

Filtrowanie listy

Posts							
<ul style="list-style-type: none"> Posts Comments Users Tags 	Search						
			ADD FILTER + CR				
	<input type="checkbox"/>	Id	Title	Published at ↓	Com.	Views	
	> <input type="checkbox"/>	13	Fusce massa lorem, pulvinar a posuere ut, acc...	01/12/2012	✓	222	ED
	> <input type="checkbox"/>	12	Qui tempore rerum et voluptates	07/11/2012	✓	719	ED
	> <input type="checkbox"/>	11	Omnis voluptate enim similique est possimus	22/10/2012	✓	294	ED
	> <input type="checkbox"/>	10	Totam vel quasi a odio et nihil	19/10/2012	✓	721	ED
	> <input type="checkbox"/>	9	A voluptas eius eveniet ut commodi dolor	16/10/2012	✓	143	ED
	> <input type="checkbox"/>	8	Culpa possimus quibusdam nostrum enim te...	02/10/2012	✗	557	ED
	> <input type="checkbox"/>	7	Illum veritatis corrupti exercitationem sed velit	29/09/2012	✓	133	ED
	> <input type="checkbox"/>	6	Minima ea vero omnis odit officiis aut	05/09/2012		208	ED
	> <input type="checkbox"/>	4	Maiores et itaque aut perspiciatis	12/08/2012	✗	685	ED
	> <input type="checkbox"/>	2	Sint dignissimos in architecto aut	08/08/2012	✓	563	ED
				Rows per page: 10	1-10 of 13	1	

Jedną z najważniejszych funkcji strony Lista jest możliwość filtrowania wyników. React-admin dokłada wszelkich starań, aby zaoferować potężną funkcjonalność filtrowania i zejść z drogi, gdy chcesz iść dalej.

W kolejnych sekcjach wyjaśniono, jak korzystać z funkcji filtrowania. Na początek kilka wyjaśnień na temat wewnętrznego działania filtrów:

- Filtruj parametr zapytania
- Łączenie z wstępnie przefiltrowaną listą

React-admin proponuje kilka komponentów interfejsu użytkownika, aby umożliwić użytkownikom przeglądanie i modyfikowanie filtrów oraz udostępnia narzędzia do tworzenia niestandardowych.

- Kombinacja przycisku filtrowania/formularza
 - Stosowanie
 - Wyszukiwanie pełnotekstowe
 - Szybkie filtry
- Pasek <FilterList>boczny

000001

- Zapisane zapytania: pozwól użytkownikom zapisywać filtrowanie i sortowanie
- Budowanie niestandardowego filtra

Filtruj parametr zapytania

React-admin używa `filterparametru` zapytania z adresu URL, aby określić filtry, które należy zastosować do listy. Aby zmienić filtry, React-admin po prostu zmienia ten `filterparametr` zapytania, a `<List>` komponenty pobierają `dataProvider.getList()` się ponownie z nowymi filtrami.

Oto typowy adres URL listy:

```
https://myadmin.dev/#/posts?
displayedFilters=%7B%22komentarz%22%3Atrue%7D&filter=%7B%22komentarz%22%3Atrue%2C%22q%22%3A%22lorem%20%22%7D&order=
DESC&page=1&perPage=10&sort=published_at
```

Po zdekodowaniu `filterparametr` zapytania ujawnia się jako wartość JSON:

```
filter={"commentable":true,"q":"lorem "}
```

Możesz zmienić filtry, aktualizując parametr zapytania, np. używając `<Link>` komponentu lub `useNavigate()` hooka z `react-router-dom`.

Wskazówka : Gdy użytkownik ustawi filtr, React-admin utrzuca wartość filtra w stanie aplikacji, dzięki czemu gdy użytkownik wróci do listy, powinien zobaczyć przefiltrowaną listę. To wybór projektu.

Łączenie z wstępnie przefiltrowaną listą

Ponieważ wartości filtrów są pobierane z adresu URL, możesz utworzyć link do wstępnie przefiltrowanej listy, ustawiając `filterparametr` zapytania.

