Лабораторна робота № 2. Створення компонентів. Взаємодія між компонентами.

Мета: навчитися створювати компоненти та налагоджувати взаємодію між батьківським і дочірніми компонентами

Завдання

1) Ознайомитися з матеріалом лекції № 2;
2) Виконати пункти 1-5 ходу роботи;
3) У звіті відобразити кроки 1-5 ходу роботи;
4) Надати відповіді на контрольні питання.

Хід роботи

1) Створити два компонента:
    a) TaskListComponent (батьківський компонент) - буде містити масив тасків та відклик на подію taskDeleted від TaskItemComponent;
    b) TaskItemComponent (дочірній компонент) - буде приймати таск від батьківського компонента через @Input() і передає подію про його видалення (taskDeleted) через @Output().

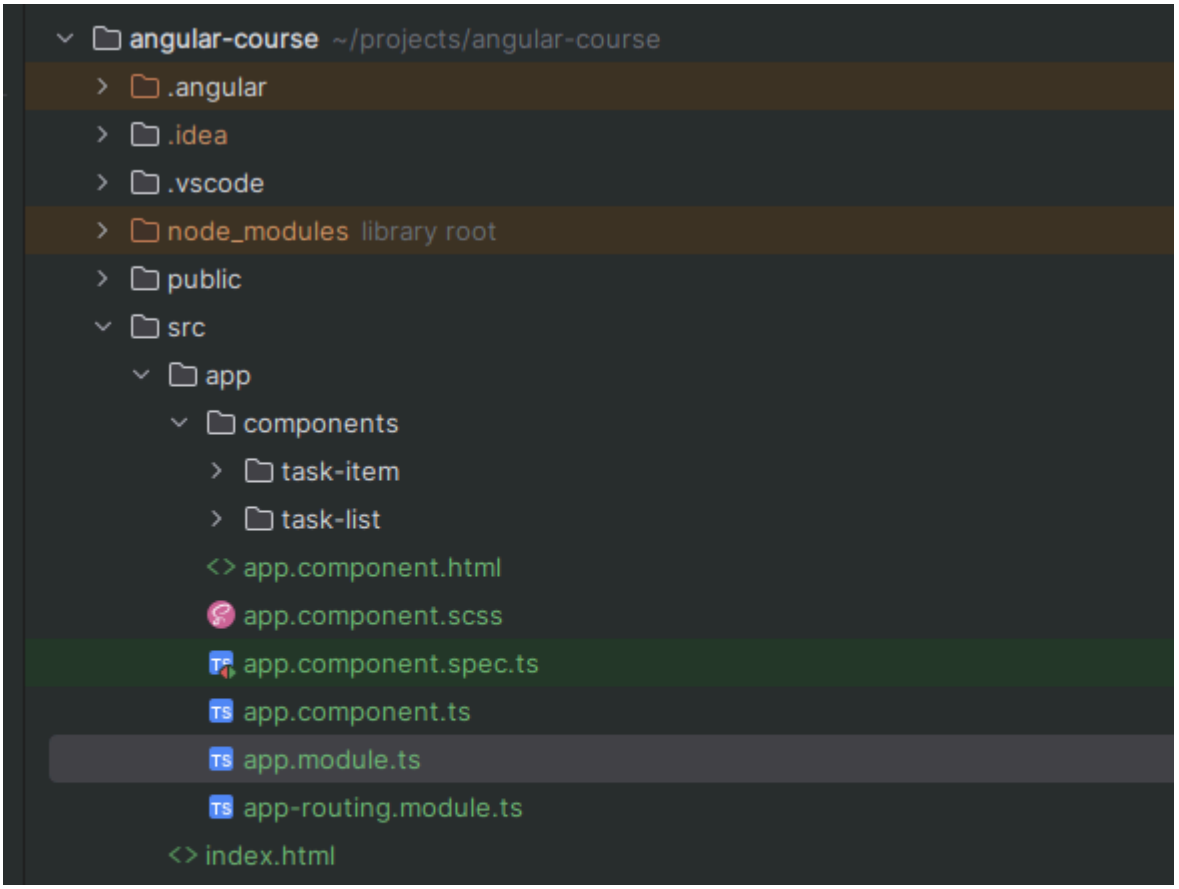
```
Terminal   Local
svarog@svarog:~/projects/angular-course$ ng g c components/task-list
CREATE src/app/components/task-list/task-list.component.scss (0 bytes)
CREATE src/app/components/task-list/task-list.component.html (24 bytes)
CREATE src/app/components/task-list/task-list.component.spec.ts (612 bytes)
CREATE src/app/components/task-list/task-list.component.ts (232 bytes)
UPDATE src/app/app.module.ts (496 bytes)
svarog@svarog:~/projects/angular-course$ ng g c components/task-item
CREATE src/app/components/task-item/task-item.component.scss (0 bytes)
CREATE src/app/components/task-item/task-item.component.html (24 bytes)
CREATE src/app/components/task-item/task-item.component.spec.ts (612 bytes)
CREATE src/app/components/task-item/task-item.component.ts (232 bytes)
UPDATE src/app/app.module.ts (599 bytes)
svarog@svarog:~/projects/angular-course$
```

Після виконання команд, фреймворк автоматично запише створені компоненти у масив declarations[] файла app.module.ts

