

KOMUNIKACJA Z BACKEND

SPIS TREŚCI

Spis treści	1
Cel zajęć.....	1
Rozpoczęcie	1
Uwaga	1
Omówienie wytycznych aplikacji	2
Inicjalizacja projektu.....	2
Serwer Backend API	3
Interfejs i serwis	6
Komponent Tasks	8
Implementacja komponentu ArchiveComponent.....	22
Commit projektu do GIT.....	30
Podsumowanie.....	30

CEL ZAJĘĆ

Celem głównym zajęć jest zdobycie umiejętności:

- obsługi routingu w Angular,
- komunikacji z warstwą backend za pomocą serwisów.

W praktycznym wymiarze stworzona zostanie prosta aplikacja „Task manager” do zapisywania zadań do wykonania.

ROZPOCZĘCIE

Rozpoczęcie zajęć. Powtórzenie tworzenia komponentów, serwisów. Powtórzenie wiązań.

Wejściówka?

UWAGA

Ten dokument aktywnie wykorzystuje niestandardowe właściwości. Podobnie jak w LAB A wejdź do **Plik** -> **Informacje** -> **Właściwości** -> **Właściwości zaawansowane** -> **Niestandardowe** i zaktualizuj pola. Następnie uruchom ten dokument ponownie lub **Ctrl+A** -> **F9**.

OMÓWIENIE WYTYCZNYCH APLIKACJI

Widok bieżących zadań:

- Umożliwia dodanie nazwy nowego zadania wraz z opcjonalną datą jego wykonania. Data powinna być ustawiana poprzez standardowy element formularza HTML5 typu `type="date"`.
- Wyświetla listę aktualnych zadań.
- Umożliwia zaznaczenie wybranego zadania jako wykonane poprzez użycie checkbox-a.
- Umożliwia przesunięcie wszystkich zadań oznaczonych jako wykonane do archiwum.

Widok zadań zarchiwizowanych:

- Wyświetla listę zadań o statusie archiwalnym
- Umożliwia trwałe kasowanie wybranych zadań poprzez kliknięcie umieszczonego w zadaniu przycisku [usuń].

INICJALIZACJA PROJEKTU

Wejdź terminalem do katalogu `C:\...\Desktop\ai2b` i zainicjalizuj projekt z wykorzystaniem komendy:

```
> ng new lab-d
```

Standardowo kreator zapyta o konfigurację routingu (wybrać **Tak**) oraz preprocesor CSS (zostawić zwykły CSS). Zainicjalizowane zostanie także repozytorium GIT.

Po zakończonej instalacji, uruchom aplikację w trybie deweloperskim z wykorzystaniem komendy:

```
> cd C:\Users\Damian\Desktop\ai2b\lab-d
> ng serve --port=49451
```

Tradycyjnie, do pliku `src/styles.css` dodaj znak wodny ze swoim numerem albumu:

```
body {
  background: url("https://placeholder.co/100x100/FFFFFF/EFEFEF/png?text=49451");
}
```

Uruchom przeglądarkę pod adresem: `http://localhost:49451`.

Edytuj zawartość pliku `src/app/app.component.html`. Usuń domyślnie wygenerowaną zawartość. Zostaw jedynie znacznik `<router-outlet>`. Utwórz własne, stałe elementy interfejsu aplikacji, np. nagłówek i stopkę. Np.:

```
<header>
  <h1>Backend-Based Task Manager</h1>
</header>

<div class="router-outlet">
  <router-outlet></router-outlet>
</div>

<footer>Imię Nazwisko</footer>
```

Dodaj style w plikach `src/styles.css` (globalne) i `src/app/app.component.css` wg własnego uznania.

Przykładowy efekt:

Backend-Based Task Manager

00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000

00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000

00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000

00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000

00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000

Imię Nazwisko

SERWER BACKEND API

Do głównego katalogu projektu wgraj otrzymany wraz z zadaniem plik `todos.json`. W terminalu wykonaj polecenie, które zainstaluje `json-server`:

```
> npm install -g json-server
```

Uruchom serwer poleceniem (zastąp `00000` przez swój numer albumu, obniżony o `1000`):

```
> json-server --watch --port=00000 todos.json
```

Po uruchomieniu serwera, można odwiedzić w przeglądarce testowe API:

- `http://localhost:00000/todos`
- `http://localhost:00000/todos/1`

Lista dostępnych operacji i przykładowe wywołania znajdują się poniżej (kompatybilne z klientem HTTP PhpStorm/WebStorm):

```
### GET list of tasks
GET {{baseUrl}}/
Accept: application/json

### GET list of non-completed tasks
GET {{baseUrl}}/?completed=false
Accept: application/json

### GET list of completed tasks
GET {{baseUrl}}/?completed=true
Accept: application/json

### GET list of archived tasks
GET {{baseUrl}}/?archived=false
```

```

Accept: application/json

### GET list of tasks sorted by ID ASC
GET {{baseUrl}}/?_sort=id&_order=asc
Accept: application/json

### GET list of tasks sorted by ID DESC
GET {{baseUrl}}/?_sort=id&_order=desc
Accept: application/json

### GET single task
GET {{baseUrl}}/1
Accept: application/json

### POST new task
POST {{baseUrl}}/
Content-Type: application/json

{
  "title": "Pierwsze zadanie",
  "deadline": "2022-01-28",
  "completed": false,
  "archived": false
}

### PUT update task
PUT {{baseUrl}}/3
Content-Type: application/json

{
  "title": "Pierwsze zadanie",
  "deadline": "2022-01-28",
  "completed": true,
  "archived": false
}

### DELETE task
DELETE {{baseUrl}}/3
Content-Type: application/json

```

Szczególną uwagę należy zwrócić na filtrowanie po polu `completed` i `archived` oraz sortowaniu po dowolnym polu.

