

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ  
СІКОРСЬКОГО**

Факультет інформатики та обчислювальної техніки Кафедра  
інформатики та програмної інженерії

Звіт по лабораторній роботі №1  
«Створення Angular-додатків “HelloApp” і «Shopping list». Прив'язка  
даних в Angular.»  
роботи з дисципліни: «Реактивне програмування»

Студент: Мешков Андрій Ігорович

Група: ІІІ-15

Дата захисту роботи: 27 «вересня» 2024

Викладач: доц. Полупан Юлія Вікторівна

Захищено з оцінкою: \_\_\_\_\_

Київ, 2024

## ЗМІСТ

|   |    |
|---|----|
| ЗМІСТ .....   | 2  |
| ЧАСТИНА 1: СТВОРЕННЯ ANGULAR-ДОДАТКІВ “HELLOAPP” І<br>«SHOPPING LIST» .....                               | 3  |
| А) ОПИС ОСНОВНИХ СТРУКТУРНИХ БЛОКІВ ANGULAR-ДОДАТКУ «HELLOAPP»:<br>МОДУЛІ, КОМПОНЕНТИ, ШАБЛОНИ. ....      | 4  |
| В) ОПИС ОСНОВНИХ СТРУКТУРНИХ БЛОКІВ ANGULAR-ДОДАТКУ «SHOPPING<br>LIST»: МОДУЛІ, КОМПОНЕНТИ, ШАБЛОНИ. .... | 7  |
| С) ОПИС ФАЙЛУ PACKAGE.JSON. ПРИЗНАЧЕННЯ, ОСНОВНІ ПАРАМЕТРИ. ....  | 11 |
| D) ОПИС ФАЙЛУ TSCONFIG.JSON. ПРИЗНАЧЕННЯ, ОСНОВНІ ПАРАМЕТРИ. ....   | 14 |
| Е) ОПИС ФАЙЛУ ANGULAR.JSON. ПРИЗНАЧЕННЯ, ОСНОВНІ ПАРАМЕТРИ.....   | 17 |
| ЧАСТИНА 2 ПРИВ'ЯЗКА ДАНИХ.....  | 20 |
| 1) ІНТЕРПОЛЯЦІЯ В ANGULAR: ОГЛЯД ПРИКЛАДИ ВИКОРИСТАННЯ.....   | 21 |
| 2) ПРИВ'ЯЗКА ВЛАСТИВОСТЕЙ ЕЛЕМЕНТІВ HTML: ОГЛЯД ПРИКЛАДИ<br>ВИКОРИСТАННЯ. ....                            | 23 |
| 3) ПРИВ'ЯЗКА ДО АТРИБУТУ: ОГЛЯД ПРИКЛАДИ ВИКОРИСТАННЯ. ....   | 25 |
| 4) ПРИВ'ЯЗКА ДО ПОДІЇ: ОГЛЯД ПРИКЛАДИ ВИКОРИСТАННЯ. ....  | 27 |
| 5) ДВОСТОРОННЯ ПРИВ'ЯЗКА: ОГЛЯД ПРИКЛАДИ ВИКОРИСТАННЯ.....  | 29 |
| 6) ПРИВ'ЯЗКА ДО КЛАСІВ CSS: ОГЛЯД ПРИКЛАДИ ВИКОРИСТАННЯ. ....   | 31 |
| 7) ПРИВ'ЯЗКА СТИЛІВ: ОГЛЯД, ПРИКЛАДИ ВИКОРИСТАННЯ .....   | 34 |
| ВИСНОВКИ.....   | 36 |
| СПИСОК ДЖЕРЕЛ .....   | 37 |

## ЧАСТИНА 1: СТВОРЕННЯ ANGULAR-ДОДАТКІВ “HELLOAPP” І «SHOPPING LIST»

**Мета:** навчитися встановлювати необхідне ПО для створення Angular-додатку. Навчитися створювати шаблон в компоненті Angular.

Завдання:

- 1) створити при допомозі текстового редактора простий Angular-додаток “HelloApp”;
- 2) при допомозі текстового редактора створити простий додаток «Shopping list»;
- 3) зробити звіт по роботі. Звіт повинен включати: титульний лист, зміст, основна частина, список використаних джерел. Звіт повинен бути один для Частини 1 та Частини 2 лабораторної роботи №1;
- 4) розгорнути Angular-додаток «Shopping list» на платформі Firebase.

а) Опис основних структурних блоків Angular-додатку «HelloApp»:  
модулі, компоненти, шаблони.

#### 1. app.component.ts

Цей файл визначає основний компонент додатку — `AppComponent`. Компонент є важливим будівельним блоком Angular, який визначає логіку, дані та вигляд частини інтерфейсу користувача. У даному файлі наведено наступні елементи:

- Імпорт `Component` з `@angular/core`: Імпортує декоратор `Component`, який використовується для визначення метаданих та поведінки компонента.

