Лабораторна робота № 4. Форми в Angular.

Мета: навчитися використовувати різні підходи до створення форм в Angular.

Завдання

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

Хід роботи

1) Для початку модернізуємо інтерфейс завдання, змінимо тип “dueData” зі string на Date

```
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: Date; // дата дедлайну
  status: TaskStatus; // статус завдання
}
```

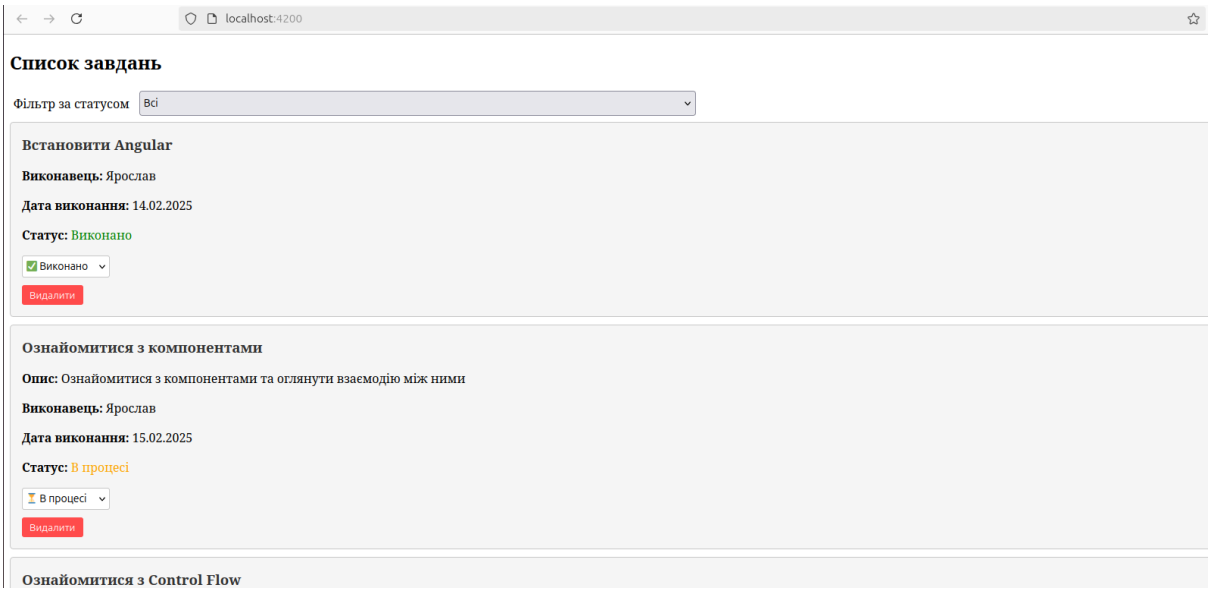
2) Змінимо тестові дані, щоб вони відповідали новому типу, конструктор Date очікує формат, який підтримується браузером, наприклад, YYYY-MM-DD або MM/DD/YYYY

```ts
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: new Date('2025-02-14'),
    status: TaskStatus.DONE
  },
  {
    id: 1,
    title: 'Ознайомитися з компонентами',
    description: 'Ознайомитися з компонентами та оглянути взаємодію між ними',
    assignee: 'Ярослав',
    dueDate: new Date('2025-02-15'),
    status: TaskStatus.IN_PROGRESS
  },
  {
    id: 2,
    title: 'Ознайомитися з Control Flow',
    description: 'Ознайомитися з старим і новим підходом до Control Flow',
    assignee: 'Ярослав',
    dueDate: new Date('2025-02-21'),
    status: TaskStatus.DONE
```

3) Відкорегуємо шаблон **task.item.component.html** для коректного відображення дати, для цього використаємо DatePipe

```
<!-- виводимо дату виконання звертаючись то властивості dueDate об'єкта tasks -->
<p><strong>Дата виконання:</strong> {{ task.dueDate | date: format: 'd.MM.yyyy'}}</p>
<!-- виводимо статус звертаючись то властивості status об'єкта tasks -->
```

Перевіримо відображення



4) Переходимо до створення форми, вона буде використовуватися для додавання нового завдання, а також редагування існуючого. Створимо форму з використанням 2 підходів: Template-driven, Reactive Form. Перед початком роботи створюємо новий компонент task-form

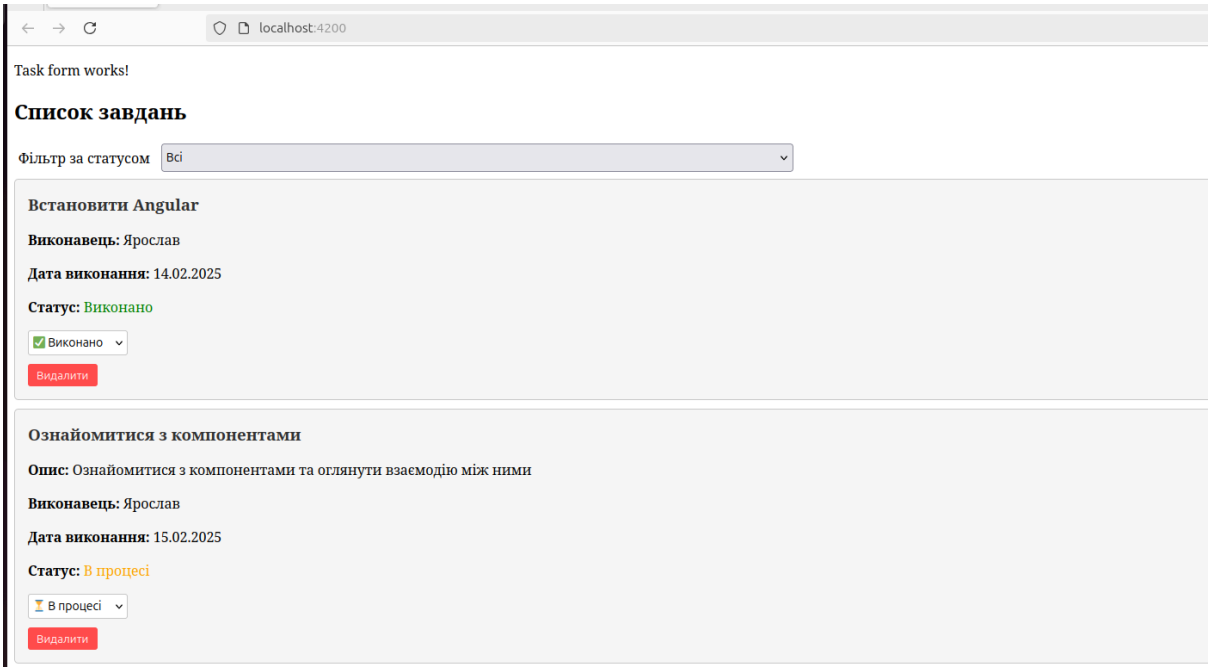
```
Terminal   Local   Local (2)
svarog@svarog:~/projects/angular-course$ ng g c components/task-form
CREATE src/app/components/task-form/task-form.component.scss (0 bytes)
CREATE src/app/components/task-form/task-form.component.html (24 bytes)
CREATE src/app/components/task-form/task-form.component.spec.ts (612 bytes)
CREATE src/app/components/task-form/task-form.component.ts (232 bytes)
UPDATE src/app/app.module.ts (857 bytes)
svarog@svarog:~/projects/angular-course$
```

