Лабораторна робота №1. Створення Angular-додатків “HelloApp” і «Shopping list». Прив'язка даних в Angular.

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.**

1) Angular представляє фреймворк від компанії Google для створення клієнтських програм. Насамперед він орієнтований на розробку SPA-рішень (Single Page Application), тобто односторінкових додатків. 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 і так далі.

Однією з ключових особливостей Angular є те, що він використовує мову програмування TypeScript. Тому перед початком роботи рекомендується ознайомитись з основами цієї мови.

Але за бажанням можно писати програми на Angular за допомогою таких мов як Dart або JavaScript. Однак TypeScript є основною мовою для Angular.

Остання версія Angular – Angular 16 вийшла у травні 2023 року. Офіційний репозиторій фреймворку на гітхабі: https://github.com/angular/angular. Там можна знайти самі вихідні файли, а також деяку додаткову інформацію.

Активно підтримувані версії

У наведеній нижче таблиці наведено статус версій Angular, які підтримуються на момент написання навчального підручника (січень 2024):

Версія	Статус	Реліз	Кінець активного періоду	Кінець LTS
^17.0.0	Active	2023-11-08	2024-05-08	2025-05-15
^16.0.0	LTS	2023-05-03	2023-11-08	2024-11-08
^15.0.0	LTS	2022-11-18	2023-05-03	2024-05-18

Версії Angular від 2 до 14 більше не підтримуються.

Активно підтримувані версії

Наведена нижче таблиця охоплює версії Angular з активною підтримкою.

Angular	Node.js	TypeScript	RxJS
17.0.x	^18.13.0 || ^20.9.0	>=4.9.3 <5.3.0	^6.5.3 || ^7.4.0
16.1.x || 16.2.x	^16.14.0 || ^18.10.0	>=4.9.3 <5.2.0	^6.5.3 || ^7.4.0
16.0.x	^16.14.0 || ^18.10.0	>=4.9.3 <5.1.0	^6.5.3 || ^7.4.0
15.1.x || 15.2.x	^14.20.0 || ^16.13.0 || ^18.10.0	>=4.8.2 <5.0.0	^6.5.3 || ^7.4.0
15.0.x	^14.20.0 || ^16.13.0 || ^18.10.0	~4.8.2	^6.5.3 || ^7.4.0

Початок роботи з Angular

Для роботи з Angular необхідно встановити сервер Node.js та пакетний менеджер npm, якщо вони відсутні на робочій машині. При цьому, особливого знання для роботи з NodeJS і npm не потрібно. Для інсталяції можна використовувати програму інсталяції node.js. Разом із сервером вона також встановить і npm.

Слід враховувати, що Angular підтримує ті версії node.js, які зараз перебувають у статусі "Active LTS" або "Maintenance LTS". Зараз актуальними вважаються версії 16 і 18. Тому, якщо Node.js вже раніше був встановлений, але має більш стару або навпаки нову, але ще не підтримувану версію, то краще його оновити. Перевірити сумісність версій node.js (а також typescript та бібліотеки RxJS) для певних версій Angular можна за адресою: https://angular.io/guide/versions.

Те саме стосується і npm. Якщо версія дуже стара або, навпаки, одна з останніх, то Angular може її не підтримувати. При роботі з Angular краще покладатися на ту версію npm, яка встановлюється разом із LTS-версією Node.js.

Після встановлення необхідних інструментів створимо найпростіший додаток. Для цього визначимо на жорсткому диску папку програми. Хай вона буде називатися helloapp.

Установка Angular CLI

Для компіляції програми будемо використовувати інфраструктуру Angular CLI. Angular CLI спрощує створення програми, її компіляцію. Angular CLI поширюється як

пакет npm, тому для його використання необхідно його спочатку встановити. Для встановлення Angular CLI відкриємо консоль/командний рядок і виконаємо в ній наступну команду:

```
npm install -g @angular/cli
```

Ця команда встановить пакет @angular/cli як глобальний модуль, у подальшому під час створення нових проектів Angular його не потрібно буде встановлювати заново.

Ту ж команду можна використовувати для оновлення Angluar CLI під час виходу нової версії фреймворку. Перевірити версію CLI можна у командному рядку/консолі за допомогою команди:

```
ng version
```

При роботі на Windows і виконанні команд в PowerShell замість командного рядка варто враховувати, що за замовчуванням виконання скриптів PowerShell відключено. Щоб дозволити виконання скриптів PowerShell (що потрібно для npm), потрібно виконати таку команду:

```
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
```

Встановлення пакетів. Package.json

У папці проекту створимо новий файл package.json із таким вмістом:

```
{
    "name": "helloapp",
    "version": "1.0.0",
    "description": "First Angular 16 Project",
    "author": "Juliy Polupan",
    "scripts": {
          "ng": "ng",
          "start": "ng serve",
          "build": "ng build"
    },
    "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"
    },
    "devDependencies": {
```

```
        "@angular-devkit/build-angular": "~16.0.0",
        "@angular/cli": "~16.0.0",
        "@angular/compiler-cli": "~16.0.0",
        "typescript": "~5.0.4"
    }
}
```

Цей файл встановлює пакети та залежності, які будуть використовуватися проектом. У секції dependencies в основному визначаються пакети angular, які необхідні додатку для роботи. У секції devDependencies прописані лише пакети, які використовуватимуться для розробки. Зокрема, це пакети для роботи з мовою typescript (оскільки ми писатимемо код програми мовою TypeScript), а також пакети, необхідні для компіляції програми за допомогою інфраструктури Angular CLI.

У секції "scripts" описані команди, що використовуються в проекті. Зокрема, команда ng serve запускає простий веб-сервер для тестування програми та її запуску. А команда ng build компілює програму.

Потім відкриємо командний рядок (термінал) і перейдемо до папки проекту за допомогою команди cd:

```
C:\WINDOWS\system32>cd C:\angular\helloapp
```

І потім виконаємо команду npm install, яка встановить усі необхідні модулі:

```
C:\angular\helloapp>npm install

npm WARN deprecated @npmcli/move-file@2.0.1: This functionality has been moved to @npmcli/fs

added 799 packages, and audited 800 packages in 43s

81 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\angular\helloapp>
```

Після виконання цієї команди в папці проекту повинна з'явитися підпапка node_modules, яка містить всі залежності та пакети, що використовуються.

Визначення програми

Потім створимо в папці проекту підпапку, яку назвемо src - вона міститиме всі вихідні файли. І далі в папці src створимо підкаталог app.

Створення компонента Angular

Компоненти представляють основні будівельні блоки Angular. Кожна програма Angular має як мінімум один компонент. Тому створимо у папці src/app новий файл, який назвемо app.component.ts і в якому визначимо наступний код компонента:

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

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

На початку файлу визначається директива import, яка імпортує функціональність модуля angular/core, надаючи доступом до функції декоратора @Component.