- Декоратор `@Component`:

```
@Component({
  selector: 'my-app',
  template: `<label>Введіть назву:</label>
    <input [(ngModel)]="name" placeholder="name">
    <h1>Ласкаво просимо {{name}}!</h1>`
})
```

Декоратор `@Component` додається перед класом `AppComponent` і приймає об'єкт з налаштуваннями:

- selector: 'my-app': Вказує, що цей компонент буде рендеритися у DOM (документі HTML) за допомогою тегу `<my-app>`.

- template: Описує HTML-шаблон компонента. У цьому шаблоні є:

- - Поле введення (`<input>`), зв'язане зі змінною `name` класу через двостороннє зв'язування даних (`[(ngModel)]`).
- - Заголовок (`<h1>`), який відображає вітання з використанням введеного імені.

- Клас `AppComponent`:

```
export class AppComponent {  
  name=' ';  
}
```

Описує логіку та дані компонента. У класі визначена одна змінна `name`, яка зберігає значення, введене користувачем. Це значення динамічно відображається у заголовок.

## 2. app.module.ts

Цей файл визначає головний модуль додатку — `AppModule`. Модуль — це контейнер, що об'єднує компоненти, сервіси та інші модулі, необхідні для роботи додатку. Опис файлу:

- Імпортуються основні модулі та компоненти:

- `NgModule`: Декоратор, що визначає модуль.
- `BrowserModule`: Модуль, необхідний для запуску додатку в браузері.
- `FormsModule`: Модуль, що підтримує роботу з формами та двостороннє зв'язування даних.
- `AppComponent`: Компонент, який визначений у файлі `app.component.ts`.

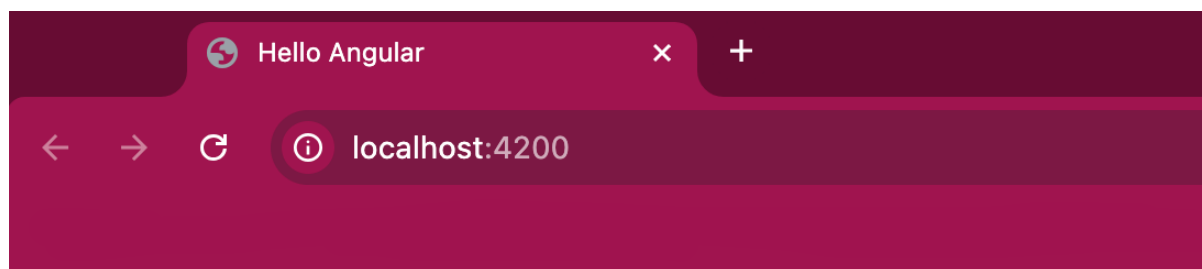
- Декоратор @NgModule:

```
@NgModule({  
  imports: [ BrowserModule, FormsModule ],  
  declarations: [ AppComponent ],  
  bootstrap: [ AppComponent ]  
})
```

Декоратор @NgModule приймає об'єкт конфігурації:

- `imports`: Містить список модулів, які використовуються в додатку. Тут підключаються `'BrowserModule'` та `'FormsModule'`.
- `declarations`: Визначає компоненти, що належать до цього модуля. У даному випадку, це компонент `'AppComponent'`.
- `bootstrap`: Вказує основний компонент для запуску додатку. В нашому випадку, це також `'AppComponent'`.

- Клас `'AppModule'` є контейнером для всієї конфігурації, що описана вище, та забезпечує завантаження основного компонента при запуску додатку.



Введіть назву:

# Ласкаво просимо Andrii Mieshkov!

b) Опис основних структурних блоків Angular-додатку «Shopping list»:  
модулі, компоненти, шаблони.

1. У додатку «Shopping list» визначений один модуль:

AppModule:

- Імпортує основні модулі Angular, такі як `BrowserModule` та `FormsModule`.
  - `BrowserModule`: Забезпечує функціонал для роботи додатку в браузері.
  - `FormsModule`: Додає підтримку роботи з формами та двостороннім зв'язуванням даних (`ngModel`).
- Включає визначення компонентів додатку в розділі `declarations`. У цьому випадку це компонент `AppComponent`.
- Вказує основний компонент для завантаження додатку через властивість `bootstrap`, що забезпечує ініціалізацію `AppComponent`.

2. Компоненти

AppComponent:

- Імпорт та клас `Item`:

```
class Item{
  purchase: string;
  done: boolean;
  price: number;
  constructor(purchase: string, price: number) {
    this.purchase = purchase;
    this.price = price;
    this.done = false;
  }
}
```

Цей клас визначає модель даних для кожного елементу списку покупок з трьома властивостями:

- `purchase`: назва товару.
- `price`: ціна товару.
- `done`: статус покупки (виконано чи ні).

