|  |  |  |
| --- | --- | --- |
| **AI1** | Sujewicz Patryk  Album 53782  Grupa 6 | **Wersja 1** |
|  |
| **LAB B** |

JS i DOM na przykładzie listy TODO

# Spis treści

[Spis treści 1](#_Toc178020089)

[Cel zajęć 1](#_Toc178020090)

[Rozpoczęcie 1](#_Toc178020091)

[Uwaga 2](#_Toc178020092)

[Wymagania 2](#_Toc178020093)

[Strona HTML](#_Toc178020094)

[2](#_Toc178020094)

[Klasa Todo 3](#_Toc178020095)

[Dodawanie pozycji listy 3](#_Toc178020096)

[Usuwanie pozycji listy 4](#_Toc178020097)

[Edycja pozycji listy 4](#_Toc178020098)

[Odczyt / Zapis LocalStorage 5](#_Toc178020099)

[Wyszukiwanie 6](#_Toc178020100)

[Commit projektu do GIT 6](#_Toc178020101)

[Podsumowanie 7](#_Toc178020102)

# Cel zajęć

Celem głównym zajęć jest zdobycie następujących umiejętności:

* przemieszczania się po drzewie DOM;
* dodawania, usuwania, edytowania elementów drzewa DOM.

W praktycznym wymiarze utworzona zostanie dynamiczna lista czynności do zrobienia (lista To Do).

# Rozpoczęcie

Rozpoczęcie zajęć. Powtórzenie metod przemieszczania się po drzewie DOM.

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.

# Wymagania

W ramach LAB B przygotowane powinny zostać:

* pojedyncza strona HTML ze skryptem ładowanym z zewnętrznego pliku JS
* lista zadań
* na dole listy pole tekstowe do dodawania nowych zadań, pole typu data/czas do określenia terminu wykonania zadania, przycisk dodawania zadania
* walidacja nowych zadań: co najmniej 3 znaki, nie więcej niż 255 znaków, data musi być pusta albo w przyszłości
* na górze listy pole wyszukiwarki
* po wpisaniu w wyszukiwarkę co najmniej 2 znaków na liście wyświetlają się wyłącznie pozycje zawierające wpisaną w wyszukiwarkę frazę
* wyszukiwana fraza zostaje wyróżniona w każdym wyniku wyszukiwania
* kliknięcie na dowolną pozycję listy zmienia ją w pole edycji; kliknięcie poza pozycję listy zapisuje zmiany
* obok każdej pozycji listy znajduje się przycisku Usuń / Śmietnik
* wpisy na liście zapisują się do Local Storage
* po odświeżeniu strony lista wypełnia się wpisami z Local Storage

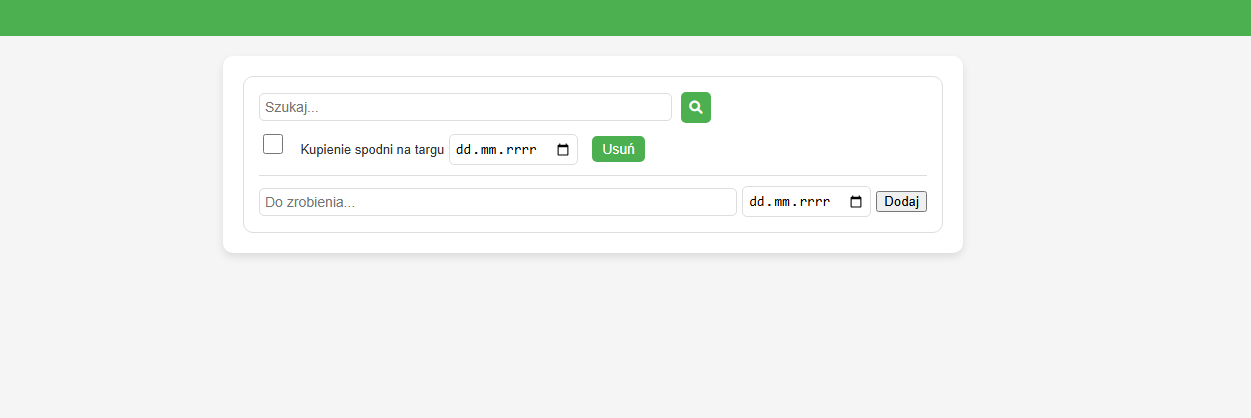
Mockupy:



# Strona HTML

Prace rozpocznij od implementacji HTML z danymi wpisanymi „na sztywno”. Upewnij się, że wstawione zostały wszystkie wymagane elementy – pole wyszukiwarki, lista, pole dodawania, przycisk usuwania. To laboratorium koncentruje się na JS, więc może być ładne, ale nie musi. Nie trać za dużo czasu na CSS.

Wstaw zrzut ekranu przedstawiający stronę HTML z polem wyszukiwarki, listą, polem dodawania, przyciskami usuwania:



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

# Klasa Todo

Pierwszym instynktem może być chęć dodania zachowań bezpośrednio do elementów listy w drzewie DOM. Chociaż na krótką metę wydaje się być to najprostsze rozwiązanie, za chwilę okaże się krótkowzroczne i trudne do implementacji przy kolejnych punktach 😊

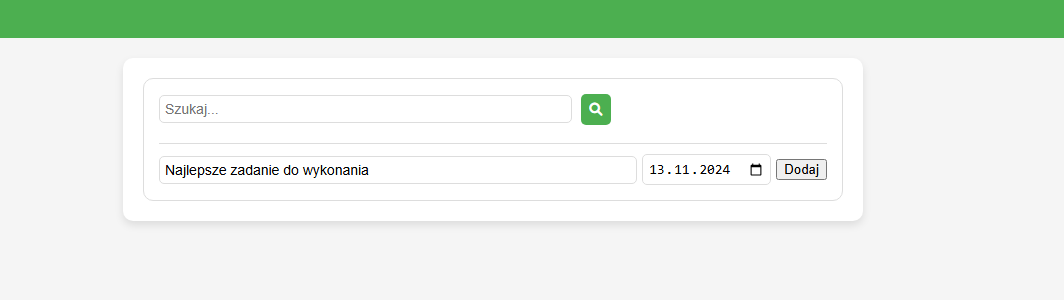
Najlepszym sposobem rozwiązania tego laboratorium jest utworzenie klasy Todo (albo po prostu obiektu z kilkoma metodami). Bez względu na przyjętą strategię, należy w tym nowoutworzonym bycie utworzyć tablicę tasks oraz metodę draw(), która wyczyści div z obecną wizualizacją zadań do zrobienia i wygeneruje ją na nowo na podstawie tablicy tasks.

W celu sprawdzenia poprawności działania, najlepiej dostać się do tablicy tasks i edytować jej zawartość, po czym ręcznie wywołać metodę draw(). Jeśli zawartość listy wyrenderuje się na nowo poprawnie – możemy iść dalej!

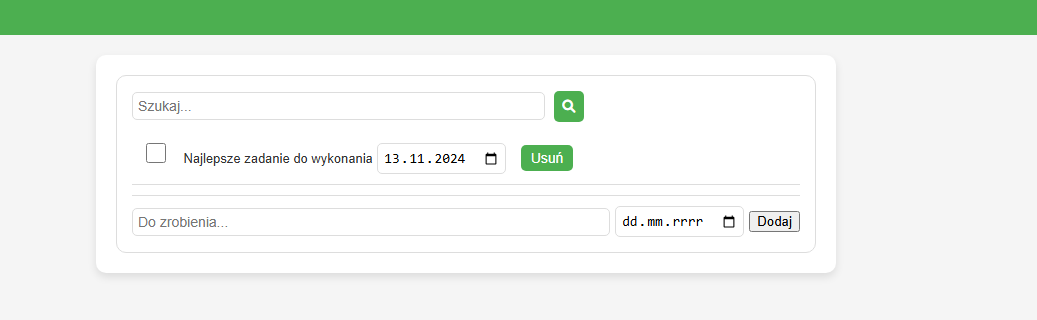
Zaimplementuj dodawanie, usuwanie, edycję pozycji listy – wszystko modyfikujące tablicę tasks i wywołujące na koniec metodę draw().

