Староста:</p><pre>{{starosta | json}}</pre>
    </div>
</div>
```

Рисунок 2.57 – Використання директиви ngIf

Переглядаємо отриманий результат та перевіряємо роботу кнопки «Детальніше» (рис. 2.58).

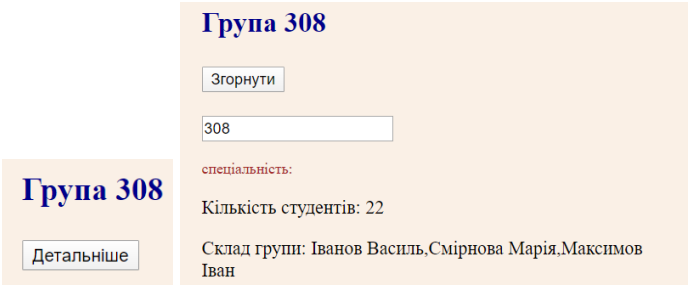

Рисунок 2.58 – Зовнішній вигляд сторінки застосунку

Познайомимось із ще однією директивою, що дозволяє реалізувати циклічний вивід елементів у шаблоні – ngFor. Виконаємо визначення масиву студентських груп та виведемо їх на сторінку. Для початку додаємо створення масиву у app.component.ts (рис. 2.59).

```
export class AppComponent {
  title = 'angular-test-proj';
  groups = [
    {
      number: 301,
      faculty: `факультет комп'ютерних наук`,
      specialty: `комп'ютерні науки`
    },
    {
      number: 308,
      faculty: `факультет комп'ютерних наук`,
      specialty: `інженерія програмного забезпечення`
    },
  ];
}
```

Рисунок 2.59. Масив студентських груп

Використаємо директиву ngFor для тегу app-group в app.component. html (рис. 2.60).

```
<h1>Hello world !!</h1>
<app-group *ngFor="let group of groups"></app-group>
```

Рисунок 2.60 – Використання ngFor

Але під час перегляду сторінки бачимо, що група 308 виводиться 2 рази, адже ми не передаємо до компоненти group жодних значень із батьківської app, і всі дані беруться із прописаних у класі GroupComponent значень. Переробимо GroupComponent таким чином, щоб він міг приймати вхідні дані та працював із ними. Також прибираємо із класу та шаблону компонента group поля «Студенти та староста» і «Вивід картинок» (рис. 2.61–2.62).

```ts
import {Component, Input, OnInit} from '@angular/core';

@Component({
  selector: 'app-group',
  templateUrl: './group.component.html',
  styleUrls: ['./group.component.scss']
})
export class GroupComponent implements OnInit {
  @Input() group;
  showInfo = false;

  constructor() { }

  ngOnInit() {
  }

  isOld() {
      return (+this.group.number.toString()[0]) > 2;
  }
}
```

Рисунок 2.61 – Зміни у group.component.ts

```html
<div class="group">
    <h3 [ngClass]="{old: isOld()}">Група {{ group.number }}</h3>

    <div *ngIf="showInfo; then btnHide else btnShow"></div>
    <ng-template #btnHide>
        <button (click)="showInfo=false">Згорнути</button>
    </ng-template>
    <ng-template #btnShow>
        <button (click)="showInfo=true">Детальніше</button>
    </ng-template>

    <div class="info" *ngIf="showInfo">
        <div class="edit-number">
            <input type="text" [(ngModel)]="group.number">
        </div>
        <p class="faculty">{{ group.faculty }}</p>
        <p class="specialty">спеціальність: {{ group.specialty }}</p>
        <p>Кількість студентів: {{group.studentsQuantity}}</p>
    </div>
</div>
```

Рисунок 2.62 – Зміни у group.component.html

Тепер наш компонент GroupComponent вміє приймати вхідні дані. Скористаймося цим, передаємо до нього дані студентських груп із AppComponent (рис. 2.63).

```
<h1>Hello World !!</h1>
<app-group *ngFor="let group of groups" [group]="group"></app-group>
```

Рисунок 2.63 – Передача даних у компонент group

Переглянемо, як змінилась сторінка нашого застосунку (рис. 2.64).

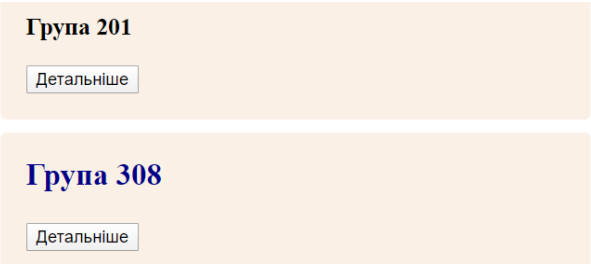

Рисунок 2.64 – Перевірка виведення студентських груп

Дочірні компоненти також можуть не тільки приймати значення від батьківських, а і передавати дані у батьківський компонент. Для цього використовується декоратор @Output та у дочірньому компоненті створюємо обробник подій, який ініціює цю передачу. Так, виконаємо у компоненті app.component виведення дати останньої зміни даних, при тому, що сама зміна буде відбуватися у компоненті group.component.

Спочатку в класі компонента group додаємо @Output параметр та метод, що ініціює його повернення (рис. 2.65).

```
import {Component, EventEmitter,
  Input, OnInit, Output} from '@angular/core';
  @Input() group;
  @Output() dataChange = new EventEmitter();
  showInfo = false;
  onChange() {
    this.dataChange.emit(new Date());
  }
```

Рисунок 2.65 – Додавання повернення даних

У шаблоні group.component.html викликаємо повернення даних із компонента у результаті зміни номера групи (рис. 2.66).

```
<div class="edit-number">
    <input type="text"
           [(ngModel)]="group.number"
           (ngModelChange)="onChange()">
</div>
```

Рисунок 2.66 – Виклик методу під час зміни даних

У класі AppComponent створимо поле для дати останньої зміни та метод, що писатиме туди значення, отримане від компонента GroupComponent (рис. 2.67).

```
title = 'angular-test-proj';
changeDate = new Date();
groups = [
onDataChange(event) {
  this.changeDate = event;
}
```

Рисунок 2.67 – Обробка отримання даних

У шаблоні app.component.html виведемо дату останньої зміни та пов'яжемо подію компонента group із методом onDataChange (рис. 2.68).

```
<h1>Hello world !!</h1>
<p>last data change:
    <span>{{changeDate|date:'dd.MM.yyyy HH:mm:ss'}}</span>
</p>
<app-group
        *ngFor="let group of groups"
        [group]="group"
        (dataChange)="onDataChange($event)">
</app-group>
```

Рисунок 2.68 – Зміни у шаблоні app.component.html

Перевіряємо роботу сторінки.

2.4. Реалізація роботи зі списком студентських груп

Підключимо до нашого проєкту стилі materialize. Встановити materialize можна, використавши менеджер пакетів npm за допомогою команди npm install materialize-css@next, але ми використаємо інший спосіб – завантажимо необхідні бібліотеки у папку src\assets та виконаємо їх підключення у angular.json (рис. 2.69).

```
        ],
        "styles": [
          "src/assets/materialize.min.css",
          "src/styles.scss"
        ],
        "scripts": [
          "src/assets/materialize.min.js"
        ]
      }
```

Рисунок 2.69 – Підключення materialize

Також за cdn-посиланням виконаємо підключення material icons в index.html (рис. 2.70).

```
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link rel="stylesheet"
        href="https://fonts.googleapis.com/icon?family=Material+Icons">
</head>
<body>
```

Рисунок 2.70 – Підключення material icons

Далі змінимо код класів та шаблонів для компонент app та group, прибираємо зайві частини коду та адаптуємо html під materialize css. Почнімо з app.component.ts – тут поки що залишається тільки масив студентських груп (рис. 2.71).

```
export class AppComponent {
  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
    },
  ];
}
```

Рисунок 2.71 – app.component.ts

Далі наведемо код шаблона для app.component.html (рис. 2.72).

```
<nav>
    <div class="nav-wrapper">
        <a class="brand-logo">Групи</a>
    </div>
</nav>
<div class="container">
    <div class="row" style="margin-top: 20px;">
        <div class="col m6 offset-m3 s12">
            <a class="btn-floating btn-large waves-effect waves-light green">
                <i class="material-icons">add</i>
            </a>
        </div>
    </div>
    <app-group
            *ngFor="let group of groups"
            [group]="group">
    </app-group>
</div>
```

Рисунок 2.72 – app.component.html

Переходимо до компоненти group, почнімо також із класу. Тут поки залишається тільки поле showInfo та вхідний параметр group (рис. 2.73).

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

  constructor() { }

  ngOnInit() {
  }
}
```

Рисунок 2.73 – group.component.ts

Шаблон group.component.html також адаптуємо під materialize css (рис. 2.74).

```
<div class="row valign-wrapper">
    <div class="col m3 offset-m3 s6 center-align">
        <h5>{{group.number}}</h5>
    </div>
    <div class="col m3 s6 center-align">
        <a class="btn-floating waves-effect waves-light blue"
           (click)="showInfo=!showInfo">
            <i *ngIf="showInfo" class="material-icons">
```

```
                keyboard_arrow_up
            </i>
            <i *ngIf="!showInfo" class="material-icons">
                keyboard_arrow_down
            </i>
        </a>
    </div>
</div>

<div class="info" *ngIf="showInfo">
    <div class="input-field">
        <input id="number" [(ngModel)]="group.number"
               type="text" class="validate">
        <label for="number" class="active">Номер групи</label>
    </div>
    <div class="input-field">
        <input id="faculty" [(ngModel)]="group.faculty"
               type="text" class="validate">
        <label for="faculty" class="active">Факультет</label>
    </div>
    <div class="input-field">
        <input id="specialty" [(ngModel)]="group.specialty"
               type="text" class="validate">
        <label for="specialty" class="active">Спеціальність</label>
    </div>
    <div class="input-field">
        <input id="quantity" [(ngModel)]="group.studentsQuantity"
               type="number" class="validate">
        <label for="quantity" class="active">Кількість студентів</label>
    </div>
    <a class="rem-row btn-floating waves-effect waves-light red">
        <i class="material-icons">delete</i>
    </a>
</div>
```

Рисунок 2.74 – group.component.html

Також додаємо трохи власної стилізації для компоненти group (рис. 2.75).

```
div.info {
  background: #fff5fd;
  border-radius: 10px;
  padding: 10px;
}
```

Рисунок 2.75 – Group.component.scss

Переглянемо отриманий результат (рис. 2.76).

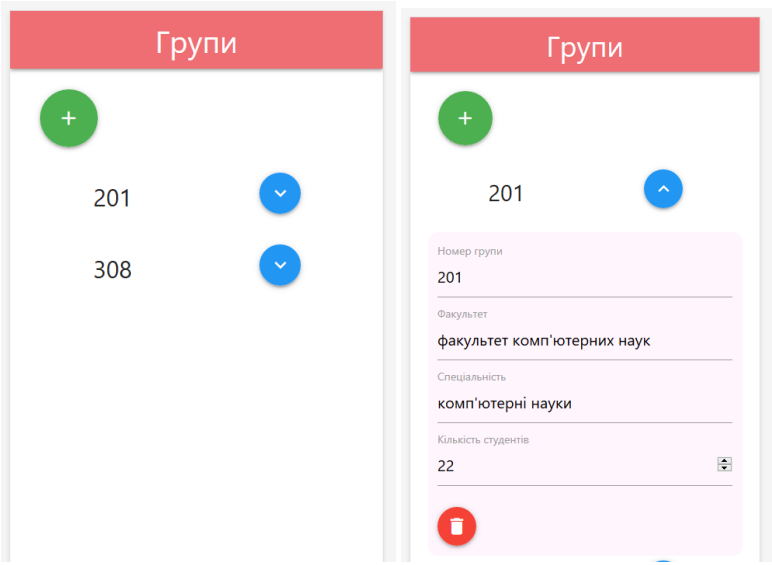

Рисунок 2.76 – Перегляд зовнішнього вигляду сторінки

Залишається реалізувати функціонал додавання та видалення студентських груп. Почнімо із додавання елементу, виділивши кнопку та форму додавання в окремий компонент. Додаємо новий компонент (рис. 2.77).

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

Рисунок 2.77 – Додавання компонента new-group

Реалізуємо клас NewGroupComponent, функція якого зводиться до передачі до батьківського компоненту об’єкта нової групи (рис. 2.78).

```
1  import {Component, EventEmitter, OnInit, Output} from '@angular/core';
2
3  @Component({
4    selector: 'app-new-group',
5    templateUrl: './new-group.component.html',
6    styleUrls: ['./new-group.component.scss']
7  })
8  export class NewGroupComponent implements OnInit {
```

```
@Output() group = new EventEmitter();
showForm = false;

constructor() { }

ngOnInit() {
}

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

Рисунок 2.78 – new-group.component.ts

Переходимо до реалізації макета new-group.component.html (рис. 2.79).

```
<div class="row">
    <div class="col m6 offset-m3 s12">
        <a *ngIf="!showForm" (click)="showForm=true"
           class="btn-floating btn-large waves-effect waves-light green">
            <i class="material-icons">add</i>
        </a>
        <form *ngIf="showForm" #myForm="ngForm" novalidate
              (ngSubmit)="onSubmit(myForm)">
            <h5>Нова група</h5>
            <div class="input-field">
                <input id="number" name="number" ngModel
                       required type="text">
                <label for="number">Номер групи</label>
            </div>
            <div class="input-field">
                <input id="faculty" name="faculty" ngModel
                       required type="text">
                <label for="faculty">Факультет</label>
            </div>
            <div class="input-field">
                <input id="specialty" name="specialty" ngModel
                       required type="text">
                <label for="specialty">Спеціальність</label>
            </div>
            <div class="input-field">
                <input id="quantity" name="studentsQuantity" ngModel
                       required type="number">
                <label for="quantity">Кількість студентів</label>
            </div>
```

```
            <div class="form-group">
                <button type="button" (click)="showForm=false"
                        class="rem-row btn-floating waves-effect
                        waves-light left blue-grey">
                    <i class="material-icons">keyboard_arrow_up</i>
                </button>
                <button type="submit" [disabled]="myForm.invalid"
                        class="rem-row btn-floating waves-effect
                        waves-light green right">
                    <i class="material-icons">done</i>
                </button>
            </div>
        </form>
    </div>
</div>
```

Рисунок 2.79 – new-group.component.html

Також додаємо трохи власної стилізації (рис. 2.80).

```
div.row {
  margin-top: 20px;
}
```

Рисунок 2.80 – new-group.component.scss

У батьківському компоненті app.component змінимо макет, використавши новий компонент new-group (рис. 2.81).

```
<nav>
    <div class="nav-wrapper">
        <a class="brand-logo">Групи</a>
    </div>
</nav>
<div class="container">
    <app-new-group (group)="addGroup($event)"></app-new-group>
    <app-group
            *ngFor="let group of groups"
            [group]="group">
    </app-group>
</div>
```

Рисунок 2.81 – Зміни у app.component.html

У класі AppComponent створимо метод addGroup, що додаватиме новий елемент до масиву (рис. 2.82).

```

  addGroup(group) {
    this.groups.push(group);
  }
```

Рисунок 2.82 – Зміни в app.component.ts

Перевіряємо роботу функції додавання нової групи (рис. 2.83).

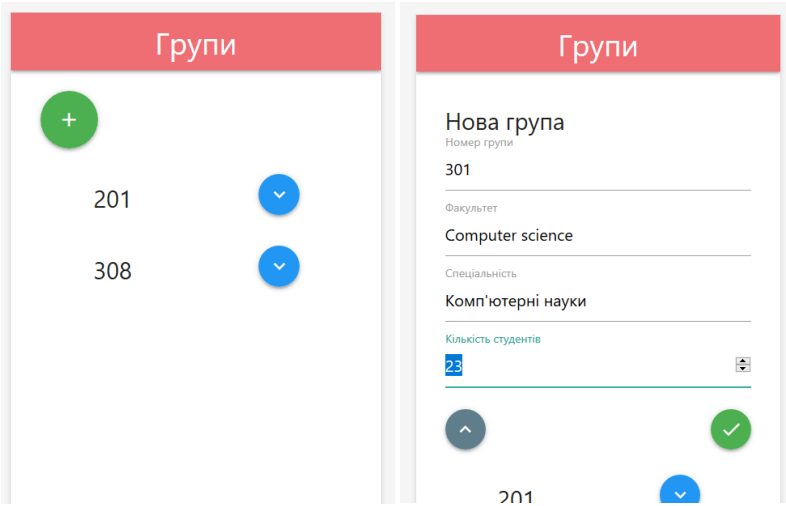

Рисунок 2.83 – Додавання нової групи

Реалізуємо функціонал видалення студентської групи. Додаємо output-параметр для компоненти GroupComponent та метод, що ініціюватиме його передачу батьківському компоненту AppComponent. Також додаємо input-параметр, у який будемо передавати значення індексу масиву для поточного елементу (рис. 2.84).

```
9   export class GroupComponent implements OnInit {
10    @Input() group;
11    @Input() grpIndex;
12    @Output() removeGroup = new EventEmitter();
13    showInfo = false;
19
20    delGroup() {
21      this.removeGroup.emit(this.grpIndex);
22    }
```

Рисунок 2.84 – Додавання коду до GroupComponent

У шаблоні group.component.html підключаємо до події натискання на кнопку «Видалити» метод delGroup (рис. 2.85).

```
39      <a class="rem-row btn-floating waves-effect waves-light red"
40          (click)="delGroup()">
41          <i class="material-icons">delete</i>
42      </a>
```

Рисунок 2.85 – Натискання на кнопку «Видалити»

Також виконаємо доопрацювання компонента AppComponent – у шаблоні під час виведення тегу app-group передаємо туди індекс поточного елементу та приймаємо від нього подію removeGroup (рис. 2.86).

```
6   <div class="container">
7       <app-new-group (group)="addGroup($event)"></app-new-group>
8       <app-group
9               *ngFor="let group of groups; let i = index"
10              [group]="group" [grpIndex]="i"
11              (removeGroup)="deleteGroup($event)">
12      </app-group>
13  </div>
```

Рисунок 2.86 – Зміни у app.component.thml

У класі AppComponent створюємо метод deleteGroup, що відповідатиме за видалення елементу масиву студентських груп (рис. 2.87).

```
27
28      deleteGroup(index) {
29        this.groups.splice(index,  deleteCount: 1);
30      }
```

Рисунок 2.87 – Зміни у app.component.ts

Перевіримо результат, спробувавши видалити одну групу.

2.5. Використання сервісів та роутингу

Сервіси в Angular представляють класи, які виконують деякі специфічні завдання, наприклад, ведення логів, роботу з даними та ін. 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.

Додаємо 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. 2.88).

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

Рисунок 2.88 – Створення сервісу groups-data

Відкриємо файл groups-data.service.ts та переглянемо його вміст (рис. 2.89).

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

Рисунок 2.89 – 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, що 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.

Стандартно 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 шляхом перенесення у нього масив зі студентськими групами та методи додавання та видалення елементів із класу компоненти AppComponent (рис. 2.90).

```
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
    },
  ];

  constructor() { }

  getGroups() {
      return this.groups;
  }

  addGroup(group) {
      this.groups.push(group);
  }

  deleteGroup(index) {
      this.groups.splice(index, deleteCount: 1);
  }
}
```

Рисунок 2.90 – Код сервісу GroupsDataService

Із класу компоненти AppComponent прибираємо масив та методи роботи з ним, які зараз представлені у сервісі GroupsDataService, у конструкторі підключаємо сервіс і отримуємо дані із нього до локальної змінної (рис. 2.91).

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

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

```
export class AppComponent {
  groups: any[];

  constructor(private groupService: GroupsDataService) {
    this.groups = groupService.getGroups();
  }
}
```

Рисунок 2.91 – 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 зі створеним сервісом (рис. 2.92).

```
<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>
```

Рисунок 2.92 – Представлення app.component.html

Далі виконаємо підключення сервісу у компоненті GroupComponent. Тепер під час видалення групи замість передачі події у батьківський елемент будемо викликати відповідний метод сервісу GroupsDataService (рис. 2.93).

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

  constructor(private groupService: GroupsDataService) { }

  ngOnInit() {
  }

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

Рисунок 2.93 – Виконання змін у GroupComponent

Аналогічні зміни виконуємо у компоненті NewGroupComponent для реалізації операції додавання студентської групи із використанням створеного сервісу (рис. 2.94).

