

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»*



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

© Дворецький М. Л., Журавська І. М.,
Дворецька С. В., Боровльова С. Ю., 2022
© ЧНУ ім. Петра Могили, 2022

ISBN 978-966-336-435-3

ЗМІСТ

ВСТУП.....	4
РОЗДІЛ 1. РЕАЛІЗАЦІЯ АДАПТИВНИХ	
ВЕБІНТЕРФЕЙСІВ.....	7
1.1. Використання twitter bootstrap	7
1.2. Реалізація адаптивного вебінтерфейсу	
засобами bulma та materialize	16
1.3. Знайомство з Ionic Framework	24
РОЗДІЛ 2. РОЗРОБКА ОДНОСТОПІНКОВИХ	
ВЕБЗАСТОСУНКІВ НА БАЗІ ANGULAR FRAMEWORK.....	31
2.1. Поняття SPA та знайомство з Angular	31
2.2. Робота із компонентами.....	37
2.3. Події та директиви	45
2.4. Реалізація роботи зі списком студентських груп	54
2.5. Використання сервісів та роутингу	62
РОЗДІЛ 3. РОЗРОБКА ГІБРИДНИХ ЗАСТОСУНКІВ	
НА ОСНОВІ IONIC FRAMEWORK	73
3.1. Налаштування середовища на розгортання проекту	73
3.2. Створення застосунку «список студентських груп»	81
3.3. Додавання сторінок та аутентифікація користувачів.....	89
3.4. Додавання сторінок та передача даних	95
3.5. Html-запити	105
РОЗДІЛ 4. ОРГАНІЗАЦІЯ РОБОТИ ІЗ BACK-END	
СКЛАДОВОЮ	113
4.1. Створення back-end складової проекту	113
4.2. Реалізація функцій модифікації даних	122
4.3. Використання Firebase	128
4.4. Модифікація даних Firebase	138
4.5. Автентифікації користувачів Firebase	145
ЗАВДАННЯ ДЛЯ ІНДИВІДУАЛЬНОГО ВИКОНАННЯ	153
РЕКОМЕНДОВАНА ЛІТЕРАТУРА	156

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 переглядається з різних пристроїв і з екранами різної роздільовальності, за технологією адаптивного вебдизайну не потрібно створювати окремі версії вебсайту для окремих видів пристроїв.

Нативні (рідні) застосунки написані мовою програмування, специфічною для платформи, за для 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 багаторічкові (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, організації збереження даних на віддаленому сервері та взаємодію із ним.

РОЗДІЛ 1.

РЕАЛІЗАЦІЯ АДАПТИВНИХ ВЕБІНТЕРФЕЙСІВ

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.

```
3 <head>
4   <meta charset="UTF-8">
5   <title>Document</title>
6   <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/
7     bootstrap/4.4.1/css/bootstrap.min.css">
8   <script
9     src="https://code.jquery.com/jquery-3.4.1.min.js"
10    integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSF1B
11    w8HfCJo="
12    crossorigin="anonymous"></script>
13   <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/
14     1.15.0/esm/popper.min.js"></script>
15   <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/
16     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).

```
1 <!DOCTYPE html>
2 <html lang="en">
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>
```

```
10 | <script src="code/my.js"></script>
11 | </head>
12 | <body>
13 |   <div class="text-center mt-5" >
14 |     <button type="button" class="btn btn-primary center"
15 |       id="click-me">Натисни мене !</button>
16 |   </div>
17 |   <div class="m-5" id="alert">
18 |   </div>
19 | </body>
20 | </html>
```

Рисунок 1.4 – Додавання кнопки та підключення my.js

```
1 $( "document").ready(function() {
2   $('#click-me').click(function() {
3     var html = `<div class="text-left alert alert-success" role="alert">
4       <h4 class="alert-heading">Привіт !
5       <button type="button" class="close" data-dismiss="alert"
6         aria-label="Close">
7         <span aria-hidden="true">&times;</span>
8       </button>
9     </h4>
10    <p>Вітаємо із успішним підключенням twitter bootstrap !</p>
11    <hr>
12    <p>Тепер можна переходити до реалізації проекту.</p>
13  </div>`;
14  $('#alert').html(html);
15 });
```

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).

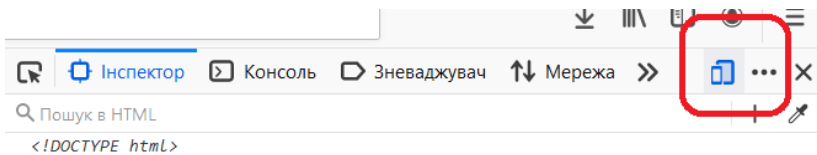


Рисунок 1.6 – Перехід до режиму адаптивного дизайну

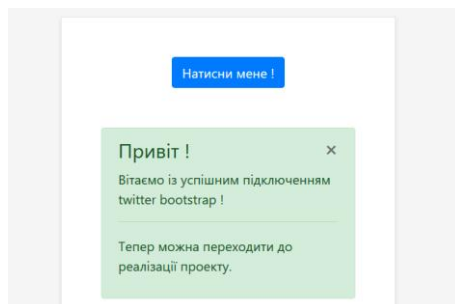


Рисунок 1.7 – Перегляд зовнішнього вигляду сторінки

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).

```
1  var groups = localStorage.getItem("groups_data");
2  if (localStorage.getItem("groups_data") === null) {
3      groups = [
4          {
5              number: 301,
6              faculty: 'Computer science'
7          },
8          {
9              number: 302,
10             faculty: 'Computer science'
11         }
12     ];
13     localStorage.setItem("groups_data", JSON.stringify(groups));
14 } else {
15     groups = JSON.parse(groups);
16 }
```

Рисунок 1.8 – Файл data.js

Тепер підключимо цей скрипт до index.html та розмістимо у елементі body нав-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.

Розробка гібридних застосунків для цифрової трансформації бізнесу

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Document</title>
6   <link rel="stylesheet" href="assets/bootstrap.min.css">
7   <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/font-awesome/
8     4.7.0/css/font-awesome.min.css">
9   <script src="assets/jquery-3.4.1.min.js"></script>
10  <script src="assets/popper.min.js"></script>
11  <script src="assets/bootstrap.min.js"></script>
12  <script src="code/data.js"></script>
13  <script src="code/my.js"></script>
14 </head>
15 <body>
16   <nav class="navbar navbar-dark bg-dark text-white">
17     <h3>Студентські групи</h3>
18   </nav>
19   <div class="container mt-3">
20     <div class="row">
21       <div class="col">
22         <a class="btn btn-outline-success" href="#">
23           <i class="fa fa-fw fa-plus"></i>
24         </a>
25       </div>
26     </div>
27     <div class="row mt-2">
28       <div class="col">
29         <table class="table">
30           <thead>
31             <tr>
32               <th scope="col" style="width: 60%">Група</th>
33               <th scope="col" style="width: 40%"></th>
34             </tr>
35           </thead>
36           <tbody id="groups">
37             <tr>
38               <td></td>
39             </tr>
40           </tbody>
41         </table>
42       </div>
43     </div>
44   </body>
45 </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).

```
1  ✓ $('document').ready(function() {  
2      var rowText;  
3      var content = $('#groups');  
4  ✓  for(var row of groups) {  
5          rowText =  
6          `<tr>  
7              <td>${row.number}</td>  
8              <td class="text-right">  
9                  <a class="btn btn-outline-secondary" href="group.html?  
10                 number=${row.number}>  
11                     <i class="fa fa-fw fa-edit"></i>  
12                 </a>  
13                 <button type="button" class="btn btn-outline-danger rem-row"  
14                 rowid="${row.number}>  
15                     <i class="fa fa-fw fa-trash"></i>  
16                 </button>  
17             </td>  
18         </tr>`;  
19         content.append(rowText);  
20     }  
21 });
```

Рисунок 1.10 – Скрипт my.js

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

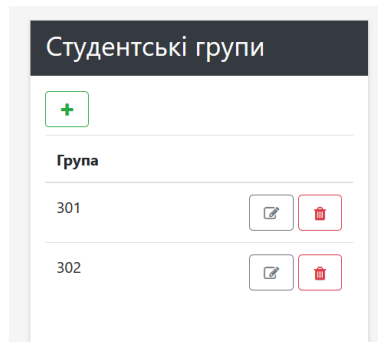


Рисунок 1.11 – Сторінка виведення студентських груп

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).


```
1 function saveGroups(groups) {  
2   |   localStorage.setItem("groups_data", JSON.stringify(groups));  
3 }
```

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.

```
1 <!DOCTYPE html>  
2 <html lang="en">  
3 <head>  
4   <meta charset="UTF-8">  
5   <title>Студентські групи</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>  
10  <script src="code/data.js"></script>  
11  <script src="code/group.js"></script>  
12 </head>  
13 <body>  
14   <nav class="navbar navbar-dark bg-dark text-white">  
15     <h3>Студентська група</h3>  
16   </nav>  
17   <div class="container mt-3">  
18     <div class="row">  
19       <div class="col">  
20         <form>  
21           <div class="form-group">  
22             <label for="number">Номер групи</label>  
23             <input type="text" class="form-control" id="number"  
24               placeholder="Номер групи">  
25           </div>  
26           <div class="form-group">  
27             <label for="faculty">Факультет</label>  
28             <input type="text" class="form-control" id="faculty"  
29               placeholder="Факультет">  
30           </div>  
31           <button type="button" id="save" class="btn  
32             btn-secondary">Зберегти</button>  
33         </form>  
34       </div>  
35     </div>  
36   </div>  
37 </body>  
38 </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».

```
19 | <div class="row">
20 |   <div class="col">
21 |     <a class="btn btn-outline-success" href="group.html">
22 |       <i class="fa fa-fw fa-plus"></i>
23 |     </a>
24 |   </div>
```

Рисунок 1.14 – Зміни у index.html

```
1 | $('document').ready(function() {
2 |   var rowText;
3 |   var content = $('#groups');
4 |   for(var row of groups) {
5 |     rowText =
6 |     `<tr>
7 |       <td>${row.number}</td>
8 |       <td class="text-right">
9 |         <a class="btn btn-outline-secondary" href="group.html?number=${
10 |           {row.number}}>
11 |           <i class="fa fa-fw fa-edit"></i>
12 |         </a>
13 |         <button type="button" class="btn btn-outline-danger rem-row"
14 |           rowid="${row.number}">
15 |           <i class="fa fa-fw fa-trash"></i>
16 |         </button>
17 |       </td>
18 |     </tr>`;
19 |     content.append(rowText);
20 |   }
21 |   $('.rem-row').click(function() {
22 |     let number = $(this).attr('rowid');
23 |     saveGroups(groups.filter((g) => g.number !== number));
24 |     location.reload();
25 |   });
26 | });
```

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.

```
1  ✓ $('document').ready(function() {  
2      let searchParams = new URLSearchParams(window.location.search);  
3      let number = 0;  
4      if (searchParams.has('number')) {  
5          number = searchParams.get('number');  
6          let group = groups.find((g)=>g.number == number);  
7          $('#number').val(group.number);  
8          $('#faculty').val(group.faculty);  
9      }  
10  
11      $('#save').click(function() {  
12          if (number === 0) {  
13              groups.push({  
14                  number: $('#number').val(),  
15                  faculty: $('#faculty').val()  
16              });  
17          } else {  
18              let group = groups.find((g)=>g.number == number);  
19              group.number = $('#number').val();  
20              group.faculty = $('#faculty').val();  
21          }  
22          saveGroups(groups);  
23          $(location).attr('href', 'index.html');  
24      });  
25  });
```

Рисунок 1.16 – Файл data/group.js

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

Студентська група

Номер групи

Факультет

Зберегти

Студентські групи

+

Група







301		
302		
201		

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

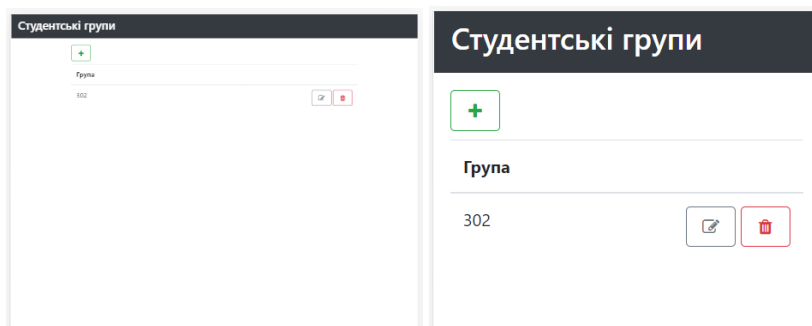


Рисунок 1.19 – Зовнішній вигляд сторінки на мобільному пристрої

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 вебпроектів. 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).

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <title>Hello Bulma!</title>
7     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.8.0/
8       css/bulma.min.css">
9     <script defer src="https://use.fontawesome.com/releases/v5.3.1/js/
10       all.js"></script>
11     <script src="assets/jquery-3.4.1.min.js"></script>
12     <script src="code/data.js"></script>
13     <script src="code/my.js"></script>
14   </head>
15   <body>
16     <section class="section">
17       <div class="container">
18         <nav class="navbar is-fixed-top has-background-dark">
19           <h3 class="title has-text-white has-text-centered">
20             Студентські групи
21           </h3>
22         </nav>
23         <div style="margin: 20px;"></div>
24         <a class="button is-primary" href="group.html"><i class="fa fa-fw
25           fa-plus"></i></a>
26         <div style="margin: 10px;"></div>
27         <table class="table is-fullwidth has-margin-top-2">
28           <thead>
29             <tr>
30               <th>Група</th>
31             <th></th>
32           </tr>
33         </thead>
34         <tbody id="groups">
35         </tbody>
36       </table>
37     </div>
38   </section>
39 </body>
40 </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).

```
1  $('document').ready(function() {
2      var rowText;
3      var content = $('#groups');
4  ✓  for(var row of groups) {
5          rowText =
6          `
7              <tr>
8                  <td>${row.number}</td>
9                  <td class="has-text-right">
10                     <a href="group.html?number=${row.number}" class="button
11                        is-link">
12                         <i class="fa fa-fw fa-edit"></i>
13                     </a>
14                     <button type="button" class="button is-danger rem-row"
15                        rowid="${row.number}">
16                         <i class="fa fa-fw fa-trash"></i>
17                     </button>
18                 </td>
19             </tr>`;
20             content.append(rowText);
21         }
22     }
23     $('#rem-row').click(function() {
24         let number = $(this).attr('rowid');
25         saveGroups(groups.filter((g)=>g.number != number));
26         location.reload();
27     });
28 });
```

Рисунок 1.21 – Корегування файлу `my.js`

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

```
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">
6          <title>Hello Bulma!</title>
7          <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/
8              bulma@0.8.0/css/bulma.min.css">
9          <script defer src="https://use.fontawesome.com/releases/v5.3.1/js/
10              all.js"></script>
```

**Розробка гібридних застосунків
для цифрової трансформації бізнесу**

```
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"
31                                     type="email" placeholder="Номер групи">
32                             </p>
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"
43                                     type="email" placeholder="Факультет">
44                             </p>
45                         </div>
46                     </div>
47                 <button type="button" class="button is-primary"
48                     id="save">Зберегти</button>
49             </form>
50         </div>
51     </section>
52 </body>
</html>
```

Рисунок. 1.22 – Файл group.html

Збережемо зміни та перевіримо результат (рис. 1.23).

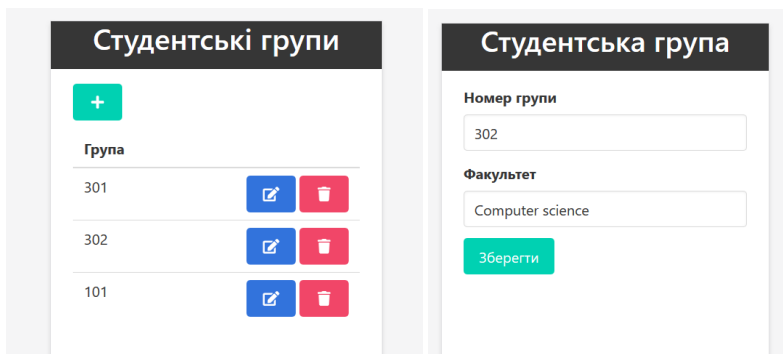


Рисунок 1.23 – Зовнішній вигляд сторінок, реалізованих на базі bulma

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 materialize.css (рис. 1.24).

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


Розробка гібридних застосунків для цифрової трансформації бізнесу

```
17 |         <a class="brand-logo">Групи</a>
18 |     </div>
19 | </nav>
20 | <div class="container">
21 |     <div class="row" style="margin-top: 20px;">
22 |         <div class="col m6 offset-m3 s12">
23 |             <a href="group.html" class="btn-floating btn-large waves-effect waves-light
24 |                 green">
25 |                 <i class="material-icons">add</i>
26 |             </a>
27 |         </div>
28 |     </div>
29 |     <div class="row">
30 |         <div class="col m6 offset-m3 s12 no-padding">
31 |             <table class="striped">
32 |                 <thead>
33 |                     <tr>
34 |                         <th style="width: 60%">Група</th>
35 |                         <th style="width: 40%"></th>
36 |                     </tr>
37 |                 </thead>
38 |                 <tbody id="groups">
39 |                 </tbody>
40 |             </table>
41 |         </div>
42 |     </div>
43 | </body>
44 | </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).

```
1 | $('document').ready(function() {
2 |     var rowText;
3 |     var content = $('#groups');
4 |     for(var row of groups) {
5 |         rowText =
6 |         `<tr>
7 |             <td><h5>${row.number}</h5></td>
8 |             <td class="right-align">
9 |                 <a href="group.html?number=${row.number}" class="btn-floating waves-effect
10 |                     waves-light blue" style="margin-right:10px;">
11 |                     <i class="material-icons">edit</i>
12 |                 </a>
13 |                 <a rowid="${row.number}" class="rem-row btn-floating waves-effect waves-light red">
14 |                     <i class="material-icons">delete</i>
15 |                 </a>
16 |             </td>
17 |         </tr>`;
18 |         content.append(rowText);
19 |     }
20 |     $('.rem-row').click(function() {
21 |         let number = $(this).attr('rowid');
22 |         saveGroups(groups.filter((g)=>g.number != number));
23 |         location.reload();
24 |     });
25 | });
```

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).

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="initial-scale=1">
6   <title>Студентські групи</title>
7   <link rel="stylesheet" href="assets/materialize.min.css">
8   <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
9   <script src="assets/jquery-3.4.1.min.js"></script>
10  <script src="assets/materialize.min.js"></script>
11  <script src="code/data.js"></script>
12  <script src="code/group.js"></script>
13 </head>
14 <body>
15   <nav>
16     <div class="nav-wrapper">
17       <a class="brand-logo">Група</a>
18     </div>
19   </nav>
20   <div class="container">
21     <div class="row">
22       <form>
23         <div class="input-field col s12">
24           <input id="number" type="text" class="validate">
25           <label for="number">Номер групи</label>
26         </div>
27         <div class="input-field col s12">
28           <input id="faculty" type="text" class="validate">
29           <label for="faculty">Факультет</label>
30         </div>
31         <div class="input-field col s12 right-align">
32           <a id="save" class="btn-floating btn-large waves-effect waves-light green">
33             <i class="material-icons">done</i>
34           </a>
35         </div>
36       </form>
37     </div>
38   </div>
39 </body>
40 </html>
```

Рисунок 1.26 – Файл group.html

```
1 $(document).ready(function() {
2   let searchParams = new URLSearchParams(window.location.search);
3   let number = 0;
4   if (searchParams.has('number')) {
5     number = searchParams.get('number');
6     let group = groups.find((g) => g.number == number);
7     $('#number').val(group.number);
8     $('#faculty').val(group.faculty);
```

```
9 |         $('#label').addClass('active');
10 |     }
11 |
12 |     $('#save').click(function() {
13 |         if (number === 0) {
14 |             groups.push({
15 |                 number: $('#number').val(),
16 |                 faculty: $('#faculty').val()
17 |             });
18 |         } else {
19 |             let group = groups.find((g) => g.number == number);
20 |             group.number = $('#number').val();
21 |             group.faculty = $('#faculty').val();
22 |         }
23 |         saveGroups(groups);
24 |         $(location).attr('href', 'index.html');
25 |     });
26 | });
```

Рисунок 1.27 – Файл group.js

Переглянемо зовнішній вигляд застосунку (рис. 1.28).