# Dodawanie pozycji listy

Wstaw zrzut ekranu listy przed dodaniem nowego zadania:



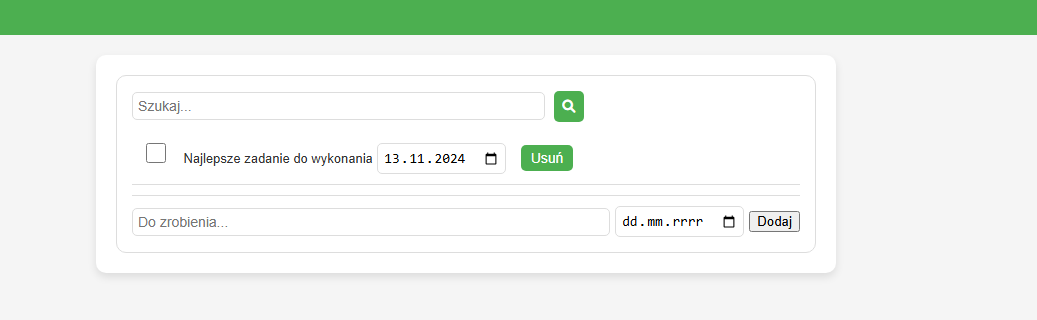
Wstaw zrzut ekranu listy po dodaniu nowego zadania:



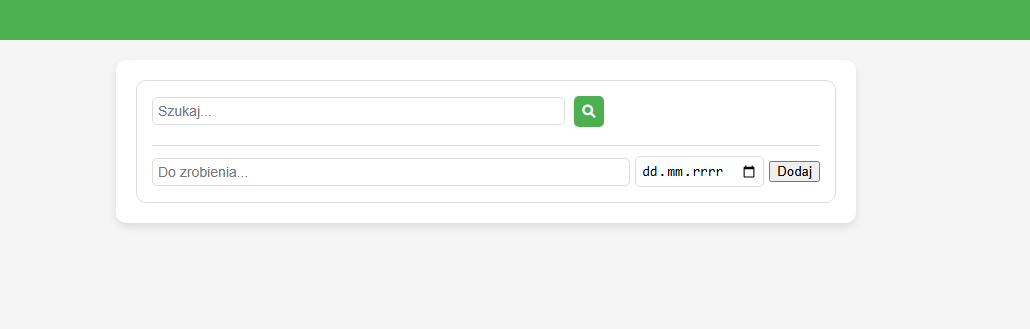
|  |  |  |
| --- | --- | --- |
| Punkty: | 0 | 1 |

# Usuwanie pozycji listy

Wstaw zrzut ekranu listy przed usunięciem wybranego zadania:



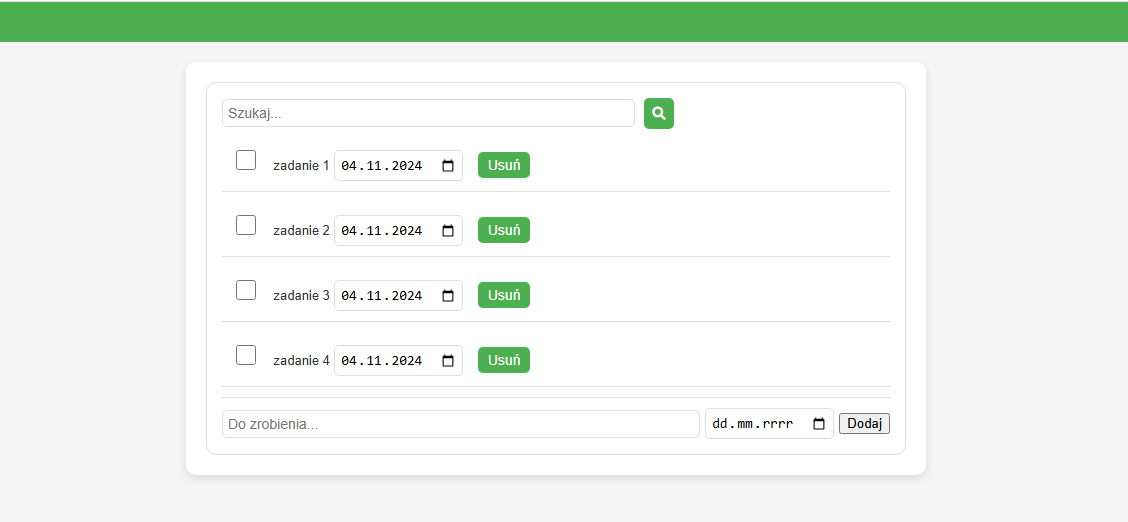
Wstaw zrzut ekranu listy po usunięciu zadania:



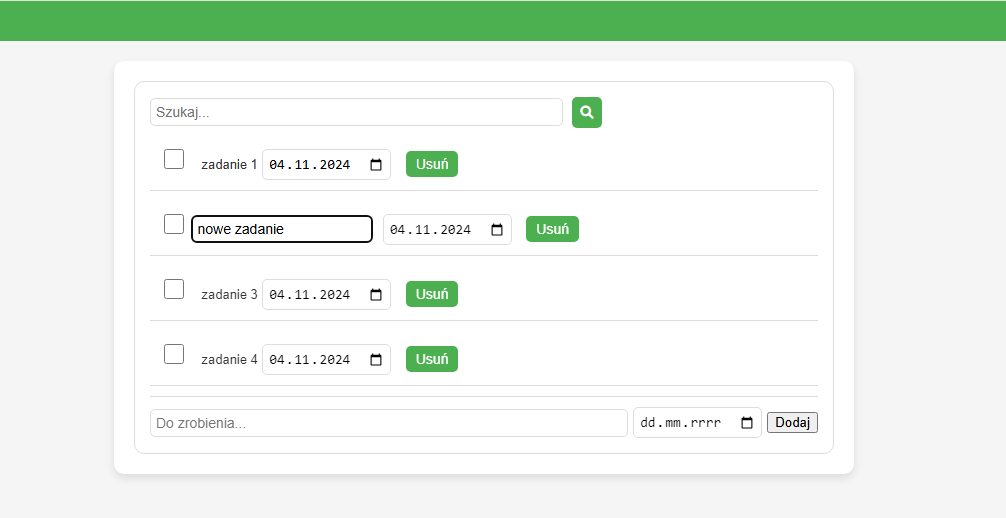
|  |  |  |
| --- | --- | --- |
| Punkty: | 0 | 1 |