```
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
    });
  }
}
```

Рисунок 2.94 – Виконання змін у NewGroupComponent

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

У реальності процес отримання даних може зайняти деякий час (читання файлу, даних БД, отримання відповіді на http-запит та ін.). Тому більш правильно у цьому 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 проєктування Observer і передбачає 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 у результаті виникнення події у observable. Для створення observable використаємо функцію бібліотеки RxJS of(), що створює інстанцію observable, яка видає елементи масиву, переданого як параметр функції один за одним. Виконаємо необхідні зміни у сервісі groups-data (рис. 2.95).

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

Рисунок 2.95 – Повернення observable із методу getGroups класу GroupsData

Інстанція observable також може бути створена із використанням конструкції new (рис. 2.96).

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

Рисунок 2.96 – Інший спосіб створення observable

Далі у конструкторі класу AppComponent підпишимося на подію отримання даних від сервісу та передаємо отримані дані у змінну groups (рис. 2.97).

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

Рисунок 2.97 – Створення observer

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

У браузері є звична модель навігації по застосунку: під час введення URL-адреси браузер перейде на відповідну сторінку; у разі переходу за посиланням браузер відкриває нову сторінку; у результаті натискання на кнопку назад і вперед браузера виконується переміщення назад і вперед згідно з історією 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 вебпереглядача, 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. 2.98).

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

Рисунок 2.98 – Додавання двох нових компонентів

У компонент group-list перенесемо функціонал із AppComponent (рис. 2.99–2.100).

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

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

  groups: any[];

  constructor(private groupService: GroupsDataService) {
    groupService.getGroups().subscribe(
      next: (groups) => this.groups = groups
    );
  }

  ngOnInit() {
  }

}
```

Рисунок 2.99 – 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>
```

Рисунок 2.100 – group-list.component.html

У модулі застосунку app.module.ts пропишемо додатковий функціонал, що дозволить використовувати маршрутизацію Angular та створить необхідні нам маршрути (рис. 2.101).

```
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 { }
```

Рисунок 2.101 – Створення та підключення маршрутів

Змінимо кореневий компонент – AppComponent, прибираємо з нього все зайве та додаємо область для виведення компонента відповідно до обраного маршруту (рис. 2.102–2.103).

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

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

  constructor() { }
}
```

Рисунок 2.102 – App.component.ts

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

Рисунок 2.103 – App.component.html

Після цього застосунок має працювати так, 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. 2.104).

```
  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},
  ];

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

Рисунок 2.104 – Додавання студентів у сервіс group-data

У шаблоні group.component.html додаємо посилання для переходу до списку студентів групи (рис. 2.105).

```
            keyboard_arrow_down
          </i>
        </a>
        <a [routerLink] = "'/students/'+group.number"
          class="btn-floating waves-effect waves-light blue-grey">
          <i class="material-icons">
            view_headline
          </i>
        </a>
    </div>
```

Рисунок 2.105 – Додавання посилання у group.component.html

Переглядаємо результат (рис. 2.106).

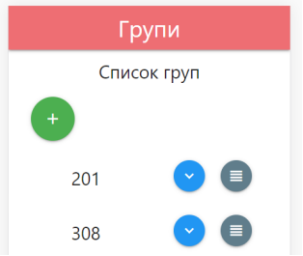

Рисунок 2.106 – Посилання на список студентів

У класі StudentsListComponent реалізуємо отримання значення переданого параметру grpId та метод, який отримає дані щодо списку студентів із сервісу group-data (рис. 2.107).

```
1  import { Component, OnInit } from '@angular/core';
2  import {GroupsDataService} from '../services/groups-data.service';
3  import {ActivatedRoute} from '@angular/router';
4
5  @Component({
6    selector: 'app-student-list',
7    templateUrl: './student-list.component.html',
8    styleUrls: ['./student-list.component.scss']
9  })
10 export class StudentListComponent implements OnInit {
11
12   students: any[];
13
14   constructor(private groupService: GroupsDataService,
15               private activatedRoute: ActivatedRoute) {
16   }
17
18   ngOnInit() {
19     this.activatedRoute.params.subscribe(
20         next: params => this.getStudents(params.grpId)
21     );
22   }
23
24   getStudents(numb: string) {
25     const n = +numb;
26     this.groupService.getStudents(n).subscribe(
27         next: (students) => {
28           this.students = students;
29         }
30     );
31   }
32 }
```

Рисунок 2.107 – Клас StudentsListComponent

Також реалізуємо представлення students-list.component.html для виведення списку студентів (рис. 2.108).

```
<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>
```

Рисунок 2.108 – Представлення students-list.component.html

Переглянемо отриманий результат (рис. 2.109).

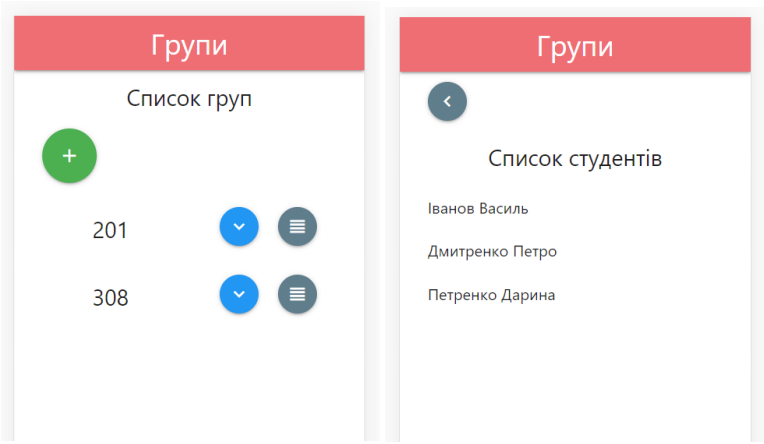

Рисунок 2.109 – Список студентів

РОЗДІЛ 3.
РОЗРОБКА ГІБРИДНИХ ЗАСТОСУНКІВ НА ОСНОВІ IONIC FRAMEWORK

3.1. Налаштування середовища на розгортання проєкту

Для успішної установки та роботи Ionic Framework спочатку необхідно встановити Node.js (платформу з відкритим кодом для виконання застосунків, написаних мовою JavaScript) та npm (менеджер пакетів для мови програмування JavaScript). Перевірити, чи встановлено на комп'ютері Node.js можна, виконавши у командному рядку таку команду (рис. 3.1).

```
C:\Users\michael>node --version
v10.15.3
```

Рисунок 3.1 – Інформація про версію node.js

Аналогічним чином може бути отримана версія менеджера пакетів npm (рис. 3.2).

```
C:\Users\michael>npm --version
6.4.1
```

Рисунок 3.2 – Інформація про версію npm

Якщо на комп'ютері не встановлено Node.js, він може бути завантажений на офіційному сайті (https://nodejs.org/en/download/). Сам процес встановлення, як правило, не має нюансів та може бути виконаний без особливих ускладнень.

Далі для зручності створення та подальшої роботи із проєктами Ionic рекомендується встановлення інструментів командного рядка Ionic CLI (рис. 3.3).

```
C:\Users\michael>npm install -g ionic
npm WARN deprecated superagent@4.1.0: Please note that v5.0.1+ of super
fore you may need to add it yourself (e.g. GitHub blocks requests witho
y with v5.0.2+ once it is released.
[                .....] - remove:readable-stream: sill remove C:\Users\mic
```

Рисунок 3.3 – Встановлення Ionic та Cordova CLI

Після завершення установки інструментів командного рядка виконаємо команду розгортання нового проєкту ionic (рис. 3.4).

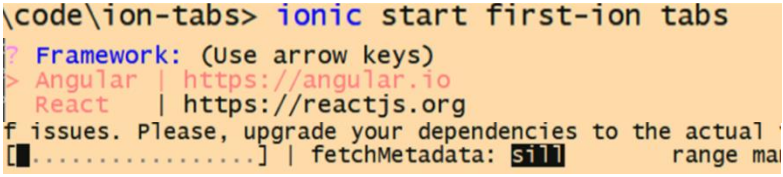

Рисунок 3.4 – Розгортання нового проєкту ionic

Відкриємо проєкт у Visual Studio Code (або іншому редакторі коду) та запустимо його на виконання (рис. 3.5). Проєкт також може бути запущено на виконання із використанням командного рядка за допомогою команди «ionic serve».

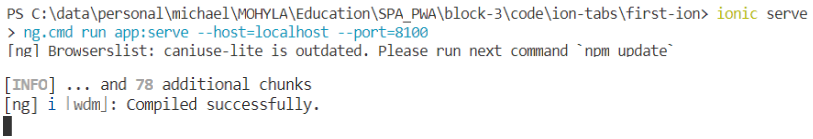

Рисунок 3.5 – Запуск проєкту на виконання

Після запуску застосунок відкривається у браузері стандартно. Також поки виконання не перервано, застосунок доступний за посиланням (http://localhost:8100/). Наблизити зовнішній вигляд застосунку до такого, яким він буде на мобільному пристрої, можна, увімкнувши панель розробника та перейшовши у режим відображення на мобільному пристрої (рис. 3.6).

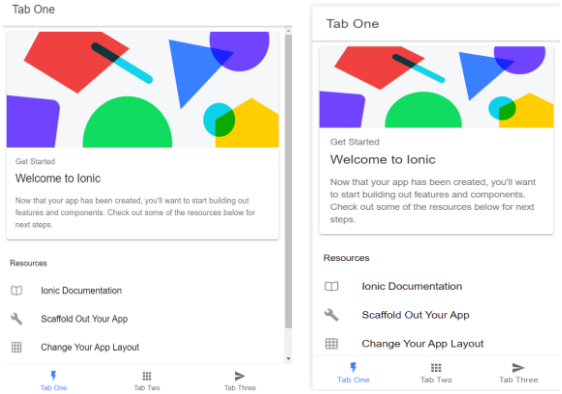

Рисунок 3.6 – Перегляд застосунку

Також є можливість скористатись інструментом ionic-labs, що дозволяє перегляд зовнішнього вигляду інтерфейсу із урахуванням специфіки цільової операційної системи. Для цього запускаємо команду ionic serve із параметром «–l». Якщо це виконується вперше, погоджуємось на встановлення ionic-lab (рис. 3.7–3.8).

```
\first-ion> ionic serve -l
  ? Install @ionic/lab? (Y/n) Y
  ? Install @ionic/lab? Yes
  > npm.cmd i -D -E @ionic/lab
```

Рисунок 3.7 – Запуск ionic serve із параметром «–l»

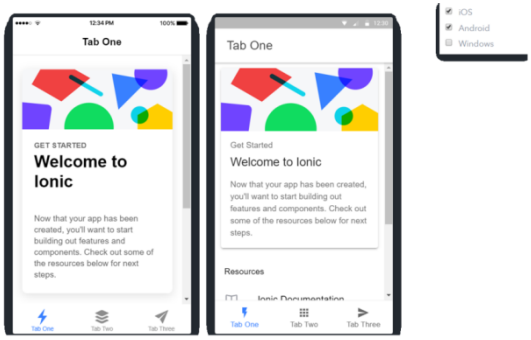

Рисунок 3.8 – Вигляд застосунку у режимі labs

Для зупинки виконання проєкту у терміналі натиснемо Ctr+C та підтвердимо дію.

Виконаємо деякі зміни щодо вмісту та кількості вкладок нашого проєкту. У дереві метаданих відкриємо src->app->tab1->tab1.page.html (рис. 3.9) та змінимо його згідно з рис. 3.10.

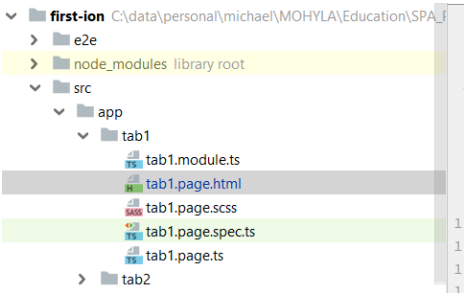

Рисунок 3.9 – Місцезнаходження tab1.page.html

```
<ion-header>
  <ion-toolbar>
    <ion-title>
      Перший проект Ionic
    </ion-title>
  </ion-toolbar>
</ion-header>

<ion-content>
    <ion-card class="welcome-card">
      <img src="https://upload.wikimedia
.org/wikipedia/commons/thumb/d/d1/Ionic_Logo.svg/2000px-Ionic_Logo.svg.png"
alt="" />
      <ion-card-header>
        <ion-card-subtitle>Початок роботи</ion-card-subtitle>
        <ion-card-title>Практична робота 1</ion-card-title>
      </ion-card-header>
      <ion-card-content>
        <p>Створення тестового tab-застосунку на базі ionic-framework</p>
      </ion-card-content>
    </ion-card>
</ion-content>
```

Рисунок 3.10 – Змінений home.html

Запустимо проєкт на виконання (рис. 3.11) та переглянемо результат змін на вкладці «Tab one» (рис. 3.12).

```
first-ion>ng serve
```

Рисунок 3.11 – Запуск проєкту

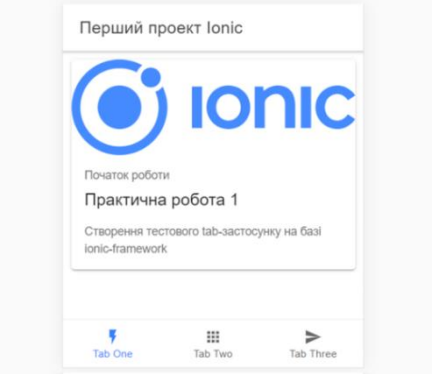

Рисунок 3.12 – Вигляд першої вкладки після виконання змін

Відредагуємо також вміст вкладки tab2 у файлі src/app/tab2/tab2.page.html (рис. 3.13) та змінимо підписи для них у src/app/tabs/tabs.page.html. Також видаляємо кнопку для переходу на третю вкладку, оскільки вона нам не потрібна (рис. 3.14).

```
<ion-header>
  <ion-toolbar>
    <ion-title>
      Додаткові відомості
    </ion-title>
  </ion-toolbar>
</ion-header>

<ion-content>
  <ion-list lines="none">
    <ion-list-header>
      <ion-label>Протягом даної роботи ми</ion-label>
    </ion-list-header>
    <ion-item>
      <ion-icon slot="start" color="medium" name="book"></ion-icon>
      <ion-label class="ion-text-wrap">
        Познайомимось із фреймворком Ionic
      </ion-label>
    </ion-item>
    <ion-item>
      <ion-icon slot="start" color="medium" name="add"></ion-icon>
      <ion-label class="ion-text-wrap">Створимо новий проект tabs</ion-label>
    </ion-item>
    <ion-item>
      <ion-icon slot="start" color="medium" name="build"></ion-icon>
      <ion-label class="ion-text-wrap">Виконаємо невеликі зміни</ion-label>
    </ion-item>
  </ion-list>
</ion-content>
```

Рисунок 3.13 – Зміни на другій вкладці

```
<ion-tabs>

  <ion-tab-bar slot="bottom">
    <ion-tab-button tab="tab1">
      <ion-icon name="logo-ionic"></ion-icon>
      <ion-label>Головна</ion-label>
    </ion-tab-button>

    <ion-tab-button tab="tab2">
      <ion-icon name="information-circle-outline"></ion-icon>
      <ion-label>Додатково</ion-label>
    </ion-tab-button>
  </ion-tab-bar>

</ion-tabs>
```

Рисунок 3.14 – Зміни у панелі навігації

Також видаляємо директорію src\app\tab3, що містить файли для роботи із третьою вкладкою, посилання на яку ми вже видалили попередньо. Після виконання цієї дії отримуємо помилку (рис. 3.15).

```
ERROR in src/app/tabs/tabs-routing.module.ts(36,22): error TS2307:
Cannot find module '../tab3/tab3.module'.
```

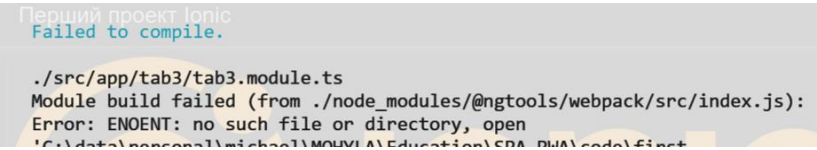

Рисунок 3.15 – Помилка після видалення файлів вкладки

Така помилка має місце, оскільки файли із класом для цієї вкладки все ще продовжують використовуватись у деяких місцях проєкту. Виправимо це. Відкриємо src/app/tabs/tabs-routing.module.ts та видалимо третю вкладку із масиву маршрутів Routes (рис. 3.16).

```
28        ]
29      },
30      {
31        path: 'tab3',
32        children: [
33          {
34            path: '',
35            loadChildren: () =>
36              import('../tab3/tab3.module').then(onfulfilled: m => m.Tab3PageModule)
37          }
38        ]
39      },
40      {
```

Рисунок 3.16 – Видалення непотрібної вкладки з tabs-routing.module.ts

Тепер застосунок виконується без помилок. Наведемо отриманий результат на рис. 3.17.

Розглянемо також послідовність дій на випадок, якщо ми хочемо додати нову вкладку до проєкту. Для цього виконаємо команду створення нової сторінки ionic g page mytab (рис. 3.18).

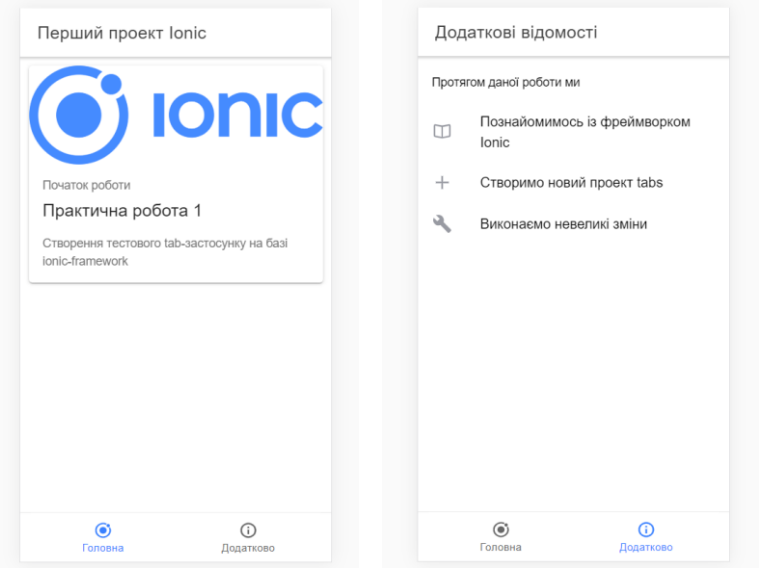

Рисунок 3.17 – Вигляд застосунку після виконання змін

```
first-ion>ionic g page mytab
```

Рисунок 3.18 – Створення нової сторінки

Відкриємо app-routing.module.ts та видалимо новий елемент масиву Routes (рис. 3.19а), створивши аналогічний у файлі tabs-routing.module.ts (рис. 3.19б).

```
3
4   const routes: Routes = [
5     {
6       path: '',
7       loadChildren: () => import('./tabs/tabs.module').then( onfulfilled: m => m.TabsPageModule)
8     },
9     {
10      path: 'mytab',
11      loadChildren: () => import('./mytab/mytab.module').then( onfulfilled: m => m.MytabPageModule)
12    }
13  ];
```

Рисунок 3.19а – Видалення створеного маршруту
до app-routing.module.ts

```
{
  path: 'mytab',
  children: [
    {
      path: '',
      loadChildren: () =>
          import('../mytab/mytab.module').then( onfulfilled: m => m.MytabPageModule)
    }
  ]
},
```

Рисунок 3.19б – Перенесення створеного маршруту
до tabs-routing.module.ts

Додаємо нову кнопку на панель навігації для переходу на створену вкладку у tabs.page.html (рис. 3.20) та трохи відредагуємо представлення самої вкладки у mytab.page.html (рис. 3.21).

```
    <ion-tab-button tab="mytab">
      <ion-icon name="clipboard"></ion-icon>
      <ion-label>Нова вкладка</ion-label>
    </ion-tab-button>
  </ion-tab-bar>
```

Рисунок 3.20 – Додавання кнопки для переходу на вкладку

```
<ion-header>
  <ion-toolbar>
    <ion-title>Нова вкладка</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content>
  <h1 class="ion-text-center">Тестовий контент</h1>
</ion-content>
```

Рисунок 3.21 – Додавання тексту на вкладку

Переглянемо результат (рис. 3.22).

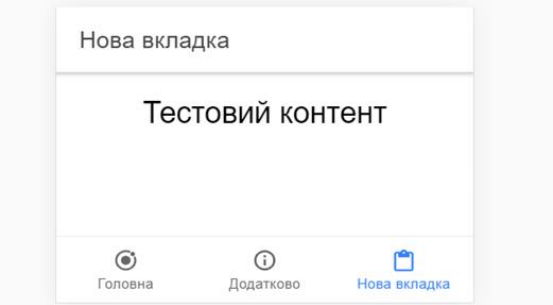

Рисунок 3.22 – Результат додавання нової вкладки

3.2. Створення застосунку «Список студентських груп»

Створимо новий ionic blank-проєкт (рис. 3.23).

```
> ionic start ion-stud-group blank
```

Рисунок 3.23 – Розгортання нового проєкту ionic blank

Відкриємо проєкт у редакторі коду та додаємо до проєкту сервіс, що буде відповідати за отримання даних (рис. 3.24).

```
>ionic g s service/data-getter
```

Рисунок 3.24 – Додавання сервісу data-getter

Відкриваємо файл service/data-getter.service.ts та реалізуємо у ньому функціонал щодо отримання даних студентських груп (рис. 3.25). Тут, за допомогою інтерфейсу, визначимо тип даних «студентська група», а у класі DataGetterService – масив студентських груп та методи для отримання, додавання та видалення даних.