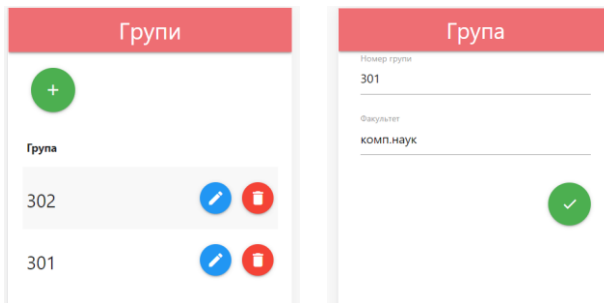


Рисунок 1.28 – Вигляд застосунку із використанням materialize.css

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

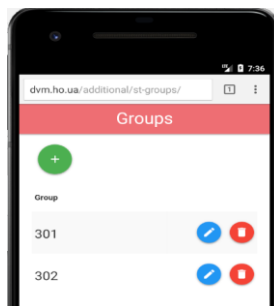


Рисунок 1.29 – Зовнішній вигляд застосунку на емуляторі

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 іоніс через cdn

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

```
12 <body>
13   <ion-header>
14     <ion-toolbar>
15       <ion-title>Groups</ion-title>
16     </ion-toolbar>
17   </ion-header>
18   <section id="main">
19     <ion-grid>
20       <ion-row>
21         <ion-col size="12" size-md="6" offset-md="3">
22           <ion-button color="success" size="large">
23             <ion-icon name="add-circle-outline"></ion-icon>
24           </ion-button>
25         </ion-col>
26       </ion-row>
27       <ion-row>
28         <ion-col size="12" size-md="6" offset-md="3">
29           <ion-list>
30             <ion-list-header>
31               <ion-label>Group</ion-label>
32             </ion-list-header>
33             <ion-item>
34               <ion-label>301</ion-label>
35               <ion-button color="primary" size="medium">
36                 <ion-icon name="create"></ion-icon>
37               </ion-button>
38               <ion-button color="danger" size="medium">
39                 <ion-icon name="trash"></ion-icon>
40               </ion-button>
41             </ion-item>
42           </ion-list>
43         </ion-col>
44       </ion-row>
45     </ion-grid>
46   </section>
47 </body>
48 </html>
```

Рисунок 1.31 – Шаблон головної сторінки

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

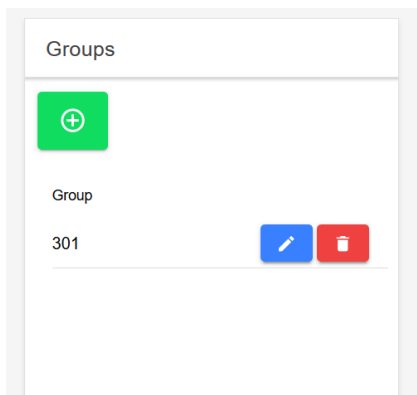


Рисунок 1.32 – Зовнішній вигляд головної сторінки

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

```
40 <section id="edit">
41   <ion-grid>
42     <ion-row>
43       <ion-col size="12" size-md="6" offset-md="3">
44         <ion-item>
45           <ion-label position="floating">Group number</ion-label>
46           <ion-input id="number"></ion-input>
47         </ion-item>
48         <ion-item>
49           <ion-label position="floating">Faculty</ion-label>
50           <ion-input id="faculty"></ion-input>
51         </ion-item>
52         <ion-button id="save" color="primary" size="medium" float-right
53           style="margin-top: 20px;">
54           <ion-icon name="checkmark"></ion-icon>
55         </ion-button>
56       </ion-col>
57     </ion-row>
58   </ion-grid>
</section>
```

Рисунок 1.33 – Шаблон сторінки редагування елемента

Зовнішній вигляд форми редагування наведено на рис. 1.34.

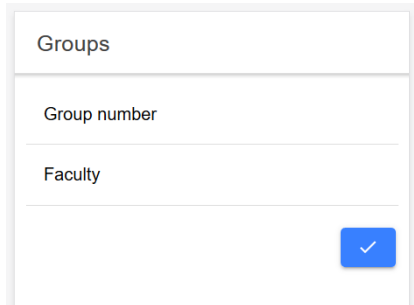


Рисунок 1.34 – Зовнішній вигляд форми редагування

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).

```
1 function saveGroups(groups) {  
2     localStorage.setItem("groups_data", JSON.stringify(groups));  
3 }  
4  
5 var groups = localStorage.getItem("groups_data");  
6 if (localStorage.getItem("groups_data") === null) {  
7     groups = [  
8         {  
9             number: 301,  
10            faculty: 'Computer science'  
11        },  
12        {  
13            number: 302,  
14            faculty: 'Computer science'  
15        }  
16    ];  
17    saveGroups(groups);  
18 } else {  
19     groups = JSON.parse(groups);  
20 }
```

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).

```
1 let editType = '';
2 let number = '';
3 document.addEventListener("DOMContentLoaded", function(event) {
4   var rowText;
5   var list = document.getElementById('list');
6
7   for(var row of groups) {
8     var rowText = document.createElement('ion-item');
9     rowText.innerHTML = `<ion-label>${row.number}</ion-label>
10      <ion-button class="edit" color="primary" size="medium" rowid="${row.number}"
11        >
12          <ion-icon name="create"></ion-icon>
13        </ion-button>
14        <ion-button class="delete" color="danger" size="medium" rowid="${
15          row.number}>
16          <ion-icon name="trash"></ion-icon>
17        </ion-button>`;
18     list.appendChild(rowText);
19
20 document.querySelector('#main').style.display = "";
21 document.querySelector('#edit').style.display = 'none';
22
23 document.querySelectorAll('.edit')
24   .forEach(input => input.addEventListener('click', ({target}) => {
25     editType = "edit";
26     number = target.getAttribute('rowid');
27     let group = groups.find((g)=>g.number == number);
28
29     document.getElementById('number').value = group.number;
30     document.getElementById('faculty').value = group.faculty;
31     document.querySelector('#main').style.display = "none";
32     document.querySelector('#edit').style.display = "";
33   }));
34
35 document.querySelectorAll('.delete')
36   .forEach(input => input.addEventListener('click', ({target}) => {
37     number = target.getAttribute('rowid');
38     saveGroups(groups.filter((g)=>g.number != number));
39     location.reload();
40   }));
41
42 document.querySelector('#save').addEventListener('click', () => {
43   if (editType == "add") {
44     groups.push({
45       number: document.getElementById('number').value,
46       faculty: document.getElementById('faculty').value
47     });
48   } else {
49     let group = groups.find((g)=>g.number == number);
50     group.number = document.getElementById('number').value;
51     group.faculty = document.getElementById('faculty').value;
52   }
53   saveGroups(groups);
54   location.reload();
55 });
```


Розробка гібридних застосунків для цифрової трансформації бізнесу

```
56 document.querySelector('#add').addEventListener('click', () => {
57     document.querySelector('#main').style.display = "none";
58     document.querySelector('#edit').style.display = "";
59     document.getElementById('number').value = "";
60     document.getElementById('faculty').value = "";
61     editType = "add";
62 });
63 });
```

Рисунок 1.36 – Файл my.js

Повернімося 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.

```
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    <script src="data.js"></script>
11    <script src="my.js"></script>
12    <title>Students groups</title>
13 </head>
14 <body>
15     <ion-header>
16         <ion-toolbar>
17             <ion-title>Groups</ion-title>
18         </ion-toolbar>
19     </ion-header>
20     <section id="main">
21         <ion-grid>
22             <ion-row>
23                 <ion-col size="12" size-md="6" offset-md="3">
24                     <ion-button id="add" color="success" size="large">
25                         <ion-icon name="add-circle-outline"></ion-icon>
26                     </ion-button>
27                 </ion-col>
28             </ion-row>
29             <ion-row>
30                 <ion-col size="12" size-md="6" offset-md="3">
31                     <ion-list id="list">
32                         <ion-list-header>
33                             <ion-label>Group</ion-label>
34                         </ion-list-header>
35                     </ion-list>
36                 </ion-col>
37             </ion-row>
38         </ion-grid>
39     </section>
```

```
40 <section id="edit">
41   <ion-grid>
42     <ion-row>
43       <ion-col size="12" size-md="6" offset-md="3">
44         <ion-item>
45           <ion-label position="floating">Group number</ion-label>
46           <ion-input id="number"></ion-input>
47         </ion-item>
48         <ion-item>
49           <ion-label position="floating">Faculty</ion-label>
50           <ion-input id="faculty"></ion-input>
51         </ion-item>
52         <ion-button id="save" color="primary" size="medium" float-right
53           style="margin-top: 20px;">
54           <ion-icon name="checkmark"></ion-icon>
55         </ion-button>
56       </ion-col>
57     </ion-row>
58   </ion-grid>
59 </section>
60 </body>
</html>
```

Рисунок 1.37 – Index.html після виконання змін

На рис. 1.38 наведемо вигляд користувацького інтерфейсу отриманого застосунку.

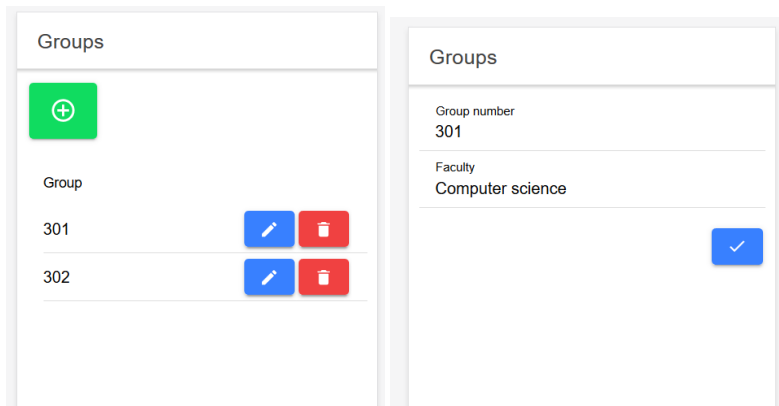


Рисунок 1.38 – Головна сторінка та форма редагування елементу

Як бачимо, тепер, з утратою активності 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 інстальатора.

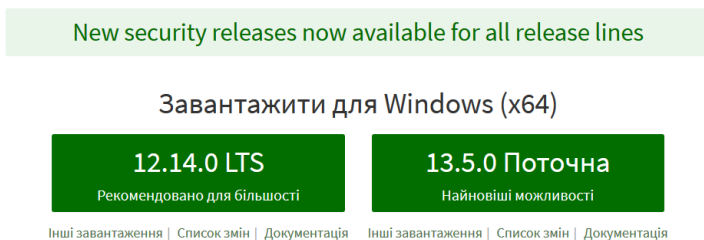


Рисунок 2.1 – Сторінка для завантаження node.js

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).

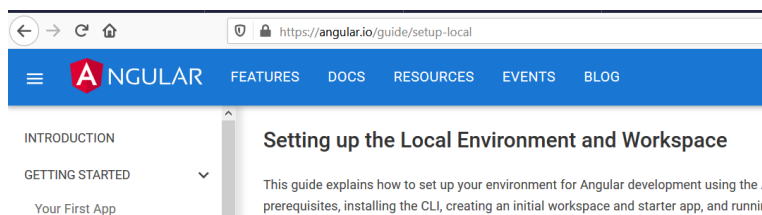


Рисунок 2.3 – Офіційний сайт проєкту Angular

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

? Would you like to share anonymous usage data with the Angular Team at Google?
  Google's Privacy Policy at https://policies.google.com/privacy? For more details
  on 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).

```
C:\>ng --version

Angular CLI
Angular CLI: 8.3.21
Node: 10.15.3
OS: win32 x64
```

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.

```
PS C:\data\personal\michael\MOHYLA\Education\SPA_PWA\code> ng new angular-test-proj
  would you like to add Angular routing? No
  which stylesheet format would you like to use?
  > SCSS [ https://sass-lang.com/documentation/syntax#scss ]
  Sass [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less [ http://lesscss.org ]
  Stylus [ http://stylus-lang.com ]
```

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).

```
C:\data\personal\michael\MOHYLA\Education\SPA_PWA\code\angular-test-proj>ng serve
10% building 3/3 modules 0 active i [wds]: Project is running at http://localhost:4200/
i [wds]: webpack output is served from /
i [wds]: 404s will fallback to //index.html
30% building 29/29 modules 0 active
```

Рисунок 2.7 – Виконання проєкту

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

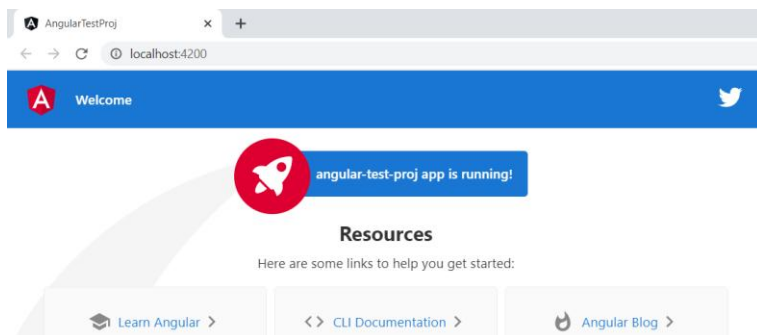


Рисунок 2.8 – Зовнішній вигляд застосунку

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

```
1 import { enableProdMode } from '@angular/core';
2 import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
3
4 import { AppModule } from './app/app.module';
5 import { environment } from './environments/environment';
6
7 if (environment.production) {
8   enableProdMode();
9 }
10
11 platformBrowserDynamic().bootstrapModule(AppModule)
12   .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).

```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AppComponent } from './app.component';
5
6 @NgModule({
7   declarations: [
8     AppComponent
9   ],
10   imports: [
11     BrowserModule
12   ],
13   providers: [],
14   bootstrap: [AppComponent]
15 })
16 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).

```
1  import { Component } from '@angular/core';
2
3  @Component({
4      selector: 'app-root',
5      templateUrl: './app.component.html',
6      styleUrls: ['./app.component.scss']
7  })
8  export class AppComponent {
9      title = 'angular-test-proj';
10 }
```

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-кодi, 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).

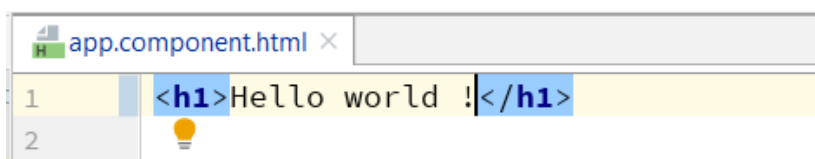
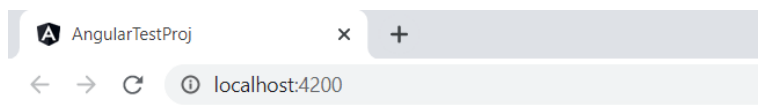


Рисунок 2.14 – Виконання змін у app.component.html

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`.



Hello world !

Рисунок 2.15 – Зовнішній вигляд сторінки після виконання змін

2.2. Робота із компонентами

Додаємо до проекту власну компоненту. Це зручніше робити, 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).

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-my',
5    template: `<div>
6      <p>This is my component</p>
7    </div>`,
8    styles: [`div {
9      border: dashed 1px black;
10    }
11    p {
12      margin: 15px;
13    }`]
14  })
15  export class MyComponent {
16
17  }
```

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).

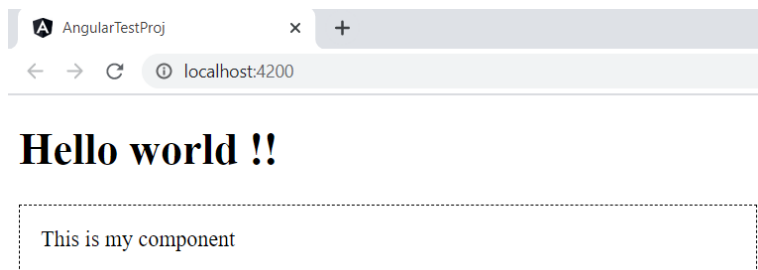


Рисунок 2.20 – Зміни на сторінці застосунку

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 компонентів, 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 src/app/my створимо файли my.component.html та my.component.scss такого вмісту (рис. 2.21–2.22).

```
1 <div>
2   <h3>This is my component</h3>
3   <p>version 0.0.2</p>
4 </div>
```

Рисунок 2.21 – Файл my.component.html

```
1 div {
2   border: dashed 1px black;
3   padding: 15px;
4   h3 {
5     color: blue;
6     font-weight: bold;
7   }
8   p {
9     margin: 0px;
10  }
11 }
```

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).

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-my',
5   templateUrl: 'my.component.html',
6   styleUrls: ['my.component.scss']
7 })
8 export class MyComponent {
9
10 }
```

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).

```
1  div.group {  
2      padding: 20px;  
3      background: linen;  
4      border-radius: 5px;  
5      h3 {  
6          margin-top: 0px;  
7      }  
8      p.faculty {  
9          color: blue;  
10     }  
11     p.specialty {  
12         font-size: 0.8em;  
13         color: brown;  
14     }  
15 }
```

Рисунок 2.27 – Файл group.component.scss

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

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

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

Hello world !!

Група 301

факультет комп'ютерних наук

спеціальність: комп'ютерні науки

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).

```
8 export class GroupComponent implements OnInit {  
9  
10     number = '308';  
11     faculty = `факультет комп'ютерних наук`;   
12     specialty = `інженерія програмного забезпечення`;   
13  
14     constructor() { }
```

Рисунок 2.30 – Додавання змінних до класу GroupComponent

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

```
1 <div class="group">  
2     <h3>Група {{ number }}</h3>  
3     <p class="faculty">{{ faculty }}</p>  
4     <p class="specialty">спеціальність: {{ specialty }}</p>  
5 </div>
```

Рисунок 2.31 – Зміни у шаблоні group.component.html

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

Hello world !!

Група 308

факультет комп'ютерних наук

спеціальність: інженерія програмного забезпечення

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).

```
10 number = '308';
11 faculty = `факультет комп'ютерних наук`;
12 specialty = `інженерія програмного забезпечення`;
13
14 studentsQuantity = 22;
15 students = [
16     'Іванов Василь',
17     'Смірнова Марія',
18     'Максимов Іван'
19 ];
20 starosta = {
21     name: 'Іванов Василь',
22     age: 20,
23     address: 'Mykolaiv, Ukraine'
24 }
```

Рисунок 2.33 – Додавання нових полів до класу GroupComponent

```
1 <div class="group">
2   <h3>Група {{ number }}</h3>
3   <p class="faculty">{{ faculty }}</p>
4   <p class="specialty">спеціальність: {{ specialty }}</p>
5   <p>Кількість студентів: {{studentsQuantity}}</p>
6   <p>Склад групи: {{students}}</p>
7   <p>Староста: {{starosta}}</p>
8 </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.

```
6 <p>Склад групи: {{students}}</p>  
7 <p>Староста:</p><pre>{{starosta | json}}</pre>  
8 </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).

```
28 ngOnInit() {  
29   setTimeout( handler: () => {  
30     this.faculty = '';  
31     this.specialty = '';  
32   }, 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.