Додамо виклик компоненту у **task.list.component.html**

ask.model.ts | tasks.ts | app.component.html | task-item.component.html | task-list.component.html

```
<app-task-form></app-task-form>
<h2>Список завдань</h2>
<div class="task-filter">
  <p>Фільтр за статусом</p>
  <select (change)="onSelected($event)">
    <option [value]="'all'">Всі</option>
    <option [value]="TaskStatus.TODO">До роботи</option>
    <option [value]="TaskStatus.IN_PROGRESS">В процесі</option>
    <option [value]="TaskStatus.DONE">Виконано</option>
  </select>
</div>
@for(task of myTasks | statusFilter: selectedStatus; track task.id; let index = $index) {
  <app-task-item
    [task]="task"
    (taskDeleted)="deleteTask($event)"
  >
  </app-task-item>
} @empty {
  <p>Завдання відсутні</p>
}
```

Перевіряємо відображення



4.1) **Template-driven**

У файлі **task.form.component.ts** пропишемо наступний код

```
export class TaskFormComponent implements OnInit {
  task!: Task;

  ngOnInit(): void {  no usages
    this.task = {
      id: -1,
      title: '',
      description: '',
      dueDate: new Date(),
      assignee: '',
      status: TaskStatus.TODO
    }
  }
```

У цьому коді створюємо зміну типу **Task** та ініціалізуємо її початковими значеннями за допомогою хука **OnInit**, який спрацьовує після створення компонента. Ця зміна нам знадобиться для зберігання значень форми.

У файлі **task.form.component.html** створюємо форму, використовуючи **ngModel** для зв’язування форми з нашою змінною

```
<form #taskForm="ngForm" class="task-form">
 <div class="task-form-control">
        <label for="title">Назва</label>
        <input type="text" id="title"
        name="title" [(ngModel)]="task.title"
        required #titleControl="ngModel"
        >
 </div>

 <div class="task-form-control">
        <label for="description">Опис</label>
        <input type="text" id="description"
        name="description" [(ngModel)]="task.description"
        #descriptionControl="ngModel"
        >
 </div>

 <div class="task-form-control">
        <label for="dueDate">Дата дедлайну</label>
        <input type="date" id="dueDate"
        name="dueDate" [(ngModel)]="task.dueDate"
        required #dueDateControl="ngModel"
        >
 </div>

 <div class="task-form-control">
        <label for="assignee">Виконавець</label>
        <input type="text" id="assignee"
        name="assignee" [(ngModel)]="task.assignee"
        required #assigneeControl="ngModel"
        >
 </div>

 <div class="task-form-control">
        <label for="status">Статус</label>
        <select id="status"
        name="status" [(ngModel)]="task.status"
        required #statusControl="ngModel"
        >
        <option [ngValue]="TaskStatus.TODO">До роботи</option>
        <option [ngValue]="TaskStatus.IN_PROGRESS">В процесі</option>
        <option [ngValue]="TaskStatus.DONE">Виконано</option>
        </select>
```

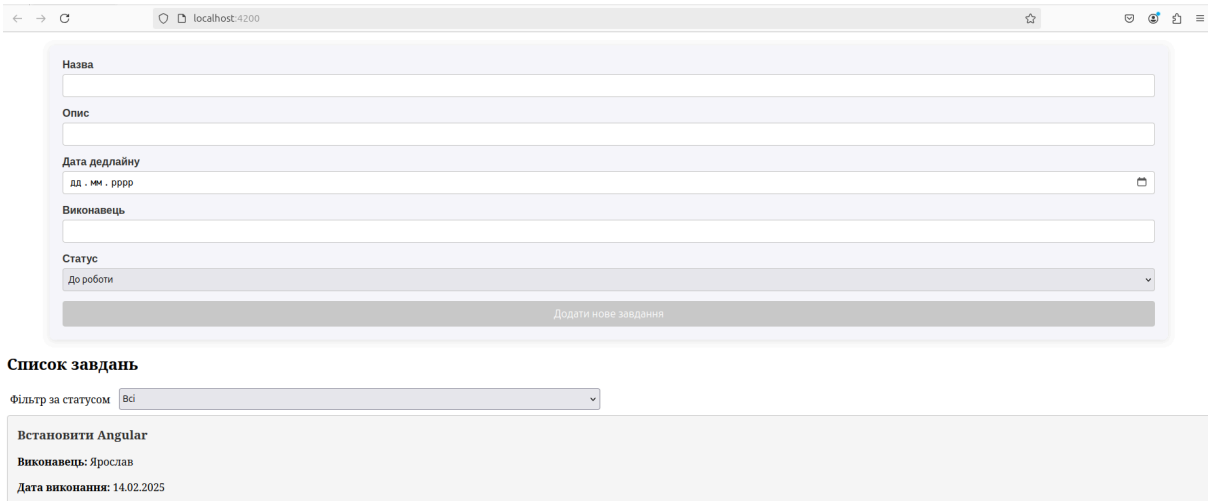
```
  </div>

        <button (click)="addTask()" type="submit" [disabled]="!taskForm.valid">Додати нове завдання</button>
</form>
```

Стилізуємо нашу форму (стилі можна писати власні)

```
.task-form {
  max-width: 90vw;
  margin: 20px auto;
  padding: 20px;
  background: #f8f9fa;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  font-family: Arial, sans-serif;
}

.task-form-control {
  display: flex;
  flex-direction: column;
  margin-bottom: 15px;

  label {
        font-weight: bold;
        margin-bottom: 5px;
        color: #333;
  }

  input, select {
        padding: 8px;
        border: 1px solid #ccc;
        border-radius: 4px;
        font-size: 14px;
        transition: border-color 0.3s ease;
  }

  input:focus, select:focus {
        border-color: #007bff;
        outline: none;
        box-shadow: 0 0 5px rgba(0, 123, 255, 0.2);
  }
}

button {
  width: 100%;
  padding: 10px;
  font-size: 16px;
```

```
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  transition: background 0.3s ease;

  &:disabled {
        background-color: #ccc;
        cursor: not-allowed;
  }

  &:not(:disabled):hover {
        background-color: #0056b3;
  }
}
```

