

Лабораторна робота № 3. Control Flow та pipes.

Мета: навчитися використовувати `@if`, `@for`, `@switch`, а також створювати власні pipes.

Завдання

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

Хід роботи

- 1) Для початку розширимо інтерфейс завдання додавши до нього нову властивість “status”, для цього виконаємо наступні кроки:
 - а) Створимо файл **core/models/status.enum.ts** і пропишемо у ньому наступні статуси:

```
export enum TaskStatus {  Show usages
  TODO = 'До роботи',
  IN_PROGRESS = 'У процесі',
  DONE = 'Виконано'
}
```

- б) Імпортуємо цей enum у файл інтерфейсу завдання

```
import { TaskStatus } from './status.enum';

export interface Task { // експортуємо інтерфейс для використання у інших частинах коду  Show usages  ± Kryvyi Yaroslav *
  id: number;
  title: string; // заголовок завдання
  description?: string; // опис завдання (? - необов'язковий параметр)
  assignee: string; // виконавець
  dueDate: string; // дата дедлайну
  status: TaskStatus; // статус завдання
}
```

- в) Редагуємо тестові дані

```

import {Task} from '../models/task.model';
import {TaskStatus} from '../models/status.enum';

export const tasks: Task[] = [ Show usages  ± Kryvyi Yaroslav *
{
  id: 0,
  title: 'Встановити Angular',
  assignee: 'Ярослав',
  dueDate: '14.02.2025',
  status: TaskStatus.DONE
},
{
  id: 1,
  title: 'Ознайомитися з компонентами',
  description: 'Ознайомитися з компонентами та оглянути взаємодію між ними',
  assignee: 'Ярослав',
  dueDate: '15.02.2025',
  status: TaskStatus.IN_PROGRESS
},
{
  id: 2,
  title: 'Ознайомитися з Control Flow',
  description: 'Ознайомитися з старим і новим підходом до Control Flow',
  assignee: 'Ярослав',
  dueDate: '25.02.2025',
  status: TaskStatus.DONE
}
]

```

2) Далі необхідно переписати шаблони **task-list.component.ts** та **task-item.component.ts**, почнемо з першого.

Як можна побачити, зараз для відображення нового завдання необхідно викликати тег `<app-task-item>` та передати у нього відповідні аргументи.

```

s task-item.component.ts task-item.component.scss task-list.component.html x task-list.component.ts <> app.component.html
1 <h2>Список завдань</h2>
2 <app-task-item
3   [task]="myTasks[0]"
4   >
5 </app-task-item>
6
7 <app-task-item
8   [task]="myTasks[1]"
9   >
10 </app-task-item>

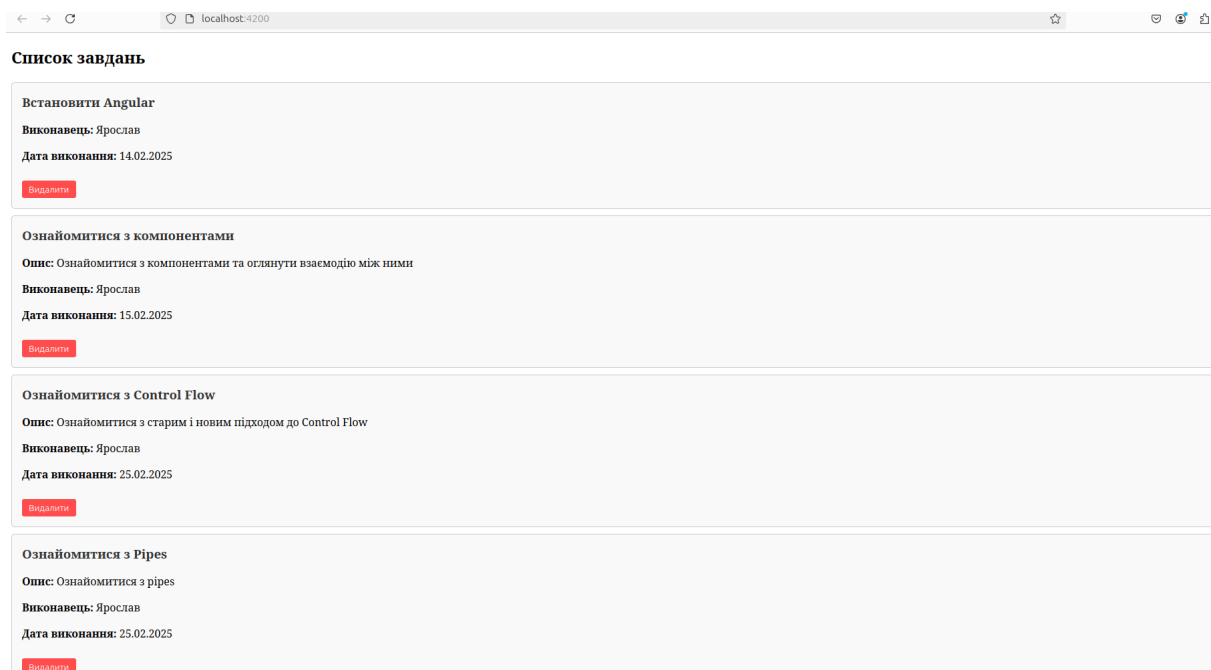
```

Це незручно тим, що список завдань може бути великий, і прописувати виклик під кожне буде займати багато часу, шаблон буде містити багато строк коду, стане незручним для редагування, а також змушує при кожній зміні списку завдань вносити зміни вручну. Для автоматичної генерації карток завдань на основі масиву даних можна

використовувати цикли `*ngFor` або `@for`. Перепишемо наш код використовуючи цикл `@for`

```
<h2>Список завдань</h2>
@for(task of myTasks; track task.id; let index = $index) {
  <app-task-item
    [task]="task"
    (taskDeleted)="deleteTask($event)"
  >
</app-task-item>
} @empty {
  <p>Курси відсутні</p>
}
```

Ми розклали колекцію `myTasks` на окремі елементи (завдання), та передали кожне із них у вигляді `@Input` для дочірнього елемента. Також можна побачити вираз `track task.id`, він допомагає Angular не рендерити увесь список завдань при додаванні або вилученні завдання, а орієнтуючись на унікальну властивість рендерити або прибирати тільки конкретний елемент. Перевіримо, що з переходом на новий синтаксис не змінилося відображення списку завдань.



Як бачимо відображення не змінилося, але тепер ми можемо вивести всі елементи колекції за раз.

Перейдемо до файлу `task-item.component.ts`, та ознайомимося з його поточним вмістом.

```
Angular CLI Server
task-item.component.ts task-item.component.scss task-list.component.html task-list.component.ts task-item.component.html x
1 <div class="task-item">
2   <h3>{{ task.title }}</h3> <!-- виводимо заголовок звертаючись то властивості title об'єкта tasks -->
3   <p *ngIf="task.description"><strong>Опис:</strong> {{ task.description }}</p>
4   <!-- виводимо опис звертаючись то властивості description об'єкта tasks, якщо вона є -->
5   <p><strong>Виконавець:</strong> {{ task.assignee }}</p>
6   <!-- виводимо виконавця звертаючись то властивості assignee об'єкта tasks -->
7   <p><strong>Дата виконання:</strong> {{ task.dueDate }}</p>
8   <!-- виводимо дату виконання звертаючись то властивості dueDate об'єкта tasks -->
9   <button (click)="deleteTask()">Видалити</button>
10 </div>
11
```

Можемо побачити вираз ***ngIf="task.description"** у 3 строці, який перевіряє, чи є поточного завдання опис, і якщо є, то відображає його. Змінимо ***ngIf** на **@if**

