Politechnika Białostocka

Wydział Informatyki

PROGRAMOWANIE APLIKACJI W JĘZYKU JAVASCRIPT

Temat projektu: **Zarządzanie grafikiem pracowników**

Wykonujący:

- **Łukasz Kaliszewicz**
- **Zuzanna Kalinowska**
- **Patrycja Żadziłko**

Studia dzienne

Kierunek: **Informatyka**

Rok: **III**

Semestr: **V**

Grupa zajęciowa: **PS7**

Prowadzący ćwiczenie: **inż. Kasper Jan Seweryn-Ładyński**

Data wykonania ćwiczenia: **18.01.2024 r.**

1. OPIS APLIKACJI

Aplikacja służy do ustalania i zarządzania grafikami pracowników. Pozwala na dodawanie, usuwanie oraz modyfikację pracowników w bazie danych. Pracownicy mają dostęp do swoich harmonogramów, podczas gdy managerowie mogą wyświetlać wspólny grafik, modyfikować godziny pracy pracowników i dodawać dla grafiki indywidualnych pracowników. Aplikacja wspomaga planowanie pracy i efektywne zarządzanie zasobami ludzkimi.

2. FUNKCJONALNOŚĆ APLIKACJI

- Dodanie pracownika do bazy pracowników (z walidacją)
- Usunięcie pracownika z bazy
- Edycja pracownika (z walidacją)
- Wyświetlanie własnego grafiku przez pracownika
- Możliwość wyświetlenia wspólnego grafiku przez managera
- Chat umożliwiający komunikację między pracownikami
- Powiadomienia na pulpicie po otrzymaniu nowej wiadomości
- Własna implementacja backendu w node.js

3. DOKUMENTACJA ELEMENTÓW TECHNICZNYCH

API serwera

Jako API serwera wykorzystujemy JSON Server. Zdecydowaliśmy się na użycie go, gdyż jest to bardzo dobre narzędzie do testowego wytwarzania oprogramowania, które służy jako baza danych i przechowuje dane w formacie JSON. Kod obsługujący żądania do serwera znajduje się w pliku “workers.service.ts”, a seedowane obiekty znajdują się w pliku “db.json”.

Wybrany przez autorów, szczególnie ciekawy fragment kodu.

Ciekawym, nadprogramowym fragmentem kodu jest własny walidator, który sprawdza, czy pierwsze litery słów we frazie (lub pojedynczym słowie) są wielkie, a pozostałe są małe. Walidator ten używany jest m. in. do imion, nazwisk i nazw miejscowości.

Wypunktowane elementy techniczne, które zostały zrealizowane w projekcie wraz z krótkim komentarzem odnośnie realizacji: jak zrealizowano i w którym pliku.

Główną klasą naszej aplikacji jest klasa WorkerClass.ts, która przechowuje informacje o pracownikach oraz ich zmianach/godzinach pracy.

```
//imports
import { ShiftClass } from "./ShiftClass";


export class WorkerClass{


  constructor(private id: number, private haslo: string,
              private imie: string, private nazwisko: string,
              private ulica: string, private numerDomu: number, private miasto: string, private kodPocztowy: string,
              private numerTelefonu: number, private email: string,
              private stanowiskoId?: number, private dzialID?: number,
              private shift?: ShiftClass){
  }


  get Id():number{return this.id;}
  get Haslo():string{return this.haslo};
  get Imie():string{return this.imie;}
  get Nazwisko():string{return this.nazwisko;}
  get Ulica():string{return this.ulica;}
  get NumerDomu():number{return this.numerDomu;}
  get Miasto():string{return this.miasto;}
  get KodPocztowy():string{return this.kodPocztowy;}
  get NumerTelefonu():number{return this.numerTelefonu;}
  get Email():string{return this.email;}
  get StanowiskoId():number | undefined{
    return this.stanowiskoId;
  }
  get DzialID():number | undefined{
    return this.dzialID;
  }
  get Shift():ShiftClass | undefined{
    return this.shift;
  }

  set Id(id: number){this.id = id;};
  set Haslo(haslo:string){this.haslo = haslo};
  set Email(email:string){this.email = email};
  set NumerDomu(numerDomu: number){this.numerDomu = numerDomu;}
  set NumerTelefonu(numerTelefonu: number){this.numerTelefonu = numerTelefonu;}
  set StanowiskoId(stanowiskoId: number){this.stanowiskoId = stanowiskoId;}
  set DzialID(dzialID: number){this.dzialID = dzialID;}
  set Imie(imie: string){this.imie = imie;}
  set Nazwisko(nazwisko: string){this.nazwisko = nazwisko;}
  set Ulica(ulica: string){this.ulica = ulica;}
  set Miasto(miasto: string){this.miasto = miasto;}
  set KodPocztowy(kodPocztowy: string){this.kodPocztowy = kodPocztowy;}
  set Shift(shift: ShiftClass){this.shift = shift;}


}
```