Перевіримо відображення



Поки форма невалідна - кнопка додавання нового завдання заблокована. Досягається це за допомогою **[disabled]="!taskForm.valid"**. Для додавання завдання назначили обробник **(click)="addTask()"**, реалізуємо його у файлі **task.form.component.ts**. Так як input типу date повертає строкове значення, то перед додаванням нового завдання необхідно конвертувати дату до типу Date

```
addTask(): void {  Show usages
  const copyTask = {
    ...this.task,
    dueDate: new Date(this.task.dueDate),
  }
}
```

...this.task це спред-оператор (spread operator ...), який створює копію всіх поточних властивостей об'єкта task. Поле dueDate, яке зараз містить рядок ('yyyy-MM-dd'), перетворюємо у об'єкт Date (для цього ми використовуємо **new Date(this.task.dueDate)**, оскільки t**his.task.dueDate** передається у вигляді string).

Якщо просто написати:

this.task.dueDate = new Date(this.task.dueDate);

то об'єкт змінюється напряму, що небажано в Angular, оскільки може порушити оновлення змінних у шаблоні.

З спред-оператором створюється новий об'єкт з оновленими даними, що краще працює з Change Detection.

Далі переходимо до передачі нового завдання до батьківського компонента, тому що масив завдань зберігається саме у ньому. Для цього скористаємося декоратором **@Output**

```
@Output()
taskAdd :EventEmitter<Task>  = new EventEmitter<Task>();
```

Модернізуємо **addTask()**

```
addTask(): void {  Show usages
  const copyTask = {
    ...this.task,
    dueDate: new Date(this.task.dueDate),
  }

  this.taskAdd.emit(copyTask);
}
```

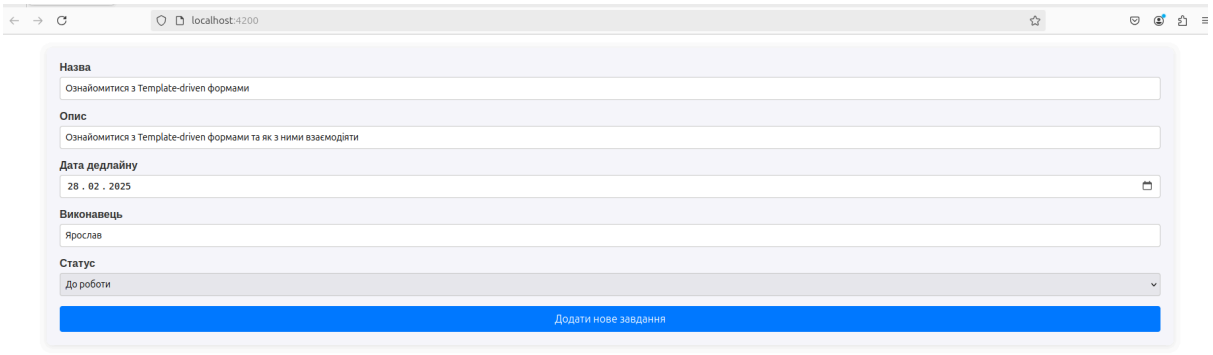
У файлі **task-list.component.html** прописуємо нашу кастомну подію та назначаємо її обробник

```
<app-task-form
  (taskAdd)="addTask($event)"
></app-task-form>
<h2>Список завдань</h2>
<div class="task-filter">
  <p>Фільтр за статусом</p>
```

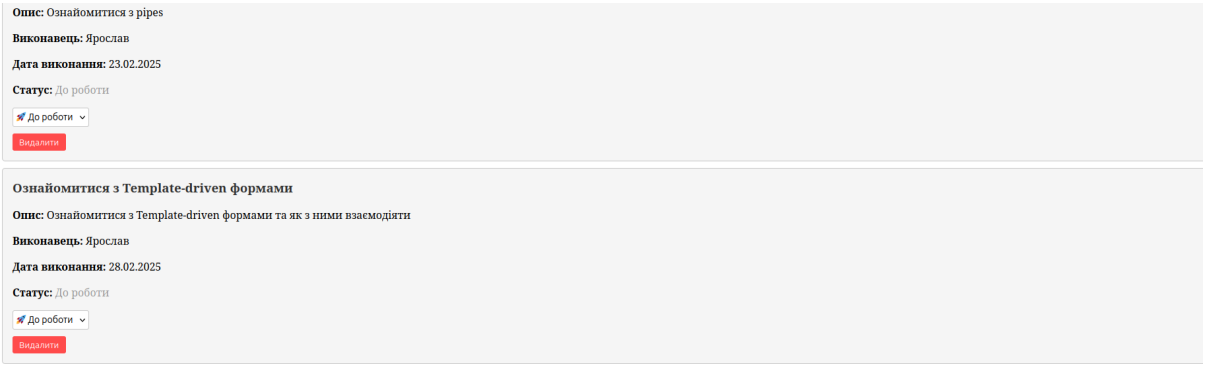
Реалізуємо обробник

```
addTask(task: Task): void {  Show usages  new *
  this.myTasks.push(task);
}
}
```

Перевіряємо відображення та роботу форми



Бачимо, що коли всі поля заповнені (форма валідна), кнопка стає активною, напишемо на неї та перевіримо, чи додається нове завдання



Також зверніть увагу, що якщо ви не скинули значення полів форми, то вони після відправки форми залишаться у полях. Для того щоб скинути значення форми необхідно викликати функцію reset для ngForm. Для цього потрібно модернізувати функцію **addTask()** так, щоб вона приймала форму, як аргумент.

```
  <button (click)="addTask(taskForm)" type="submit" [disabled]="!taskForm.valid">Додати нове завдання</button>
</form>
```

```
  addTask(taskForm: NgForm): void {  Show usages
    const copyTask = {
      ...this.task,
      dueDate: new Date(this.task.dueDate),
    }
    taskForm.reset();
    this.taskAdd.emit(copyTask);
  }
}
```

Також кожне завдання має id, яке ми не передаємо при створені. Реалізуємо логіку назначення id кожному новому завданню.

```
  addTask(task: Task): void {  Show usages  new *
    const maxId :number  = this.myTasks.length > 0 ? Math.max(...this.myTasks.map(task :Task  => task.id)) : 0;
    task = {
      ...task,
      id: maxId + 1,
    }
    this.myTasks.push(task);
  }
}
```