```
25 images = [  
26   'https://image.mel.fm/i/I/IoT9VfRJLP/590.png',  
27   'https://kartinki-vernizazh.ru/_ph/478/2/550191551.jpg?1577384607',  
28   'https://i.pinimg.com/474x/7d/14/ba/7d14bac4b8a2ff32151088ed0c020d6.jpg'  
29 ];  
30 curImageIndex = 0;  
31 curImage: string;  
32  
33 constructor() { }  
34  
35 ngOnInit() {  
36   this.curImage = this.images[this.curImageIndex];  
37   setInterval( callback: () => {  
38     this.curImageIndex++;  
39     if (this.curImageIndex >= this.images.length) {  
40       this.curImageIndex = 0;  
41     }  
42     this.curImage = this.images[this.curImageIndex];  
43   }, ms: 2000);  
}
```

Рисунок 2.40 – Зміни у group.component.ts

```
1 <div class="group">  
2   <img [src]="curImage">  
3   <h3>Група {{ number }}</h3>  
4   <div class="faculty">{{ faculty }}</div>  
5 </div>
```

Рисунок 2.41 – Зміни у group.component.html

```
15 img {  
16   width: 300px;  
17   margin-bottom: 20px;  
18 }
```

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).

```
35 ngOnInit() {  
36   this.curImage = this.images[this.curImageIndex];  
37   setInterval( callback: () => {  
38     this.changeCurImage( forward: true);  
39   }, ms: 10000);  
40 }
```

```
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).

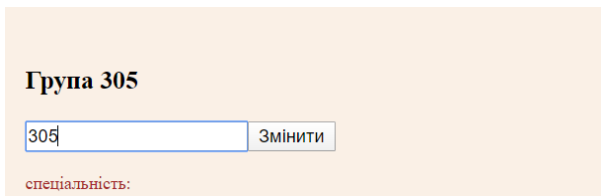


Рисунок 2.49 – Вигляд сторінки після додавання поля вводу

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.

```
10 <div class="edit-number">
11   <input #numb type="text" (input)="changeNumber (numb.value)">
12 <!-- <button (click)="changeNumber (numb.value)">Змінити</button>-->
13 </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).

```
6 import { GroupComponent } from './group/group.component';
7 import { FormsModule } from '@angular/forms';
8
12 ],
13 imports: [
14   BrowserModule,
15   FormsModule
16 ],
```

Рисунок 2.51 – Підключення FormsModule

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

```
10 <div class="edit-number">
11   <input type="text" [(ngModel)]="number">
12 <!-- <input #numb type="text" (input)="changeNumber (numb.value)"-->
13 <!-- <button (click)="changeNumber (numb.value)">Змінити</button>-->
14 </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).

```
5  h3 {  
6    margin-top: 0px;  
7    &.old {  
8      color: darkblue;  
9      font-size: 1.5em;  
10   }  
11 }
```

Рисунок 2.53 – Зміни у group.component.scss

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

```
61  
62  isOld() {  
63    return (+this.number.toString()[0]) > 2;  
64  }
```

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).

```
9    <h3 [ngClass]="{old: isOld()}">Група {{ number }}</h3>  
10   <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. Приховасмо 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).

```
11  спеціальтy = інженерія програмного забезпечення ;  
12  showInfo = false;  
13
```

Рисунок 2.56 – Додавання поля до класу GroupComponent

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

```
1 <div class="group">
2   <div class="img-container" *ngIf="showInfo">
3     <button (click)="changeCurImage(false)"><< prev</button>
4     <button (click)="changeCurImage(true)">next >></button>
5     <div>
6       <img [src]="curImage">
7     </div>
8   </div>
9
10  <h3 [ngClass]="{old: isOld()}">Група {{ number }}</h3>
11
12  <div *ngIf="showInfo; then btnHide else btnShow"></div>
13  <ng-template #btnHide>
14    <button (click)="showInfo=false">Згорнути</button>
15  </ng-template>
16  <ng-template #btnShow>
17    <button (click)="showInfo=true">Детальніше</button>
18  </ng-template>
19
20  <div class="info" *ngIf="showInfo">
21    <div class="edit-number">
22      <input type="text" [(ngModel)]="number">
23      <!-- <input #numb type="text" (input)="chang
24      <!-- <button (click)="changeNumber(numb.valu
25    </div>
26    <p class="faculty">{{ faculty }}</p>
27    <p class="specialty">спеціальність: {{ specialty }}</p>
28    <p>Кількість студентів: {{studentsQuantity}}</p>
29    <p>Склад групи: {{students}}</p>
30    <p>Староста:</p><pre>{{starosta | json}}</pre>
31  </div>
32</div>
```

Рисунок 2.57 – Використання директиви ngIf

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

Група 308

Згорнути

308

спеціальність:

Кількість студентів: 22

Склад групи: Іванов Василь,Смірнова Марія,Максимов Іван

Група 308

Детальніше

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).

```
8 export class AppComponent {
9   title = 'angular-test-proj';
10  groups = [
11    {
12      number: 301,
13      faculty: `факультет комп'ютерних наук`,
14      specialty: `комп'ютерні науки`
15    },
16    {
17      number: 308,
18      faculty: `факультет комп'ютерних наук`,
19      specialty: `інженерія програмного забезпечення`
20    },
21  ];
22 }
```

Рисунок 2.59. Масив студентських груп

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

```
1 <h1>Hello world !!</h1>
2 <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).

```
1 import {Component, Input, OnInit} from '@angular/core';
2
3 @Component({
4   selector: 'app-group',
5   templateUrl: './group.component.html',
6   styleUrls: ['./group.component.scss']
7 })
8 export class GroupComponent implements OnInit {
9   @Input() group;
10  showInfo = false;
11
12  constructor() { }
13
14  ngOnInit() {
15  }
16
17  isOld() {
18    return (+this.group.number.toString()[0]) > 2;
19  }
20 }
```

Рисунок 2.61 – Зміни у group.component.ts

```
1 <div class="group">
2   <h3 [ngClass]="{old: isOld()}">Група {{ group.number }}</h3>
3
4   <div *ngIf="showInfo; then btnHide else btnShow"></div>
5   <ng-template #btnHide>
6     <button (click)="showInfo=false">Згорнути</button>
7   </ng-template>
8   <ng-template #btnShow>
9     <button (click)="showInfo=true">Детальніше</button>
10  </ng-template>
11
12  <div class="info" *ngIf="showInfo">
13    <div class="edit-number">
14      <input type="text" [(ngModel)]="group.number">
15    </div>
16    <p class="faculty">{{ group.faculty }}</p>
17    <p class="specialty">спеціальність: {{ group.specialty }}</p>
18    <p>Кількість студентів: {{group.studentsQuantity}}</p>
19  </div>
20 </div>
```

Рисунок 2.62 – Зміни у group.component.html

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


```
1 <ng>hello world </ng>  
2 <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).

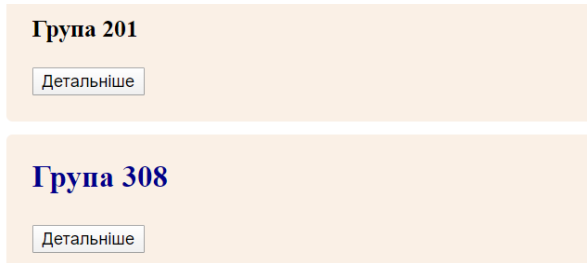


Рисунок 2.64 – Перевірка виведення студентських груп

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).

```
1 import {Component, EventEmitter,  
2       Input, OnInit, Output} from '@angular/core';  
10 @Input() group;  
11 @Output() dataChange = new EventEmitter();  
12 showInfo = false;  
23 onChange() {  
24   this.dataChange.emit(new Date());  
25 }
```

Рисунок 2.65 – Додавання повернення даних

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

```
13 <div class="edit-number">
14   <input type="text"
15     [(ngModel)]="group.number"
16     (ngModelChange)="onChange()">
17 </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).

```
9   title = 'angular-test-proj';
10   changeDate = new Date();
11   groups = [
25     onDataChange(event) {
26       this.changeDate = event;
27     }
  ]
```

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).

```
1 <h1>Hello world !!</h1>
2 <p>last data change:
3   <span>{{changeDate|date:'dd.MM.yyyy HH:mm:ss'}}</span>
4 </p>
5 <app-group
6   *ngFor="let group of groups"
7   [group]="group"
8   (dataChange)="onDataChange($event)">
9 </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).

```
29  },
30  "styles": [
31    "src/assets/materialize.min.css",
32    "src/styles.scss"
33  ],
34  "scripts": [
35    "src/assets/materialize.min.js"
36  ]
37  }
```

Рисунок 2.69 – Підключення materialize

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

```
7  <meta name="viewport" content="width=device-width, initial-scale=1">
8  <link rel="icon" type="image/x-icon" href="favicon.ico">
9  <link rel="stylesheet"
10 href="https://fonts.googleapis.com/icon?family=Material+Icons">
11 </head>
12 <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).

```
8  export class AppComponent {
9    groups = [
10     {
11       number: 201,
12       faculty: `факультет комп'ютерних наук`,
13       specialty: `комп'ютерні науки`,
14       studentsQuantity: 22
15     },
16     {
17       number: 308,
18       faculty: `факультет комп'ютерних наук`,
19       specialty: `інженерія програмного забезпечення`,
20       studentsQuantity: 24
21     },
22   ];
23 }
```

Рисунок 2.71 – app.component.ts

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

```
1 <nav>
2   <div class="nav-wrapper">
3     <a class="brand-logo">Групи</a>
4   </div>
5 </nav>
6 <div class="container">
7   <div class="row" style="margin-top: 20px;">
8     <div class="col m6 offset-m3 s12">
9       <a class="btn-floating btn-large waves-effect waves-light green">
10         <i class="material-icons">add</i>
11       </a>
12     </div>
13   </div>
14   <app-group
15     *ngFor="let group of groups"
16     [group]="group">
17   </app-group>
18 </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).

```
9 export class GroupComponent implements OnInit {
10   @Input() group;
11   showInfo = false;
12
13   constructor() { }
14
15   ngOnInit() {
16   }
17 }
```

Рисунок 2.73 – group.component.ts

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

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

Розробка гібридних застосунків для цифрової трансформації бізнесу

```
9         keyboard_arrow_up
10     </i>
11     <i *ngIf="!showInfo" class="material-icons">
12         keyboard_arrow_down
13     </i>
14 </a>
15 </div>
16 </div>
17
18 <div class="info" *ngIf="showInfo">
19     <div class="input-field">
20         <input id="number" [(ngModel)]="group.number"
21             type="text" class="validate">
22         <label for="number" class="active">Номер групи</label>
23     </div>
24     <div class="input-field">
25         <input id="faculty" [(ngModel)]="group.faculty"
26             type="text" class="validate">
27         <label for="faculty" class="active">Факультет</label>
28     </div>
29     <div class="input-field">
30         <input id="specialty" [(ngModel)]="group.specialty"
31             type="text" class="validate">
32         <label for="specialty" class="active">Спеціальність</label>
33     </div>
34     <div class="input-field">
35         <input id="quantity" [(ngModel)]="group.studentsQuantity"
36             type="number" class="validate">
37         <label for="quantity" class="active">Кількість студентів</label>
38     </div>
39     <a class="rem-row btn-floating waves-effect waves-light red">
40         <i class="material-icons">delete</i>
41     </a>
42 </div>
```

Рисунок 2.74 – group.component.html

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

```
1 div.info {
2     background: #fff5fd;
3     border-radius: 10px;
4     padding: 10px;
5 }
```

Рисунок 2.75 – Group.component.scss

Переглянемо отриманий результат (рис. 2.76).

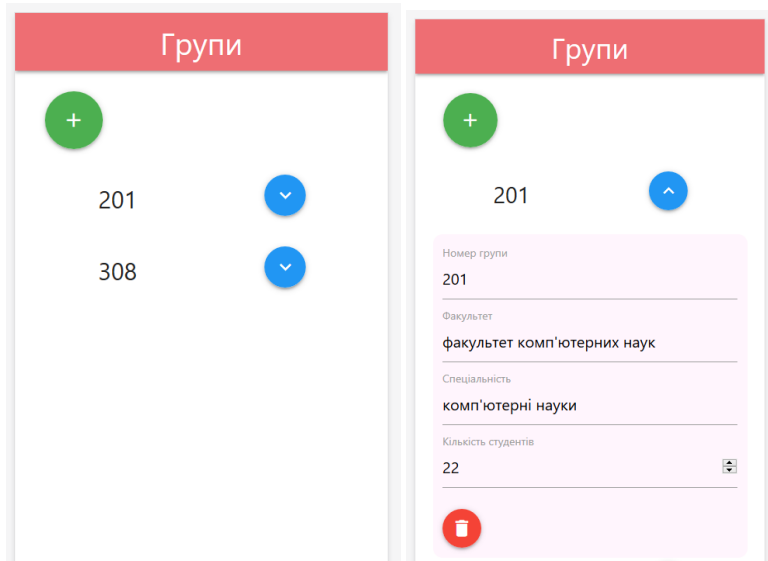


Рисунок 2.76 – Перегляд зовнішнього вигляду сторінки

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 {
```

Розробка гібридних застосунків для цифрової трансформації бізнесу

```
9
10 @Output() group = new EventEmitter();
11 showForm = false;
12
13 constructor() { }
14
15 ngOnInit() {
16 }
17
18 onSubmit(myForm) {
19   const fields = myForm.form.controls;
20   this.showForm = false;
21   this.group.emit( value: {
22     number: fields.number.value,
23     faculty: fields.faculty.value,
24     specialty: fields.specialty.value,
25     studentsQuantity: fields.studentsQuantity.value
26   });
27 }
28 }
```

Рисунок 2.78 – new-group.component.ts

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

```
1 <div class="row">
2   <div class="col m6 offset-m3 s12">
3     <a *ngIf="!showForm" (click)="showForm=true"
4       class="btn-floating btn-large waves-effect waves-light green">
5       <i class="material-icons">add</i>
6     </a>
7     <form *ngIf="showForm" #myForm="ngForm" novalidate
8       (ngSubmit)="onSubmit(myForm)">
9       <h5>Нова група</h5>
10      <div class="input-field">
11        <input id="number" name="number" ngModel
12          required type="text">
13        <label for="number">Номер групи</label>
14      </div>
15      <div class="input-field">
16        <input id="faculty" name="faculty" ngModel
17          required type="text">
18        <label for="faculty">Факультет</label>
19      </div>
20      <div class="input-field">
21        <input id="specialty" name="specialty" ngModel
22          required type="text">
23        <label for="specialty">Спеціальність</label>
24      </div>
25      <div class="input-field">
26        <input id="quantity" name="studentsQuantity" ngModel
27          required type="number">
28        <label for="quantity">Кількість студентів</label>
29      </div>
30    </div>
```

```
31 <div class="form-group">
32   <button type="button" (click)="showForm=false"
33     class="rem-row btn-floating waves-effect
34     waves-light left blue-grey">
35     <i class="material-icons">keyboard_arrow_up</i>
36   </button>
37   <button type="submit" [disabled]="myForm.invalid"
38     class="rem-row btn-floating waves-effect
39     waves-light green right">
40     <i class="material-icons">done</i>
41   </button>
42 </div>
43 </form>
44 </div>
45 </div>
```

Рисунок 2.79 – new-group.component.html

Також додаємо трохи власної стилізації (рис. 2.80).

```
1 div.row {
2   margin-top: 20px;
3 }
```

Рисунок 2.80 – new-group.component.scss

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

```
1 <nav>
2   <div class="nav-wrapper">
3     <a class="brand-logo">Групи</a>
4   </div>
5 </nav>
6 <div class="container">
7   <app-new-group (group)="addGroup($event)"></app-new-group>
8   <app-group
9     *ngFor="let group of groups"
10    [group]="group">
11 </app-group>
12 </div>
```

Рисунок 2.81 – Зміни у app.component.html

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

```
23
24 addGroup(group) {
25   this.groups.push(group);
26 }
```

Рисунок 2.82 – Зміни в app.component.ts

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

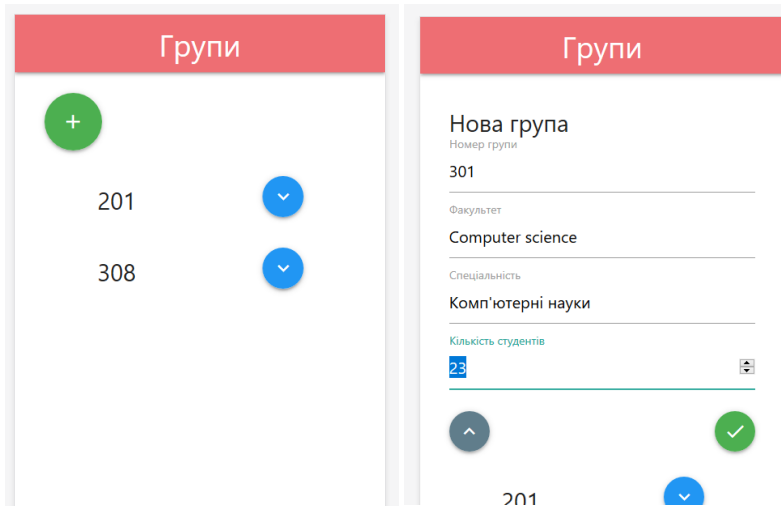


Рисунок 2.83 – Додавання нової групи

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 елемента (рис. 2.84).

```
9 export class GroupComponent implements OnInit {
10     @Input() group;
11     @Input() grpIndex;
12     @Output() removeGroup = new EventEmitter();
13     showInfo = false;
14
15
16
17
18
19
20     delGroup() {
21         this.removeGroup.emit(this.grpIndex);
22     }
23 }
```

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 елемента 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.html

У класі AppComponent створюємо метод deleteGroup, що відповідатиме за видалення елемента 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.

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

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

```
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class GroupsDataService {
7    private groups = [
8      {
9        number: 201,
10       faculty: 'факультет комп'ютерних наук',
11       specialty: 'комп'ютерні науки',
12       studentsQuantity: 22
13     },
14     {
15       number: 308,
16       faculty: 'факультет комп'ютерних наук',
17       specialty: 'інженерія програмного забезпечення',
18       studentsQuantity: 24
19     },
20   ];
21
22   constructor() { }
23
24   getGroups() {
25     return this.groups;
26   }
27
28   addGroup(group) {
29     this.groups.push(group);
30   }
31
32   deleteGroup(index) {
33     this.groups.splice(index, deleteCount: 1);
34   }
35 }
```

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).

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

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

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).

```
1 <nav>
2   <div class="nav-wrapper">
3     <a class="brand-logo">Групи</a>
4   </div>
5 </nav>
6 <div class="container">
7   <app-new-group></app-new-group>
8   <app-group
9     *ngFor="let group of groups; let i = index"
10    [group]="group" [grpIndex]="i">
11 </app-group>
12 </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).

```
10 export class GroupComponent implements OnInit {
11   @Input() group;
12   @Input() grpIndex;
13   showInfo = false;
14
15   constructor(private groupService: GroupsDataService) { }
16
17   ngOnInit() {
18   }
19
20   delGroup() {
21     this.groupService.deleteGroup(this.grpIndex);
22   }
23 }
```

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).