W komponencie add-worker wykorzystujemy formularz z walidacją. Za jego pomocą dodajemy pracownika, wpisując w pola jego dane.

```
<div id="loginDiv">
  <h3>Dodaj pracownika</h3>
  <div class="alert alert-success" role="alert" *ngIf="this.succesStateFlag == true">
    Poprawnie dodano pracownika <button (click)="this.succesStateFlag = false">X</button>
  </div>
  <form class="form-column" [formGroup]="addWorker" (ngSubmit)="SaveData()">
    <input type="text" class="form-control mb-2 mr-sm-2" id="inlineFormInputName2" name="imie" formControlName="imie" placeholder="Imię" >
    <div class="error" *ngIf="addWorker.controls['imie'].errors?.['minlength']">
      Imię musi mieć więcej, niż 2 znaki.
    </div>
    <div class="error" *ngIf="addWorker.controls['imie'].errors?.['maxlength']">
      Imię nie może mieć więcej, niż 50 znaków.
    </div>
    <div class="error" *ngIf="addWorker.controls['imie'].errors?.['required']">
      Imię jest wymagane.
    </div>
    <div class="error" *ngIf="addWorker.controls['imie'].errors?.['capitalizedFirstLettersError']">
      {{addWorker.controls['imie'].errors?.['capitalizedFirstLettersError']}}
    </div>

    <input type="text" class="form-control mb-2 mr-sm-2" id="inlineFormInputName2" name="nazwisko" formControlName="nazwisko" placeholder="Nazwisko" >
    <div class="error" *ngIf="addWorker.controls['nazwisko'].errors?.['required']">
      Nazwisko jest wymagane.
    </div>
    <div class="error" *ngIf="addWorker.controls['nazwisko'].errors?.['minlength']">
      Nazwisko musi mieć więcej, niż 2 znaki.
    </div>
    <div class="error" *ngIf="addWorker.controls['nazwisko'].errors?.['maxlength']">
      Nazwisko nie może mieć więcej, niż 100 znaków.
    </div>
    <div class="error" *ngIf="addWorker.controls['nazwisko'].errors?.['capitalizedFirstLettersError']">
      {{addWorker.controls['nazwisko'].errors?.['capitalizedFirstLettersError']}}
    </div>

    <input type="text" class="form-control mb-2 mr-sm-2" id="inlineFormInputName2" name="email" formControlName="email" placeholder="Adres Email" >
    <div class="error" *ngIf="addWorker.controls['email'].errors?.['required']">
      Email jest wymagany.
    </div>
    <div class="error" *ngIf="addWorker.controls['email'].errors?.['pattern']">
      Email jest niepoprawny.
    </div>

    <input type="text" class="form-control mb-2 mr-sm-2" id="inlineFormInputName2" name="haslo" formControlName="haslo" placeholder="Hasło" >
    <div class="error" *ngIf="addWorker.controls['haslo'].errors?.['required']">
      Hasło jest wymagane.
    </div>
    <div class="error" *ngIf="addWorker.controls['haslo'].errors?.['minlength']">
      Hasło musi mieć więcej, niż 8 znaków.
    </div>

    <input type="text" class="form-control mb-2 mr-sm-2" id="inlineFormInputName2" name="ulica" formControlName="ulica" placeholder="Nazwa Ulicy" >
    <div class="error" *ngIf="addWorker.controls['ulica'].errors?.['required']">
      Ulica jest wymagana.
    </div>
    <div class="error" *ngIf="addWorker.controls['ulica'].errors?.['minlength']">
      Ulica musi mieć więcej, niż 2 znaki.
    </div>
    <div class="error" *ngIf="addWorker.controls['ulica'].errors?.['maxlength']">
```