Uruchom `json-serwer` i przetestuj wybrane metody REST serwisu używając wybranego klienta REST (Postman, PhpStorm, Curl).

Przedstaw zrzut ekranu przedstawiającego zapytanie i odpowiedź do endpointa `GET /`

```

C:\Users\Damian>curl -X GET "http://localhost:48451/todos/?_sort=id&_order=asc" -H "Accept: application/json"
[
  {
    "id": 1,
    "title": "Hello World",
    "deadline": "1900-01-01",
    "completed": false,
    "archived": false
  },
  {
    "id": 2,
    "title": "Hello Again World",
    "deadline": "2000-01-01",
    "completed": false,
    "archived": false
  }
]

```

Przedstaw zrzut ekranu przedstawiającego zapytanie i odpowiedź do endpointa `POST /`

```
C:\Users\Damian>curl -X POST "http://localhost:48451/todos" -H "Content-Type: application/json" -d "{\"title\": \"Pierwsze zadanie\", \"deadline\": \"2022-01-28\", \"completed\": false, \"archived\": false}"
{
  "title": "Pierwsze zadanie",
  "deadline": "2022-01-28",
  "completed": false,
  "archived": false,
  "id": 3
}
```

Przedstaw zrzut ekranu przedstawiającego zapytanie i odpowiedź do endpointa **PUT /** zmieniającego status zadania na skończone:

```
C:\Users\Damian>curl -X PUT "http://localhost:48451/todos/3" -H "Content-Type: application/json" -d "{\"title\": \"Pierwsze zadanie\", \"deadline\": \"2022-01-28\", \"completed\": true, \"archived\": false}"
{
  "title": "Pierwsze zadanie",
  "deadline": "2022-01-28",
  "completed": true,
  "archived": false,
  "id": 3
}
```

Przedstaw zrzut ekranu przedstawiającego zapytanie i odpowiedź do endpointa **PUT /** zmieniającego status zadania na zarchiwizowane:

```
C:\Users\Damian>curl -X PUT "http://localhost:48451/todos/3" -H "Content-Type: application/json" -d "{\"title\": \"Pierwsze zadanie\", \"deadline\": \"2022-01-28\", \"completed\": true, \"archived\": true}"
{
  "title": "Pierwsze zadanie",
  "deadline": "2022-01-28",
  "completed": true,
  "archived": true,
  "id": 3
}
```

Przedstaw zmodyfikowaną zawartość pliku **todos.json** po wykonaniu powyższych zapytań. Zaznacz wprowadzone zmiany względem oryginału.

```
{
  "todos": [
    {
      "id": 1,
      "title": "Hello World",
      "deadline": "1900-01-01",
      "completed": false,
      "archived": false
    },
    {
      "id": 2,
      "title": "Hello Again World",
      "deadline": "2000-01-01",
      "completed": false,
      "archived": false
    },
    {
      "title": "Pierwsze zadanie",
      "deadline": "2022-01-28",
      "completed": true,
      "archived": true,
      "id": 3
    }
  ]
}
```

Punkty:

0

1

INTERFEJS I SERWIS

Utwórz interfejs Task z wykorzystaniem polecenia:

```
ng generate interface task
```

Utworzony zostanie plik `src/app/task.ts`. Zmień jego zawartość na odpowiadającą przetestowanemu powyżej API, przykładowo:

```
export interface Task {
  id?: number;
  title?: string;
  deadline?: Date;
  completed?: boolean;
  archived?: boolean;
}
```

Dzięki zastosowaniu znaku `?` umożliwiamy przechowywanie w interfejsie pól niezdefiniowanych, co przyda się przy wykorzystaniu interfejsu nie tylko w opisie elementów listy zadań, ale i przy zapytaniach POST i PUT, gdzie nie wszystkie pola są obecne.

Następnie wygeneruj serwis `TasksService` z wykorzystaniem polecenia:

```
ng generate service tasks --skip-tests
```

Utworzony zostanie plik `src/app/tasks.service.ts`. Ustaw jego zawartość na:

```
import { Injectable } from '@angular/core';
import { Observable } from "rxjs";
import { Task } from "../task";
import { HttpClient } from "@angular/common/http";

@Injectable({
  providedIn: 'root'
})
export class TasksService {

  constructor(
    private http: HttpClient,
  ) { }

  public index(archived = false): Observable<Task[]> {
    // ...
  }

  public post(task: Task): Observable<Task> {
    // ...
  }

  public put(task: Task): Observable<Task> {
    // ...
  }

  public delete(task: Task): Observable<any> {
    // ...
  }
}
```

Zaimplementuj ciała funkcji `index()`, `post()`, `put()`, `delete()`. W przypadku `index()` uwzględnij przekazanie parametru `archived` do pozyskania listy zadań aktywnych i archiwalnych.