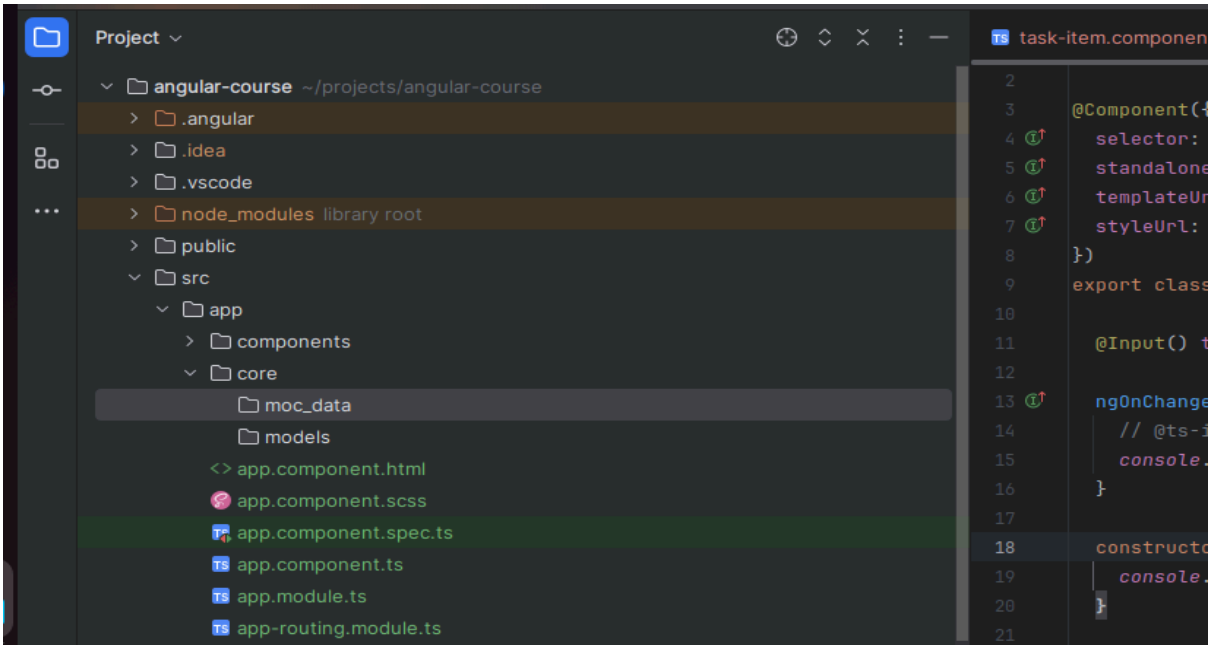
README.md app.module.ts

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { TaskListComponent } from './components/task-list/task-list.component';
import { TaskItemComponent } from './components/task-item/task-item.component';

@NgModule({  Show usages  new *
  declarations: [
    AppComponent,
    TaskListComponent,
    TaskItemComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

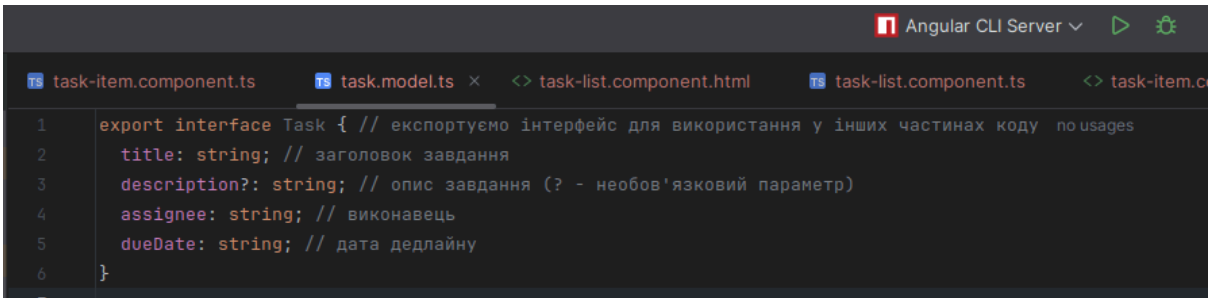
Новостворені компоненти можна знайти за шляхом your_project/src/app/components



2) Створюємо теку **core** у теці **app,** у створеній теці створюємо дві теки:
    a) models - буде зберігати інтерфейси;
    b) moc_data - буде зберігати тестові набори даних.

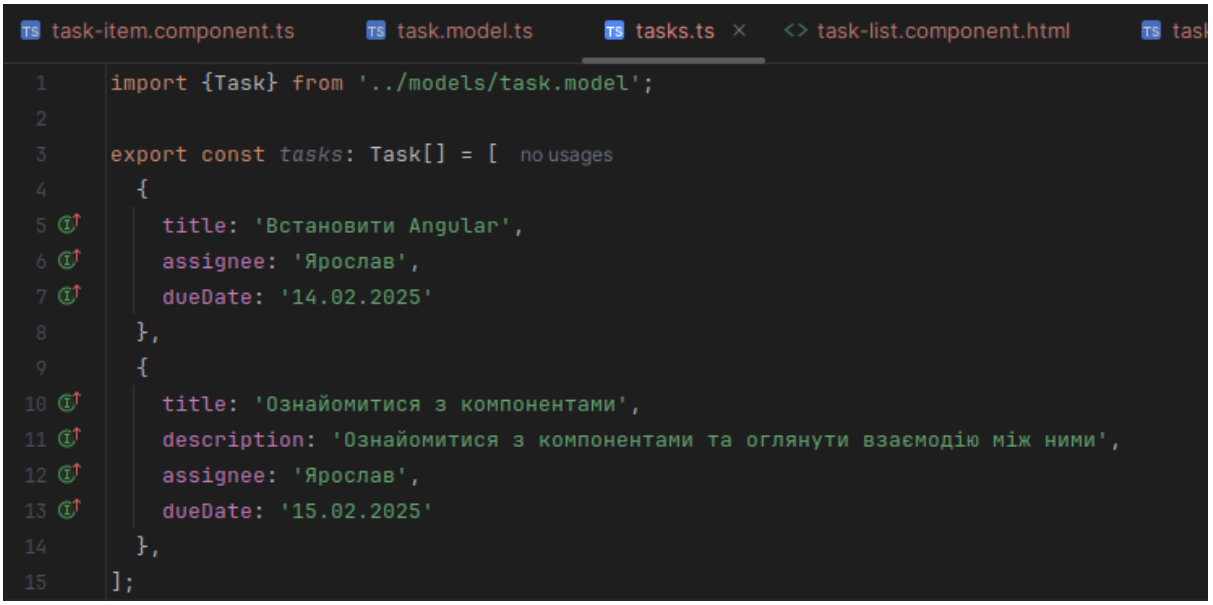


3) У теці **models** створюємо файл **task.model.ts** та прописуємо наступний код:



Цей інтерфейс ми будемо використовувати як тип даних для змінних, що містять завдання.