```
1  import { Injectable } from '@angular/core';
2  import {Observable, of} from 'rxjs';
3
4  export interface StudGroup {
5    number: number;
6    faculty: string;
7    specialty: string;
8    studentsQuantity: number;
9  }
10
11 @Injectable({
12   providedIn: 'root'
13 })
14 export class DataGetterService {
15   private groups: StudGroup[] = [
16     {
17       number: 201,
18       faculty: `факультет комп'ютерних наук`,
19       specialty: `комп'ютерні науки`,
20       studentsQuantity: 22
21     },
22     {
23       number: 308,
24       faculty: `факультет комп'ютерних наук`,
25       specialty: `інженерія програмного забезпечення`,
26       studentsQuantity: 24
27     },
28   ];
```

```
29
30      constructor() { }
31
32      getGroups(): Observable<StudGroup[]> {
33        return of(this.groups);
34      }
35
36      addGroup(group: StudGroup) {
37        this.groups.push(group);
38      }
39
40      deleteGroup(index) {
41        this.groups.splice(index,  deleteCount: 1);
42      }
43  }
```

Рисунок 3.25 – Сервіс data-getter

У класі HomePage у конструкторі підключаємо створений сервіс та використовуємо його для отримання масиву студентських груп у локальну змінну. Також декларуємо три методи для додавання, зміни та видалення даних (рис. 3.26).

```
1   import { Component } from '@angular/core';
2   import {DataGetterService, StudGroup} from '../service/data-getter.service';
3
4   @Component({
5     selector: 'app-home',
6     templateUrl: 'home.page.html',
7     styleUrls: ['home.page.scss'],
8   })
9   export class HomePage {
10
11    groups: StudGroup[];
12
13    constructor(private dataGetter: DataGetterService) {
14      this.dataGetter.getGroups().subscribe(
15            next: (data) => {
16              this.groups = data;
17            }
18      );
19    }
20
21    add() {}
22
23    edit(group: StudGroup) {}
24
25    delete(group: StudGroup) {}
26  }
```

Рисунок 3.26 – Файл home.page.ts

Тепер створимо представлення для виведення списку студентських груп на сторінці застосунку. Для цього виконаємо відповідні зміни у файлі home.page.html (рис. 3.27). Під час реалізації представлення використаємо такі компоненти ionic-framework, як ion-list, ion-item, ion- item-sliding, ion-button, ion-icon та деякі інші. Із детальним оглядом цих та інших компонент Ionic framework можна ознайомитись на офіційному сайті проєкту (https://ionicframework.com/docs/components).

```
<ion-header>
  <ion-toolbar>
    <ion-title>
      Студентські групи
    </ion-title>
  </ion-toolbar>
</ion-header>

<ion-content>
  <ion-list>
    <ion-list-header>
      <ion-label>Групи</ion-label>
      <ion-button shape="round" fill="outline"
                  color="success" (click)="add()">
        <ion-icon slot="start" name="add"></ion-icon>Додати
      </ion-button>
    </ion-list-header>

    <ion-item-sliding *ngFor="let group of groups">
      <ion-item-options side="start">
        <ion-item-option color="primary" (click)="edit(group)">
          <ion-icon name="create"></ion-icon>
          Змінити
        </ion-item-option>
        <ion-item-option color="danger" (click)="delete(group)">
          <ion-icon name="trash"></ion-icon>
          Видалити
        </ion-item-option>
        </ion-item-option>
      </ion-item-options>
      <ion-item>
        <ion-icon name="people" slot="start"></ion-icon>
        <ion-label>{{group.number}}</ion-label>
        <ion-note slot="end">{{group.specialty}}</ion-note>
      </ion-item>
    </ion-item-sliding>
  </ion-list>
</ion-content>
```

Рисунок 3.27 – Файл представлення home.page.html

У файлі home.page.scss визначимо невеликий відступ справа для іконок у розділі ion-item-sliding (рис. 3.28).

```
ion-item-sliding {
  ion-icon {
    margin-right: 15px;
  }
}
```

Рисунок 3.28 – Файл ion-item-sliding

Запустимо застосунок та переглянемо отриманий результат (рис. 3.29).

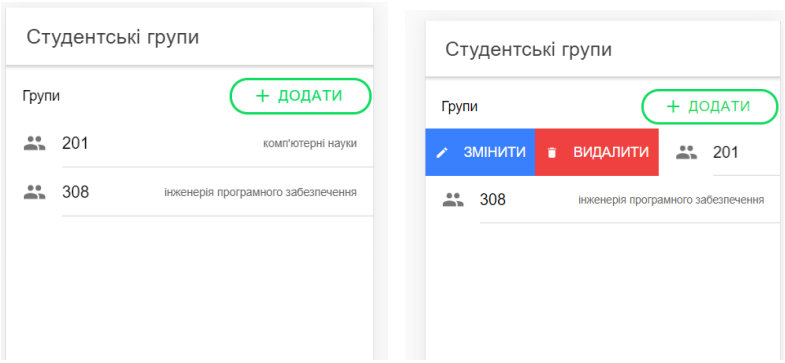

Рисунок 3.29 – Сторінка списку студентських груп

Далі реалізуємо функціонал додавання, зміни та видалення студентських груп. Для цього створимо новий компонент, у що виводитиме детальні дані студентської групи та буде використовуватись для додавання нової або зміни існуючої групи (рис. 3.30).

```
>ionic g c components/stud-group
```

Рисунок 3.30 – Створення компоненти stud-group

Відкриємо файл класу TypeScript та виконаємо реалізацію логіки компонента (рис. 3.31).

```
import {Component, EventEmitter, Input, OnInit, Output} from
 '@angular/core';
import {StudGroup} from '../../service/data-getter.service';

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

  @Input() studentGroup: StudGroup;
  @Input() isNew: boolean;
  @Output() addGroup = new EventEmitter();
  @Output() cancelAddingGroup = new EventEmitter();
  title: string;

  constructor() {}

  ngOnInit() {
    if (this.isNew) {
      this.studentGroup = {
        number: null,
        specialty: '',
        studentsQuantity: null,
        faculty: ''
      };
      this.title = 'Нова група';
    }
  }

  addNew() {
    if (this.isNew) {
      this.addGroup.emit(this.studentGroup);
    }
  }

  cancelAdding() {
    if (this.isNew) {
      this.cancelAddingGroup.emit();
    }
  }
}
```

Рисунок 3.31 – Файл stud-group.component.ts

Реалізуємо представлення компоненти, що представлятиме собою декілька полів вводу даних. Всі вони за допомогою механізму двостороннього зв’язування пов’язані із відповідними полями об’єкта group. Також для режиму додавання виводимо 2 кнопки (рис. 3.32).

```
<ion-card>
  <ion-card-header>
    <ion-card-title>{{title}}</ion-card-title>
  </ion-card-header>

  <ion-card-content>
    <ion-item>
      <ion-label position="floating">Номер</ion-label>
      <ion-input type="number" [(ngModel)]="studentGroup.number" name="number"></ion-input>
    </ion-item>
    <ion-item>
      <ion-label position="floating">Факультет</ion-label>
      <ion-input type="text" [(ngModel)]="studentGroup.faculty" name="faculty"></ion-input>
    </ion-item>
    <ion-item>
      <ion-label position="floating">Спеціальність</ion-label>
      <ion-input type="text" [(ngModel)]="studentGroup.specialty" name="specialty"></ion-input>
    </ion-item>
    <ion-item>
      <ion-label position="floating">Студентів</ion-label>
      <ion-input type="number" [(ngModel)]="studentGroup.studentsQuantity" name="studentQuantity"></ion-input>
    </ion-item>
    <ion-button *ngIf="isNew" color="primary" (click)="addNew()">
      <ion-icon slot="start" name="add-circle"></ion-icon> Додати
    </ion-button>
    <ion-button *ngIf="isNew" color="danger"  class="ion-float-right" (click)="cancelAdding()">
      <ion-icon slot="start" name="close"></ion-icon> Відміна
    </ion-button>
  </ion-card-content>
</ion-card>
```

Рисунок 3.32 – Файл stud-group.component.html

Використаємо наш новостворений компонент на сторінці home. Почнімо зі змін у представленні home.page.html, де, окрім додавання нової компоненти для кожного рядку у циклі, також реалізуємо події, за якими буде виконуватись відображення та приховання компоненти. Далі наводимо повний код представлення сторінки home після виконання необхідних змін (рис. 3.33).

```
<ion-header>
  <ion-toolbar>
    <ion-title>
      Студентські групи
    </ion-title>
  </ion-toolbar>
</ion-header>
```

```
<ion-content>
  <ion-list>
    <ion-list-header>
      <ion-label>Групи</ion-label>
      <ion-button shape="round" fill="outline"
                  color="success" (click)="add()">
        <ion-icon slot="start" name="add"></ion-icon>Додати
      </ion-button>
    </ion-list-header>

    <app-stud-group *ngIf="showNew" [isNew]="true"
                    (addGroup)="addGroup($event)"
                    (cancelAddingGroup)="showNew=false"></app-stud-group>

    <div *ngFor="let group of groups;let i = index">
      <ion-item-sliding>
        <ion-item-options side="start">
          <ion-item-option color="primary" (click)="showEdit=i">
            <ion-icon name="create"></ion-icon>
            Змінити
          </ion-item-option>
          <ion-item-option color="danger" (click)="delete(i)">
            <ion-icon name="trash"></ion-icon>
            Видалити
          </ion-item-option>
        </ion-item-options>
        <ion-item (click)="showEdit=-1">
          <ion-icon name="people" slot="start"></ion-icon>
          <ion-label>{{group.number}}</ion-label>
          <ion-note slot="end">{{group.specialty}}</ion-note>
        </ion-item>
      </ion-item-sliding>
      <app-stud-group *ngIf="showEdit==i" [isNew]="false"
                      [studentGroup]="group"></app-stud-group>
    </div>
  </ion-list>
</ion-content>
```

Рисунок 3.33 – Представлення home.page.html

У класі HomePage додаємо поля, що відповідатимуть за відображення полів вводу, та реалізуємо методи додавання та видалення даних (рис. 3.34).

```
import { Component } from '@angular/core';
import {DataGetterService, StudGroup} from '../service/data-getter.service';

@Component({
  selector: 'app-home',
  templateUrl: 'home.page.html',
  styleUrls: ['home.page.scss'],
})
```

```
9   export class HomePage {
10
11    groups: StudGroup[];
12
13    showNew = false;
14    showEdit = -1;
15
16    constructor(private dataGetter: DataGetterService) {
17      this.dataGetter.getGroups().subscribe(
18            next: (data) => {
19              this.groups = data;
20            }
21      );
22    }
23
24    add() {
25      this.showNew = true;
26    }
27
28    delete(index: number) {
29        this.dataGetter.deleteGroup(index)
30    }
31
32    addGroup(group) {
33      this.dataGetter.addGroup(group);
34      this.showNew = false;
35    }
36  }
```

Рисунок 3.34 – Клас HomePage

Перевіряємо роботу додавання нової групу, змінюємо дані та видаляємо (рис. 3.35).

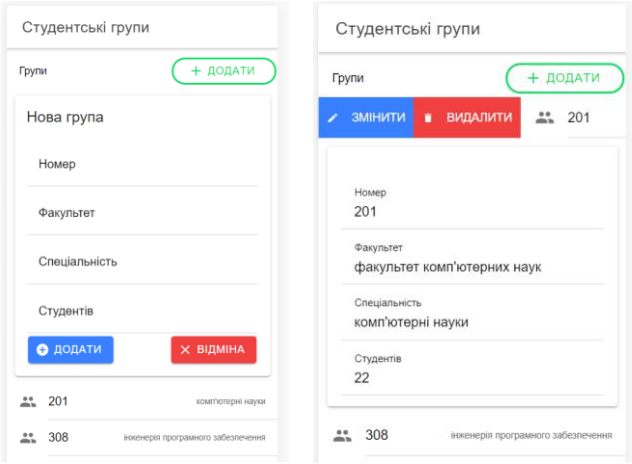

Рисунок 3.35 – Додавання та зміна елементу

3.3. Додавання сторінок та аутентифікація користувачів

Продовжимо роботу над проєктом «Студентські групи» та додамо ще одну сторінку login, яка буде використовуватись для входу користувачів до застосунку (рис. 3.36).

```
>ng g page login
```

Рисунок 3.36 – Додавання нової сторінки

Відкриємо представлення сторінки login.page.html та розмістимо там поле вводу для імені користувача та кнопку «Увійти». Із натисканням на кнопку виконаємо метод, що відповідатиме за перехід на головну сторінку (рис. 3.37).

```
<ion-header>
  <ion-toolbar>
    <ion-title>Студентські групи</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content>
  <ion-grid>
    <ion-row color="primary" justify-content-center>
      <ion-col align-self-center size-md="6" size-lg="5" size-xs="12">
        <div text-center>
          <h3>ВХІД</h3>
        </div>
        <ion-item>
          <ion-label position="floating">Ім'я користувача</ion-label>
          <ion-input type="text" name="username"></ion-input>
        </ion-item>
        <ion-button color="primary"  class="ion-float-right" (click)="login()">
          <ion-icon slot="start" name="log-in"></ion-icon> Увійти
        </ion-button>
      </ion-col>
    </ion-row>
  </ion-grid>
</ion-content>
```

Рисунок 3.37 – Файл login.page.html

У класі LoginPage підключимо @angular/router, у методі login виконаємо перехід на домашню сторінку home (рис. 3.38).

```
import { Component, OnInit } from '@angular/core';
import {Router} from '@angular/router';

@Component({
  selector: 'app-login',
  templateUrl: './login.page.html',
  styleUrls: ['./login.page.scss'],
})
export class LoginPage implements OnInit {
  constructor(private router: Router) { }

  ngOnInit() {
  }

  login() {
    this.router.navigate( commands: ['/home']);
  }
}
```

Рисунок 3.38 – Файл login.page.ts

Також додамо трохи власної стилізації для сторінки у login.page.scss (рис. 3.39).

```
ion-header {
  margin-bottom: 25px;
}

ion-item {
  margin-bottom: 10px;
}
```

Рисунок 3.39 – Додавання стилів для login page

Для відображення сторінки логіну при старті застосунку виконаємо зміну у масиві маршрутів у app-routing.module.ts, а саме – властивість redirectTo для кореневого роуту, який маркується (рис. 3.40).

```
const routes: Routes = [
  { path: '', redirectTo: 'login', pathMatch: 'full' },
  { path: 'home', loadChildren: () => import('./home/home.module').then(
    onfulfilled: m => m.HomePageModule)},
  {
    path: 'login',
    loadChildren: () => import('./login/login.module').then( onfulfilled: m =>
  m.LoginPageModule)
  },
];
```

Рисунок 3.40 – Зміни у app-routing.module.ts

У представленні головної сторінки home.page.html до панелі інструментів додаємо кнопку logout, що повертатиме нас до логін-сторінки (рис. 3.41).

```
<ion-header>
  <ion-toolbar>
    <ion-title>
      Студентські групи
    </ion-title>
    <ion-buttons slot="secondary">
      <ion-button routerLink="/login" routerDirection="root">
          <ion-icon slot="icon-only" name="log-out"></ion-icon>
      </ion-button>
    </ion-buttons>
  </ion-toolbar>
</ion-header>
```

Рисунок 3.41 – Зміни у home.page.html

Перевіримо роботу логін-форми, запустимо застосунок та натиснемо кнопку «Увійти» (рис. 3.42).

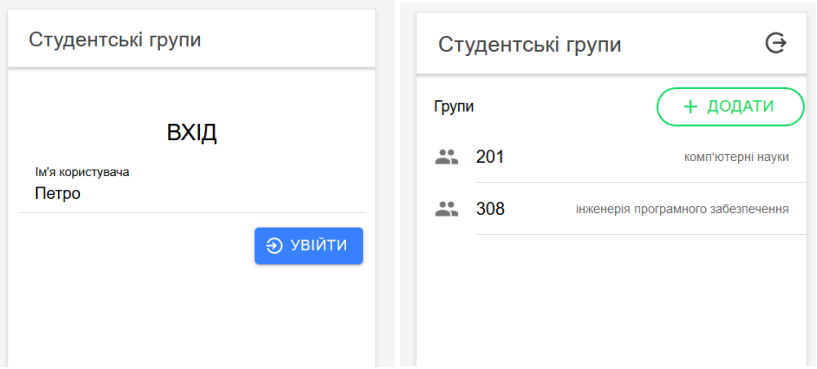

Рисунок 3.42 – Вхід до застосунку

Зараз наша логін-форма не враховує, чи є такий користувач і взагалі дозволяє нічого не вводити у поле вводу, а просто натиснути «Увійти». Виправимо це, розширимо наш сервіс data-getter, додамо до нього масив користувачів, змінну для поточного користувача та методи для роботи із ними (рис. 3.43).

```
  private userName = '';

  private users = [
    'Василь', 'Петро', 'Олена'
  ];

  getUser() {
    return this.userName;
  }

  setUser(name: string) {
    this.userName = name;
  }

  userExists(name: string): boolean {
    return this.users.indexOf(name) !== -1;
  }

  constructor() { }
```

Рисунок 3.43 – Зміни у сервісі data-getter.service.ts

Використаємо доданий до сервісу функціонал під час виконання входу – підключимо сервіс data-getter та будемо перевіряти, чи існує користувач, введений у поле вводу. Якщо користувача немає, буде виводитись повідомлення (рис. 3.44).

```
import { Component, OnInit } from '@angular/core';
import {Router} from '@angular/router';
import {DataGetterService} from '../service/data-getter.service';
import {AlertController} from '@ionic/angular';

@Component({
  selector: 'app-login',
  templateUrl: './login.page.html',
  styleUrls: ['./login.page.scss'],
})
export class LoginPage implements OnInit {
  userName: string;

  constructor(
      private router: Router,
      private dataGetter: DataGetterService,
      public alertController: AlertController) {}

  ngOnInit() {
  }

  login() {
    if (this.dataGetter.userExists(this.userName)) {
      this.dataGetter.setUser(this.userName);
      this.router.navigate( commands: ['/home']);
    } else {
      this.userNotExistAlert();
    }
  }
```

```
30
31      async userNotExistAlert() {
32        const alert = await this.alertController.create({
33          header: 'Увага !',
34          subHeader: 'Помилка аутентифікації',
35          message: `Користувача ${this.userName} на знайдено. Невірне ім'я
    користувача.`,
36          buttons: ['OK']
37        });
38
39        await alert.present();
40      }
41    }
```

Рисунок 3.44 – Зміни у login.page.ts

Також на головній сторінці виведемо ім'я поточного користувача (рис. 3.45).

```
6       <ion-buttons slot="secondary">
7         ({{userName}})
8         <ion-button routerLink="/login" routerDirection="root">
9             <ion-icon slot="icon-only" name="log-out"></ion-icon>
10        </ion-button>
11      </ion-buttons>
```

Рисунок 3.45 – Додавання імені користувача
до представлення home.page.html

Перевіримо результат (рис. 3.46).

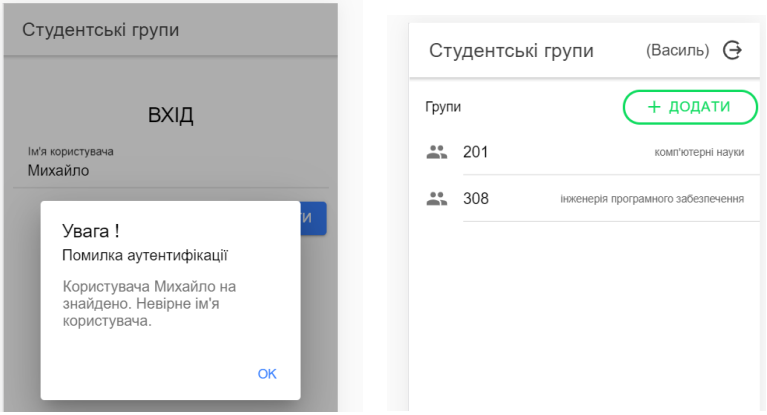

Рисунок 3.46 – Перевірка логін-форми

Залишається наступна проблема – під час переходу за посиланням у адресному рядку до http://localhost:4200/home ми можемо оминути процедуру аутентифікації (рис. 3.47).

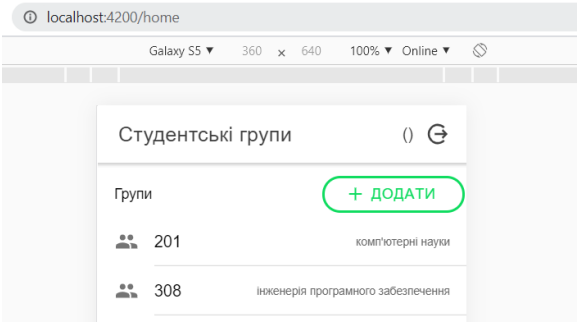

Рисунок 3.47 – Перехід до головної сторінки
в обхід сторінки автентифікації

Виправимо цю проблему, додавши до проєкту guard (ng generate guard guards/auth). Інтерфейс обираємо CanActivate. Далі наведемо код класу AuthGuard (рис. 3.48).

```
import { Injectable } from '@angular/core';
import {ActivatedRouteSnapshot, CanActivate, Router, RouterStateSnapshot}
 from '@angular/router';
import {DataGetterService} from '../service/data-getter.service';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor(private dataGetter: DataGetterService,
              private router: Router) {}

  canActivate(
      next: ActivatedRouteSnapshot,
      state: RouterStateSnapshot): boolean {

    const isLoggenIn = this.dataGetter.getUser() !== '';
    if (!isLoggenIn) {
      this.router.navigateByUrl( url: '/login');
    }
    return isLoggenIn;
  }
}
```

Рисунок 3.48 – Guard Auth

У файлі маршрутизації підключимо створений guard до маршруту home (рис. 3.49).