W razie problemów, porównaj kod: <https://github.com/ideaspot-pl/ai2-lab-d-todo/commit/db67085e6949ffe587ee512e7266f27b91c95ba7>

Wstaw zrzut ekranu metody `index()` serwisu `TasksService`:

```
public index(archived = false): Observable<Task[]> {
  const url = this.baseUrl + '/todos';
  return this.http.get<Task[]>(url, {
    params: {
      archived: archived,
      _sort: 'id',
      _order: 'desc',
    }
  })
}
```

Wstaw zrzut ekranu metody `post()` serwisu `TasksService`:

```
public post(task: Task): Observable<Task> {
  const url = this.baseUrl + '/todos';
  return this.http.post(url, task);
}
```

Wstaw zrzut ekranu metody `put()` serwisu `TasksService`:

```
public put(task: Task): Observable<Task> {
  const url = this.baseUrl + '/todos/' + task.id;
  return this.http.put(url, task);
}
```

Wstaw zrzut ekranu metody `delete()` serwisu `TasksService`:

```
public delete(task: Task): Observable<any> {
  const url = this.baseUrl + '/todos/' + task.id;
  return this.http.delete(url);
}
```

Punkty:	0	1
---------	---	---

KOMPONENT TASKS

Wygeneruj komponent `TasksComponent` używając polecenia terminala:

```
> ng generate component tasks --skip-tests
```

Utworzony zostanie katalog `src/app/tasks`. W klasie komponentu w pliku `src/app/tasks/tasks.component.ts` dodaj pola `tasks: Task[]` and `newTask: Task`, odpowiednio do przechowywania pobranej z backendu listy zadań oraz do edytowania nowego zadania. Pola te będą powiązane z widokiem.

W konstruktorze wstrzyknij serwis `TasksService`, a pod konstruktorem zaimplementuj metodę `ngOnInit`, która podczas inicjalizacji komponentu wypełni go aktualnymi zadaniami (`archived === false`).

W pliku `src/app/app-routing.module.ts` skonfiguruj routy:

```
const routes: Routes = [
  {path: 'tasks', component: TasksComponent},
  {path: '', redirectTo: '/tasks', pathMatch: 'full'},
];
```

Zaimplementuj widok listy zadań (na razie bez akcji).

Przykładowa implementacja:


```
export class TasksComponent implements OnInit {
  public tasks: Task[] = [];
  public newTask: Task = {};

  no usages  ↳ Artur Karczmarczyk
  constructor(
    private tasksService: TasksService
  ) {
  }

  no usages  ↳ Artur Karczmarczyk
  ngOnInit() : void {
    this.tasksService.index().subscribe( observerOrNext: (tasks : Task[]) : void => {
      this.tasks = tasks;
    });
  }
}
```

```
<div class="container">
  <form class="new-task"></form>

  <div class="tasks">
    <div class="task" *ngFor="let task of tasks">
      <div class="title">{{ task.title }}</div>
      <div class="deadline">
        <span *ngIf="task.deadline">{{ task.deadline }}</span>
      </div>
      <div class="actions">
        <input type="checkbox" />
      </div>
    </div>
  </div>
</div>
```

Backend-Based Task Manager

Hello Again World

2000-01-01



Hello World

1900-01-01



W razie problemów, porównaj kod: <https://github.com/ideaspot-pl/ai2-lab-d-todo/commit/1377e7c8f819acca1f9a6898e98a8d3310bff9bc>

Wstaw rzut ekranu listy zadań:

Backend-Based Task Manager

Hello Again World

2000-01-01



Hello World

1900-01-01



To jest stopka

Wstaw rzut ekranu kodu `src/app/tasks/tasks.component.ts`:

```
import {Component, OnInit} from '@angular/core';
import {TasksService} from "../tasks.service";
import {Task} from "../task";

@Component({
  selector: 'app-tasks',
  templateUrl: './tasks.component.html',
  styleUrls: ['./tasks.component.css']
})
export class TasksComponent implements OnInit {
  public tasks: Task[] = [];
  public newTask: Task = {};

  constructor(
    private tasksService: TasksService
  ) {
  }

  ngOnInit() {
    this.tasksService.index().subscribe((tasks) => {
      this.tasks = tasks;
    });
  }
}
```

Wstaw rzut ekranu kodu src/app/tasks/tasks.component.html:

```
<div class="container">
  <form class="new-task"></form>

  <div class="tasks">
    <div class="task" *ngFor="let task of tasks">
      <div class="title">{{ task.title }}</div>
      <div class="deadline">
        <span *ngIf="task.deadline">{{ task.deadline }}</span>
      </div>
      <div class="actions">
        <input type="checkbox" />
      </div>
    </div>
  </div>
</div>
```

Punkty:	0	1
---------	---	---

Następnie zaimplementuj widok formularza dodawania nowego zadania, powyżej lub poniżej listy zadań. Pola formularza powinny być połączone z polem `newTask`. Zaimplementuj i wykorzystaj metodę `addTask()`, która zbierze dane nowego zadania i doda zadanie korzystając z metody `post()` serwisu.

