Лабораторне заняття №3: Створення проекту «Todo».

Мета: Знайомство з основним процесом розробки Angular-додатку з використанням редактора Visual Studio Code. Навчитися застосовувати компоненти Angular Material для стилізації контенту та створювати моделі даних. Навчитися створювати двосторонні прив'язки даних.

Завдання:

I) Встановити Visual Studio Code, пакет Angular Material. Створити каркас додатку «Todo». Додати наступні можливості до проекту «Todo»: відображення списку завдань у вигляді таблиці; додавання нових завдань користувачем; фільтрування завдань користувачем.

II) Зробити звіт по роботі.

III) Angular-додаток «Todo» розгорнути на платформі Firebase у проекті з ім'ям «ПрізвищеГрупаLaba3».

Підготовка середовища розробки

Для розробки Angular потрібна певна підготовка.

Встановлення Node.js

Node.js — це середовище виконання JavaScript програм на стороні сервера, яке використовується більшістю фреймворків веб-додатків, в тому числі Angular. Версія Node.js, яка використовується в цій роботі — 16.13.0. Повний набір випусків 16.13.0 з інсталяторами для Windows і macOS, а також двійкові пакети для інших платформ доступні за адресою https://nodejs.org/dist/v16.13.0.

Завантажте та запустіть інсталятор і переконайтеся, що опція «npm package manager» Add to PATH та її два параметри обрано, як показано на рис. 2-1.

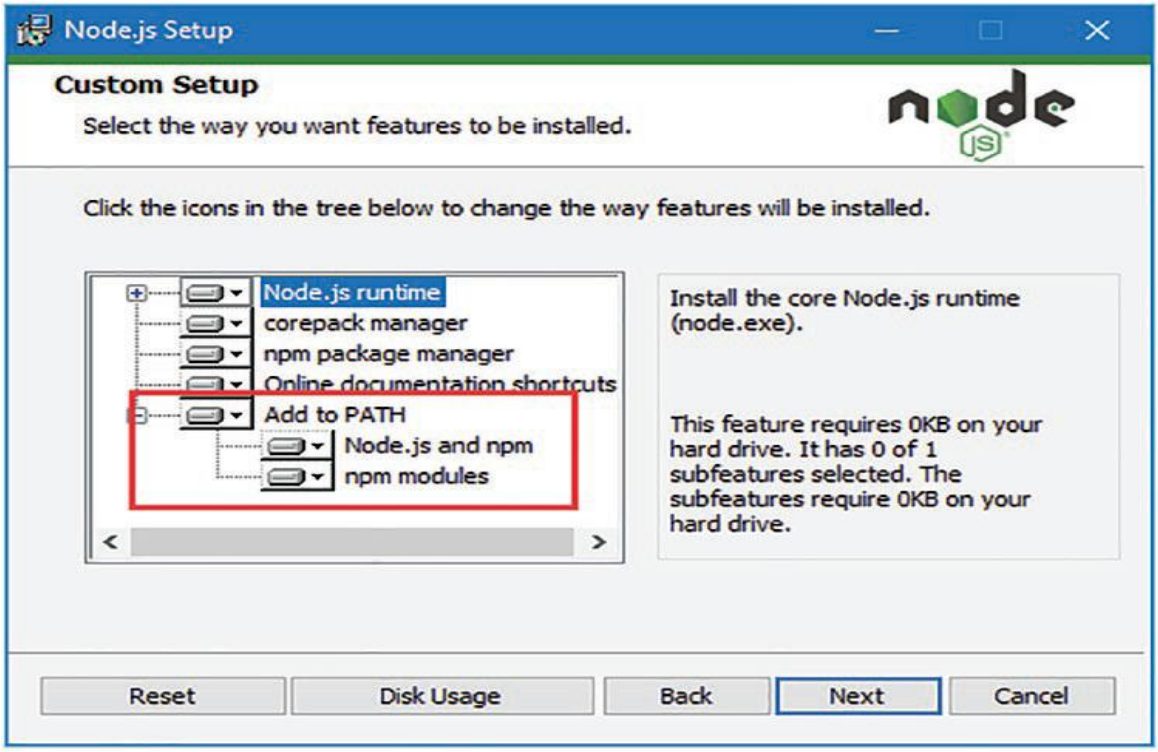

Рис. 2-1.Встановлення Node.js

Після завершення встановлення відкрийте новий командний рядок і виконайте команду, показану в Лістингу 2-1.

Лістинг 2-1. Запуск Node.js

```
node -v
```

Якщо інсталяція пройшла належним чином, то ви побачите такий номер версії:

```
v16.13.0
```

Інсталятор Node.js містить менеджер пакетів проекту (NPM), який використовується для керування пакетами в проекті. Виконайте команду, показану в лістингу 2-2, щоб забезпечити роботу NPM.

Лістинг 2-2.Запуск npm

```
npm –v
```

Якщо все працює як треба, ви побачите наступний номер версії:

```
8.1.0
```

Встановлення редактора

Angular розробку можна виконати будь-яким редактором для програмістів, яких існує нескінченна кількість. Деякі редактори мають розширену підтримку для роботи з Angular, включаючи виділення ключових термінів і хорошу інтеграцію з інструментарієм.

При виборі редактора одним з найважливіших міркувань є можливість фільтрації вмісту проекту, щоб ви могли зосередитися на роботі з деякою підмножиною файлів. У проекті Angular може бути багато файлів, і багато з них мають подібні назви, тому можливість знайти та відредагувати потрібний файл є важливою. Редактори роблять це можливим різними способами, або шляхом представлення списку файлів, відкритих для редагування, або шляхом надання можливості виключити файли з певними розширеннями.

Таблиця 2.1. Популярні редактори з підтримкою Angular

Назва	Огляд
Sublime Text	Sublime Text — комерційний кросплатформенний редактор із пакетами для підтримки більшості мов програмування, фреймворків і платформ. За деталями звертайтесь за адресою www.sublimetext.com
Atom	Atom — безкоштовний кросплатформенний редактор із відкритим кодом, що надає особливу увагу можливостям налаштувань і розширення. За подробицями звертайтеся за адресою atom.io
Brackets	Brackets — безкоштовний редактор з відкритим кодом, розроблений компанією Adobe. За деталями звертайтеся за адресою brackets.io
WebStorm	WebStorm — платний кросплатформенний редактор з безліччю інтегрованих інструментів, щоб розробнику не прийшлось користуватися командним рядком під час розробки. За подробицями звертайтеся за адресою www.jetbrains.com/webstorm
Visual Studio Code	Visual Studio Code — безкоштовний кросплатформенний редактор із відкритим кодом, розроблений компанією Microsoft, із хорошими можливостями розширення. За подробицями звертайтеся за адресою code.visualstudio.com

Якщо у вас ще немає бажаного редактора для розробки веб-додатків, тоді рекомендується встановити Visual Studio Code, який безкоштовно надається корпорацією Майкрософт і має чудову підтримку розробки Angular. Ви можете завантажити Visual Studio Code з https://code. visualstudio.com.

Встановлення пакета розробки Angular

Команда Angular надає повний набір інструментів командного рядка, які спрощують розробку Angular. Ці інструменти поширюються в пакеті під назвою @angular/cli. Виконайте команду, показану в лістингу 2-3, щоб встановити інструменти розробки Angular.

Лістинг 2-3.Встановлення пакета розробки Angular
```
npm install --global @angular/ cli@13.0.3
```

Зверніть увагу на два дефіси перед глобальним аргументом. Якщо ви використовуєте Linux або macOS, вам може знадобитися використовувати sudo, як показано в лістингу 2-4.

Лістинг 2-4.Використання sudo для встановлення пакета розробки Angular
```
sudo npm install --global @angular/ cli@13.0.3
```

Вибір браузера

Остаточним вибором є браузер, який ви використовуватимете для перевірки своєї роботи під час розробки. Всі Браузери поточного покоління мають хорошу підтримку розробників і добре працюють з Angular. В роботі використовується Google Chrome.

Створення проекту Angular

Розробка Angular виконується як частина проекту, який містить усі файли, необхідні для збірки та запуску програми разом із файлами конфігурації та статичним вмістом (наприклад, файлами HTML і CSS). Для створення нового проекту, відкрийте командний рядок, перейдіть у зручне розташування та запустіть команду, показану в Лістинг2-5. Зверніть увагу на використання подвійних і одинарних дефісів під час введення цієї команди.

Лістинг 2-5. Створення нового проекту Angular
```
ng new todo --routing false --style css --skip-git --skip-tests
```

Команда ng є частиною @angular-cli пакету, а ng new створює новий проект. Аргументи конфигурації проекту - це ті обрані параметри, які підходять для першого проекту. Процес створення нового проекту може зайняти деякий час, оскільки там потрібна велика кількість інших пакетів, які потрібно завантажити під час першого запуску команди ng new.

Відкриття проекту для редагування

Після завершення роботи команди ng new, скористайтеся встановленим редактором коду, щоб відкрити папку todo , яка містить новий проект. Папка todo містить конфігураційні файли та інструменти, які використовуються в розробці Angular. Папка src/app містить програмний код та вміст і є папкою, у якій виконується більшість розробок. Рис. 2-2 показує початковий вміст папки проекту, як він відображається в Visual Studio Code та основні моменти папки src/app. Ви можете бачити дещо інший вигляд в інших редакторах, деякі з яких приховують файли та папки, що не часто використовується безпосередньо під час розробки, наприклад папку node_modules, яка містить пакети, на які покладається розробка Angular додатку.