Na przykład, jeśli masz listę tagów, możesz wyświetlić przycisk dla każdej kategorii, aby połączyć się z listą postów przefiltrowanych według tego tagu:

```
import Button from '@mui/material/Button';
import { Link } from 'react-router-dom';

const LinkToRelatedProducts = ({ record }) => {
  const translate = useTranslate();
  return record ? (
    <Button
      color="primary"
      component={Link}
      to={{
        pathname: '/posts',
        search: `filter=${JSON.stringify({ category_id: record.id })}`,
      }}
    >
      All posts with the category {record.name} ;
    </Button>
  ) : null;
};
```

Możesz użyć tego przycisku np. jako dziecko `<Datagrid>`. Możesz również utworzyć niestandardowy przycisk Menu za pomocą tej techniki, aby połączyć się z niefiltrowaną listą, ustawiając wartość filtra na `{}`.

Kombinacja przycisku filtrowania/formularza

Posts							
<div>Posts</div> <div>Comments</div> <div>Users</div> <div>Tags</div>	Search		ADD FILTER + CRE				
	<input type="checkbox"/>	Id	Title	Published at ↓	Com.	Views	
	> <input type="checkbox"/>	13	Fusce massa lorem, pulvinar a posuere ut, acc...	01/12/2012	✓	222	EDIT
	> <input type="checkbox"/>	12	Qui tempore rerum et voluptates	07/11/2012	✓	719	EDIT
	> <input type="checkbox"/>	11	Omnis voluptate enim similique est possimus	22/10/2012	✓	294	EDIT
	> <input type="checkbox"/>	10	Totam vel quasi a odio et nihil	19/10/2012	✓	721	EDIT
	> <input type="checkbox"/>	9	A voluptas eius eveniet ut commodi dolor	16/10/2012	✓	143	EDIT
	> <input type="checkbox"/>	8	Culpa possimus quibusdam nostrum enim te...	02/10/2012	✗	557	EDIT
	> <input type="checkbox"/>	7	Illum veritatis corrupti exercitationem sed velit	29/09/2012	✓	133	EDIT
	> <input type="checkbox"/>	6	Minima ea vero omnis odit officiis aut	05/09/2012		208	EDIT
	> <input type="checkbox"/>	4	Maiores et itaque aut perspiciatis	12/08/2012	✗	685	EDIT
	> <input type="checkbox"/>	2	Sint dignissimos in architecto aut	08/08/2012	✓	563	EDIT
	Rows per page: 10 ▾ 1-10 of 13 1						

Domyślnym wyglądem filtrów jest wbudowany formularz wyświetlany na górze listy. Użytkownicy widzą również przycisk rozwijany, który pozwala dodać więcej danych wejściowych do tego formularza. Ta funkcjonalność opiera się na `<List filters>` rek wizycie:

```
import { TextInput } from 'react-admin';

const postFilters = [
  <TextInput label="Search" source="q" alwaysOn />,
  <TextInput label="Title" source="title" defaultValue="Hello, World!" />,
];

export const PostList = (props) => (
  <List {...props} filters={postFilters}>
    ...
  </List>
);
```

Elementy przekazywane jako `filters` są zwykłymi danymi wejściowymi. Oznacza to, że możesz budować zaawansowane filtry oparte na referencjach, wartościach tablic itp. `<List>` Domyślnie ukrywa wszystkie dane wejściowe w formie filtra, z wyjątkiem tych, które mają właściwość `alwaysOn`.

Wskazówka : Z przyczyn technicznych, React-admin nie akceptuje danych wejściowych filtra zawierających zarówno `a`, jak `defaultValue` i `alwaysOn`. Aby ustawić domyślne wartości dla zawsze włączonych filtrów, użyj `filterDefaultValues` właściwości `<List>` komponentu.