Далі власне йде функція-декоратор @Component, яка асоціює метадані із класом компонента AppComponent. У цій функції, по-перше, визначається параметр selector або селектор css для HTML-елемента, який представлятиме компонент. По-друге, тут визначається параметр template або шаблон, який вказує на те, як треба візуалізувати компонент. У цьому шаблоні задана двостороння прив'язка за допомогою виразів [(ngModel)]="name" і {{name}} до певної моделі name.

І наприкінці власне експортується клас компонента AppComponent, у якому саме визначається змінна name – в даному випадку це порожній рядок.

Створення модуля програми

Додаток Angular складається з модулів. Модульна структура дозволяє легко підвантажувати та задіяти лише ті модулі, які безпосередньо необхідні. І кожен додаток має щонайменше один кореневий модуль. Тому створимо у папці src/app новий файл, який назвемо app.module.ts з таким вмістом:

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

```
import { AppComponent } from './app.component';
@NgModule({
	imports: [ BrowserModule, FormsModule ],
	declarations: [ AppComponent ],
	bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Цей модуль, який в даному випадку називається AppModule, буде вхідною точкою додатка.

За допомогою директив import тут імпортується низка потрібних нам модулів. Насамперед, це модуль NgModule. Для роботи з браузером також потрібний модуль BrowserModule. Оскільки наш компонент використовує елемент input або елемент форми, також підключаємо модуль FormsModule. І далі імпортується створений раніше компонент.

Запуск програми

Тепер нам треба вказати Angular, як запускати нашу програму. Для цього створимо в папці src (на рівень вище, ніж розташовані файли app.component.ts та app.module.ts) файл main.ts з таким вмістом:

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';
const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);
```

Цей код ініціалізує платформу, яка запускає програму, а потім використовує цю платформу для завантаження модуля AppModule.

Також у папці src визначимо ще один файл, який назвемо polyfills.ts з наступним кодом:

```
// zone використовується angular
import 'zone.js/dist/zone';
```

Цей файл визначає поліфіли - інструменти, які необхідні для підтримки програми на Angular різними браузерами.

Створення головної сторінки

Далі визначимо в папці src головну сторінку index.html програми:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Hello Angular</title>
</head>
<body>
    <my-app>Завантаження...</my-app>
</body>
</html>
```

В елементі body визначено елемент <my-app>, до якого власне і завантажуватиметься програма.

Визначення конфігурації

Оскільки для визначення коду програми використовується мова TypeScript, тому також створимо у кореневій папці проекту новий файл tsconfig.json:

```
{
  "compileOnSave": false,
  "compilerOptions": {
    "baseUrl": "./",
    "sourceMap": true,
    "declaration": false,
    "DownlevelIteration": true,
    "experimentalDecorators": true,
    "module": "esnext",
    "moduleResolution": "node",
    "target": "es2022",
    "typeRoots": [
        "node_modules/@types"
            ],
    "lib": [
        "es2022",
        "dom"
        ]
  },
    "files": [
        "src/main.ts",
        "src/polyfills.ts"
         ],
    "include": [
        "src/**/*.d.ts"
            ]
}
```

Цей файл визначає параметри компілятора TypeScript. Опція compilerOptions встановлює параметри компіляції. А опція "files" визначає файли, що компілюються. У нашому випадку це файл програми - main.ts, який підтягує решту файлів програми, і файл поліфілів polyfills.ts.

Angular.json

Для компіляції програми ми будемо використовувати Angular CLI, тому ми повинні описати поведінку CLI за допомогою файлу angular.json. Отже, додамо до кореневої папки проекту новий файл angular.json і визначимо в ньому такий вміст:

```
{
  "version": 1,
  "projects": {
    "helloapp": {
      "projectType": "application",
      "root": "",
      "sourceRoot": "src",
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-angular:browser",
          "options": {
            "outputPath": "dist/helloapp",
            "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": "helloapp:build"
          }
        }
      }
    }},
  "defaultProject": "helloapp"
}
```

Коротко пройдемося по структурі файлу. Спочатку визначається параметр version. Він визначає версію конфігурації проекту.

Далі йде секція projects, яка визначає налаштування для кожного проекту. У нашому випадку у нас лише один проект, який називається за назвою каталогу проекту – helloapp.

Проект визначає такі опції:

- projectType: тип проекту. Значення "application" вказує, що проект представлятиме додаток, який можна буде запускати у браузері;
- root: вказує на папку файлів проекту відносно робочого середовища. Порожнє значення відповідає кореневій папці проекту, тому що в даному випадку робоче середовище та каталог проекту збігаються;
- sourceRoot: визначає кореневу папку файлів із вихідним кодом. У нашому випадку це папка src, де власне визначено всі файли програми;
- architect: задає параметри для побудови проекту. У файлі package.json визначені команди build та serve, і для кожної з цих команд у секції architect задані свої налаштування.

Для кожної команди задається параметр builder, який визначає інструмент для побудови проекту. Так, для команди "build" встановлено значення "@angular-devkit/build-angular:browser" - даний білдер для побудови використовує збирач пакетів webpack. А для команди "serve" встановлено значення "@angular-devkit/build-angular:dev-server" - даний білдер запускає веб-сервер і розгортає на ньому скомпільовану програму.

Параметр options визначає параметри побудови файлів. Для команди "build" тут визначено такі опції:

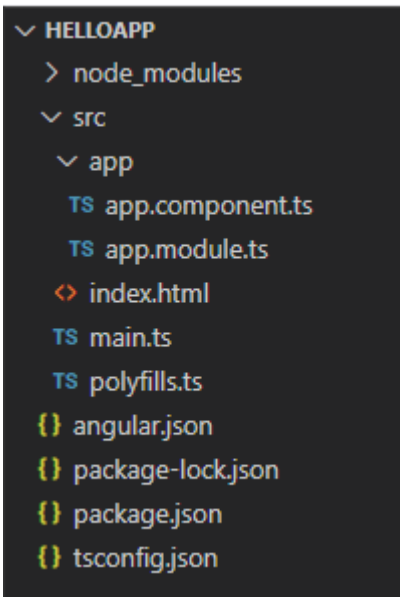
- ❖ outputPath: шлях, за яким буде публікуватися скомпільований додаток;
- ❖ index: шлях до головної сторінки програми;
- ❖ main: шлях до головного файлу програми, де власне запускається програма Angular;
- ❖ polyfills: шлях до файлу поліфілів;
- ❖ tsConfig: шлях до файлу конфігурації TypeScript;
- ❖ aot: вказує, чи використовуватиметься компіляція AOT (Ahead-Of-Time) (попередня компіляція перед виконанням). У цьому випадку значення true означає, що вона використовується.

Для команди "serve" вказана лише одна опція - browserTarget, яка містить посилання на конфігурацію для команди build - "helloapp:build". Тобто, по суті, ця команда використовує ту ж конфігурацію, що і команда build.

- Остання опція defaultProject вказує на проект за замовчуванням. У цьому випадку це наш єдиний проект.

Якщо ми використовуємо TypeScript для роботи з Angular і Angular CLI для компіляції, то файли package.json, tsconfig.json і angular.json фактично будуть присутні в кожному проекті. І їх можна переносити з проекту до проекту з мінімальними змінами. Наприклад, у файлі angular.json замість назви проекту "helloapp" буде відповідна назва проекту. У файлі package.json можна буде задати інші версії пакетів, якщо попередні версії застаріли. Можна буде змінити назву проекту, версію. Можна підправити налаштування TypeScript або Angular CLI, але в цілому загальна організація цих файлів буде такою ж самою.

У результаті в нас вийде така структура проекту:



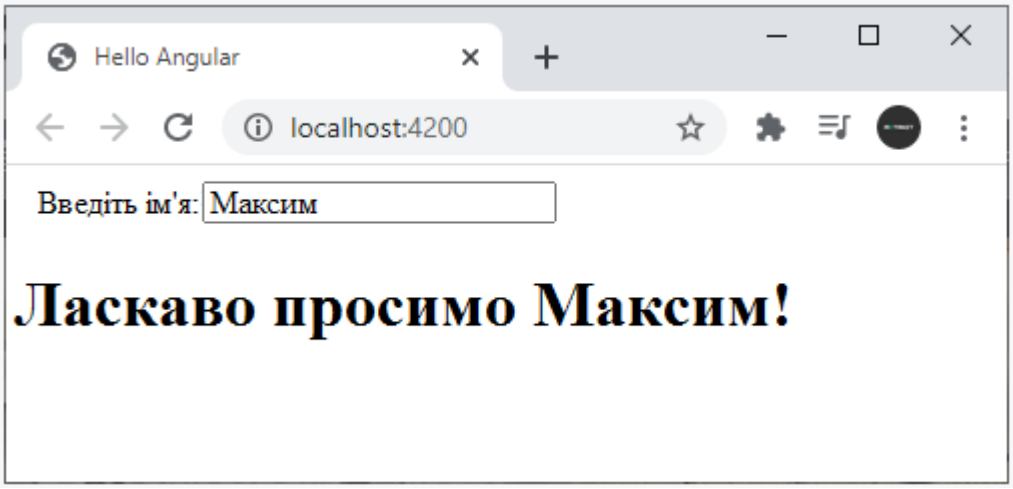
Запуск проекту

І тепер коли все готове, ми можемо запустити проект. Для цього в командному рядку (терміналі) перейдемо до папки проекту за допомогою команди cd і виконаємо команду ng serve:

```
C:\WINDOWS\system32>cd C:\angular\helloapp
C:\angular\helloapp>ng serve --open
```

```
C:\angular\helloapp>ng serve --open
**********************************************************************************************
This is a simple server for use in testing or debugging Angular applications locally.
It hasn't been reviewed for security issues.

DON'T USE IT FOR PRODUCTION!
**********************************************************************************************
✓ Browser application bundle generation complete.

Initial Chunk Files | Names         |  Raw Size | Estimated Transfer Size
main.js             | main          | 232.41 kB |                61.73 kB
polyfills.js        | polyfills     | 162.17 kB |                42.09 kB
runtime.js          | runtime       |   1.24 kB |               680 bytes

                    | Initial Total | 395.82 kB |               104.48 kB

Build at: 2022-06-05T14:17:42.408Z - Hash: 35e33ad985698b20 - Time: 5698ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://
localhost:4200/ **


✓ Compiled successfully.
```

Консольний виведення проінформує нас, які файли якого розміру створено. Крім того, ми зможемо побачити адресу, за якою запущено тестовий веб-сервер - за замовчуванням це "http://localhost:4200/". Якщо ми передаємо команді прапор --open, як у випадку вище, то Angular CLI автоматично відкриває браузер із запущеним додатком. І ми можемо звернутися до додатку:



Введемо в текстове поле якесь ім'я, і воно відразу відобразиться в заголовку.

Поки програма запущена, ми можемо змінити код, і Angular CLI майже моментально перекомпілює і перезапустить програму.

Варто зазначити, що консоль може бути прихована після запуску програми, але вона буде запущена, і ми зможемо до неї звертатися. При зміні в проекті і збереженні цих змін автоматично відбувається перекомпіляція і перезапуск проекту.

2) У попередній темі були розглянуті початкова інформація про Angular та встановлення необхідних інструментів та налаштування конфігурації для роботи з фреймворком. Тепер створимо додаток «Shopping list».

Для написання програми нам знадобиться звичайний текстовий редактор, хоча можна використовувати спеціальні середовища програмування, як Visual Studio Code, Netbeans, WebStorm та інші.

Крім того, для запуску програми Angular буде потрібно веб-сервер. Як веб-сервер знову ж таки можна використовувати безліч різних серверів - Apache, IIS, NodeJS і т.д. В даному випадку ми спиратимемося на NodeJS.

Отже, створимо на жорсткому диску папку програми. Хай вона буде називатися purchaseapp. У цій папці створимо новий файл package.json з таким вмістом:

```
{
"name": "purchaseapp",
"version": "1.0.0",
"description": "First Angular 16 Project",
"author": "Your name",
"scripts": {
      "ng": "ng",
      "start": "ng serve",
      "build": "ng build"
},
"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"
},
"devDependencies": {
      "@angular-devkit/build-angular": "~16.0.0",
      "@angular/cli": "~16.0.0",
      "@angular/compiler-cli": "~16.0.0",
      "typescript": "~5.0.4"
}
}
```

Також додамо до папки проекту новий файл tsconfig.json:

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

```
"compilerOptions": {
    "baseUrl": "./",
    "sourceMap": true,
    "declaration": false,
    "DownlevelIteration": true,
    "experimentalDecorators": true,
    "module": "esnext",
    "moduleResolution": "node",
    "target": "es2022",
    "typeRoots": [
        "node_modules/@types"
                    ],
    "lib": [
        "es2022",
        "dom"
          ]
                      },
    "files": [
        "src/main.ts",
        "src/polyfills.ts"
             ],
    "include": [
        "src/**/*.d.ts"
                ]
}
```

Файл package.json встановлює пакети та залежності, які будуть використовуватися проектом.

Файл tsconfig.json встановлює конфігурацію компілятора TypeScript.

Для збирання проекту будемо використовувати Angular CLI, тому також визначимо у папці проекту файл angular.json:

```
{
  "version": 1,
  "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"
      }
    }
  }},
  "defaultProject": "purchaseapp"
}
```

Після створення цих трьох файлів у папці проекту відкриємо командний рядок (термінал) і перейдемо до папки проекту за допомогою команди cd:

```
C:\WINDOWS\system32>cd C:\angular\purchaseapp
```

І потім виконаємо команду npm install, яка встановить усі необхідні модулі і створить папку node_modules:

```
C:\angular\purchaseapp>npm install
```

Після виконання цієї команди в папці проекту повинна з'явитися підпапка node_modules, яка містить всі залежності та пакети, що використовуються.

Потім створимо у папці проекту каталог src, а в цьому каталозі визначимо папку app.

До каталогу src/app додамо новий файл, який назвемо app.component.ts і який матиме наступний код:

```
import { Component } from '@angular/core';
class Item{
    purchase: string;
    done: boolean;
    price: number;

    constructor(purchase: string, price: number) {

        this.purchase = purchase;
        this.price = price;
        this.done = false;
    }
```

```
}

@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>`
})
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));
    }
}
```

Першим рядком імпортується функціональність компонента з angular/core.

Для роботи нам знадобиться допоміжний клас Item. Цей клас містить три поля: purchase (назва покупки), done (чи зроблена покупка) та price (її ціна).

У самому класі компонента визначається початковий Shopping list, який виводитиметься на сторінку:

```
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 }
   ];
```

І також у класі визначено метод додавання до цього списку:

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

Для виведення покупок в нашому компоненті визначено великий шаблон. Взагалі такі великі шаблони варто виносити в окремі файли, щоб зробити код компонента простіше. Але в нашому випадку нехай усе буде визначено в самому компоненті.

У шаблоні для виведення даних з масиву items в таблицю передбачена директива:

```
*ngFor="let item of items"
```

Крім того, зверху таблиці розташована форма для введення нового об'єкта Item. До натискання кнопки прив'язаний метод addItem() компонента.

Щоб задіяти цей компонент, додамо до каталогу src/app файл модуля app.module.ts:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
@NgModule({
    imports: [ BrowserModule, FormsModule ],
    declarations: [ AppComponent ],
    bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Рівнем вище в каталозі src визначимо файл main.ts для запуску проекту:

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';
const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);
```

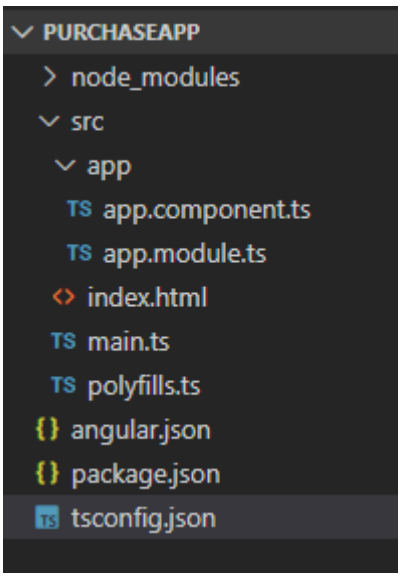
Також у каталозі src визначимо файл polyfills.ts, який необхідний для запуску програми:

```
import 'zone.js/dist/zone';
```

Наприкінці визначимо головну сторінку index.html у папці src:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title> Покупки</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css/bootstrap.min.css">
</head>
<body>
    <my-app>Завантаження...</my-app>
</body>
</html>
```

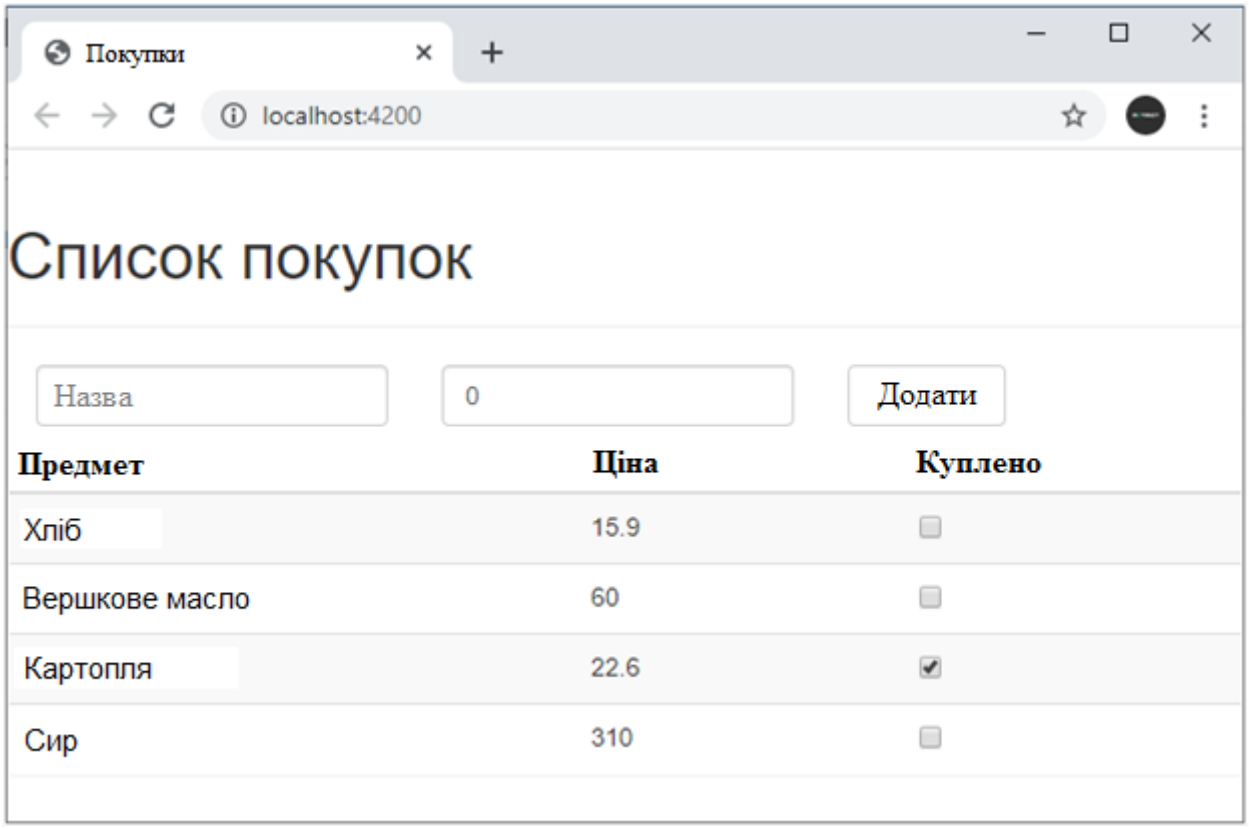
У результаті в нас вийде така структура проекту:



Тепер запустимо проект. Для цього в командному рядку (терміналі) перейдемо до папки проекту і виконаємо команду ng serve:

```
C:\angular\purchaseapp>ng serve --open
```

Після цього у веб-браузері звернемося до нашої програми та побачимо таблицю з покупками та форму для додавання нової покупки:



3) Зробити звіт по частині №1 роботи. Звіт повинен бути не менше 6 сторінок основної частини (шрифт Times New Roman, 14, полуторний інтервал). Титульний аркуш

приводиться у додатку. По Частині 1 лабораторної роботи звіт повинен містити наступні розділи:

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

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

c) Опис файлу package.json. Призначення, основні параметри.

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

e) Опис файлу angular.json. Призначення, основні параметри.

4) Розгорнути Angular-додаток «Shopping list» на платформі FireBase у проекті «ПрізвищеГрупаLaba1-1», наприклад, «KovalenkoIP01Laba1-1».

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.

I) Створіть новий Angular-додаток Binding1.

Angular підтримує механізм прив'язки, завдяки якому різні частини шаблону можуть бути прив'язані до деяких значень, визначених у компоненті.

У Angular є наступні форми прив'язки даних:

1) Прив'язка DOM до значення компонента (одностороння). У подвійних фігурних дужках вказується вираз, до якого йде прив'язка: {{вираз}}. Наприклад:
```
<h1>Ласкаво просимо {{name}}!</h1>
```

2) Прив'язка властивості елемента DOM до значення компонента (одностороння). Наприклад:
```
<input type="text" [value]="name" />
```

3) Прив'язка методу компонента до події DOM (генерація події DOM викликає метод на компоненті) (одностороння). Наприклад:
```
<button (click)="addItem(text, price)">Додати</button>
```

4) Двостороння прив'язка (two-way binding), коли елемент DOM прив'язаний до значення на компоненті, зміни на одному кінці прив'язки відразу призводять до змін на іншому кінці. Наприклад:
```
<input [(ngModel)]="name" placeholder="name">
```

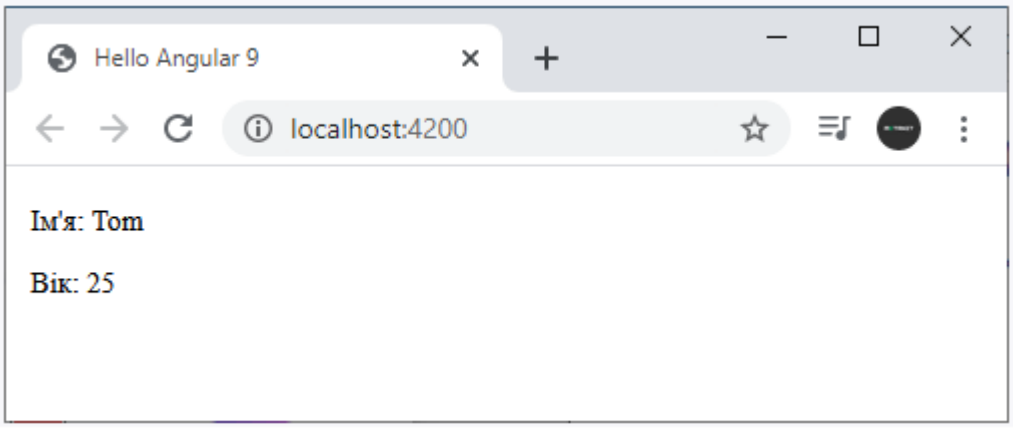
5) Прив'язка до атрибуту елемента html;
6) Прив'язка до класу CSS;
7) Прив'язка до атрибуту елемента html.

Вправа 1: Інтерполяція

Перший вид прив'язки полягає у використанні фігурних дужок, в які передається значення з компонента. Наприклад, нехай у нас буде визначено наступний компонент:

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

І при запуску програми вирази типу {{name}} автоматично замінюватимуться відповідними значеннями, визначеними в компоненті:



І якщо в процесі роботи програми властивості name і age в компоненті змінять своє значення, то також зміниться значення в розмітці html, яка прив'язана до цих властивостей.

Вправа 2: Прив'язка властивостей елементів HTML

Ми можемо прив'язати значення властивості елемента html. У цьому випадку властивість вказується у квадратних дужках:

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

@Component({
   selector: 'my-app',
   template: ` <p>Имя: {{name}}</p>
                  <p>Возраст: {{age}}</p>
                  <input type="text" [value]="name" />
                  <input type="text" [value]="age" />`
})
export class AppComponent {
```

```
    name = "Tom";
    age = 25;
}
```

Важливо розуміти, що тут йде прив'язка не до атрибуту, а саме до властивості елемента в JavaScript, який представляє цей елемент html. Тобто html-елемент <input> в javascript представлений інтерфейсом HTMLInputElement, який має властивість value.

Або інший приклад:

```
template: `<p [textContent]="name"></p>`
```

HTML-елемент <p> не має атрибуту textContent. Зате інтерфейс Node, який представляє даний елемент DOM, має властивість textContent, до якого можно здійснити прив'язку.

Вправа 3. Прив'язка до атрибуту

Іноді виникає необхідність виконати прив'язку не до властивості, а саме до атрибуту HTML-елемента. Хоча властивості та атрибути HTML-елементів можуть перетинатися, як це було показано вище у прикладі з властивістю/атрибутом value, але така відповідність буває не завжди. У цьому випадку ми можемо використовувати вираз:

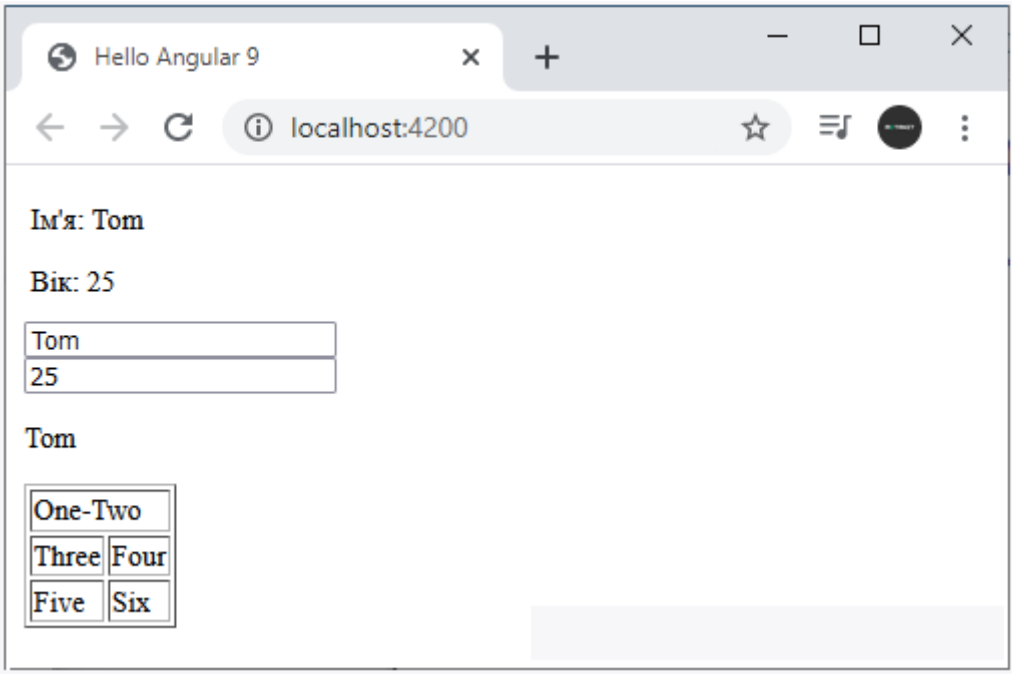
```
[attr.назва_атрибута]="значення"
```

Після префікса attr через точку йде назва атрибута.