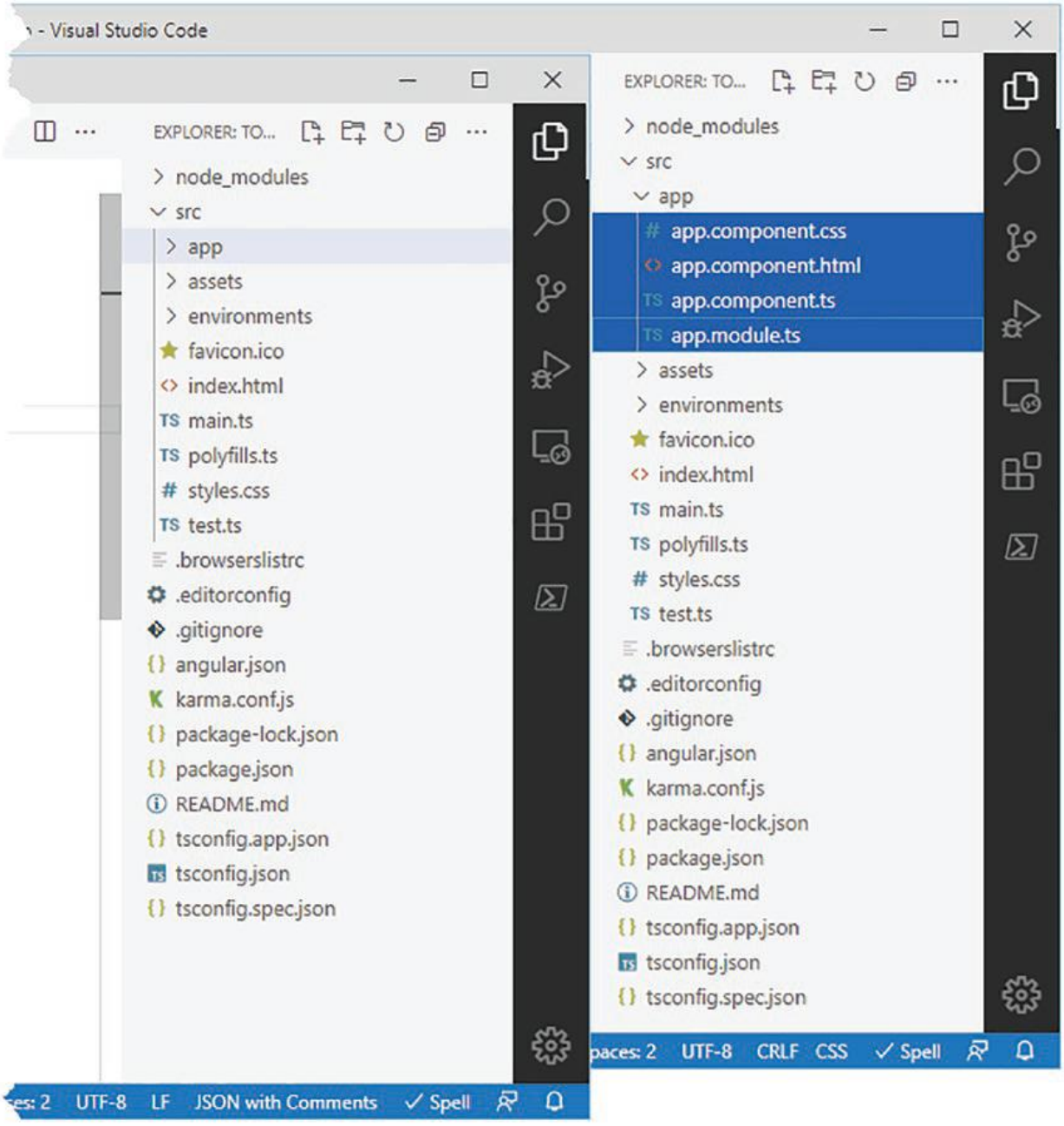

Рис. 2-2. Початковий вміст проекту Angular

Запуск Angular Development Tools

Останньою частиною процесу налаштування є запуск інструментів розробки, які будуть компілювати, заповнювати вміст, доданий до проекту командою ng new. Щоб запустити проект Angular з папки todo, використовуйте командний рядок терміналу Visual Studio Code та команду, показану в лістингу2-6.

Лістинг 2-6. Запуск Angular додатку з папки todo.

```
ng serve
```

Ця команда запускає інструменти розробки Angular, які включають компілятор і веб-сервер, який використовується для тестування програми Angular у браузері. Цей процес може зайняти деякий час. Під час процесу запуску ви побачите такі повідомлення:

```
Browser application bundle generation complete.
Initial Chunk Files   | Names         |      Size
vendor.js             | vendor        |   1.83 MB
polyfills.js          | polyfills     | 339.11 kB
styles.css, styles.js | styles        | 212.38 kB
main.js               | main          |  51.42 kB
runtime.js            | runtime       |   6.84 kB
                      | Initial Total |   2.43 MB
Build at: 2021-11-25T17:35:50.484Z - Hash: 8c7aa6b9cef0e8e7 - Time: 13641ms
** Angular Live Development Server is listening on localhost:4200, open your browser on
http://localhost:4200/ **
Compiled successfully.
```

Якщо ви бачите «compiled successfully» в кінці процесу, то компіляція пройшла успішно. Інтегрований веб-сервер прослуховує запити на порту 4200, тому відкрийте нове вікно браузера та введіть запит http://localhost:4200, який покаже контент за замовчуванням Angular-додатку, показаний на рис. 2-3.

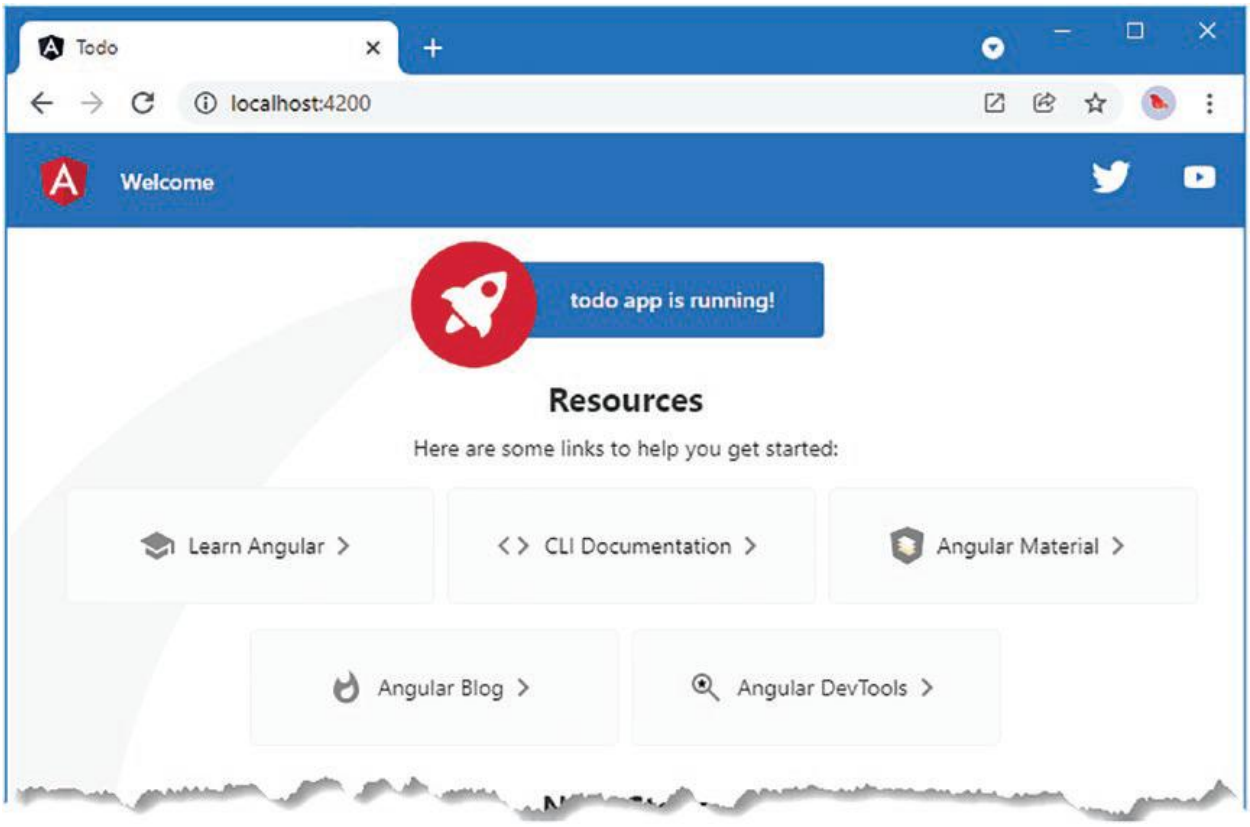

Рис. 2-3. Вміст заповнювача в новому проекті Angular

Додавання функцій до програми

Тепер, коли інструменти розробки запущені, почнемо працювати над процесом створення простого Angular-додатку, яке керуватиме списком справ. Користувач зможе побачити список справ, перевіряти, вимикати завершені справи та створювати нові. Будемо припускати, що є лише один користувач і не потрібно турбуватися про збереження стану даних у програмі, що означає, що зміни в списку справ будуть втрачені, якщо вікно браузера закрити або перезавантажити.

Створення моделі даних

Відправною точкою для більшості програм є модель даних, з якою додаток працює. Моделі даних можуть бути великими та складними, але для програми todo потрібно описати лише дві речі: пункт справ і чи виконано справу чи ні.

Програми Angular написані на TypeScript, який є надмножиною JavaScript. Головною перевагою TypeScript є те, що він підтримує статичні типи даних, що робить JavaScript розробку більш схожу на розробку мовами C# та Java. Команда ng new містить пакети, необхідні для компіляції коду TypeScript у чистий JavaScript, який може виконуватись у браузерах.

Щоб створити модель даних для програми, додайте файл під назвою todoItem.ts у папку todo/src/app із наступним вмістом, показаним у лістингу2-7. (Файли TypeScript мають .ts розширення.)

Лістинг 2-7.Вміст файлу todoItem.ts у папці src/app

```
export class TodoItem {
        constructor(
        public task: string,
        public complete: boolean = false) {
// оголошення не потрібні
        }
}
```

Особливості мови TypeScript, є те, що код можна писати, використовуючи стандартні функції JavaScript та додаткові функції які надає TypeScript. Коли код компілюється, функції TypeScript видаляються, а результатом є код JavaScript, який можуть виконувати браузери.