Тепер реалізуємо логіку редагування існуючого завдання. Для цього потрібно додати до **task-item-component.html** кнопку, яка буде викликати цю подію, а у **task-item-component.ts** додати **@Output**, який буде передавати завдання для редагування

```
<div class="select-status">
  <select (change)="updateStatus($event)">
    <option [value]="TaskStatus.TODO" [selected]="task.status === TaskStatus.TODO">🚀 До роботи
    <option [value]="TaskStatus.IN_PROGRESS" [selected]="task.status === TaskStatus.IN_PROGRESS
    <option [value]="TaskStatus.DONE" [selected]="task.status === TaskStatus.DONE">✅ Виконано
  </select>
</div>

<button class="task-item-button edit" (click)="editTask()">Редагувати</button>
<button class="task-item-button delete" (click)="deleteTask(task.id)">Видалити</button>
</div>
```

Передавати можно як об’єкт завдання, так і його id. В залежності від способу передачі, будуть відрізнятися способи обробки

```
@Input() task!: Task; // отримуємо об'єкт завдання
@Output() taskDeleted: EventEmitter<number>  = new EventEmitter<number>(); // подія для видалення завдання
@Output() taskEdited: EventEmitter<Task>  = new EventEmitter<Task>(); // подія для редагування завдання

protected readonly TaskStatus :typeof TaskStatus  = TaskStatus;

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


editTask(): void {  Show usages  new *
  this.taskEdited.emit(this.task);
}
```

У файлі **task-list.component.html** прописуємо нашу кастомну подію та назначаємо її обробник

```
    <option [value]="TaskStatus.IN_PROGRESS">В процесі</option>
    <option [value]="TaskStatus.DONE">Виконано</option>
  </select>
</div>
@for(task of myTasks | statusFilter: selectedStatus; track task.id; let index = $index) {
  <app-task-item
    [task]="task"
    (taskDeleted)="deleteTask($event)"
    (taskEdited)="editTask($event)"
  >
  </app-task-item>
} @empty {
  <p>Завдання відсутні</p>
}
```

Реалізуємо обробник у файлі **task.list.component.ts**

```
  editingTask: Task | null = null;

  editTask(task: Task): void {  Show usages  new *
    this.editingTask = {...task};
  }
```

Змінна **editingTask** нам необхідна для передачі завдання до форми у якості **@Input**

Пропишемо передачу параметра у файлі **task-list.component.html**

```
<app-task-form
  [editTask]="editingTask"
  (taskAdd)="addTask($event)"
></app-task-form>
<h2>Список завдань</h2>
<div class="task-filter">
```

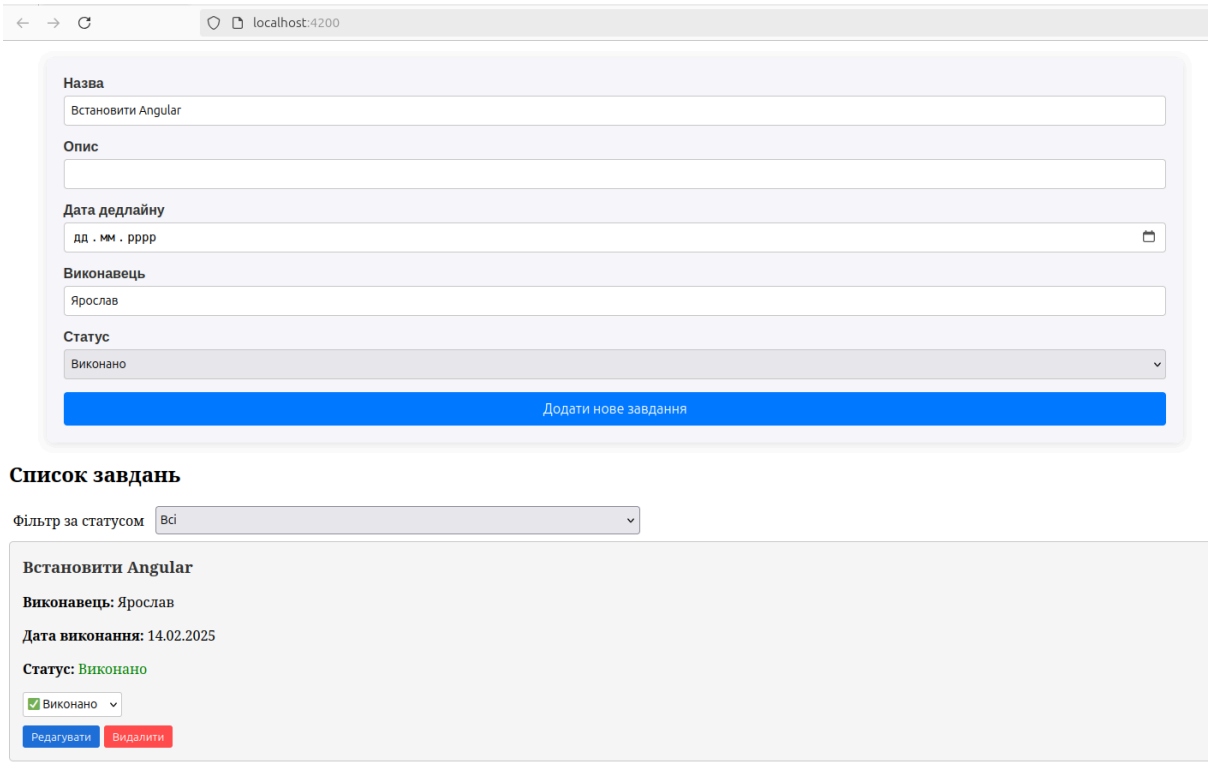
Переходимо до файлу t**ask-form.component.ts** та прописуємо **@Input**

```
  protected readonly TaskStatus :typeof TaskStatus  = TaskStatus;

  @Output()
  taskAdd :EventEmitter<Task>  = new EventEmitter<Task>();
  @Input() editTask: Task | null = null;

  ngOnInit(): void {  no usages
```

За допомогою хука **OnChanges**, який спрацьовує після оновлення **@Input** реалізуємо передачу даних завдання нашій змінній **task**

```
export class TaskFormComponent implements OnInit, OnChanges {
  task!: Task;
  protected readonly TaskStatus :typeof TaskStatus  = TaskStatus;

  @Output()
  taskAdd :EventEmitter<Task>  = new EventEmitter<Task>();
  @Input() editTask: Task | null = null;

  ngOnChanges(changes: SimpleChanges): void {  no usages
    if(changes['editTask'] && this.editTask) {
      this.task = {...this.editTask};
    }
  }
```

Перевіряємо відображення



Для відображення також дати необхідно вводити змінну, яка буде конвертувати дату між типами при отримані та відправці. Також до такої змінної потрібно буде прив’язати ngModel у формі.

