Zadania na ćwiczenia nr 3

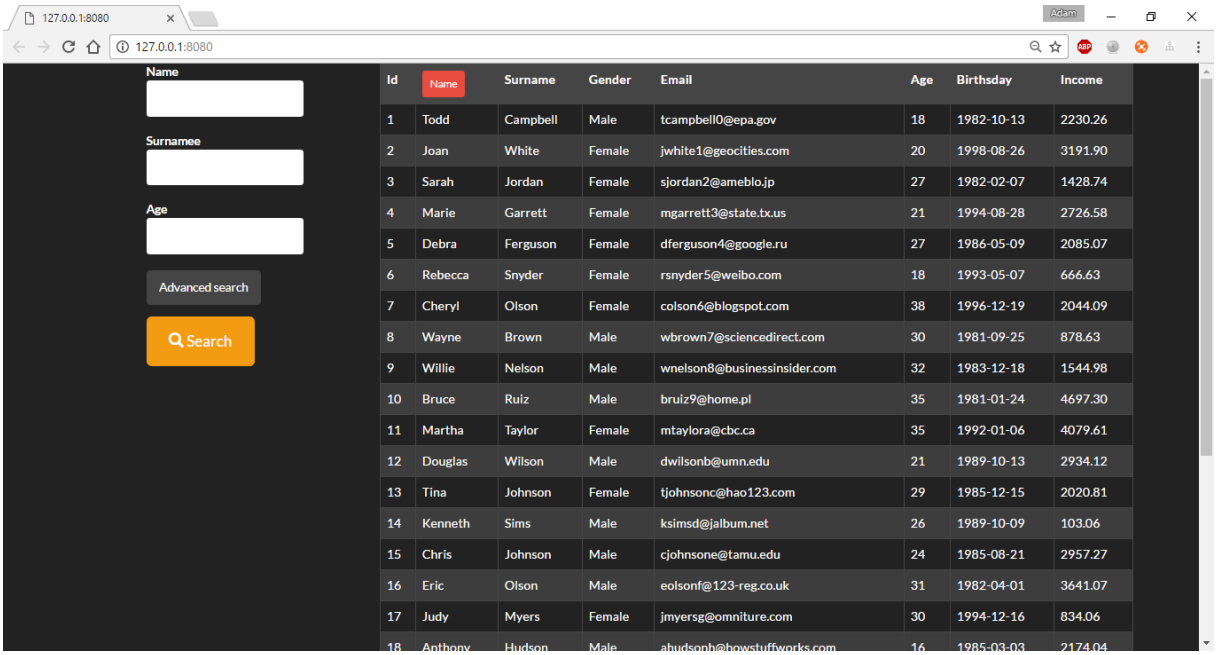
Cele ćwiczenia:

- Zapoznanie się z podstawowymi aspektami programowania w TypeScripcie
- Zapoznanie się projektowaniem klas w TypeScripcie

1. Zapoznanie się z projektem przykładowym.
   - Ściągnij i rozpakuj projekt z materiałów do ćwiczeń 3 z portalu Edux
   - Uruchom go

```
>npm install
>npm run go
```

Node uruchomi aplikację na adresie http://localhost:8080



2. Naszym zadaniem będzie przepisanie skryptów JS na skrypty TS.
   Zacznijmy od przepisania pliku Person.js – ten skrypt zawiera konstruktor, który na podstawie json'a tworzy obiekt, oraz funkcję toTableRow, która generuje rząd w tabelce HTML. W TypeScripcie utworzymy klasę z identycznymi polami oraz funkcjami.

   Do katalogu scripts dodaj plik o nazwie person-list-item.ts i zdefiniuj w nim klasę PersonListItem

```
export class PersonListItem {

    public id: number;
    public age: number;
    public firstname: string;
    public lastname: string;
    public gender: string;
    public email: string;
    public birthsday: Date
```

```
    public income: number;

    public toTableRow(): string {
        return '<tr><td>'
            + this.id
            + '</td><td>'
            + this.firstname
            + '</td><td>'
            + this.lastname
            + '</td><td>'
            + this.gender
            + '</td><td>'
            + this.email
            + '</td><td>'
            + this.age
            + '</td><td>'
            + this.birthsday.toISOString()
            + '</td><td>'
            + this.income
            + '</td></tr>'
    }
}
```

Słówko kluczowe ‘export’ użyte w deklaracji klasy (ale także interfejsu, funkcji, zmiennej, obiektu) pozwala na importowanie tejże klasy w innych skryptach ts.

