

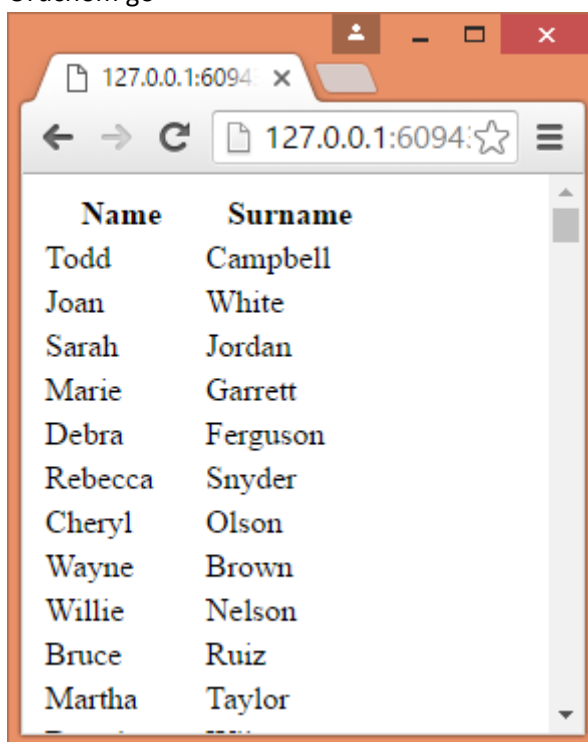
Zadania na ćwiczenia nr 2

Cele ćwiczenia:

- Zapoznanie się z podstawowymi aspektami programowania w JavaScriptcie
- Zapoznanie się z projektowaniem obiektowym w JavaScriptcie
- Umiejętność oprogramowywania zdarzeń html
- Sortowanie obiektów

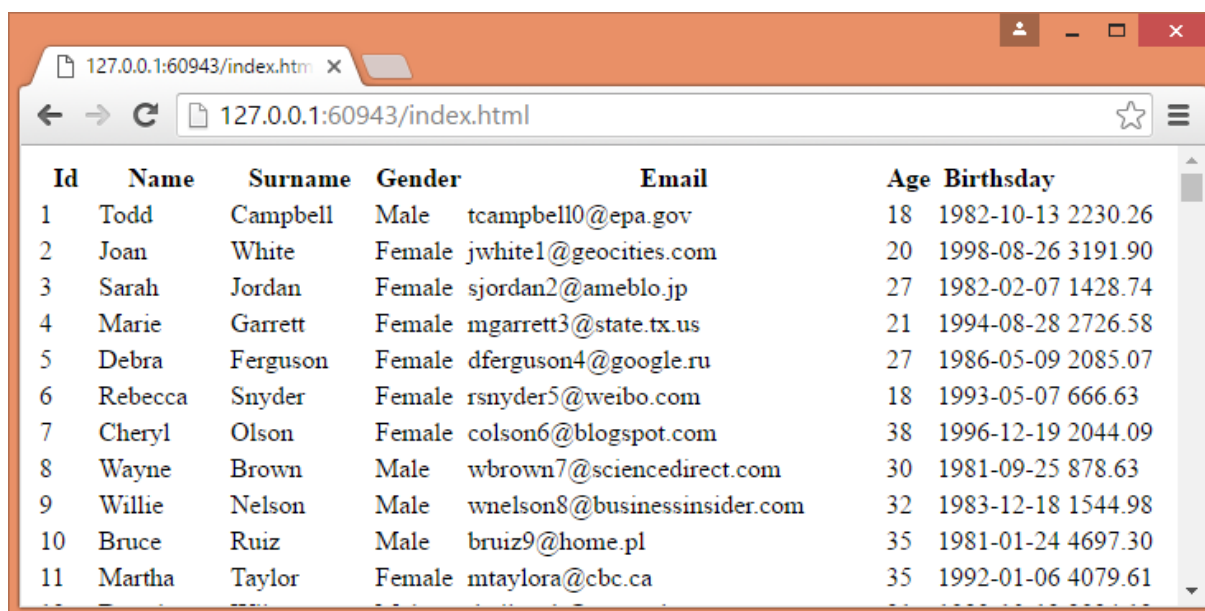
1. Zapoznanie się z projektem przykładowym. Jako przypomnienie treści z wykładu, wykonaj poszczególne kroki:

- Ściągnij projekt z materiałów do ćwiczeń 2 z portalu Edux.
- Uruchom go



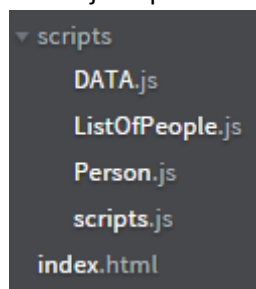
- Przeanalizuj projekt
 - Gdzie znajdują się dane do tabelki?
 - Jak załączone są skrypty JS do pliku index.html?
 - Czy kolejność załączania plików JS jest istotna?
 - Przeanalizuj jak generowana jest tabelka?
 - Jakie jeszcze posiadamy dane odnośnie osób?