```
9  export class NewGroupComponent implements OnInit {  
10  
11      showForm = false;  
12  
13      constructor(private groupService: GroupsDataService) { }  
14  
15      ngOnInit() {  
16      }  
17  
18      onSubmit(myForm) {  
19          const fields = myForm.form.controls;  
20          this.showForm = false;  
21          this.groupService.addGroup( group: {  
22              number: fields.number.value,  
23              faculty: fields.faculty.value,  
24              specialty: fields.specialty.value,  
25              studentsQuantity: fields.studentsQuantity.value  
26          });  
27      }  
28  }
```

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 підпишемося 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 працювати 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).

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

Рисунок 2.99 – `group-list.component.ts`

```
1  <h5 class="center-align">Список груп</h5>  
2  <app-new-group></app-new-group>  
3  <app-group  
4    *ngFor="let group of groups; let i = index"  
5    [group]="group" [grpIndex]="i">  
6  </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).

```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AppComponent } from './app.component';
5 import { MyComponent } from './my/my.component';
6 import { GroupComponent } from './group/group.component';
7 import { FormsModule } from '@angular/forms';
8 import { NewGroupComponent } from './new-group/new-group.component';
9 import { RouterModule, Routes } from '@angular/router';
10 import { GroupListComponent } from './group-list/group-list.component';
11 import { StudentListComponent } from './student-list/student-list.component';
12
13 const routes: Routes = [
14   {path: 'groups', component: GroupLayoutComponent},
15   {path: 'students/:grpId', component: StudentListComponent},
16   {path: '', redirectTo: 'groups', pathMatch: 'full'}
17 ];
18
19 @NgModule({
20   declarations: [
21     AppComponent, MyComponent, GroupComponent, NewGroupComponent,
22     GroupLayoutComponent, StudentListComponent
23   ],
24   imports: [
25     BrowserModule,
26     FormsModule,
27     RouterModule.forRoot(routes)
28   ],
29   providers: [],
30   bootstrap: [AppComponent]
31 })
32 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).

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.scss']
7 })
8 export class AppComponent {
9
10   constructor() { }
11 }
```

Рисунок 2.102 – `App.component.ts`

```
1 <nav>
2   <div class="nav-wrapper">
3     <a class="brand-logo">Групи</a>
4   </div>
5 </nav>
6 <div class="container">
7   <router-outlet></router-outlet>
8 </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 студентів students та метод getStudents, що повертає список студентів певної групи, загорнутий у observable (рис. 2.104).

```
23 private students = [
24   {name: 'Іванов Василь', groupNum: 201},
25   {name: 'Дмитренко Петро', groupNum: 201},
26   {name: 'Петренко Дарина', groupNum: 201},
27   {name: 'Васильєва Марина', groupNum: 308},
28   {name: 'Павлов Микита', groupNum: 308},
29   {name: 'Васін Андрій', groupNum: 308},
30 ];
42
43 getStudents(groupNumber: number): Observable<any[]> {
44   return of(this.students.filter( callbackfn: elem => {
45     return elem.groupNum === groupNumber;
46   }));
47 }
```

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).

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

Рисунок 2.105 – Додавання посилання у group.component.html

Переглядаємо результат (рис. 2.106).

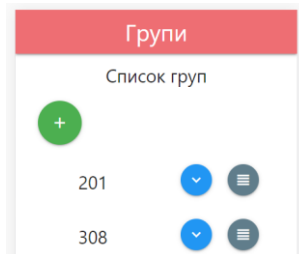


Рисунок 2.106 – Посилання на список студентів

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
19   ngOnInit() {
20     this.activatedRoute.params.subscribe(
21       next: params => this.getStudents(params.grpId)
22     );
23   }
24
25   getStudents(numb: string) {
26     const n = +numb;
27     this.groupService.getStudents(n).subscribe(
28       next: (students) => {
29         this.students = students;
30       }
31     );
32   }
33 }
```

Рисунок 2.107 – Клас `StudentsListComponent`

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

```
1 <div class="row valign-wrapper">
2   <div class="col m6 offset-m3 s12">
3     <a [routerLink] = "'/groups'"
4       class="btn-floating waves-effect waves-light blue-grey">
5       <i class="material-icons">
6         keyboard_arrow_left
7       </i>
8     </a>
9   </div>
10 </div>
11 <div class="row valign-wrapper">
12   <div class="col m6 offset-m3 s12">
13     <h5 class="center-align">Список студентів</h5>
14   </div>
15 </div>
16 <div *ngFor="let student of students" class="row valign-wrapper">
17   <div class="col m6 offset-m3 s12">
18     <h6>{{student.name}}</h6>
19   </div>
20 </div>
```

Рисунок 2.108 – Представлення `students-list.component.html`
Переглянемо отриманий результат (рис. 2.109).

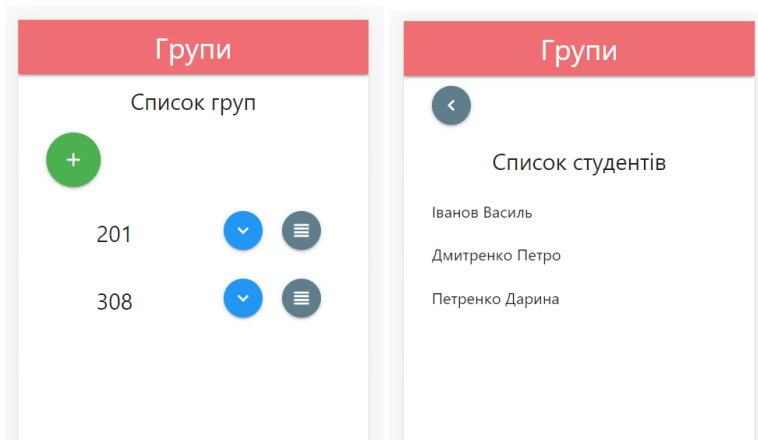


Рисунок 2.109 – Список студентів

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.  
[REDACTED].....] - 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 іоніс (рис. 3.4).

```
\code\ion-tabs> ionic start first-ion tabs
? Framework: (Use arrow keys)
> Angular | https://angular.io
  React   | https://reactjs.org
f issues. Please, upgrade your dependencies to the actual
[.....] | fetchMetadata: sill range ma
```

Рисунок 3.4 – Розгортання нового проєкту іоніс

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».

```
PS C:\data\personal\michael\MOHYLA\Education\SPA_PWA\block-3\code\ion-tabs\first-ion> ionic serve
> ng.cmd run app:serve --host=localhost --port=8100
[ng] Browserslist: caniuse-lite is outdated. Please run next command `npm update`

[INFO] ... and 78 additional chunks
[ng] i |wdm]: Compiled successfully.
```

Рисунок 3.5 – Запуск проєкту на виконання

Після запуску застосунок відкривається у браузері за замовчуванням. 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).

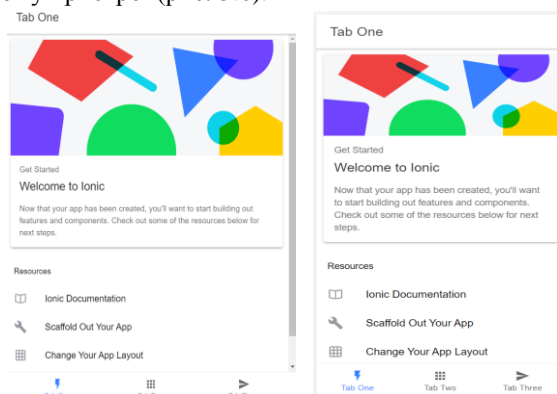


Рисунок 3.6 – Перегляд застосунку

Розробка гібридних застосунків для цифрової трансформації бізнесу

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 -F @ionic/lab
```

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

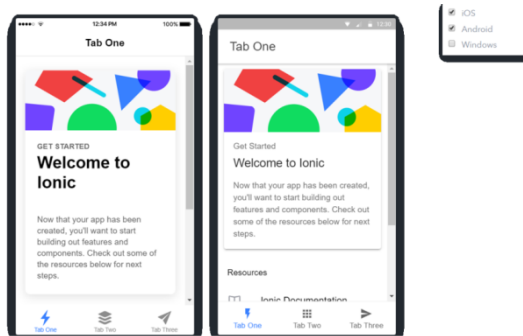


Рисунок 3.8 – Вигляд застосунку у режимі `labs`

Для зупинки виконання проєкту у терміналі натиснемо `Ctrl+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.

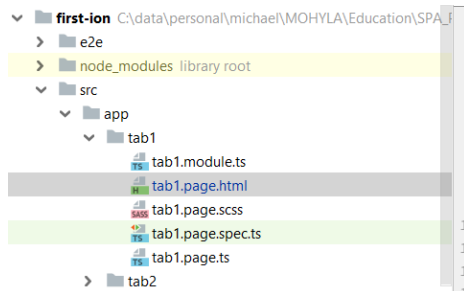


Рисунок 3.9 – Місцезнаходження `tab1.page.html`

```
1 <ion-header>
2   <ion-toolbar>
3     <ion-title>
4       Перший проект Ionic
5     </ion-title>
6   </ion-toolbar>
7 </ion-header>
8
9 <ion-content>
10   <ion-card class="welcome-card">
11     
14     <ion-card-header>
15       <ion-card-subtitle>Початок роботи</ion-card-subtitle>
16       <ion-card-title>Практична робота 1</ion-card-title>
17     </ion-card-header>
18     <ion-card-content>
19       <p>Створення тестового tab-застосунку на базі ionic-framework</p>
20     </ion-card-content>
21   </ion-card>
22 </ion-content>
```

Рисунок 3.10 – Змінений home.html

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

`first-ion>ng serve`

Рисунок 3.11 – Запуск проекту

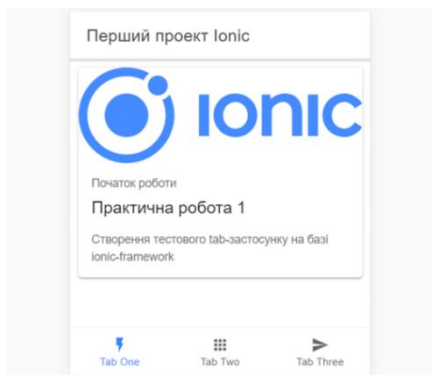


Рисунок 3.12 – Вигляд першої вкладки після виконання змін

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).

Розробка гібридних застосунків для цифрової трансформації бізнесу

```
1 <ion-header>
2   <ion-toolbar>
3     <ion-title>
4       Додаткові відомості
5     </ion-title>
6   </ion-toolbar>
7 </ion-header>
8
9 <ion-content>
10  <ion-list lines="none">
11    <ion-list-header>
12      <ion-label>Протягом даної роботи ми</ion-label>
13    </ion-list-header>
14    <ion-item>
15      <ion-icon slot="start" color="medium" name="book"></ion-icon>
16      <ion-label class="ion-text-wrap">
17        Познайомимось із фреймворком Ionic
18      </ion-label>
19    </ion-item>
20    <ion-item>
21      <ion-icon slot="start" color="medium" name="add"></ion-icon>
22      <ion-label class="ion-text-wrap">Створимо новий проект tabs</ion-label>
23    </ion-item>
24    <ion-item>
25      <ion-icon slot="start" color="medium" name="build"></ion-icon>
26      <ion-label class="ion-text-wrap">Виконаємо невеликі зміни</ion-label>
27    </ion-item>
28  </ion-list>
29 </ion-content>
```

Рисунок 3.13 – Зміни на другій вкладці

```
1 <ion-tabs>
2
3   <ion-tab-bar slot="bottom">
4     <ion-tab-button tab="tab1">
5       <ion-icon name="logo-ionic"></ion-icon>
6       <ion-label>Головна</ion-label>
7     </ion-tab-button>
8
9     <ion-tab-button tab="tab2">
10      <ion-icon name="information-circle-outline"></ion-icon>
11      <ion-label>Додатково</ion-label>
12    </ion-tab-button>
13  </ion-tab-bar>
14
15 </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'.
```

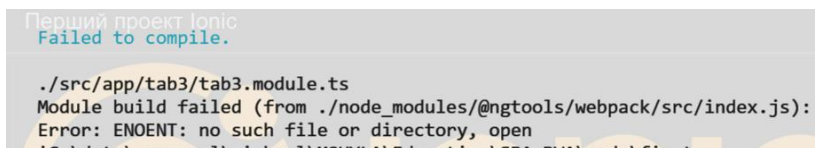


Рисунок 3.15 – Помилка після видалення файлів вкладки

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

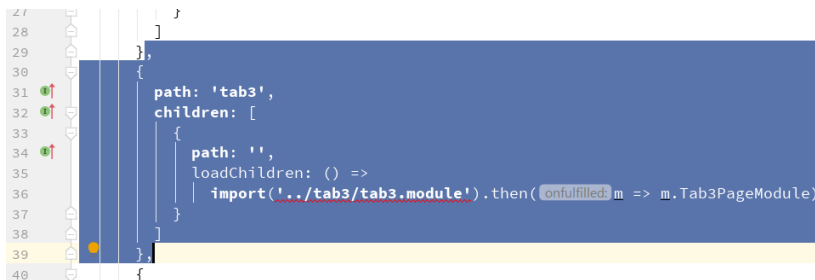


Рисунок 3.16 – Видалення непотрібної вкладки з `tabs-routing.module.ts`

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 проекту. Для цього виконаємо команду створення нової сторінки `ionic g page mytab` (рис. 3.18).

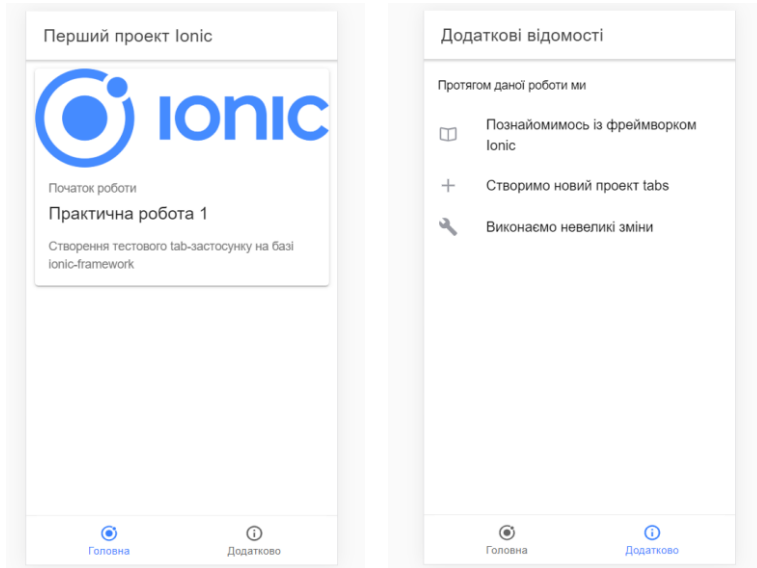


Рисунок 3.17 – Вигляд застосунку після виконання змін

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

Рисунок 3.18 – Створення нової сторінки

Відкриємо `app-routing.module.ts` та видалимо новий елемент масиву `Routes` (рис. 3.19), створивши аналогічний у файлі `tabs-routing.module.ts` (рис. 3.20).

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

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

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

Рисунок 3.19 – 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).

```
13  
14 <ion-tab-button tab="mytab">  
15   <ion-icon name="clipboard"></ion-icon>  
16   <ion-label>Нова вкладка</ion-label>  
17 </ion-tab-button>  
18 </ion-tab-bar>
```

Рисунок 3.20 – Додавання кнопки для переходу на вкладку

```
1 <ion-header>  
2   <ion-toolbar>  
3     <ion-title>Нова вкладка</ion-title>  
4   </ion-toolbar>  
5 </ion-header>  
6  
7 <ion-content>  
8   <h1 class="ion-text-center">Тестовий контент</h1>  
9 </ion-content>
```

Рисунок 3.21 – Додавання тексту на вкладку

Переглянемо результат (рис. 3.22).

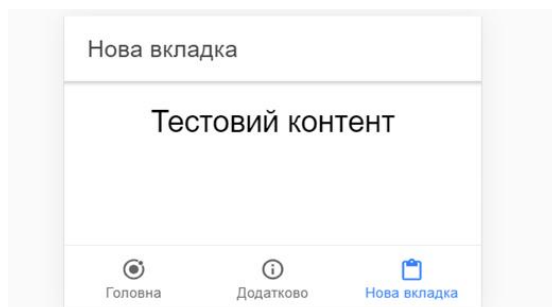


Рисунок 3.22 – Результат додавання нової вкладки

3.2. Створення застосунку «список студентських груп»

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

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

Рисунок 3.23 – Розгортання нового проекту ionic blank

Відкриємо проект у редакторі коду та додаємо до проекту 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>).

```
1 <ion-header>
2   <ion-toolbar>
3     <ion-title>
4       Студентські групи
5     </ion-title>
6   </ion-toolbar>
7 </ion-header>
8
9 <ion-content>
10  <ion-list>
11    <ion-list-header>
12      <ion-label>Групи</ion-label>
13      <ion-button shape="round" fill="outline"
14        color="success" (click)="add()">
15        <ion-icon slot="start" name="add"></ion-icon>Додати
16      </ion-button>
17    </ion-list-header>
18
19    <ion-item-sliding *ngFor="let group of groups">
20      <ion-item-options side="start">
21        <ion-item-option color="primary" (click)="edit(group)">
22          <ion-icon name="create"></ion-icon>
23          Змінити
24        </ion-item-option>
25        <ion-item-option color="danger" (click)="delete(group)">
26          <ion-icon name="trash"></ion-icon>
27          Видалити
28        </ion-item-option>
29      </ion-item-options>
30      <ion-item>
31        <ion-icon name="people" slot="start"></ion-icon>
32        <ion-label>{{group.number}}</ion-label>
33        <ion-note slot="end">{{group.specialty}}</ion-note>
34      </ion-item>
35    </ion-item-sliding>
36  </ion-list>
37 </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).

```
1 ion-item-sliding {  
2   ion-icon {  
3     margin-right: 15px;  
4   }  
5 }
```

Рисунок 3.28 – Файл `ion-item-sliding`

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

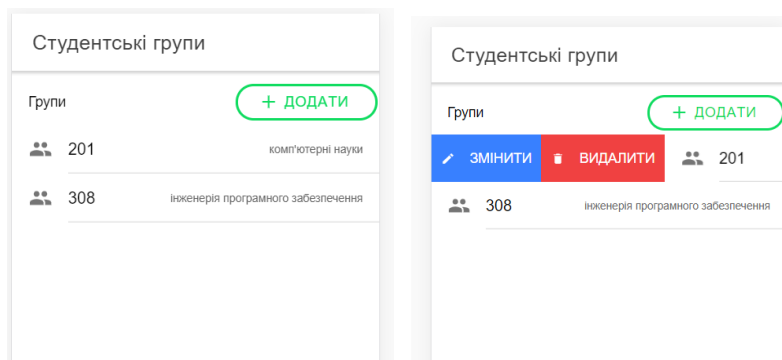


Рисунок 3.29 – Сторінка списку студентських груп

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).


```
1 import {Component, EventEmitter, Input, OnInit, Output} from
  '@angular/core';
2 import {StudGroup} from '../service/data-getter.service';
3
4 @Component({
5   selector: 'app-stud-group',
6   templateUrl: './stud-group.component.html',
7   styleUrls: ['./stud-group.component.scss'],
8 })
9 export class StudGroupComponent implements OnInit {
10
11   @Input() studentGroup: StudGroup;
12   @Input() isNew: boolean;
13   @Output() addGroup = new EventEmitter();
14   @Output() cancelAddingGroup = new EventEmitter();
15   title: string;
16
17   constructor() {}
18
19   ngOnInit() {
20     if (this.isNew) {
21       this.studentGroup = {
22         number: null,
23         specialty: '',
24         studentsQuantity: null,
25         faculty: ''
26       };
27       this.title = 'Нова група';
28     }
29   }
30
31   addNew() {
32     if (this.isNew) {
33       this.addGroup.emit(this.studentGroup);
34     }
35   }
36
37   cancelAdding() {
38     if (this.isNew) {
39       this.cancelAddingGroup.emit();
40     }
41   }
42 }
```

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).

```
1 <ion-card>
2   <ion-card-header>
3     <ion-card-title>{{title}}</ion-card-title>
4   </ion-card-header>
5
6   <ion-card-content>
7     <ion-item>
8       <ion-label position="floating">Номер</ion-label>
9       <ion-input type="number" [(ngModel)]="studentGroup.number"
10        name="number"></ion-input>
11     </ion-item>
12     <ion-item>
13       <ion-label position="floating">Факультет</ion-label>
14       <ion-input type="text" [(ngModel)]="studentGroup.faculty"
15        name="faculty"></ion-input>
16     </ion-item>
17     <ion-item>
18       <ion-label position="floating">Спеціальність</ion-label>
19       <ion-input type="text" [(ngModel)]="studentGroup.specialty"
20        name="specialty"></ion-input>
21     </ion-item>
22     <ion-item>
23       <ion-label position="floating">Студентів</ion-label>
24       <ion-input type="number" [(ngModel)]="studentGroup.studentsQuantity"
25        name="studentQuantity"></ion-input>
26     </ion-item>
27     <ion-button *ngIf="isNew" color="primary" (click)="addNew()">
28       <ion-icon slot="start" name="add-circle"></ion-icon> Додати
29     </ion-button>
30     <ion-button *ngIf="isNew" color="danger" class="ion-float-right"
31       (click)="cancelAdding()">
32       <ion-icon slot="start" name="close"></ion-icon> Відміна
33     </ion-button>
34   </ion-card-content>
35 </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).