Зазвичай подібна прив'язка застосовується до атрибутів елементів aria, svg та table. Наприклад, атрибут colspan, який поєднує стовпці таблиці, не має відповідної властивості. І в цьому випадку ми можемо застосовувати прив'язку до атрибутів:

```
import {Component} from '@angular/core';
@Component({
selector: 'my-app',
template: `
    <p>Ім'я: {{name}}</p>
    <p>Вік: {{age}}</p>
    <input type="text" [value]="name" /><br/>
    <input type="text" [value]="age" />
    <p [textContent]="name"></p>
    <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{
    name = 'Tom';
    age = 25;
    colspan = 2;
}
```



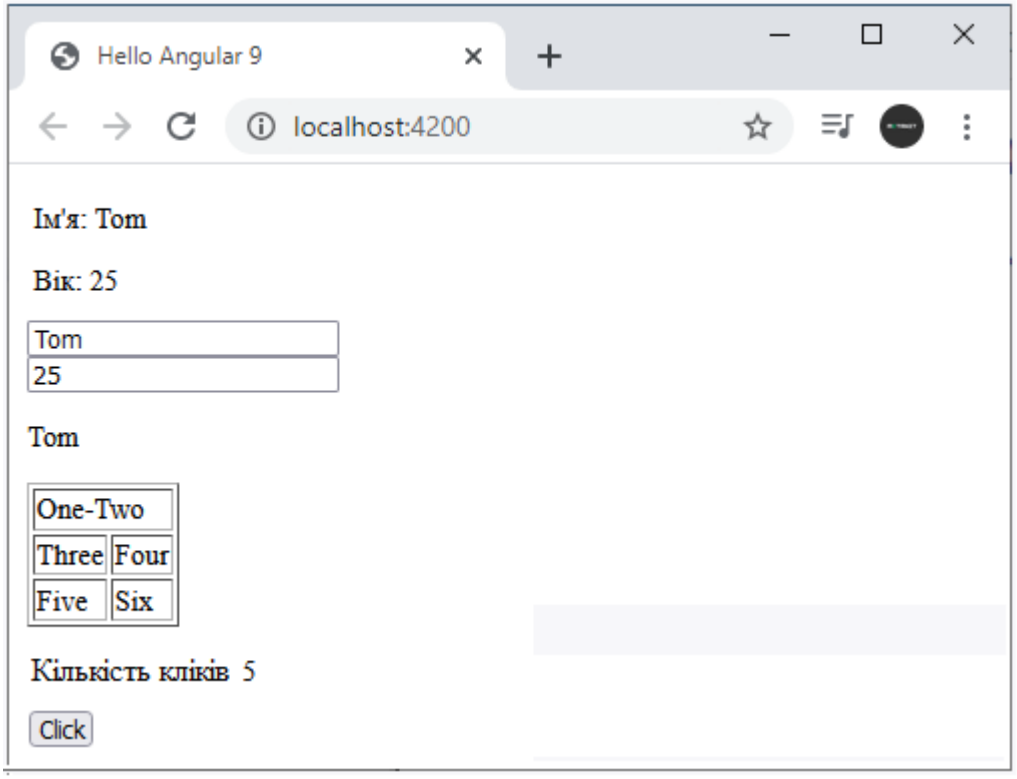
Вправа 4. Прив'язка до події

Прив'язка до події дозволяє пов'язати з подією елемента метод компонента:

```
import { Component } from '@angular/core';
@Component({
    selector: 'my-app',
    template: `<p>Ім'я: {{name}}</p>
                <p>Вік: {{age}}</p>
                <input type="text" [value]="name" /><br/>
                <input type="text" [value]="age" />
                <p [textContent]="name"></p>
                <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>
                <p>Кількість кліків {{count}}</p>
                <button (click)="increase()">Click</button>`
})
export class AppComponent {
    name = 'Tom';
    age = 25;
    colspan = 2;
```

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

У шаблоні визначено елемент button, який має подію click. Для обробки цієї події в класі AppComponent визначено метод increase(), який збільшує кількість умовних кліків. У результаті при натисканні на кнопку спрацює цей метод:



Як альтернативу ми могли б встановити прив'язку до події так:

```
template: `<p>Кількість кліків {{count}}</p>
<button on-click="increase()">Click</button>`
```

Після префікса on через дефіс йде назва події.

Ми також можемо передавати інформацію про подію через об'єкт $event:

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

@Component({
    selector: 'my-app',
    template: `<p>Ім'я: {{name}}</p>
                <p>Вік: {{age}}</p>
                <input type="text" [value]="name" /><br/>
                <input type="text" [value]="age" />
```