4) У теці **moc_data** створюємо файл **tasks.ts** з наступним вмістом:



5) Переходимо до реалізації компонентів, спочатку реалізуємо батьківський компонент, для цього виконаємо наступні дії:
   a) Замінимо стандартний html у файлі **app.component.html** на виклик тегу нашого батьківського компонента (він вказаний у властивості selector нашого файлу **task-list.component.ts**)

```
task-item.component.ts  task.model.ts  tasks.ts  task-list.component.html  task
1  import { Component } from '@angular/core';
2
3  @Component({  Show usages
4    selector: 'app-task-list',
5    standalone: false,
6    templateUrl: './task-list.component.html',
7    styleUrl: './task-list.component.scss'
8  })
9  export class TaskListComponent {
```

Прописуємо селектор у вигляді html-тегу у файлі **app.component.html:**

```
task-item.component.ts  task.model.ts  tasks.ts  task-list.component.html  task-list.component.ts  app.component.html
1  <app-task-list></app-task-list>
2
```

b) У файлі **task-list.component.ts** прописуємо наступний код:

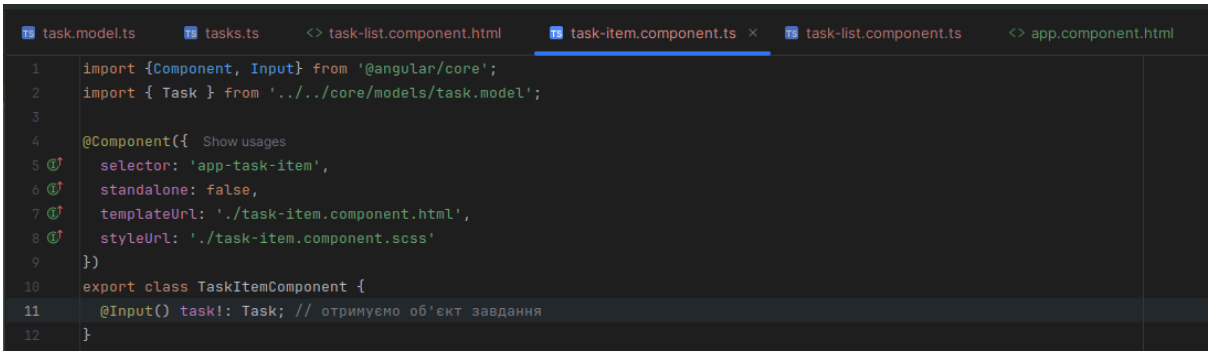
```
task-item.component.ts  task.model.ts  tasks.ts  task-list.component.html  task-list.component.ts  app.component.html
1  import { Component } from '@angular/core';
2  import { Task } from '../../core/models/task.model';
3  import {tasks} from '../../core/moc_data/tasks';
4
5  @Component({  Show usages
6    selector: 'app-task-list',
7    standalone: false,
8    templateUrl: './task-list.component.html',
9    styleUrl: './task-list.component.scss'
10 })
11 export class TaskListComponent {
12
13   myTasks: Task[] = tasks; // підключаємо тестові дані, вказавши за тип наш інтерфейс, так як записів
14                           // декілька, то вказуємо, що це буде масив
15
16 }
17
```

c) У файлі **task-list.component.html** викликаємо тег дочірнього елементу (його назва у селекті відповідного компонента) та вказуємо, що передаємо йому елемент масиву myTasks

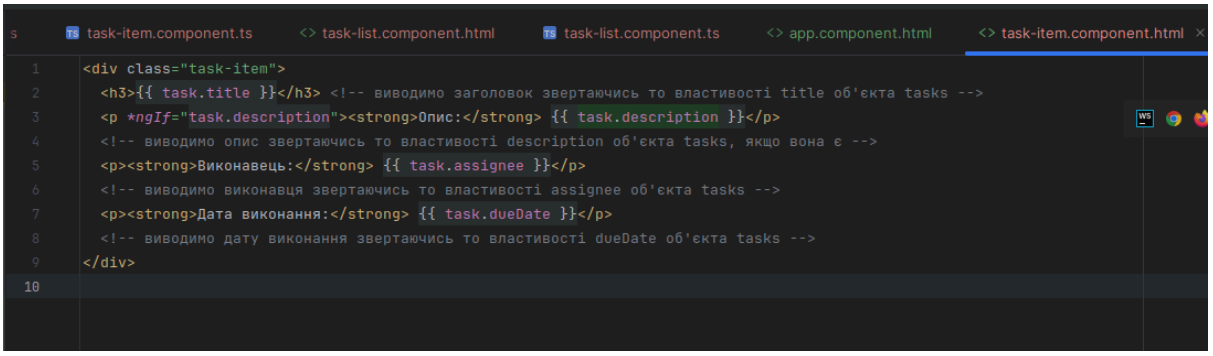
```
task.model.ts  tasks.ts  task-list.component.html  task-list.component.ts  app.component.html  task-item.component.html
1  <h2>Список завдань</h2>
2  <app-task-item
3    [task]="myTasks[0]"
4    >
5  </app-task-item>
6
```

d) У файлі **task-item.component.ts** прописуємо **@Input,** який приймає значення, яке ми передали у попередньому файлі

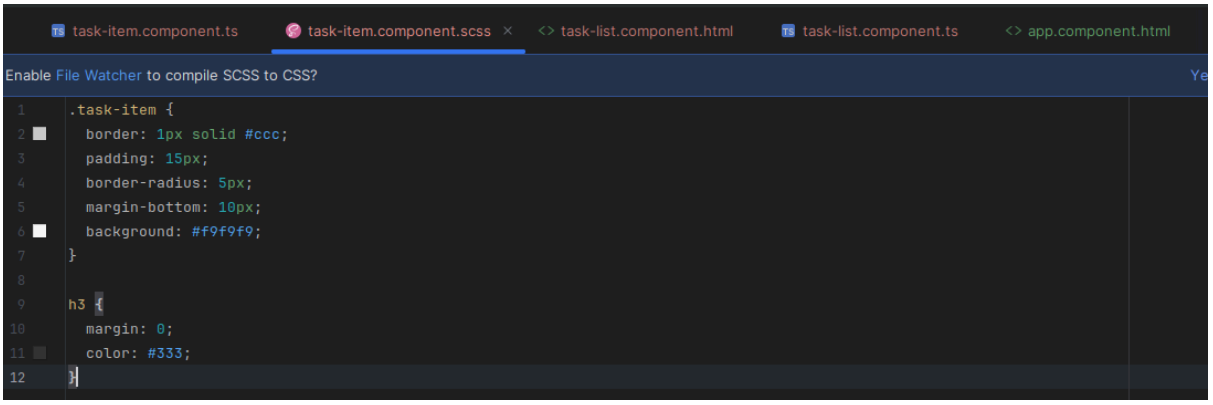
(оператор ! після task каже TS, що змінна **task** буде ініціалізована пізніше, і він не повинен вважати її потенційно **null** або **undefined**)