Przykładowa implementacja:

Backend-Based Task Manager

Title

Deadline

mm/dd/yyyy

Add Task

Added	2023-11-21	<input type="checkbox"/>
Hello Again World	2000-01-01	<input type="checkbox"/>
Hello World	1900-01-01	<input type="checkbox"/>

```
addTask() : void {
  if (this.newTask.title === undefined) {
    return;
  }

  this.newTask.completed = false;
  this.newTask.archived = false;

  this.tasks.unshift(this.newTask); // optimistic update; try commenting this line off and compare the difference


  this.tasksService.post(this.newTask).subscribe( observerOrNext: (task : Task) : void => {
    this.newTask = {};
    this.ngOnInit();
  });
}
```

```
<form class="new-task">
  <div class="form-group">
    <label for="title">Title</label>
    <input type="text" id="title" name="title" [(ngModel)]="newTask.title" class="form-control" [disabled]="isProcessing" />
  </div>
  <div class="form-group">
    <label for="deadline">Deadline</label>
    <input type="date" id="deadline" name="title" [(ngModel)]="newTask.deadline" class="form-control" [disabled]="isProcessing" />
  </div>
  <div class="form-group">
    <button type="button" class="btn btn-primary" (click)="addTask()" [disabled]="isProcessing">
      {{isProcessing ? 'Processing...' : 'Add Task'}}
    </button>
  </div>
</form>
```

W razie problemów, porównaj kod: <https://github.com/ideaspot-pl/ai2-lab-d-todo/commit/76ada7324473c9aa60df2e4dc213297f44ad5620>

Wstaw zrzut ekranu listy zadań z wypełnionym formularzem dodawania zadania:

Backend-Based Task Manager


Title Deadline  Add Task

Hello Again World	2000-01-01	<input type="checkbox"/>
Hello World	1900-01-01	<input type="checkbox"/>

To jest stopka

Wstaw rzut ekranu listy zadań po dodaniu zadania:

Backend-Based Task Manager

Title Deadline  Add Task

HI	2023-12-22	<input type="checkbox"/>
Hello Again World	2000-01-01	<input type="checkbox"/>
Hello World	1900-01-01	<input type="checkbox"/>

To jest stopka

Wstaw rzut ekranu kodu `src/app/tasks/tasks.component.ts`:

```

import {Component, OnInit} from '@angular/core';
import {TasksService} from "../tasks.service";
import {Task} from "../task";

@Component({
  selector: 'app-tasks',
  templateUrl: './tasks.component.html',
  styleUrls: ['./tasks.component.css']
})
export class TasksComponent implements OnInit {
  public tasks: Task[] = [];
  public newTask: Task = {};
  public isProcessing = false;

  constructor(private tasksService: TasksService) {}
  ngOnInit() {
    this.tasksService.index().subscribe((tasks) => {
      this.tasks = tasks;
    });
    this.isProcessing = false;
  }
  addTask() {
    if (this.newTask.title === undefined) {
      return;
    }

    this.newTask.completed = false;
    this.newTask.archived = false;

    this.tasks.unshift(this.newTask); // optimistic update; try commenting this line off and compare the difference

    this.tasksService.post(this.newTask).subscribe((task) => {
      this.newTask = {};
      this.ngOnInit();
    });
  }
}

```

Wstaw zrzut ekranu kodu `src/app/tasks/tasks.component.html`:

```

<div class="container">
  <form class="new-task">
    <div class="form-group">
      <label for="title">Title</label>
      <input type="text" id="title" name="title" [(ngModel)]="newTask.title" class="form-control" [disabled]="isProcessing" />
    </div>
    <div class="form-group">
      <label for="deadline">Deadline</label>
      <input type="date" id="deadline" name="title" [(ngModel)]="newTask.deadline" class="form-control" [disabled]="isProcessing" />
    </div>
    <div class="form-group">
      <button type="button" class="btn btn-primary" (click)="addTask()" [disabled]="isProcessing">
        {{isProcessing ? 'Processing...' : 'Add Task'}}
      </button>
    </div>
  </form>

  <div class="tasks">
    <div class="task" *ngFor="let task of tasks">
      <div class="title">{{ task.title }}</div>
      <div class="deadline">
        <span *ngIf="task.deadline">{{ task.deadline }}</span>
      </div>
      <div class="actions">
        <input type="checkbox" />
      </div>
    </div>
  </div>
</div>