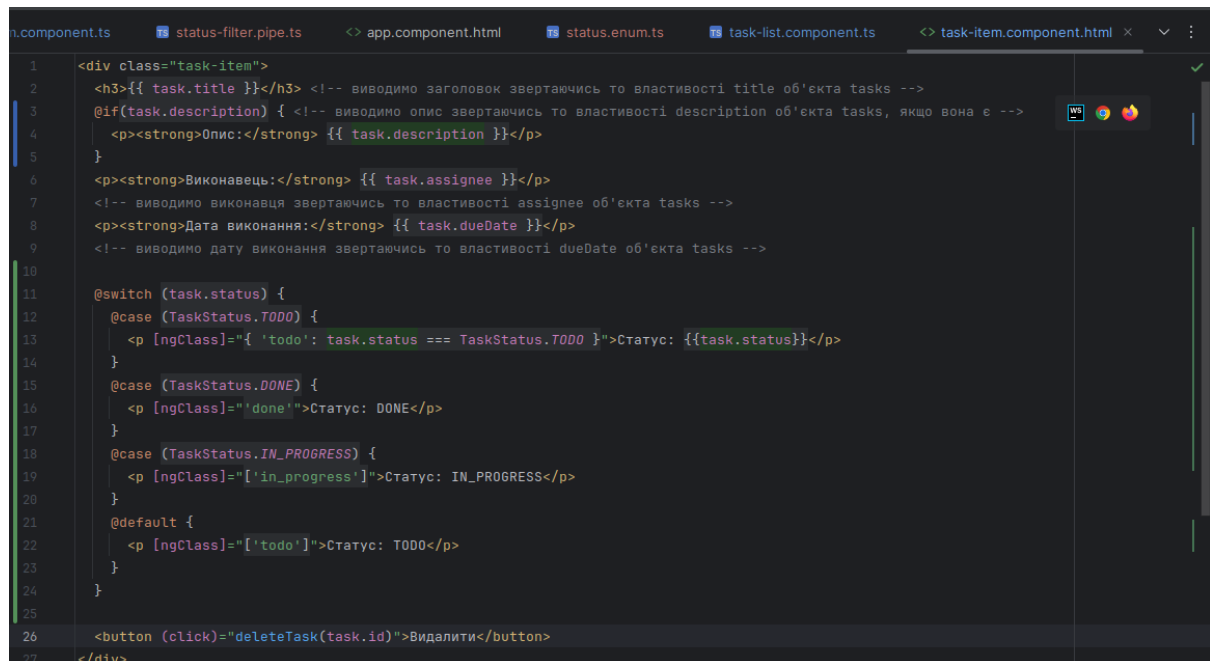
```
<div class="task-item">
  <h3>{{ task.title }}</h3> <!-- виводимо заголовок звертаючись то властивості title об'єкта tasks -->
  @if(task.description) { <!-- виводимо опис звертаючись то властивості description об'єкта tasks, якщо вона є -->
    <p><strong>Опис:</strong> {{ task.description }}</p>
  }
  <p><strong>Виконавець:</strong> {{ task.assignee }}</p>
  <!-- виводимо виконавця звертаючись то властивості assignee об'єкта tasks -->
  <p><strong>Дата виконання:</strong> {{ task.dueDate }}</p>
  <!-- виводимо дату виконання звертаючись то властивості dueDate об'єкта tasks -->
  <button (click)="deleteTask(task.id)">Видалити</button>
</div>
```

Перевіряємо відображення

Список завдань
Встановити Angular Виконавець: Ярослав Дата виконання: 14.02.2025 Видалити
Ознайомитися з компонентами Опис: Ознайомитися з компонентами та оглянути взаємодію між ними Виконавець: Ярослав Дата виконання: 15.02.2025 Видалити
Ознайомитися з Control Flow Опис: Ознайомитися з старим і новим підходом до Control Flow Виконавець: Ярослав Дата виконання: 25.02.2025 Видалити
Ознайомитися з Pipes Опис: Ознайомитися з pipes Виконавець: Ярослав Дата виконання: 25.02.2025 Видалити

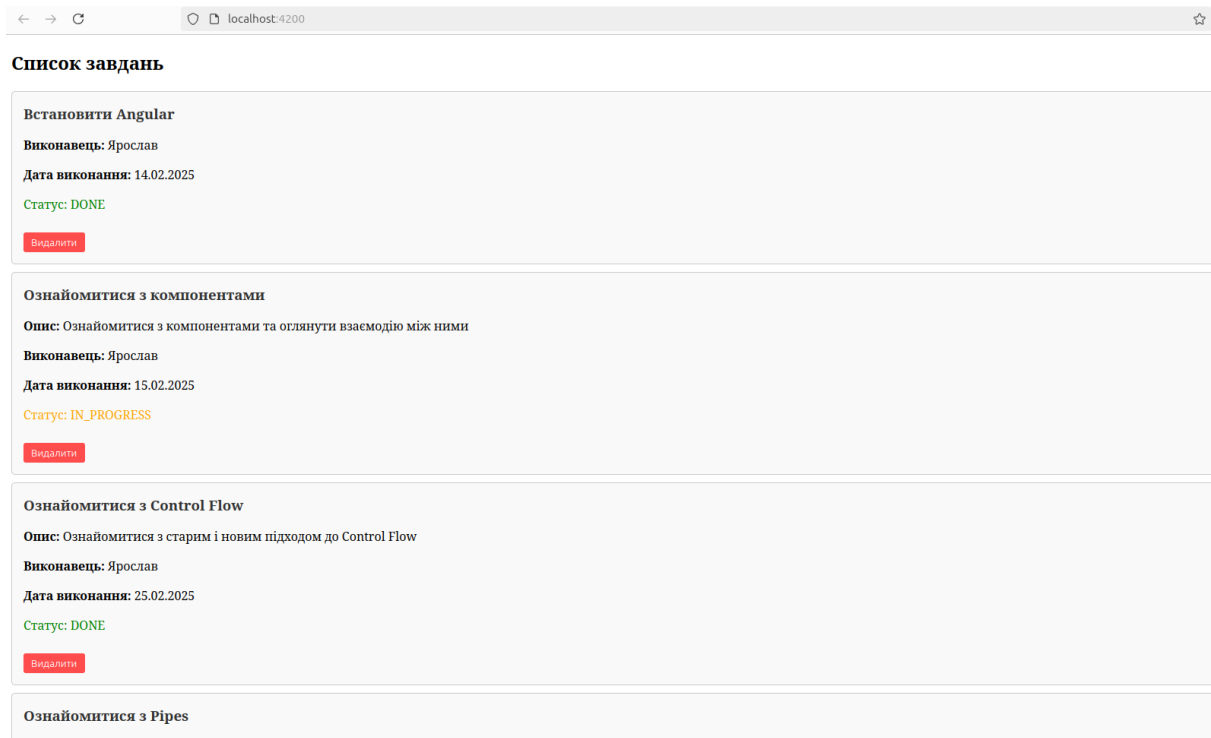
Можемо побачити, що відображення працює коректно і обробляє випадки, коли відсутній опис. Також у завдання з'явилася нова властивість. Її можна відобразити декількома способами, давайте їх розглянемо.