```
                <p [textContent]="name"></p>
                <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>
                <p>Кількість кліків {{count}}</p>
                <button (click)="increase()">Click</button>
                <p>Кількість кліків {{count_2}}</p>
                <button (click)="increase_2($event)">Click</button>`
})
export class AppComponent {
    name = 'Tom';
    age = 25;
    colspan = 2;
    count: number = 0;
    count_2: number = 0;
    increase() : void {
        this.count++;
                }
    increase_2($event : any) : void {
        this.count_2++;
        console.log($event);
                                    }
}
```

$event – це вбудований об'єкт, через який Angular передає інформацію про подію.

Вправа 5. Двостороння прив'язка

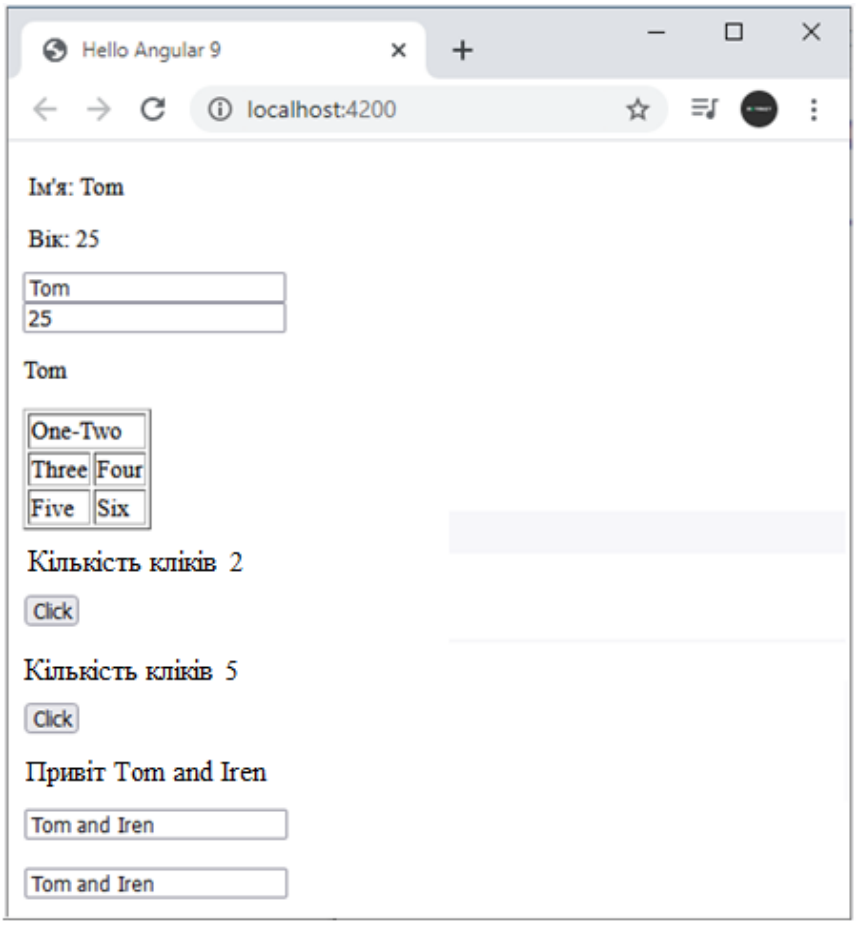
Двостороння прив'язка дозволяє динамічно змінювати значення на одному кінці прив'язки при змінах на іншому кінці. Як правило, двостороння прив'язка застосовується під час роботи з елементами введення, наприклад, елементами типу input. Наприклад:

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

@Component({
    selector: 'my-app',
    template: `<p>Ім'я: {{name}}</p>
                <p>Вік: {{age}}</p>
                <input type="text" [value]="name" /><br/>
                <input type="text" [value]="age" />
                <p [textContent]="name"></p>
                <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>
                <p>Кількість кліків {{count}}</p>
                <button (click)="increase()">Click</button>
                <p>Кількість кліків {{count_2}}</p>
                <button (click)="increase_2($event)">Click</button>
                <p>Привіт {{name}}</p>
                <input type="text" [(ngModel)]="name" /> <br><br>
                <input type="text" [(ngModel)]="name" />`
})
export class AppComponent {
    name = 'Tom';
    age = 25;
    colspan = 2;
    count: number = 0;
    count_2: number = 0;
    increase() : void {
        this.count++;
                }
    increase_2($event : any) : void {
        this.count_2++;
        console.log($event);
                                    }
}
```

Тут до властивості name класу AppComponent прив'язані одразу три елементи: параграф і два текстові поля. Тектові поля пов'язані з властивістю name двосторонньою прив'язкою. Для її створення застосовується вираз [(ngModel)] = "вираз".

У результаті зміни в текстовому полі позначатимуться на тексті у другому текстовому полі та параграфі:



II) Створіть новий додаток Binding2.

Вправа 6. Прив'язка до класів CSS

Прив'язка до класу CSS має таку форму:

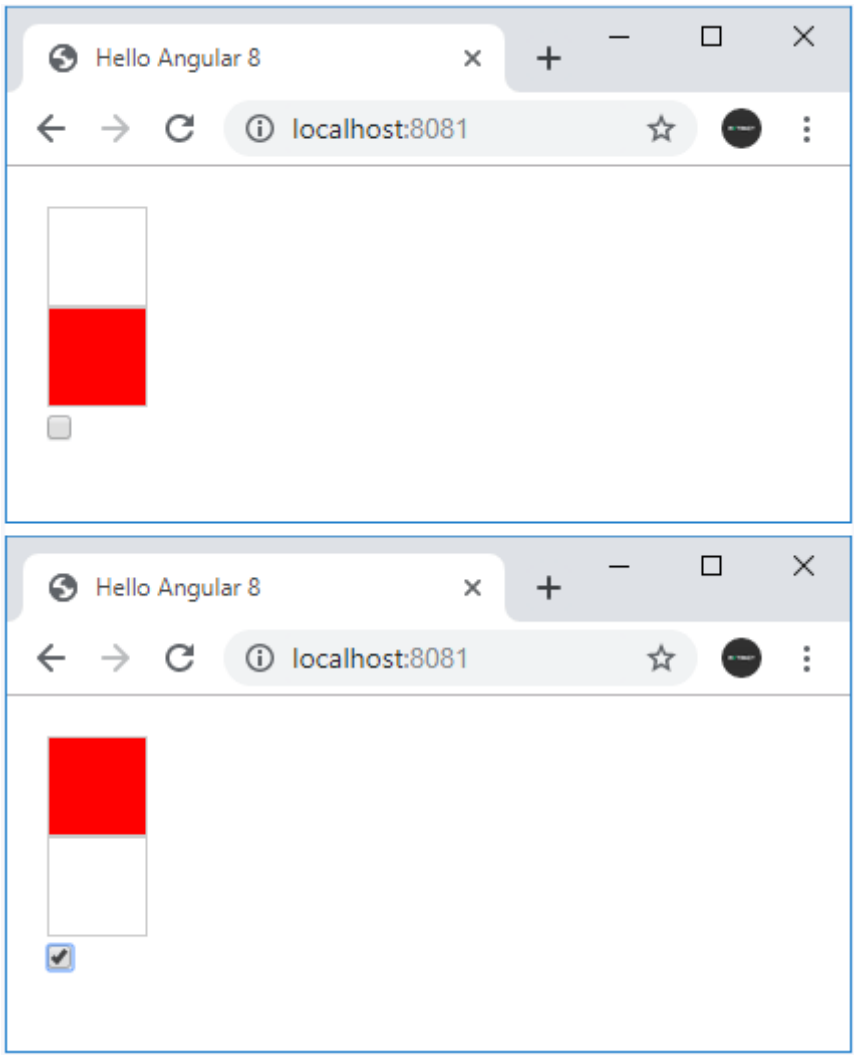
```
[class.ім'я_класу]="true/false"
```

Після префіксу class через точку вказується ім'я класу, яке хочемо додати чи видалити. Причому прив'язка йде до логічного значення. Якщо дорівнює true, то клас застосовується, якщо false - клас не застосовується. Наприклад:

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

```
    `]
})
export class AppComponent{
    isRed = false;
}
```