# Edycja pozycji listy

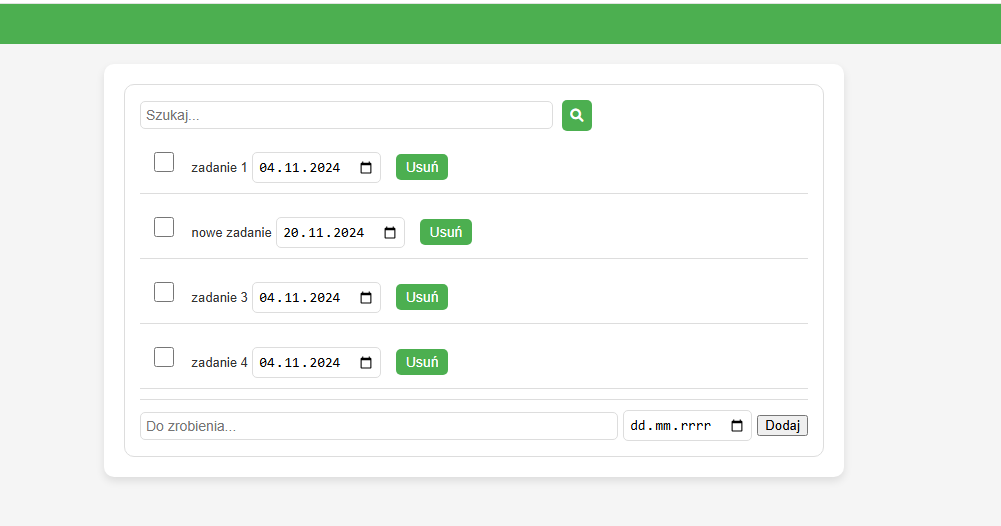
Wstaw zrzut ekranu listy przed edycją wybranego zadania:



Wstaw zrzut ekranu listy w trakcie edytowania zadania i daty:



Wstaw zrzut ekranu listy po edycji zadania i daty. Upewnij się, że dane się zapisały i zadanie jest zmienione:

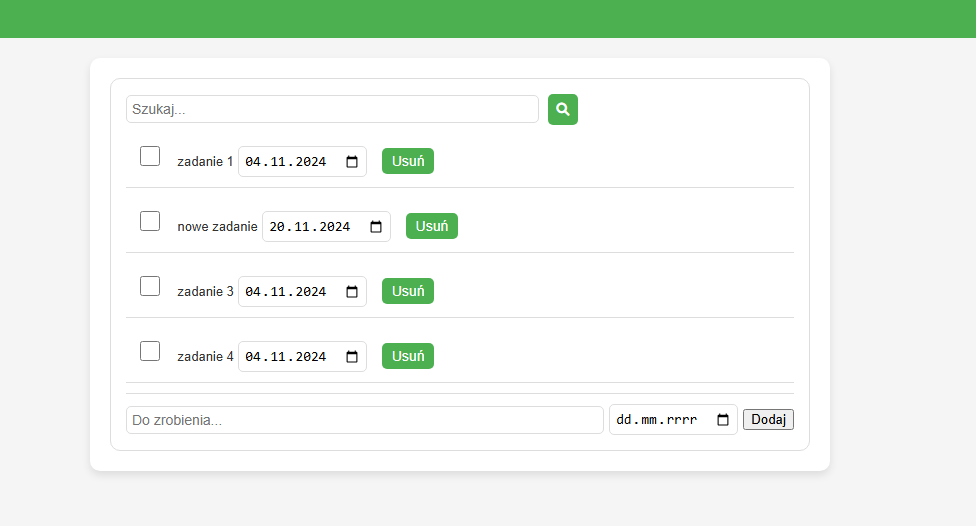


|  |  |  |
| --- | --- | --- |
| Punkty: | 0 | 1 |

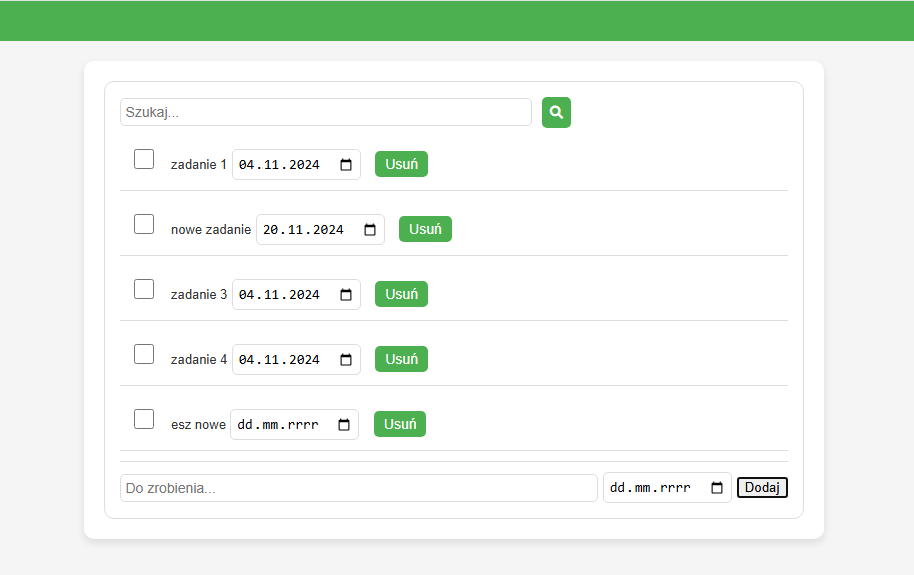
# Odczyt / Zapis LocalStorage

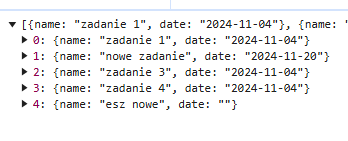
Zastosowanie klasy Todo w realizacji tego laboratorium pozwala w bardzo łatwy sposób odczytywać i zapisywać stan listy do pamięci przeglądarki. Wystarczy serializacja / deserializacja za pomocą JSON.parse() i JSON.stringify().

Wstaw zrzuty ekranu przedstawiające wygląd listy i zawartość local storage gdy na liście są pewne zadania:



Wstaw zrzuty ekranu przedstawiające wygląd listy i zawartość local storage po dodaniu nowej pozycji listy. Upewnij się, że widoczne w local storage są dane dotyczące nowego zadania:



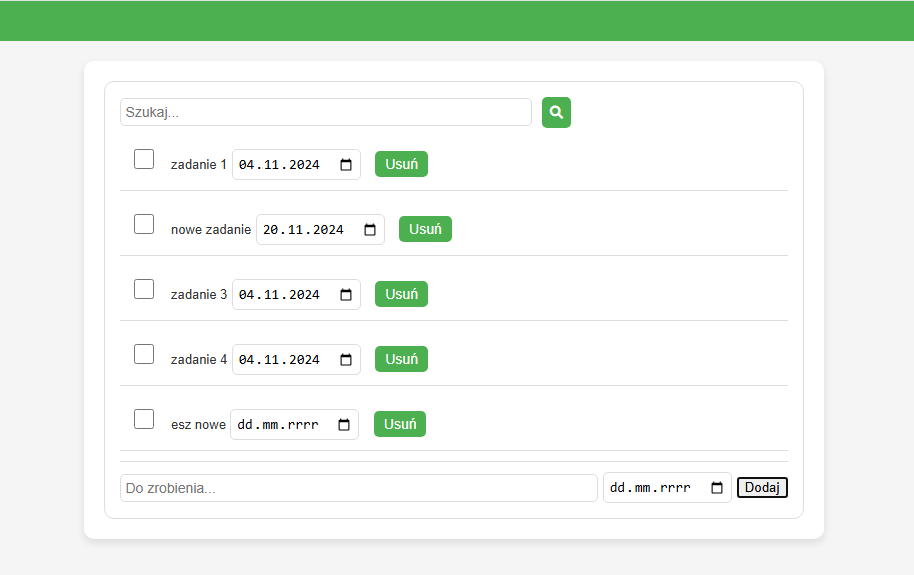


|  |  |  |
| --- | --- | --- |
| Punkty: | 0 | 1 |

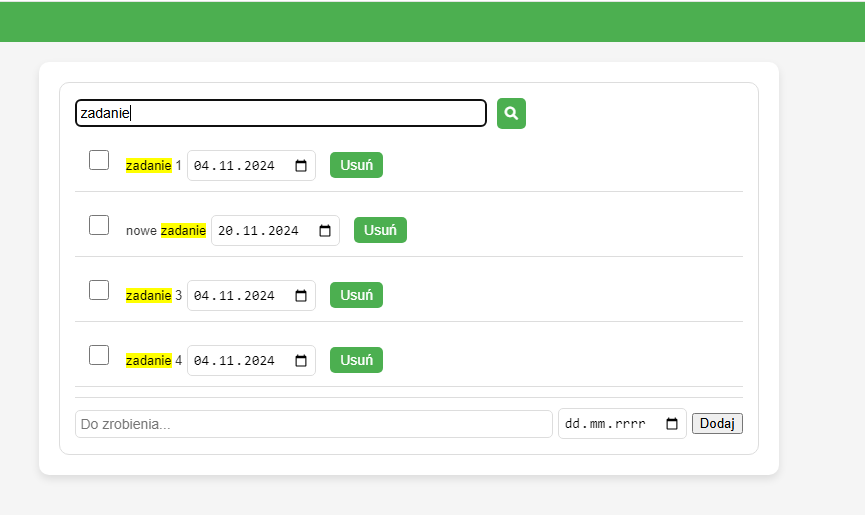
# Wyszukiwanie

Na koniec zostało filtrowanie wyników. Proponowanym podejściem do tego tematu jest umieszczenie w klasie Todo właściwości term – frazy wyszukiwanej przez użytkownika. Następnie można utworzyć metodę getFilteredTasks, albo getter filteredTasks, która zwracać będzie te elementy tablicy tasks, które odpowiadają zapytaniu. Można użyć funkcji wyższego rzędu filter().

Wstaw zrzut ekranu listy, gdy pole wyszukiwania jest puste:

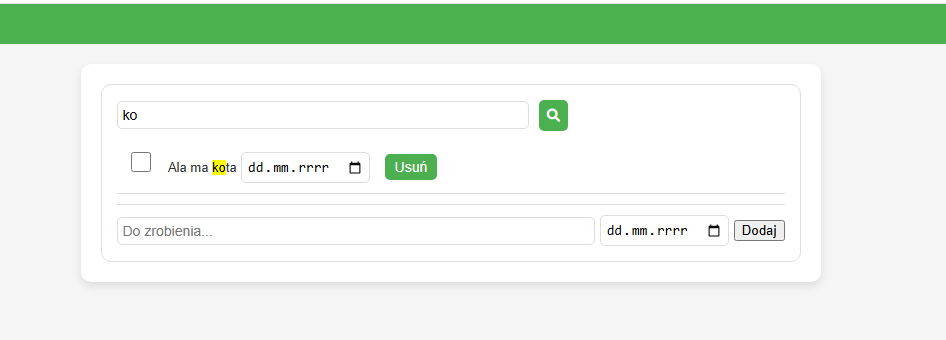


Wstaw zrzut ekranu listy, gdy w polu wyszukiwania wpisano wystarczająco dużo znaków, by zadziałało filtrowanie. Upewnij się, że chociaż 2 wyniki będą wciąż widoczne:



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

Wstaw zrzut ekranu przedstawiający podświetlenie szukanej frazy w wynikach wyszukiwania, przykładowo dla frazy ko i zadania Ala ma kota otrzymujemy: Ala ma kota:



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

# Commit projektu do GIT

Zacommituj i pushnij swoje rozwiązanie do repozytorium GIT.

Upewnij się, czy wszystko dobrze się wysłało. Jeśli tak, to z poziomu przeglądarki utwórz branch o nazwie lab-b na podstawie głównej gałęzi kodu.

Podaj link do brancha lab-b w swoim repozytorium:

…link, np. <https://github.com/inazwisko/ai1-lab/tree/lab-b…>

# Podsumowanie

W kilku zdaniach podsumuj zdobyte podczas tego laboratorium umiejętności.

…podsumowanie…

Zweryfikuj kompletność sprawozdania. Utwórz PDF i wyślij w terminie.