```
const routes: Routes = [
  { path: '', redirectTo: 'login', pathMatch: 'full' },
  {
    path: 'home',
    loadChildren: () => import('./home/home.module').then(
        onfulfilled: m => m.HomePageModule
    ),
    canActivate: [AuthGuard]
  },
```

Рисунок 3.49 – Зміни у app-routing.module.ts

Тепер у разі спроби переходу до http://localhost:4200/home без проходження аутентифікації будемо перенаправлятися до сторінки логіну.

3.4. Додавання сторінок та передача даних

Продовжимо роботу над проєктом «Студентські групи». Додаємо нову сторінку, на якій будемо відображати список студентів певної групи (рис. 3.50).

```
>ng g page students
```

Рисунок 3.50 – Додавання сторінки списку студентів

Далі перейдемо до файлу маршрутизації app-routing.module.ts та трохи змінимо маршрут для сторінки виводу списку студентів задля можливості передачі у маршрут номера студентської групи. Також підключимо до цього маршруту створений у попередній роботі guard для заборони роботи неаутентифікованих користувачів (рис. 3.51).

```
const routes: Routes = [
  { path: '', redirectTo: 'login', pathMatch: 'full' },
  {
    path: 'home',
    loadChildren: () => import('./home/home.module').then(
        onfulfilled: m => m.HomePageModule
    ),
    canActivate: [AuthGuard]
  },
  {
    path: 'login',
    loadChildren: () => import('./login/login.module').then(
        onfulfilled: m => m.LoginPageModule
    )
  },
```

```
  {
    path: 'students/:grpnumb',
    loadChildren: () => import('./students/students.module').then(
        onfulfilled: m => m.StudentsPageModule
    ),
    canActivate: [AuthGuard]
  },
];
```

Рисунок 3.51 – Зміни у app-routing.module.ts

У сервіс DataGetter додаємо масив студентів та метод для отримання даних студентів певної групи (рис. 3.52).

```
  private students = [
    {name: 'Іванов Василь', groupNumb: 201,
        gender: 'man', rating: 91},
    {name: 'Дмитренко Петро', groupNumb: 201,
        gender: 'man', rating: 85},
    {name: 'Петренко Дарина', groupNumb: 201,
        gender: 'woman', rating: 68},
    {name: 'Васильєва Марина', groupNumb: 308,
        gender: 'woman', rating: 78},
    {name: 'Павлов Микита', groupNumb: 308,
        gender: 'man', rating: 82},
    {name: 'Васін Андрій', groupNumb: 308,
        gender: 'man', rating: 75},
  ];

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

Рисунок 3.52 – Зміни у сервісі data-getter.service.ts

У класі StudentsPage підключаємо інстанцію ActivatedRoute для отримання номера групи із маршруту та сервіс отримання даних DataGetter для отримання списку студентів за цим номером (рис. 3.53).

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

@Component({
  selector: 'app-students',
  templateUrl: './students.page.html',
  styleUrls: ['./students.page.scss'],
})
```

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

  constructor(private dataGetter: DataGetterService,
              private route: ActivatedRoute) { }

  ngOnInit() {
    this.grpnumb = +this.route.snapshot.paramMap.get('grpnumb');
    this.dataGetter.getStudents(this.grpnumb).subscribe(
        next: data => {
          this.students = data;
        }
    );
  }
}
```

Рисунок 3.53 – Файл students.page.ts

Також реалізуємо представлення для новоствореної сторінки (рис. 3.54).

```
<ion-header>
  <ion-toolbar>
    <ion-buttons slot="start">
      <ion-back-button></ion-back-button>
    </ion-buttons>
    <ion-title>Студенти групи {{grpnumb}}</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content>
  <ion-list>
    <ion-item *ngFor="let student of students" text-wrap>
      <ion-icon [name]="student.gender" slot="start"></ion-icon>
      <ion-label text-wrap>{{student.name}}</ion-label>
      <ion-note slot="end">{{student.rating}}</ion-note>
    </ion-item>
  </ion-list>
</ion-content>
```

Рисунок 3.54 – Представлення students.page.html

У представленні списку студентських груп у елементі ion-item-sliding додаємо посилання для переходу до сторінки списку студентів. Нижче наводимо повний код елементу списку ion-item-sliding (рис. 3.55).

```
<ion-item-sliding>
  <ion-item-options side="start">
    <ion-item-option color="primary" (click)="showEdit=i">
      <ion-icon name="create"></ion-icon>
      Змінити
    </ion-item-option>
    <ion-item-option color="danger" (click)="delete(i)">
      <ion-icon name="trash"></ion-icon>
      Видалити
    </ion-item-option>
  </ion-item-options>
  <ion-item (click)="showEdit=-1">
    <ion-icon name="people" slot="start"></ion-icon>
    <ion-label>{{group.number}}</ion-label>
    <ion-note slot="end">{{group.specialty}}</ion-note>
  </ion-item>
   <ion-item-options side="end">
     <ion-item-option color="secondary"
                      routerLink="/students/{{group.number}}"
                      routerDirection="forward">
        <ion-icon name="reorder"></ion-icon>
        Студенти
     </ion-item-option>
   </ion-item-options>
</ion-item-sliding>
```

Рисунок 3.55 – Елемент списку студентських груп

Переглянемо результат, завантажимо сторінку списку студентських груп та перейдемо до списку студентів однієї з груп (рис. 3.56).

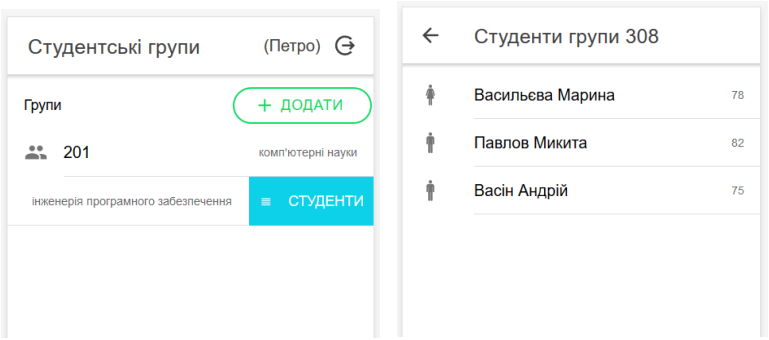

Рисунок 3.56 – Перехід до сторінки списку студентів

Організуємо передачу даних зі сторінки списку студентів до головної сторінки. Для цього скористаємось інструментом публікації та підписки на події в Angular. На сторінку списку студентів додаємо поле вводу, у яке будемо заносити дані, які хочемо передати до головної сторінки (рис. 3.57)

```
      <ion-note slot="end">{{student.rating}}</ion-note>
    </ion-item>
  </ion-list>
  <ion-item>
    <ion-label position="floating">Дані на home</ion-label>
    <ion-input type="text" #dataToHome></ion-input>
  </ion-item>
  <ion-button color="primary"  class="ion-float-right" (click)="passData
(dataToHome.value)">
    <ion-icon slot="start" name="send"></ion-icon> Переслати на home
  </ion-button>
</ion-content>
```

Рисунок 3.57 – Додавання поля вводу до макету students.page.html

Після виконаних змін сторінка студентів матиме такий вигляд (рис. 3.58).

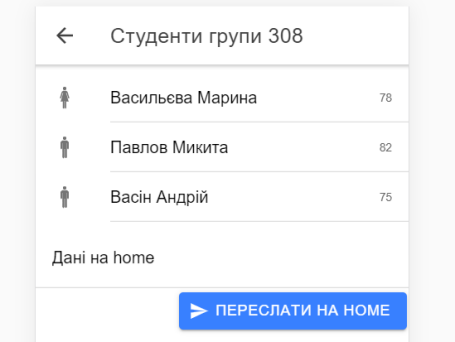

Рисунок 3.58 – Сторінка студентів після додавання поля вводу

У клас students додаємо метод passData, у якому ініціюємо подію, передаємо туди дані та повертаємось на домашню сторінку. Також у конструкторі підключаємо необхідні залежності (рис. 3.59).

```
import {Events, NavController} from '@ionic/angular';

  constructor(private dataGetter: DataGetterService,
              private route: ActivatedRoute,
              private events: Events,
              private navCtrl: NavController) { }

  passData(data: string) {
    this.events.publish( topic: 'students:data', data);
    this.navCtrl.back();
  }
```

Рисунок 3.59 – Зміни у класі students.page.ts

На home-сторінці також додаємо label для відображення даних, переданих зі сторінки студентів (рис. 3.60).

```
                [studentGroup]="group"></app-stud-group>
      </div>
    </ion-list>
    <ion-item text-center>
      <ion-label color="primary">{{dataFromStudents}}</ion-label>
    </ion-item>
</ion-content>
```

Рисунок 3.60 – Додавання label до home.page.html

У конструкторі класу home підписуємось на подію, що ініціюємо на сторінці студентів, та передаємо отримані дані у змінну, яка відображається у полі label (рис. 3.61).

```
import {Events} from '@ionic/angular';

  dataFromStudents: string;

  constructor(private dataGetter: DataGetterService, private events: Events) {
    this.dataGetter.getGroups().subscribe(
        next: (data) => {
          this.groups = data;
        }
    );
    this.userName = this.dataGetter.getUser();
    this.events.subscribe( topic: 'students:data', handlers: data => {
      this.dataFromStudents = data;
    });
  }
```

Рисунок 3.61 – Підписка на подію у home.page.ts

Перевіримо роботу передачі даних на домашню сторінку (рис. 3.62).

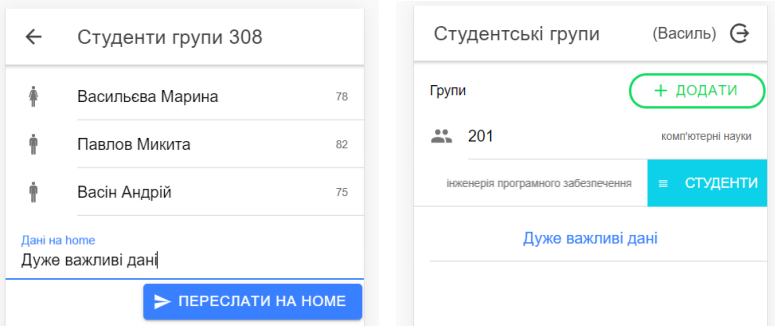

Рисунок 3.62 – Перевірка передачі даних

Виділимо функціонал передачі та повернення даних у окрему сторінку. Для цього створимо сторінку DataSender (рис. 3.63).

```
ionic generate page DataSender
```

Рисунок 3.63 – Створення сторінки

На рис. 3.64 та рис. 3.65 наведено представлення та клас нової сторінки відповідно.

```
<ion-header>
  <ion-toolbar>
    <ion-buttons slot="start">
      <ion-back-button></ion-back-button>
    </ion-buttons>
    <ion-title>Повернення даних</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content>
  <ion-item>
    <ion-label position="floating">Дані на home</ion-label>
    <ion-input type="text" [(ngModel)]="textData"></ion-input>
  </ion-item>
  <ion-button color="primary"  class="ion-float-right" (click)="passData()">
    <ion-icon slot="start" name="sync"></ion-icon> Повернути дані
  </ion-button>
</ion-content>
```

Рисунок 3.64 – data-sender.page.html

```
import { Component, OnInit } from '@angular/core';
import {Events, NavController} from '@ionic/angular';
import {ActivatedRoute, Router} from '@angular/router';

@Component({
  selector: 'app-data-sender',
  templateUrl: './data-sender.page.html',
  styleUrls: ['./data-sender.page.scss'],
})
export class DataSenderPage implements OnInit {

  textData: string;

  constructor(private events: Events,
              private navCtrl: NavController,
              private route: ActivatedRoute) {
    this.textData = this.route.snapshot.paramMap.get('data');
  }
```

```
19
20    ngOnInit() {
21    }
22
23    passData() {
24      this.events.publish( topic: 'students:data', this.textData);
25      this.navCtrl.back();
26    }
27  }
```

Рисунок 3.65 – data-sender.page.ts

Також на сторінці home змінимо label для відображення даних на input, виконаємо початкову передачу даних із домашньої сторінки через параметр маршруту (рис. 3.66–3.67).

```
58      </div>
59    </ion-list>
60    <ion-item text-center>
61      <ion-input type="text" [(ngModel)]="extraData"></ion-input>
62    </ion-item>
63    <ion-button color="primary"  class="ion-float-right" (click)="getData()">
64      <ion-icon slot="start" name="sync"></ion-icon> Передати дані
65    </ion-button>
66  </ion-content>
```

Рисунок 3.66 – home.page.html

```
4   import {Router} from '@angular/router';
5
20    extraData: string;
22    constructor(private dataGetter: DataGetterService,
23                private events: Events,
24                private router: Router) {
25      this.dataGetter.getGroups().subscribe(
26          next: (data) => {
27            this.groups = data;
28          }
29      );
30      this.userName = this.dataGetter.getUser();
31      this.events.subscribe( topic: 'students:data',  handlers: data => {
32        this.extraData = data;
33      });
34    }
52
53    getData() {
54      this.router.navigate( commands: ['/data-sender', {data: this.extraData}]);
55    }
```

Рисунок 3.67 – home.page.ts

Перевіримо передачу даних з головної сторінки та повернення їх назад (рис. 3.68).

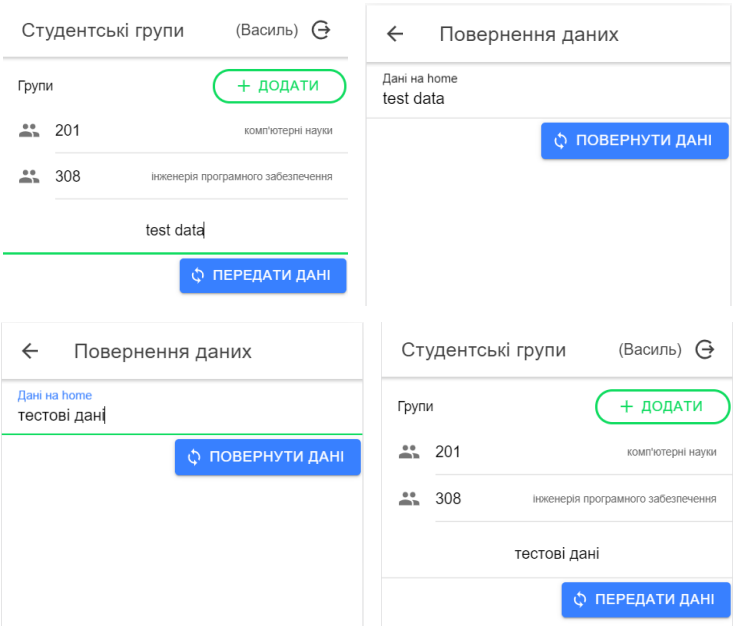

Рисунок 3.68 – Передача та повернення даних

Змінимо спосіб повернення даних на головну сторінку із події та передачу параметра маршруту. Для цього виконаємо невеликі зміни у класі data-sender (рис. 3.69) та у класі home (рис. 3.70).

```
2   import {ActivatedRoute, Router} from '@angular/router';
13    constructor(private route: ActivatedRoute,
14                private router: Router) {
15      this.textData = this.route.snapshot.paramMap.get('data');
16    }
21    passData() {
22      this.router.navigate( commands: ['/home', {data: this.textData}]);
23    }
```

Рисунок 3.69 – Зміни у data-sender.page.ts

```
4   import {ActivatedRoute, Router} from '@angular/router';
22    constructor(private dataGetter: DataGetterService,
23                private events: Events,
24                private router: Router,
25                private route: ActivatedRoute) {
26      this.dataGetter.getGroups().subscribe(
```

```
        next: (data) => {
          this.groups = data;
        }
      );
      this.userName = this.dataGetter.getUser();
      this.extraData = this.route.snapshot.paramMap.get('data');
    }
```

Рисунок 3.70 – Зміни у home.page.ts

У результаті перевірки передачі та повернення даних все має працювати так, як і раніше.

І, наостанок, розглянемо передачу даних із використанням власного сервісу. Створимо новий сервіс (рис. 3.71).

```
ionic generate service servece/DataExchanger
```

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

interface DataElem {
  k: string;
  d: any;
}

@Injectable({
  providedIn: 'root'
})

export class DataExchangerService {

  data: DataElem[] = [];

  constructor() { }

  getData(key: string) {
    const arr = this.data.filter( callbackfn: elem => elem.k === key);
    if (arr.length) {
      return arr[0].d;
    }
    return '';
  }

  setData(key: string, data: any) {
    const arr = this.data.filter( callbackfn: elem => elem.k === key);
    if (arr.length) {
      arr[0].d = data;
    } else {
      this.data.push({
        k: key, d: data
      });
    }
  }
}
```

Рисунок 3.71 – Створення нового сервісу data-exchanger.service.ts

Також виконаємо необхідні зміни у класах home та data-sender відповідно (рис. 3.72–3.73).

```
22  constructor(private dataGetter: DataGetterService,
23              private router: Router,
24              private dataExchanger: DataExchangerService) {
25    this.dataGetter.getGroups().subscribe(
26        next: (data) => {
27          this.groups = data;
28        }
29    );
30    this.userName = this.dataGetter.getUser();
31  }
32
33  ionViewDidEnter() {
34    console.log('home');
35    this.extraData = this.dataExchanger.getData( key: 'myData');
36  }
53
54  getData() {
55    this.dataExchanger.setData( key: 'myData', this.extraData);
56    this.router.navigate( commands: ['/data-sender']);
57  }
```

Рисунок 3.72 – Зміни у home.page.ts

```
13
14  constructor(private dataExchanger: DataExchangerService,
15              private router: Router) {
16    this.textData = this.dataExchanger.getData( key: 'myData');
17  }
21
22  passData() {
23    this.dataExchanger.setData( key: 'myData', this.textData);
24    this.router.navigate( commands: ['/home']);
25  }
```

Рисунок 3.73 – Зміни у data-sender.page.ts

3.5. Html-запити

Розглянемо процедуру отримання даних з віддаленого вебсервісу. Для цього скористаємось механізмом http-сервісів Angular.

Для тестування скористаємось сервісом http://jsonplaceholder.typicode.com. Так, наприклад, за адресою http://jsonplaceholder.typicode.com/posts/10 сервіс повертає публікацію (post) із id = 10 у json-форматі (рис. 3.74).

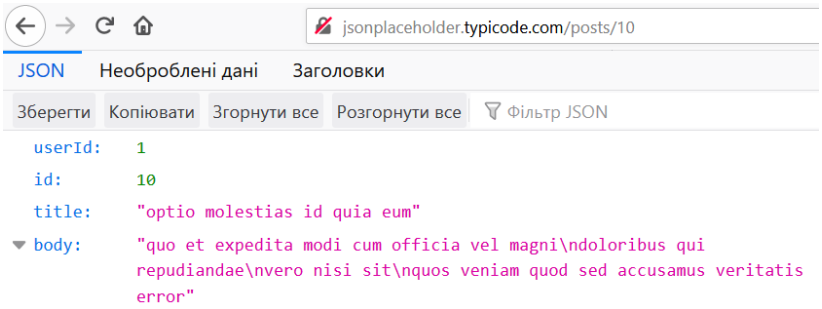

Рисунок 3.74 – Повернення публікації у json-форматі

Тепер отримаємо наведені дані з нашого ionic-застосунку. Для цього створимо нову сторінку (рис. 3.75) та тимчасово зробимо її стартовою (рис. 3.76).

```
ionic generate page HttpTest
```

Рисунок 3.75 – Створення сторінки http-test

```
5   const routes: Routes = [
6     { path: '', redirectTo: 'http-test', pathMatch: 'full' },
```

Рисунок 3.76 – Зміна стартової сторінки у app-routing.module.ts

У файлі app.module.ts виконаємо підключення модуля HttpClientModule, що дозволяє використовувати http-сервіс Angular (рис. 3.77).

```
11  import {HttpClientModule} from '@angular/common/http';
12
13  @NgModule({
14    declarations: [AppComponent],
15    entryComponents: [],
16    imports: [
17        BrowserModule,
18        IonicModule.forRoot(),
19        AppRoutingModule,
20        HttpClientModule
21    ],
```

Рисунок 3.77 – Додавання модуля HttpClientModule до масиву imports

У представлення нової сторінки додаємо кнопку, з натисканням на яку будемо виконувати http-запит на отримання даних (рис. 3.78).

```
<ion-header>
  <ion-toolbar>
    <ion-title>HttpTest</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content>
  <ion-button color="primary" margin (click)="getData()">
    <ion-icon name="cloud-download" margin-end></ion-icon> Виконати http-get
  </ion-button>
</ion-content>
```

Рисунок 3.78 – Додавання кнопки на сторінку в http-test.page.html

У класі http-test реалізуємо метод getData, що виконуватиме http-запит (рис. 3.79).