Декоратор `@Component`:

```
@Component({
  selector: 'my-app',
  template: `<div class="page-header">
    <h1> Shopping list </h1>
  </div>
  <div class="panel">
    <div class="form-inline">
      <div class="form-group">
        <div class="col-md-8">
          <input class="form-control" [(ngModel)]="text" placeholder =
"Назва" />
        </div>
      </div>
      <div class="form-group">
        <div class="col-md-6">
          <input type="number" class="form-control" [(ngModel)]="price"
placeholder="Ціна" />
        </div>
      </div>
      <div class="form-group">
        <div class="col-md-offset-2 col-md-8">
          <button class="btn btn-default" (click)="addItem(text,
price)">Додати</button>
        </div>
      </div>
    </div>
    <table class="table table-striped">
      <thead>
        <tr>
          <th>Предмет</th>
          <th>Ціна</th>
          <th>Куплено</th>
        </tr>
      </thead>
      <tbody>
        <tr *ngFor="let item of items">
          <td>{{item.purchase}}</td>
          <td>{{item.price}}</td>
```



```

        <td><input type="checkbox" [(ngModel)]="item.done" /></td>
      </tr>
    </tbody>
  </table>
</div>`
})

```

- selector: 'my-app': Вказує, що компонент буде рендеритися у DOM за допомогою тегу '<my-app>`'.
- 'template': Містить HTML-шаблон компонента, який складається з форми додавання елементів та таблиці, де відображається список покупок.

Клас 'AppComponent':

```

export class AppComponent {
  text: string = "";
  price: number = 0;
  items: Item[] =
  [
    { purchase: "Хліб", done: false, price: 15.9 },
    { purchase: "Вершкове масло", done: false, price: 60 },
    { purchase: "Картопля", done: true, price: 22.6 },
    { purchase: "Сир", done: false, price: 310 }
  ];

  addItem(text: string, price: number): void {
    if(text==null || text.trim()==" || price==null)
      return;
    this.items.push(new Item(text, price));
  }
}

```

Цей клас містить:

- 'text' та 'price': змінні, що використовуються для зберігання введених користувачем даних для нового елементу списку.
- 'items': масив об'єктів 'Item', які представляють початковий список покупок.
- Метод 'addItem': Метод додає новий елемент до списку покупок, якщо введені дані коректні.


### 3. Шаблони

Форма для додавання нових елементів:

- Поля введення для назви (`[(ngModel)]="text"`) та ціни (`[(ngModel)]="price"`).
- Кнопка, яка викликає метод `addItem` для додавання нового елемента до списку.

Таблиця з елементами списку:

- Відображає всі елементи масиву `items` за допомогою директиви `*ngFor`.
- Включає назву товару, ціну та чекбокс для позначення статусу виконання (`done`).



Shopping list

| Предмет        | Ціна   | Куплено                             |
|----------------|--------|-------------------------------------|
| Хліб           | 15.9   | <input type="checkbox"/>            |
| Вершкове масло | 60     | <input type="checkbox"/>            |
| Картопля       | 22.6   | <input checked="" type="checkbox"/> |
| Сир            | 310    | <input type="checkbox"/>            |
| Майонез        | 123.33 | <input type="checkbox"/>            |

### с) Опис файлу package.json. Призначення, основні параметри.

Файл `package.json` є конфігураційним файлом для проектів, які використовують платформу Node.js, і обов'язковий для роботи з Angular-додатками. Він містить метадані проекту, залежності, команди для запуску та збірки додатку, а також інші налаштування. Розглянемо основні параметри на прикладі Angular-додатків «helloapp» та «purchaseapp».

#### Призначення `package.json`

- Опис проекту: Містить основну інформацію про проект, таку як назва, версія, автор тощо.
- Залежності: Визначає, які бібліотеки та пакети необхідні для роботи додатку.
- Скрипти: Вказує команди для запуску та збірки додатку.
- Конфігурація розробника: Включає налаштування для інструментів розробки.

#### Основні параметри

##### 1. Інформація про проект:

```
"name": "purchaseapp",  
"version": "1.0.0",  
"description": "First Angular 16 Project",  
"author": "Juliya Polupan",
```

- `name`: Ім'я проекту (`helloapp` або `purchaseapp`).
- `version`: Версія проекту, зазвичай у форматі `MAJOR.MINOR.PATCH`. У даному випадку `1.0.0`.
- `description`: Опис проекту, який надає базову інформацію (наприклад, «First Angular 16 Project»).
- `author`: Автор проекту (`Juliya Polupan`).

## 2. Скрипти:

```
"scripts": {  
  "ng": "ng",  
  "start": "ng serve",  
  "build": "ng build"  
},
```

Розділ `scripts` визначає команди для автоматизації різних завдань:

- `ng`: Вказує на виконання Angular CLI (інструмент командного рядка).
- `start`: Запускає додаток у режимі розробки за допомогою команди `ng serve`.
- `build`: Збирає додаток для продакшн середовища за допомогою `ng build`.