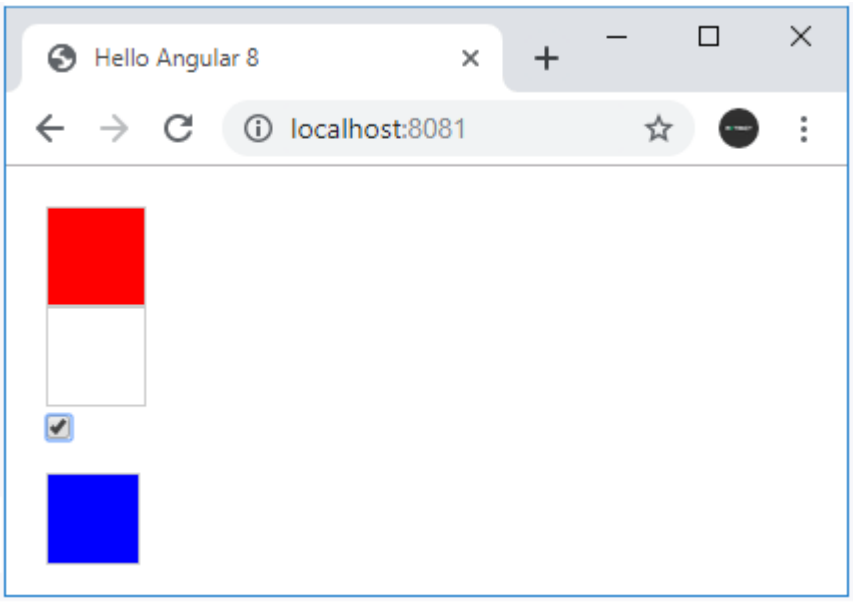
У даному випадку йде прив'язка змінної isRed до класу isredbox, який встановлює червоний колір тіла. У першого блоку div встановлюється клас, якщо зміна має значення true: [class.isredbox]="isRed". У другому блоці, навпаки, якщо змінна має значення false: [class.isredbox]="!isRed". Використовуючи двосторонню прив'язку до змінної isRed в елементі checkbox, ми можемо змінити її значення.



Варто зазначити, що ми також можемо використовувати прив'язку властивостей для встановлення класу:

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

@Component({
    selector: 'my-app',
    template: `  <div [class.isredbox]="isRed"></div>
          <div [class.isredbox]="!isRed"></div>
```

```
            <input type="checkbox" [(ngModel)]="isRed" />
            <div [class]="blue"></div>`,
    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"
}
```



Вправа 7. Прив'язка стилів

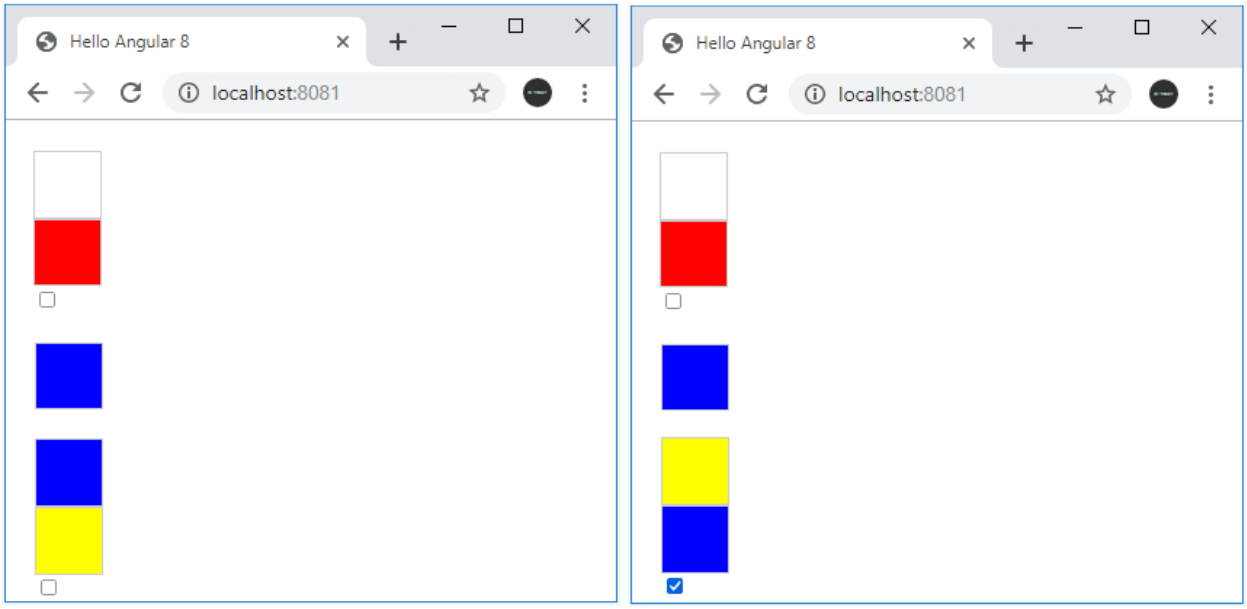
Прив'язка стилів має наступний синтаксис:

```
[style.стильова_властивость]="вираз ? A : B"
```

Після префіксу style через точку йде назва властивості стилю. Як значення передається деякий вираз: якщо воно повертає true, то стильовій властивості надається значення A; якщо воно повертає false, то стильовій властивості присвоюється значення B. Наприклад:

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

@Component({
    selector: 'my-app',
    template: `
            <div [class.isredbox]="isRed"></div>
          <div [class.isredbox]="!isRed"></div>
```

```
            <input type="checkbox" [(ngModel)]="isRed" />

                <div [class]="blue"></div> <br><br>

                <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}
     .isredbox{background-color:red;}
     .isbluebox{background-color:blue;}
          `]
})
export class AppComponent{
      isRed = false;
      isyellow=false;
      blue = "isbluebox"
}
```



Вираз [style.backgroundColor]="isyellow? 'yellow' : 'blue'" вказує, що якщо змінна isyellow дорівнює true, то стильовій властивості background-color передається жовтий колір, інакше передається голубий колір. Причому можна писати як style.background-color, і style.backgroundColor, тобто замість дефісу переводити наступний символ у верхній регістр.

III) Зробити звіт по частині №2 роботи (по Angular-додатках Binding1 та Binding2). Звіт повинен бути не менше 8 сторінок основної частини (шрифт Times New Roman, 14,

полуторний інтервал). Титульний аркуш приводиться у додатку. По Частині 2 звіт повинен містити наступні розділи:

1) Інтерполяція в Angular: огляд приклади використання.
2) Прив'язка властивостей елементів HTML: огляд приклади використання.
3) Прив'язка до атрибуту: огляд приклади використання.
4) Прив'язка до події: огляд приклади використання.
5) Двостороння прив'язка: огляд приклади використання.
6) Прив'язка до класів CSS: огляд приклади використання.
7) Прив'язка стилів: огляд, приклади використання.

IV) Angular-додаток Binding1 розвернути на платформі FireBase у відповідному проекті з назвою «ПрізвищеГрупаLaba1-2», наприклад, KovalenkoIP01Laba1-2.

ДОДАТОК

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

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт по лабораторній роботі №____

_____

назва лабораторної роботи

з дисципліни: «Реактивне програмування»

Студент: _____

Група: _____

Дата захисту роботи: _____

Викладач: доц. Полупан Юлія Вікторівна

Захищено з оцінкою: _____

Київ, 2024