```
import { Component, OnInit } from '@angular/core';
import {HttpClient} from '@angular/common/http';

@Component({
  selector: 'app-http-test',
  templateUrl: './http-test.page.html',
  styleUrls: ['./http-test.page.scss'],
})
export class HttpTestPage implements OnInit {

  constructor(private http: HttpClient) { }

  ngOnInit() {
  }

  getData() {
    this.http.get( url: 'http://jsonplaceholder.typicode.com/posts/10')
        .subscribe(
          next: data => {
            console.log(data);
          }
    );
  }
}
```

Рисунок 3.79 – Клас http-test.page.ts

Запустимо застосунок та виконаємо реалізований запит (рис. 3.80–3.81).

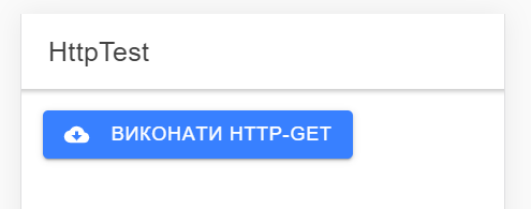

Рисунок 3.80 – Зовнішній вигляд сторінки http-test

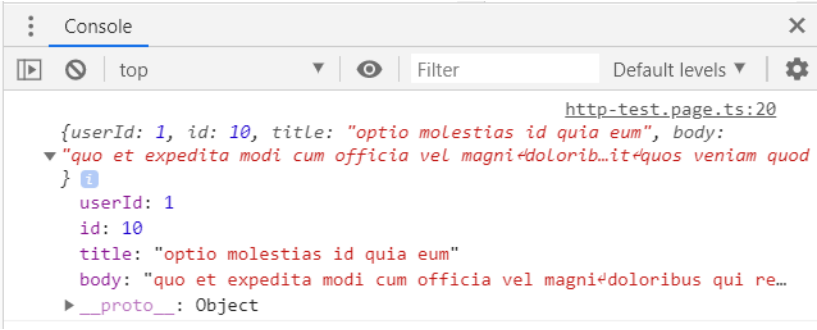

Рисунок 3.81 – Результат виведення результату запиту на консоль

Виконаємо запит для отримання списку публікацій (http://jsonplaceholder.typicode.com/posts/) та розмістимо на сторінці перші 10 у вигляді списку 1(рис. 3.82–3.83).

```
1  import { Component, OnInit } from '@angular/core';
2  import {HttpClient} from '@angular/common/http';
3  import { map } from 'rxjs/operators';
4
5  @Component({
6    selector: 'app-http-test',
7    templateUrl: './http-test.page.html',
8    styleUrls: ['./http-test.page.scss'],
9  })
10 export class HttpTestPage implements OnInit {
11
12   posts: any[];
13
14   constructor(private http: HttpClient) { }
15
16   ngOnInit() {
17   }
18
```

```
19      getData() {
20        this.http.get( url: 'http://jsonplaceholder.typicode.com/posts')
21            .pipe(map(
22                project: (res: Array<any>) => res.filter(
23                    callbackfn: row => row.id < 10
24                )
25            )).subscribe(
26                next: data => {
27                  this.posts = data;
28                }
29            );
30        }
31    }
```

Рисунок 3.82 – Код файлу http-test.page.ts

```
7   <ion-content>
8     <ion-button color="primary" margin (click)="getData()">
9       <ion-icon name="cloud-download" margin-end></ion-icon> Виконати http-get
10    </ion-button>
11    <ion-list>
12      <ion-card *ngFor ="let post of posts">
13        <ion-card-header>
14          <ion-card-title>{{post.title}}</ion-card-title>
15        </ion-card-header>
16        <ion-card-content>
17          {{post.body}}
18        </ion-card-content>
19      </ion-card>
20    </ion-list>
21  </ion-content>
```

Рисунок 3.83 – Код файлу http-test.page.html

Переглянемо отриманий результат (рис. 3.84).

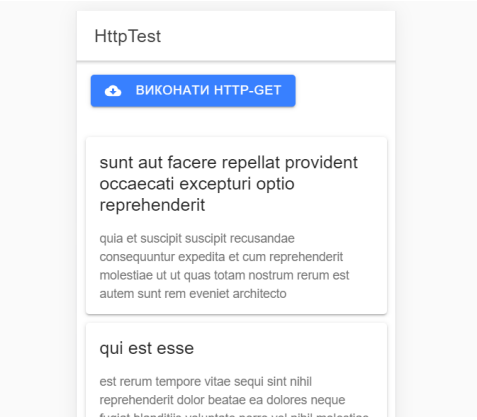

Рисунок 3.84 – Результат виведення списку публікацій

Приберемо кнопку «Виконати http-get» та будемо завантажувати дані разом із завантаженням сторінки. Також реалізуємо механізм оновлення даних за подією pull down (потягнути сторінку вниз). Для цього до представлення додаємо елемент управління refresher (рис. 3.85) та виконуємо деякі зміни у коді класу HttpTest (рис. 3.86).

```
<ion-content>
  <ion-refresher slot="fixed" (ionRefresh)="getData($event)">
    <ion-refresher-content
            pullingIcon="arrow-dropdown"
            pullingText="Pull to refresh"
            refreshingSpinner="circles"
            refreshingText="Refreshing...">
    </ion-refresher-content>
  </ion-refresher>
  <ion-list>
    <ion-card *ngFor ="let post of posts">
```

Рисунок 3.85 – http-test.page.html

```
  ngOnInit() {
    this.getData( refresher: false);
  }

  getData(refresher) {
    this.http.get( url: 'http://jsonplaceholder.typicode.com/posts')
        .pipe(map(
            project: (res: Array<any>) => res.filter(
                callbackfn: row => row.id < 10
            )
        )).subscribe(
            next: data => {
              this.posts = data;
              if (refresher) {
                refresher.target.complete();
              }
            }
    );
  }
}
```

Рисунок 3.86 – http-test.page.ts

Переглянемо отриманий результат (рис. 3.87).

Рисунок 3.87 – Використання refresher для оновлення даних

Виконаємо довантаження нових даних на сторінку у міру їх необхідності. Тобто під час прокручення вмісту сторінки вниз із досяганням кінця даних будемо завантажувати на сторінку нову порцію даних (10 наступних публікацій). Додаємо елемент управління InfiniteScroll (рис. 3.88).

```
6
7   <ion-content>
8     <ion-refresher slot="fixed" (ionRefresh)="refreshData($event)">
9       <ion-refresher-content
10              pullingIcon="arrow-dropdown"
24      </ion-card>
25    </ion-list>
26    <ion-infinite-scroll threshold="100px" (ionInfinite)="addData($event)">
27      <ion-infinite-scroll-content
28              loadingSpinner="bubbles"
29              loadingText="Loading more data...">
30      </ion-infinite-scroll-content>
31    </ion-infinite-scroll>
32  </ion-content>
```

Рисунок 3.88 – Додавання елементу InfiniteScroll

У класі HttpTest додаємо декілька змінних для фіксації кількості поточних даних. Також розділяємо окремо оновлення та додавання нових даних (рис. 3.89).

```
10  export class HttpTestPage implements OnInit {
11
12    private posts: any[];
13
14    private postCount = 0;
15    private postStep = 10;
16
17    constructor(private http: HttpClient) {
18    }
```

```
19
20    ngOnInit() {
21      this.refreshData( refresher: false);
22    }
23
24    refreshData(refresher) {
25      this.posts = [];
26      this.postCount = 0;
27      this.addData(refresher);
28    }
29
30    addData(refresher) {
31      this.http.get( url: 'http://jsonplaceholder.typicode.com/posts')
32          .pipe(map(
33              project: (res: Array<any>) => res.filter(
34                  callbackfn: row => row.id > this.postCount &&
35                      row.id < this.postCount + this.postStep
36              )
37          )).subscribe(
38              next: data => {
39                this.posts = this.posts.concat(data);
40                if (refresher) {
41                  refresher.target.complete();
42                }
43                this.postCount += this.postStep;
44              }
45      );
46    }
47  }
```

Рисунок 3.89 – Зміни у коді http-test.page.ts

Переглядаємо результат, завантажуємо застосунок та переходимо у кінець списку (рис. 3.90).

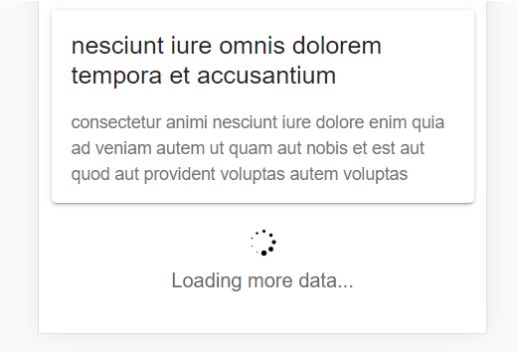

Рисунок 3.90 – Робота елементу InfiniteScroll

РОЗДІЛ 4.
ОРГАНІЗАЦІЯ РОБОТИ ІЗ BACK-END СКЛАДОВОЮ

4.1. Створення back-end складової проєкту

У цій роботі створимо власний back-end сервіс, що відповідатиме за збереження даних та обробку команд нашого застосунку. Для реалізації сервісу обрана мова PHP. Сервіс може бути розміщено в мережі інтернет, або, як у нашому випадку, може бути використано локальний вебсервер. У рамках нашої роботи використано пакет openserver.

У папці ospanel\domains\localhost створимо папку api, а в ній розміщуємо файл index.php. Спочатку реалізуємо тестовий функціонал, що отримуватиме дані від front-end та відправлятиме їх назад (рис. 4.1).

```php
<?php
header( string: 'Content-type: application/json');
header( string: 'Access-Control-Allow-Origin: *');
header( string: 'Access-Control-Allow-Methods: GET, PUT, POST, DELETE,
 OPTIONS');
header( string: 'Access-Control-Allow-Headers: Origin, Content-Type,
 X-Auth-Token , Authorization');

$input = json_decode(file_get_contents( filename: 'php://input'),
  assoc: true);

echo json_encode(
    [
        'get-data' => $_GET,
        'input-data' => $input
    ]
);
```

Рисунок 4.1 – Створення тестового API

Реалізуємо у ionic-застосунку get та post запити до тестового API. Після отримання даних з back-end виводимо їх на консоль (рис. 4.2).

```
ngOnInit() {
  this.refreshData( refresher: false);

  this.http.get( url: 'http://localhost/api/?a=10&b=30').subscribe(
      next: data => console.log(data)
  );
```

```
27      this.http.post( url: 'http://localhost/api/',
28          body: {
29            a: 10,
30            b: 30,
31            c: 'text data'
32          }).subscribe(
33            next: data => console.log(data)
34      );
35    }
```

Рисунок 4.2 – Запити до API з коду http-test.page.ts

Переглянемо результат, якщо він відповідає рис. 4.3, то все виконано правильно.

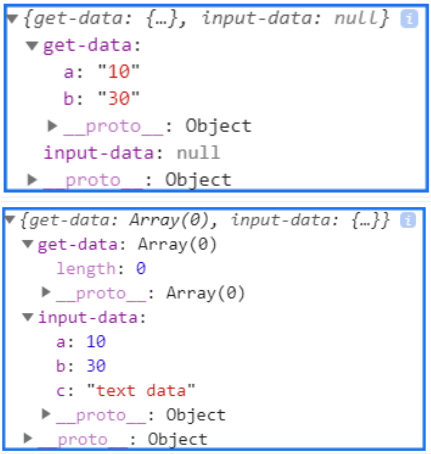

Рисунок 4.3 – Виведення даних API на консоль

Далі переходимо до реалізації роботи із даними студентських груп та студентів. Дані будемо зберігати у БД MySQL. Створимо БД, 3 таблиці та наповнимо їх даними (рис. 4.4–4.7).

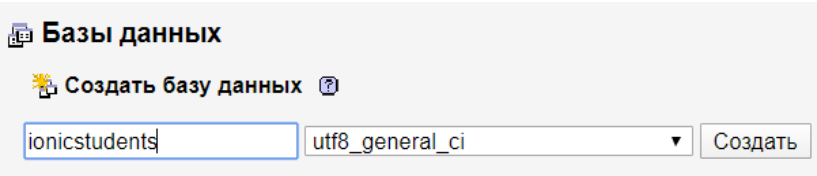

Рисунок 4.4 – Створення БД

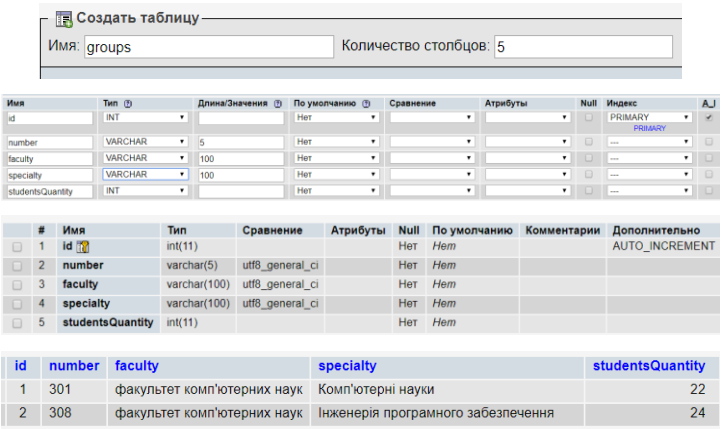

Рисунок 4.5 – Таблиця студентських груп

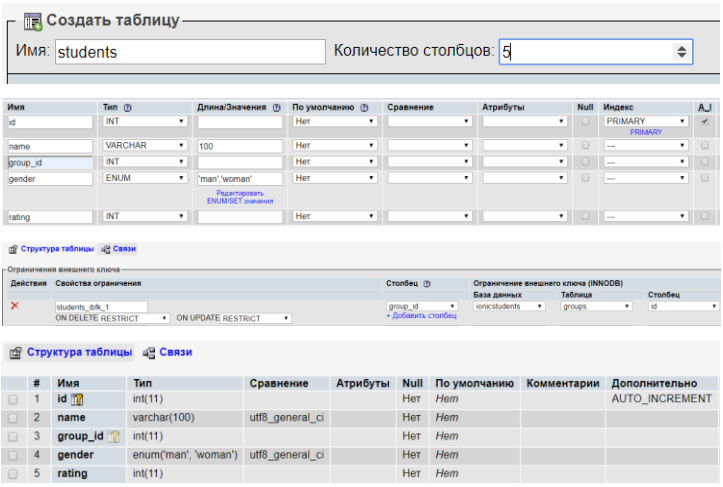

Рисунок 4.6 – Таблиця студентів

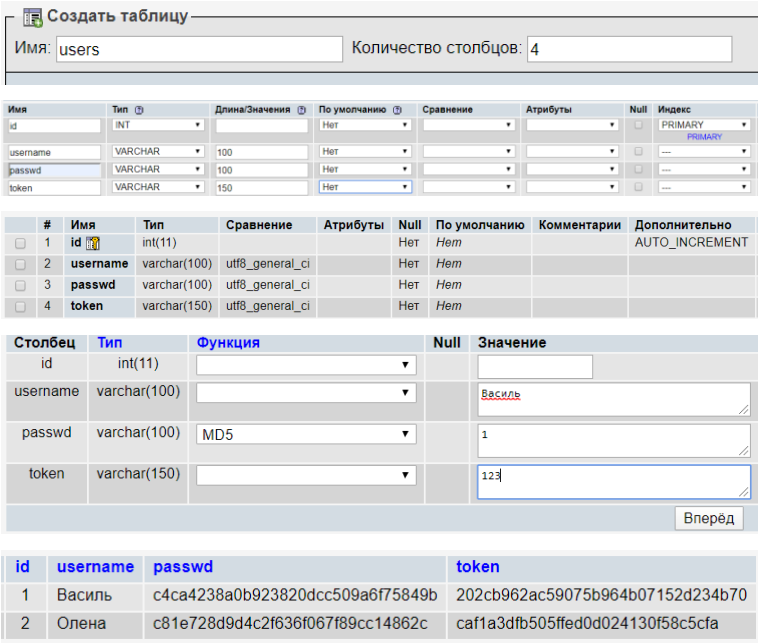

Рисунок 4.7 – Таблиця користувачів

Реалізуємо на back-end аутентифікацію та вибірку списку студентських груп. У папці api поряд із index.php створимо файли config.php, db.php, model.php, controller.php та auth.php. У файлі config.php зберігаємо налаштування підключення до БД (рис. 4.8).

```php
<?php
class Config {
    public static $server = "localhost:3306";
    public static $user = "root";
    public static $pwd = "";
    public static $db = "ionicstudents";
}
```

Рисунок 4.8 – config.php

Файл db.php відповідає за підключення до БД, виконання запиту або команди та повернення результату (рис. 4.9).

```php
<?php

class DB {
    private $link;
    public $err;
    public function connect() {
        $this->link = new \mysqli(
            \Config::$server, \Config::$user, \Config::$pwd, \Config::$db
        );
        if (!$this->link) {
            return false;
        }
        $this->runQuery( sql: "SET NAMES 'uft-8'");
        return true;
    }
    public function disconnect() {
        $this->link->close();
        unset($this->link);
    }
    public function runQuery($sql) {
        if (!$this->link) {
            $this->connect();
        }
        $res = $this->link->query($sql);
        if (!$res) {
            $this->err = $this->link->error;
        }
        return $res;
    }
    public function getArrFromQuery($sql) {
        $res_arr = [];
        $rs = $this->runQuery($sql);
        while($row = $rs->fetch_assoc()) {
            $res_arr[] = $row;
        }
        return $res_arr;
    }
}
```

Рисунок 4.9 – Файл db.php

Файл model.php відповідає за вибірку конкретних даних із БД або виконання конкретних команд. На цьому етапі ми реалізуємо лише один метод для отримання списку груп, але надалі цей клас буде розширюватися (рис. 4.10).

```php
<?php

class Model {
    public static function getGroupsList() {
        return (new DB())->getArrFromQuery(
            sql: "SELECT id, number, faculty, specialty, studentsQuantity
                FROM groups
                order by number");
    }
}
```

Рисунок 4.10 – Файл model.php

У файлі auth.php реалізуємо процедуру логіну користувача і видачу нового token-а, процедуру перевірки token-а під час виконання подальших запитів на вибірку та модифікацію даних (рис. 4.11).

```php
<?php

class Auth {
    public static function getUserToken($user) {
        $res = (new DB())->getArrFromQuery(
            sql: "select id from users where username='".$user['username']."'
            and passwd=md5('".$user['passwd']."')"
        );
        if (count($res) > 0) {
            $id = $res[0]['id'];
            $token = bin2hex(random_bytes( length: 64));
            $res2 = (new DB())->runQuery(
                sql: "update users set token='$token' where id=$id"
            );
            if ($res2) {
                return ['token' => $token];
            }
        }
        return ['error' => 'Username or password is incorrect'];
    }

    public static function checkToken($token) {
        $res = (new DB())->getArrFromQuery(
            sql: "select id from users where token='$token'"
        );
        if (count($res) > 0) {
            return true;
        }
        return false;
    }
}
```

Рисунок 4.11 – Файл auth.php

У файлі controller.php визначаємо, яку саме дію хоче виконати користувач, делегуємо виконання відповідному методу класу model та повертаємо результат користувачу у вигляді json (рис. 4.12).

```php
<?php
class Controller {
    private $data;
    private $action;
    private $protectedActions = ['get-groups'];

    function __construct() {
        $this->action = $_GET['action'];
        $this->data = json_decode(file_get_contents( filename: 'php://input'), assoc: true);
    }
```

```
    function run() {
        $res = [];
        if (in_array($this->action, $this->protectedActions)
            && !Auth::checkToken($_GET['token'])) {
            return ['error' => 'authentication failed !'];
        }
        switch ($this->action) {
            case 'login':
                $res = Auth::getUserToken($this->data);
                break;
            case 'get-groups':
                $res = Model::getGroupsList();
                break;
            default:
                $res = ['error' => 'this route is incorrect !'];
                break;
        }
        echo json_encode($res);
    }
}
```

Рисунок 4.12 – Файл controller.php

І, нарешті, розглянемо index.php, що є своєрідною точкою входу до back-end api (рис. 4.13).

```
<?php
require 'config.php';
require 'db.php';
require 'model.php';
require 'auth.php';
require 'controller.php';

header( string: 'Content-type: application/json');
header( string: 'Access-Control-Allow-Origin: *');
header( string: 'Access-Control-Allow-Methods: GET, PUT, POST, DELETE,
 OPTIONS');
header( string: 'Access-Control-Allow-Headers: Origin, Content-Type,
 X-Auth-Token , Authorization');

$input = json_decode(file_get_contents( filename: 'php://input'), assoc: true);

(new Controller())->run();
```

Рисунок 4.13 – Файл index.php

Наступний крок – зміна ionic-застосунку для взаємодії із back-end api. Почнімо із сервісу DataGetterService, що відповідає за отримання даних та передачу їх до необхідної сторінки. Реалізуємо механізми взаємодії із api автентифікацією та отриманням списку студентських груп (рис. 4.14).

```
export class DataGetterService {
  baseUrl = 'http://localhost/api/';
  groups = [];
  students = [];
  users = [];

  constructor(private http: HttpClient) {
  }

  private userName = '';
  private token = '';

  checkUser(user) {
    return this.http.post<any>( url: this.baseUrl + '?action=login',
user);
  }

  getUser() {
    return this.userName;
  }

  setUser(name: string) {
    this.userName = name;
  }

  setToken(token: string) {
    this.token = token;
  }

  getGroups() {
    return this.http.get<any>( url: this.baseUrl +
'?action=get-groups&token=' + this.token);
  }

  getStudents() {}
}
```

Рисунок 4.14 – Файл service/data-getter.service.ts

У файлі app-routing.module.ts повертаємо сторінку login як стартову (рис. 4.15).