```

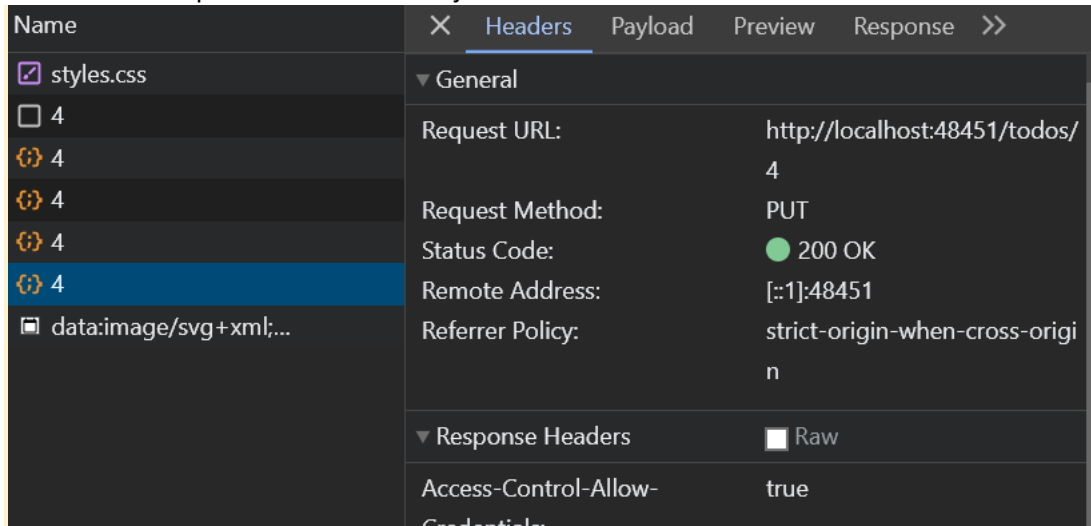
Punkty:	0	1
---------	---	---

Następnym krokiem jest implementacja oznaczania zadań jako zakończone. Powiąż checkboxy z właściwością `completed` dla poszczególnych zadań na liście za pomocą `[(ngModel)]` oraz dociąg zdarzenie (`change`) z nową metodą `handleChange(task: Task)`, która wysyła zaktualizowane zadanie do backendu. Zmień styl zakończonych zadań. Przykładowa implementacja:

```
handleChange(task: Task) : void {
  this.tasksService.put(task).subscribe( observerOrNext: {
    error: err => {
      alert(err);
      this.ngOnInit();
    }
  });
}
```

W razie problemów, porównaj kod: <https://github.com/ideaspot-pl/ai2-lab-d-todo/commit/4d604c8651a31e648841e8f4a651ad8c39b7c3e2>

Wstaw zrzut ekranu żądania `PUT` w narzędziach deweloperskich po zmianie statusu zadania:



Wstaw zrzut ekranu odpowiedniego fragmentu kodu `src/app/tasks/tasks.component.ts`:

```
@Component({
  selector: 'app-tasks',
  templateUrl: './tasks.component.html',
  styleUrls: ['./tasks.component.css']
})
export class TasksComponent implements OnInit {
  public tasks: Task[] = [];
  public newTask: Task = {};
  public isProcessing = false;

  constructor(private tasksService: TasksService) {}
  ngOnInit() {
    this.tasksService.index().subscribe((tasks) => {
      this.tasks = tasks;
    });
    this.isProcessing = false;
  }
  addTask() {
    if (this.newTask.title === undefined) {
      return;
    }
    this.newTask.completed = false;
    this.newTask.archived = false;
    this.tasks.unshift(this.newTask); // optimistic update; try commenting this line off and compare the difference
    this.tasksService.post(this.newTask).subscribe((task) => {
      this.newTask = {};
      this.ngOnInit();
    });
  }

  handleChange(task: Task) {
    this.tasksService.put(task).subscribe({
      error: err => {
        alert(err);
        this.ngOnInit();
      }
    });
  }
}
```

Wstaw zrzut ekranu odpowiedniego fragmentu kodu `src/app/tasks/tasks.component.html`:

```

<div class="container">
  <form class="new-task">
    <div class="form-group">
      <label for="title">Title</label>
      <input type="text" id="title" name="title" [(ngModel)]="newTask.title" class="form-control" [disabled]="isProcessing" />
    </div>
    <div class="form-group">
      <label for="deadline">Deadline</label>
      <input type="date" id="deadline" name="title" [(ngModel)]="newTask.deadline" class="form-control" [disabled]="isProcessing" />
    </div>
    <div class="form-group">
      <button type="button" class="btn btn-primary" (click)="addTask()" [disabled]="isProcessing">
        {{isProcessing ? 'Processing...' : 'Add Task'}}
      </button>
    </div>
  </form>

  <div class="tasks">
    <div class="task" *ngFor="let task of tasks">
      <div class="title">{{ task.title }}</div>
      <div class="deadline">
        <span *ngIf="task.deadline">{{ task.deadline }}</span>
      </div>
      <div class="actions">
        <input type="checkbox" [(ngModel)]="task.completed" (change)="handleChange(task)" />
      </div>
    </div>
  </div>
</div>

```

Punkty:

0

1

Na koniec implementacji komponentu `TasksComponent`, na widoku dodaj przycisk `Archive Completed`, powiązany z metodą `archiveCompleted`, która zarchiwizuje wszystkie zadania oznaczone jako skończone, tj. ustawi ich pole `archived` na wartość `true` i wyśle aktualizację do backendu. Z wykorzystaniem `forkJoin` poczekaj aż wszystkie aktualizacje się zakończą i odśwież komponent ponownie wywołując `ngOnInit()`.