Ключові слова export, class і constructor стандартні для JavaScript. Не всі браузери підтримують ці функції, тому процес створення додатків Angular може перетворити цей тип функцій на код, який можуть зрозуміти старі браузери.

Ключове слово export стосується модулів JavaScript. При використанні модулів кожен TypeScript або JavaScript файл вважається самостійною функціональною одиницею, а ключове слово export використовується для ідентифікації даних або типів, які ви хочете використовувати в іншому місці програми. JavaScript модули використовуються для управління залежностями, які виникають між файлами в проекті.

Ключове слово class оголошує клас, а constructor позначає конструктор класу. На відміну від в інших мов, таких як C#, JavaScript не використовує назву класу для визначення конструктора.

Клас TodoItem визначає конструктор, який отримує два параметри з назвою task і complete. Значенням цих параметрів присвоєно public властивість. Якщо значення не надано для параметру complete, тоді значення за замовчуванням буде false:

```
...
constructor(public task: string, public complete: boolean = false) {
...
```

Стислий конструктор уникає блоку шаблонного коду, в якому інакше потрібно було б визначити властивості та присвоювати їм значення, отримані конструктором.

Обидва параметри конструктора в лістингу 2-7 позначаються типом даних:

```
...
constructor(public task: string, public complete: boolean = false) {
```

...

У стандартному JavaScript значення мають типи і можуть бути призначені будь-якій змінній, яка є джерелом плутанини для програмістів, що звикли до змінних, які визначені для зберігання певного типу даних. TypeScript приймає більш традиційний підхід до типів даних, і компілятор TypeScript повідомить про помилку, якщо використовуються несумісні типи.

Створення класу «TodoList».

Щоб створити клас, який представляє список завдань, додайте файл з іменем todoList.ts до папки src/app із наступним вмістом, показаним у лістингу2-8.

Лістинг 2-8. Вміст файлу todoList.ts у папці src/app

```
import { TodoItem } from "./todoItem";
export class TodoList {
constructor (public user: string, private todoItems: TodoItem[] = [])
{
// no statements required
}
get items(): readonly TodoItem[] {
        return this.todoItems;
        }
addItem(task: string) {
        this.todoItems.push(new TodoItem(task));
        }
}
```

Ключове слово import оголошує залежність від класу TodoItem. Класс TodoList визначає конструктор, який отримує початковий набір значень. Щоб не надавати необмежений доступ до массиву TodoItem об’єктів, визначимо властивість під назвою items, яка повертає масив лише для читання, і це зроблено використовуючи лише одне ключове слово readonly. Компілятор TypeScript згенерує помилку для будь-якої спроби змінити вміст масиву.

Відображення даних для користувача

Зараз нам знадобиться механізм відображення значень даних моделі користувачу. В Angular це робиться за допомогою шаблону, який є фрагментом HTML і який містить вирази, що обчислюються Angular, а результат надсилається до браузера.

Коли проект був створений за допомогою команди ng new, було додано файл шаблону під назвою app.component.html у папку src/app. Відкрийте цей файл для редагування та замініть вміст тим, що показано в лістингу2-9.

Лістинг 2-9. Заміна вмісту файлу app.component.html у папці src/app

```
<h3>
        {{ username }}: список справ
        <h5>{{ itemCount }} елементів</h5>
</h3>
```

Незабаром додамо інші функції до шаблону, але цього достатньо, щоб почати. Відображення даних у шаблоні виконується за допомогою подвійних дужок—{{i}}— і Angular обчислює все, що розміщено між подвійними дужками щоб отримати значення для виводу.

Символи {{i}} визначають прив'язку даних; інакше кажучи, вони створюють відношення між шаблоном і значенням даних. Прив’язки даних є важливою функцією Angular. У цьому випадку прив’язки даних повідомляють Angular отримати значення username та itemCount властивостей та вставити їх у вміст h3 і div елементів.

Як тільки ви збережете файл, інструменти розробки Angular спробують створити проект. Компілятор згенерує такі помилки:

```
Error: src/app/app.component.html:2:6 - error TS2339: Property 'username' does not exist
on type 'AppComponent'.
2   {{ username }}'s To Do List
        ~~~~~~~~

  src/app/app.component.ts:5:16
    5   templateUrl: './app.component.html',
                     ~~~~~~~~~~~~~~~~~~~~~~

    Error occurs in the template of component AppComponent.
Error: src/app/app.component.html:3:10 - error TS2339: Property 'itemCount' does not exist
on type 'AppComponent'.
3   <h6>{{ itemCount }} Items</h6>
            ~~~~~~~~~

  src/app/app.component.ts:5:16
    5   templateUrl: './app.component.html',
                     ~~~~~~~~~~~~~~~~~~~~~~

    Error occurs in the template of component AppComponent.
```

Ці помилки виникають через те, що вирази в прив’язках даних покладаються на властивості, які не існують. Далі цю проблему буде вирішено, але ці помилки показують важливу характеристику Angular, яка заключається в тому, що шаблони включені в процес компіляції та що будь-які помилки в шаблоні обробляються просто як помилки у звичайних кодових файлах.

Оновлення компонента

Angular компонент відповідає за керування шаблоном і надання йому даних і логіки. Компоненти є частиною програми Angular, що виконують більшу частину важкої роботи. Як наслідок, їх можна використовувати для будь-яких завдань.

У нашому випадку потрібен компонент, який би діяв як міст між класами моделі даних і шаблоном, для цього необхідно створити екземпляр класса TodoList, заповнити його зразком TodoItem об’єктів, і при цьому надати шаблону с username і itemCount необхідні властивості.

Коли проект був створений командою ng new у проект було додано файл під назвою app.component.ts у папці src/app. Як випливає з назви файлу, це компонент. Застосуйте зміни, показані в лістингу 2-10 до файлу app.component.ts.

Лістинг 2-10. Редагування вмісту файлу app.component.ts у папці src/app

```
import { Component } from '@angular/core';
import { TodoList } from "./todoList";
import { TodoItem } from "./todoItem";
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {
    private list = new TodoList("Тарас", [
        new TodoItem("Зробити пробіжку", true),
        new TodoItem("Купити квіти"),
        new TodoItem("Забрати квитки"),
    ]);
    get username(): string {
        return this.list.user;
    }
    get itemCount(): number {
        return this.list.items
            .filter(item => !item.complete).length;
    }
}
```

Код у списку можна розбити на три основні області, які описані нижче.

Розуміння імпорту

Ключове слово import оголошує залежності від модулів JavaScript як у проекті, так і в сторонніх пакетах. Ключове слово import використовується тричі в лістингу 2-10:

```
...
import { Component } from '@angular/core';
import { TodoList } from "./todoList";
import { TodoItem } from "./todoItem";
...
```

Перша команда import використовується для завантаження модуля @angular/core, який містить ключові функції Angular, включаючи підтримку компонентів. При роботі з

модулями команда import перераховує в фігурних дужках типи, які імпортуються. У цьому випадку команда import використовується для завантаження типу Component з модуля. Модуль @angular/core містить багато класів, упакованих разом, щоб браузер міг завантажити їх усі в одному файлі JavaScript.

Інші команди import використовуються для оголошення залежностей від визначених класів моделі даних раніше. Шлях для цього типу імпорту починається з ./; що вказує на те, що модуль визначено відносно поточного файлу.

Зауважте, що в жодній команді import не вказано розширення файлів.

Розуміння декоратора

Найдивніша частина коду в списку виглядає так:

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

Це декоратор (decorator), який надає метадані про клас. Декоратор @Component, як випливає з назви, повідомляє Angular, що це компонент. Декоратор передає конфігураційну інформацію через свої властивості: selector, templateUrl, styleUrls.

Властивість selector визначає селектор CSS, який відповідає елементу HTML, до якого буде застосовано компонент.

Коли ви запрошуєте http://localhost:4200, браузер отримує вміст файлу index.html, який був доданий до папки src під час створення проекту. Цей файл містить спеціальний елемент HTML, представлений нижче:

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Список справ</title>
    <base href="/">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
    <app-root></app-root>
</body>
</html>
```

Інструменти розробки Angular автоматично додають елементи сценарію до цього HTML, які вказують браузеру на запит файлів JavaScript, і які предоставляються фреймворком Angular і визначають спеціальні функції в проекті.

Коли виконується код Angular, значення властивості selector, визначена компонентом, використовується для пошуку зазначеного елемента в HTML-документі, і саме цей елемент містить вміст, який виводиться додатком. Наразі достатньо зрозуміти, що значення властивості selector відповідає елементу в документі HTML.

Властивість templateUrl вказує шаблон компонента, який має назву app.component.html.

Властивість styleUrl визначає одну або кілька таблиць стилів CSS, які використовуються для стилізації елементів, що створюються компонентом і його шаблоном. Налаштування в цьому компоненті визначає файл з іменем app.component.css, який використовується для створення стилів CSS.

Розуміння класу

Остання частина списку визначає клас, який Angular може створити для створення компонента.