Спочатку використаємо `@switch`, ідея полягає у тому, що ми отримуємо на вхід статус поточно завдання, тоді як `@case` будуть підв'язані під список завдань зі створеного `enum`

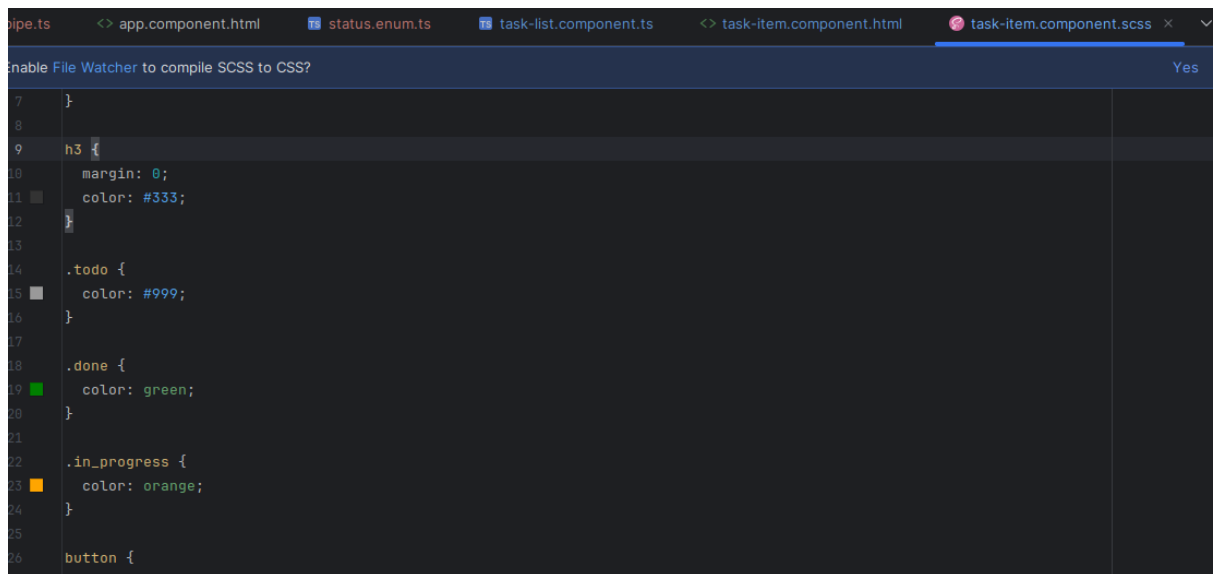


```
1 <div class="task-item">
2   <h3>{{ task.title }}</h3> <!-- виводимо заголовок звертаючись до властивості title об'єкта tasks -->
3   @if(task.description) { <!-- виводимо опис звертаючись до властивості description об'єкта tasks, якщо вона є -->
4     <p><strong>Опис:</strong> {{ task.description }}</p>
5   }
6   <p><strong>Виконавець:</strong> {{ task.assignee }}</p>
7   <!-- виводимо виконавця звертаючись до властивості assignee об'єкта tasks -->
8   <p><strong>Дата виконання:</strong> {{ task.dueDate }}</p>
9   <!-- виводимо дату виконання звертаючись до властивості dueDate об'єкта tasks -->
10
11   @switch (task.status) {
12     @case (TaskStatus.TODO) {
13       <p [ngClass]="{'todo': task.status === TaskStatus.TODO}">Статус: {{task.status}}</p>
14     }
15     @case (TaskStatus.DONE) {
16       <p [ngClass]="{'done'}">Статус: DONE</p>
17     }
18     @case (TaskStatus.IN_PROGRESS) {
19       <p [ngClass]="{'in_progress'}">Статус: IN_PROGRESS</p>
20     }
21     @default {
22       <p [ngClass]="{'todo'}">Статус: TODO</p>
23     }
24   }
25
26   <button (click)="deleteTask(task.id)">Видалити</button>
27 </div>
```

Перевіримо відображення на сторінці



Можемо побачити, що у картці з'явилося нове поле статусу, також ми додали динамічні стилі через [ngClass]



Розглянемо другий спосіб, пропишемо властивість статусу, як і інші у картці завдання

```
<div class="task-item">
  <h3>{{ task.title }}</h3> <!-- виводимо заголовок звертаючись то властивості title об'єкта tasks -->
  @if(task.description) { <!-- виводимо опис звертаючись то властивості description об'єкта tasks, якщо вона є -->
    <p><strong>Опис:</strong> {{ task.description }}</p>
  }
  <!-- виводимо виконавця звертаючись то властивості assignee об'єкта tasks -->
  <p><strong>Виконавець:</strong> {{ task.assignee }}</p>
  <!-- виводимо дату виконання звертаючись то властивості dueDate об'єкта tasks -->
  <p><strong>Дата виконання:</strong> {{ task.dueDate }}</p>
  <!-- виводимо статус звертаючись то властивості status об'єкта tasks -->
  <p><strong>Статус:</strong> <span>{{ task.status }}</span></p>

  <button (click)="deleteTask(task.id)">Видалити</button>
</div>
```

Перевіримо відображення

Виконавець: Ярослав
Дата виконання: 14.02.2025
Статус: Виконано
Видалити

Ознайомитися з компонентами
Опис: Ознайомитися з компонентами та оглянути взаємодію між ними
Виконавець: Ярослав
Дата виконання: 15.02.2025
Статус: У процесі
Видалити

Ознайомитися з Control Flow
Опис: Ознайомитися з старим і новим підходом до Control Flow
Виконавець: Ярослав
Дата виконання: 25.02.2025
Статус: Виконано
Видалити

Ознайомитися з Pipes
Опис: Ознайомитися з pipes
Виконавець: Ярослав
Дата виконання: 25.02.2025
Статус: До роботи

Бачимо, що статус виводиться згідно enum, але ми втратили стилі. Для того щоб їх повернути, використаємо [ngClass], передавши йому, як об'єкт, наші класи

```
Angular CLI Server
app.component.html status.enum.ts task-list.component.ts task-item.component.html x task-item.component.scss
<div class="task-item">
  <h3>{{ task.title }}</h3> <!-- виводимо заголовок звертаючись то властивості title об'єкта tasks -->
  @if(task.description) { <!-- виводимо опис звертаючись то властивості description об'єкта tasks, якщо вона є -->
    <p><strong>Опис:</strong> {{ task.description }}</p>
  }
  <!-- виводимо виконавця звертаючись то властивості assignee об'єкта tasks -->
  <p><strong>Виконавець:</strong> {{ task.assignee }}</p>
  <!-- виводимо дату виконання звертаючись то властивості dueDate об'єкта tasks -->
  <p><strong>Дата виконання:</strong> {{ task.dueDate }}</p>
  <!-- виводимо статус звертаючись то властивості status об'єкта tasks -->
  <p><strong>Статус:</strong>
    <span
      [ngClass]="{
        'done' : task.status === TaskStatus.DONE,
        'todo' : task.status === TaskStatus.TODO,
        'in-progress' : task.status === TaskStatus.IN_PROGRESS
      }"
    >{{ task.status }}
    </span>
  </p>
  <button (click)="deleteTask(task.id)">Видалити</button>
</div>
```

Перевіримо відображення

Дата виконання: 14.02.2025
Статус: Виконано
Видалити

Ознайомитися з компонентами
Опис: Ознайомитися з компонентами та оглянути взаємодію між ними
Виконавець: Ярослав
Дата виконання: 15.02.2025
Статус: У процесі
Видалити

Ознайомитися з Control Flow
Опис: Ознайомитися з старим і новим підходом до Control Flow
Виконавець: Ярослав
Дата виконання: 25.02.2025
Статус: Виконано
Видалити

Ознайомитися з Pipes
Опис: Ознайомитися з pipes
Виконавець: Ярослав
Дата виконання: 25.02.2025
Статус: До роботи
Видалити