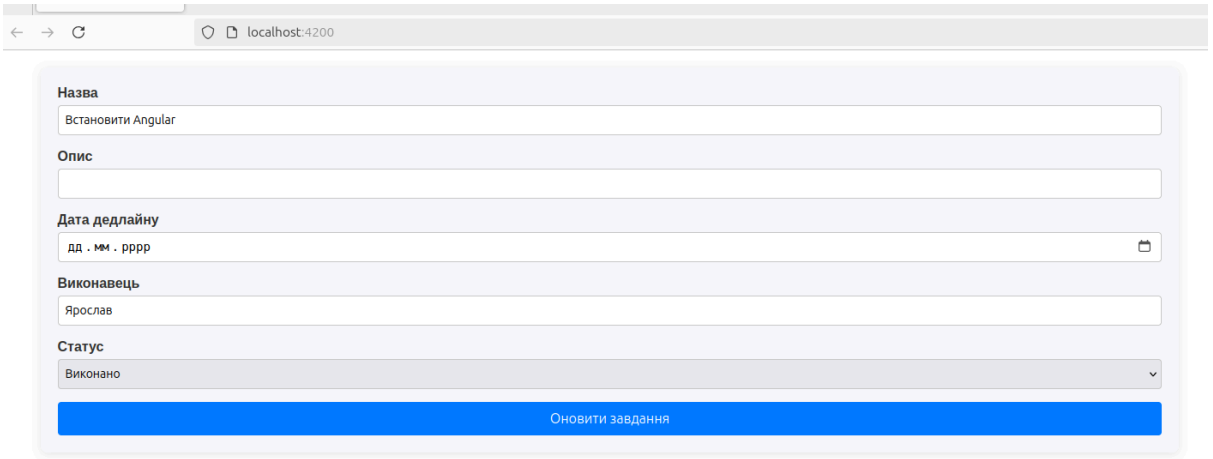
Тепер реалізуємо оновлення даних, щоб не створювати новий **@Output**, підв’яжемося вже під існуючий. Для цього необхідно внести певні зміни у компонент. Для початку зробимо динамічний напис кнопки у залежності від того додаємо чи редагуємо завдання.

```
    </div>

    <div class="task-form-control">
      <label for="status">Статус</label>
      <select id="status"
              name="status" [(ngModel)]="task.status"
              required #statusControl="ngModel"
      >
        <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)="addTask(taskForm)" type="submit" [disabled]="!taskForm.valid"
    >{{ editTask ? 'Оновити завдання' : 'Додати нове завдання'}}</button>
  </form>
```

Тепер необхідно модернізувати функцію **addTask()** у **task.list.component.ts**

```
addTask(task: Task): void {  Show usages  new *
  if (this.editingTask) {
    this.myTasks = this.myTasks.map(t :Task =>
      t.id === task.id ? { ...task } : t
    );
    this.editingTask = null;
  } else {
    const maxId :number = this.myTasks.length > 0 ? Math.max(...this.myTasks.map(task :Task => task.id)) : 0;
    task = {
      ...task,
      id: maxId + 1,
    }
    this.myTasks.push(task);
  }

}
```