`<List>` używa elementów przekazanych jako `filters` dwukrotnie:

- raz, aby wyrenderować formularz filtra
- raz, aby wyrenderować przycisk filtra (używając każdego elementu `label`, wracając do humanizowanego `source`)

<SearchInput>

Search

<input type="checkbox"/>	Id	Title	Published at ↓
>	<input type="checkbox"/>	13	Fusce massa lorem, pulvinar a posuere ut, accumsan ac nisi01/12/2012
>	<input type="checkbox"/>	12	Qui tempore rerum et voluptates07/11/2012
>	<input type="checkbox"/>	11	Omnis voluptate enim similique est possimus22/10/2012
>	<input type="checkbox"/>	10	Totam vel quasi a odio et nihil19/10/2012
>	<input type="checkbox"/>	9	A voluptas eius eveniet ut commodi dolor16/10/2012
>	<input type="checkbox"/>	8	Culpa possimus quibusdam nostrum enim tempore rerum odit ex...02/10/2012
>	<input type="checkbox"/>	7	Illum veritatis corrupti exercitationem sed velit29/09/2012

Oprócz zwykłych typów danych wejściowych (`<TextInput>`, `<SelectInput>`, `<ReferenceInput>`, itp.) możesz użyć `<SearchInput>` w `filter` tablicy. To wejście zostało zaprojektowane specjalnie dla formularza filtra. To jest jak `<TextInput resettable>` ikona szkła powiększającego — dokładnie taki typ danych wejściowych, jakiego szukają użytkownicy, gdy chcą przeprowadzić wyszukiwanie pełnotekstowe.

```
import { SearchInput, TextInput } from 'react-admin';

const postFilters = [
  <SearchInput source="q" alwaysOn />
];
```

W powyższym przykładzie `q` filtr wyzwała wyszukiwanie pełnotekstowe we wszystkich polach. Twoim obowiązkiem jest zaimplementowanie funkcji filtrowania pełnego tekstu w Twoim `dataProvider` lub w Twoim interfejsie API.

Szybkie filtry

Posts

Comments

Users

Tags

Search

<input type="checkbox"/>	Id	Title	Published at ↓	Com.	Views	Tags	
>	<input type="checkbox"/>	13	Fusce massa lorem, pulvinar a posuere ut, ...	01/12/2012	✓	222	Code Music
>	<input type="checkbox"/>	12	Qui tempore rerum et voluptates	07/11/2012	✓	719	
>	<input type="checkbox"/>	11	Omnis voluptate enim similique est possi...	22/10/2012	✓	294	Photo Code
>	<input type="checkbox"/>	10	Totam vel quasi a odio et nihil	19/10/2012	✓	721	Sport Photo
>	<input type="checkbox"/>	9	A voluptas eius eveniet ut commodi dolor	16/10/2012	✓	143	
>	<input type="checkbox"/>	8	Culpa possimus quibusdam nostrum enim ...	02/10/2012	✗	557	Music Sport
>	<input type="checkbox"/>	7	Illum veritatis corrupti exercitationem sed ...	29/09/2012	✓	133	Code Photo
>	<input type="checkbox"/>	6	Minima ea vero omnis odit officiis aut	05/09/2012		208	Sport Photo
>	<input type="checkbox"/>	4	Maiores et itaque aut porro iuste	12/08/2012	✗	606	

Użytkownicy zwykle nie lubią używać klawiatury do filtrowania listy (zwłaszcza na urządzeniach mobilnych). Dobrym sposobem na spełnienie tego wymagania użytkownika jest przekształcenie filtrów w *szybki filtr* . Szybki filtr to filtr z nieedytowalnym `defaultValue` . Użytkownicy mogą je tylko włączać lub wyłączać.

Oto jak zaimplementować ogólny `<QuickFilter>` komponent:

```
import { SearchInput } from 'react-admin';
import { Chip } from '@mui/material';

const QuickFilter = ({ label }) => {
  const translate = useTranslate();
  return <Chip sx={{ marginBottom: 1 }} label={translate(label)} />;
};

const postFilters = [
  <SearchInput source="q" alwaysOn />,
  <QuickFilter source="commentable" label="Commentable" defaultValue={true} />,
  <QuickFilter source="views_lte" label="Low views" defaultValue={150} />,
  <QuickFilter source="tags" label="Tagged Code" defaultValue={[3]} />,
];
```

Wskazówka : obecnie nie można używać dwóch szybkich filtrów dla tego samego źródła.