Бачимо, що тепер стилі коректно працюють залежно від статусу. Але що робити, якщо статусів буде 10-15 або потрібно буде цю логіку використовувати повторно у шаблоні. Можна звісно прописувати все вручну, але стикаємося з тією же проблемою, що була до використання цикла `@for` - незручність та надлишковість. Щоб це виправити, ми можемо повертати класи стилів за допомогою функції, давайте це реалізуємо. У

task-item.component.ts створимо нову функцію **getStatusClasses()**, яка буде повертати об'єкт з CSS-класами

```
10  })
11  export class TaskItemComponent {
12
13    @Input() task!: Task; // отримуємо об'єкт завдання
14    @Output() taskDeleted: EventEmitter<number> = new EventEmitter<number>(); // подія для видалення завдання
15
16    protected readonly TaskStatus :typeof TaskStatus = TaskStatus;
17
18    deleteTask(id: number): void { Show usages  Kryvyi Yaroslav
19      this.taskDeleted.emit(id); // викликаємо подію при натисканні кнопки "Видалити"
20    }
21
22    getStatusClasses(): {done: boolean; todo: boolean; 'in-progr...} { no usages  new *
23      return {
24        'done': this.task.status === TaskStatus.DONE,
25        'todo': this.task.status === TaskStatus.TODO,
26        'in-progress': this.task.status === TaskStatus.IN_PROGRESS
27      };
28    }
29  }
30  }
```

У **task-item.component.html** змінимо об'єкт у **[ngClass]** на виклик функції

```
1  <div class="task-item">
2    <h3>{{ task.title }}</h3> <!-- виводимо заголовок звертаючись то властивості title об'єкта tasks -->
3    @if(task.description) { <!-- виводимо опис звертаючись то властивості description об'єкта tasks, якщо вона є -->
4      <p><strong>Опис:</strong> {{ task.description }}</p>
5    }
6    <!-- виводимо виконавця звертаючись то властивості assignee об'єкта tasks -->
7    <p><strong>Виконавець:</strong> {{ task.assignee }}</p>
8    <!-- виводимо дату виконання звертаючись то властивості dueDate об'єкта tasks -->
9    <p><strong>Дата виконання:</strong> {{ task.dueDate }}</p>
10   <!-- виводимо статус звертаючись то властивості status об'єкта tasks -->
11   <p><strong>Статус:</strong> <span [ngClass]="getStatusClasses()">{{ task.status }}</span>
12   </p>
13
14   <button (click)="deleteTask(task.id)">Видалити</button>
15 </div>
```

Перевіряємо відображення

Виконавець: Ярослав Дата виконання: 14.02.2025 Статус: Виконано Видалити
Ознайомитися з компонентами Опис: Ознайомитися з компонентами та оглянути взаємодію між ними Виконавець: Ярослав Дата виконання: 15.02.2025 Статус: У процесі Видалити
Ознайомитися з Control Flow Опис: Ознайомитися з старим і новим підходом до Control Flow Виконавець: Ярослав Дата виконання: 25.02.2025 Статус: Виконано Видалити
Ознайомитися з Pipes Опис: Ознайомитися з pipes Виконавець: Ярослав Дата виконання: 25.02.2025 Статус: До роботи

Можемо побачити, що стилі працюють, як і у випадку прямої передачі, але тепер код у шаблоні став більш читабельним та з'явилася можливість повторно використовувати класи стилів.

Що робити, якщо завдання було виконане або потрібно просто змінити статус, для цього внесемо корективи у **task-item.component.html**, додавши можливість змінювати статус кожного елементу окремо. Для цього використаємо **select** та івент (**change**)

```

1  <div class="task-item">
2    <h3>{{ task.title }}</h3> <!-- виводимо заголовок звертаючись то властивості title об'єкта tasks -->
3    @if(task.description) { <!-- виводимо опис звертаючись то властивості description об'єкта tasks, якщо вона є -->
4      <p><strong>Опис:</strong> {{ task.description }}</p>
5    }
6    <!-- виводимо виконавця звертаючись то властивості assignee об'єкта tasks -->
7    <p><strong>Виконавець:</strong> {{ task.assignee }}</p>
8    <!-- виводимо дату виконання звертаючись то властивості dueDate об'єкта tasks -->
9    <p><strong>Дата виконання:</strong> {{ task.dueDate }}</p>
10   <!-- виводимо статус звертаючись то властивості status об'єкта tasks -->
11   <p><strong>Статус:</strong> <span [ngClass]="getStatusClasses()">{{ task.status }}</span>
12 </p>
13
14   <div class="select-status">
15     <select (change)="updateStatus($event)">
16       <option [value]="TaskStatus.TODO">🔴 До роботи</option>
17       <option [value]="TaskStatus.IN_PROGRESS">🟡 В процесі</option>
18       <option [value]="TaskStatus.DONE">✅ Виконано</option>
19     </select>
20   </div>
21
22
23   <button (click)="deleteTask(task.id)">Видалити</button>
24 </div>
25
26

```

Перевіримо наше відображення

Список завдань Встановити Angular Виконавець: Ярослав Дата виконання: 14.02.2025 Статус: Виконано До роботи Видалити
Ознайомитися з компонентами Опис: Ознайомитися з компонентами та оглянути взаємодію між ними Виконавець: Ярослав Дата виконання: 15.02.2025 Статус: В процесі До роботи Видалити
Ознайомитися з Control Flow Опис: Ознайомитися з старим і новим підходом до Control Flow Виконавець: Ярослав Дата виконання: 25.02.2025 Статус: Виконано До роботи Видалити

Можемо побачити, що з'явилося нове поле, яке дозволяє обрати статус, але за замовчуванням у кожному стоїть перший статус, а не його поточний. Для того, щоб це виправити можна використати `[(ngModel)]` з `[ngValue]`