```
1 <ion-header>
2   <ion-toolbar>
3     <ion-title>
4       Студентські групи
5     </ion-title>
6   </ion-toolbar>
7 </ion-header>
8
```

Розробка гібридних застосунків для цифрової трансформації бізнесу

```
9 <ion-content>
10   <ion-list>
11     <ion-list-header>
12       <ion-label>Групи</ion-label>
13       <ion-button shape="round" fill="outline"
14         color="success" (click)="add()">
15         <ion-icon slot="start" name="add"></ion-icon>Додати
16       </ion-button>
17     </ion-list-header>
18
19     <app-stud-group *ngIf="showNew" [isNew]="true"
20       (addGroup)="addGroup($event)"
21       (cancelAddingGroup)="showNew=false"></app-stud-group>
22
23     <div *ngFor="let group of groups;let i = index">
24       <ion-item-sliding>
25         <ion-item-options side="start">
26           <ion-item-option color="primary" (click)="showEdit=i">
27             <ion-icon name="create"></ion-icon>
28             Змінити
29           </ion-item-option>
30           <ion-item-option color="danger" (click)="delete(i)">
31             <ion-icon name="trash"></ion-icon>
32             Видалити
33           </ion-item-option>
34         </ion-item-options>
35         <ion-item (click)="showEdit=-1">
36           <ion-icon name="people" slot="start"></ion-icon>
37           <ion-label>{{group.number}}</ion-label>
38           <ion-note slot="end">{{group.specialty}}</ion-note>
39         </ion-item>
40       </ion-item-sliding>
41       <app-stud-group *ngIf="showEdit==i" [isNew]="false"
42         [studentGroup]="group"></app-stud-group>
43     </div>
44   </ion-list>
45 </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).

```
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   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 групи, змінюємо дані та видаляємо (рис. 3.35).

Студентські групи

Групи + ДОДАТИ

Нова група

Номер

Факультет

Спеціальність

Студентів

+ ДОДАТИ ✗ ВІДМІНА

201 комп'ютерні науки

308 інженерія програмного забезпечення

Студентські групи

Групи + ДОДАТИ

✓ ЗМІНИТИ ✗ ВИДАЛИТИ 201

Номер

201

Факультет

факультет комп'ютерних наук

Спеціальність

комп'ютерні науки

Студентів

22

308 інженерія програмного забезпечення

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).

```
1  <ion-header>
2    <ion-toolbar>
3      <ion-title>Студентські групи</ion-title>
4    </ion-toolbar>
5  </ion-header>
6
7  <ion-content>
8    <ion-grid>
9      <ion-row color="primary" justify-content-center>
10       <ion-col align-self-center size-md="6" size-lg="5" size-xs="12">
11         <div text-center>
12           <h3>ВХІД</h3>
13         </div>
14         <ion-item>
15           <ion-label position="floating">Ім'я користувача</ion-label>
16           <ion-input type="text" name="username"></ion-input>
17         </ion-item>
18         <ion-button color="primary" class="ion-float-right" (click)="login()">
19           <ion-icon slot="start" name="log-in"></ion-icon> Увійти
20         </ion-button>
21       </ion-col>
22     </ion-row>
23   </ion-grid>
24 </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).

```
1 import { Component, OnInit } from '@angular/core';
2 import { Router } from '@angular/router';
3
4 @Component({
5   selector: 'app-login',
6   templateUrl: './login.page.html',
7   styleUrls: ['./login.page.scss'],
8 })
9 export class LoginPage implements OnInit {
10   constructor(private router: Router) { }
11
12   ngOnInit() {
13   }
14
15   login() {
16     this.router.navigate( commands: ['/home']);
17   }
18 }
```

Рисунок 3.38 – Файл login.page.ts

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

```
1 ion-header {
2   margin-bottom: 25px;
3 }
4
5 ion-item {
6   margin-bottom: 10px;
7 }
```

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).

```
4 const routes: Routes = [
5   { path: '', redirectTo: 'login', pathMatch: 'full' },
6   { path: 'home', loadChildren: () => import('./home/home.module').then(
7     onfulfilled: m => m.HomePageModule)},
8   {
9     path: 'login',
10    loadChildren: () => import('./login/login.module').then(
11      onfulfilled: m => m.LoginPageModule)
12  },
13 ];
```

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).

```
1 <ion-header>
2   <ion-toolbar>
3     <ion-title>
4       Студентські групи
5     </ion-title>
6     <ion-buttons slot="secondary">
7       <ion-button routerLink="/login" routerDirection="root">
8         <ion-icon slot="icon-only" name="log-out"></ion-icon>
9       </ion-button>
10    </ion-buttons>
11  </ion-toolbar>
12</ion-header>
```

Рисунок 3.41 – Зміни у `home.page.html`

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

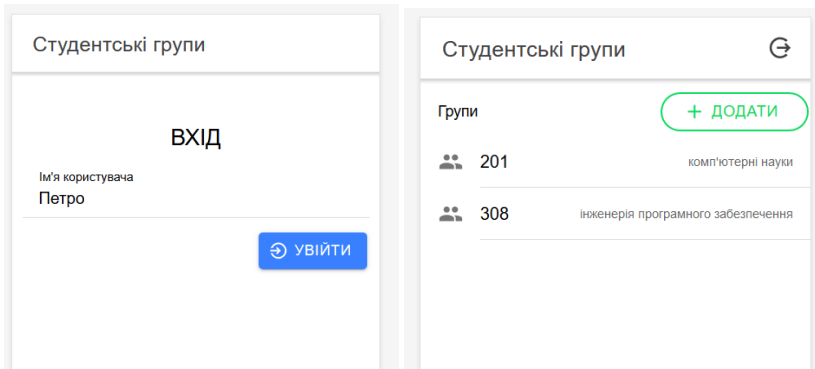


Рисунок 3.42 – Вхід до застосунку

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).

```
31     private userName = '';
32
33     private users = [
34         'Василь', 'Петро', 'Олена'
35     ];
36
37     getUser() {
38         return this.userName;
39     }
40
41     setUser(name: string) {
42         this.userName = name;
43     }
44
45     userExists(name: string): boolean {
46         return this.users.indexOf(name) !== -1;
47     }
48
49     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).

```
1  import { Component, OnInit } from '@angular/core';
2  import { Router } from '@angular/router';
3  import { DataGetterService } from '../service/data-getter.service';
4  import { AlertController } from '@ionic/angular';
5
6  @Component({
7      selector: 'app-login',
8      templateUrl: './login.page.html',
9      styleUrls: ['./login.page.scss'],
10  })
11  export class LoginPage implements OnInit {
12      userName: string;
13
14      constructor(
15          private router: Router,
16          private dataGetter: DataGetterService,
17          public alertController: AlertController) {}
18
19      ngOnInit() {
20      }
21
22      login() {
23          if (this.dataGetter.userExists(this.userName)) {
24              this.dataGetter.setUser(this.userName);
25              this.router.navigate([ '/home' ]);
26          } else {
27              this.userNotExistAlert();
28          }
29      }
30  }
```


Розробка гібридних застосунків для цифрової трансформації бізнесу

```
30
31 async userNotExistAlert() {
32   const alert = await this.alertController.create({
33     header: 'Увага !',
34     subHeader: 'Помилка аутентифікації',
35     message: `Користувача ${this.userName} не 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).

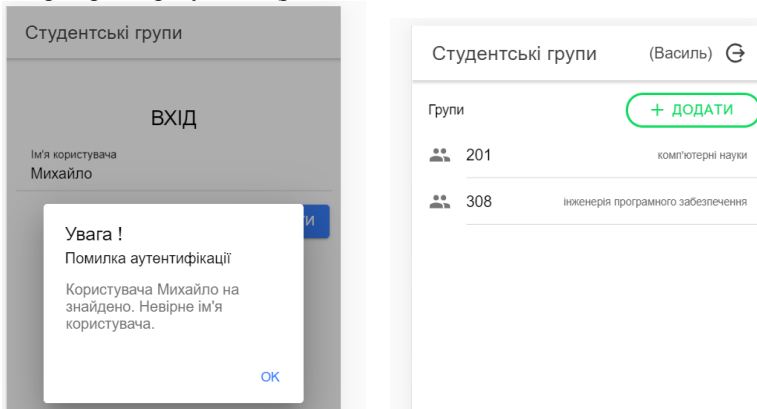


Рисунок 3.46 – Перевірка логін-форми

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

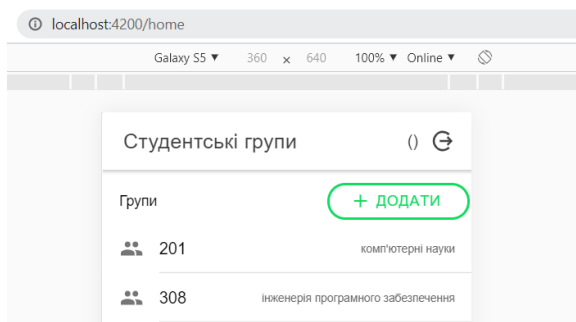


Рисунок 3.47 – Перехід до головної сторінки
в обхід сторінки автентифікації

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

```
1 import { Injectable } from '@angular/core';
2 import { ActivatedRouteSnapshot, CanActivate, Router, RouterStateSnapshot }
3   from '@angular/router';
4 import { DataGetterService } from '../service/data-getter.service';
5
6 @Injectable({
7   providedIn: 'root'
8 })
9 export class AuthGuard implements CanActivate {
10   constructor(private dataGetter: DataGetterService,
11               private router: Router) {}
12
13   canActivate(
14     next: ActivatedRouteSnapshot,
15     state: RouterStateSnapshot): boolean {
16
17     const isLoggenIn = this.dataGetter.getUser() !== '';
18     if (!isLoggenIn) {
19       this.router.navigateByUrl( url: '/login');
20     }
21     return isLoggenIn;
22   }
23 }
```

Рисунок 3.48 – Guard Auth

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

```
5 const routes: Routes = [  
6   { path: '', redirectTo: 'login', pathMatch: 'full' },  
7   {  
8     path: 'home',  
9     loadChildren: () => import('./home/home.module').then(  
10       onfulfilled: m => m.HomePageModule  
11     ),  
12     canActivate: [AuthGuard]  
13   }],
```

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 проектом «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).

```
5 const routes: Routes = [  
6   { path: '', redirectTo: 'login', pathMatch: 'full' },  
7   {  
8     path: 'home',  
9     loadChildren: () => import('./home/home.module').then(  
10       onfulfilled: m => m.HomePageModule  
11     ),  
12     canActivate: [AuthGuard]  
13   },  
14   {  
15     path: 'login',  
16     loadChildren: () => import('./login/login.module').then(  
17       onfulfilled: m => m.LoginPageModule  
18     )  
19   }],
```

```
20 {
21   path: 'students/:grpnumb',
22   loadChildren: () => import('./students/students.module').then(
23     onfulfilled: m => m.StudentsPageModule
24   ),
25   canActivate: [AuthGuard]
26 },
27 ];
```

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).

```
31 private students = [
32   {name: 'Іванов Василь', groupNumb: 201,
33     gender: 'man', rating: 91},
34   {name: 'Дмитренко Петро', groupNumb: 201,
35     gender: 'man', rating: 85},
36   {name: 'Петренко Дарина', groupNumb: 201,
37     gender: 'woman', rating: 68},
38   {name: 'Васильєва Марина', groupNumb: 308,
39     gender: 'woman', rating: 78},
40   {name: 'Павлов Микита', groupNumb: 308,
41     gender: 'man', rating: 82},
42   {name: 'Васін Андрій', groupNumb: 308,
43     gender: 'man', rating: 75},
44 ];
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78 getStudents(groupNumber: number): Observable<any[]> {
79   return of(this.students.filter( callbackfn: elem => {
80     return elem.groupNumb === groupNumber;
81   }));
82 }
```

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).

```
1 import {Component, OnInit} from '@angular/core';
2 import {DataGetterService} from '../service/data-getter.service';
3 import {ActivatedRoute} from '@angular/router';
4
5 @Component({
6   selector: 'app-students',
7   templateUrl: './students.page.html',
8   styleUrls: ['./students.page.scss'],
9 })
```

```
10 export class StudentsPage implements OnInit {
11   grpnumb: number;
12   students: any[];
13
14   constructor(private dataGetter: DataGetterService,
15               private route: ActivatedRoute) { }
16
17   ngOnInit() {
18     this.grpnumb = +this.route.snapshot.paramMap.get('grpnumb');
19     this.dataGetter.getStudents(this.grpnumb).subscribe(
20       next: data => {
21         this.students = data;
22       }
23     );
24   }
25 }
```

Рисунок 3.53 – Файл students.page.ts

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

```
1 <ion-header>
2   <ion-toolbar>
3     <ion-buttons slot="start">
4       <ion-back-button></ion-back-button>
5     </ion-buttons>
6     <ion-title>Студенти групи {{grpnumb}}</ion-title>
7   </ion-toolbar>
8 </ion-header>
9
10 <ion-content>
11   <ion-list>
12     <ion-item *ngFor="let student of students" text-wrap>
13       <ion-icon [name]="student.gender" slot="start"></ion-icon>
14       <ion-label text-wrap>{{student.name}}</ion-label>
15       <ion-note slot="end">{{student.rating}}</ion-note>
16     </ion-item>
17   </ion-list>
18 </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).



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).

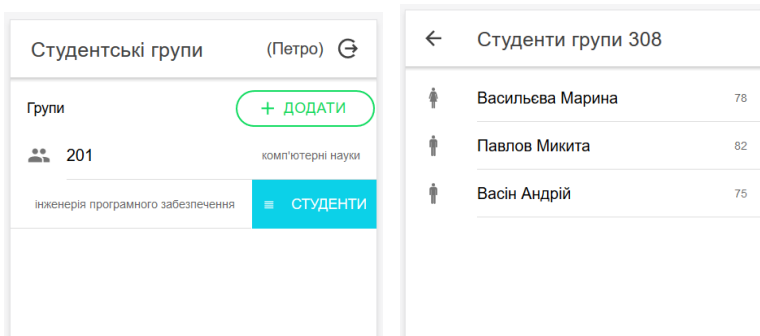


Рисунок 3.56 – Перехід до сторінки списку студентів

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)

Розробка гібридних застосунків для цифрової трансформації бізнесу

```
15 <ion-note slot="end">{{student.rating}}</ion-note>
16 </ion-item>
17 </ion-list>
18 <ion-item>
19 <ion-label position="floating">Дані на home</ion-label>
20 <ion-input type="text" #dataToHome</ion-input>
21 </ion-item>
22 <ion-button color="primary" class="ion-float-right" (click)="passData(
23   (dataToHome.value)">
24   <ion-icon slot="start" name="send"></ion-icon> Переслати на home
25 </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).

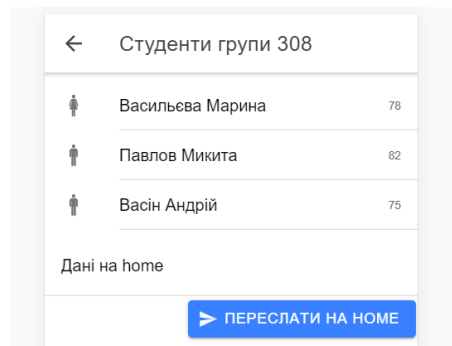


Рисунок 3.58 – Сторінка студентів після додавання поля вводу

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).

```
4 import {Events, NavController} from '@ionic/angular';
5
15 constructor(private dataGetter: DataGetterService,
16              private route: ActivatedRoute,
17              private events: Events,
18              private navCtrl: NavController) { }
19
28
29 passData(data: string) {
30   this.events.publish( topic 'students:data', data);
31   this.navCtrl.back();
32 }
```

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).

```
57 </div> [studentGroup]="group"></app-stud-group>
58
59 </ion-list>
60 <ion-item text-center>
61   <ion-label color="primary">{{dataFromStudents}}</ion-label>
62 </ion-item>
63 </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).

```
3 import {Events} from '@ionic/angular';
4
19 dataFromStudents: string;
20
21 constructor(private dataGetter: DataGetterService, private events: Events) {
22   this.dataGetter.getGroups().subscribe(
23     next: (data) => {
24       this.groups = data;
25     }
26   );
27   this.userName = this.dataGetter.getUser();
28   this.events.subscribe( topic 'students:data', handlers: data => {
29     this.dataFromStudents = data;
30   });
31 }
```

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

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

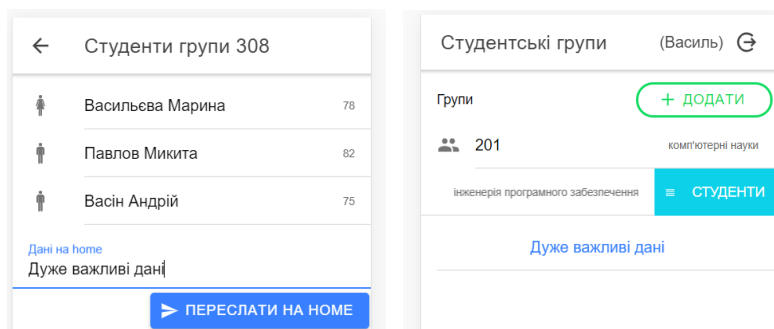


Рисунок 3.62 – Перевірка передачі даних

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.



Рисунок 3.64 – data-sender.page.html

```
1 import { Component, OnInit } from '@angular/core';
2 import { Events, NavController } from '@ionic/angular';
3 import { ActivatedRoute, Router } from '@angular/router';
4
5 @Component({
6   selector: 'app-data-sender',
7   templateUrl: './data-sender.page.html',
8   styleUrls: ['./data-sender.page.scss'],
9 })
10 export class DataSenderPage implements OnInit {
11
12   textData: string;
13
14   constructor(private events: Events,
15               private navCtrl: NavController,
16               private route: ActivatedRoute) {
17     this.textData = this.route.snapshot.paramMap.get('data');
18   }
```

```
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).