Zadanie do samodzielnego wykonania: popraw kod tak aby na stronie generowała się tabelka ze wszystkimi danymi osobowymi



Id	Name	Surname	Gender	Email	Age	Birthsday
1	Todd	Campbell	Male	tcampbell0@epa.gov	18	1982-10-13 2230.26
2	Joan	White	Female	jwhite1@geocities.com	20	1998-08-26 3191.90
3	Sarah	Jordan	Female	sjordan2@ameblo.jp	27	1982-02-07 1428.74
4	Marie	Garrett	Female	mgarrett3@state.tx.us	21	1994-08-28 2726.58
5	Debra	Ferguson	Female	dferguson4@google.ru	27	1986-05-09 2085.07
6	Rebecca	Snyder	Female	rsnyder5@weibo.com	18	1993-05-07 666.63
7	Cheryl	Olson	Female	colson6@blogspot.com	38	1996-12-19 2044.09
8	Wayne	Brown	Male	wbrown7@sciencedirect.com	30	1981-09-25 878.63
9	Willie	Nelson	Male	wnelson8@businessinsider.com	32	1983-12-18 1544.98
10	Bruce	Ruiz	Male	bruiz9@home.pl	35	1981-01-24 4697.30
11	Martha	Taylor	Female	mtaylora@cbc.ca	35	1992-01-06 4079.61

- Podział na pliki. Zauważ, że kod JS zaczyna rosnąć – podziel go na pliki, w których umieścimy definicje odpowiednich klas oraz funkcji.



Poprawmy index.html dodając linijki z dodatkowymi skryptami

```
<html>
  <body onload="init()">

    <div id="table"></div>

    <script type="application/javascript" src="scripts/DATA.js"></script>
    <script type="application/javascript" src="scripts/Person.js"></script>
    <script type="application/javascript" src="scripts/ListOfPeople.js"></script>
    <script type="application/javascript" src="scripts/scripts.js"></script>
  </body>
</html>
```

- Nawigacja po wynikach. Zauważ, że w chwili obecnej użytkownik dostaje 1000 wyników w tabelce. Przeglądanie tak dużej ilości danych dla niego może być ciężkie, aby uprościć mu nawigację, a także 'upiększyć' aktualny widok dodajmy przyciski, dzięki którym będziemy przeglądać wyniki strona po stronie.

```
<div id="table"></div>
<button>PREV</button>
<button>NEXT</button>
```

Utwórzmy kolejną klasę, która będzie opisywała wygląd strony (widoku). Klasy które definiują zachowywanie się widoku są nazywane ViewModelami. Dodaj nowy plik JS a w nim klasę

```
function PeopleTableViewModel(config){
    var self = this;
    self.people = new ListOfPeople();
    self.currentPage = 0;
    self.pageSize = config.pageSize;
    self.context = config.context;

    self.next = function(){
        alert('kliknąłeś następny');
    }

    self.prev = function(){
        alert('kliknąłeś następny');
    }
}
```

Dodaj plik do strony

```
<script type="application/javascript" src="scripts/DATA.js"></script>
<script type="application/javascript" src="scripts/Person.js"></script>
<script type="application/javascript" src="scripts/ListOfPeople.js"></script>
<script type="application/javascript" src="scripts/PeopleTableViewModel.js">
</script>
<script type="application/javascript" src="scripts/scripts.js"></script>
```

Teraz w pliku 'script.js' utwórz obiekt powyższej klasy

```
var viewModel = new PeopleTableViewModel({
    pageSize:25,
    count:data.length,
    context:document.getElementById('table')
});

function init(){
```

Do przycisków pod zdarzenia 'onclick' podepnij metody 'next', 'prev' z obiektu

```
<div id="table"></div>
<button onclick="viewModel.prev()">PREV</button>
<button onclick="viewModel.next()">NEXT</button>
```

Sprawdź czy wyświetlają się komunikaty.