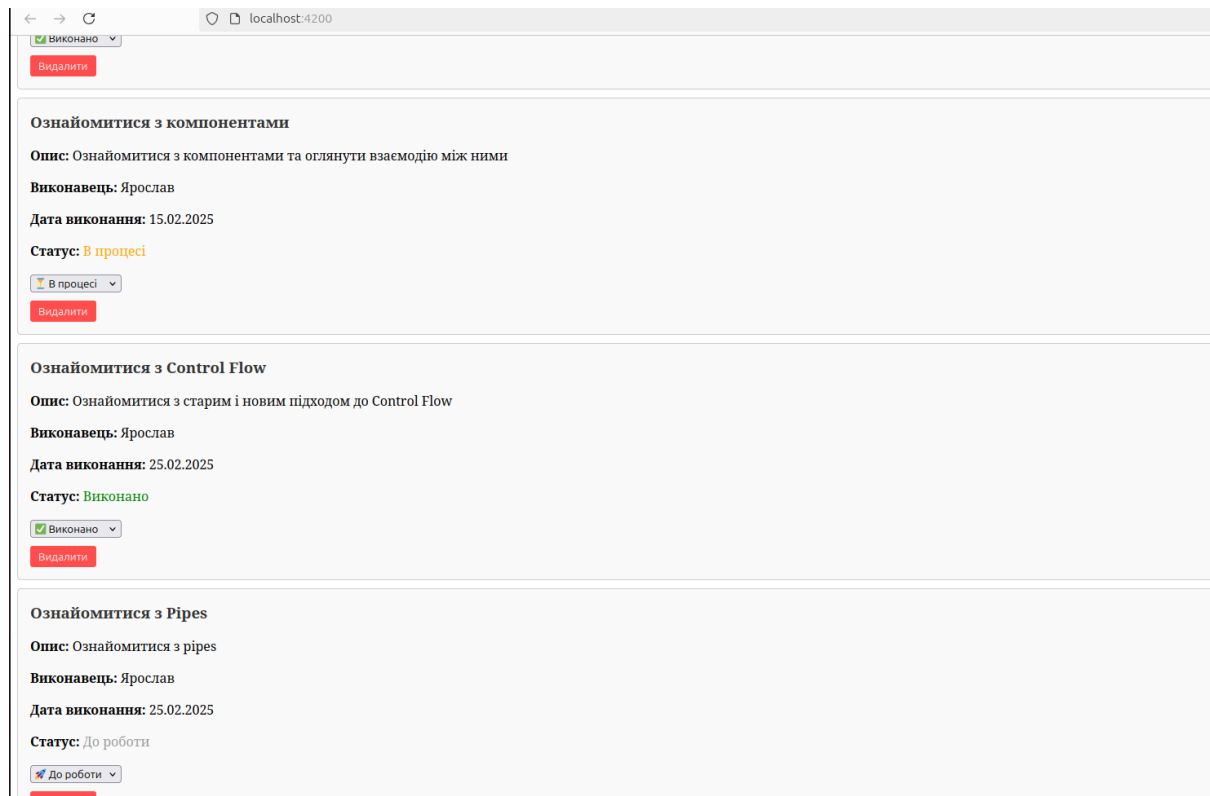
```
model.ts tasks.ts task-list.component.html task-item.component.ts task-item.component.html x status.enum.ts tasi
<div class="task-item">
  <h3>{{ task.title }}</h3> <!-- виводимо заголовок звертаючись до властивості title об'єкта tasks -->
  @if(task.description) { <!-- виводимо опис звертаючись до властивості description об'єкта tasks, якщо вона є -->
    <p><strong>Опис:</strong> {{ task.description }}</p>
  }

  <!-- виводимо виконавця звертаючись до властивості assignee об'єкта tasks -->
  <p><strong>Виконавець:</strong> {{ task.assignee }}</p>
  <!-- виводимо дату виконання звертаючись до властивості dueDate об'єкта tasks -->
  <p><strong>Дата виконання:</strong> {{ task.dueDate }}</p>
  <!-- виводимо статус звертаючись до властивості status об'єкта tasks -->
  <p><strong>Статус:</strong> <span [ngClass]="getStatusClasses()">{{ task.status }}</span>
  </p>

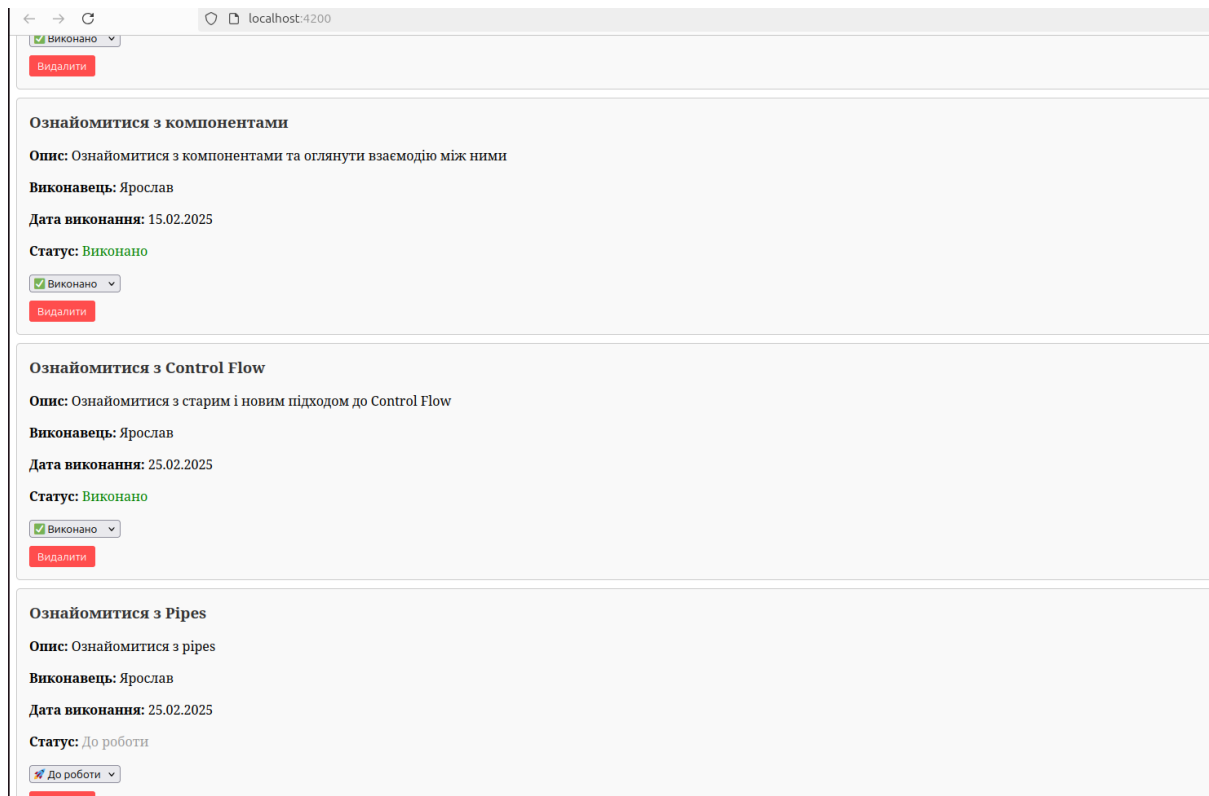
  <div class="select-status">
    <select [(ngModel)]="task.status" (change)="updateStatus($event)">
      <option [ngValue]="TaskStatus.TODO">🔥 До роботи</option>
      <option [ngValue]="TaskStatus.IN_PROGRESS">🔄 В процесі</option>
      <option [ngValue]="TaskStatus.DONE">✅ Виконано</option>
    </select>
  </div>

  <button (click)="deleteTask(task.id)">Видалити</button>
</div>
```

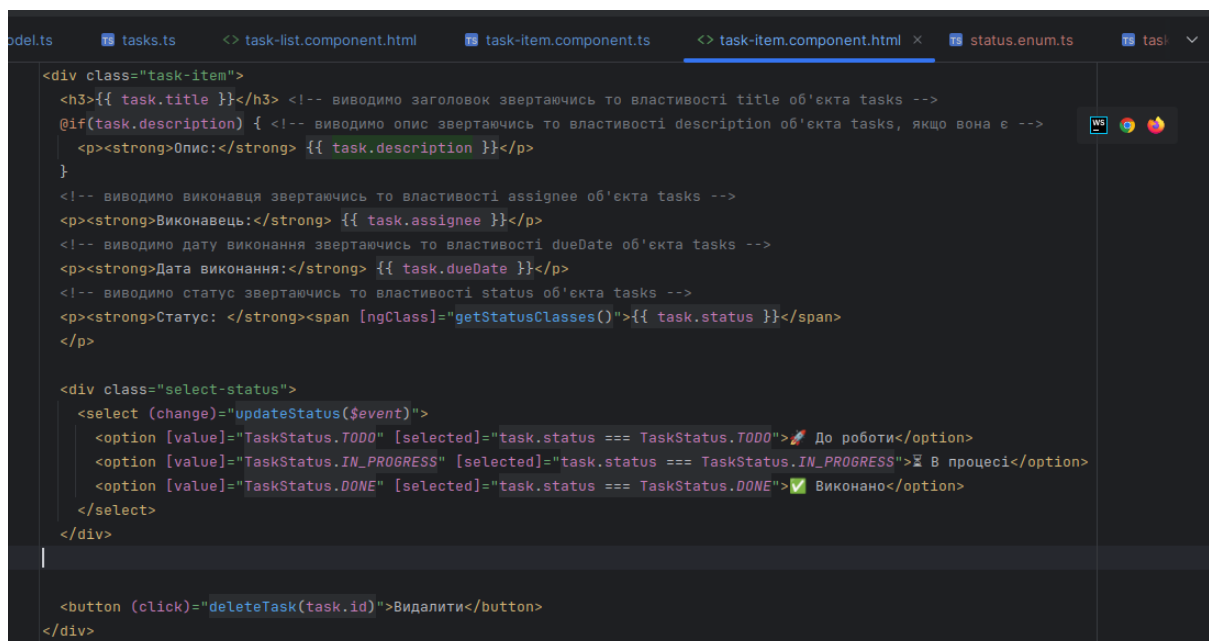
Перевіримо відображення



Як бачимо, тепер статуси відповідають статусу поточного завдання і якщо ми його змінимо, то за рахунок [(ngModel)] він запишеться у змінну task.status автоматично