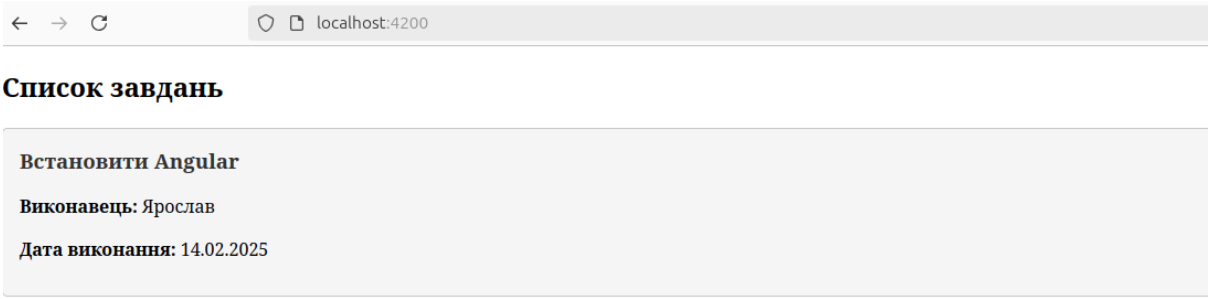
е) Далі прописуємо код, який буде відповідати за відображення у файлі **task-item.component.html**



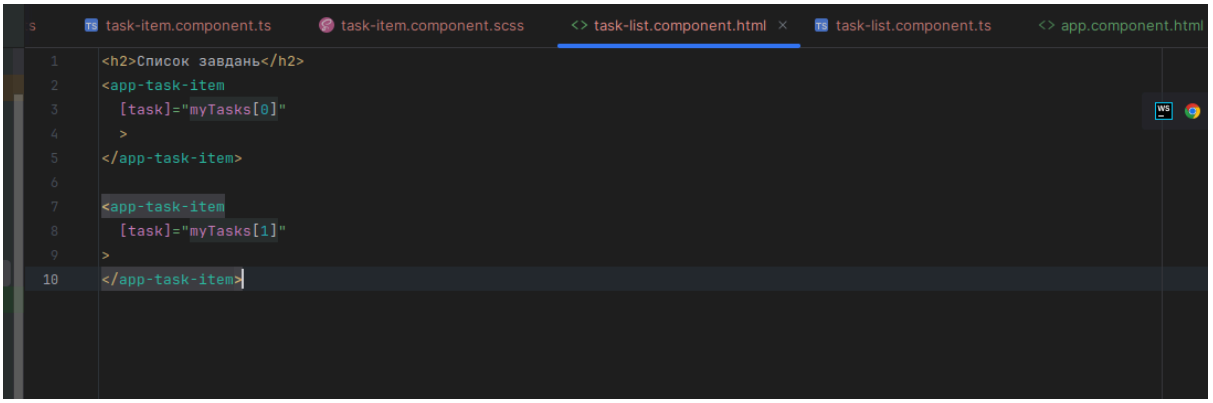
Опціонально прописуємо стилі у файлі **task-item.component.scss**