Przykładowa implementacja:

```

archiveCompleted() : void {
  const observables: Observable<any>[] = [];
  for (const task : Task of this.tasks) {
    if (!task.completed) {
      continue;
    }

    task.archived = true;
    observables.push(this.tasksService.put(task));
  }

  // refresh page when all updates finished
  forkJoin(observables).subscribe( observerOrNext: () : void => {
    this.ngOnInit();
  });
}

```







Wstaw zrzut ekranu zawartości pliku `todos.json` przed archiwizacją:

```
{
  "todos": [
    {
      "id": 1,
      "title": "Hello World",
      "deadline": "1900-01-01",
      "completed": false,
      "archived": false
    },
    {
      "id": 2,
      "title": "Hello Again World",
      "deadline": "2000-01-01",
      "completed": true,
      "archived": false
    },
    {
      "title": "Pierwsze zadanie",
      "deadline": "2022-01-28",
      "completed": true,
      "archived": true,
      "id": 3
    },
    {
      "title": "HI",
      "deadline": "2023-12-22",
      "completed": true,
      "archived": false,
      "id": 4
    }
  ]
}
```

Wstaw zrzut ekranu zawartości pliku `todos.json` po archiwizacji:

```
{
  "todos": [
    {
      "id": 1,
      "title": "Hello World",
      "deadline": "1900-01-01",
      "completed": false,
      "archived": false
    },
    {
      "id": 2,
      "title": "Hello Again World",
      "deadline": "2000-01-01",
      "completed": true,
      "archived": true
    },
    {
      "title": "Pierwsze zadanie",
      "deadline": "2022-01-28",
      "completed": true,
      "archived": true,
      "id": 3
    },
    {
      "title": "HI",
      "deadline": "2023-12-22",
      "completed": true,
      "archived": true,
      "id": 4
    }
  ]
}
```

Wstaw zrzut ekranu zakładki **Network** przeglądarki, obrazujący wysłanie wielu zapytań **PUT** po kliknięciu na przycisk archiwizacji zadań:

Name	✕ Headers	Payload	Preview	Response >>
<input checked="" type="checkbox"/> styles.css	▼ General			
<input type="checkbox"/> 1	Request URL:		http://localhost:48451/todos?	
 1			archived=false&_sort=id&_or	
 1			der=desc	
<input type="checkbox"/> 1	Request Method:		GET	
 1	Status Code:		● 200 OK	
 1	Remote Address:		[::1]:48451	
 todos?archived=false&_sort=i...	Referrer Policy:		strict-origin-when-cross-origi	
			n	
	▼ Response Headers		<input type="checkbox"/> Raw	
	Access-Control-Allow-		true	

Wstaw zrzut ekranu odpowiedniego fragmentu kodu `src/app/tasks/tasks.component.ts`:

```
public isProcessing = false;

constructor(private tasksService: TasksService) {}
ngOnInit() {
  this.tasksService.index().subscribe((tasks) => {
    this.tasks = tasks;
  });
  this.isProcessing = false;
}
addTask() {
  if (this.newTask.title === undefined) {
    return;
  }
  this.newTask.completed = false;
  this.newTask.archived = false;
  this.tasks.unshift(this.newTask); // optimistic update; try commenting this line off and compare the difference
  this.tasksService.post(this.newTask).subscribe((task) => {
    this.newTask = task;
    this.ngOnInit();
  });
}

handleChange(task: Task) {
  this.tasksService.put(task).subscribe({
    error: err => {alert(err); this.ngOnInit();}});
}

archiveCompleted() {
  const observables: Observable<any>[] = [];
  for (const task of this.tasks) {
    if (!task.completed) {continue;}
    task.archived = true;
    observables.push(this.tasksService.put(task));
  }
  forkJoin(observables).subscribe(() => {this.ngOnInit();});
}
```

Wstaw zrzut ekranu odpowiedniego fragmentu kodu `src/app/tasks/tasks.component.html`:

```

<div class="container">
  <form class="new-task">
    <div class="form-group">
      <label for="title">Title</label>
      <input type="text" id="title" name="title" [(ngModel)]="newTask.title" class="form-control" [disabled]="isProcessing" />
    </div>
    <div class="form-group">
      <label for="deadline">Deadline</label>
      <input type="date" id="deadline" name="title" [(ngModel)]="newTask.deadline" class="form-control" [disabled]="isProcessing" />
    </div>
    <div class="form-group">
      <button type="button" class="btn btn-primary" (click)="addTask()" [disabled]="isProcessing">
        {{isProcessing ? 'Processing...' : 'Add Task'}}
      </button>
    </div>
  </form>

  <div class="tasks">
    <div class="task" *ngFor="let task of tasks">
      <div class="title">{{ task.title }}</div>
      <div class="deadline">
        <span *ngIf="task.deadline">{{ task.deadline }}</span>
      </div>
      <div class="actions">
        <input type="checkbox" [(ngModel)]="task.completed" (change)="handleChange(task)" />
      </div>
    </div>
  </div>
  <button (click)="archiveCompleted()">Archive Completed</button>
</div>