Розробка гібридних застосунків для цифрової трансформації бізнесу

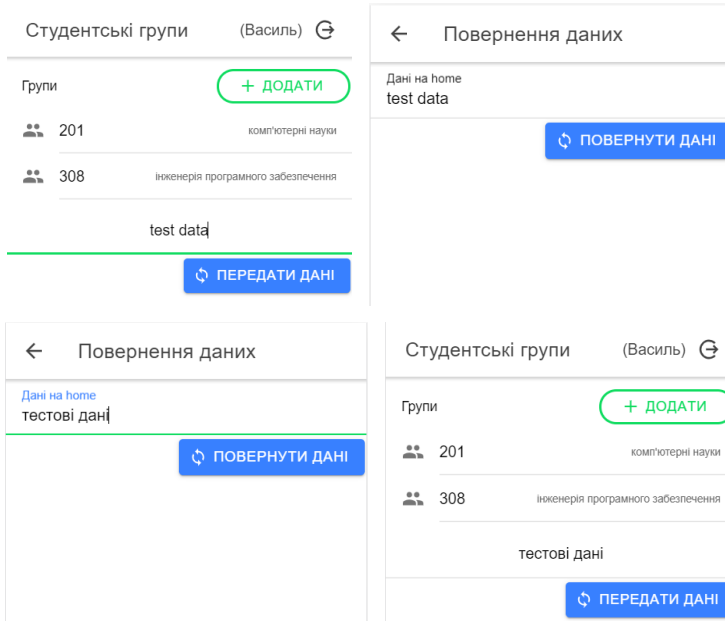


Рисунок 3.68 – Передача та повернення даних

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(

```

```
27     next: (data) => {
28         this.groups = data;
29     }
30 };
31 this.userName = this.dataGetter.getUser();
32 this.extraData = this.route.snapshot.paramMap.get('data');
33 }
```

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 service/DataExchanger

```
1  import { Injectable } from '@angular/core';
2
3  interface DataElem {
4      k: string;
5      d: any;
6  }
7
8  @Injectable({
9      providedIn: 'root'
10 })
11
12 export class DataExchangerService {
13
14     data: DataElem[] = [];
15
16     constructor() { }
17
18     getData(key: string) {
19         const arr = this.data.filter( callbackfn: elem => elem.k === key);
20         if (arr.length) {
21             return arr[0].d;
22         }
23         return '';
24     }
25
26     setData(key: string, data: any) {
27         const arr = this.data.filter( callbackfn: elem => elem.k === key);
28         if (arr.length) {
29             arr[0].d = data;
30         } else {
31             this.data.push({
32                 k: key, d: data
33             });
34         }
35     }
36 }
```

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 }  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

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 }  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

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).

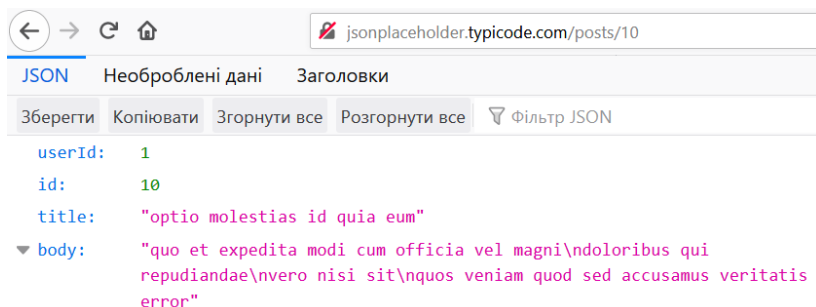


Рисунок 3.74 – Повернення публікації у json-форматі

Тепер отримаємо наведені дані з нашого іоніс-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



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).

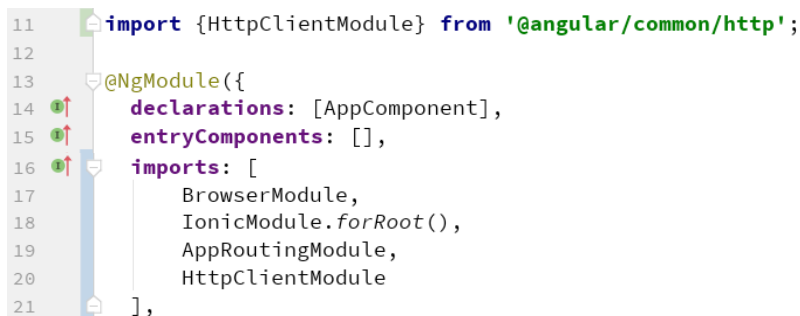


Рисунок 3.77 – Додавання модуля HttpClientModule до масиву imports

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

```
1 <ion-header>
2   <ion-toolbar>
3     <ion-title>HttpTest</ion-title>
4   </ion-toolbar>
5 </ion-header>
6
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-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).

```
1 import { Component, OnInit } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3
4 @Component({
5   selector: 'app-http-test',
6   templateUrl: './http-test.page.html',
7   styleUrls: ['./http-test.page.scss'],
8 })
9 export class HttpTestPage implements OnInit {
10
11   constructor(private http: HttpClient) { }
12
13   ngOnInit() {
14   }
15
16   getData() {
17     this.http.get( url: 'http://jsonplaceholder.typicode.com/posts/10')
18       .subscribe(
19         next: data => {
20           console.log(data);
21         }
22       );
23   }
24 }
```

Рисунок 3.79 – Клас http-test.page.ts

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

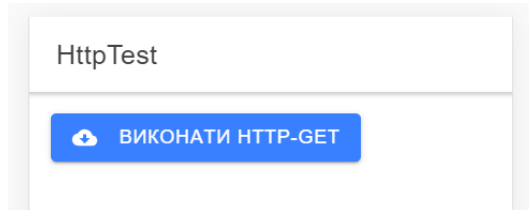


Рисунок 3.80 – Зовнішній вигляд сторінки http-test

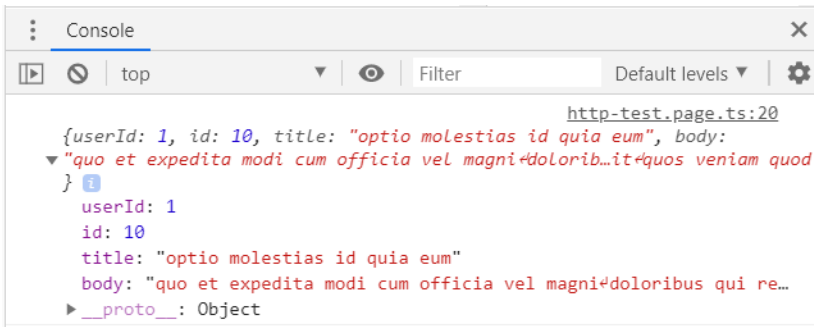


Рисунок 3.81 – Результат виведення результату запиту на консоль

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 }
```


Розробка гібридних застосунків для цифрової трансформації бізнесу

```
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).

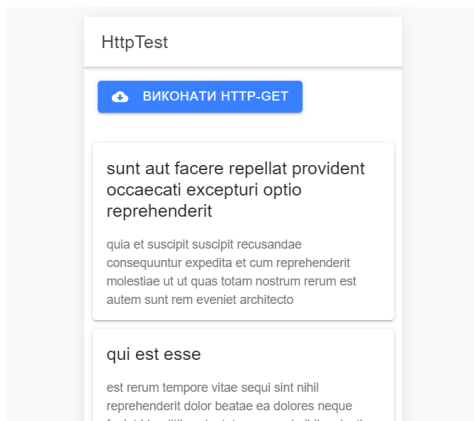


Рисунок 3.84 – Результат виведення списку публікацій

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).

```
7 <ion-content>
8   <ion-refresher slot="fixed" (ionRefresh)="getData($event)">
9     <ion-refresher-content
10       pullingIcon="arrow-dropdown"
11       pullingText="Pull to refresh"
12       refreshingSpinner="circles"
13       refreshingText="Refreshing...">
14   </ion-refresher-content>
15 </ion-refresher>
16 <ion-list>
17   <ion-card *ngFor ="let post of posts">
```

Рисунок 3.85 – http-test.page.html

```
17 ngOnInit() {
18   this.getData( refresher: false);
19 }
20
21 getData(refresher) {
22   this.http.get( url: 'http://jsonplaceholder.typicode.com/posts')
23     .pipe(map(
24       project: (res: Array<any>) => res.filter(
25         callbackfn: row => row.id < 10
26       )
27     )).subscribe(
28       next: data => {
29         this.posts = data;
30         if (refresher) {
31           refresher.target.complete();
32         }
33       }
34     );
35 }
36 }
```

Рисунок 3.86 – http-test.page.ts

Переглянемо отриманий результат (рис. 3.87).

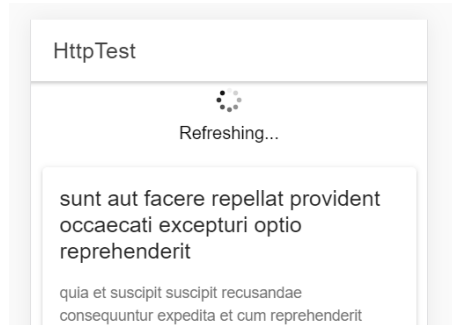


Рисунок 3.87 – Використання refresher для оновлення даних

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).

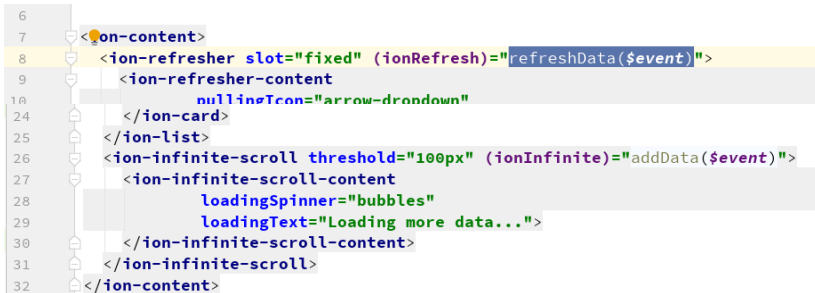


Рисунок 3.88 – Додавання елементу InfiniteScroll

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, завантажуюємо застосунок та переходимо у кінець списку (рис. 3.90).

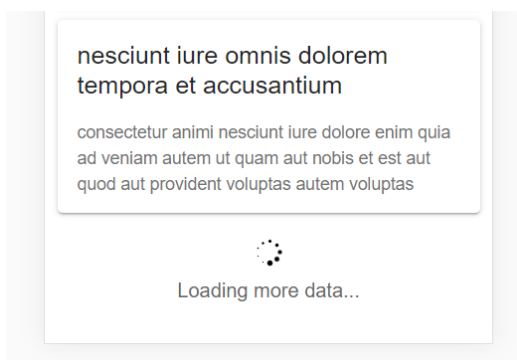


Рисунок 3.90 – Робота елемента InfiniteScroll

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 `opensever`.

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).

```
1  <?php
2  header( string: 'Content-type: application/json');
3  header( string: 'Access-Control-Allow-Origin: *');
4  header( string: 'Access-Control-Allow-Methods: GET, PUT, POST, DELETE,
5  OPTIONS');
6  header( string: 'Access-Control-Allow-Headers: Origin, Content-Type,
7  X-Auth-Token , Authorization');
8
9  $input = json_decode(file_get_contents( filename: 'php://input'),
10  assoc: true);
11
12  echo json_encode(
13  [
14  'get-data' => $_GET,
15  'input-data' => $input
16  ]
17  );
```

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).

```
20  ngOnInit() {
21    this.refreshData( refresher: false);
22
23    this.http.get( url: 'http://localhost/api?a=10&b=30').subscribe(
24      next: data => console.log(data)
25    );
26  }
```

```
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.

```
{get-data: {a: '10', b: '30'}, input-data: null}  
{get-data: Array(0), input-data: {a: 10, b: 30, c: 'text data'}}
```

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).

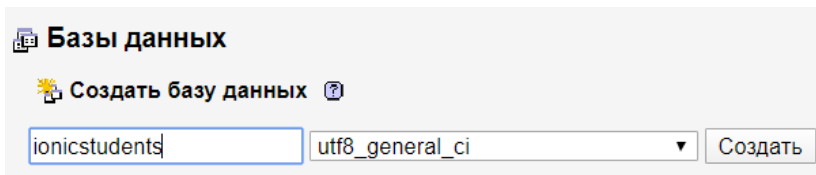


Рисунок 4.4 – Створення БД

Розробка гібридних застосунків для цифрової трансформації бізнесу

Создать таблицу

Имя: Количество столбцов:

Имя	Тип	Длина/Значения	По умолчанию	Сравнение	Атрибуты	Null	Индекс	AJ
id	INT		Нет			<input type="checkbox"/>	PRIMARY	<input checked="" type="checkbox"/>
number	VARCHAR	5	Нет			<input type="checkbox"/>	PRIMARY	<input type="checkbox"/>
faculty	VARCHAR	100	Нет			<input type="checkbox"/>		<input type="checkbox"/>
specialty	VARCHAR	100	Нет			<input type="checkbox"/>		<input type="checkbox"/>
studentsQuantity	INT		Нет			<input type="checkbox"/>		<input type="checkbox"/>

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Комментарии	Дополнительно
1	id	int(11)			Нет	Нет		AUTO_INCREMENT
2	number	varchar(5)	utf8_general_ci		Нет	Нет		
3	faculty	varchar(100)	utf8_general_ci		Нет	Нет		
4	specialty	varchar(100)	utf8_general_ci		Нет	Нет		
5	studentsQuantity	int(11)			Нет	Нет		

id	number	faculty	specialty	studentsQuantity
1	301	факультет комп'ютерних наук	Комп'ютерні науки	22
2	308	факультет комп'ютерних наук	Інженерія програмного забезпечення	24

Рисунок 4.5 – Таблица студентських груп

Создать таблицу

Имя: Количество столбцов:

Имя	Тип	Длина/Значения	По умолчанию	Сравнение	Атрибуты	Null	Индекс	AJ
id	INT		Нет			<input type="checkbox"/>	PRIMARY	<input checked="" type="checkbox"/>
name	VARCHAR	100	Нет			<input type="checkbox"/>		<input type="checkbox"/>
group_id	INT		Нет			<input type="checkbox"/>		<input type="checkbox"/>
gender	ENUM	'man' 'woman'	Нет			<input type="checkbox"/>		<input type="checkbox"/>
rating	INT		Нет			<input type="checkbox"/>		<input type="checkbox"/>

Структура таблиц **Связи**

Ограничения внешнего ключа

Действия	Свойства ограничения	Столбец	Ограничение внешнего ключа (INNODB)
ON DELETE RESTRICT	ON UPDATE RESTRICT	group_id	Базы данных: students, Таблица: groups, Столбец: id

Структура таблицы **Связи**

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Комментарии	Дополнительно
1	id	int(11)			Нет	Нет		AUTO_INCREMENT
2	name	varchar(100)	utf8_general_ci		Нет	Нет		
3	group_id	int(11)			Нет	Нет		
4	gender	enum('man', 'woman')	utf8_general_ci		Нет	Нет		
5	rating	int(11)			Нет	Нет		

	id	name	group_id	gender	rating
<input type="checkbox"/>	1	Іванов Василь	1	man	91
<input type="checkbox"/>	2	Дмитренко Петро	1	man	85
<input type="checkbox"/>	3	Петренко Дарина	1	woman	68
<input type="checkbox"/>	4	Васильева Марина	2	woman	78
<input type="checkbox"/>	5	Павлов Микита	2	man	72
<input type="checkbox"/>	6	Васін Андрій	2	man	75

Рисунок 4.6 – Таблица студентів

Создать таблицу

Имя: Количество столбцов:

Имя	Тип	Длина/Значения	По умолчанию	Сравнение	Атрибуты	Null	Индекс
id	INT		Нет			<input type="checkbox"/>	PRIMARY
username	VARCHAR	100	Нет			<input type="checkbox"/>	
passwd	VARCHAR	100	Нет			<input type="checkbox"/>	
token	VARCHAR	150	Нет			<input type="checkbox"/>	

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Комментарии	Дополнительно
1	id	int(11)			Нет	Нет		AUTO_INCREMENT
2	username	varchar(100)	utf8_general_ci		Нет	Нет		
3	passwd	varchar(100)	utf8_general_ci		Нет	Нет		
4	token	varchar(150)	utf8_general_ci		Нет	Нет		

Столбец	Тип	Функция	Null	Значение
id	int(11)			
username	varchar(100)			Василь
passwd	varchar(100)	MD5		1
token	varchar(150)			123

Вперед

id	username	passwd	token
1	Василь	c4ca4238a0b923820dcc509a6f75849b	202cb962ac59075b964b07152d234b70
2	Олена	c81e728d9d4c2f636f067f89cc14862c	caf1a3dfb505ffed0d024130f58c5cfa

Рисунок 4.7 – Таблица 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).

```

1  <?php
2  class Config {
3      public static $server = "localhost:3306";
4      public static $user = "root";
5      public static $pwd = "";
6      public static $db = "ionicstudents";
7  }
```

Рисунок 4.8 – config.php

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


```
1 <?php
2
3 class DB {
4     private $link;
5     public $err;
6     public function connect() {
7         $this->link = new \mysqli(
8             \Config::$server, \Config::$user, \Config::$pwd, \Config::$db
9         );
10        if (!$this->link) {
11            return false;
12        }
13        $this->runQuery( 'sql: "SET NAMES 'uft-8'" );
14        return true;
15    }
16    public function disconnect() {
17        $this->link->close();
18        unset($this->link);
19    }
20    public function runQuery($sql) {
21        if (!$this->link) {
22            $this->connect();
23        }
24        $res = $this->link->query($sql);
25        if (!$res) {
26            $this->err = $this->link->error;
27        }
28        return $res;
29    }
30    public function getArrFromQuery($sql) {
31        $res_arr = [];
32        $rs = $this->runQuery($sql);
33        while($row = $rs->fetch_assoc()) {
34            $res_arr[] = $row;
35        }
36        return $res_arr;
37    }
38 }
```

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).

```
1 <?php
2
3 class Model {
4     public static function getGroupsList() {
5         return (new DB())->getArrFromQuery(
6             'sql: "SELECT id, number, faculty, specialty, studentsQuantity
7             FROM groups
8             order by number" );
9     }
10 }
```

Рисунок 4.10 – Файл model.php

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

```
1  <?php
2
3  class Auth {
4      public static function getUserToken($user) {
5          $res = (new DB())->getArrFromQuery(
6              sql: "select id from users where username='".$user['username']."'
7                  and passwd=md5('".$user['passwd']."' )"
8          );
9          if (count($res) > 0) {
10             $id = $res[0]['id'];
11             $token = bin2hex(random_bytes( 64));
12             $res2 = (new DB())->runQuery(
13                 sql: "update users set token='".$token"' where id=$id"
14             );
15             if ($res2) {
16                 return ['token' => $token];
17             }
18         }
19         return ['error' => 'Username or password is incorrect'];
20     }
21
22     public static function checkToken($token) {
23         $res = (new DB())->getArrFromQuery(
24             sql: "select id from users where token='".$token"'
25         );
26         if (count($res) > 0) {
27             return true;
28         }
29         return false;
30     }
31 }
```

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).

```
1  <?php
2  class Controller {
3      private $data;
4      private $action;
5      private $protectedActions = ['get-groups'];
6
7      function __construct() {
8          $this->action = $_GET['action'];
9          $this->data = json_decode(file_get_contents( filenames: 'php://input'), assoc: true);
10     }
```

```
11
12 function run() {
13     $res = [];
14     if (in_array($this->action, $this->protectedActions)
15         && !Auth::checkToken($_GET['token'])) {
16         return ['error' => 'authentication failed !'];
17     }
18     switch ($this->action) {
19         case 'login':
20             $res = Auth::getUserToken($this->data);
21             break;
22         case 'get-groups':
23             $res = Model::getGroupsList();
24             break;
25         default:
26             $res = ['error' => 'this route is incorrect !'];
27             break;
28     }
29     echo json_encode($res);
30 }
31 }
```

Рисунок 4.12 – Файл controller.php

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

```
1 <?php
2 require 'config.php';
3 require 'db.php';
4 require 'model.php';
5 require 'auth.php';
6 require 'controller.php';
7
8 header( string: 'Content-type: application/json');
9 header( string: 'Access-Control-Allow-Origin: *');
10 header( string: 'Access-Control-Allow-Methods: GET, PUT, POST, DELETE,
11     OPTIONS');
12 header( string: 'Access-Control-Allow-Headers: Origin, Content-Type,
13     X-Auth-Token , Authorization');
14
15 $input = json_decode(file_get_contents( filename: 'php://input'), assoc: true);
16
17 (new Controller())->run();
```

Рисунок 4.13 – Файл index.php

Наступний крок – зміна іоніс-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).