Kolejnym etapem będzie zaimplementowanie metod 'next' oraz 'prev'. Zaczniemy od metody pomocniczej, dzięki której będziemy mogli wybierać dowolny przedział danych

```
var getData = function(begin, end){
    if(end>data.length){
        end=data.length
    }
    if(begin<0) {
        begin =0;
    }
    for(var i = begin; i<end;i+=1){
        self.people.addPerson(data[i]);
    }
}
```

Ogólnie w bardziej realnej sytuacji metoda getData powinna pobierać dane z jakiegoś restowego serwisu, ale jako że tematyka budowania serwisów restowych wykracza poza zakres zajęć, zostaniemy przy sztywnych danych wzięty z pliku 'DATA.js'. Korzystając z tej metody zmienimy ciała metod 'next' oraz 'prev'

```

self.next = function(){
    self.people.clear();
    var begin = (self.currentPage)*self.pageSize;
    var end =(self.currentPage+1)*self.pageSize;
    getData(begin,end);
    self.currentPage++;
    self.context.innerHTML=self.people.toTable();
}

self.prev = function(){
    self.people.clear();
    if(self.currentPage-1>=0){
        self.currentPage--;
    }
    var begin = (self.currentPage)*self.pageSize;
    var end =(self.currentPage+1)*self.pageSize;
    getData(begin,end);
    self.context.innerHTML=self.people.toTable();
}

```

Na zakończenie należy delikatnie zmienić metodę 'init' w scripts.js

```

function init(){
    viewModel.next();
}

```

Sprawdź, czy strony w tabli danych się zmieniają.

Zadanie do samodzielnego wykonania: między przyciskami do nawigacji dodaj pole do wyboru, gdzie będziesz mógł ustawić ilość wyników na stronę. Wartości przykładowe: 25, 50, 100. Zmodyfikuj view model tak aby uwzględnił wybrane ustawienia i aplikował je do wyników wyświetlanych w tabelce

4. Sortowanie danych. W klasie viewModelu dopiszmy metodę która będzie sortować dane z tabelki po imieniu, nazwisku i wieku. Ogólnie JavaScript nie ma problemu z sortowaniem kolekcji w której znajdują się proste typy danych, jak łańcuchy znaków czy liczby. W przypadku obiektów sprawa delikatnie się komplikuje, gdyż mechanizmy do sortowania należy przekazać informację co to znaczy, że jeden obiekt jest większy od drugiego. Dodajmy klasę w której będziemy trzymać możliwe reguły sortowania. Nazwijmy ją 'Comparators'

```

function Comparators(){
    var self = this;
    self.byName = function(person1, person2){
        return person1.firstName.localeCompare(person2.firstName);
    }

    self.bySurname = function(person1, person2){
        return person1.lastName.localeCompare(person2.lastName);
    }

    self.byAge = function(person1, person2){
        return person1.age - person2.age
    }
}

```

Zwróć uwagę, że stringi posiadają metody do porównywania.

Dodajmy ten skrypt do strony, przed deklaracją skryptu z ViewModelem

```
<script type="application/javascript" src="scripts/DATA.js"></script>
<script type="application/javascript" src="scripts/Person.js"></script>
<script type="application/javascript" src="scripts/ListOfPeople.js"></script>
<script type="application/javascript" src="scripts/comparators.js"></script>
<script type="application/javascript" src="scripts/PeopleTableViewModel.js">
</script>
<script type="application/javascript" src="scripts/scripts.js"></script>
```

Utwórz instancję obiektu comparator w pliku script.js

```
var comparator = new Comparators();

function init(){
```

W ViewModelu dodaj metodę 'sort' która w parametrze przyjmie odpowiednią funkcję do porównywania i ją wykorzysta do mechanizmu który już jest zaimplementowany w JavaScriptcie

```
self.sort = function(comparer){
    data.sort(comparer);
    self.currentPage=0;
    self.next();
}
```

Aby zobaczyć jak działa sortowanie należy jeszcze poprawić generowanie widoku tabeli. Zrobimy tak, aby w nagłówku kolumny znajdował się przycisk do sortowania. W klasie 'ListOfPeople' zmień metodę 'generateTableHeader'

```
var generateTableHeader= function(){
    return '<tr><th>Id</th>'
        +' <th><button onclick="viewModel.sort(comparator.byName)">Name</button>'
        + '</th>'
        +' <th>Surname</th>'
        +' <th>Gender</th>'
        +' <th>Email</th>'
        +' <th>Age</th>'
        +' <th>Birthsday</th>'
        +' <th>Income</th>'
        +' </tr>'
}
```

Sprawdź działanie sortowania

Zadanie do samodzielnego wykonania: zaimplementuj możliwość sortowania dla pozostałych kolumn.