## 3. Залежності (dependencies):

```
"dependencies": {  
  "@angular/common": "~16.0.0",  
  "@angular/compiler": "~16.0.0",  
  "@angular/core": "~16.0.0",  
  "@angular/forms": "~16.0.0",  
  "@angular/platform-browser": "~16.0.0",  
  "@angular/platform-browser-dynamic": "~16.0.0",  
  "@angular/router": "~16.0.0",  
  "rxjs": "7.8.0",  
  "zone.js": "~0.13.0"  
},
```

Містить бібліотеки та фреймворки, необхідні для роботи додатку.

- `@angular/\*`: Основні модулі Angular, такі як `core`, `forms`, `router`, що забезпечують базовий функціонал додатку.
- `rxjs`: Реактивні розширення для роботи з асинхронними даними.
- `zone.js`: Використовується Angular для виявлення та обробки змін у моделі даних.

#### 4. Залежності для розробника (devDependencies):

```
"devDependencies": {
  "@angular-devkit/build-angular": "~16.0.0",
  "@angular/cli": "~16.0.0",
  "@angular/compiler-cli": "~16.0.0",
  "firebase-tools": "^13.19.0",
  "typescript": "~5.0.4"
}
```

- Містить інструменти, необхідні для розробки, тестування та збірки додатку.

- `@angular-devkit/build-angular`: Пакет для збору Angular-додатків.
- `@angular/cli`: Angular CLI для управління проектом.
- `@angular/compiler-cli`: Angular Compiler CLI для компіляції Angular-додатків.
- `typescript`: Підтримка TypeScript, мови програмування, яка використовується в Angular.
- `firebase-tools`: Інструменти для роботи з Firebase, зокрема для розгортання додатку на платформі Firebase.

#### d) Опис файлу tsconfig.json. Призначення, основні параметри.

Файл `tsconfig.json` використовується для налаштування компіляції TypeScript-проектів. Він визначає параметри компілятора, які файли слід компілювати, і яким чином. Для Angular-додатків цей файл є критично важливим, оскільки дозволяє налаштувати поведінку компілятора TypeScript, забезпечуючи правильну компіляцію та сумісність коду.

#### Призначення `tsconfig.json`

- Налаштування параметрів компілятора TypeScript: Визначає, як саме TypeScript компілює код, які файли та папки включати або виключати, які версії JavaScript підтримувати тощо.
- Вказівка на файли та каталоги для компіляції: Дозволяє задати конкретні файли або директорії, які повинні бути включені або виключені з процесу компіляції.
- Налаштування бібліотек та модулів: Вказує, які бібліотеки та модулі повинні бути доступні для компілятора.

#### Основні параметри `tsconfig.json`

##### 1. `compileOnSave`:

```
"compileOnSave": false,
```

- Вказує, чи повинні файли автоматично перекомпілюватися при їх збереженні у редакторі.
- `false`: компіляція не виконується автоматично при збереженні.

##### 2. `compilerOptions`:

Це ключова секція, яка визначає параметри компілятора TypeScript.

```
"compilerOptions": {  
  "baseUrl": "./",
```

```

    "sourceMap": true,
    "declaration": false,
    "downlevelIteration": true,
    "experimentalDecorators": true,
    "module": "esnext",
    "moduleResolution": "node",
    "target": "es2022",
    "typeRoots": [
      "node_modules/@types"
    ],
    "lib": [
      "es2022",
      "dom"
    ]
  },

```

- ``baseUrl``: Базовий шлях для модульного резолювання відносних шляхів (``.`` — поточний каталог).
- ``sourceMap``: Створює файли ``.map`` для відстеження вихідного TypeScript-коду у скомпільованому JavaScript, що полегшує відлагодження.
- ``declaration``: Не створювати ``.d.ts`` файли (деклараційні файли типів) при компіляції.
- ``downlevelIteration``: Дозволяє використовувати ітераційні конструкції ES2015, такі як ``for..of`` з пониженням версії JavaScript.
- ``experimentalDecorators``: Дозволяє використовувати декоратори, які часто використовуються в Angular для таких конструкцій, як компоненти та інжектвані сервіси.
- ``module``: Використовується для підтримки новітнього формату модулів JavaScript (ESNext).
- ``moduleResolution``: Використовує алгоритм Node.js для пошуку модулів, що дозволяє правильно резолювати модулі.
- ``target``: Вказує версію ECMAScript, в яку компілюватиметься TypeScript-код. ``es2022`` — нова версія JavaScript, що включає останні функції мови.

- ``typeRoots``: Вказує директорії, де компілятор шукатиме типи. У даному випадку, `"node_modules/@types"` включає типи для всіх залежностей, встановлених через npm.
- ``lib``: Перелік стандартних бібліотек, які повинні бути доступні компілятору:
  - ``es2022``: Включає всі функції ECMAScript 2022.
  - ``dom``: Додає типи для об'єктів DOM (Document Object Model).

### 3. ``files``:

```
"files": [  
  "src/main.ts",  
  "src/polyfills.ts"  
],
```

- Вказує конкретні файли, які повинні бути скомпільовані.
- У цьому випадку, ``src/main.ts`` — головний файл додатку, а ``src/polyfills.ts`` — файл для підтримки різних браузерів.