```
...
export class AppComponent {
    private list = new TodoList("Тарас", [
        new TodoItem("Зробити пробіжку", true),
        new TodoItem("Купити квіти"),
        new TodoItem("Забрати квитки"),
    ]);
    get username(): string {
        return this.list.user;
    }
    get itemCount(): number {
        return this.list.items.filter(item => !item.complete).length;
    }
}
...
```

Ці оператори визначають клас під назвою AppComponent, що має приватну властивість list, якій призначено об’єкт TodoList і заповнюється масивом TodoItem об'єктів. Класс AppComponent визначає лише для читання властивості username та itemCount, які покладаються на об’єкт TodoList для створення своїх значень.

Властивість username повертає значення властивості TodoList.user, а itemCount використовуючі стандартні функції масиву JavaScript фільтрує Todoitem об'єкти, якими

керує TodoList, щоб вибрати їх, які є невиконаними, і повертає кількість знайдених відповідних об’єктів.

Значення для властивості itemCount отримується за допомогою лямбда-функції, також відомої як функція жирної стрілки, що є більш стислим способом вираження стандартної функції JavaScript. Стрілка в лямбда виразі читається як «йде до», наприклад «елемент йде до не item.complete.”

Коли ви збережете зміни у файлі TypeScript, інструменти розробки Angular побудують проект. Цього разу помилок бути не повинно, оскільки компонент визначив необхідні властивості шаблону. Вікно браузера буде автоматично перезавантажено, результат буде такий, як на рис. 2-4.

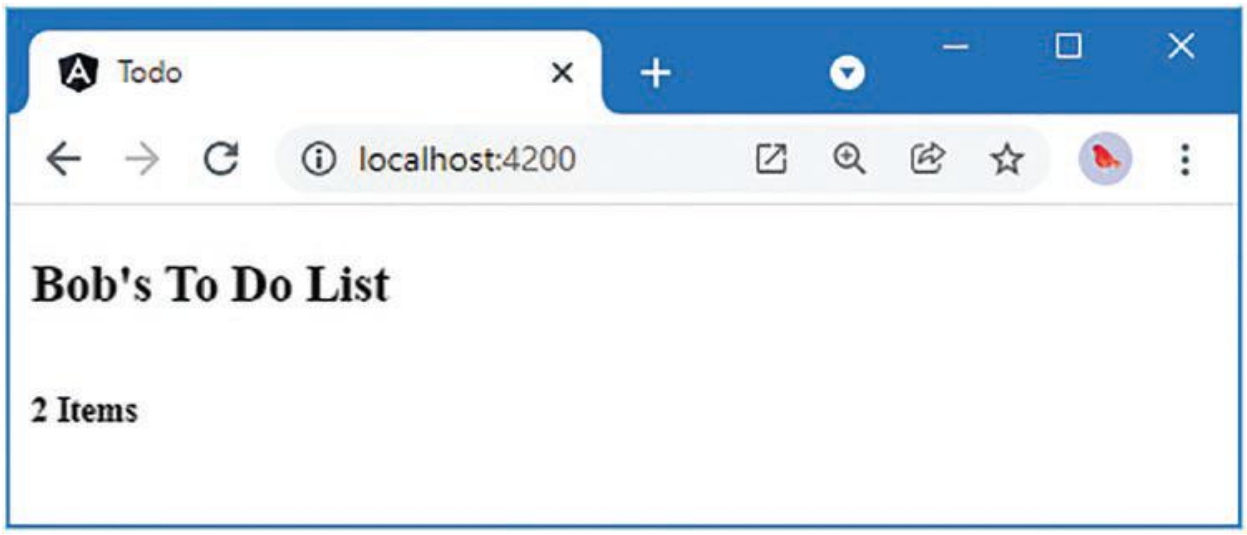

Рис. 2-4. Генерація вмісту в прикладі програми.

Стилізація контенту програми

Для оформлення HTML-вмісту, створеного додатку, будемо використовувати пакет Angular Material, який містить набір компонентів для використання в програмах Angular. Пакет Angular Material дає можливість отримати доступ до «офіційної» бібліотеки компонентів, і вона безкоштовна у використанні, повна корисних функцій, і добре інтегрована в решту фреймворку Angular.

Використовуйте Ctrl+C, щоб зупинити інструменти розробки Angular, і використовуйте окремий командний рядок, щоб запустити команду, показану в лістингу 2-11. Запускати команду потрібно в папці проекту.

Лістинг 2-11.Додавання пакета Angular Material

```
ng add @angular/ material@13.0.2 --defaults
```

Коли буде запропоновано, натисніть Y, щоб встановити пакет. Після встановлення пакета відкрийте файл app.module.ts у папці src і внесіть зміни, показані в лістингу 2-12. Ці зміни декларують залежності від функцій Angular Material, які використовуються в нашому проекті. Цей файл також називається модулем, що означає, що в проекті Angular є два типи модулів: модулі JavaScript і Angular модулі. Це приклад модуля Angular. Зараз достатньо знати, що саме так використовуються функції пакету Angular Material включені в наш проект.

Лістинг 2-12. Додавання залежностей у файл app.module.ts у папці src

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { FormsModule } from '@angular/forms'
import { MatButtonModule } from '@angular/material/button';
import { MatToolbarModule } from '@angular/material/toolbar';
import { MatIconModule } from '@angular/material/icon';
import { MatBadgeModule } from '@angular/material/badge';
import { MatTableModule } from '@angular/material/table';
import { MatCheckboxModule } from '@angular/material/checkbox';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatInputModule } from '@angular/material/input';
import { MatSlideToggleModule } from '@angular/material/slide-toggle';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    FormsModule,
    MatButtonModule, MatToolbarModule, MatIconModule, MatBadgeModule,
    MatTableModule, MatCheckboxModule, MatFormFieldModule, MatInputModule,
    MatSlideToggleModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Кожна функція, яку використовує програма, збільшує кількість коду JavaScript, який потрібно завантажити браузером, тому функції вмикаються окремо.

Застосування компонентів Angular Material

Наступним кроком є використання компонентів, що містяться в пакеті Angular Material, для стилізації створеного додатком контенту. Компоненти застосовуються за допомогою елементів і атрибутів HTML у файлі шаблону, як показано в лістингу 2-13.

Лістинг 2-13.Застосування компонентів у файлі app.components.html папки src/app

```
<mat-toolbar color="primary" class="mat-elevation-z3">
<span>{{ username }}: список справ</span>
<mat-icon matBadge="{{ itemCount }}" matBadgeColor="accent">checklist</mat-icon>
</mat-toolbar>
```

Новий вміст шаблону базується на функціях пакета Angular Material, кожна з яких застосовується по-різному. Першою функцією є панель інструментів, яка застосовується за допомогою елемента mat-toolbar, з вмістом панелі інструментів, що міститься у відкриваючому та закриваючому тегах:

```
...
<mat-toolbar color="primary" class="mat-elevation-z3">
...
</mat-toolbar>
...
```

Атрибут color використовується для визначення кольору панелі інструментів. Пакет Angular Material використовує кольорові теми, а основне значення, яке використовується для налаштування панелі інструментів, представляє колір теми за замовчуванням. Клас, який було призначено елементу mat-toolbar, застосовує стиль, наданий пакетом Angular Material для створення рельєфного вигляду контенту:

```
...
<mat-toolbar color="primary" class="mat-elevation-z3">
...
```

Іншими функціями є іконка та значок, які використовуються разом, щоб вказати кількість незавершених елементів у списку справ користувача. Іконка наноситься за допомогою mat-icon елементу:

```
...
<mat-icon matBadge="{{ itemCount }}" matBadgeColor="accent">елеменів</mat-icon>
...
```

Іконки вибираються шляхом вказівки імені тега mat-icon. У цьому випадку іконка контрольного списку буде виділена:

```
...
<mat-icon matBadge="{{ itemCount }}" matBadgeColor="accent">елементів</mat-
icon>
...
```

Ви можете переглянути повний набір доступних значків, відвідавши сторінку https://fonts.google.com/icons?selected=Material+Icons. Іконки поширюються за допомогою файлів шрифтів і команди, яка використовується для додавання Angular Material до проекту. У лістингу 2-13 додаються посилання до файлу index.html з папки src, необхідні для цих файлів.

Значок застосовується як опція до елемента mat-icon за допомогою matBadge і matBadgeColor атрибутів:

```
...
<mat-icon matBadge="{{ itemCount }}" matBadgeColor="accent">елементів</mat-
icon>
...
```

Бейджі – це невеликі круглі індикатори стану, які використовуються для представлення користувачеві числа або символів, що робить їх ідеальними для вказівки кількості справ в нашому додатку. Значення атрибута matBadge встановлює вміст значка, а атрибут matBadgeColor використовується для встановлення кольору, який у цьому випадку є акцентним, позначає колір теми, яка використовується для виділення.

Збережіть зміни у файлі шаблону та скористайтеся командою ng serve, щоб почати побудову Angular-додатку. Після завершення побудови проекту скористайтеся браузером для запиту http://localhost:4200 і ви побачите вміст, показаний на рис. 2-5.

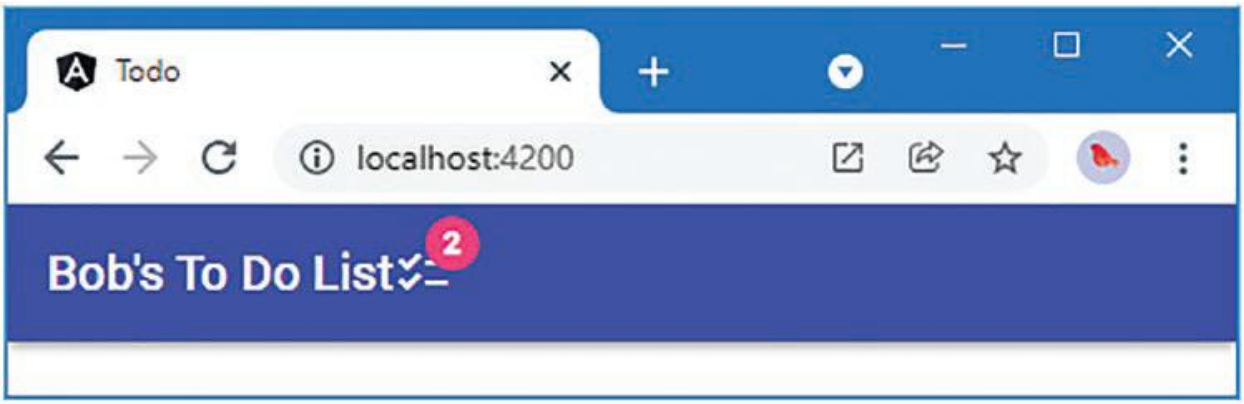

Рис. 2-5. Знайомство з компонентами Angular Material

Зверніть увагу, що шаблон у лістингу 2-13 містить ті самі прив’язки даних, представлені раніше в цій роботі, і вони все ще працюють таким же чином, надаючи доступ до даних у компоненті.

Визначення стилю CSS Spacer

Пакет Angular Material загалом вичерпний, але в ньому відсутні прокладки, які допомагають позиціонувати контент. Я хочу позиціонувати span елемент, який містить ім’я користувача по центру в рядку заголовка, іконку та значок праворуч. Першим кроком є створення класу CSS, який буде налаштовувати HTML елементи, які повинні заповнити доступний простір. Як зазначалося раніше, декоратор в app.component.ts файл містить властивість styleUrls, яка використовується для вибору файлів CSS, що застосовуються до шаблону компонента. Додайте стиль, показаний у лістингу 2-14 до файлу app.component.css, що створений при створенні проекту.

Лістинг 2-14. Додавання стилю CSS у файл app.component.css у папці src/app