3. Kolejnym krokiem będzie utworzenie klasy, która będzie dostarczać danych z tablicy złożonej z obiektów json, które przechowują dane o osobach. Zacznijmy od utworzenia podkatalogu w katalogu scripts o nazwie services– dodajmy do niego dwa pliki : data.ts (plik który w postaci zmiennej będzie przechowywać dane o osobach) oraz person-service.ts

W pliku data.ts zdefiniujmy interfejs który pomorze nam w wygodnym mapowaniu obiektu json na obiekt klasy PersonListItem

```
export interface IPersonJson {
    id: number;
    gender: string;
    firstName: string;
    lastName: string;
    email: string;
    income: string;
    birthsday: string;
    age: number;
}
```

oraz zmienną people która będzie przetrzymywać tablicę json’ów (dane przekopiuj z pliku DATA.js)

```
export var people: Array<IPersonJson> = [
    {
```

```
        "id": 1,
        "gender": "Male",
        "firstName": "Todd",
        "lastName": "Campbell",
        "email": "tcampbell0@epa.gov",
        "income": "2230.26",
        "birthsday": "1982-10-13",
        "age": 18
    },
```

Skrypt person-service.ts zacznijmy od deklaracji importów typów i zmiennych których użyjemy w skrypcie

```
import { people } from './data'; //-> import danych o osobach
import { PersonListItem } from "../person-list-item";//-> import klasy
```

Ze względu na dużą ilość danych przyda nam się klasa która pozwoli nam na stronicowanie wyników

```
export class PagingInfo {
    constructor(public page: number, public count: number) {
    }
}
```

Zwróć uwagę na inny sposób definiowania pól w klasie – tym razem zrobiliśmy to przez konstruktor klasy.

Użycie modyfikatorów dostępu w parametrach konstruktora automatycznie utworzy nam odpowiednie pola w obiekcie klasy.

Teraz przejdźmy do implementacji właściwej klasy PersonService

```
export class PersonService {

    public getPeople(pagingInfo: PagingInfo): Array<PersonListItem> {

        let begin = pagingInfo.page - 1;
        if (begin < 0) begin = 0;

        return people
            .slice(begin * pagingInfo.count,
                    begin * pagingInfo.count + pagingInfo.count)
                //-> z zaimportowanej kolekcji wybieramy stronę wyników
            .map(x => {
                let person = new PersonListItem();
                person.firstname = x.firstName;
                person.lastname = x.lastName;
                person.gender = x.gender;
                person.email = x.email;
                person.income = +x.income
                person.age = x.age;
```

```
                person.birthsday = new Date(x.birthsday);
                person.id = x.id;
                return person;
            });
            //-> pobrane wyniki mapujemy na obiekty PersonListItem
    }
}
```

Na koniec skryptu dodajmy jeszcze dwie linijki aby móc sprawdzić działanie powyższej klasy

```
var service = new PersonService();

console.log(service.getPeople(new PagingInfo(1, 5)));
```

teraz jeśli z terminala wejdziesz do katalogu ze skryptem to komendą ts-node uruchomisz powyższy skrypt

```
src\scripts\services> ts-node .\person-service.ts
```

Pełny kod skryptu person-service.ts

```
import { people } from './data'; //-> import danych o osobach
import { PersonListItem } from "../person-list-item";//-> impoert klasy
PersonListItem

export class PagingInfo {
    constructor(public page: number, public count: number) {
    }
}

export class PersonService {

    public getPeople(pagingInfo: PagingInfo): Array<PersonListItem> {

        let begin = pagingInfo.page - 1;
        if (begin < 0) begin = 0;

        return people
            .slice(begin * pagingInfo.count,
                    begin * pagingInfo.count + pagingInfo.count)
                //-> z zaimportowanej kolekcji wybieramy stronę wynikow
            .map(x => {
                let person = new PersonListItem();
                person.firstname = x.firstName;
                person.lastname = x.lastName;
                person.gender = x.gender;
                person.email = x.email;
                person.income = +x.income
                person.age = x.age;
```

```
                person.birthsday = new Date(x.birthsday);
                person.id = x.id;
                return person;
            });
            //-> pobrane wyniki mapujemy na obiekty PersonListItem
    }
}

var service = new PersonService();

console.log(service.getPeople(new PagingInfo(1, 5)));
```

4. Teraz przepiszmy skrypty ListOfPeople.js oraz PeopleTableViewModel.js na TypeScripta – w tym celu utworzymy dwa pliki: person-list.ts oraz person-table.ts w katalogu scripts.

person-list.ts

```
import { PersonListItem } from "./person-list-item";

export class PersonList {

    private _people : Array<PersonListItem> = [];
    public get people() : Array<PersonListItem> {
        return this._people;
    }
    public set people(v : Array<PersonListItem>) {
        this._people = v;
    }

    public toTable(): string {
        var table = '<table class="table table-striped table-hover table-
bordered">';
        table += this.generateTableHeader();
        this._people.forEach(person => table += person.toTableRow());
        table += '</table>'
        return table;
    }

    public clear() {
        this._people = [];
    }

    private generateTableHeader(): string {
        return '<tr><th>Id</th>'
            + ' <th><button  class="btn btn-sm btn-danger js-sort-
name">Name</button></th>'
            + ' <th>Surname</th>'
            + ' <th>Gender</th>'
```

```
            + ' <th>Email</th>'
            + ' <th>Age</th>'
            + ' <th>Birthsday</th>'
            + ' <th>Income</th>'
            + '</tr>'
    }
}
```