Walidację stosujemy przy dodawaniu i edycji pracowników. Fragment komponentu add-worker.component.ts, w którym polom klasy WorkerClass.ts przypisywane są odpowiednie walidatory:


src > app > add-worker > add-worker.component.ts > AddWorkerComponent > addWorker
addWorker = new FormGroup({
imie: new FormControl('', [
Validators.required,
Validators.minLength(2),
Validators.maxLength(50),
capitalizedFirstLettersValidator()
]),
nazwisko: new FormControl('', [
Validators.required,
Validators.minLength(2),
Validators.maxLength(100),
capitalizedFirstLettersValidator()
]),
email: new FormControl('', [
Validators.required,
Validators.pattern('[A-Za-z0-9]+@[A-Za-z0-9]+\.[A-Za-z0-9]+'),
]),
haslo: new FormControl('', [
Validators.required,
Validators.minLength(8),
]),
ulica: new FormControl('', [
Validators.required,
Validators.minLength(2),
Validators.maxLength(100),
capitalizedFirstLettersValidator()
]),
numerDomu: new FormControl('', [
Validators.required,
Validators.min(0),
Validators.max(9999)
]),
miasto: new FormControl('', [
Validators.required,
Validators.minLength(2),
Validators.maxLength(100),
capitalizedFirstLettersValidator()
]),
kodPocztowy: new FormControl('', [
Validators.required,
Validators.pattern('[0-9][0-9]-[0-9][0-9][0-9]')
]),
numerTelefonu: new FormControl('', [
Validators.required,
Validators.min(100000000),
Validators.max(999999999)
]),
stanowidkoId: new FormControl('', [
Validators.min(0),
Validators.max(9999),
]),
dzialId: new FormControl('', [
Validators.min(0),
Validators.max(9999),

W serwisie workers.service.ts wykorzystywane są 4 rodzaje żądań http: GET, POST, PUT i DELETE.

```ts
export class WorkersService {

  url = "http://localhost:3000/workers/";
  urlManagers = "http://localhost:3000/managers/";

  constructor(private http:HttpClient) { }

  getAllWorkers(){
    return this.http.get(this.url);
  }

  getAllManagers(){
    return this.http.get(this.urlManagers);
  }

  saveWorkerData(data:any){
    console.log(data);
    return this.http.post(this.url, data);
  }

  deleteWorker(id:number){
    return this.http.delete(`${this.url}/${id}`)
  }

  updateWorkerData(id: number, data:any){
    return this.http.put(`${this.url}/${id}`, data);
  }
}
```

Zaimplementowany został własny filtr worker-search.pipe.ts, który wyszukuje pracowników z listy po nazwiskach. Używany jest on w komponencie list-workers.

src > app > pipes > TS worker-search.pipe.ts > WorkerSearchPipe

```ts
import { Pipe, PipeTransform } from '@angular/core';
import { WorkerClass } from '../model/WorkerClass';

@Pipe({
  name: 'workerSearch',
})
export class WorkerSearchPipe implements PipeTransform {

  transform(list: WorkerClass[], ...args: string[]): WorkerClass[] {
    let searchBy = args[0];
    let criteria=args[1];
    console.log('criteria',criteria)
    if(criteria==undefined) criteria='nazwisko';
    searchBy=searchBy.toLowerCase();
    if (!searchBy) { return list; }
    const filteredList = list.filter(el => {
      if(criteria=='nazwisko'){
        if (el.Nazwisko.toLowerCase().includes(searchBy)) { return el; }
      }
      return null;
    });
    return filteredList;
  }

}
```

Wykonanie własnej implementacji backendu w node.js. Napisano prostą implementację backendu w node.js, która dostarcza czas zalogowania do systemu przez użytkownika.

Kod node.js w pliku index.js

src > nodejs > JS index.js > ...

```
const express = require('express');
const cors = require('cors');
const app = express();
const port = 3002;

app.use(cors());

app.get('/current-time', (req, res) => {
  const currentTime = new Date().toLocaleTimeString();
  res.json({ time: currentTime });
});

app.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
});
```

Kod serwisu obsługującego backend node.js

Serwis: node-service

src > app > TS node-service.service.ts > ...

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

@Injectable({
  providedIn: 'root',
})
export class NodeService {
  private baseUrl = 'http://localhost:3002';

  constructor(private http: HttpClient) {}

  getCurrentTime(): Observable<any> {
    return this.http.get(`${this.baseUrl}/current-time`);
  }
}
```

4. DODATKOWE BIBLIOTEKI

W projekcie wykorzystana została biblioteka JSON Server, która umożliwia stworzenie serwisu bazy danych przechowujących dane w formacie JSON.