Pasek <FilterList>boczny

☰ Customers

Dashboard

Sales

Orders

Invoices

Catalog

Customers

Customers

Segments

Reviews

Search

🔍

🕒 LAST VISITED

Today

This week

Last week

This month

Last month

Earlier

💰 HAS ORDERED

Yes

No

✉ HAS NEWSLETTER

Yes

No

🏷 SEGMENT

Compulsive


Collector


Ordered once


Regular


Returns


☐ Customer


☐  Graham Mohr


☐  Candice Hilll


☐  Annamae Zemlak


☐  Emmanuel Hauck


☐  Thalia Quitzon


☐  Eva Osinski


☐  Maia Johnston


☐  Gianni Koch


☐  Imani Sipes


☐  Lisa Kunze


☐  Maggie Robel


☐  Ilene Beier


☐  Christine Shanahan


☐  Jedediah Cummings


☐  Garnett Simonis

☐  Xavier Gottlieb

☐  Edgardo Zemlak

☐  Oma Cole

☐  Winfield Hilll

☐  Walton Hane

000005

Alternatywnym interfejsem użytkownika dla kombinacji przycisk/formularz filtru jest pasek boczny `FilterList`. Podobnie do tego, co użytkownicy zwykle widzą w witrynach e-commerce, jest to panel z wieloma prostymi filtrami, które można włączać i łączyć za pomocą myszy. Środowisko użytkownika jest lepsze niż kombinacja przycisku/formularza, ponieważ wartości filtrów są jawne i nie wymagają wpisywania niczego w formularzu. Jest jednak nieco mniej wydajny, ponieważ w `<FilterList>`.

Sprawdź [dokumentację <FilterList>](#), aby uzyskać więcej informacji.

Jeśli używasz `FilterList`, prawdopodobnie będziesz potrzebować danych wejściowych wyszukiwania. Ponieważ pasek boczny `FilterList` nie jest formularzem, wymaga to trochę dodatkowej pracy. Na szczęście, `React-admin` udostępnia w tym celu wyspecjalizowany komponent wejściowy wyszukiwania: sprawdź [dokumentację <FilterLiveSearch>](#), aby uzyskać szczegółowe informacje.

☰

Customers

~POSTE

Dashboard

Sales

Orders

Invoices

Catalog

Customers

Customers

Segments

Reviews

Search

🔍

🕒 LAST VISITED

Today

This week

Last week

This month

Last month

Earlier

💰 HAS ORDERED

Yes

No

✉ HAS NEWSLETTER

Yes

No

🏷 SEGMENT

Compulsive

Collector

Ordered once

Regular

Returns


☐

Customer

L


s

☐

Graham Mohr


0

☒

Edison Ledner


0

☐

Dovie Howell


3

☐

Maeve Witting


2

☐

Rogers Wilkinson


2

☐

Toby Denesik


2

☐

Willis Ondricka


2

☐

Godfrey Gulowski


1

☐

Winnifred Ondricka


1

☐

Shaun Zulauf


0

☐

Judy Murazik

1

☐

Daija Rippin

2

Operatorzy filtrów

Wewnętrznym formatem przechowywania filtrów i przesyłania ich do `dataProvider` jest obiekt, np.:

```
{ commentable: true, q: "lorem " }
```

Jest to dobre dla filtrów równości, ale jak można zrobić bardziej złożone filtry, takie jak „pomiędzy”, „zawiera”, „zaczyna się od”, „większe niż”

<https://marmelab.com/react-admin/doc/4.0/FilteringTutorial.html#the-filterlist-sidebar>

6/12

Ponieważ nie ma standardowego sposobu przekazywania tak złożonych filtrów do API, React-admin nie podejmuje w tej sprawie decyzji. Od Ciebie zależy, jak przechowywać je w obiekcie filtrującym.