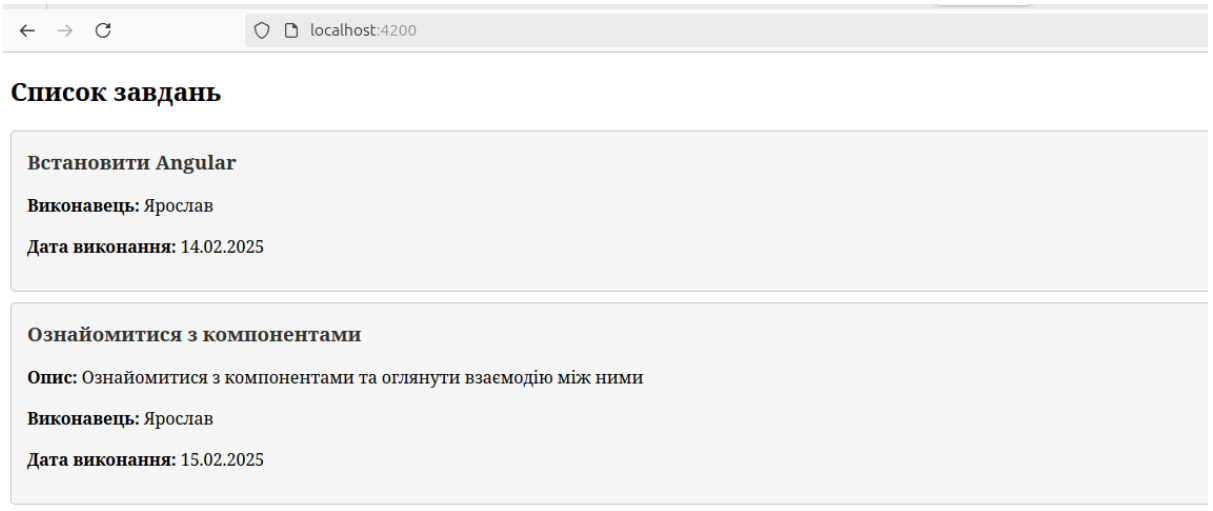
Отримали результат



Додамо ще одне завдання

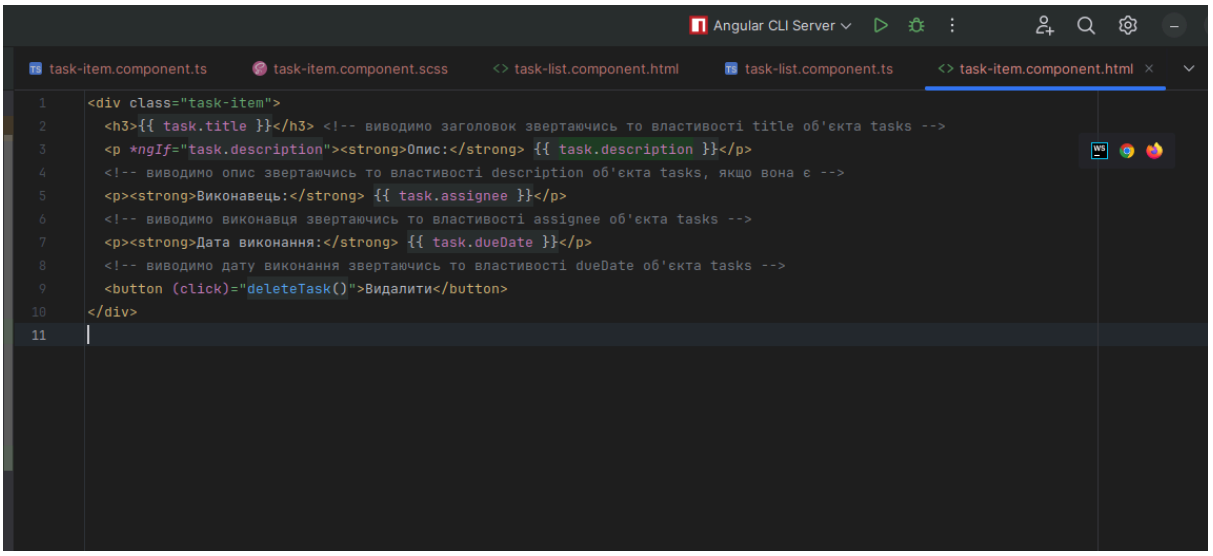


Отримуємо

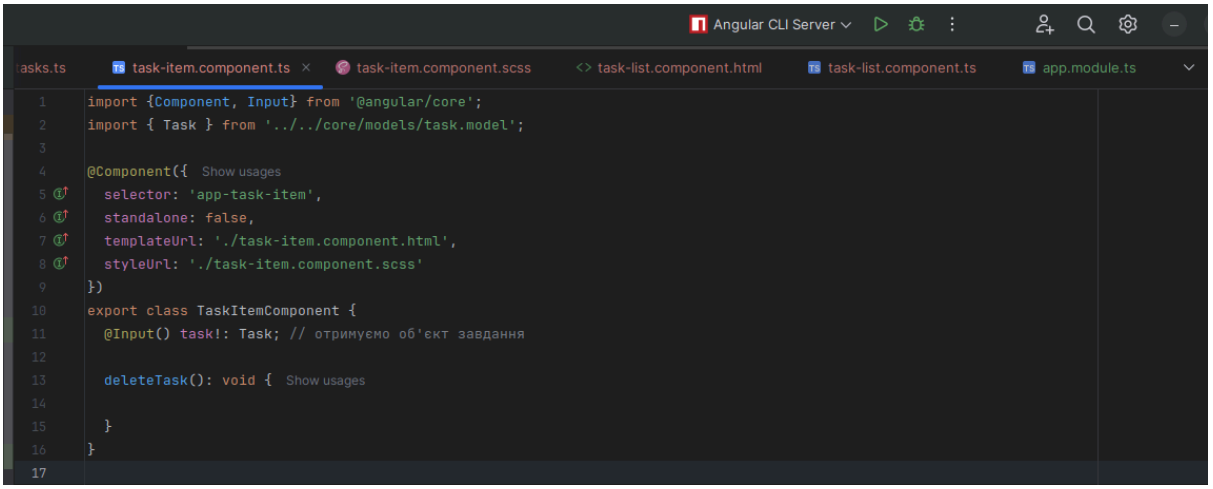


Тепер зробимо зворотній потік даних за допомогою **@Output** на прикладі видалення завдання

a) Для початку до файлу **task-item.component.html** додамо кнопку у якій пропишемо відгук на подію **click** назначивши її обробником метод **deleteTask()**



b) У файлі **task-item.component.ts** прописуємо метод **deleteTask()**



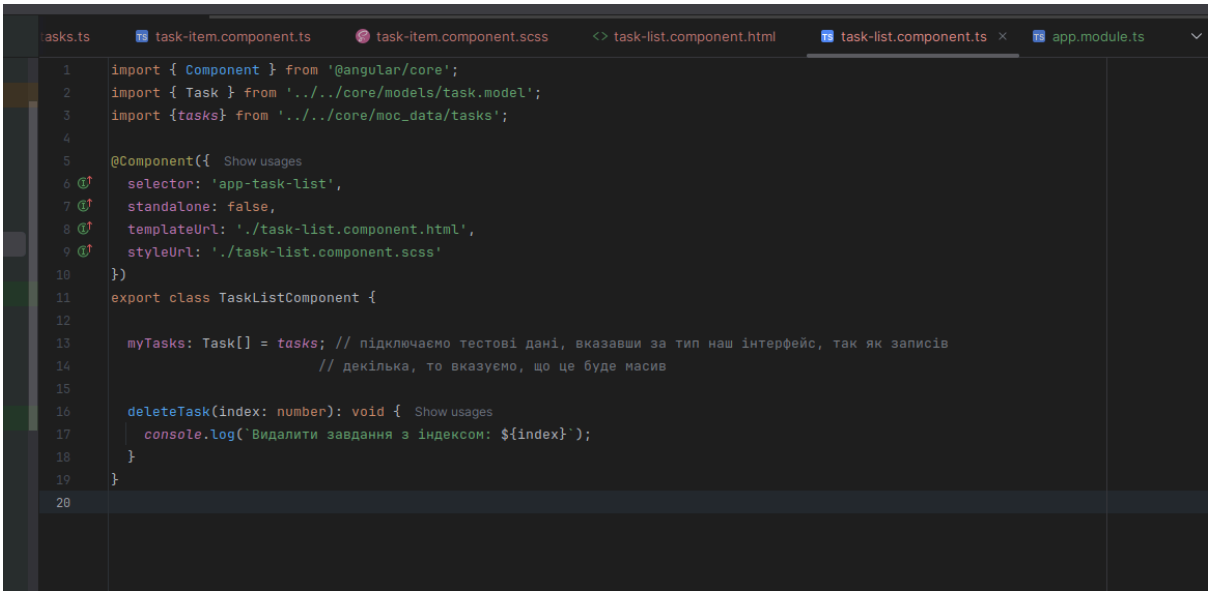
Реалізуємо його за допомогою **@Output**

```ts
import {Component, EventEmitter, Input, Output} from '@angular/core';
import { Task } from '../../core/models/task.model';

@Component({  Show usages
  selector: 'app-task-item',
  standalone: false,
  templateUrl: './task-item.component.html',
  styleUrl: './task-item.component.scss'
})
export class TaskItemComponent {
  @Input() task!: Task; // отримуємо об'єкт завдання
  @Output() taskDeleted :EventEmitter<void>  = new EventEmitter<void>(); // подія для видалення завдання

  deleteTask(): void {  Show usages
    this.taskDeleted.emit(); // викликаємо подію при натисканні кнопки "Видалити"
  }
}
```