Zauważ, że TypeScript pozwala na definiowanie właściwości dostępu do pól (gettery i settery)

person-table.ts

```
import { PersonList } from "./person-list";
import { PersonService, PagingInfo } from "./services/person-service";

export class PersonTable{

    constructor(public context: JQuery) {
    }

    personService = new PersonService()
    list = new PersonList();
    currentPage = 0;
    pageSize = 10;

    public next() {
        this.list.clear();
        this.currentPage++;
        this.list.people = this.personService.getPeople(new
PagingInfo(this.currentPage, this.pageSize));
        this.refreshTable();
    }

    public prev(): void {
        this.list.clear();
        if (this.currentPage <= 1) return;
        this.currentPage--;
        this.list.people = this.personService.getPeople(new
PagingInfo(this.currentPage, this.pageSize));
        this.refreshTable();
    }

    private refreshTable() {
        this.context.html(this.list.toTable());
    }
}
```

Zwróć uwagę, że w konstruktorze klasy zadeklarowaliśmy parametr context o typie JQuery – możemy to zrobić dzięki importowi modułu dla noda w projekcie '@types/jquery' (patrz plik package.json). Import tego modułu pozwala na stosowanie biblioteki JQuery w naszych skryptach. Warto odwiedzić stronkę do wyszukiwania definicji typów dla typescripta – dzięki nim możemy korzystać z już napisanych frameworków JavaScriptowyh w naszym kodzie https://microsoft.github.io/TypeSearch/

5. Na zakończenie potrzebujemy skryptu inicjującego. W katalogu scripts dodaj plik index.ts Zacznijmy od dodania importu klasy PersonTable

```
import { PersonTable } from "./person-table";
```

zdefiniowania kilku stałych obiektów JQuery, które odpowiadają za obsługę niektórych elementów strony index.html

```
const advancedSearch = $('.js-adv-search');
const advancedSearchButton = $('.js-adv-search-btn');
const peopleTable = $('div#table');
const tableNext = $('#js-button-next');
const tablePrev = $('#js-button-prev');
```

const użyte przy zmiennej powoduje, że raz nadana wartość tej zmiennej nie może już ulec zmianie

Zdefiniujmy klasę Startup z metodą main, w której nastąpi przypisanie zdarzeń poszczególnym elementom na stronie

```
class Startup {
    public static main(): void {
        advancedSearch.hide();
        advancedSearchButton.click((event) =>
Startup.onAdvancedSearchClicked(event));
        let table = new PersonTable(peopleTable);
        table.next();
        tableNext.click(() => table.next());
        tablePrev.click(() => table.prev());
    }

    private static onAdvancedSearchClicked(event: JQueryEventObject) {
        event.preventDefault();
        // -> powoduje, że strona nie będzie się przeładowywać
        if (advancedSearch.is(':visible')) {
            advancedSearch.fadeOut(1000);
        } else {
            advancedSearch.fadeIn(1000);
        }
    }
}
```

Na koniec podepnijmy metodę main na zdarzenie końca ładowania strony

```
$(Startup.main)
```

index.ts

```
import { PersonTable } from "./person-table";

const advancedSearch = $('.js-adv-search');
const advancedSearchButton = $('.js-adv-search-btn');
const peopleTable = $('div#table');
const tableNext = $('#js-button-next');
const tablePrev = $('#js-button-prev');

class Startup {
    public static main(): void {
        advancedSearch.hide();
        advancedSearchButton.click((event) =>
Startup.onAdvancedSearchClicked(event));
        let table = new PersonTable(peopleTable);
        table.next();
        tableNext.click(() => table.next());
        tablePrev.click(() => table.prev());
    }

    private static onAdvancedSearchClicked(event: JQueryEventObject) {
        event.preventDefault();
        // -> powouje, że strona nie będzie się przeładowywac
        if (advancedSearch.is(':visible')) {
            advancedSearch.fadeOut(1000);
        } else {
            advancedSearch.fadeIn(1000);
        }
    }
}

$(Startup.main)
```

Zostało nam jeszcze wyrzucenie starych skryptów z index.html – usuń deklarowanie wszystkich skryptów JavaScrypt'owych a zamiast nich daj deklarację do pliku bundle.js który zostanie wygenerowany przez webpacka'a.

```
<script type="application/javascript" src="scripts/bundle.js"></script>
```

6. Konfiguracja skryptów uruchomieniowych w 'package.json'. zmodyfikuj wpis skryptu go do postaci:

```
    "go": "concurrently \"webpack -w\" \"node server.js\""
```

Uruchom ponownie projekt. Jeśli strona działa prawidłowo, to możesz z projektu usunąć stare pliki js z folderu scripts.

Zadania do samodzielnego wykonania:

1. Dodaj możliwość wyboru ilości wyników reprezentowanych na stronie (opcje do wyboru:10, 25, 50, 100 wyników).
2. Dodaj przyciski w nagłówkach kolumn tabelki, które będą sortować dane, aktualnie reprezentowane, w tabelce po danym polu (po imieniu, nazwisku, wieku, dacie urodzin itp.).
3. Dodaj pole tekstowe w nagłówkach kolumn tabelki, dzięki którym będzie można filtrować aktualną stronę wyników po danym polu.
4. Oprogramuj formularz do wyszukiwania osób.
5. Pochwal się efektem swojej pracy osobie prowadzącej laboratorium.