Prezentacje pokazują jeden możliwy sposób: do nazwy filtra należy dodać operator, np. „_gte” dla „większego lub równego”.

```
const postFilters = [
  <DateInput source="released_gte" label="Released after" />,
  <DateInput source="released_lte" label="Released before" />
];
```

Niektóre backendy API (np. JSON Server) wiedzą, jak obsługiwać tę składnię. Jeśli Twój interfejs API nie rozumie tych „pól wirtualnych”, będziesz musiał przekształcić je w oczekiwaną składnię w dostawcy danych.

```
// in dataProvider.js
export default {
  getList: (resource, params) => {
    // transform a filter object to a filters array with operators
    // filter is like { commentable: true, released_gte: '2018-01-01' }
    const filter = params.filter;
    const operators = { '_gte': '>=', '_lte': '<=', '_neq': '!= ' };
    // filters is like [
    //   { field: "commentable", operator: "=", value: true },
    //   { field: "released", operator: ">=", value: '2018-01-01' }
    // ]
    const filters = Object.keys(filter).map(key => {
      const operator = operators[key.slice(-4)];
      return operator
        ? { field: key.slice(0, -4), operator, value: filter[key] }
        : { field: key, operator: '=', value: filter[key] };
    });
    const query = {
      pagination: params.pagination,
      sort: params.sort,
      filter: filters,
    };
    const url = `${apiUrl}/${resource}?${stringify(query)}`;
    return httpClient(url).then(({ json }) => ({
      data: json,
      total: parseInt(headers.get('content-range').split('/').pop(), 10),
    }));
  },
  // ...
}
```

Zapisane zapytania: pozwól użytkownikom zapisywać filtrowanie i sortowanie

Songs

Zapisane zapytania umożliwiają użytkownikom zapisywanie kombinacji filtrów i sortowanie parametrów w nowym, osobistym filtrze. Zapisane zapytania utrzymują się między sesjami, dzięki czemu użytkownicy mogą znaleźć swoje niestandardowe zapytania nawet po zamknięciu i ponownym otwarciu administratora. Zapisane zapytania są dostępne zarówno dla kombinacji Przycisk filtru/formularz, jak i dla <FilterList>paska bocznego. Jest on domyślnie włączony dla kombinacji Przycisk filtra/formularz, ale musisz dodać go samodzielnie na <FilterList>pasku bocznym.

<SavedQueriesList>jest uzupełnieniem <FilterList>sekcji dla paska bocznego filtra

```
import { FilterList, FilterListItem, List, Datagrid } from 'react-admin';
import { Card, CardContent } from '@mui/material';

+import { SavedQueriesList } from 'react-admin';

const SongFilterSidebar = () => (
  <Card>
    <CardContent>
      + <SavedQueriesList />
      <FilterList label="Record Company" icon={ <BusinessIcon /> }>
        ...
      </FilterList>
      <FilterList label="Released" icon={ <DateRangeIcon /> }>
        ...
      </FilterList>
    </CardContent>
  </Card>
);

const SongList = props => (
  <List {...props} aside={ <SongFilterSidebar /> }>
    <Datagrid>
      ...
    </Datagrid>
  </List>
);
```

Wyszukiwanie globalne

Chociaż filtry list umożliwiają wykonywanie precyzyjnych zapytań przy użyciu kryteriów dla poszczególnych pól, użytkownicy często preferują prostsze interfejsy, takie jak wyszukiwanie pełnotekstowe. W końcu tego właśnie używają na co dzień w wyszukiwarkach, klientach poczty e-mail oraz w eksploratorze plików.