```
.spacer { flex: 1 1 auto }
```

Запис з лістингу 2-14 застосовує стиль до будь-якого елементу, якому призначено клас з іменем spacer. Стиль встановлює flex властивість, яка є частиною CSS *flexible box*, також відома як *flexbox*. Використовується Flexbox щоб розмістити елементи HTML, щоб вони адаптувалися до доступного простору та могли реагувати на зміни, наприклад коли змінюється розмір вікна браузера або повертається екран пристрою. Налаштування в лістингу 2-14 налаштовує елемент, який потрібно збільшити, щоб заповнити будь-який доступний простір. Якщо маємо кілька елементів HTML в якомусь контейнері, якому призначено spacer класс, то доступний простір буде розподілено рівномірно між ними. Додайте елементи, показані в лістингу 2-15 до файлу шаблону, щоб застосувати spacers в макеті.

Лістинг 2-15.Додавання елементів у файл app.component.html у папці src/app

```
<mat-toolbar color="primary" class="mat-elevation-z3">
   <span class="spacer"></span>
   <span> span>{{ username }}'s To Do List </span>
   <span class="spacer"></span>
   <mat-icon matBadge="{{ itemCount }}" matBadgeColor="accent"> checklist </mat-
icon>
</mat-toolbar>
```

Коли ви збережете файл, інструменти розробки Angular виявлять зміни, перекомпілюють проект і перезавантажать браузер, створюючи новий макет, показаний на рис. 2-6.

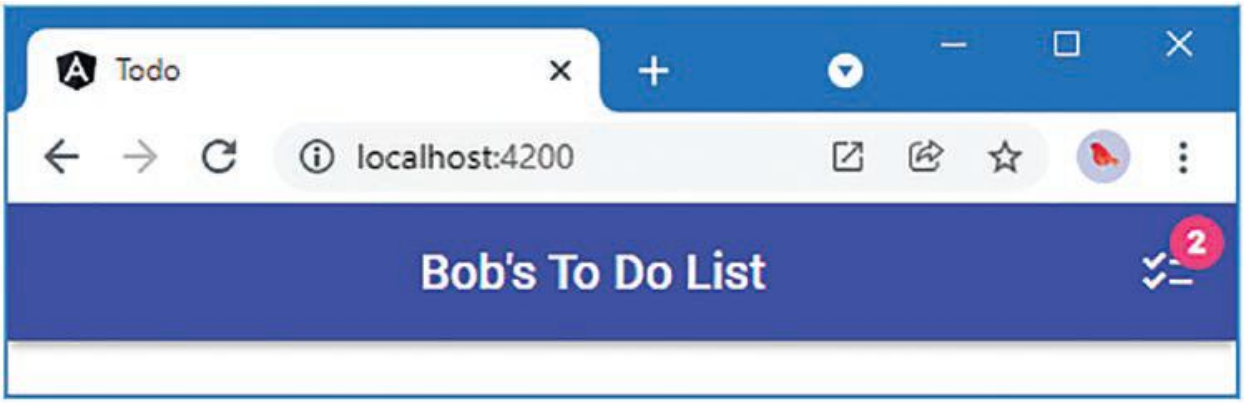

Рис. 2-6. Додавання прокладок до компонування компонентів

Відображення списку завдань

Наступним кроком буде відображення завдань. Лістинг 2-16 додає властивість до компонента, який забезпечує доступ до елементів у списку.

Лістинг 2-16. Додавання властивості у файл app.component.ts у папці src/app

```
import { Component } from '@angular/core';
import { TodoList } from "./todoList";
import { TodoItem } from "./todoItem";
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {
    private list = new TodoList("Bob", [
        new TodoItem("Go for run", true),
        new TodoItem("Get flowers"),
        new TodoItem("Collect tickets"),
    ]);
    get username(): string {
        return this.list.user;
    }
    get itemCount(): number {
        return this.list.items.filter(item => !item.complete).length;
    }
    get items(): readonly TodoItem[] {
        return this.list.items;
    }
}
```

Щоб відобразити користувачеві деталі кожного елемента, будемо використовувати компонент таблиці Angular Material, як показано в лістингу 2-17, що дозволяє легко представити користувачеві табличні дані.

Лістинг 2-17.Додавання таблиці у файл app.component.html у папці src/app

```
<mat-toolbar color="primary" class="mat-elevation-z3">
    <span class="spacer"></span>
    <span>{{ username }} : список справ</span>
    <span class="spacer"></span>
    <mat-icon matBadge="{{ itemCount }}" matBadgeColor="accent">checklist</mat-icon>
</mat-toolbar>
<div class="tableContainer">
  <table mat-table [dataSource]="items" class="mat-elevation-z3 fullWidth">
      <ng-container matColumnDef="id">
      <th mat-header-cell *matHeaderCellDef>№</th>
      <td mat-cell *matCellDef="let i = index"> {{ i + 1 }} </td>
      </ng-container>

      <ng-container matColumnDef="task">
      <th mat-header-cell *matHeaderCellDef>Task</th>
      <td mat-cell *matCellDef="let item"> {{ item.task }} </td>
      </ng-container>

      <ng-container matColumnDef="done">
          <th mat-header-cell *matHeaderCellDef>Done</th>
          <td mat-cell *matCellDef="let item"> {{ item.complete }} </td>
      </ng-container>

      <tr mat-header-row *matHeaderRowDef="['id', 'task', 'done']"></tr>
      <tr mat-row *matRowDef="let row; columns: ['id', 'task', 'done'];"></tr>
  </table>
</div>
```

Компонент Angular Material table застосовується шляхом додавання mat-table атрибут до стандартного HTML елементу таблиці, а дані, які міститиме таблиця, вказуються за допомогою спеціального dataSource атрибуту:

```
...
<table mat-table [dataSource]="items" class="mat-elevation-z3 fullWidth">
...
```

Квадратні дужки ([i]) позначають прив’язку атрибута, яка є прив’язкою даних, що використовується для встановлення атрибута елемента, надаючи компоненту таблиці Angular Material дані для виводу на дисплей. Це зв’язування налаштовує таблицю для відображення значень, які повертаються властивістю items, як визначено в лістингу 2-16.

Компонент таблиці налаштовується шляхом визначення стовпців, які відображатимуться користувачеві. Елемент ng-container використовується для групування

вмісту контенту, і в цьому випадку він використовується для групування елементів, які визначають заголовок і вміст стовпців:

```
...
<ng-container matColumnDef="task">
<th mat-header-cell *matHeaderCellDef>Task</th>
<td mat-cell *matCellDef="let item"> {{ item.task }} </td>
</ng-container>
...
```

Таке розташування елементів визначає заголовок і вміст осередку колонки з назвою task. Заголовок визначається за допомогою елементу <th>, до якого застосовані атрибути mat-header-cell та *matHeaderCellDef:

```
...
<th mat-header-cell *matHeaderCellDef>Task</th>
...
```

Ефект від mat-header-cell атрибута полягає в налаштуванні зовнішнього вигляду осередку заголовка так, щоб він відповідав решті таблиці. Ефект від *matHeaderCellDef атрибута полягає в налаштуванні поведінки осередку.

Контент осередків таблиці визначається за допомогою елементу <td>, до якого застосовано атрибути mat-cell та *matCellDef. Атрибут *matCellDef використовується для вибору вмісту, який буде відображатися в кожному осередку таблиці:

```
...
<td mat-cell *matCellDef=" let item"> {{ item.task }} </td>
...
```

Вираз, якому призначений атрибут *matCellDef отримує значення для кожного елемента в джерелі даних з назвою item, і ця змінна використовується в прив’язки даних для заповнення осередків таблиці. У цьому випадку значення властивості task буде відображено в осередку таблиці.

Angular Material при роботі з таблицями надає додаткові контекстні дані для створення рядків таблиці, включаючи індекс елемента даних, для якого створюється поточний рядок і до якого можна отримати доступ:

```
...
<td mat-cell *matCellDef=" let i = index"> {{ i + 1 }} </td>
...
```

Вираз, який використовується для цієї клітинки таблиці має значення index, яке надане компонентом таблиці з назвою «i» та використовується в прив’язці даних для створення простого лічильника.

Колонки для заголовка таблиці та тіла таблиці обираються шляхом застосування атрибутів до елементів <tr>:

```
...
<tr mat-header-row *matHeaderRowDef="['id', 'task', 'done']"></tr>
<tr mat-row *matRowDef="let row; columns: ['id', 'task', 'done'];"></tr>
...
```

Ці елементи вибирають id, task та done рядків визначених в лістингу 2-17. Може здатися дивним, що колонки не застосовуються автоматично, але цей підхід корисний, коли потрібно вибрати різні колонки на основі введення користувача.

Визначення додаткових стилів

Останнім кроком налаштування таблиці є визначення додаткових стилів CSS, як показано в лістингу 2-18.

Лістинг 2-18. Визначення стилів у файлі app.component.css у папці src/app