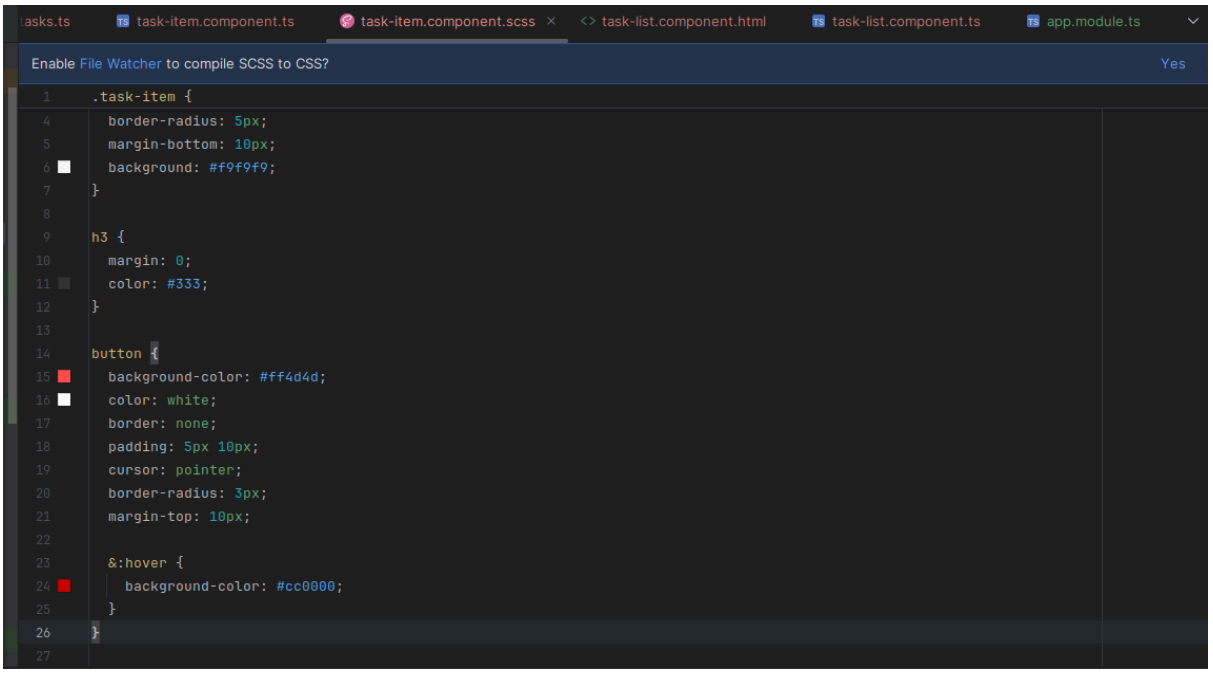
c) У файлі **task-list.component.html** прописуємо виклик кастомної події для кожного дочірнього елементу

```html
<h2>Список завдань</h2>
<app-task-item
  [task]="myTasks[0]"
  (taskDeleted)="deleteTask( number: 0)">
  >
</app-task-item>

<app-task-item
  [task]="myTasks[1]"
  (taskDeleted)="deleteTask( number: 1)">
>
</app-task-item>
```

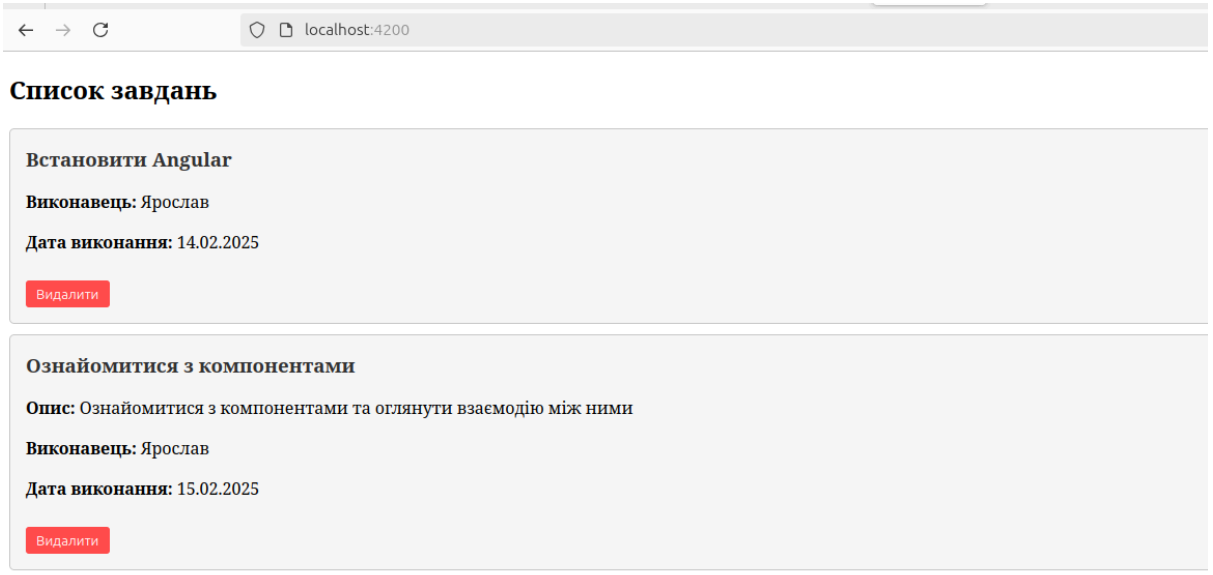
d) У файлі **task-list.component.ts** реалізуємо наш метод, який обробляє кастомну подію (емулюємо видалення)