```
15 export class DataGetterService {
16     baseUrl = 'http://localhost/api/';
17     groups = [];
18     students = [];
19     users = [];
20
21     constructor(private http: HttpClient) {
22     }
23
24     private userName = '';
25     private token = '';
26
27     checkUser(user) {
28         return this.http.post<any>(url: this.baseUrl + '?action=login',
29             user);
30     }
31
32     getUser() {
33         return this.userName;
34     }
35
36     setUser(name: string) {
37         this.userName = name;
38     }
39
40     setToken(token: string) {
41         this.token = token;
42     }
43
44     getGroups() {
45         return this.http.get<any>(url: this.baseUrl +
46             '?action=get-groups&token=' + this.token);
47     }
48
49     getStudents() {}
50 }
```

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

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

```
5 const routes: Routes = [
6     { path: '', redirectTo: 'login', pathMatch: 'full' },
7     {
```

Рисунок 4.15 – Редірект на сторінку login

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

```
14 <ion-item>
15   <ion-label position="floating">Ім'я користувача</ion-label>
16   <ion-input type="text" [(ngModel)]="userName"></ion-input>
17 </ion-item>
18 <ion-item>
19   <ion-label position="floating">Пароль</ion-label>
20   <ion-input type="password" [(ngModel)]="passWord"></ion-input>
21 </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).

```
11 export class LoginPage implements OnInit {
12   userName: string;
13   passWord: string;
23   login() {
24     this.dataGetter.checkUser( user: {
25       username: this.userName,
26       passwd: this.passWord
27     }).subscribe(
28       next: result => {
29         if (result.hasOwnProperty( v: 'error')) {
30           this.userNotExistAlert(result.error);
31         } else {
32           if (result.hasOwnProperty( v: 'token')) {
33             this.dataGetter.setUser(this.userName);
34             this.dataGetter.setToken(result.token);
35             this.router.navigate( commands: ['/home']);
36           } else {
37             this.userNotExistAlert( message: 'Unexpected error');
38           }
39         }
40       }
41     );
42   }
43
44   async userNotExistAlert(message) {
45     const alert = await this.alertController.create({
46       header: 'Увага !',
47       subHeader: 'Помилка аутентифікації',
48       message: message,
49       buttons: ['OK']
50     });
51
52     await alert.present();
53   }
```

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.

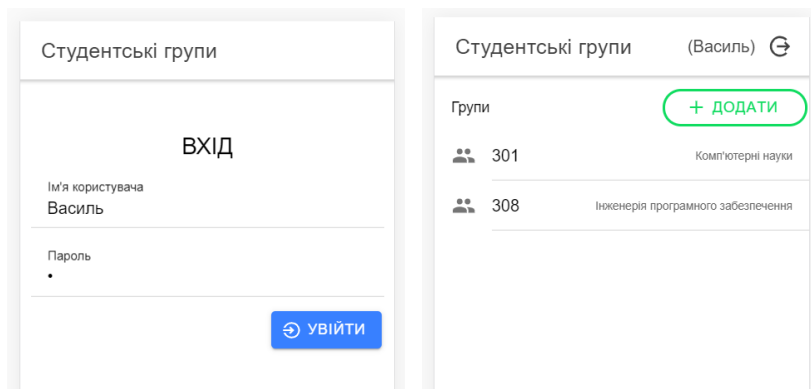


Рисунок 4.18 – Перевірка роботи за стосунку

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 = '". $group['faculty']. "',  
15              specialty = '". $group['specialty']. "',  
16              studentsQuantity = '". $group['studentsQuantity']. "'  
17              where id = '". $group['id'] );  
18  }
```

Рисунок 4.21 – Метод editGroup класу Model

```
5      private $protectedActions = ['get-groups', 'edit_group'];  
25  
26      case 'edit_group':  
27          if (Model::editGroup($this->data)) {  
28              $res = ['update' => 'success'];  
29          } else {  
30              $res = ['error' => 'groups update error !'];  
31          }  
32          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($group) {
21          return (new DB())->runQuery(
22              sql: "insert into groups(number, faculty, specialty,
23                  studentsQuantity)
24                  values('".$group['number']."' , ' " . str_replace( search: '"',
25                      replace: "\\'", $group['faculty'])."',
26                      ' " . str_replace( search: '"', replace: "\\'", $group['specialty'])
27                      ."',
28                      ".$group['studentsQuantity'].")");
29      }
30
31      public static function removeGroup($group) {
32          return (new DB())->runQuery(
33              sql: "delete from groups
34                  where id = " . $group['id'];
35      }
```

Рисунок 4.26 – Доповнення класу моделі API


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

Реалізуємо 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 Номе також корегуємо методи додавання та видалення елемента (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  
38  
39  
40  
    <ion-item-option color="danger" (click)="delete(group)">  
      <ion-icon name="trash"></ion-icon>  
      Видалити  
    </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  
48  
49  
50  
51  
52  
53  
54  
    <ion-item-options side="end">  
      <ion-item-option color="secondary">  
        [routerLink]="['/students/'+group.number, {id: group.id}]"  
        routerDirection="forward">  
        <ion-icon name="reorder"></ion-icon>  
        Студенти  
      </ion-item-option>  
    </ion-item-options>
```

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-сповіщення, 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 BaaS 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).

Розробка гібридних застосунків для цифрової трансформації бізнесу

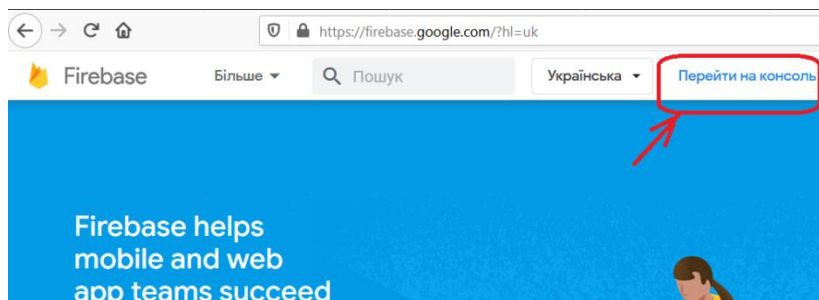


Рисунок 4.36 – Офіційний сайт Firebase

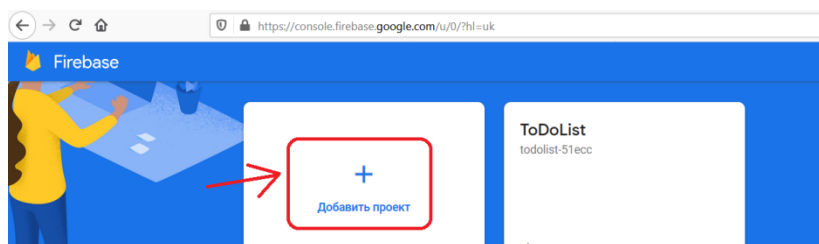


Рисунок 4.37 – Створення нового проекту

Вказуємо назву проекту (рис. 4.38).

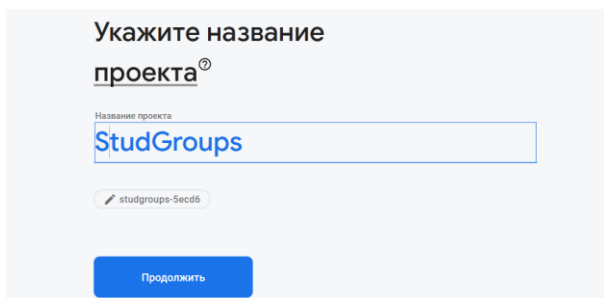


Рисунок 4.38 – Уведення назви проекту

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

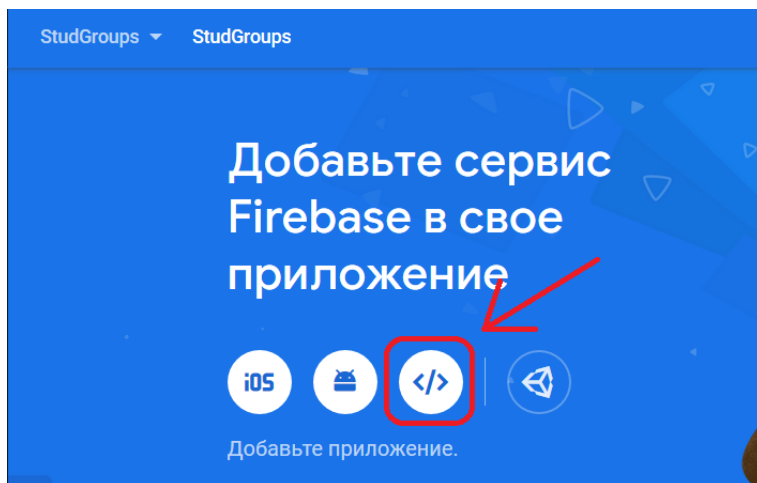


Рисунок 4.39 – Додавання застосунку до проекту

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

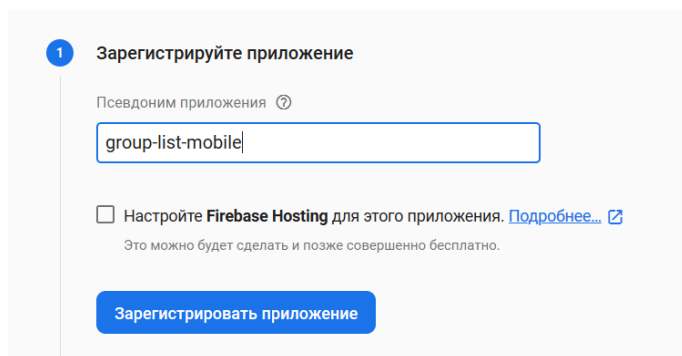


Рисунок 4.40 – Реєстрація застосунку

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).

Розробка гібридних застосунків для цифрової трансформації бізнесу

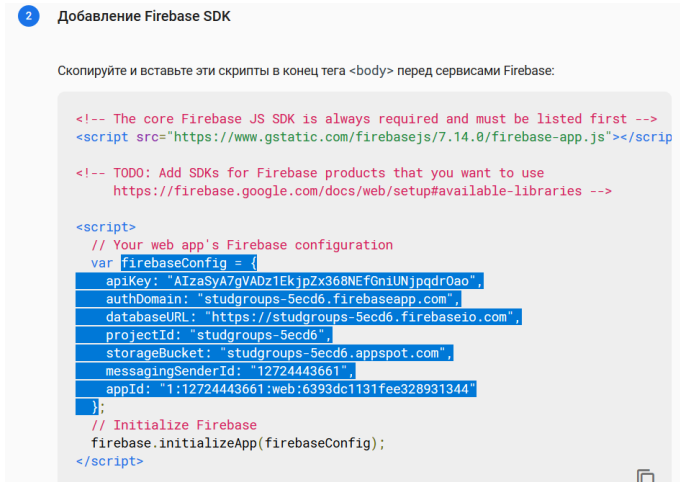


Рисунок 4.41 – Дані, необхідні на налаштування підключення до Firebase

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

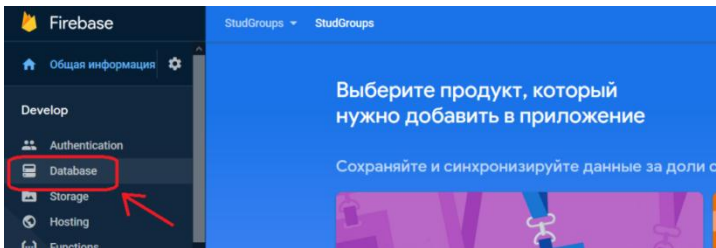


Рисунок 4.42 – Додавання бази даних

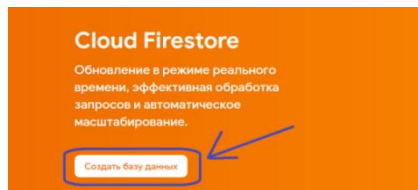


Рисунок 4.43 – Створення бази даних

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

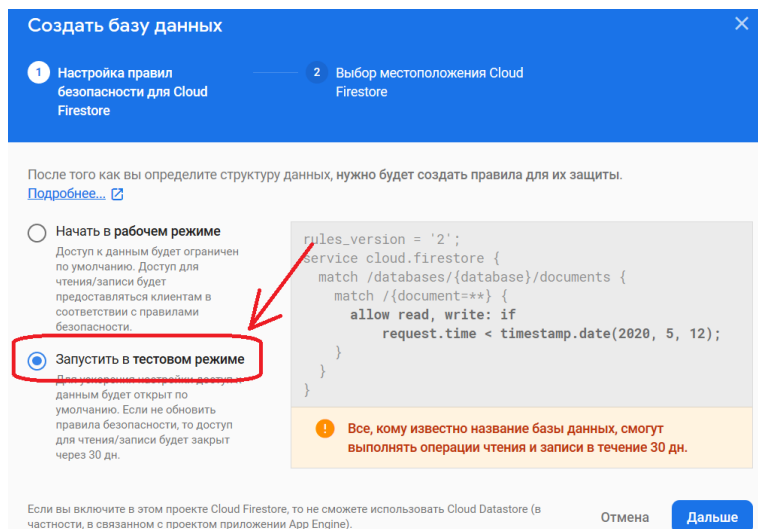


Рисунок 4.44 – Запуск у тестовому режимі

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

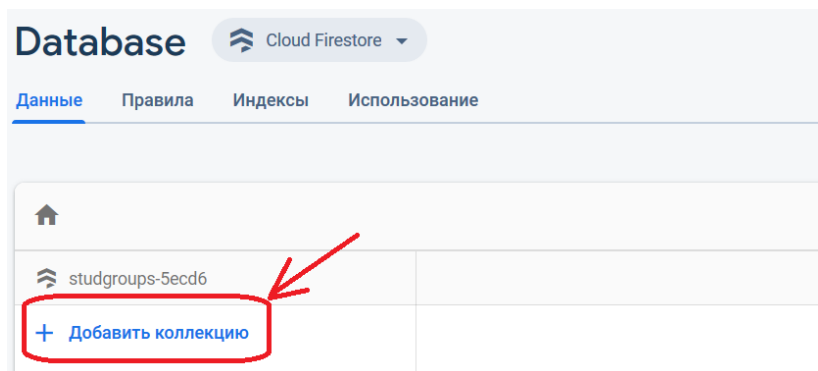


Рисунок 4.45 – Додавання нової колекції groups

Розробка гібридних застосунків для цифрової трансформації бізнесу

Добавить коллекцию

1 Добавление идентификатора коллекции

2 Добавление первого документа

Путь к родительскому объекту

/

Идентификатор коллекции

groups

Коллекция – это набор документов с данными.

Пример: в коллекцию users можно объединить документы с данными о пользователях.

Отмена Далее

Рисунок 4.46 – Введення ідентифікатора колекції

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

Идентификатор документа

Сгенерировать

Поле	Тип	Значение
number	= string	301
specialty	= string	computer scienc
studentsQuantity	= number	25
faculty	= string	computer scienc

Рисунок 4.47 – Додавання нового документа

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

groups > RSn4p8gtETdA21DvcI,n

studgroups-5ec06	groups	RSn4p8gtETdA21DvcI,n
+ Добавить коллекцию	+ Добавить документ	+ Добавить поле
groups >	RSn4p8gtETdA21DvcI,n >	

faculty: 'computer science'
number: '301'
specialty: 'computer science'
studentsQuantity: 25

Рисунок 4.48 – Стан БД після додавання колекції та документа

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

+ Додати документ	+ Додати колекцію
RSn4p8gtETdA21DVcIJn	
1UbACoUN7BrFnBC8G464 >	+ Додати поле
	faculty: "computer science"
	number: "308"
	specialty: "application engeneering"
	studentsQuantity: "27"

Рисунок 4.49 – Додавання документа до колекції

Далі переходимо до реалізації підключення до бази даних Firebase із нашого іоніс-застосунку. Починаємо із додавання до проєкту іоніс-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: 'AIzaSyA7gVADz1EkjpZx368NEfGniUNjpdqr0ao',  
9     authDomain: 'studgroups-5ecd6.firebaseio.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).

Розробка гібридних застосунків для цифрової трансформації бізнесу

```
9   import { AppComponent } from './app.component';
10  import { AppRoutingModule } from './app-routing.module';
11  import { HttpClientModule } from '@angular/common/http';
12  import { AngularFireModule } from '@angular/fire';
13  import {environment} from '../environments/environment';
14  import {AngularFirestoreModule} from '@angular/fire/storage';
15  import {AngularFirestore} from '@angular/fire/firestore';
16
17  @NgModule({
18    declarations: [AppComponent],
19    entryComponents: [],
20    imports: [
21      BrowserModule,
22      IonicModule.forRoot(),
23      AppRoutingModule,
24      HttpClientModule,
25      AngularFireModule.initializeApp(environment.firebase),
26      AngularFirestoreModule
27    ],
28    providers: [
29      StatusBar,
30      SplashScreen,
31      { provide: RouteReuseStrategy, useClass: IonicRouteStrategy },
32      AngularFirestore
33    ],
34  })
```

Рисунок 4.52 – Зміни у app.module.ts

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

```
3   "compilerOptions": {
4     "baseUrl": "./",
5     "outDir": "./dist/out-tsc",
6     "sourceMap": true,
7     "declaration": false,
8     "module": "esnext",
9     "moduleResolution": "node",
10    "emitDecoratorMetadata": true,
11    "experimentalDecorators": true,
12    "importHelpers": true,
13    "target": "es2015",
14    "typeRoots": [
15      "node_modules/@types"
16    ],
17    "lib": [
18      "es2018",
19      "dom"
20    ],
21    "skipLibCheck": true
22  },
```

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`

```
1 import { Injectable } from '@angular/core';
2 import { AngularFireStore } from '@angular/fire/firestore';
3 import { Observable } from 'rxjs';
4 import { map } from 'rxjs/operators';
5
6 @Injectable({
7   providedIn: 'root'
8 })
9
10 export class FireDataGetterService {
11
12   groups: Observable<any[]>;
13
14   constructor(private readonly afs: AngularFireStore) {
15     const groupsCollection = afs.collection<any>(path: 'groups');
16     this.groups = groupsCollection.snapshotChanges().pipe(
17       map( project: actions => actions.map( callbackfn: a => {
18         const data = a.payload.doc.data();
19         const id = a.payload.doc.id;
20         return { id, ...data };
21       })
22     );
23   }
24
25   getGroups() {
26     return this.groups;
27   }
28 }
```

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).

```
16 constructor(
17   private router: Router,
18   private dataGetter: DataGetterService,
19   public alertController: AlertController,
20   private fireDataGetter: FireDataGetterService) {}
21
22 ngOnInit() {
23   this.fireDataGetter.getGroups().subscribe(
24     next: data => console.log(data)
25   );
26 }
```

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

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

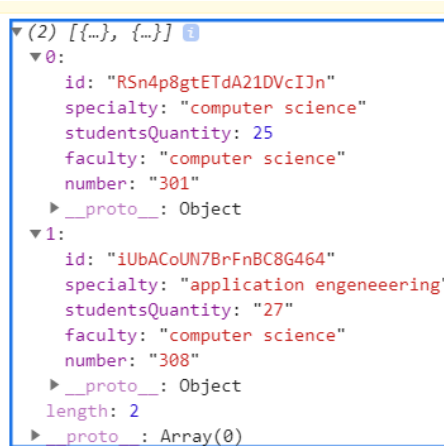


Рисунок 4.57 – Перевірка виводу списку груп на консоль

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).

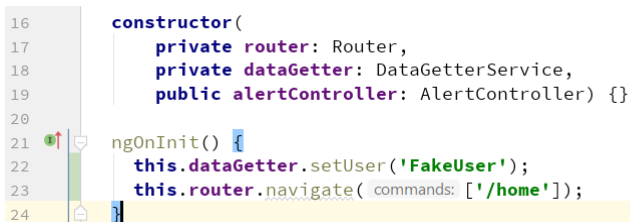


Рисунок 4.58 – Зміни у login.page.ts

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

```
5 import {FireDataGetterService} from '../services/fire-data-getter.service';
6
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.dataGetter.getUser();
31 }
```

Рисунок 4.59 – Зміни у home.page.ts

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

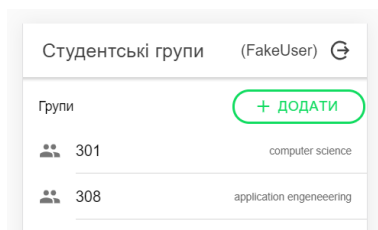


Рисунок 4.60 – Виведення списку студентських груп

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).