```

Wstaw zrzut ekranu odpowiedniego fragmentu kodu `src/app/tasks/tasks.component.css`:

```
.container {
  max-width: 800px;
  margin: 20px auto;
  padding: 0 10px;
}

.new-task {
  display: grid;
  grid-template-columns: 1fr;
  grid-gap: 10px;
  margin-bottom: 20px;
}

.new-task .form-group {
  display: flex;
  flex-direction: column;
}

@media screen and (min-width: 768px) {
  .new-task {
    grid-template-columns: 1fr 1fr auto;
  }

  .new-task .form-group {
    display: flex;
    flex-direction: row;
    justify-content: space-between;
    gap: 10px;
  }

  .new-task .form-group .form-control {
    flex-grow: 1;
  }
}
```

```

.tasks .task {
  background-color: white;
  border: 1px solid black;
  margin-bottom: 10px;
  padding: 10px;

  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-areas: 'deadline actions' 'title title';
  grid-row-gap: 10px;
}

@media screen and (min-width: 768px) {
  .tasks .task {
    grid-template-columns: 3fr 1fr auto;
    grid-template-areas: 'title deadline actions';
  }
}

.tasks .task .title {
  grid-area: title;
}

.tasks .task .deadline {
  grid-area: deadline;
}

.tasks .task .deadline span {
  padding: 5px;
  background-color: orange;
  border-radius: 8px;
  font-size: 0.8em;
}

.tasks .task .actions {
  grid-area: actions;
  text-align: right;
}

```

Punkty:	0	1
---------	---	---

IMPLEMENTACJA KOMPONENTU ARCHIVECOMPONENT

W tej części zaimplementujemy komponent `ArchiveComponent` odpowiedzialny za wyświetlanie archiwum zadań oraz ich kasowanie.

Wygeneruj komponent `ArchiveComponent` używając polecenia terminala:

```
> ng generate component archive --skip-tests
```

Utworzony zostanie katalog `src/app/archive`. Wstrzyknij do niego serwis `TaskService` i zaimplementuj metodę `ngOnInit()`, tak żeby podczas inicjalizacji komponent wypełnił się wyłącznie zarchiwizowanymi zadaniami. Zaimplementuj metodę `delete(task: Task)`, która permanentnie usunie zadanie, wykorzystując serwis `TaskService`.

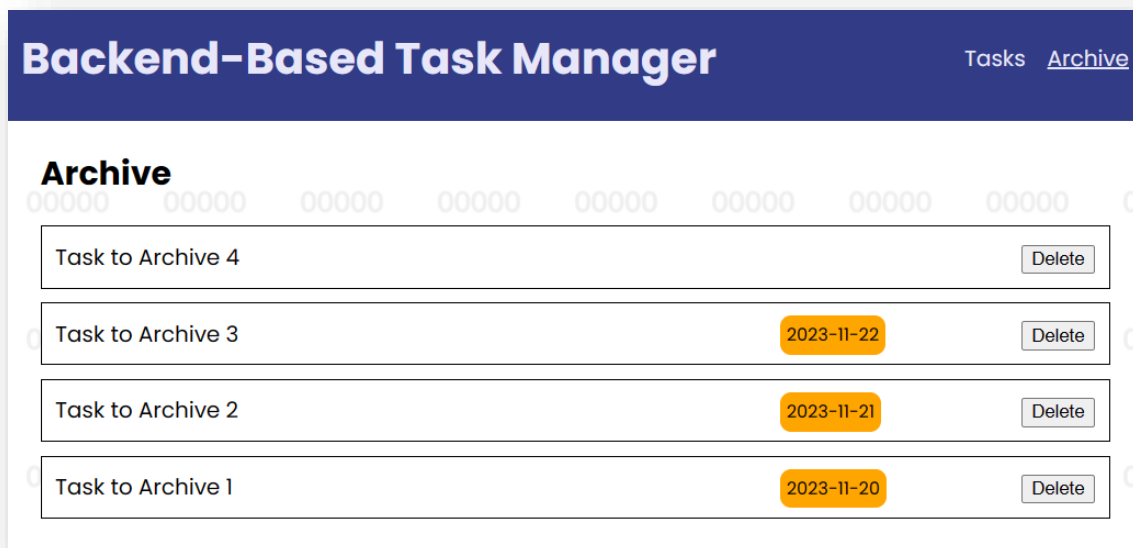
Dodaj do routingu ścieżkę do archiwum:

```
const routes: Routes = [  
  {path: 'tasks', component: TasksComponent},  
  {path: 'archive', component: ArchiveComponent},  
  {path: '', redirectTo: '/tasks', pathMatch: 'full'},  
];
```

Dodaj do widoku głównego komponentu `AppComponent` łącza do obydwu widoków.

Zaimplementuj widok listy archiwum wraz z przyciskami `Delete` przy każdym zadaniu, odpowiedzialnymi za permanentne kasowanie zadania.

Przykładowa implementacja:



```
export class ArchiveComponent implements OnInit {
  public tasks: Task[] = [];

  no usages
  constructor(
    private tasksService: TasksService
  ) {
  }

  no usages
  ngOnInit(): void {
    this.tasksService.index({ archived: true }).subscribe( observerOrNext: (tasks: Task[]) : void => {
      this.tasks = tasks;
    });
  }

  1 usage
  delete(task: Task): void {
    if (!confirm("Are you sure?")) {
      return;
    }

    this.tasksService.delete(task).subscribe( observerOrNext: () : void => {
      this.ngOnInit();
    });
  }
}
```

```
<div class="container">
  <h2>Archive</h2>
  <div class="tasks">
    <div class="task" *ngFor="let task of tasks">
      <div class="title">{{ task.title }}</div>
      <div class="deadline">
        <span *ngIf="task.deadline">{{ task.deadline }}</span>
      </div>
      <div class="actions">
        <button (click)="delete(task)">Delete</button>
      </div>
    </div>
  </div>
</div>
```

```
<nav>
  <a routerLink="/tasks" routerLinkActive="active">Tasks</a>
  <a routerLink="/archive" routerLinkActive="active">Archive</a>
</nav>
```