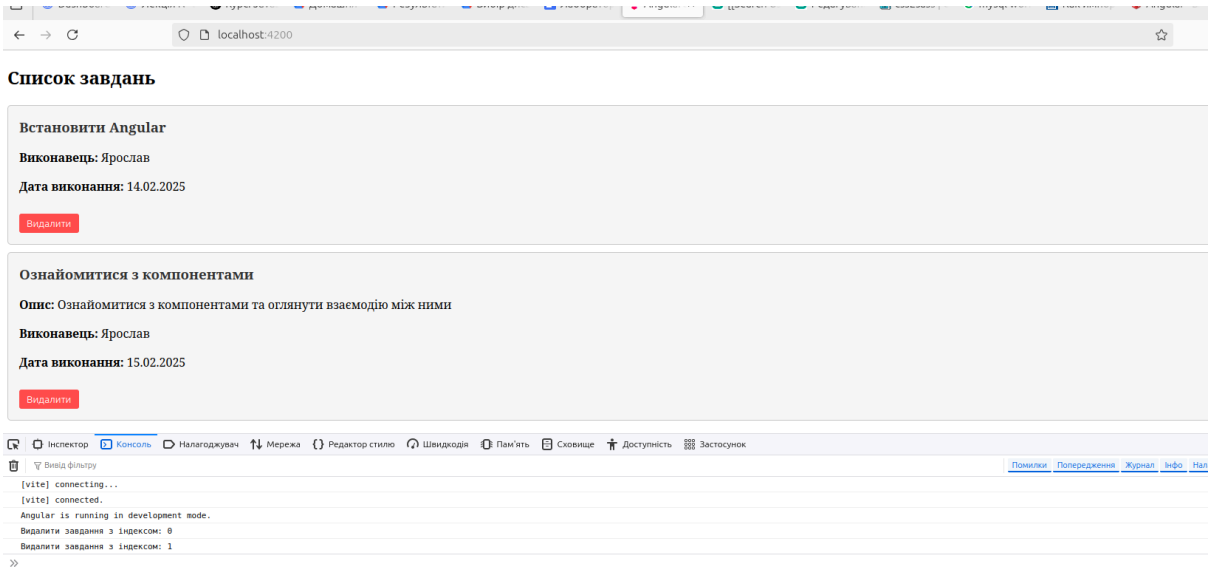


Опціонально стилізуємо кнопку видалення



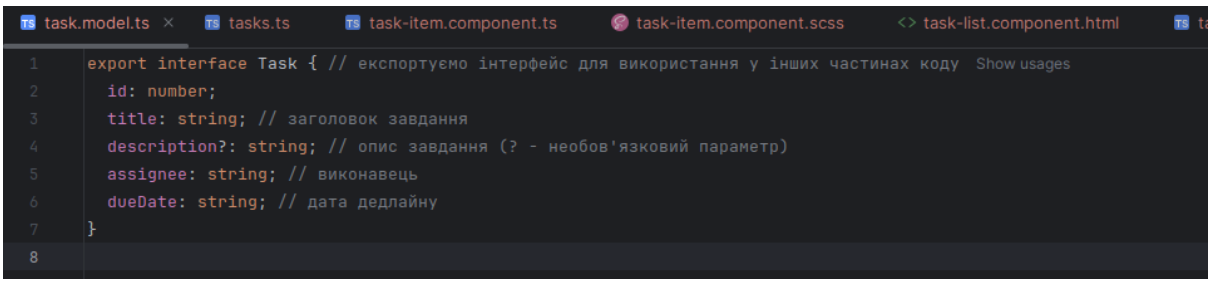
Отримуємо наступний результат



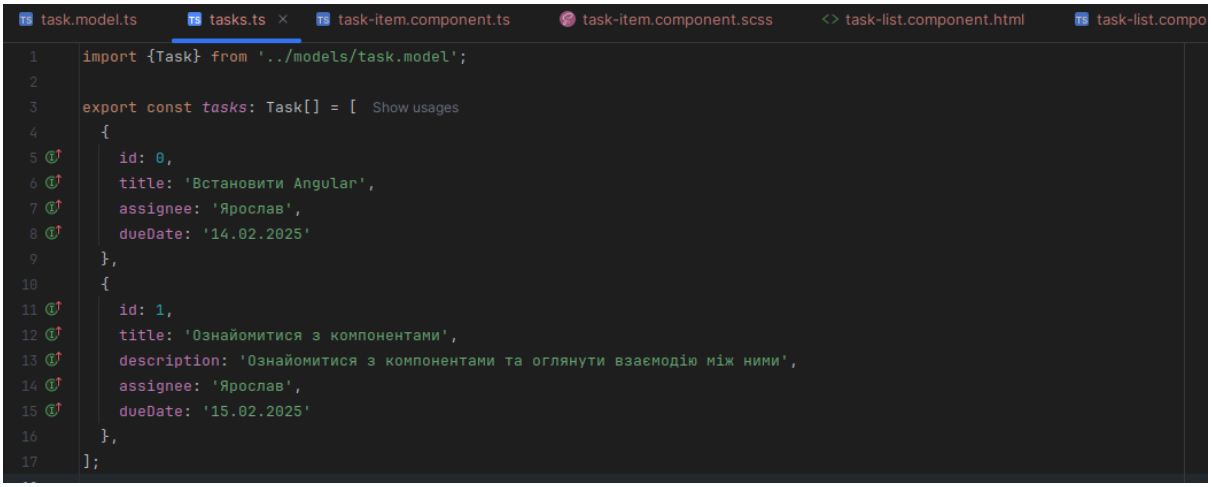


Також можна передавати відповідні параметри, при виклику кастомної події. Для цього внесемо певні корективи у наш код