Перевіряємо відображення та роботу



Список завдань

Фільтр за статусом Всі

Встановити Angular

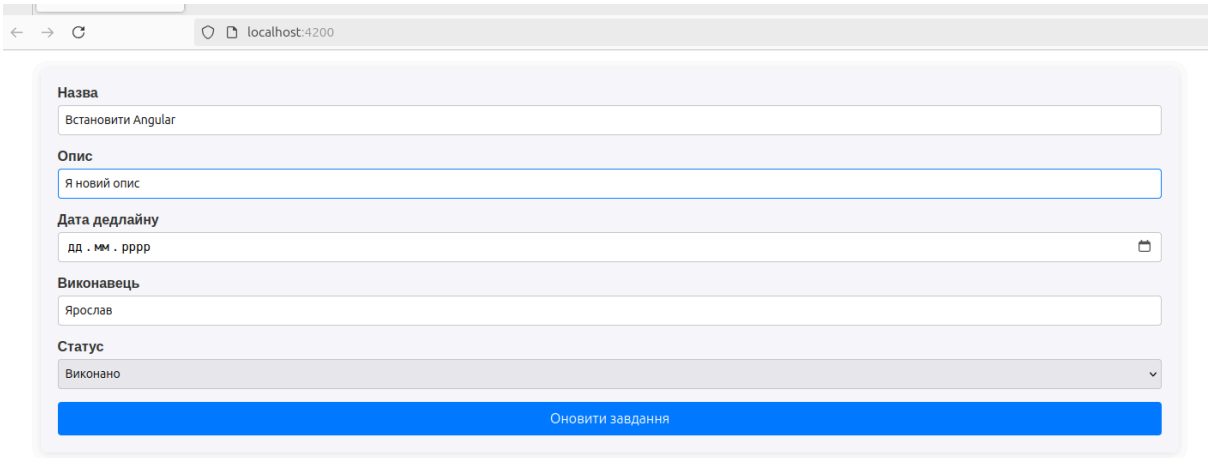
Виконавець: Ярослав

Дата виконання: 14.02.2025

Статус: Виконано

✅ Виконано

Редагувати Видалити



Список завдань

Фільтр за статусом Всі

Встановити Angular

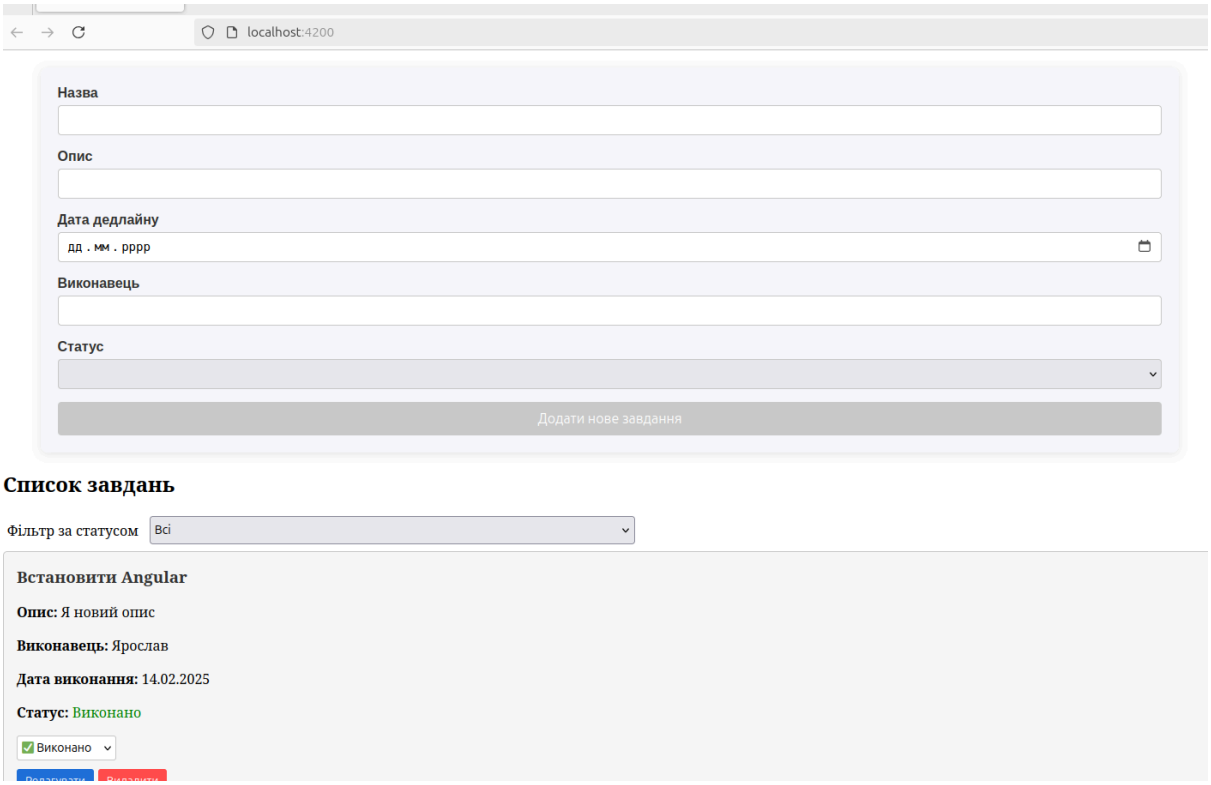
Виконавець: Ярослав

Дата виконання: 14.02.2025

Статус: Виконано

✅ Виконано

Редагувати Видалити



4.2) **Reactive Forms**

Розглянемо приклад реалізації за допомогою Reactive Forms. У цьому прикладі вже будемо використовувати обробку валідації.

Модернізуємо **task.form.component.ts**

```
export class TaskFormComponent implements OnChanges {

  taskForm : FormGroup<{ title: FormControl<string | n... = new FormGroup({
    title: new FormControl('', Validators.required),
    description: new FormControl(''),
    dueDate: new FormControl('', Validators.required),
    assignee: new FormControl('', Validators.required),
    status: new FormControl(TaskStatus.TODO, Validators.required),
  })
```

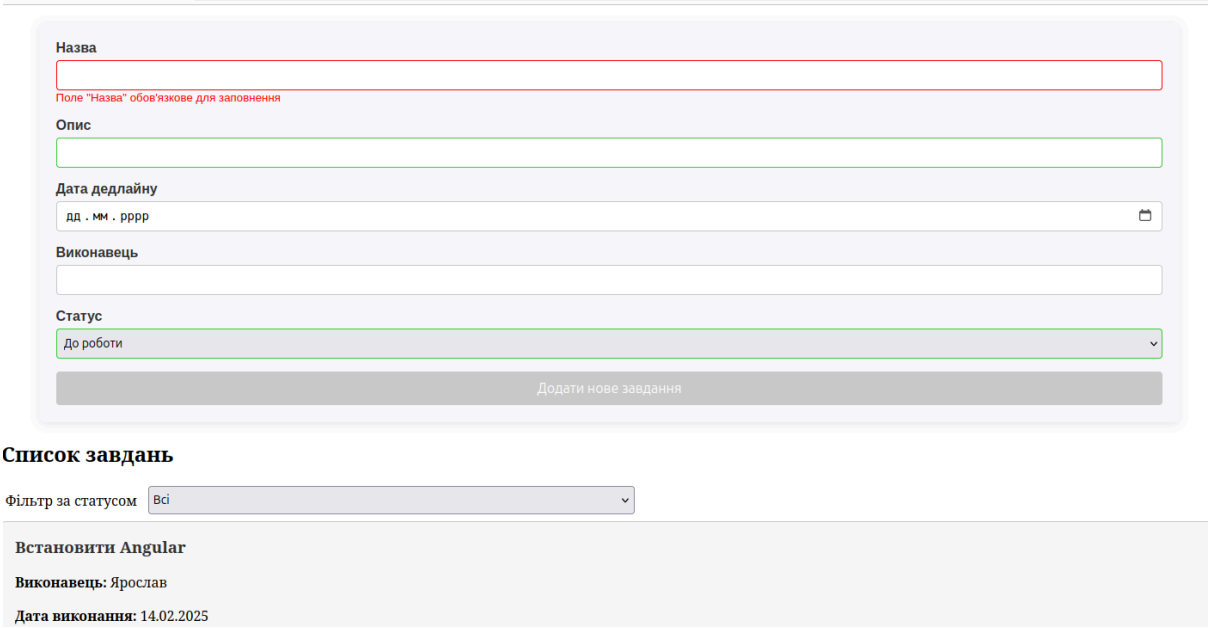
Модернізуємо **task.form.component.html**

```
<form [formGroup]="taskForm" class="task-form">
  <div class="task-form-control">
        <label for="title">Назва</label>
        <input type="text" id="title"
        name="title" formControlName="title"
```

```
        >
        @if (taskForm.get('title')?.hasError('required') && taskForm.get('title')?.dirty) {
        <small class="error">Поле "Назва" обов'язкове для заповнення</small>
        }
 </div>

 <div class="task-form-control">
        <label for="description">Опис</label>
        <input type="text" id="description"
        name="description" formControlName="description"
        >
 </div>

 <div class="task-form-control">
        <label for="dueDate">Дата дедлайну</label>
        <input type="date" id="dueDate"
        name="dueDate" formControlName="dueDate"
        >
        @if (taskForm.get('dueDate')?.hasError('required') && taskForm.get('dueDate')?.dirty) {
        <small class="error">Поле "Дата дедлайну" обов'язкове для заповнення</small>
        }
 </div>

 <div class="task-form-control">
        <label for="assignee">Виконавець</label>
        <input type="text" id="assignee"
        name="assignee" formControlName="assignee"
        >
        @if (taskForm.get('assignee')?.hasError('required') && taskForm.get('assignee')?.dirty) {
        <small class="error">Поле "Виконавець" обов'язкове для заповнення</small>
        }
 </div>

 <div class="task-form-control">
        <label for="status">Статус</label>
        <select id="status"
        name="status" formControlName="status"
        >
        <option [value]="TaskStatus.TODO">До роботи</option>
        <option [value]="TaskStatus.IN_PROGRESS">В процесі</option>
        <option [value]="TaskStatus.DONE">Виконано</option>
        </select>
        @if (taskForm.get('status')?.hasError('required') && taskForm.get('status')?.dirty) {
        <small class="error">Поле "Статус" обов'язкове для заповнення</small>
        }
 </div>

 <button
```

```
        (click)="addTask()" type="submit" [disabled]="!taskForm.valid"
 >{{ editTask ? 'Оновити завдання' : 'Додати нове завдання'}}</button>
</form>
```