```
const routes: Routes = [
  { path: '', redirectTo: 'login', pathMatch: 'full' },
  {
```

Рисунок 4.15 – Редірект на сторінку login

У представлення сторінки login додаємо поле вводу для пароля користувача (рис. 4.16).

```
<ion-item>
  <ion-label position="floating">Ім'я користувача</ion-label>
  <ion-input type="text" [(ngModel)]="userName"></ion-input>
</ion-item>
<ion-item>
  <ion-label position="floating">Пароль</ion-label>
  <ion-input type="password" [(ngModel)]="passWord"></ion-input>
</ion-item>
```

Рисунок 4.16 – Зміни у представленні login.page.html.

Далі доробимо клас LoginPage для підтримки пароля та взаємодії із сервісом back-end (рис. 4.17).

```
export class LoginPage implements OnInit {
  userName: string;
  passWord: string;
  login() {
    this.dataGetter.checkUser( user: {
      username: this.userName,
      passwd: this.passWord
    }).subscribe(
        next: result => {
          if (result.hasOwnProperty( v: 'error')) {
            this.userNotExistAlert(result.error);
          } else {
            if (result.hasOwnProperty( v: 'token')) {
              this.dataGetter.setUser(this.userName);
              this.dataGetter.setToken(result.token);
              this.router.navigate( commands: ['/home']);
            } else {
            } else {
              this.userNotExistAlert( massage: 'Unexpected error');
            }
          }
        }
    );
  }

  async userNotExistAlert(massage) {
    const alert = await this.alertController.create({
      header: 'Увага !',
      subHeader: 'Помилка аутентифікації',
      message: massage,
      buttons: ['OK']
    });

    await alert.present();
  }
```

Рисунок 4.17 – Зміни у login.page.ts

Перевіримо роботу застосунку, виконаємо вхід та перегляд списку студентських груп (рис. 4.18). Звернімо увагу, що редагування даних поки не працює.

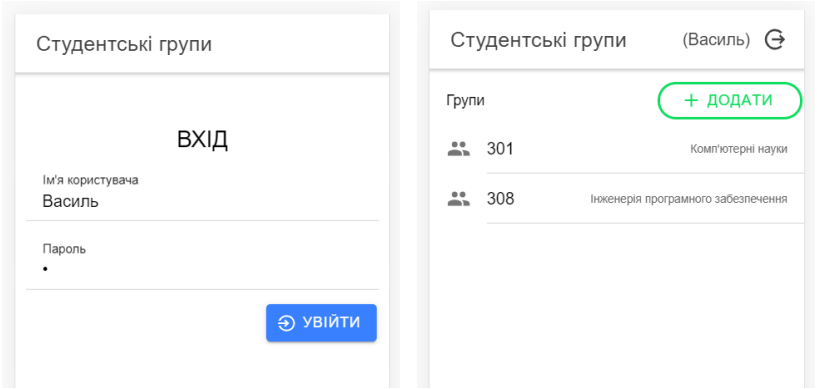

Рисунок 4.18 – Перевірка роботи за стосунку

4.2. Реалізація функцій модифікації даних

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. 4.19) та ініціалізуємо його в ngOnInit компоненти stud-group (рис. 4.20).

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

Рисунок 4.19 – 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   }
```

Рисунок 4.20 – ngOnInit класу компоненти stud-group

У back-end API у класі моделі реалізуємо метод зміни студентської групи (рис. 4.21) та обробимо відповідну подію у класі контролера. Також додаємо новий маршрут до масиву захищених (рис. 4.22).

```
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   }
```

Рисунок 4.21 – 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;
```

Рисунок 4.22 – Доповнення методу run() класу контролера API

До сервісу data-getter.service.ts застосунку додаємо метод editGroup, що взаємодіятиме із API (рис. 4.23).

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

Рисунок 4.23 – Додавання методу до сервісу data-getter.service.ts

У компоненті stud-group також реалізуємо метод збереження змін (рис. 4.24).

```
16
17      constructor(private dataGetter: DataGetterService) {}
43
44      saveGroup() {
45        this.dataGetter.editGroup(this.studentGroup).subscribe(
46            next: data => console.log(data)
47        );
48      }
```

Рисунок 4.24 – Додавання методу до компоненти stud-group

І, нарешті, додаємо до представлення компоненти stud-group кнопку, за якою будемо виконувати збереження даних (рис. 4.25).

```
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>
```

Рисунок 4.25 – 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. 4.26–4.27).

```
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      }
```

Рисунок 4.26 – 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;
```

Рисунок 4.27 – Доповнення контролеру API

Реалізовуємо необхідні методи у сервісі data-getter.service.ts (рис. 4.28).

```
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  }
```

Рисунок 4.28 – Методи addGroup та delGroup сервісу data-getter

У класі Home також корегуємо методи додавання та видалення елементу (рис. 4.29).

```
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      }
```

Рисунок 4.29 – Методи addGroup та delete у класі home.page.ts

Також незначних змін потребує представлення home.page.html – як параметр методу delete передаємо не індекс масиву, а елемент (рис. 4.30).

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

Рисунок 4.30 – 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 ідентифікатора групи під час переходу за посиланням зі сторінки home (рис. 4.31).

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

Рисунок 4.31 – Зміна посилання переходу до списку студентів

На back-end виконаємо доробку моделі – додаємо метод отримання студентів певної групи (рис. 4.32), та доробку контролера – обробка відповідного action та виклик доданого до моделі методу (рис. 4.33).

```
33
34      public static function getStudents($groupId) {
35          return (new DB())->getArrFromQuery(
36                  sql: "SELECT id, name, gender, rating
37                  FROM `students`
38                  where group_id=$groupId order by name");
39      }
```

Рисунок 4.32 – Back-end API – зміни у model.php

```
5       private $protectedActions = ['get-groups', 'edit_group', 'add_group',
'del_group', 'get-students'];
46              case 'get-students':
47                  $res = Model::getStudents($_GET['group']);
48                  break;
```

Рисунок 4.33 – Back-end API – зміни у controller.php

У сервісі отримання даних від back-end DataGetterService додаємо метод для отримання списку студентських групи (рис. 4.34).

```
68
69    getStudents(id: number) {
70      return this.http.get<any>(
71            url: this.baseUrl + `?action=get-students&group=${id}&token=$
{this.token}`
72      );
73    }
```

Рисунок 4.34 – Метод getStudents сервісу DataGetterService

У класі сторінки списку студентів отримуємо список студентів відповідно до доданого у DataGetterService методу getStudents (рис. 4.35).

```
3   import {ActivatedRoute, Router} from '@angular/router';
10  export class StudentsPage implements OnInit {
11    grpid: number;
12    grpnumb: string;
13    students: any[];
14
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  }
```

Рисунок 4.35 – Зміни у students.page.ts

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

4.3. Використання Firebase

Backend as a Service (BaaS) – модель, що дозволяє розробникам веб- та мобільних застосунків зв'язати їх застосування із серверним хмарним сховищем і API, а також надає такі функції, як управління користувачами, повідомлення-оповіщення, інтеграція зі службами соціальних мереж. Звідси інша назва терміна – mBaaS, мобільний back-end як послуга. Перераховані вище послуги надаються через використання інтерфейсів прикладного програмування (API).

Реалізація back-end API – часозатратний і трудомісткий для розробників процес. Ідея BaaS у тому, що замість того, щоб самим розробляти і надалі підтримувати серверні сервіси, можна скористатися готовими, набір яких разом формує необхідний універсальний кросплатформний back-end для будь-якого проєкту.

Серед найкращих і найбільш популярних BaaSs можна виділити Firebase від компанії Google. Firebase служить базою даних, яка змінюється в реальному часі і зберігає дані в JSON. Будь-які зміни в базі даних відразу синхронізуються між усіма клієнтами або девайсами, які використовують одну і ту ж базу даних. Разом зі сховищем, Firebase також надає призначену для користувача аутентифікацію, і тому всі дані передаються через захищене з'єднання SSL. Ми можемо вибрати будь-яку комбінацію email і пароля для автентифікації, будь-то Facebook, Twitter, GitHub, Google або щось інше.

Для використання Firebase для збереження даних нашого проєкту відкриємо сайт ресурсу https://firebase.google.com, перейдемо у консоль (рис. 4.36) та створимо новий проєкт (рис. 4.37).

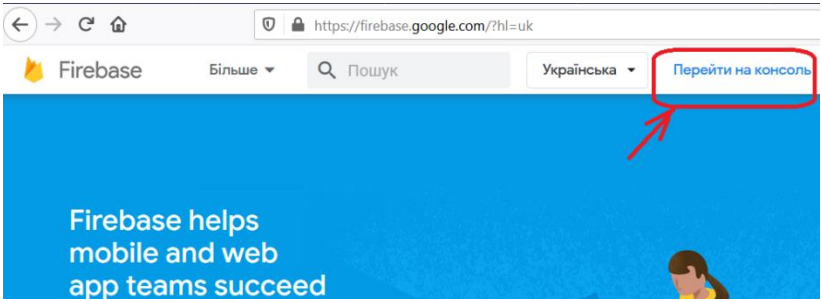

Рисунок 4.36 – Офіційний сайт Firebase

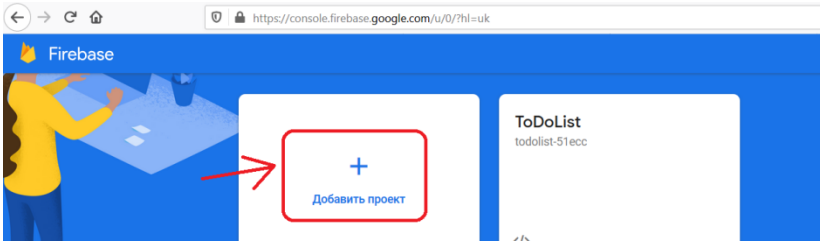

Рисунок 4.37 – Створення нового проєкту

Вказуємо назву проєкту (рис. 4.38).

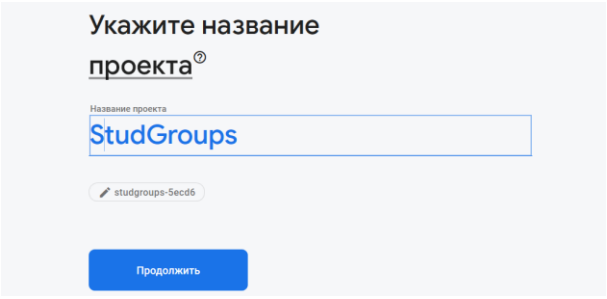

Рисунок 4.38 – Уведення назви проєкту

Після створення проєкту необхідно додати до нього застосунок, що буде його використовувати, обравши необхідний тип – iOS, Android або Web. Для гібридного застосунку ionic на базі angular обираємо вебзастосунок (рис. 4.39).

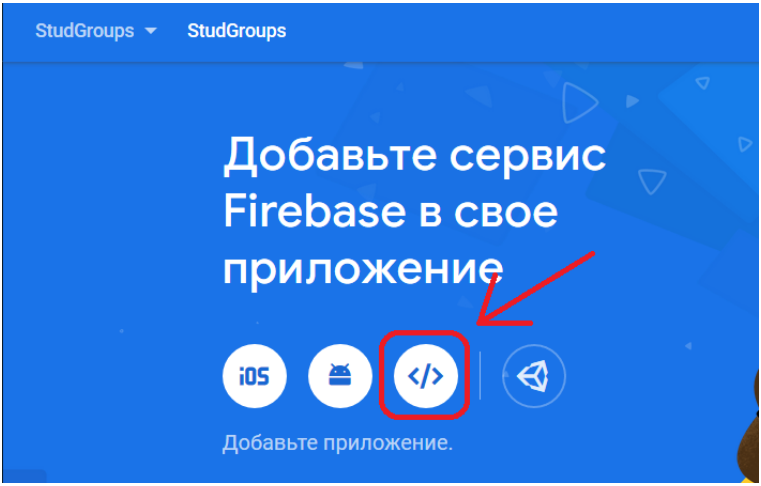

Рисунок 4.39 – Додавання застосунку до проєкту

Вводимо псевдонім застосунку та натискаємо «зареєструвати застосунок» (рис. 4.40).

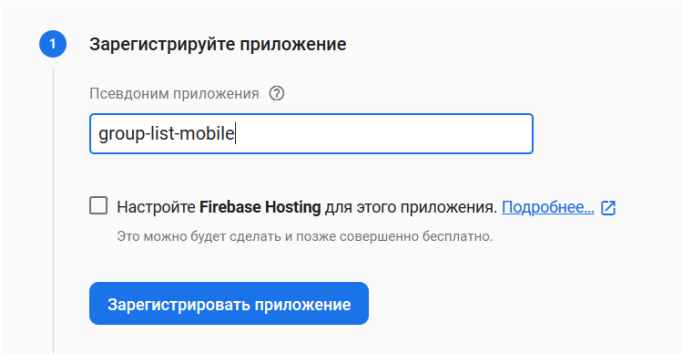

Рисунок 4.40 – Реєстрація застосунку

Із наступного вікна копіюємо поля об’єкта firebaseConfig – вони нам знадобляться у майбутньому під час налаштування підключення до Firebase із застосунку «Список студентських груп» (рис. 4.41).

Рисунок 4.41 – Дані, необхідні на налаштування підключення до Firebase

Далі у нашому проєкті Firebase створимо базу даних для збереження списку студентських груп (рис. 4.42–4.43).

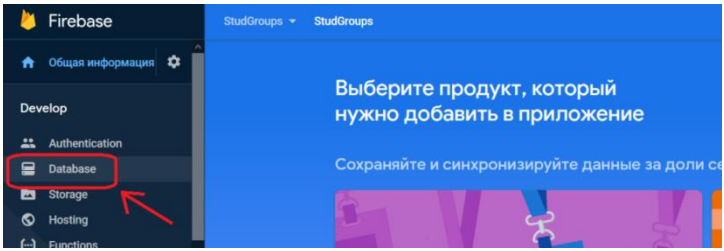

Рисунок 4.42 – Додавання бази даних

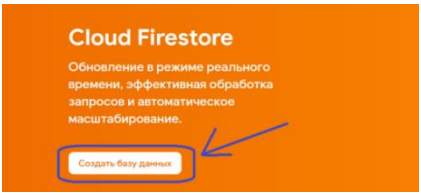

Рисунок 4.43 – Створення бази даних

На першому етапі для можливості роботи зі створеною БД без проходження етапу аутентифікації обираємо «Запустити у тестовому режимі» (рис. 4.44).

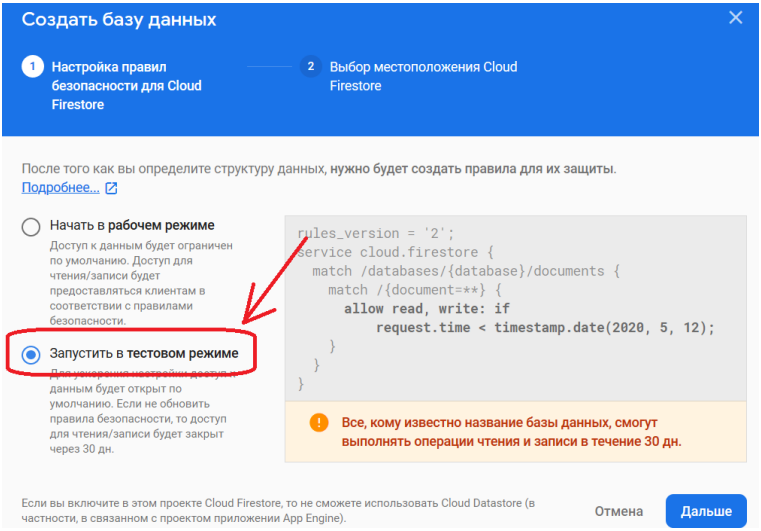

Рисунок 4.44 – Запуск у тестовому режимі

Перейдемо до створеної бази даних та додамо нову колекцію groups, що ідентична таблиці реляційної БД (рис. 4.45–4.46).

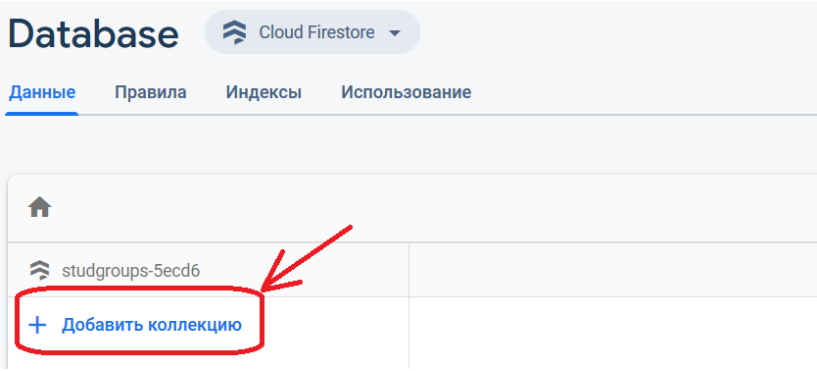

Рисунок 4.45 – Додавання нової колекції groups

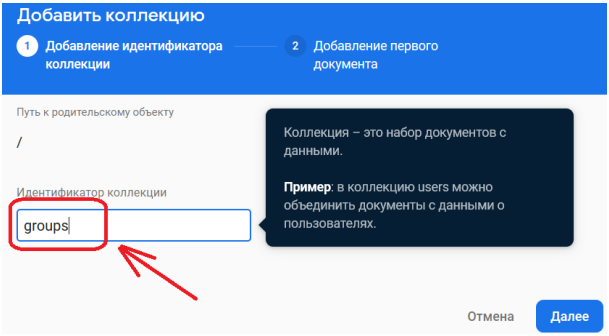

Рисунок 4.46 – Введення ідентифікатора колекції

Далі буде запропоновано ввести новий документ колекції (рядок таблиці). Зробимо це (рис. 4.47).

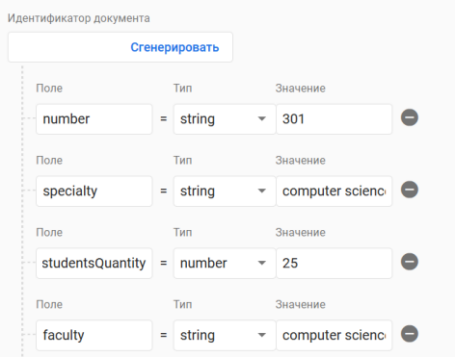

Рисунок 4.47 – Додавання нового документа

Після виконання вищенаведених операцій маємо такий вигляд БД (рис. 4.48).

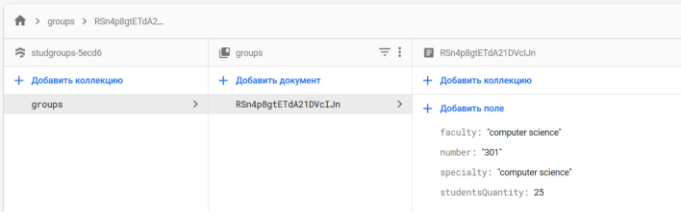

Рисунок 4.48 – Стан БД після додавання колекції та документа

Додаємо до колекції ще хоча б один, а краще декілька документів (рис. 4.49).