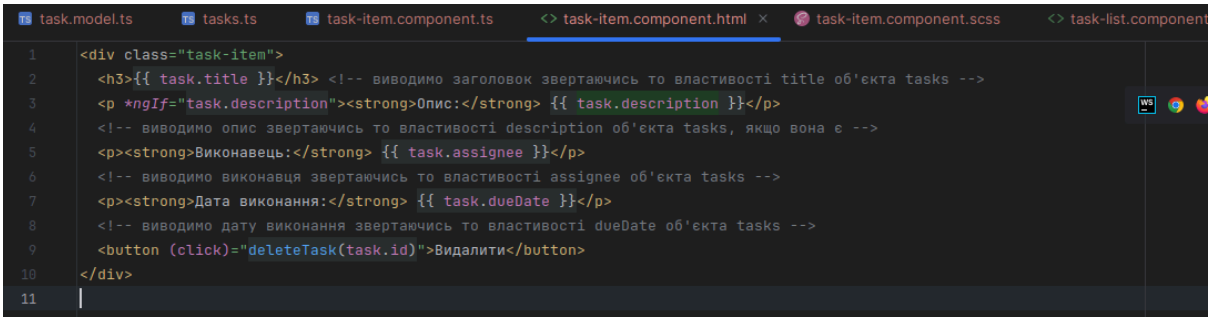
У файл **task.model.ts** додамо значення **id**



Модернізуємо файл **tasks.ts**



Пропишемо зміни у **task-item.component.html** додавши параметр **task.id** для методу



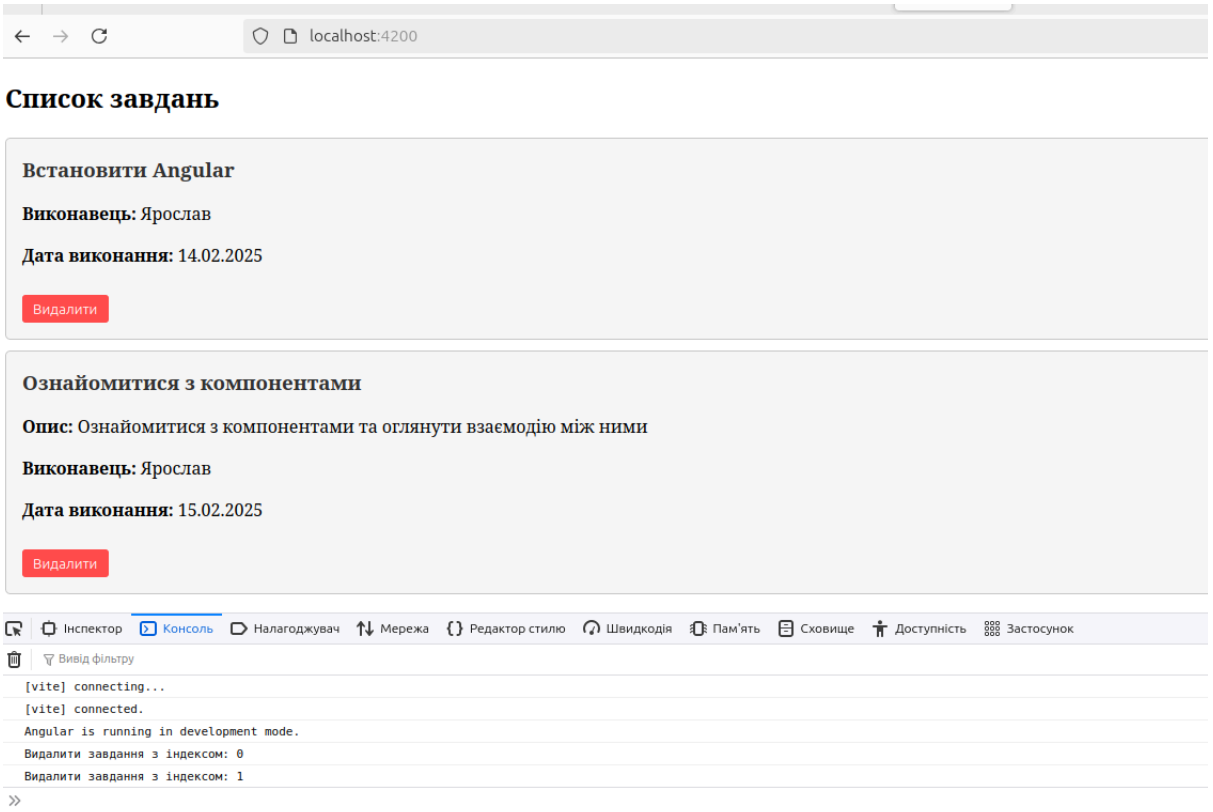
Внесемо зміни у **task-item.component.ts** змінивши тип даних для ***EventEmitter*** та передавши параметр id у ***this.taskDeleted.emit***

```ts
import {Component, EventEmitter, Input, Output} from '@angular/core';
import { Task } from '../../core/models/task.model';

@Component({  Show usages
  selector: 'app-task-item',
  standalone: false,
  templateUrl: './task-item.component.html',
  styleUrl: './task-item.component.scss'
})
export class TaskItemComponent {
  @Input() task!: Task; // отримуємо об'єкт завдання
  @Output() taskDeleted :EventEmitter<number> = new EventEmitter<number>(); // подія для видалення завдання

  deleteTask(id: number): void {  Show usages
    this.taskDeleted.emit(id); // викликаємо подію при натисканні кнопки "Видалити"
  }
}
```

Модернізуємо файл **task-list.component.html** додавши до параметрів функції **$event**

```html
<h2>Список завдань</h2>
<app-task-item
  [task]="myTasks[0]"
  (taskDeleted)="deleteTask($event)">
  >
</app-task-item>

<app-task-item
  [task]="myTasks[1]"
  (taskDeleted)="deleteTask($event)">
>
</app-task-item>
```

Перевіряємо роботу застосунку



Контрольні питання

1) Як створити новий компонент?
2) Для чого використовується @Input?
3) Для чого використовується @Output?
4) Яка властивість відповідає за унікальний ідентифікатор компонента, який дозволяє використовувати його на інших сторінках?
5) Як вивести змінну у шаблоні?