### 4. ``include``:

```
"include": [  
  "src/**/*.d.ts"  
]
```

- Включає всі ``.d.ts`` файли в каталозі ``src``, тобто деклараційні файли типів, які можуть бути використані для додаткової типізації.



е) Опис файлу `angular.json`. Призначення, основні параметри.

Файл ``angular.json`` є конфігураційним файлом Angular-проекту, який використовується Angular CLI для управління процесом збірки, налаштуваннями та різними середовищами. Цей файл дозволяє налаштувати проектні параметри, такі як шляхи до файлів, цільові середовища та параметри збірки для кожного проекту в репозиторії.

Призначення ``angular.json``:

- Керування конфігурацією проекту: Дозволяє налаштовувати шляхи до основних файлів проекту, директорій збірки, цільових середовищ і параметрів збірки.
- Управління проектами в монорепозіторії: Підтримує кілька проектів в одній кодовій базі, що дозволяє керувати налаштуваннями для кожного проекту окремо.
- Налаштування архітектурних цілей (builders): Визначає різні завдання для збірки, тестування, розгортання додатку та інших операцій.

Основні параметри ``angular.json``

1. ``version``:

```
"version": 1,
```

- Визначає версію конфігураційного файлу. Зазвичай значення 1 використовується для файлів Angular CLI.

2. ``projects``:

```
"projects": {  
  "purchaseapp": {  
    "projectType": "application",  
    "root": "",
```

```

    "sourceRoot": "src",
    "architect": {
      "build": {
        "builder": "@angular-devkit/build-angular:browser",
        "options": {
          "outputPath": "dist/purchaseapp",
          "index": "src/index.html",
          "main": "src/main.ts",
          "polyfills": "src/polyfills.ts",
          "tsConfig": "tsconfig.json",
          "aot": true
        }
      },
      "serve": {
        "builder": "@angular-devkit/build-angular:dev-server",
        "options": {
          "browserTarget": "purchaseapp:build"
        }
      }
    }
  },
},

```

- Містить конфігурацію для одного або кількох проектів у монорепозіторії. У цьому прикладі описується конфігурація для проекту `purchaseapp`.
- `projectType`: Визначає тип проекту, наприклад, `application` для додатків або `library` для бібліотек.
- `root`: Коренева директорія проекту (у даному випадку — порожнє значення, що означає корінь репозиторію).
- `sourceRoot`: Вказує директорію, в якій зберігаються вихідні файли проекту (у даному прикладі `src`).
- `architect`: Цей параметр містить налаштування для різних завдань, які можуть бути виконані Angular CLI, таких як збірка (`build`), тестування (`test`), запуск локального сервера (`serve`) тощо.

- `build`:

- `'builder'`: Визначає, який збирач (builder) використовувати для створення проекту. У цьому випадку `'@angular-devkit/build-angular:browser'` використовується для збірки браузерного додатку.
- `'options'`: Основні опції для збірки:
  - `'outputPath'`: Директорія, куди будуть збережені зібрані файли (у даному випадку `'dist/purchaseapp'`).
  - `'index'`: Шлях до HTML-файлу, який використовується як вхідна точка (`'src/index.html'`).
  - `'main'`: Основний файл додатку, який компілюється першим (`'src/main.ts'`).
  - `'polyfills'`: Шлях до файлу з полімерами для підтримки різних браузерів (`'src/polyfills.ts'`).
  - `'tsConfig'`: Шлях до файлу налаштувань TypeScript (`'tsconfig.json'`).
  - `'aot'`: Увімкнення Ahead-of-Time компіляції для оптимізації продуктивності (`'true'`).

- `'serve'`:

- Використовується для налаштування локального сервера для розробки.
- `'browserTarget'`: Вказує ціль для завдання `'serve'`, яка відповідає збірці проекту `'purchaseapp'` з параметрами збірки `'build'`.

### 3. `'defaultProject'`:

```
"defaultProject": "purchaseapp"
```

- Визначає проект, який буде використовуватись за замовчуванням при виконанні команд Angular CLI, якщо не вказано інший проект. У даному випадку, це `'purchaseapp'` або `'helloapp'`.

## ЧАСТИНА 2 ПРИВ'ЯЗКА ДАНИХ.

**Тема:** Навчитися працювати з прив'язкою даних.

**Завдання:** Створити два Angular-додатки під назвою Binding1 та Binding2, як показано в частині 1.

I) Для Angular-додатку Binding1 виконати вправи 1-5;

II) Для Angular-додатку Binding2 виконати вправи 6-7;

III) Зробити звіт по роботі (по Angular-додатках Binding1 та Binding2).

Звіт повинен бути один для Частини 1 та Частини 2 лабораторної роботи №1 і включати: титульний лист, зміст, основну частину, список використаних джерел;

IV) Angular-додаток Binding1 розвернути на платформі Firebase.