Але є і альтернативний підхід, можна звернутися до властивості `selected` кожного option



Перевіримо відображення та побачимо, що і з цим варіантом статуси відображаються коректно

← → ↻

localhost:4200

✓ Виконано

Видалити

Ознайомитися з компонентами

Опис: Ознайомитися з компонентами та оглянути взаємодію між ними

Виконавець: Ярослав

Дата виконання: 15.02.2025

Статус: В процесі

В процесі

Видалити

Ознайомитися з Control Flow

Опис: Ознайомитися з старим і новим підходом до Control Flow

Виконавець: Ярослав

Дата виконання: 25.02.2025

Статус: Виконано

✓ Виконано

Видалити

Ознайомитися з Pipes

Опис: Ознайомитися з pipes

Виконавець: Ярослав

Дата виконання: 25.02.2025

Статус: До роботи

До роботи

Але у цьому випадку, при зміні статусу, властивість автоматично не зміниться

← → ↻

localhost:4200

✓ Виконано

Видалити

Ознайомитися з компонентами

Опис: Ознайомитися з компонентами та оглянути взаємодію між ними

Виконавець: Ярослав

Дата виконання: 15.02.2025

Статус: В процесі

✓ Виконано

Видалити

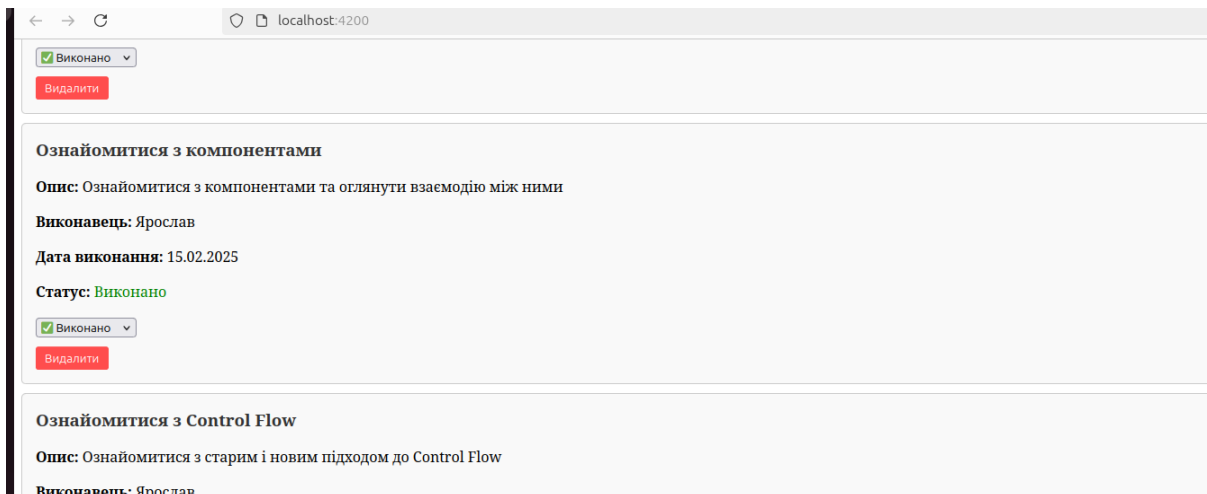
Ознайомитися з Control Flow

Опис: Ознайомитися з старим і новим підходом до Control Flow

Виконавець: Ярослав

Дата виконання: 25.02.2025

Ось тут нам і потрібен обробник події change, назначимо за його обробку функцію updateStatus та реалізуємо її



Бачимо, що статус автоматично змінюється при зміні його через select. Також додамо стилі для нашого select

```
18 select {
19   padding: 5px;
20   border: 1px solid #ccc;
21   border-radius: 3px;
22   cursor: pointer;
23   background: white;
24   font-size: 14px;
25   transition: all 0.2s ease-in-out;
26 }
27
28 select:hover {
29   border-color: #888;
30 }
31
32 select:focus {
33   outline: none;
34   border-color: #555;
35 }
36
37 option {
38   padding: 5px;
39 }
40
```

3) Далі попрацюємо з pipes, як приклад створимо пайп фільтрації завдань за статусом. Для цього потрібно створити файл нашого пайпу вручну або за допомогою команди:

ng g p share/pipes/status-filter

```

svarog@svarog:~/projects/angular-course$ ng g p share/pipes/status-filter
CREATE src/app/share/pipes/status-filter.pipe.spec.ts (212 bytes)
CREATE src/app/share/pipes/status-filter.pipe.ts (250 bytes)
UPDATE src/app/app.module.ts (857 bytes)
svarog@svarog:~/projects/angular-course$

```

Наш пайп був автоматично зареєстрований у app.module.ts

```

1  > import ...
12
13  @NgModule({ Show usages  Kryvyi Yaroslav
14    declarations: [
15      AppComponent,
16      TaskListComponent,
17      TaskItemComponent,
18      TaskFormComponent,
19      StatusFilterPipe
20    ],
21    imports: [
22      BrowserModule,
23      AppRoutingModule,
24      FormsModule
25    ],
26    providers: [],
27    bootstrap: [AppComponent]
28  })
29  export class AppModule { }
30

```

Переходимо до нашого пайпу

```

1  import { Pipe, PipeTransform } from '@angular/core';
2
3  @Pipe({ Show usages
4    name: 'statusFilter',
5    standalone: false
6  })
7  export class StatusFilterPipe implements PipeTransform {
8
9    transform(value: unknown, ...args: unknown[]): unknown { no usages
10      return null;
11    }
12
13  }

```

Ми бачимо у декораторі @Pipe назву нашого пайпу, саме її ми будемо використовувати при зверненні до нього. Також інтерфейс

PipeTransform зобов'язує нас реалізувати функцію transform. Можемо побачити, що за замовчуванням функція приймає дані, які потрібно обробити та опціонально параметри. Змінимо базові значення на наші, ми будемо приймати масив завдань, тому перше значення у нас буде типу Task[], як параметр, ми будемо передавати статус через enum або значення “all” і повертати ми будемо також Task[]

```
1 import { Pipe, PipeTransform } from '@angular/core';
2 import { Task } from '../../core/models/task.model';
3 import { TaskStatus } from '../../core/models/status.enum';
4
5 @Pipe({ Show usages
6   name: 'statusFilter',
7   standalone: false
8 })
9 export class StatusFilterPipe implements PipeTransform {
10   transform(tasks: Task[], status: TaskStatus | 'all'): Task[] { no usages
11     return tasks;
12   }
13 }
```

