|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opis: Opis: https://lh5.googleusercontent.com/liXomy-dBkIy4NJmsn4fAgV_CNgWcVTfBg6tOwZSQPQZLj7z_GjYTpWMVUi_bg8bLUZOqP0A8Ggohr_cIvCVaZ5EL551sQASeT-w_3nZ2-qaraX7_HI4tEgBSBMWIRANvgSMCFsZ | Uniwersytet Technologiczno-Przyrodniczy  Im. J. J. Śniadeckich w Bydgoszczy  Wydział Telekomunikacji, Informatyki i  Elektrotechniki | | | Opis: Opis: https://lh3.googleusercontent.com/67FmnUfRkRkpAdvVrMTIpRSYwASuOFOAq6IRqQDxpys1qbjXCRsffOougIZsMLj2kW5QSfklmJrF5i43aEGYExBaBoLAEyN-K4l9mlTUEfewPUcczztEFMrvShu-eVM7XIslmNps |
| Przedmiot | **Zaawansowane programowanie obiektowe** | | | |
| Prowadzący | **Dr inż. Marcin Szczegielniak** | | | |
| Temat | **Stream API** | | | |
| Student | Kacper Dąbrowski | | | |
| Nr albumu | 113018 | | | |
| Nr lab. | 3 | Data wykonania | 4.06.2020 | |
| Ocena |  | Data oddania: | 05.06.2020 | |

Spis treści

[Opis projektu i użyte technologie 2](#_Toc36831861)

[Kluczowe fragmenty kodu 3](#_Toc36831862)

[Mechanizm dynamicznego renderowania widoku na podstawie zmiennej przekazanej jako parametr funkcji render() 3](#_Toc36831863)

[NodeJS – Obsługa zdarzeń 5](#_Toc36831864)

[Wnioski 6](#_Toc36831865)

[Dokumentacja 6](#_Toc36831866)

# 

## Array.filter

Metoda, która zwraca tablicę, która zawiera tylko elementy zwracające boolowską wartość true

## Array.sort

Funkcja, która przyjmuje dwa elementy do porównania jako argumenty i funkcję porównującą. Postępuje według następującego algorytmu:

* Jeżeli funkcja sortująca zwróci liczbę większą od 0, to pierwszy argument wystąpi przed drugim
* Jeżeli funkcja zwróci 0, to pierwszy i drugi argument są sobie równe
* Jeżeli funkcja zwróci liczbę mniejszą od 0, to drugi element wystąpi przed pierwszym.

## Array.map

Funkcja, która zwraca nową tablicę ze zmodyfikowanymi elementami tablicy „macierzystej” nie modyfikując jej.

## Flatmap

Ta metoda nie występuje natywnie w Javascript, ale użyłem zastępczego kontekstu. Spread operator to semantyczny zabieg w tym języku, który pozwala na zastąpienie kolejnego wypisywania elementów tablicy […tablica], co pozwoli na wypisanie wszystkich elementów obu tablic i utworzenie nowej, połączonej tablicy z obu grup.

* Reduce
* parallelStream

# Kluczowe fragmenty kodu

## Mechanizm dynamicznego renderowania widoku na podstawie zmiennej przekazanej jako parametr funkcji render()

Po stronie serwera wydajemy polecenie wyrenderowania strony i jako argument podajemy tablicę, którą serwer odtworzył z pliku data/data.json

Istotne dla tej funkcjonalności elementy kodu:

Klasa Note i jej integracja z systemem plików

1. **static** fetchAll() {
2. **const** notesFromFile = fs.readFileSync(dataFilePath);
3. **const** notes = JSON.parse(notesFromFile);
4. **return** notes;
5. }
6. **static** readNote(id) {
7. **const** notes = **this**.fetchAll();
8. **const** foundNote = notes.find(n => n.id === id);
9. **return** foundNote !== -1 ? foundNote : **new** Error("Note not found");
10. }
11. createNote() {
12. **const** notes = Note.fetchAll();
13. notes.push(**this**);
14. fs.writeFileSync(dataFilePath, JSON.stringify(notes));
15. }

Kontroler dla klasy Note

1. **const** Note = require("../models/Note");
3. exports.addNote = (req, res, next) => {
4. **const** { title, description, priority } = req.body;
5. **const** testNote = **new** Note(title, description, priority);
6. testNote.createNote();
7. **const** notes = Note.fetchAll()
8. res.render('index',{notes})
9. };

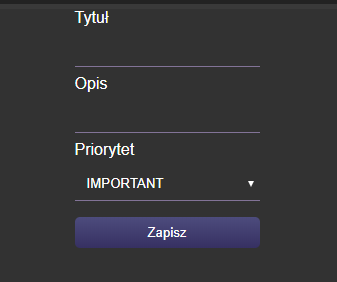
12. exports.deleteNote = (req, res, next) => {
13. **const** { noteId } = req.body;
14. Note.removeNote(noteId);
15. res.redirect("/");
16. };