W przypadku problemów, porównaj kod: <https://github.com/ideaspot-pl/ai2-lab-d-todo/commit/0b5722430094c50313b5a87f9d5cdbbb56a451d3>

Wstaw zrzut ekranu strony archiwum wypełnionej zadaniami:

Backend-Based Task Manager

Archive

HI	2023-12-22	Delete
Pierwsze zadanie	2022-01-28	Delete
Hello Again World	2000-01-01	Delete
Hello World	1900-01-01	Delete

To jest stopka

Wstaw zrzut ekranu zawartości pliku `todos.json` przed kasowaniem archiwalnych zadań:

```
{
  "todos": [
    {
      "id": 1,
      "title": "Hello World",
      "deadline": "1900-01-01",
      "completed": true,
      "archived": true
    },
    {
      "id": 2,
      "title": "Hello Again World",
      "deadline": "2000-01-01",
      "completed": true,
      "archived": true
    },
    {
      "title": "Pierwsze zadanie",
      "deadline": "2022-01-28",
      "completed": true,
      "archived": true,
      "id": 3
    },
    {
      "title": "HI",
      "deadline": "2023-12-22",
      "completed": true,
      "archived": true,
      "id": 4
    }
  ]
}
```

Wstaw zrzut ekranu strony archiwum po skasowaniu archiwalnych zadań:

Backend-Based Task Manager

Archive

Hello World	1900-01-01	Delete
-------------	------------	--------

To jest stopka

Wstaw zrzut ekranu zawartości pliku `todos.json` po skasowaniu archiwalnych zadań:

```
{
  "todos": [
    {
      "id": 1,
      "title": "Hello World",
      "deadline": "1900-01-01",
      "completed": true,
      "archived": true
    }
  ]
}
```

Wstaw zrzut ekranu odpowiedniego fragmentu kodu `src/app/archive/archive.component.ts`:

```
import {Component, OnInit} from '@angular/core';
import {TasksService} from "../tasks.service";
import {Task} from "../task";

@Component({
  selector: 'app-archive',
  templateUrl: './archive.component.html',
  styleUrls: ['./archive.component.css']
})
export class ArchiveComponent implements OnInit {
  public tasks: Task[] = [];

  constructor(
    private tasksService: TasksService
  ) {
  }

  ngOnInit() {
    this.tasksService.index(true).subscribe((tasks) => {
      this.tasks = tasks;
    });
  }

  delete(task: Task) {
    if (!confirm('Are you sure?')) {
      return;
    }

    this.tasksService.delete(task).subscribe(() => {
      this.ngOnInit();
    });
  }
}
```

Wstaw zrzut ekranu odpowiedniego fragmentu kodu `src/app/archive/archive.component.html`:

```
<div class="container">
  <h2>Archive</h2>
  <div class="tasks">
    <div class="task" *ngFor="let task of tasks">
      <div class="title">{{ task.title }}</div>
      <div class="deadline">
        <span *ngIf="task.deadline">{{ task.deadline }}</span>
      </div>
      <div class="actions">
        <button (click)="delete(task)">Delete</button>
      </div>
    </div>
  </div>
</div>
```

Wstaw zrzut ekranu odpowiedniego fragmentu kodu `src/app/archive/archive.component.css`:

```

.container {
  max-width: 800px;
  margin: 20px auto;
  padding: 0 10px;
}

.tasks .task {
  background-color: white;
  border: 1px solid black;
  margin-bottom: 10px;
  padding: 10px;

  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-areas: 'deadline actions' 'title title';
  grid-row-gap: 10px;
}

@media screen and (min-width: 768px) {
  .tasks .task {
    grid-template-columns: 3fr 1fr auto;
    grid-template-areas: 'title deadline actions';
  }
}

.tasks .task .title {
  grid-area: title;
}

.tasks .task .deadline {
  grid-area: deadline;
}

.tasks .task .deadline span {
  padding: 5px;
  background-color: orange;
  border-radius: 8px;
  font-size: 0.8em;
}

.tasks .task .actions {
  grid-area: actions;
  text-align: right;
}

```

Punkty:

0

1

COMMIT PROJEKTU DO GIT

Utwórz repozytorium publiczne GitHub na tę część kursu. Wyślij swój projekt do repozytorium (push). Upewnij się, czy wszystko dobrze się wysłało. Jeśli tak, to z poziomu przeglądarki utwórz branch o nazwie `lab-d` na podstawie bieżącej gałęzi kodu.

W zależności od przyjętej konwencji (jedno repo per laboratorium kontra jedno repo na wszystkie laboratoria), konieczne może być usunięcie katalogu `.git` i ponowna samodzielna inicjalizacja.

Podaj link do brancha `lab-d` w swoim repozytorium:

https://github.com/Dz229/AI2_Angular/tree/lab-d

PODSUMOWANIE

W kilku zdaniach podsumuj zdobyte podczas tego laboratorium umiejętności.

Nauczyłem się obsługi routingów oraz ich tworzenie.

Nauczyłem się komunikacji z backendem aplikacji.

Zweryfikuj kompletność sprawozdania. Utwórz PDF i wyślij w terminie.