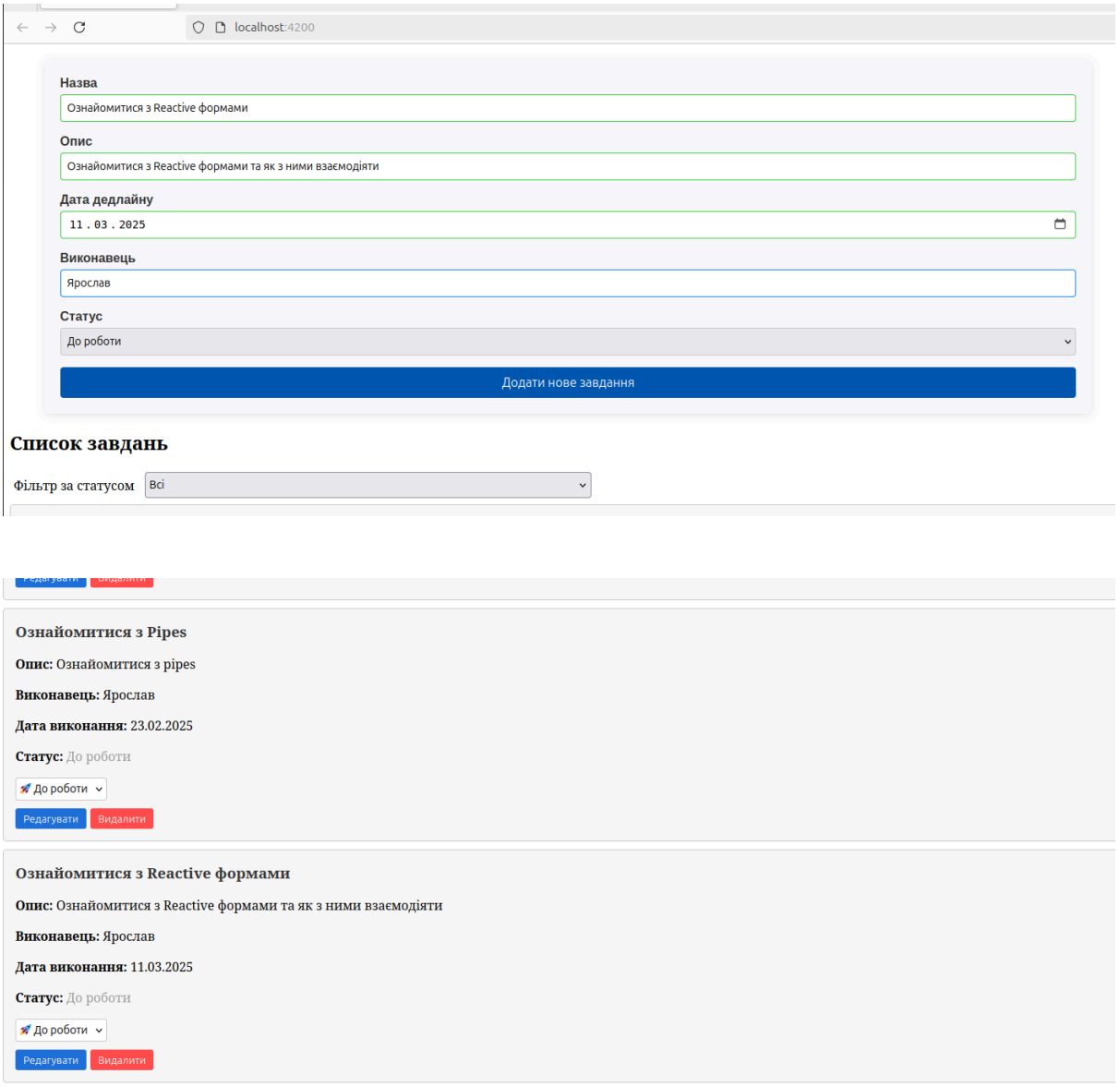
Перевіримо відображення форми та реакцію на невалідні поля



Далі модернізуємо функцію додавання нового завдання. Так як дату отримуємо у виді строки, то необхідно її конвертувати до типу Date. Так як поле у конструкторі може бути порожнім, то додатково необхідно обробити цю можливість.

```
addTask(): void {  Show usages
  if (this.taskForm.valid) {
    let taskData : { dueDate: Date; title?: string | null | ... } = {
      ...this.taskForm.value,
      dueDate: this.taskForm.value.dueDate ? new Date(this.taskForm.value.dueDate) : new Date(),
    };
    this.taskAdd.emit(taskData as Task);
    this.taskForm.reset();
  }
}
```

Обробка у файлі **task.list.component.ts** залишається без змін
Перевіримо відображення та роботу



Далі переробимо логіку оновлення. Так як дата приходить у вигляді об’єкта Date, то input з типом date не може його відобразити. Для цього потрібно конвертувати у зрозумілий для браузера вид.

```
ngOnChanges(changes: SimpleChanges): void {  no usages
  if(changes['editTask'] && this.editTask) {
    this.taskForm.patchValue({
      ...this.editTask,
      dueDate: this.editTask.dueDate.toISOString().split("T")[0]
    });
  }
}
```

```
addTask(): void {  Show usages
  if (this.taskForm.valid) {
    let taskData : { dueDate: Date; id: number | undefined; ... } = {
      ...this.taskForm.value,
      dueDate: this.taskForm.value.dueDate ? new Date(this.taskForm.value.dueDate) : new Date(),
      id: this.editTask ? this.editTask.id : undefined,
    };
    this.taskAdd.emit(taskData as Task);
    this.taskForm.reset();
  }
}
```

Створимо два кастомних валідатора, один буде перевіряти заборонені слова, а інший валідувати дату.