## 1) Інтерполяція в Angular: огляд приклади використання.

Інтерполяція в Angular — це механізм, що дозволяє вставляти значення з компонентів у HTML-шаблони. Вона дозволяє відображати дані, збережені в змінних компонентів, безпосередньо в шаблоні, використовуючи подвійні фігурні дужки (`{{ }}`).

Інтерполяція дозволяє відображати значення змінних, результат виконання виразів або виклики функцій прямо в HTML. В Angular ви можете використовувати інтерполяцію для відображення тексту, чисел, а також результатів обчислень.

Синтаксис: `{{expression}}`

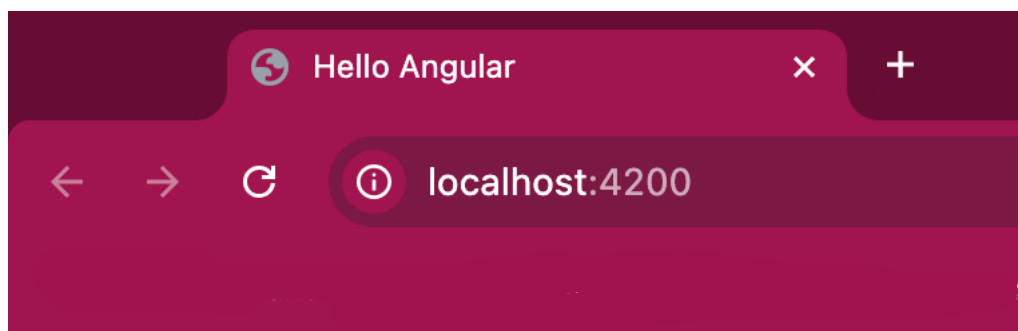
- `expression` — це будь-який вираз JavaScript, який може включати змінні, функції або обчислення.

### Приклад використання інтерполяції

Розглянемо приклад, в якому показано, як інтерполяція використовується в Angular-компоненті:

```
@Component({
  selector: 'my-app',
  template: `
    <p>Ім'я: {{name}}</p>
    <p>Вік: {{age}}</p>
  `
})
export class AppComponent {
  name = "Andrii Mieshkov";
  age = 25;
}
```

- Значення змінних `name` та `age` відображаються в абзацах. При зміні цих змінних значення в шаблоні автоматично оновляться.



Ім'я: Andrii Mieshkov

Вік: 25

## 2) Прив'язка властивостей елементів HTML: огляд приклади використання.

Прив'язка властивостей в Angular — це механізм, який дозволяє зв'язувати дані з компоненту з властивостями елементів HTML у шаблоні. Це дозволяє динамічно змінювати значення властивостей елементів у відповідь на зміни в даних компоненту, без необхідності ручного оновлення DOM.

Прив'язка властивостей в Angular може бути реалізована кількома способами, але найпоширеніші — це прив'язка до атрибутів, прив'язка до класів і стилів. Давайте розглянемо ці методи детальніше.

Прив'язка властивостей реалізується за допомогою квадратних дужок ('[]'). Синтаксис виглядає так:

```
<element [property]="expression"></element>
```

- `element` — HTML-елемент, до якого застосовується прив'язка.
- `property` — властивість елемента, яку потрібно прив'язати.
- `expression` — вираз, який буде оцінено для встановлення значення властивості.

Розглянемо приклад прив'язки властивостей в Angular.

```
@Component({
  selector: 'my-app',
  template: `
    <input type="text" [value]="status" />
    <input type="text" [value]="statusValue" />
  `
})
export class AppComponent {
  status = "Status";
  statusValue = "Student";
}
```

Значення ``status`` і ``statusValue`` відображаються у текстових полях. Це означає, що при ініціалізації компоненту в текстових полях буде видно "Status" та "Student" відповідно.

|        |         |
|--------|---------|
| Status | Student |
|--------|---------|



### 3) Прив'язка до атрибуту: огляд приклади використання.

Прив'язка до атрибуту в Angular дозволяє динамічно змінювати значення атрибутів HTML-елементів відповідно до значень змінних в компоненті. Цей механізм надає можливість гнучко змінювати структуру та властивості інтерфейсу на основі стану даних, що робить інтерфейс більш інтерактивним.

Для прив'язки до атрибуту в Angular використовується квадратні дужки ('[]') разом із префіксом 'attr.'. Синтаксис виглядає так:

```
<element [attr.attributeName]="expression"></element>
```

- 'element' — HTML-елемент, до якого застосовується прив'язка.
- 'attributeName' — ім'я атрибута, який потрібно прив'язати.
- 'expression' — вираз, який буде оцінений для встановлення значення атрибута.