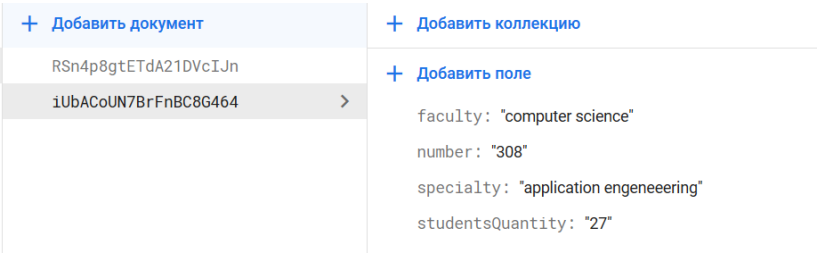

Рисунок 4.49 – Додавання документа до колекції

Далі переходимо до реалізації підключення до бази даних Firebase із нашого ionic-застосунку. Починаємо із додавання до проєкту ionic-бібліотеки angular firebase. Для цього, перебуваючи у директорії проєкту, у командному рядку виконуємо «npm install @angular/fire firebase --save» (рис. 4.50).

```
›npm install @angular/fire firebase --save
```

Рисунок 4.50 – Встановлення angular firebase

Після завершення встановлення, перейдемо до файлу src/ environments/environment.ts та вставимо скопійований на рис. 4.41 код, привівши його до такого вигляду (рис. 4.51).

```
5   export const environment = {
6     production: false,
7     firebase: {
8       apiKey: 'AIzaSyA7gVADz1EkjpZx368NEfGniUNjpqdrOao',
9       authDomain: 'studgroups-5ecd6.firebaseapp.com',
10      databaseURL: 'https://studgroups-5ecd6.firebaseio.com',
11      projectId: 'studgroups-5ecd6',
12      storageBucket: 'studgroups-5ecd6.appspot.com',
13      messagingSenderId: '12724443661',
14      appId: '1:12724443661:web:6393dc1131fee328931344'
15    }
16  };
17
```

Рисунок 4.51 – Налаштування підключення до Firebase

У файлі \src\app\app.module.ts виконаємо необхідні підключення для подальшої роботи із angular firebase (рис. 4.52).

```
import { AppComponent } from './app.component';
import { AppRoutingModule } from './app-routing.module';
import {HttpClientModule} from '@angular/common/http';
import {AngularFireModule} from '@angular/fire';
import {environment} from '../environments/environment';
import {AngularFireStorageModule} from '@angular/fire/storage';
import {AngularFirestore} from '@angular/fire/firestore';

@NgModule({
  declarations: [AppComponent],
  entryComponents: [],
  imports: [
      BrowserModule,
      IonicModule.forRoot(),
      AppRoutingModule,
      HttpClientModule,
      AngularFireModule.initializeApp(environment.firebase),
      AngularFireStorageModule
  ],
  providers: [
    StatusBar,
    SplashScreen,
    { provide: RouteReuseStrategy, useClass: IonicRouteStrategy },
    AngularFirestore
  ],
```

Рисунок 4.52 – Зміни у app.module.ts

Також виконаємо невеликі доповнення до файлу tsconfig.json у розділі compilerOptions (рис. 4.53).

```
  "compilerOptions": {
    "baseUrl": "./",
    "outDir": "./dist/out-tsc",
    "sourceMap": true,
    "declaration": false,
    "module": "esnext",
    "moduleResolution": "node",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "importHelpers": true,
    "target": "es2015",
    "typeRoots": [
      "node_modules/@types"
    ],
    "lib": [
      "es2018",
      "dom"
    ],
    "skipLibCheck": true
  },
```

Рисунок 4.53 – Зміни у файлі tsconfig.json

Створимо сервіс fire-data-getter для роботи із даними FireStore (рис. 4.54), що повертатиме список студентських груп через метод getGroups (рис. 4.55).

```
>>ionic g service services/fire-data-getter
```

Рисунок 4.54 – Створення сервісу fire-data-getter

```
import { Injectable } from '@angular/core';
import {AngularFirestore} from '@angular/fire/firestore';
import {Observable} from 'rxjs';
import {map} from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})

export class FireDataGetterService {

  groups: Observable<any[]>;

  constructor(private readonly afs: AngularFirestore) {
    const groupsCollection = afs.collection( path: 'groups');
    this.groups = groupsCollection.snapshotChanges().pipe(
        map( project: actions => actions.map( callbackfn: a => {
          const data = a.payload.doc.data();
          const id = a.payload.doc.id;
          return { id, ...data };
        }))
    );
  }

  getGroups() {
    return this.groups;
  }
}
```

Рисунок 4.55 – Код сервісу fire-data-getter

Перевіримо роботу сервісу, підключивши його до сторінки логіну та вивівши на консоль список студентських груп (рис. 4.56).

```
  constructor(
      private router: Router,
      private dataGetter: DataGetterService,
      public alertController: AlertController,
      private fireDataGetter: FireDataGetterService) {}

  ngOnInit() {
    this.fireDataGetter.getGroups().subscribe(
        next: data => console.log(data)
    );
  }
```

Рисунок 4.56 – Зміни у \src\app\login\login.page.ts

Запустимо наш застосунок та перевіримо вивід даних (рис. 4.57).

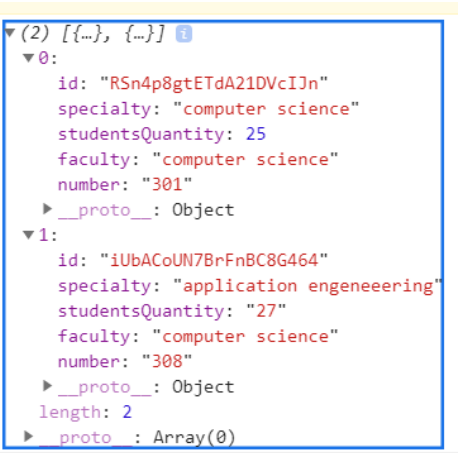

Рисунок 4.57 – Перевірка виводу списку груп на консоль

Виконаємо виведення отриманого з FireStore списку студентських груп на сторінці home нашого застосунку. Збереження та роботу із користувачами з боку Firebase ми поки не налаштовували, тому відключимо також тимчасово аутентифікацію і з боку клієнтського застосунку. Для цього змінимо метод onInit на сторінці login (рис. 4.58).

```
16    constructor(
17        private router: Router,
18        private dataGetter: DataGetterService,
19        public alertController: AlertController) {}
20
21    ngOnInit() {
22      this.dataGetter.setUser('FakeUser');
23      this.router.navigate( commands: ['/home']);
24    }
```

Рисунок 4.58 – Зміни у login.page.ts

Тепер виконаємо підключення до сервісу та отримання даних у home.page.ts (рис. 4.59).

```
import {FireDataGetterService} from '../services/fire-data-getter.service';

  constructor(private dataGetter: DataGetterService,
              private router: Router,
              private dataExchanger: DataExchangerService,
              private fireData: FireDataGetterService) {
    this.fireData.getGroups().subscribe(
        next: data => this.groups = data
    );
    this.userName = this.dataGetter.getUser();
  }
```

Рисунок 4.59 – Зміни у home.page.ts

Запускаємо застосунок та перевіряємо відображення списку студентських груп на сторінці (рис. 4.60).

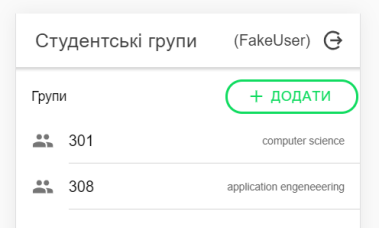

Рисунок 4.60 – Виведення списку студентських груп

4.4. Модифікація даних Firebase

Реалізуємо збереження списку студентів у нашій БД. Для цього відкриємо консоль Firebase (https://console.firebase.google.com), перейдемо до нашої БД на вкладці Database та додамо колекцію в одну зі студентських груп (рис. 4.61).

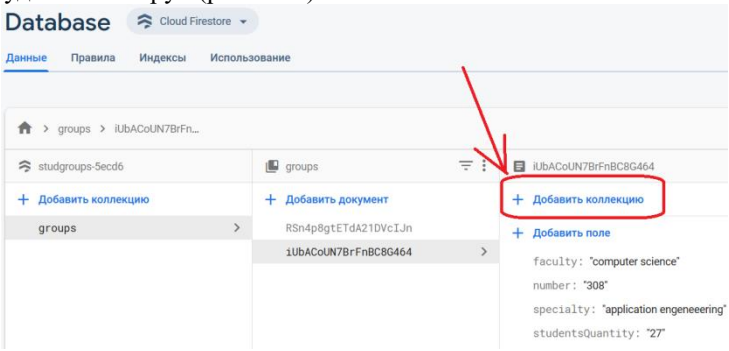

Рисунок 4.61 – Додавання колекції у документ

Вказуємо ідентифікатор для колекції (рис. 4.62) та додаємо перший елемент колекції (рис. 4.63).

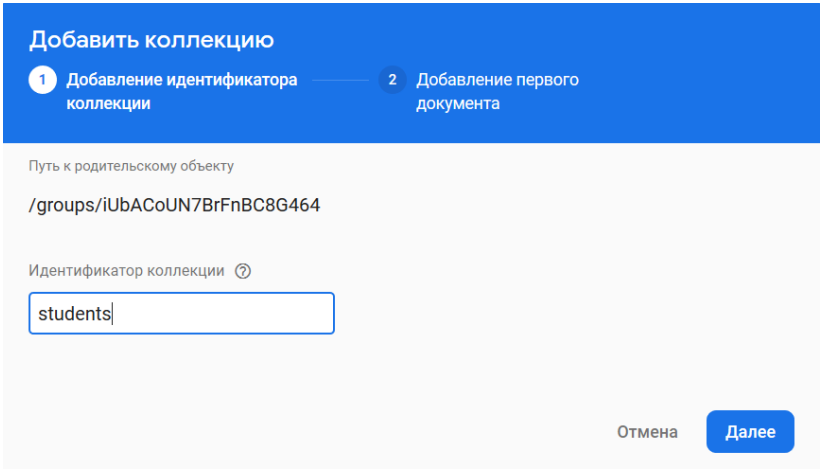

Рисунок 4.62 – Ідентифікатор колекції

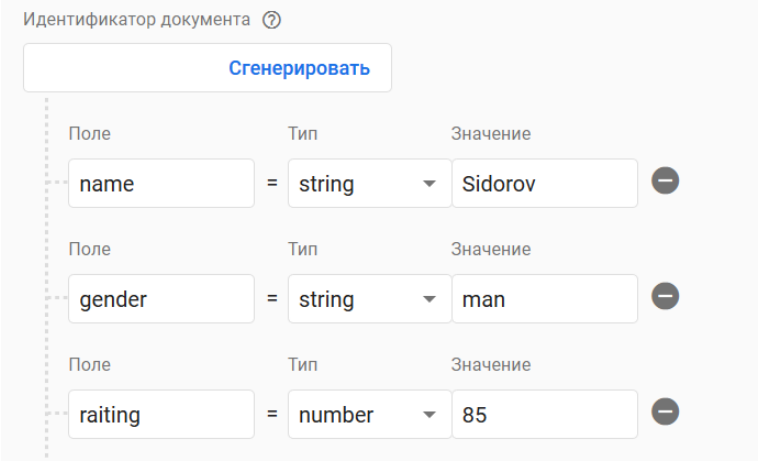

Рисунок 4.63 – Додавання елементу колекції

Аналогічно додаємо ще декілька елементів (студентів) до цієї та іншої групи (рис. 4.64).

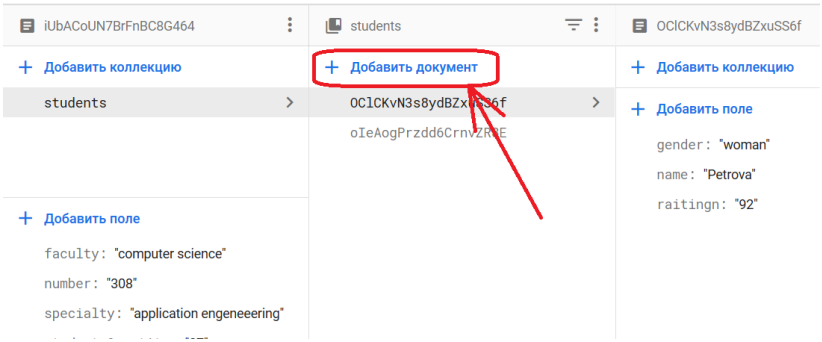

Рисунок 4.64 – Додавання декількох студентів

Перейдемо до коду клієнтського застосунку та виконаємо виведення списку студентів. Для цього спочатку додаємо відповідний метод до сервісу fire-data-getter (рис. 4.65).

```
28
29      getStudents(id) {
30          return this.afs
31              .doc( path: 'groups/' + id)
32              .collection( path: 'students')
33              .valueChanges();
34      }
```

Рисунок 4.65 – Додавання методу
для виведення списку студентів

Тепер виконуємо звернення до цього методу під час завантаження сторінки списку студентів (рис. 4.66).

```
4   import {FireDataGetterService} from '../services/fire-data-getter.service';
5
11  export class StudentsPage implements OnInit {
12    grpid: string;
13    grpnumb: string;
14    students: any[];
15
16    constructor(private dataGetter: DataGetterService,
17                private route: ActivatedRoute,
18                private router: Router,
19                private fireData: FireDataGetterService) {
20      this.grpid = this.route.snapshot.paramMap.get('id');
21    }
22
```

```
23  ngOnInit() {
24    this.grpnumb = this.route.snapshot.paramMap.get('grpnumb');
25    this.fireData.getStudents(this.grpid).subscribe(
26        next: data => {
27          this.students = data;
28        }
29    );
30  }
```

Рисунок 4.66 – Отримання списку студентів

Перейдемо до сторінки списку студентів та переглянемо результат (рис. 4.67).

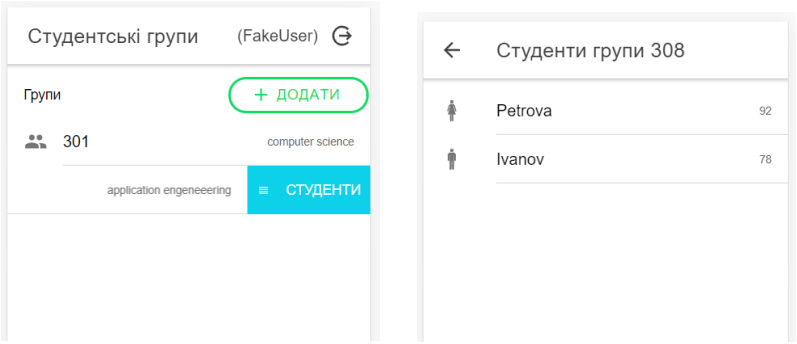

Рисунок 4.67 – Виведення списку студентів групи

Реалізуємо також команди модифікації даних студентських груп. Почнімо із операції зміни даних. Реалізуємо відповідний метод у сервісі fire-data-getter (рис. 4.68).

```
36  editGroup(group) {
37      return this.afs
38          .doc( path: 'groups/' + group.id)
39          .update( data: {
40              number: group.number,
41              specialty: group.specialty,
42              faculty: group.faculty,
43              studentsQuantity: group.studentsQuantity
44          });
45  }
```

Рисунок 4.68 – Реалізація методу зміни даних студентської групи

В інтерфейсі StudGroup у data-getter.service.ts змінимо тип поля id з number на string (рис. 4.69).

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

Рисунок 4.69 – Зміна типу поля id у інтерфейсі StudGroup

Виконаємо звернення до створеного методу у компоненті stud-group.component.ts (рис. 4.70).

```
3   import {FireDataGetterService} from '../../services/fire-data-getter.service';
4
17
18    constructor(private dataGetter: DataGetterService,
19                private fireData: FireDataGetterService) {}
20
45
46    saveGroup() {
47      this.fireData.editGroup(this.studentGroup);
48    }
```

Рисунок 4.70 – Виклик методу editGroup за натисканням на кнопки у компоненті

Перейдемо до застосунку та змінимо номер в одній зі студентських груп (рис. 4.71).

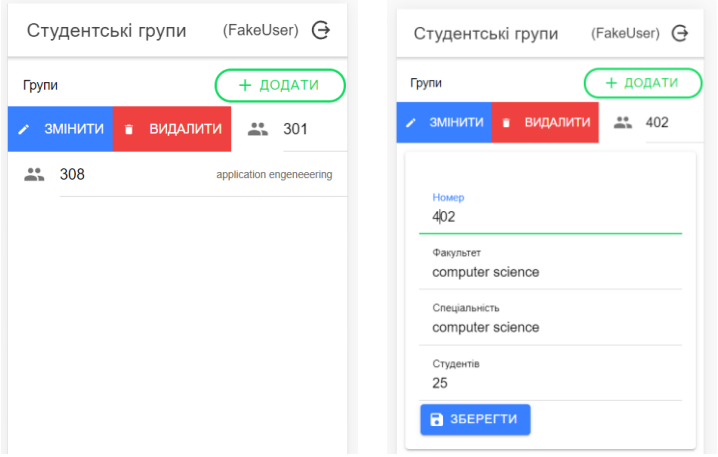

Рисунок 4.71 – Зміна номера групи

Зберігаємо зміни та перевіряємо їх у консолі Firebase (рис. 4.72).

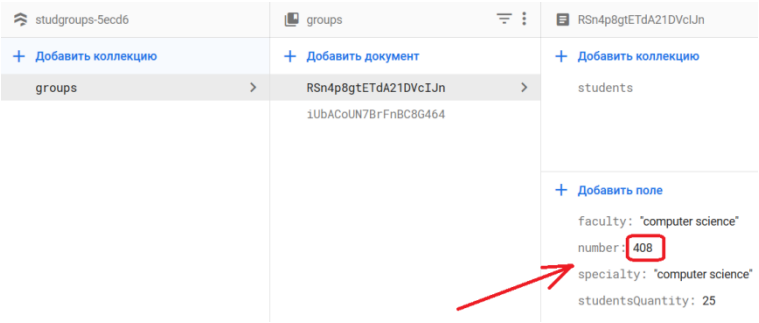

Рисунок 4.72 – Перевірка виконаних змін у консолі

Виконаємо реалізацію функції додавання нової групи. Додаємо метод addGroup до нашого сервісу fire-data-getter (рис. 4.73).

```
46
47      addGroup(group) {
48          this.afs.collection( path: 'groups')
49              .add({
50                  number: group.number,
51                  specialty: group.specialty,
52                  faculty: group.faculty,
53                  studentsQuantity: group.studentsQuantity
54              });
55      }
```

Рисунок 4.73 – Метод addGroup

Виконуємо звернення до методу на сторінці home.page.ts у результаті отримання даних від компоненти студентських груп (рис. 4.74).

```
51
52      addGroup(group) {
53        this.fireData.addGroup(group);
54        this.showNew = false;
55      }
```

Рисунок 4.74 – Зміни у home.page.ts

Додаємо нову групу (рис. 4.75) та перевіряємо результат у консолі Firebase (рис. 4.76).

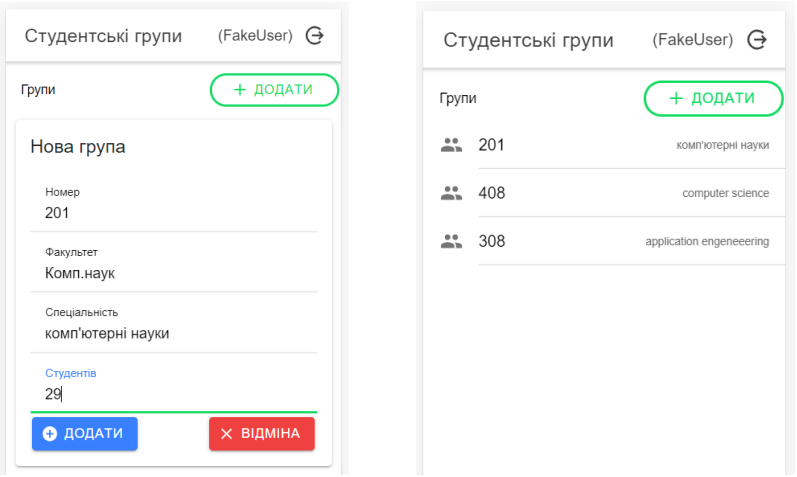

Рисунок 4.75 – Додавання нової групи

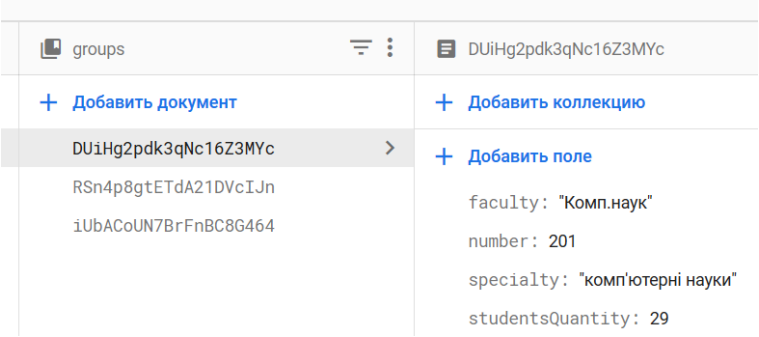

Рисунок 4.76 – Перевірка операції додавання групи

Реалізуємо останню операцію – видалення студентської групи. Також починаємо із відповідного методу сервісу fire-data-getter (рис. 4.77).

```
56
57      delGroup(group) {
58          return this.afs
59              .doc( path: 'groups/' + group.id)
60              .delete();
61      }
```

Рисунок 4.77 – Метод видалення студентської групи

Звертаємось до реалізованого методу із відповідної події на сторінці home.page.ts (рис. 4.78).

```
43
44    delete(group) {
45      this.fireData.delGroup(group);
46    }
```

Рисунок 4.78 – Звернення до методу видалення студентської групи

Аналогічно до зміни та додавання, видаляємо одну з груп та перевіряємо результат у консолі Firebase.

4.5. Автентифікації користувачів Firebase

Тепер виконаємо у нашому застосунку реалізації аутентифікації користувачів. Запустимо консоль Firebase, відкриємо наш проєкт, перейдемо на вкладку Authentication та натискаємо «Налаштувати засіб входу» (рис. 4.79).

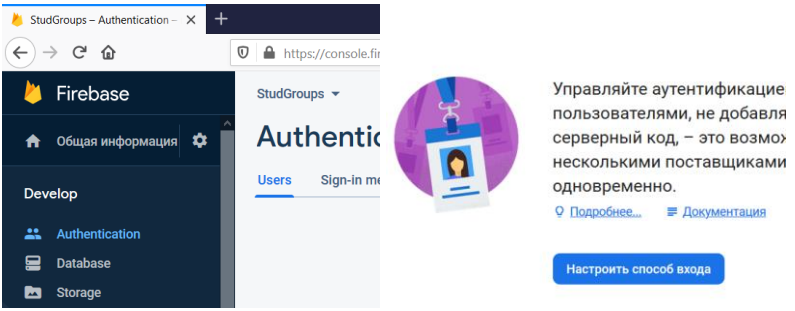

Рисунок 4.79 – Налаштування засобу входу у проєкті Firebase

Обираємо провайдера авторизації «Адреса електронної пошти і пароль» (рис. 4.80).