Тепер реалізуємо логіку, потрібно повертати вхідний масив, якщо значення “all”, а також опціонально null або undefined, і фільтрований, якщо є статус.

```
1 import { Pipe, PipeTransform } from '@angular/core';
2 import { Task } from '../../core/models/task.model';
3 import { TaskStatus } from '../../core/models/status.enum';
4
5 @Pipe({ Show usages
6   name: 'statusFilter',
7   standalone: false
8 })
9 export class StatusFilterPipe implements PipeTransform {
10   transform(tasks: Task[], status: TaskStatus | 'all'): Task[] { no usages
11     if (!tasks) {
12       return [];
13     }
14
15     if (status === 'all' || status === null || status === undefined) {
16       return tasks;
17     }
18
19     return tasks.filter(task: Task => task.status === status);
20   }
21 }
```

Тепер додамо до **task-list.component.html** select для вибору статусу

```

1 <h2>Список завдань</h2>
2 <div class="task-filter">
3   <p>Фільтр за статусом</p>
4   <select>
5     <option [value]="all">Всі</option>
6     <option [value]="TaskStatus.TODO">До роботи</option>
7     <option [value]="TaskStatus.IN_PROGRESS">В процесі</option>
8     <option [value]="TaskStatus.DONE">Виконано</option>
9   </select>
10 </div>
11 @for(task of myTasks; track task.id; let index = $index) {
12   <app-task-item
13     [task]="task"
14     (taskDeleted)="deleteTask($event)"
15   >
16   </app-task-item>
17 } @empty {
18   <p>Курси відсутні</p>
19 }

```

І трохи стилізуємо

enable File Watcher to compile SCSS to CSS?

```

1 .task-filter {
2   display: flex;
3   flex-direction: row;
4   flex-wrap: wrap;
5   gap: 0.5rem;
6   margin-bottom: 0.5rem;
7
8   p {
9     margin: 5px;
10  }
11
12  select {
13    width: 40%;
14    cursor: pointer;
15    font-size: 14px;
16  }
17 }
18

```

Отримаємо

Список завдань

Фільтр за статусом

Всі

Встановити Angular

Виконавець: Ярослав

Дата виконання: 14.02.2025

Статус: Виконано

Виконано

Видалити

Ознайомитися з компонентами

Опис: Ознайомитися з компонентами та оглянути взаємодію між ними

Виконавець: Ярослав

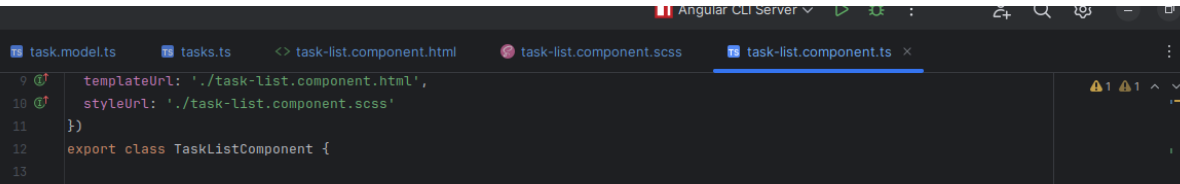
Дата виконання: 15.02.2025

Статус: В процесі

В процесі

Видалити

Тепер необхідно реалізувати логіку по отриманню параметра фільтрації та передачі його пайпу. Для цього у select пропишемо (change)="onSelected(\$event)" і реалізуємо її.



```
9  @Component({
10  templateUrl: './task-list.component.html',
11  styleUrls: ['./task-list.component.scss']
12  })
13  export class TaskListComponent {
14
15      myTasks: Task[] = tasks; // підключаємо тестові дані, вказавши за тип наш інтерфейс, так як записів
16      // декілька, то вказуємо, що це буде масив
17      protected readonly TaskStatus :typeof TaskStatus = TaskStatus;
18
19      selectedStatus!: TaskStatus | 'all';
20
21      deleteTask(index: number): void { Show usages 1 Kryvyi Yaroslav
22          console.log(`Видалити завдання з індексом: ${index}`);
23      }
24
25      onSelect(event: Event): void { Show usages new *
26          const status :string = (event.target as HTMLSelectElement).value;
27          this.selectedStatus = status as TaskStatus | 'all';
28      }
29  }
30
```

Перевіряємо роботу

← → ↻ localhost:4200

Список завдань

Фільтр за статусом Всі

Встановити Angular

Виконавець: Ярослав

Дата виконання: 14.02.2025

Статус: Виконано

☒ Виконано

Видалити

Ознайомитися з компонентами

Опис: Ознайомитися з компонентами та оглянути взаємодію між ними

Виконавець: Ярослав

Дата виконання: 15.02.2025

Статус: В процесі

☐ В процесі

Видалити

Ознайомитися з Control Flow

Опис: Ознайомитися з старим і новим підходом до Control Flow

Виконавець: Ярослав

Дата виконання: 25.02.2025

Статус: Виконано

☒ Виконано

← → ↻ localhost:4200 ☆

Список завдань

Фільтр за статусом До роботи

Ознайомитися з Pipes

Опис: Ознайомитися з pipes

Виконавець: Ярослав

Дата виконання: 25.02.2025

Статус: До роботи

☐ До роботи

Видалити

Але що буде, якщо під час фільтрації змінити статус?

← → ↻ localhost:4200 ☆

Список завдань

Фільтр за статусом До роботи

Ознайомитися з Pipes

Опис: Ознайомитися з pipes

Виконавець: Ярослав

Дата виконання: 25.02.2025

Статус: Виконано

☒ Виконано

Видалити

Бачимо, що елемент не відфільтрувався. Чому так?

Вся справа у властивості декоратора `@Pipe`, а саме `pure`, яка за замовчуванням кешує значення і змінює його тільки якщо зміняться вхідні

дані. Щоб це обійти, потрібно встановити `pure: false`, і тоді пайп буде виконуватися при кожній зміні компонента

```
task.model.ts tasks.ts task-list.component.html task-list.component.scss task-list.component.ts status-filter.pipe.ts x
1 import { Pipe, PipeTransform } from '@angular/core';
2 import { Task } from '../../core/models/task.model';
3 import { TaskStatus } from '../../core/models/status.enum';
4
5 @Pipe({ Show usages
6   name: 'statusFilter',
7   pure: false,
8   standalone: false
9 })
10 export class StatusFilterPipe implements PipeTransform {
11   transform(tasks: Task[], status: TaskStatus | 'all'): Task[] { Show usages
12     if (!tasks) {
13       return [];
14     }
15
16     if (status === 'all' || status === null || status === undefined) {
17       return tasks;
18     }
19
20     return tasks.filter(task : Task => task.status === status);
21   }
22 }
23
```