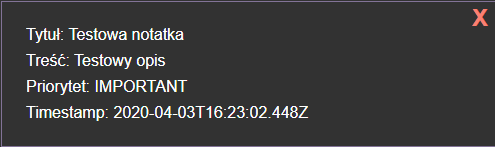
Funkcja render w tym wypadku powoduje, że następujący fragment kodu EJS (plik /views/index.ejs) jest renderowany na podstawie zmiennej notes, którą podajemy w jej wywołaniu.

1. **<ul>**
2. **<**%if(notes){for(let i =0;i**<notes.length**;i++){%**>**
3. **<li** class="note"**>**
4. **<form** method="POST" action="/delete"**>**
5. **<input** type="hidden" name="noteId" value="<%=notes[i].id%>"**>**
6. **<button** class="delete-note" type="submit"**>**X**</button>**
7. **</form>**
8. **<p** class="note-title"**>**Tytuł: **<**%=notes[i].title%**></p>**
9. **<p** class="note-description"**>**Treść: **<**%=notes[i].description%**></p>**
10. **<p** class="note-priority"**>**Priorytet: **<**%=notes[i].priority%**></p>**
11. **<p** class="note-timestamp"**>**Timestamp: **<**%=notes[i].id%**></p>**
12. **</li>**
13. **<**%}}%**>**
14. **</ul>**

Dzięki takiemu ustawieniu, funkcja for wewnątrz elementu <ul> generuje element <li> o zadanych parametrach. Umieściłem tutaj też drugi formularz, który dynamicznie przydziela ukrytemu elementowi <input /> wartość elementu w tablicy. Dzięki temu jesteśmy w stanie wyrenderować listę elementów, którą będziemy w stanie dynamicznie usuwać za pomocą przycisku „X”.

Po odpowiednim załadowaniu styli, po wejściu na adres localhost:3000 ukazuje nam się formularz do dodania notatki. Po jej dodaniu, mamy dostęp do instancji klasy Note, którą możemy dowolnie przedstawić na ekranie za pomocą HTML i CSS





## NodeJS – Obsługa zdarzeń

Express.js pozwala serwerowi na ustawienie „listenerów” na konkretne zdarzenia podczas komunikacji z nim. Tutaj, użyliśmy app.get oraz app.post, aby oznajmić serwerowi, że pod adresami podanymi jako parametr funkcji, ma oczekiwać na requesty metodą GET i POST oraz reagować na nie zdefiniowanymi kontrolerami.

1. app.get('/',(req,res,next)=>{
2. **const** notes = Note.fetchAll();
3. res.render('index',{notes})
4. })
5. app.set('view engine', 'ejs');
6. app.use(express.**static**('public'))
7. app.set("views", path.join(\_\_dirname, "views"));
8. app.use(bodyParser.urlencoded())
9. app.post('/add',notesController.addNote)
10. app.post('/delete',notesController.deleteNote);

13. app.listen(3000,()=>{
14. console.log('I am running!');
16. })

app.set pełni tu funkcję konfiguracyjną. Ustawiamy folder publiczny, folder zawierający szablony dla silnika renderującego, oraz definiujemy, z jakiego silnika będziemy korzystali.

Funkcję app.listen wywołujemy z konkretnym portem do nasłuchu i powoduje ona, że serwer nie kończy wykonywania z ostatnią linijką, ale nasłuchuje requestów.

# Wnioski

JavaScript okazuje się znakomitym narzędziem do tworzenia aplikacji typu to-do-list ponieważ możemy łatwo i szybko zintegrować go z systemem plików i korzystać z jego zalet.

Co można poprawić?

Gdybym miał dopracowywać aplikację w dalszym stopniu, to przede wszystkim dodałbym obsługę baz danych, ponieważ operacja na pliku jest dużo wolniejsza (musimy wczytać cały plik nieważne, czy chcemy jedną notatkę, czy wszystkie). Oraz dodał styl pola z notatki według ich priorytetu. Kolejną zmianą, którą można wprowadzić, jest funkcjonalność edycji notatki.

# Dokumentacja

1. Node.js [path](https://nodejs.org/api/path.html) module
2. Node.js [filesystem](https://nodejs.org/api/fs.html) module
3. [Express.js](https://expressjs.com/)
4. [Embedded JavaScript template engine](https://ejs.co/#docs)
5. [Body-parser](https://www.npmjs.com/package/body-parser)

[](https://github.com/kacdab003/zpo-lab7)