```
.spacer { flex: 1 1 auto }
.tableContainer { padding: 15px }
.fullWidth { width: 100% }
```

Перший новий стиль обирає будь-який елемент, з класом tableContainer і застосовує відступи навколо нього. Другий новий стиль встановлює стиль, який застосовується для елементів з класом fullWidth і призначений для заповнення 100 відсотків доступної для них ширини.

Збережіть зміни і інструменти розробки Angular скомпільують проект і перезавантажать браузер, створюючи контент, показаний на рис. 2-7.

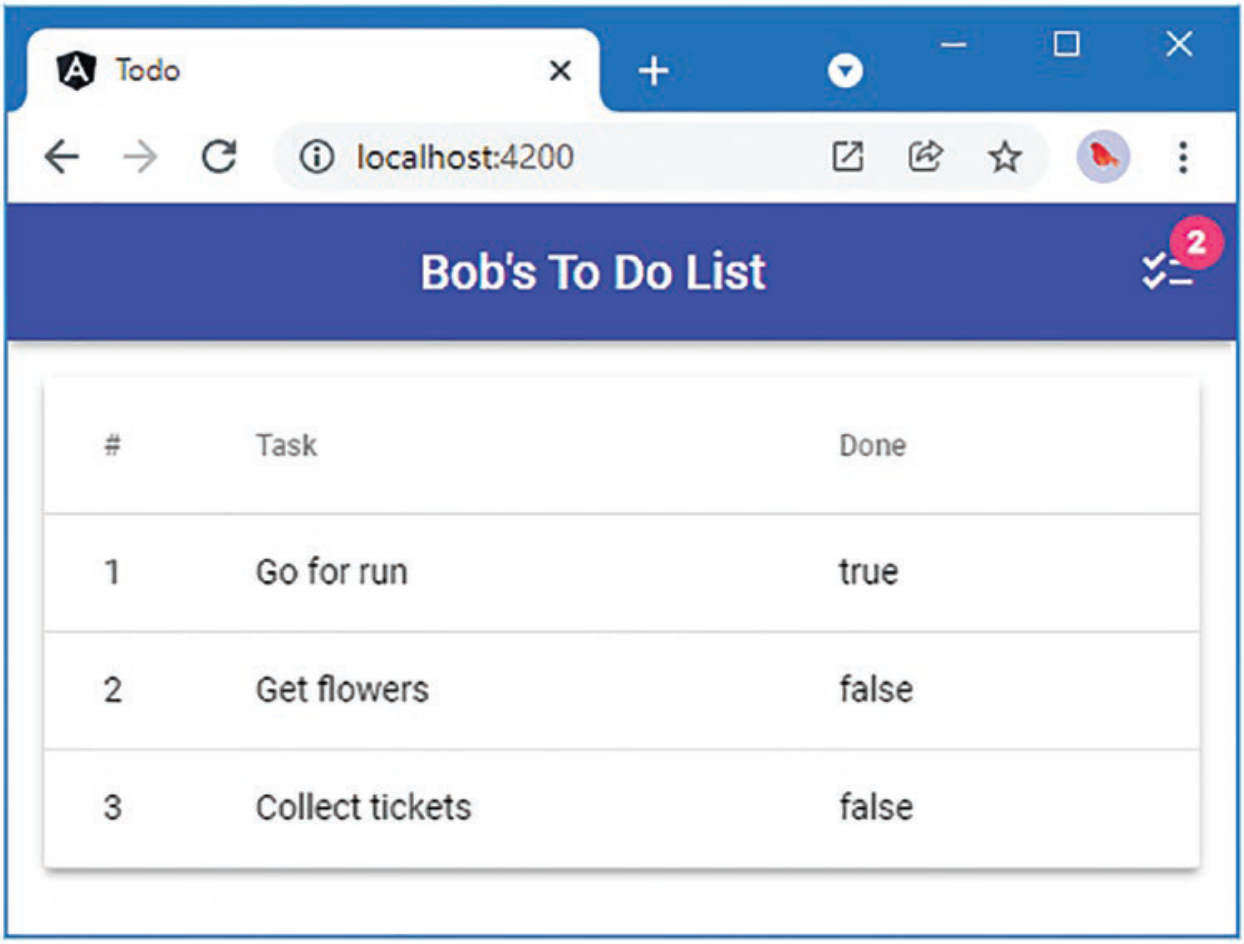

Рис. 2.7. Відображення списку справ

Створення двосторонньої прив’язки даних

На даний момент шаблон містить лише односторонні прив'язки даних, що означає, що вони використовуються для відображення значення даних, але не можуть його змінити. Angular також підтримує двосторонню прив'язку даних, яку можна використовувати щоб відобразити значення даних і змінювати їх. Двосторонні прив’язки використовуються з елементами форми HTML. В лістинг 2-19 при допомозі Angular Material додається до шаблону checkbox, який дозволяє користувачу позначати завдання як виконані.

Лістинг 2-19. Додавання Checkbox до файлу app.component.html у папці src/app

```
<mat-toolbar color="primary" class="mat-elevation-z3">
    <span class="spacer"></span>
    <span>{{ username }}'s To Do List</span>
    <span class="spacer"></span>
    <mat-icon matBadge="{{ itemCount }}" matBadgeColor="accent">checklist</mat-icon>
</mat-toolbar>
<div class="tableContainer">
```

```
    <table mat-table [dataSource]="items" class="mat-elevation-z3 fullWidth">
        <ng-container matColumnDef="id">
            <th mat-header-cell *matHeaderCellDef>#</th>
            <td mat-cell *matCellDef="let i = index"> {{ i + 1 }} </td>
        </ng-container>
        <ng-container matColumnDef="task">
            <th mat-header-cell *matHeaderCellDef>Task</th>
            <td mat-cell *matCellDef="let item"> {{ item.task }} </td>
        </ng-container>
        <ng-container matColumnDef="done">
            <th mat-header-cell *matHeaderCellDef>Done</th>
            <td mat-cell *matCellDef="let item">
                <mat-checkbox [(ngModel)]="item.complete" color="primary">
                  {{ item.complete }}
                </mat-checkbox>
            </td>
        </ng-container>
        <tr mat-header-row *matHeaderRowDef="['id', 'task', 'done']"></tr>
        <tr mat-row *matRowDef="let row; columns: ['id', 'task', 'done'];"></tr>
    </table>
</div>
```

Елементи mat-checkbox застосовуються Angular Material до checkbox компоненту. Двостороння прив'язка виражається за допомогою спеціального атрибута:

```
...
<mat-checkbox [(ngModel)]="item.complete" color="primary">
...
```

Комбінація дужок відома як *банан в коробці*. Ці дужки позначають двосторонню прив'язку даних, а ngModel є функцією Angular і використовується для встановлення двосторонніх прив’язок до елементів форми, таких як checkbox.

Збережіть зміни у файлі, і інструменти розробки Angular перекомпілюють проект і перезавантажать браузер для відображення вмісту, показаного на рис. 2-8. Ефект полягає у додаванні checkbox до кожного елементу завдання item.complete і використовується для встановлення виконаного завдання або ні. Відповідна властивість complete також буде оновлюватися, коли користувач перемикає checkbox.

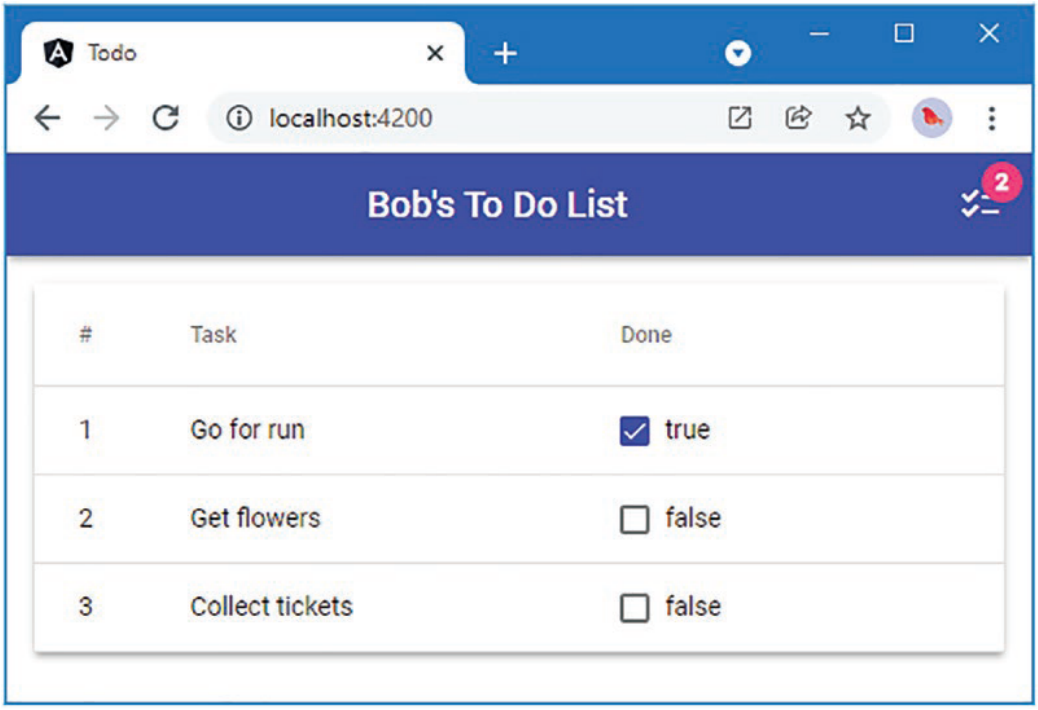

Рис. 2-8. Використання двосторонніх прив'язок

Значення true/false у виводі залишені спеціально, щоб продемонструвати важливий аспект того, як працює Angular зі змінами. Кожного разу, коли ви перемикаєте checkbox, відповідне текстове значення змінюється, а також лічильник відображається значком, як показано на рис. 2-9.

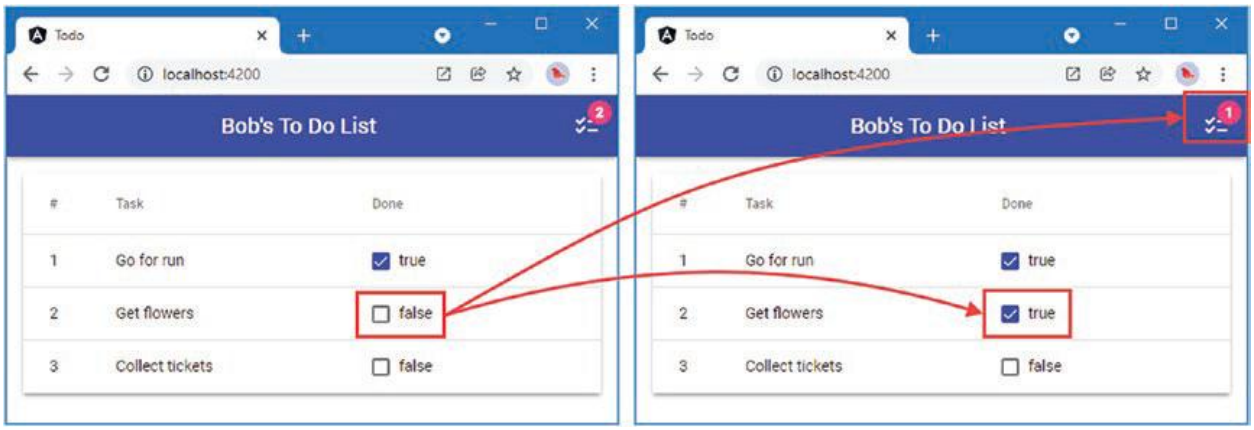

Рис. 2.9. Перемикання прапорця

Така поведінка розкриває важливу функцію Angular: *жива* модель даних. Це означає прив’язки даних — навіть односторонні прив’язки даних — оновлюються, коли змінюється модель даних. Це спрощує розробку веб-додатку, оскільки це означає, що вам не потрібно турбуватися про те, щоб забезпечувати відображення оновленнь, коли програма змінює свій стан.

Можна легко забути, що під шаблонами, компонентами та живою моделлю даних, Angular використовує JavaScript API браузера для створення та відображення звичайних елементів HTML. Клацніть правою кнопкою миші по одному з checkboxes у вікні браузера та виберіть «Інспектор» з меню браузера «Інструменти веб-розробника» (точний пункт меню залежатиме від обраного браузера). Відкриються інструменти розробника в браузері, і ви може досліджувати вміст HTML, який відображається браузером. В Інспекторі ви побачите ефект від застосування Angular Material checkbox в шаблоні як звичайний HTML checkbox, ось так:

```
...
<input type="checkbox" class="mat-checkbox-input cdk-visually-hidden"
      id="mat-checkbox-1-input" tabindex="0" aria-checked="false">
...
```

Меню браузера «Інструменти веб-розробника» це простий і ефективний спосіб зрозуміти, що робить програма.

Фільтрування завершених завдань

Checkboxes дозволяють оновлювати модель даних, а наступним кроком є видалення елементів справ, які були позначені як виконані. Лістинг 2-20 змінює властивість items компоненту, щоб вона відфільтровувала будь-які елементи, які були завершені.

Лістинг 2-20.Фільтрування завдань у файлі app.component.ts у папці src/app

```
import { Component } from '@angular/core';
import { TodoList } from "./todoList";
import { TodoItem } from "./todoItem";
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {
    private list = new TodoList("Bob", [
        new TodoItem("Go for run", true),
        new TodoItem("Get flowers"),
        new TodoItem("Collect tickets"),
    ]);
    get username(): string {
        return this.list.user;
    }
    get itemCount(): number {
        return this.list.items.filter(item => !item.complete).length;
    }
    get items(): readonly TodoItem[] {
```

```
            return this.list.items.filter(item => !item.complete);
        }
    }
```

Метод filter є стандартною функцією JavaScript для роботи з масивами. Це той самий вираз, який використовувався раніше для властивості itemCount.

Оскільки ми маємо живу модель даних, а зміни відображаються в прив'язках даних миттєво, встановлення checkbox'а для елемента item видаляє його з представлення, як показано на рис. 2-10.

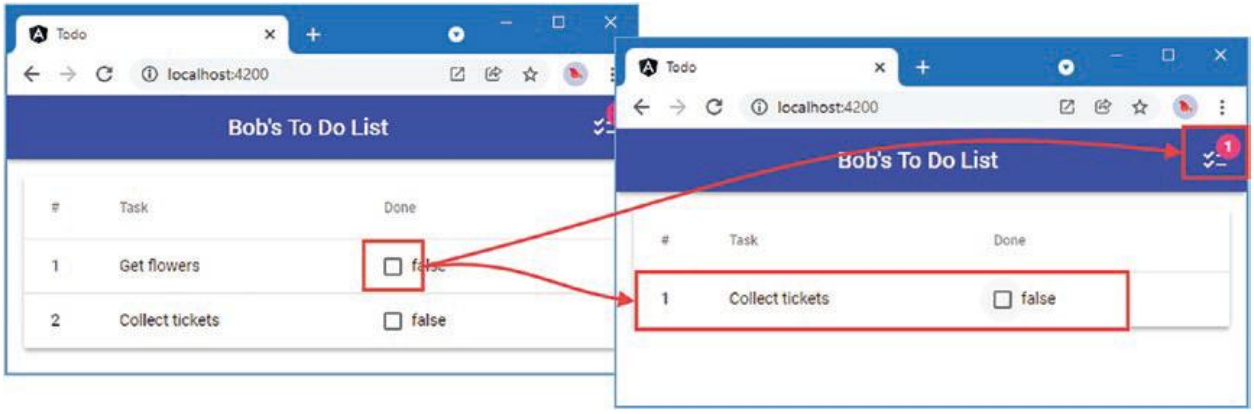

Рис. 2-10. Фільтрування завдань

Додавання завдань

Наша програма Todo не дуже корисна без можливості додавати нові елементи до списку. Лістинг 2-21 використовує Angular Material компоненти, щоб представити користувачеві елемент введення, до якого може бути внесено опис нового завдання, і кнопку, яка використовуватиметься для створення нового елемента справ.

Лістинг 2-21. Додавання елементів у файл app.component.html у папці src/app

```
<mat-toolbar color="primary" class="mat-elevation-z3">
    <span class="spacer"></span>
    <span>{{ username }}'s To Do List</span>
    <span class="spacer"></span>
    <mat-icon matBadge="{{ itemCount }}" matBadgeColor="accent">checklist</mat-icon>
</mat-toolbar>

<div class="inputContainer">
    <mat-form-field class="fullWidth">
        <mat-label style="padding-left: 5px;">Нова справа</mat-label>
        <input matInput placeholder="Enter to-do description" #todoText>
        <button matSuffix mat-raised-button color="accent" class="addButton"
        (click)="addItem(todoText.value); todoText.value = ''">Додати
        </button>
    </mat-form-field>
```

```
</div>
<div class="tableContainer">
    <table mat-table [dataSource]="items" class="mat-elevation-z3 fullWidth">
        <ng-container matColumnDef="id">
            <th mat-header-cell *matHeaderCellDef>#</th>
            <td mat-cell *matCellDef="let i = index"> {{ i + 1 }} </td>
        </ng-container>
        <ng-container matColumnDef="task">
            <th mat-header-cell *matHeaderCellDef>Task</th>
            <td mat-cell *matCellDef="let item"> {{ item.task }} </td>
        </ng-container>
        <ng-container matColumnDef="done">
            <th mat-header-cell *matHeaderCellDef>Done</th>
            <td mat-cell *matCellDef="let item">
                <mat-checkbox [(ngModel)]="item.complete" color="primary">
                  {{ item.complete }}
                </mat-checkbox>
            </td>
        </ng-container>
        <tr mat-header-row *matHeaderRowDef="['id', 'task', 'done']"></tr>
        <tr mat-row *matRowDef="let row; columns: ['id', 'task', 'done'];"></tr>
    </table>
        </div>
```

Нові елементи відображають елемент input і кнопку button. Елемент mat-form-field і *mat атрибути на інших елементах налаштовують стиль Angular Material.

Елемент input має атрибут, ім'я якого починається з символу #, який використовується для визначення змінної посилання на елемент у прив'язках даних шаблону:

```
...
<input matInput placeholder="Enter to-do description" #todoText>
...
```

Ім'я змінної todoText використовується прив'язкою і застосовується елементом button.