Перевіряємо відображення

← → ↻ localhost:4200

Список завдань

Фільтр за статусом Виконано

Встановити Angular

Виконавець: Ярослав

Дата виконання: 14.02.2025

Статус: Виконано

☒ Виконано

Видалити

Ознайомитися з Control Flow

Опис: Ознайомитися з старим і новим підходом до Control Flow

Виконавець: Ярослав

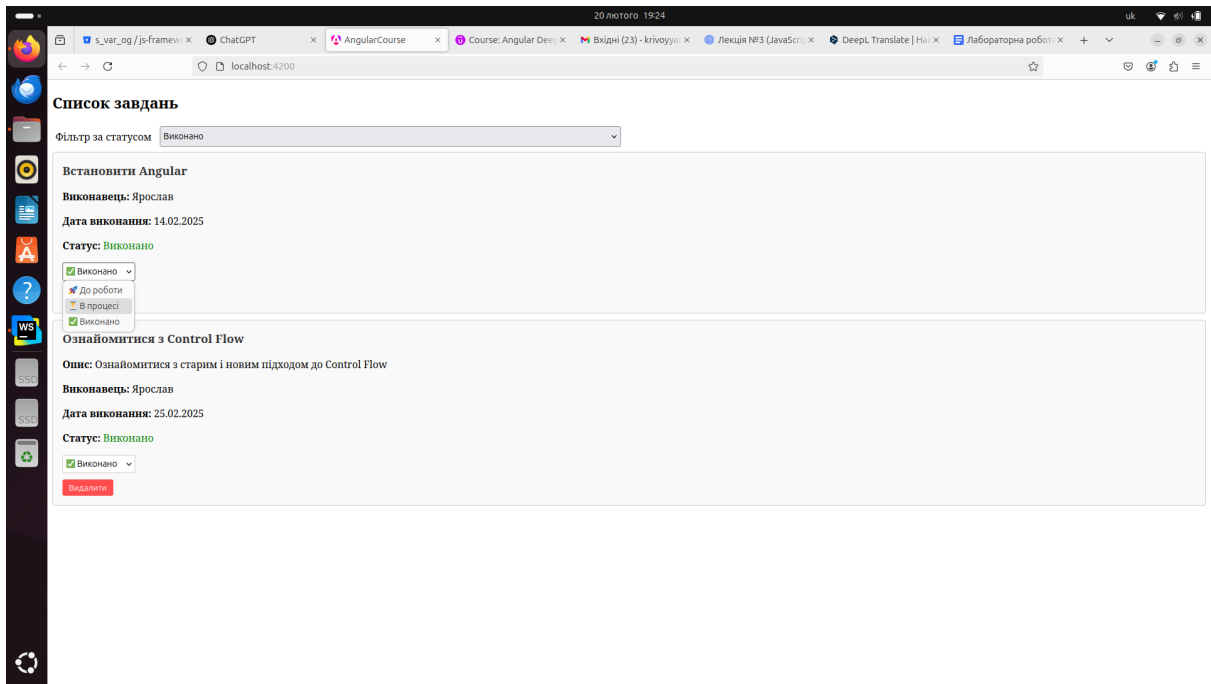
Дата виконання: 25.02.2025

Статус: Виконано

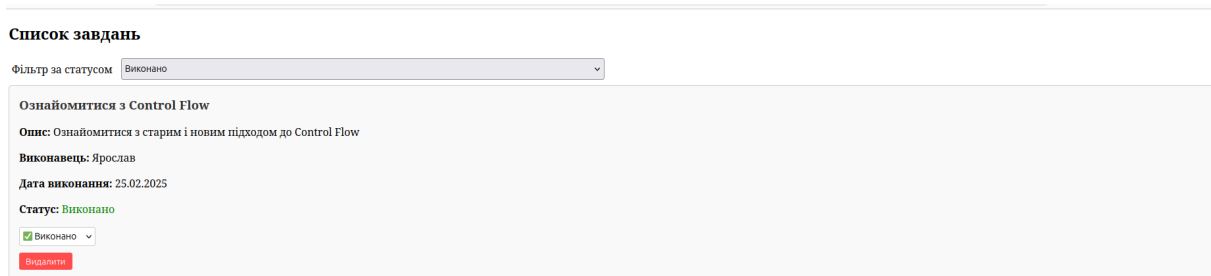
☒ Виконано

Видалити

Змінимо статус



Перевіримо результат



Також за рахунок відображення завдань за допомогою `@for` можемо переробити функцію видалення завдань

```
templateUrl: './task-list.component.html',
styleUrl: './task-list.component.scss'
})
export class TaskListComponent {

  myTasks: Task[] = tasks; // підключаємо тестові дані, вказавши за тип наш інтерфейс, так як записів
                           // декілька, то вказуємо, що це буде масив
  protected readonly TaskStatus: typeof TaskStatus = TaskStatus;

  selectedStatus!: TaskStatus | 'all';

  deleteTask(index: number): void { Show usages  Kryvyi Yaroslav *
    this.myTasks = this.myTasks.filter(task: Task => task.id !== index);
  }

  onSelect(event: Event): void { Show usages  new *
    const status: string = (event.target as HTMLSelectElement).value;
    this.selectedStatus = status as TaskStatus | 'all';
  }
}
```

Як приклад

localhost:4200

Список завдань

Фільтр за статусом

Всі

Встановити Angular

Виконавець: Ярослав

Дата виконання: 14.02.2025

Статус: Виконано

Виконано

Видалити

Ознайомитися з компонентами

Опис: Ознайомитися з компонентами та оглянути взаємодію між ними

Виконавець: Ярослав

Дата виконання: 15.02.2025

Статус: В процесі

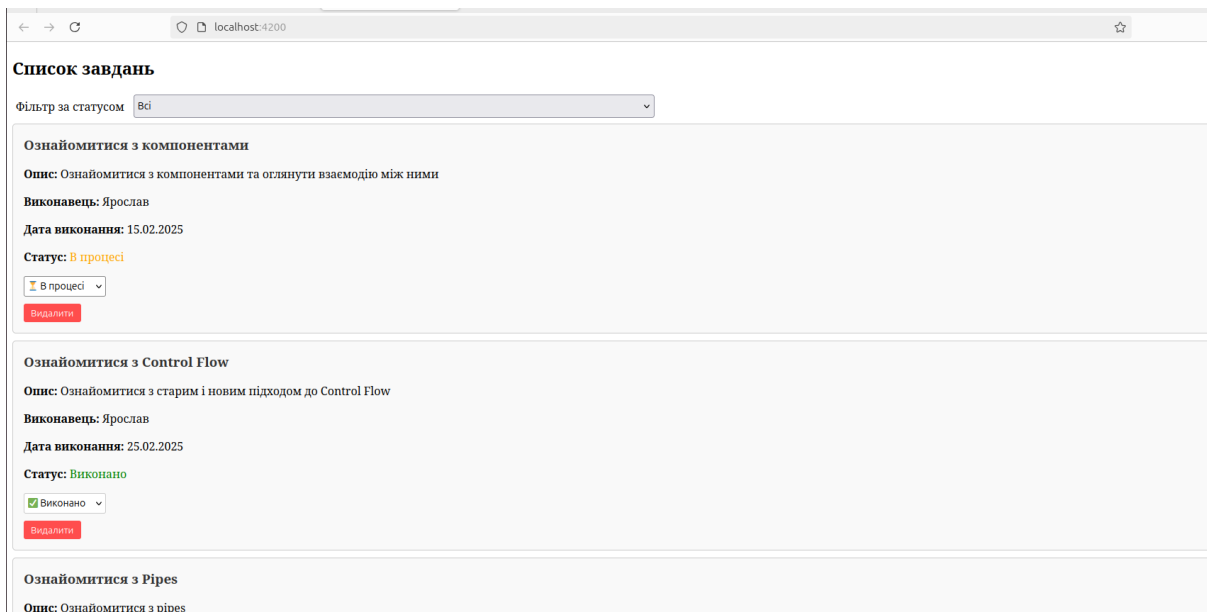
В процесі

Видалити

Ознайомитися з Control Flow

Опис: Ознайомитися з старим і новим підходом до Control Flow

Виконавець: Ярослав



Контрольні питання

- 1) Для чого потрібен track у @for? Чи є він обов'язковим?
- 2) Для чого використовується @Pipe?
- 3) На що впливає властивість pure у @Pipe?
- 4) За яким принципом @switch розуміє, який @case відобразити? Чи потрібно писати break?
- 5) Що може слугувати за ідентифікатор для track?
- 6) Чи можна за допомогою @ViewChild отримати доступ до дочірнього шаблону?