Jeśli chcesz wyświetlić wyszukiwanie pełnotekstowe, pozwalające na wyszukanie dowolnego rekordu w administratorze za pomocą jednego formularza, sprawdź [ra-search](#) , moduł Enterprise Edition . 🧑🏻

8.04.2022, 18:00

React-admin – Filtrowanie listy

Songs

Artists

Songs

<input type="checkbox"/>	Id ↑	Title	Artist	Writer	Producer	Released	Rec con
<input type="checkbox"/>	1	Like a Rolling Stone	Bob Dylan	Bob Dylan	Tom Wilson	01/07/1965	Col
<input type="checkbox"/>	2	(I Can't Get No) Satisfaction	The Rolling Stones	Mick Jagger, Keith Richards	Andrew Loog Oldham	01/05/1965	Lor
<input type="checkbox"/>	3	Imagine	John Lennon	John Lennon	Lennon, Phil Spector, Yoko Ono	01/10/1971	App
<input type="checkbox"/>	4	What's Going On	Marvin Gaye	Gaye, Renaldo Benson, Al Cleveland	Gaye	01/02/1971	Tan

Rows per page: 10

ra-searchmożna podłączyć do dowolnej istniejącej wyszukiwarki (ElasticSearch, Lucene lub własnej wyszukiwarki) i umożliwia dostosowanie wyników wyszukiwania w celu zapewnienia szybkiej nawigacji do powiązanych elementów, zmieniając wyszukiwarkę w „omniboks”:

~POSTERS GALORE~

Search

ENGLISH

Dashboard

Catalog

Orders

Invoices

Audience

Reviews

Stores

Welcome to the react-admin enterprise edition demo

This is the admin of an imaginary poster shop showcasing enterprise edition private modules usage. Feel free to explore and modify the data - it's local to your computer, and will reset each time you reload.

SEE THE SHOWCASE

REACT-ADMIN ENTERPRISE EDITION SITE

Monthly Revenue

3012 \$US

New Orders

19

Pending Reviews

4

New Custom

30 Day Revenue History

Miracle Bode

Betty Brown

Arno Fadel

Jordan Wilkinson

Davin Goodwin

Amira Pfannerstill

Caleigh Toy

Aby uzyskać szczegółowe informacje o trybie wyszukiwania globalnego, sprawdź ra-searchmoduł w React-Admin Enterprise Edition.

Budowanie niestandardowego filtra

Posts

Comments

Tags

Jeśli ani kombinacja przycisku/formularza Filtruj, ani `<FilterList>` paska bocznego nie odpowiadają Twoim potrzebom, zawsze możesz stworzyć własny. React-admin udostępnia skróty ułatwiające tworzenie niestandardowych filtrów.

Na przykład domyślnie kombinacja przycisk/formularz filtru nie udostępnia przycisku przesyłania i jest przesyłana automatycznie po zakończeniu interakcji użytkownika z formularzem. Zapewnia to płynne działanie użytkownika, ale w przypadku niektórych interfejsów API może powodować zbyt wiele wywołań.

W takim przypadku rozwiązaniem jest przetwarzanie filtru, gdy użytkownicy klikają przycisk przesyłania, a nie podczas wpisywania wartości w danych wejściowych formularza. React-admin nie dostarcza żadnego komponentu do tego, ale jest to dobra okazja do zilustrowania wewnętrznej funkcjonalności filtra. W rzeczywistości udostępnimy alternatywną implementację do kombinacji przycisk/formularz filtru.

Aby utworzyć niestandardowy interfejs użytkownika filtra, musimy zastąpić domyślny składnik paska narzędzi List, który będzie zawierał zarówno przycisk filtru, jak i formularz filtru, współdziałając z filtrami listy za pośrednictwem `ListContext`.

Filtruj wywołania zwrotne

Nowy element może używać [haka do useListContext](#) łatwiejszej interakcji z filtrem listy. Hak zwraca następujące stałe:

- `filterValues`: Wartość filtrów na podstawie URI, np. `{ "commentable": true, "q": "lorem" }`
- `setFilters()`: Oddzwon, aby ustawić wartości filtra, np. `setFilters({ "commentable": true })`
- `displayedFilters`: Nazwy filtrów wyświetlanych w formularzu, np. `['commentable', 'title']`
- `showFilter()`: Callback w celu wyświetlenia dodatkowego filtra w formularzu, np. `showFilter('views')`
- `hideFilter()`: Callback, aby ukryć filtr w formularzu, np. `hideFilter('title')`

Wykorzystajmy tę wiedzę do napisania niestandardowego `<List>` komponentu, który filtruje po przesłaniu.