Для цього у теці share створимо теку directives. Створимо у теці файл task-form.validator.ts (див. лекцію). Створимо у ньому клас TaskFormValidator та дві статичні функції forbiddenWordsValidator та dateValidator

```
export class TaskFormValidator {  no usages
  static forbiddenWordsValidator(words: string[]): ValidatorFn {  no usages
    return (control: AbstractControl): ValidationErrors | null => {
      return null;
    }
  }

  static dateValidator(control: AbstractControl): ValidationErrors | null {  no usages
    return null;
  }
}
```

Реалізуємо функції

```ts
import {AbstractControl, ValidationErrors, ValidatorFn} from '@angular/forms';

export class TaskFormValidator {  Show usages

  static forbiddenWordsValidator(words: string[]): ValidatorFn {  Show usages
    return (control: AbstractControl): ValidationErrors | null => {
      if (!control.value || typeof control.value !== 'string') {
        return null;
      }

      const hasForbiddenWord : boolean  = words.some(word : string  => control.value.includes(word.toLowerCase()));
      return hasForbiddenWord ? { forbiddenWord: true } : null;
    }
  }

  static dateValidator: ValidatorFn = (control: AbstractControl): ValidationErrors | null => {  Show usages
    if (!control.value) return null;

    const inputDate = new Date(control.value);
    const currentYear : number  = new Date().getFullYear();

    return inputDate.getFullYear() < currentYear ? { dateVal: true } : null;
  }
}
```

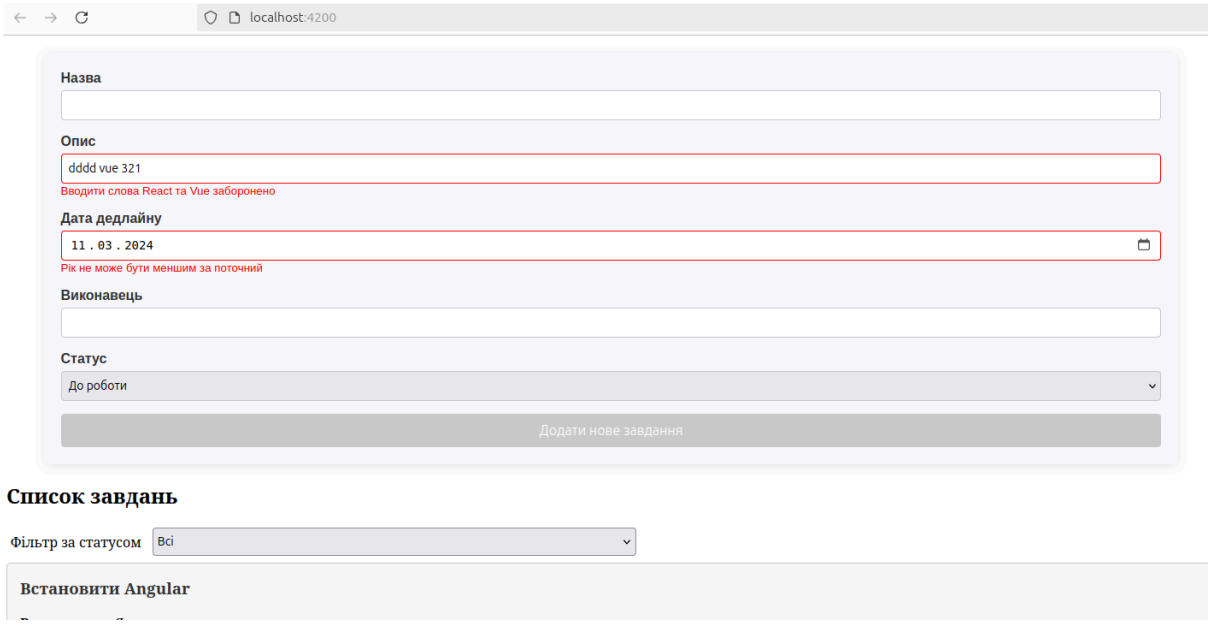
Підключимо наші кастомні валідатори до полів форми

```ts
@Component({  Show usages
  selector: 'app-task-form',
  standalone: false,
  templateUrl: './task-form.component.html',
  styleUrl: './task-form.component.scss',
})
export class TaskFormComponent implements OnChanges {

  taskForm : FormGroup<{ title: FormControl<string | n...  = new FormGroup({
    title: new FormControl('', Validators.required),
    description: new FormControl('', TaskFormValidator.forbiddenWordsValidator(['React', 'Vue'])),
    dueDate: new FormControl('', [Validators.required, TaskFormValidator.dateValidator]),
    assignee: new FormControl('', Validators.required),
    status: new FormControl(TaskStatus.TODO, Validators.required),
  })

  protected readonly TaskStatus : typeof TaskStatus  = TaskStatus;

  @Output()
  taskAdd : EventEmitter<Task>  = new EventEmitter<Task>();
  @Input() editTask: Task | null = null;


  ngOnChanges(changes: SimpleChanges): void {  no usages
```

У шаблоні пропишемо обробку кастомних валідаторів

```
<form [formGroup]="taskForm" class="task-form">
  <div class="task-form-control">
        name="description" formControlName="description"
  >
  @if (taskForm.get('description')?.hasError( errorCode: 'forbiddenWord')) {
    <small class="error">Вводити слова React та Vue заборонено</small>
  }
</div>

<div class="task-form-control">
  <label for="dueDate">Дата дедлайну</label>
  <input type="date" id="dueDate"
        name="dueDate" formControlName="dueDate"
  >
  @if (taskForm.get('dueDate')?.hasError( errorCode: 'required') && taskForm.get('dueDate')?.dirty) {
    <small class="error">Поле "Дата дедлайну" обов'язкове для заповнення</small>
  }
  @if (taskForm.get('dueDate')?.hasError( errorCode: 'dateVal')) {
    <small class="error">Рік не може бути меншим за поточний</small>
  }
</div>

<div class="task-form-control">
  <label for="assignee">Виконавець</label>
```

Перевіримо обробку помилок



Контрольні питання

1) Для чого потрібен FormGroup? Чи можна використовувати декілька у одній формі?
2) Для чого потрібен FormControl? Які початкові значення може приймати?
3) Як звернутися до FormControl у шаблоні?
4) Які методи та властивості дозволяють отримати або перевірити наявність помилок валідації?
5) Як відбувається зв’язування між FormControl/FormGroup з шаблоном?