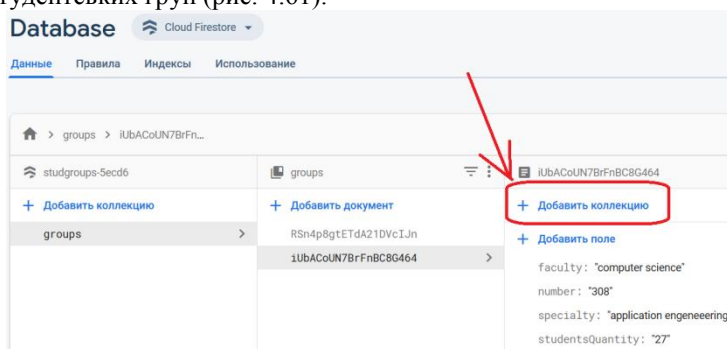
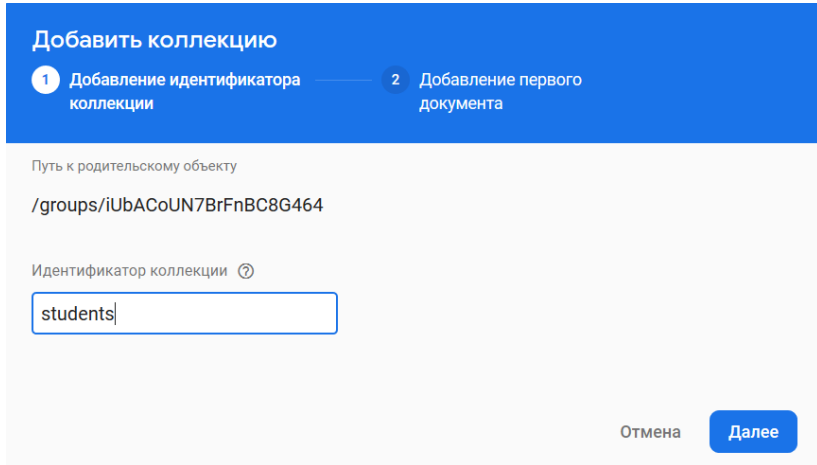


Рисунок 4.61 – Додавання колекції у документ

Розробка гібридних застосунків для цифрової трансформації бізнесу

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



Добавить коллекцию

1 Добавление идентификатора коллекции — 2 Добавление первого документа

Путь к родительскому объекту

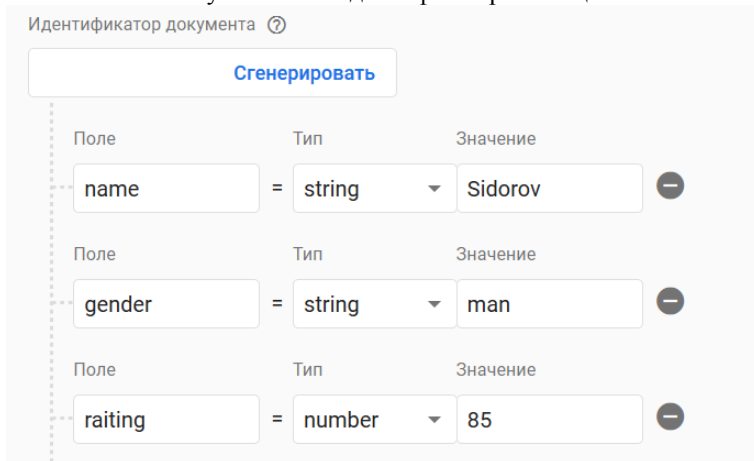
/groups/iUbACoUN7BrFnBC8G464

Идентификатор коллекции ?

students

Отмена Далее

Рисунок 4.62 – Ідентифікатор колекції



Идентификатор документа ?

Сгенерировать

Поле	Тип	Значение
name	string	Sidorov
gender	string	man
raiting	number	85

Рисунок 4.63 – Додавання елемента колекції

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

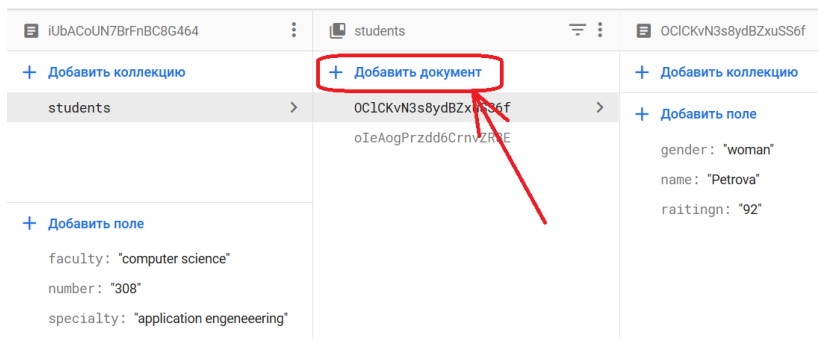


Рисунок 4.64 – Додавання декількох студентів

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

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



```
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).

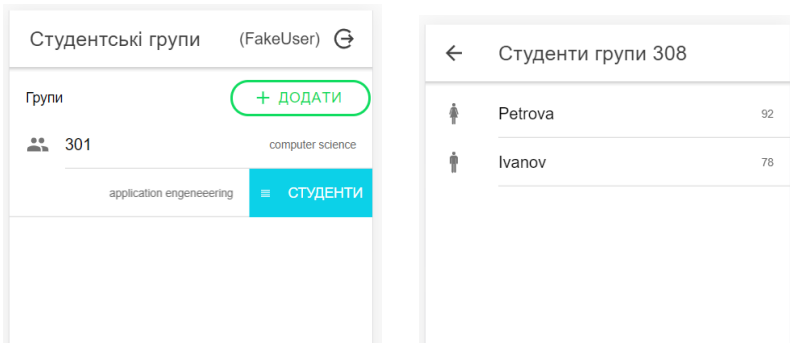


Рисунок 4.67 – Виведення списку студентів групи

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).

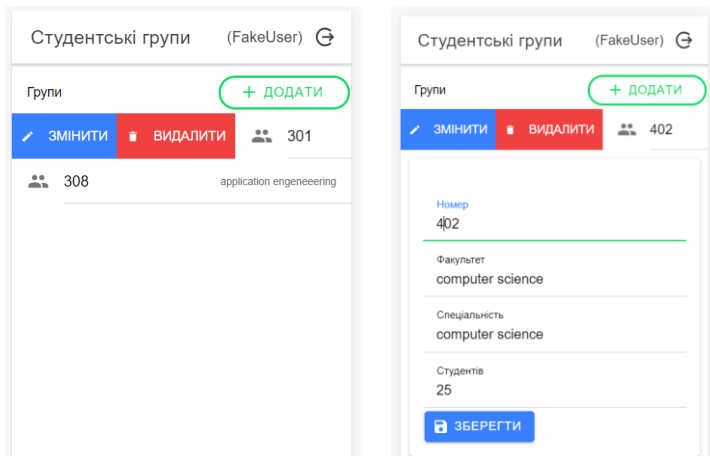


Рисунок 4.71 – Зміна номера групи

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

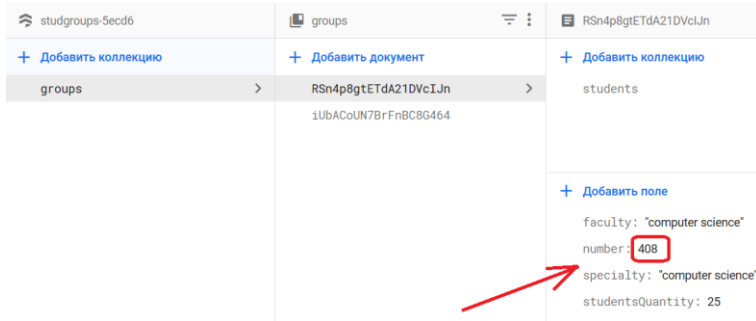


Рисунок 4.72 – Перевірка виконаних змін у консолі

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

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

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

Рисунок 4.74 – Зміни у `home.page.ts`

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

Скріншоти інтерфейсу для додавання та перегляду студентських груп.

Рисунок 4.75 – Додавання нової групи

groups	DUiHg2pdk3qNc16Z3MYc
+ Додати документ	+ Додати колекцію
DUiHg2pdk3qNc16Z3MYc	+ Додати поле
RSn4p8gtETdA21DVcIJn	faculty: "Комп.наук"
iUbACoUN7BrFnBC8G464	number: 201
	specialty: "комп'ютерні науки"
	studentsQuantity: 29

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  
45  
46  
delete(group) {  
    this.firestore.deleteGroup(group);  
}
```

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. Аутентифікації 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).

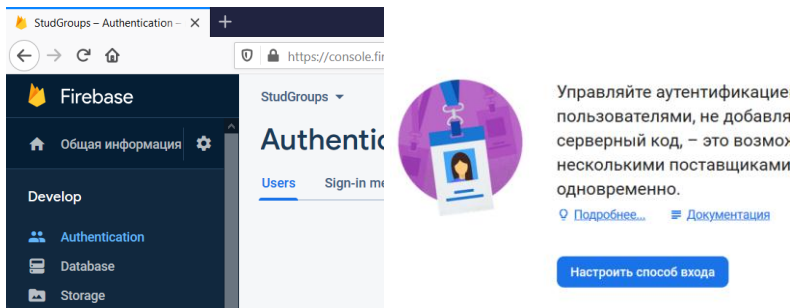


Рисунок 4.79 – Налаштування засобу входу у проєкті Firebase

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

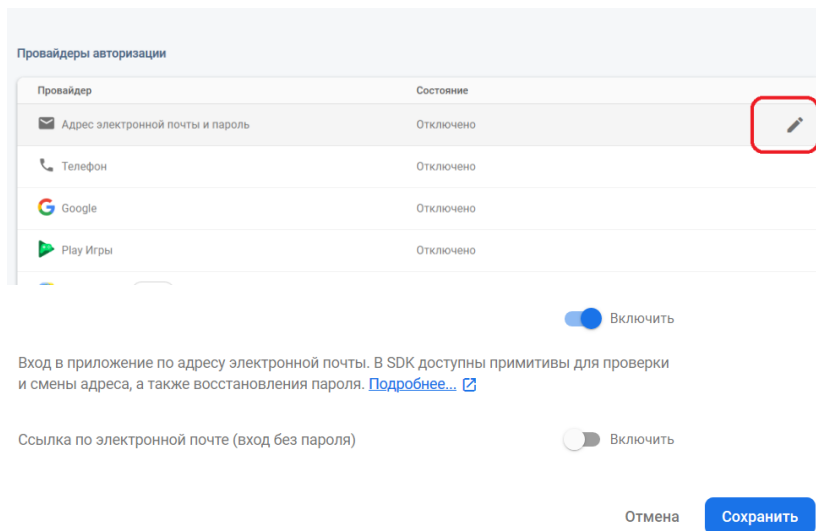


Рисунок 4.80 – Вибір провайдера авторизації

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

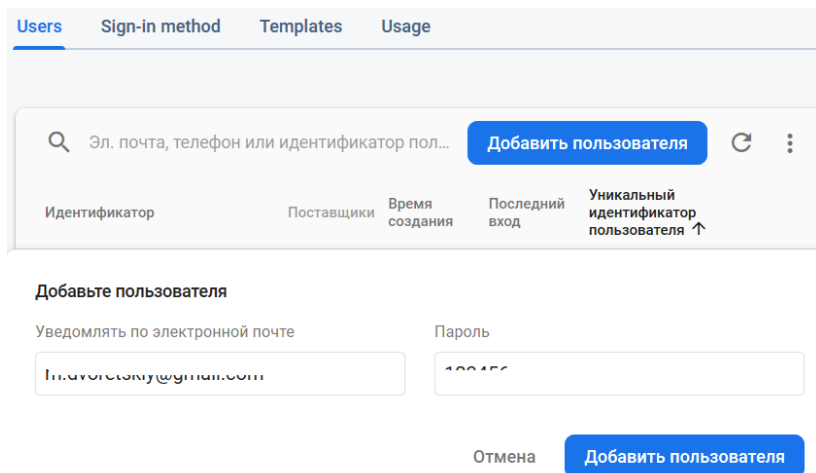


Рисунок 4.81 – Додавання нового користувача

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) {
18
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     }
42 }
```

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).

```
5 import {FireDataGetterService} from '../services/fire-data-getter.service';
6
15
16 constructor(
17     private router: Router,
18     private dataGetter: DataGetterService,
19     public alertController: AlertController,
20     private fireData: FireDataGetterService) {}
21
22 ngOnInit() {}
23
24 login() {
25     this.fireData.checkUser( user: {
26         username: this.userName,
27         passwd: this.passWord
28     }).then(
29         onfulfilled: res => {
30             this.fireData.setUser(this.userName);
31             this.router.navigate( commands: ['/home']);
32         },
33         onrejected: err => {
34             this.userNotExistAlert(err.message);
35         }
36     );
37 }
```

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).

```
4 import {FireDataGetterService} from '../services/fire-data-getter.service';
5
10 constructor(private dataGetter: DataGetterService,
11             private router: Router,
12             private fireData: FireDataGetterService) {}
13
14 canActivate(
15     next: ActivatedRouteSnapshot,
16     state: RouterStateSnapshot): boolean {
17
18     const isLoggenIn = this.fireData.getUser() !== '';
19     if (!isLoggenIn) {
20         this.router.navigateByUrl( url: '/login');
21     }
22     return isLoggenIn;
23 }
```

Рисунок 4.85 – Зміни guard-у

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

Розробка гібридних застосунків для цифрової трансформації бізнесу

```
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).

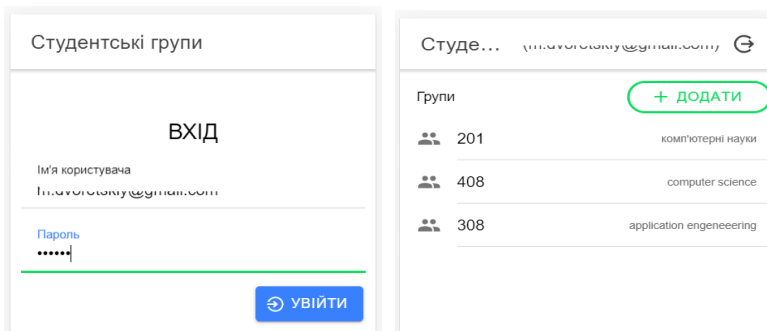


Рисунок 4.87 – Перевірка роботи аутентифікації після виконаних змін

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

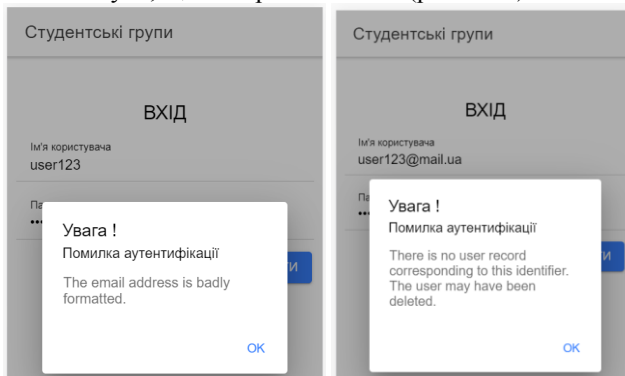


Рисунок 4.88 – Спроби невдалої авторизації

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.firestore.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: "iUbACoUM7BrFnBC8G464", specialty: "application engineeering", 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.

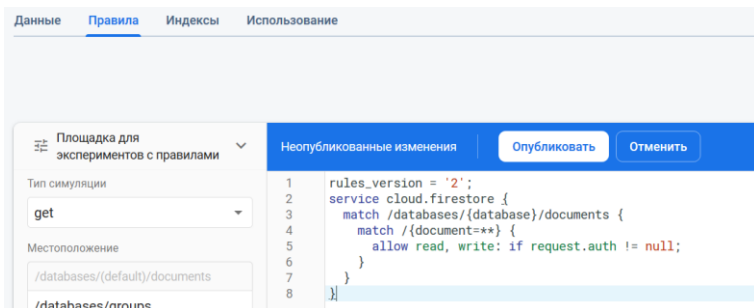


Рисунок 4.91 – Створення правила для контролю доступу до даних

Розробка гібридних застосунків для цифрової трансформації бізнесу

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

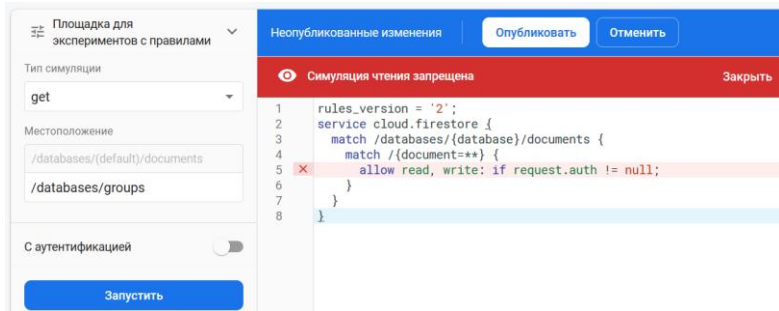


Рисунок 4.92 – Симуляція доступу до даних без аутентифікації

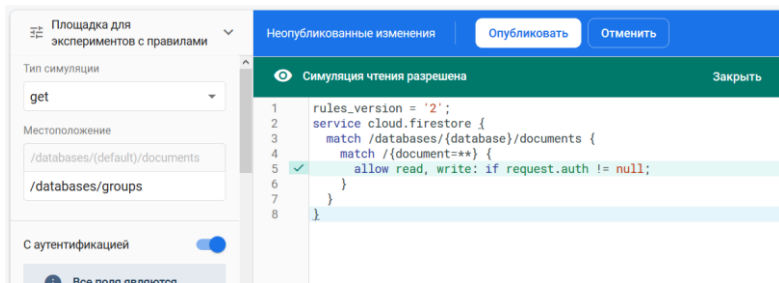


Рисунок 4.93 – Симуляція доступу до даних з аутентифікацією

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).

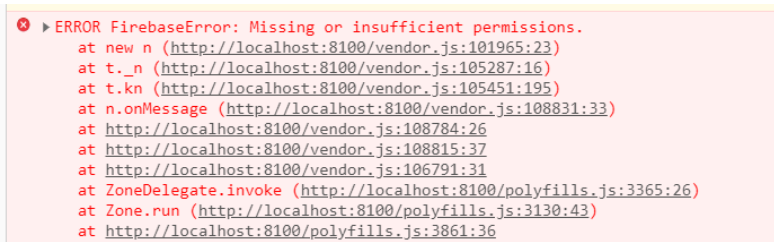


Рисунок 4.94 – Спроба звернення до даних без аутентифікації

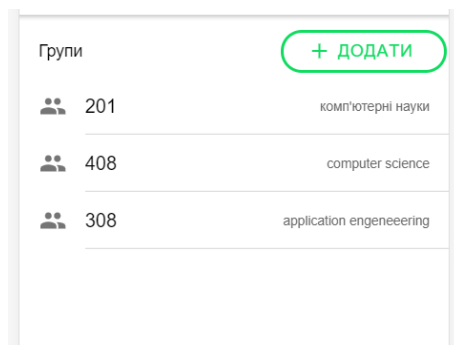


Рисунок 4.95 – Читання даних після виконання входу

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 іonic 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 іonic 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 іonic 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 проектом 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 порції)	Склад (Продукт, вага на порцію)
7	Автостоянка (Адреса, ПІБ директора)	Автомобілі (Держ. номер, марка, номер місця)

**Розробка гібридних застосунків
для цифрової трансформації бізнесу**

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 ЛИТЕРАТУРА

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 ПОДАТОК

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. Петrenchенко.
Комп'ютерна верстка, дизайн обкладинки Н. Кардаш.
Друк С. Волинець. Фальцювальні-палітурні роботи, О. Мішалкіна.*

*Підп. до друку 10.01.2022.
Формат 60х84¹/₁₆. 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.*