Розглянемо приклад, у якому ми використовуємо прив'язку до атрибута для зміни значення 'colspan' в елементі '<td>' таблиці.

```
@Component({
  selector: 'my-app',
  template: `
    <table border="1">
      <tr><td [attr.colspan]="colspan">One-Two</td></tr>
      <tr><td>Three</td><td>Four</td></tr>
      <tr><td>Five</td><td>Six</td></tr>
    </table>
  `,
})

export class AppComponent {
  colspan = 2;
}
```

- У цьому прикладі таблиця складається з трьох рядків, перший з яких містить одне поле `<td>` з атрибутом `colspan`, що визначає, скільки стовпців цей `<td>` займає.

- Атрибут `colspan` прив'язується до змінної `colspan` у компоненті. Спочатку `colspan` дорівнює 2, тому елемент `<td>` охоплює два стовпці.

|         |      |
|---------|------|
| One-Two |      |
| Three   | Four |
| Five    | Six  |

#### 4) Прив'язка до події: огляд приклади використання.

Прив'язка до події в Angular дозволяє реагувати на дії користувача, такі як натискання кнопки, введення тексту, зміна вибору тощо. Це досягається через синтаксис прив'язки подій, який надає можливість викликати методи компонента при виникненні певних подій.

Для прив'язки до події в Angular використовується круглі дужки (`()`) навколо назви події. Синтаксис виглядає так:

```
<element (event)="method()"></element>
```

- `element` — HTML-елемент, на якому ви слухаєте подію.
- `event` — назва події (наприклад, `click`, `keyup`, `change`).
- `method()` — метод компонента, який буде викликаний при виникненні події.

Розглянемо приклад, де ми будемо відстежувати кількість кліків на кнопку і оновлювати цю інформацію в інтерфейсі.

```
@Component({
  selector: 'my-app',
  template: `
    <p>Кількість кліків {{count}}</p>
    <button (click)="increase()">Click</button>
  `
})

export class AppComponent {
  count: number = 0;
  increase(): void {
    this.count++;
  }
}
```

1. Відображення кількості кліків:

- У шаблоні компонента є елемент `<p>`, який відображає кількість кліків. Змінна `count` ініціалізується нулем:

- При першому рендерингу значення `count` дорівнює 0, тому в інтерфейсі з'явиться "Кількість кліків 0".

## 2. Кнопка для збільшення лічильника:

- Додаємо кнопку, на якій прив'язано подію `click` до методу `increase()`:

- Коли користувач натискає на кнопку, викликається метод `increase()`, що збільшує значення `count` на одиницю.

## 3. Оновлення інтерфейсу:

- Angular автоматично відстежує зміни у значенні `count` і перерисовує DOM, щоб відобразити нове значення. Таким чином, при кожному натисканні на кнопку значення кількості кліків оновлюється на екрані.

Кількість кліків 0

Click

## 5) Двостороння прив'язка: огляд приклади використання.

Двостороння прив'язка в Angular є потужним механізмом, що дозволяє синхронізувати дані між компонентами та елементами інтерфейсу. Це означає, що зміни, внесені в дані компоненту, автоматично відображаються в інтерфейсі, і навпаки — зміни в інтерфейсі негайно оновлюють дані компоненту.

Для реалізації двосторонньої прив'язки в Angular використовується синтаксис `[(ngModel)]`. Це поєднання прив'язки до властивості та прив'язки до події. Синтаксис виглядає так:

```
<input [(ngModel)]="propertyName" />
```

- `'propertyName'` — це змінна в компоненті, яка буде синхронізована з елементом інтерфейсу.

Розглянемо приклад, де ми використовуємо двосторонню прив'язку для введення імені користувача та відображення його на екрані.

```
@Component({
  selector: 'my-app',
  template: `
    <p>Привіт {{name}}</p>
    <input type="text" [(ngModel)]="name" /> <br><br>
    <input type="text" [(ngModel)]="name" />
  `
})

export class AppComponent {
  name = "Andrii Mieshkov";
}
```

### 1. Відображення привітання:

- У шаблоні компонента є параграф `<p>`, який відображає привітання з іменем користувача:

- При першому рендерингу значення ``name`` дорівнює "Andrii Mieshkov", тому в інтерфейсі відобразиться "Привіт Andrii Mieshkov".

## 2. Введення імені:

- Два текстових поля `<input>` мають прив'язку до змінної ``name``:
- Це означає, що будь-яке введення значення в одному з полів автоматично синхронізується з змінною ``name``, і відповідно, значення ``name`` буде відображатися у параграфі.

## 3. Динамічні зміни:

- Якщо користувач змінює текст у будь-якому з полів, значення ``name`` буде оновлене, а Angular автоматично оновить вміст параграфа, відобразивши нове ім'я.

Привіт Andrii Mieshkov

## 6) Прив'язка до класів CSS: огляд приклади використання.

Прив'язка до класів CSS в Angular дозволяє динамічно додавати або видаляти CSS-класи з елементів на основі значень змінних у компоненті. Це зручно для зміни стилю елементів в залежності від стану програми, що робить інтерфейс більш інтерактивним та адаптивним.