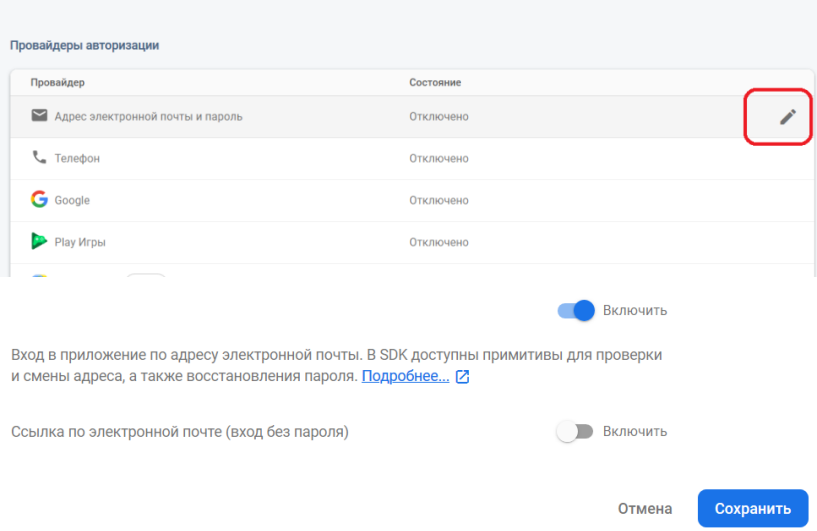

Рисунок 4.80 – Вибір провайдера авторизації

Перейдемо на вкладку Users та додаємо нового користувача (рис. 4.81).

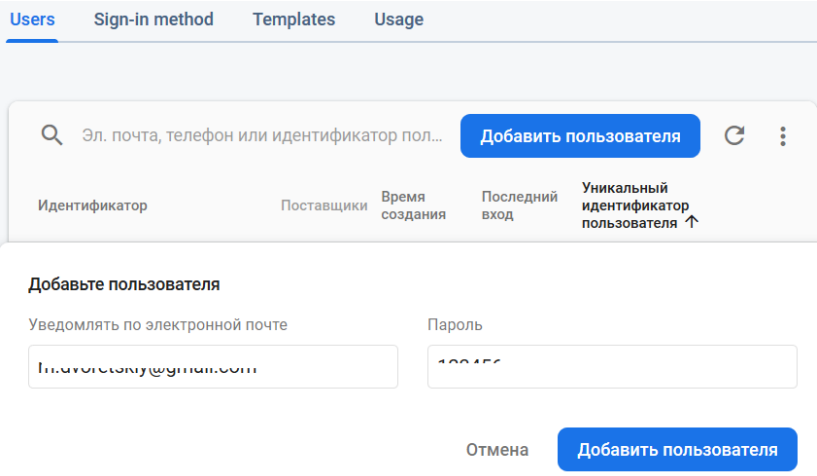

Рисунок 4.81 – Додавання нового користувача

Переходимо до сервісу fire-data-getter у нашому клієнтському застосунку та реалізуємо функціонал аутентифікації користувача (рис. 4.82).

```
5   import {AngularFireAuth} from '@angular/fire/auth';
6
11  export class FireDataGetterService {
12
13    groups: Observable<any[]>;
14    private userName = '';
15
16    constructor(private readonly afs: AngularFirestore,
17                private afAuth: AngularFireAuth) {
27
28    checkUser(user) {
29        return this.afAuth.signInWithEmailAndPassword(
30            user.username,
31            user.passwd
32        );
33    }
34
35    getUser() {
36      return this.userName;
37    }
38
39    setUser(name: string) {
40      this.userName = name;
41    }
```

Рисунок 4.82 – Зміни у fire-data-getter.service.ts

Також у app.module.ts додаємо до масиву провайдерів AngularFireAuth (рис. 4.83).

```
16  import {AngularFireAuth} from '@angular/fire/auth';
17
29    providers: [
30      StatusBar,
31      SplashScreen,
32      { provide: RouteReuseStrategy, useClass: IonicRouteStrategy },
33      AngularFirestore, AngularFireAuth
34    ],
```

Рисунок 4.83 – Зміни у app.module.ts

Використаємо створений метод checkUser на сторінці login.page.ts. Також приберемо код, що відразу перенаправляв нас на сторінку home.page.ts (рис. 4.84).

```
import {FireDataGetterService} from '../services/fire-data-getter.service';


  constructor(
      private router: Router,
      private dataGetter: DataGetterService,
      public alertController: AlertController,
      private fireData: FireDataGetterService) {}

  ngOnInit() {}

  login() {
    this.fireData.checkUser( user: {
      username: this.userName,
      passwd: this.passWord
    }).then(
        onfulfilled: res => {
          this.fireData.setUser(this.userName);
          this.router.navigate( commands: ['/home']);
        },
        onrejected: err => {
          this.userNotExistAlert(err.message);
        }
    );
  }
```

Рисунок 4.84 – Зміни у login.page.ts

Змінимо код src\app\guards\auth.guard.ts, щоб отримувати дані про успішне проходження аутентифікації із сервісу fire-data-getter (рис. 4.85).

```
import {FireDataGetterService} from '../services/fire-data-getter.service';

  constructor(private dataGetter: DataGetterService,
              private router: Router,
              private fireData: FireDataGetterService) {}

  canActivate(
      next: ActivatedRouteSnapshot,
      state: RouterStateSnapshot): boolean {
    const isLoggenIn = this.fireData.getUser() !== '';
    if (!isLoggenIn) {
      this.router.navigateByUrl( url: '/login');
    }
    return isLoggenIn;
  }
```

Рисунок 4.85 – Зміни guard-у

І відкоригуємо home.page.ts для виведення імені користувача у верхній частині сторінки (рис. 4.86).

```
23    constructor(private dataGetter: DataGetterService,
24                private router: Router,
25                private dataExchanger: DataExchangerService,
26                private fireData: FireDataGetterService) {
27      this.fireData.getGroups().subscribe(
28          next: data => this.groups = data
29      );
30      this.userName = this.fireData.getUser();
31    }
```

Рисунок 4.86 – Отримання імені користувача в home.page.ts

Виконаємо вхід до нашого застосунку та перевіримо його роботу (рис. 4.87).

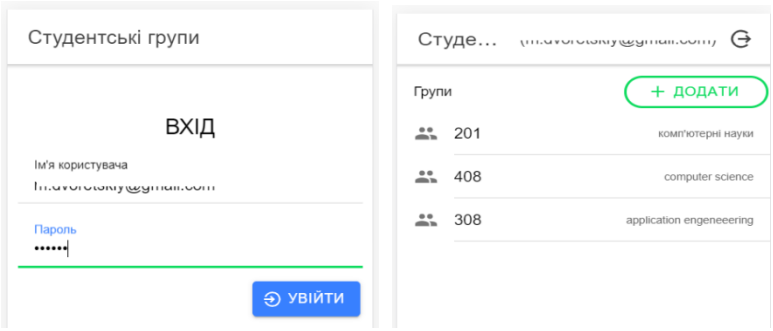

Рисунок 4.87 – Перевірка роботи аутентифікації після виконаних змін

Також виконаємо декілька невдалих спроб авторизації та переглянемо статуси, що повертає Firebase (рис. 4.88).

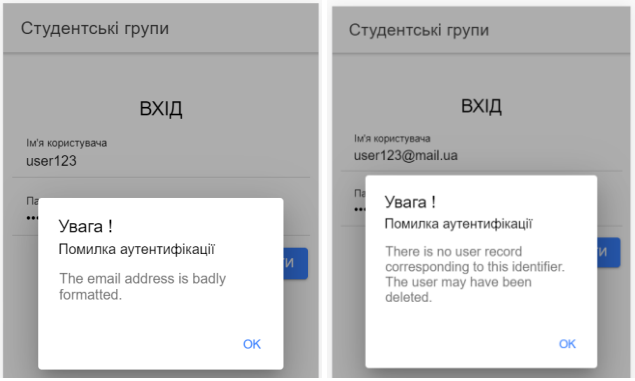

Рисунок 4.88 – Спроби невдалої авторизації

Отже, ми реалізували клієнтську аутентифікацію, але на сьогодні залишається можливість доступу до даних напряму неаутентифікованих користувачів. Переконаємось у цьому – додамо у подію onInit сторінки login операцію отримання списку студентських груп (рис. 4.89). Звернімо увагу, що ця подія виникає до авторизації.

```
21
22    ngOnInit() {
23        this.fireData.getGroups().subscribe(
24            next: data => console.log(data)
25        );
26    }
```

Рисунок 4.89 – Додавання запиту на отримання даних

Запустимо застосунок та переконаємось, що дані отримано (рис. 4.90).

```
▼(3) [{…}, {…}, {…}]
  ▼0:
      id: "DUiHg2pdk3qNc16Z3MYc"
      number: 201
      specialty: "комп'ютерні науки"
      studentsQuantity: 29
      faculty: "Комп.наук"
    ▶__proto__: Object
  ▶1: {id: "RSn4p8gtETdA21DVcIJn", specialty: "computer science", studentsQuantity: 25, fa
  ▶2: {id: "iUbACoUN7BrFnBC8G464", specialty: "application engeneeering", studentsQuantity
    length: 3
  ▶__proto__: Array(0)
```

Рисунок 4.90 – Отримання даних без авторизації

Для усунення цього недоліку необхідно додати перевірку прав доступу до даних на сервері. Для цього переходимо у консоль Firebase на вкладку database rules (правила) та замінимо існуюче правило вказаним кодом (рис. 4.91), що надаватиме право на читання даних тільки аутентифікованим користувачам.

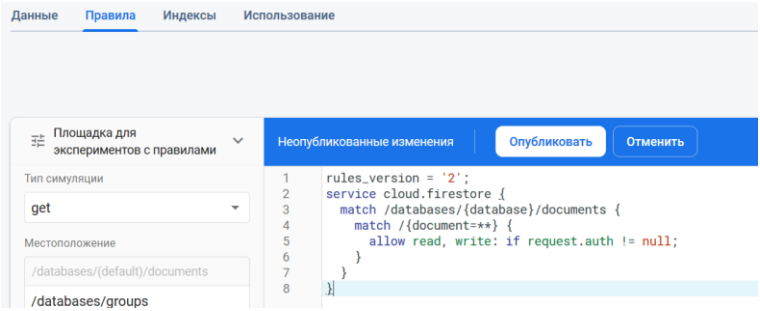

Рисунок 4.91 – Створення правила для контролю доступу до даних

Перед застосуванням змін запустимо симуляцію для варіанта без аутентифікації (рис. 4.92) та з аутентифікацією (рис. 4.93).

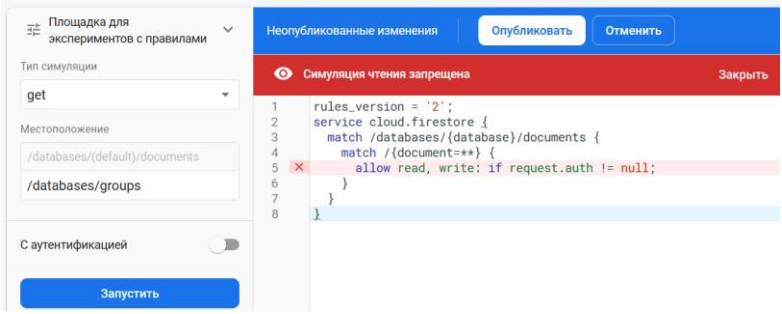

Рисунок 4.92 – Симуляція доступу до даних без аутентифікації

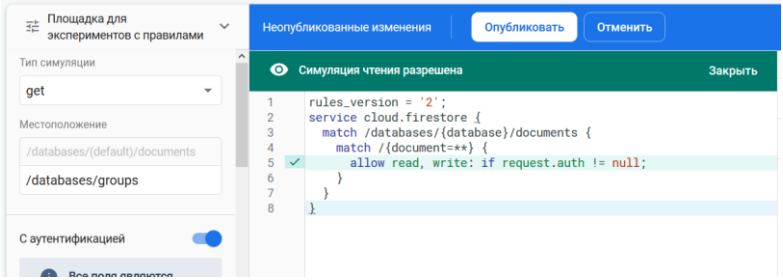

Рисунок 4.93 – Симуляція доступу до даних з аутентифікацією

Переходимо до нашого застосунку та переконуємося у тому, що до проходження аутентифікації спроба доступу до даних веде до помилки (рис. 4.94), а операція читання даних після авторизації цілком успішна (рис. 4.95).

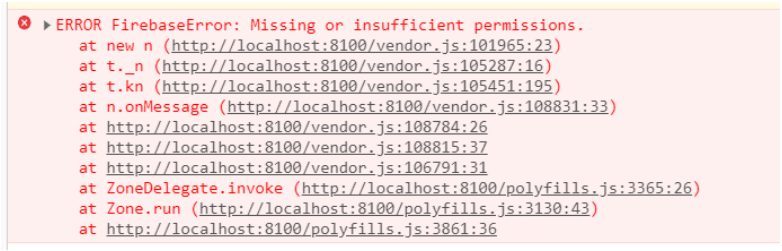

Рисунок 4.94 – Спроба звернення до даних без аутентифікації

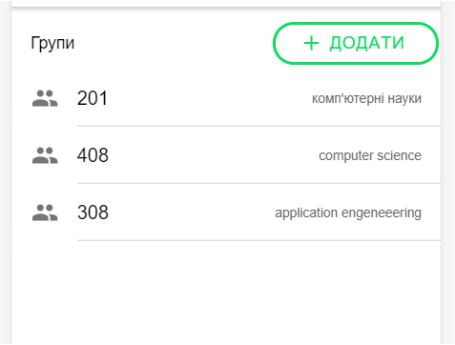

Рисунок 4.95 – Читання даних після виконання входу

Backend as a Service (BaaS) – це модель, яка дозволяє розробникам веб- та мобільних застосунків не взаємодіяти з сервером застосунку, базою даних, клієнт-серверною бібліотекою, хостингом, не писати адміністративну панель, не продумувати дизайн власного API, а скористатися готовими серверними сервісами та зекономити час на розробку та впровадження зазначених застосунків для найшвидшої цифрової трансформації бізнесу.

ЗАВДАННЯ ДЛЯ ІНДИВІДУАЛЬНОГО ВИКОНАННЯ

Завдання № 1.

Завдання № 1.1. Створити застосунок відповідно до власного варіанта індивідуального завдання (таблиця 1). Реалізувати мінімум 2 варіанти адаптивних інтерфейсів на базі двох різних бібліотек.

Завдання № 1.2. На базі ionic та JS без використання додаткових фреймворків реалізувати застосунок згідно з варіантом індивідуального завдання (таблиця 1).

Завдання № 2.

На базі Angular Framework реалізувати застосунок із адаптивним інтерфейсом, що забезпечує перегляд та редагування (включаючи додавання та видалення) даних списку елементів відповідно до свого варіанта індивідуального завдання. Реалізувати перегляд елементів підпорядкованого списку елементів (згідно з варіантом табл. 1).

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

Під час виконання завдання створити мінімум три власні компоненти, реалізувати передачу даних у дочірній компонент та повернення події у батьківський, роботу з сервісами та маршрутизацію.

Завдання № 3.

Завдання № 3.1. Створити новий ionic tabs проєкт, що надає інформацію про його автора на трьох вкладках: «Загальна інформація» (прізвище, номер групи, спеціальність), «Контактні дані» (електронна пошта, телефон) та «Інтереси» (список інтересів та захоплень).

Завдання № 3.2. На базі ionic framework реалізувати застосунок, що виконує виведення та подальше редагування списку сутностей згідно з варіантом індивідуального завдання.

Завдання № 3.3. Реалізувати у застосунку згідно з власним варіантом індивідуального завдання логін-форму та аутентифікацію користувачів.

Завдання № 3.4. Реалізувати у застосунку перехід до нової сторінки передачею параметра до маршруту та фільтрацією даних, що відображаються відповідно до переданого параметра. Реалізувати редагування даних на дочірній сторінці. Передбачити передачу та повернення даних із дочірньої сторінки (будь-яким способом на вибір).

Завдання № 4.

Завдання № 4.1. 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 двох зв'язаних таблиць). Реалізувати аутентифікацію користувачів (список з паролями має зберігатись на сервері) та контроль у разі спроби отримання неавторизованими користувачами захищених даних.

Під час виведення 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 адмінпанель.

Завдання № 4.2. Виконати реалізацію функціоналу, що наведено у завданні 4.1, на основі Firebase. Організувати збереження даних у firestore, використати механізми автентифікації Firebase та налагодити взаємодію застосунку із проєктом Firebase.

Таблиця 1 – Варіанти індивідуальних завдань

Варіант	Об'єкт (атрибути)	Елемент списку (атрибути)
1	Рецепт *(ПІБ пацієнта, рік народж. пацієнта, ПІБ лікаря)*	Лікарський засіб *(назва, дозування, форма випуску)*
2	Лікар *(ПІБ, спеціалізація, досвід роботи – років)*	Пацієнт *(ПІБ, рік народж., адреса)*
3	Пацієнт *(ПІБ, рік народж., адреса)*	Хвороби (Назва, ступінь)
4	Власник нерухомості *(ПІБ, вік, юр. / фіз. особа)*	Нерухомість *(Місто, вулиця, номер будинку, кількість кімнат)*
5	Виробник *(Назва, країна, кількість співробітників)*	Продукт *(Назва, тип, термін придатності – місяців)*
6	Страва *(Назва, тип, вага порція)*	Склад *(Продукт, вага на порцію)*
7	Автостоянка *(Адреса, ПІБ директора)*	Автомобілі *(Держ. номер, марка, номер місця)*

8	Акваріум *(Форма, об'єм, є/відсутня підсвічення)*	Рибки *(Назва виду, кількість)*
9	Людина *(ПІБ, адреса, вік)*	Родичі *(ПІБ, тип: батько / мати / брат /…, вік, чи проживає разом – так / ні)*
10	Фільм *(Назва, рік виходу, країна)*	Актори *(ПІБ, роль: головна / другорядна / епізод)*
11	Актор *(ПІБ, рік народження, країна)*	Фільми *(Назва, країна, рік виходу)*
12	Зоопарк *(Місто, ПІБ директора, рік заснування)*	Тварини *(Назва, кількість, тип)*
13	Автор *(ПІБ, країна, рік народження)*	Книги *(Назва, жанр, видавництво)*
14	Спортивна команда *(Назва, вид спорту, ПІБ тренера)*	Спортсмени *(ПІБ, рік народження, досвід – років)*
15	Місто *(Назва, область, країна)*	Адміністративні райони *(Назва, площа, кількість жителів)*

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Chris Griffith. Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova 1st Edition. Kindle Edition. 2017. 450 p.
2. Greg Lim. Beginning Ionic for iOS and Android Development. Kindle Edition. 2017. 154 p.
3. Hoc Phan. Ionic Cookbook: 1st Edition. Kindle Edition. 2015. 266 p.
4. Jakobus B., Jason M. Mastering Bootstrap 4, Second Edition. Packt Publishing, 2018. 354 p.
5. Mikowski M., Powell J. Single Page Web Applications: JavaScript end-to-end 1st Edition. Manning Publications, 2013. 432 p.
6. Murray N., Lerner A., Coury F., Taborda C. ng-book: The Complete Guide to Angular 4, 4th edition. CreateSpace Independent Publishing Platform, 2017. 622 p.
7. Prabhu A., Shenoy A. Introducing Materialize, 1st ed. Edition. Apress, 2016. 149 p.
8. Spurlock J. Bootstrap: Responsive Web Development, 1st edition. O'Reilly Media, 2013. 128 p.
9. The Complete Guide to Angular. Published in San Francisco, California by Fullstack.io, 2017. 683 p.
10. Thomas J., Potiekhin O., Lauhakari M., Shah A., Berning D. Creating interfaces with Bulma. Bleeding Edge Press, 2018. 222 p.
11. Морето С. Bootstrap в примерах. ДМК Пресс, 2016. 314 с.
12. Файн Я., Моисеев А. Angular и TypeScript. Сайтостроение для профессионалов. СПб. : Питер, 2018. 464 с.

ДЛЯ НОТАТОК

ДЛЯ НОТАТОК

ДЛЯ НОТАТОК

Навчальне видання

Михайло Леонідович
ДВОРЕЦЬКИЙ,
Ірина Миколаївна
ЖУРАВСЬКА,
Світлана Володимирівна
ДВОРЕЦЬКА,
Світлана Юріївна
БОРОВЛЬОВА

РОЗРОБКА ГІБРИДНИХ ЗАСТОСУНКІВ ДЛЯ ЦИФРОВОЇ ТРАНСФОРМАЦІЇ БІЗНЕСУ

Навчальний посібник

Для підготовки здобувачів вищої освіти
в галузі знань 12 «Інформаційні технології»

Друкується у авторській редакції.
Технічний редактор *О. Петроченко*.
Комп'ютерна верстка, дизайн обкладинки *Н. Кардаш.*
Друк *С. Волинець*. Фальцювально-палітурні роботи, *О. Мішалкіна.*

Підп. до друку 10.01.2022.
Формат 60x84$^{1}/_{16}$. Папір офсет.
Гарнітура «Times New Roman». Друк ризограф.
Ум. друк. арк. 9,3. Обл.-вид. арк. 6,67.
Тираж 300 пр. Зам. № 6461.

54003, м. Миколаїв, вул. 68 Десантників, 10.
Тел.: 8 (0512) 50–03–32, 8 (0512) 76–55–81, e-mail: rector@chmnu.edu.ua.
Свідоцтво суб'єкта видавничої справи ДК № 6124 від 05.04.2018.