```
...
<button matSuffix mat-raised-button color="accent" class="addButton"
(click)="addItem(todoText.value); todoText.value = "">
...
```

Це приклад *прив'язки події*, і вона повідомляє Angular, що треба викликати метод компонента addItem, використовуючи значення елемента input як аргумент методу, а потім очистити елемент input, встановивши для його властивості значення пустого рядка '' ".

Настроювані стилі CSS потрібні для керування макетом нових елементів, як показано в лістингу 2-22.

Лістинг 2-22. Визначення стилів у файлі app.component.css у папці src/app

```
.spacer { flex: 1 1 auto }
.tableContainer { padding: 15px }
.fullWidth { ширина: 100% }
.inputContainer { margin: 15px 15px 5px }
.addButton { margin: 5px }
```

Лістинг 2-23 додає метод addItem, який викликається кнопкою button.

Лістинг 2-23. Додавання методу у файл app.component.ts у папці src/app

```
import { Component } from '@angular/core';
import { TodoList } from "./todoList";
import { TodoItem } from "./todoItem";
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {
    private list = new TodoList("Bob", [
        new TodoItem("Go for run", true),
        new TodoItem("Get flowers"),
        new TodoItem("Collect tickets"),
    ]);
    get username(): string {
        return this.list.user;
    }
    get itemCount(): number {
        return this.list.items.filter(item => !item.complete).length;
    }
    get items(): readonly TodoItem[] {
        return this.list.items.filter(item => !item.complete);
    }
    addItem(newItem: string) {
        if (newItem != "") {
            this.list.addItem(newItem);
        }
    }
}
```

Метод addItem отримує текст, надісланий прив’язкою події в шаблоні, і використовує його для додавання нового елементу до списку справ. Результатом цих змін є те, що користувач може створювати нові елементи справ, вводячи текст в елемент input та натиснувши кнопку «Add», як показано на рис. 2-11.

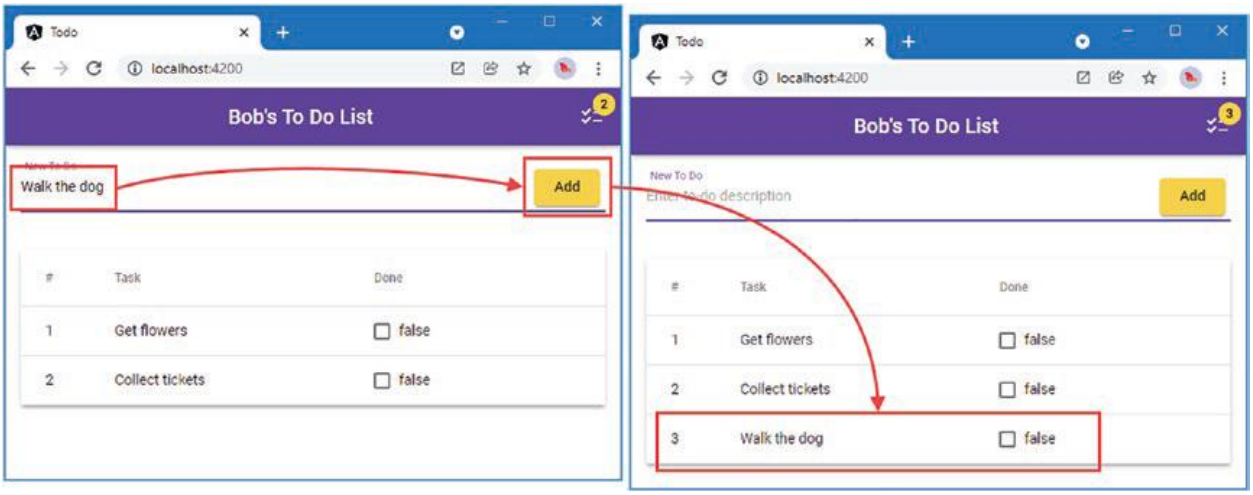

Рис. 2-11. Створення завдання

Завершення

Основні функції готові, і тепер настав час завершити проект. У лістингу 2-24 видалено текст true/false з колонки Done таблиці в шаблоні та додано опцію для показу виконаних завданнь.

Лістинг 2-24. Змінення шаблону у файлі app.component.html у папці src/app

```
<mat-toolbar color="primary" class="mat-elevation-z3">
    <span class="spacer"></span>
    <span>{{ username }}'s To Do List</span>
    <span class="spacer"></span>
    <mat-icon matBadge="{{ itemCount }}" matBadgeColor="accent">checklist</mat-icon>
</mat-toolbar>

<div class="inputContainer">
    <mat-form-field class="fullWidth">
        <mat-label style="padding-left: 5px;">New To Do</mat-label>
        <input matInput placeholder="Enter to-do description" #todoText>
        <button matSuffix mat-raised-button color="accent" class="addButton"
        (click)="addItem(todoText.value); todoText.value = ''">Add
        </button>
    </mat-form-field>
</div>

<div class="tableContainer">
    <table mat-table [dataSource]="items" class="mat-elevation-z3 fullWidth">

        <ng-container matColumnDef="id">
            <th mat-header-cell *matHeaderCellDef>#</th>
            <td mat-cell *matCellDef="let i = index"> {{ i + 1 }} </td>
        </ng-container>

        <ng-container matColumnDef="task">
```

```
            <th mat-header-cell *matHeaderCellDef>Task</th>
            <td mat-cell *matCellDef="let item"> {{ item.task }} </td>
        </ng-container>

        <ng-container matColumnDef="done">
            <th mat-header-cell *matHeaderCellDef>Done</th>
            <td mat-cell *matCellDef="let item">
                <mat-checkbox [(ngModel)]="item.complete" color="primary">
                  <!-- {{ item.complete }} -- >
                </mat-checkbox>
            </td>
        </ng-container>
        <tr mat-header-row *matHeaderRowDef="['id', 'task', 'done']"></tr>
        <tr mat-row *matRowDef="let row; columns: ['id', 'task', 'done'];"></tr>
    </table>
</div>
<div class="toggleContainer">
    <span class="spacer"></span>
    <mat-slide-toggle [(ngModel)]="showComplete">
        Show Completed Items
    </mat-slide-toggle>
    <span class="spacer"></span>
</div>
```

Нові елементи представляють перемикач, який має двосторонню прив'язку даних для властивості з назвою showComplete. Лістинг 2-25 додає визначення для властивості showComplete компонента та використовує його значення, щоб визначити, чи відображаються виконані завдання користувачеві.

Лістинг 2-25. Відображення виконаних завдань у файлі app.component.ts у папці src/app

```
import { Component } from '@angular/core';
import { TodoList } from "./todoList";
import { TodoItem } from "./todoItem";
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {
    private list = new TodoList("Bob", [
        new TodoItem("Go for run", true),
        new TodoItem("Get flowers"),
        new TodoItem("Collect tickets"),
    ]);
    get username(): string {
        return this.list.user;
    }
```

```
    get itemCount(): number {
        return this.list.items.filter(item => !item.complete).length;
    }
    get items(): readonly TodoItem[] {
    return this.list.items.filter(item => this.showComplete || !item.complete);
    }

    addItem(newItem: string) {
        if (newItem != "") {
            this.list.addItem(newItem);
        }
    }
    showComplete: boolean = false;
}
```

Для компонування перемикача потрібні додаткові стилі CSS, як показано в лістингу 2-26.

Лістинг 2-26.Додавання стилів у файл app.component.css у папці src/app

```
.spacer { flex: 1 1 auto }
.tableContainer { padding: 15px }
.fullWidth { ширина: 100% }
.inputContainer { margin: 15px 15px 5px }
.addButton { margin: 5px }
.toggleContainer { margin: 15px; display: flex }
```

У результаті користувач може вирішувати, чи бачити йому виконані завдання, як показано на рис. 2-12.

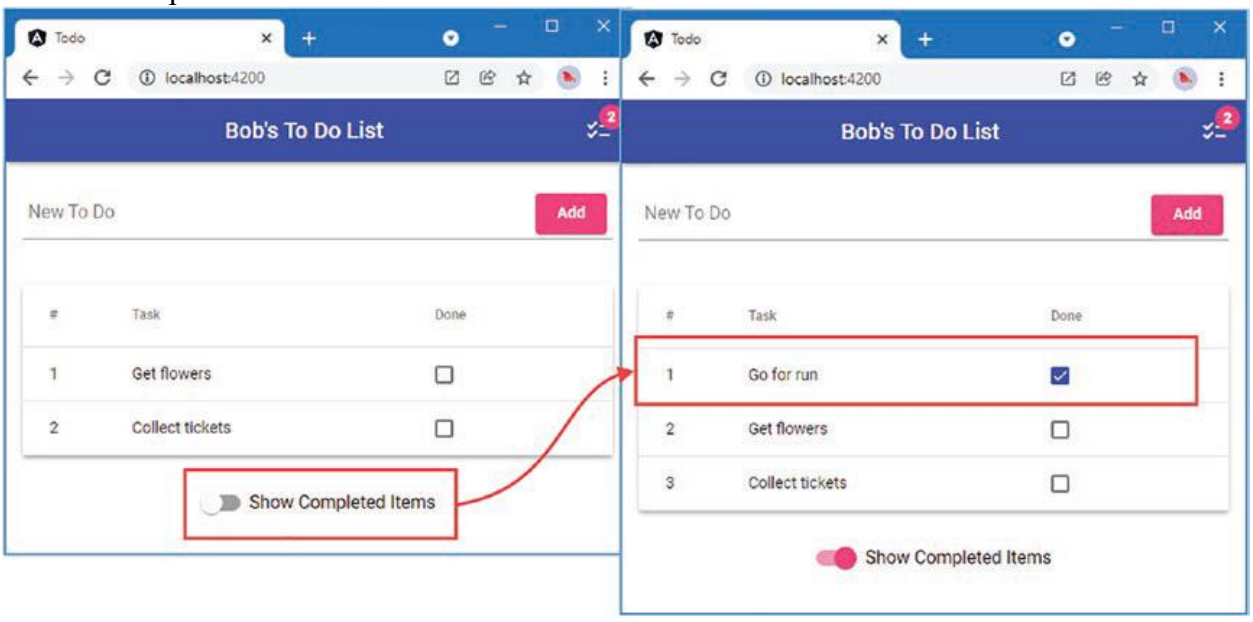

Рис. 2-12. Показ виконаних завдань

Резюме

В практичних завданнях № 4-5 було показано як створити програму Angular, яка дозволяє користувачеві створювати нові завдання та позначати існуючі як завершені. На цьому етапі важливо зрозуміти загальну форму програми Angular, яка побудована навколо моделі даних, компонентів і шаблонів.

II) Зробити звіт по роботі. Звіт повинен бути не менше 8 сторінок без титульного аркуша (шрифт Times New Roman, 14, полуторний інтервал). Титульний аркуш приводиться у додатку. Звіт повинен містити наступні розділи:

a) огляд моделі даних у проекті «Todo»;

b) пакет Angular Material: призначення, використання;

b) двосторонні прив’язки даних: призначення, використання;

c) функція filter у JavaScript: призначення, застосування, приклади;

III) Angular-додаток «Todo» розгорнути на платформі Firebase у проекті з ім’ям «ПрізвищеГрупаLaba3», наприклад «KovalenkoIP01Laba3».

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, 2023