Для прив'язки до класів CSS в Angular використовується квадратні дужки `[]` навколо атрибуту `class`. Синтаксис виглядає так:

```
<div [class.className]="condition"></div>
```

- `className` — назва класу, який потрібно додати або видалити.

- `condition` — умова, що визначає, чи клас буде застосовано (якщо `true`, клас буде додано; якщо `false`, клас буде видалено).

Розглянемо приклад, де ми використовуємо прив'язку до класів CSS для зміни кольору фону блоків залежно від значення перемінної `isRed`.

```
@Component({
  selector: 'my-app',
  template: `
    <div [class.isredbox]="isRed"></div>
    <div [class.isredbox]="!isRed"></div>
    <input type="checkbox" [(ngModel)]="isRed" /><br><br>
    <div [class]="blue"></div><br><br>
  `,
  styles: [`
    div {width:50px; height:50px; border:1px solid #ccc}
    .isredbox{background-color:red;}
    .isbluebox{background-color:blue;}
  `]
})

export class AppComponent {
  isRed = false;
  blue = "isbluebox";
}
```

### 1. Динамічна зміна кольору фону:

- Перший `<div>` використовує прив'язку до класу `isredbox`:

- Якщо `isRed` дорівнює `true`, фон блоку буде червоним; якщо `false` — клас не буде застосовано, і фон залишиться прозорим.

## 2. Другий блок:

- Другий `<div>` навпаки використовує прив'язку до класу `isredbox` з умовою `!isRed`:

- Цей блок буде червоним, коли `isRed` буде `false`, і прозорим, коли `isRed` — `true`.

## 3. Перемикач:

- Інпут типу `checkbox` дозволяє користувачеві змінювати значення `isRed`:

- Коли користувач відмічає або знімає позначку, значення `isRed` змінюється, а Angular автоматично оновлює стилі обох блоків.

## 4. Третій блок:

- Третій блок використовує прив'язку до класу, вказаного у змінній `blue`:

- У цьому випадку, блок буде синім завдяки класу `isbluebox`, що визначений у стилях.





## 7) Прив'язка стилів: огляд, приклади використання

Прив'язка стилів в Angular дозволяє динамічно змінювати CSS-стилі елементів на основі значень змінних у компоненті. Це забезпечує високу гнучкість в управлінні виглядом елементів інтерфейсу, дозволяючи створювати адаптивні та інтерактивні додатки.

Для прив'язки стилів в Angular використовується квадратні дужки `[]` навколо атрибуту `style`. Синтаксис виглядає так:

```
<div [style.property]="expression"></div>
```

- `property` — назва CSS-властивості, яку потрібно змінити.

- `expression` — вираз, що визначає значення, яке буде застосоване до CSS-властивості.

Розглянемо приклад, де ми використовуємо прив'язку стилів для зміни кольору фону блоків залежно від значення перемінної `isyellow`.

```
@Component({
  selector: 'my-app',
  template: `

    <div [style.backgroundColor]="isyellow? 'yellow' : 'blue'"></div>
    <div [style.background-color]="!isyellow ? 'yellow' : 'blue'"></div>
    <input type="checkbox" [(ngModel)]="isyellow" />

  `,
  styles: [`
    div {width:50px; height:50px; border:1px solid #ccc}
  `]
})

export class AppComponent {
  isyellow=false;
}
```

### 1. Динамічна зміна кольору фону:

- Перший `<div>` використовує прив'язку стилю для зміни фону:

- Якщо `isyellow` дорівнює `true`, фон буде жовтим; якщо `false`, фон буде синім.

2. Другий блок:

- Другий `<div>` використовує прив'язку стилю з умовою `!isyellow`:

- Цей блок буде жовтим, коли `isyellow` дорівнює `false`, і синім, коли `true`.

3. Перемикач:

- Інпут типу `checkbox` дозволяє користувачеві змінювати значення `isyellow`:

- Коли користувач відмічає або знімає позначку, значення `isyellow` змінюється, а Angular автоматично оновлює стилі обох блоків.



## ВИСНОВКИ

Під час виконання лабораторної роботи було виконано всі поставлені завдання. У Частині 1: створено простий Angular-додаток “HelloApp”; створено простий додаток «Shopping list»; розгорнуто Angular-додаток «Shopping list» на платформі Firebase за посиланням <https://mieshkovip15laba1-1.web.app/>.

У Частині 2: створено два Angular-додатки під назвою Binding1 та Binding2, для Angular-додатку Binding1 виконано вправи 1-5; для Angular-додатку Binding2 виконано вправи 6-7; Angular-додаток Binding1 розвернуто на платформі Firebase за посиланням <https://mieshkovip15laba1-2.web.app/>.

## СПИСОК ДЖЕРЕЛ

1. Документація Angular. URL: <https://v17.angular.io/docs>
2. Документація Nodejs. URL: <https://nodejs.org/docs/latest/api/>
3. Документація Firebase. URL: <https://firebase.google.com/docs?hl=en>