Przycisk filtra niestandardowego

Poniższy komponent pokazuje formularz filtrowania po kliknięciu. Skorzystamy z `showFilter` funkcji:

```
import { useListContext } from 'react-admin';
import { Button } from '@mui/material';
import ContentFilter from '@mui/icons-material/FilterList';

const PostFilterButton = () => {
  const { showFilter } = useListContext();
  return (
    <Button
      size="small"
      color="primary"
      onClick={() => showFilter("main")}
      startIcon={<ContentFilter />}
    >
      Filter
    </Button>
  );
};
```

Zwykle `showFilter()` dodaje jedno wejście do `displayedFilters` listy. Ponieważ formularz filtra będzie całkowicie ukryty lub pokazany, używamy `showFilter()` wirtualnego „głównego” wejścia, które reprezentuje cały formularz.

Formularz filtra niestandardowego

Dalej jest komponent formularza filtra, wyświetlany tylko wtedy, gdy wyświetlany jest filtr „główny” (tj. gdy użytkownik kliknął przycisk filtra). Dane wejściowe formularza pojawiają się bezpośrednio w formularzu, a przesłanie formularza wyzwala `setFilters()` wywołanie zwrotne przekazane jako parametr. Do `react-hook-form` obsługi stanu formularza użyjemy:

```

import * as React from 'react';
import { useForm } from 'react-hook-form';
import { Box, Button, InputAdornment } from '@mui/material';
import SearchIcon from '@mui/icons-material/Search';
import { TextInput, NullableBooleanInput, useListContext } from 'react-admin';

const PostFilterForm = () => {
  const {
    displayedFilters,
    filterValues,
    setFilters,
    hideFilter
  } = useListContext();

  const form = useForm({
    defaultValues: filterValues,
  });

  if (!displayedFilters.main) return null;

  const onSubmit = (values) => {
    if (Object.keys(values).length > 0) {
      setFilters(values);
    } else {
      hideFilter("main");
    }
  };

  const resetFilter = () => {
    setFilters({}, []);
  };

  return (
    <form onSubmit={form.handleSubmit(onSubmit)}>
      <Box display="flex" alignItems="flex-end" mb={1}>
        <Box component="span" mr={2}>
          {/* Full-text search filter. We don't use <SearchFilter> to force a large form input */}
          <TextInput
            resettable
            helperText={false}
            source="q"
            label="Search"
            InputProps={{
              endAdornment: (
                <InputAdornment>
                  <SearchIcon color="disabled" />
                </InputAdornment>
              )
            }}
          />
        </Box>
        <Box component="span" mr={2}>
          {/* Commentable filter */}
          <NullableBooleanInput
            helperText={false}
            source="commentable"
          />
        </Box>
        <Box component="span" mr={2} mb={1.5}>
          <Button variant="outlined" color="primary" type="submit">
            Filter
          </Button>
        </Box>
        <Box component="span" mb={1.5}>
          <Button variant="outlined" onClick={resetFilter}>
            Close
          </Button>
        </Box>
      </form>
    </Box>
  );
};

```

Korzystanie z niestandardowych filtrów w działaniach listy

Aby zakończyć, utwórz <ListAction>komponent i przełącz go do <List>komponentu za pomocą actionsrekwizytu:

```
import { TopToolbar, ExportButton } from 'react-admin';
import { Box } from '@mui/material';

const ListActions = () => (
  <Box width="100%">
    <TopToolbar>
      <PostFilterButton />
      <ExportButton />
    </TopToolbar>
    <PostFilterForm />
  </Box>
);

export const PostList = (props) => (
  <List {...props} actions={<ListActions />}>
    ...
  </List>
);
```

Wskazówka : nie trzeba już przekazywać żadnych `filters` do listy, ponieważ `<PostFilterForm>` komponent je wyświetli.

Możesz użyć podobnego podejścia, aby zaoferować alternatywne środowiska użytkownika do filtrowania danych, np. aby wyświetlić filtry jako wiersz w nagłówkach datagrid.