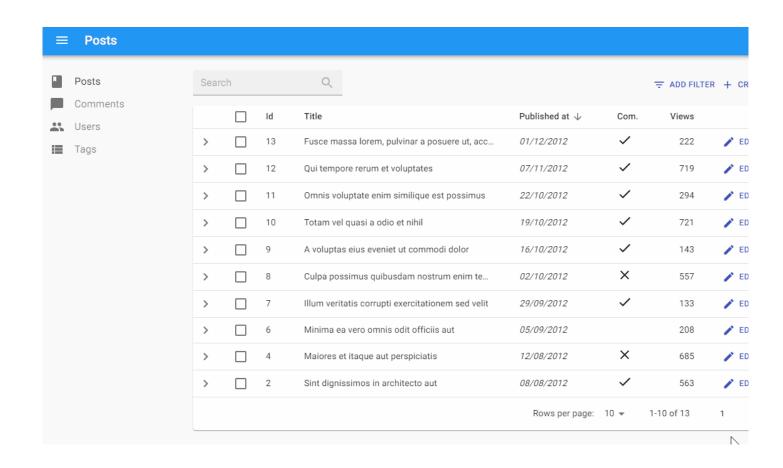
# **Filtrowanie listy**



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:

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

Po zdekodowaniu filter parametr 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 utrwala wartość filtra w stanie aplikacji, dzięki czemu gdy użytkownik wróci do listy, powinien zobaczyć przefiltrowaną listę. To wybór projektu.

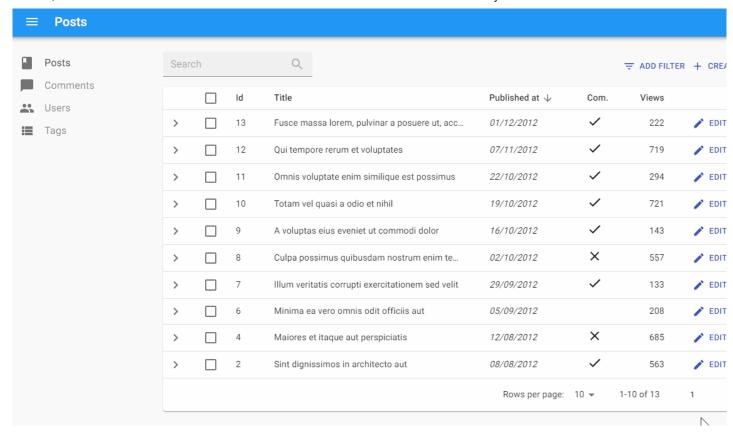
### Łączenie z wstępnie przefiltrowaną listą

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:

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



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>rekwizycie:

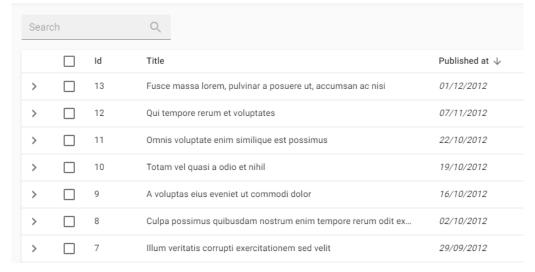
Elementy przekazywane jako filterssą zwykłymi danymi wejściowymi. Oznacza to, że możesz budować zaawansowane filtry oparte na referencjach, wartościach tablic itp. <List>Domyślnie ukrywa wszystkie dane wejściowe w formie filtra, z wyjątkiem tych, które mają właściwość alwayson.

Wskazówka: Z przyczyn technicznych, React-admin nie akceptuje danych wejściowych filtra zawierających zarówno a, jak defaultValuei alwayson. Aby ustawić domyślne wartości dla zawsze włączonych filtrów, użyj filterDefaultValuesWłaściwości <List>komponentu.

<List>używa elementów przekazanych jako filtersdwukrotnie:

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

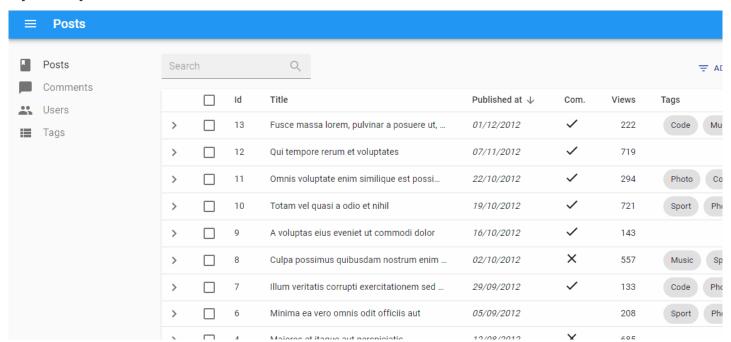
#### <SearchInput>



Oprócz zwykłych typów danych wejściowych ( <TextInput>, <SelectInput>, <ReferenceInput>, itp.) możesz użyć <SearchInput>w filterstablicy. 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.

W powyższym przykładzie qfiltr wyzwala wyszukiwanie pełnotekstowe we wszystkich polach. Twoim obowiązkiem jest zaimplementowanie funkcji filtrowania pełnego tekstu w Twoim dataProviderlub w Twoim interfejsie API.

#### Szybkie filtry



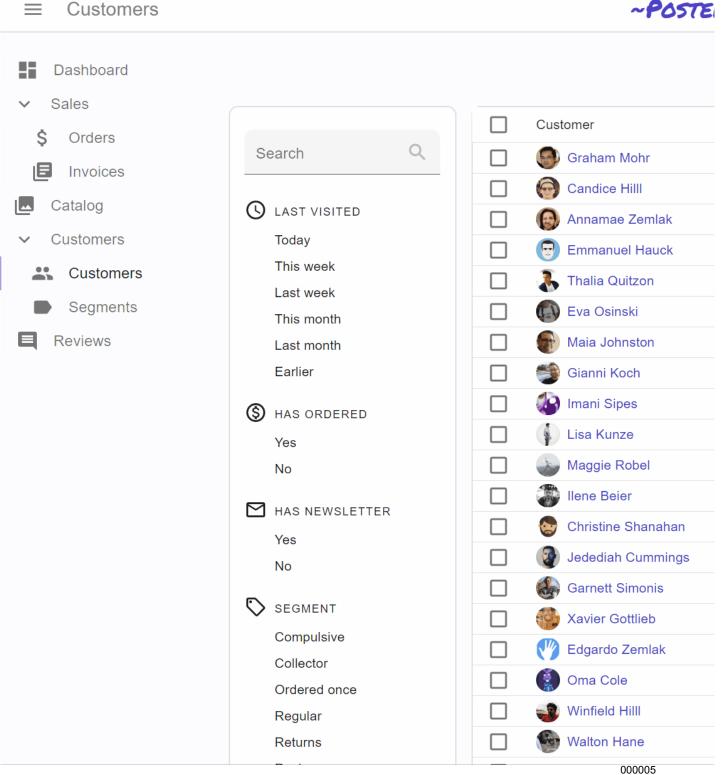
Użytkownicy zwykle nie lubią używać klawiatury do filtrowania listy (zwłaszcza na urządzeniach mobilnych). Dobrym sposobem na spełnienie tego wymagania użytkownika jest przekształcenie filtrów w szybki filtr . Szybki filtr to filtr z nieedytowalnym defaultvalue. Użytkownicy mogą je tylko włączać lub 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" always0n />,
    <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.

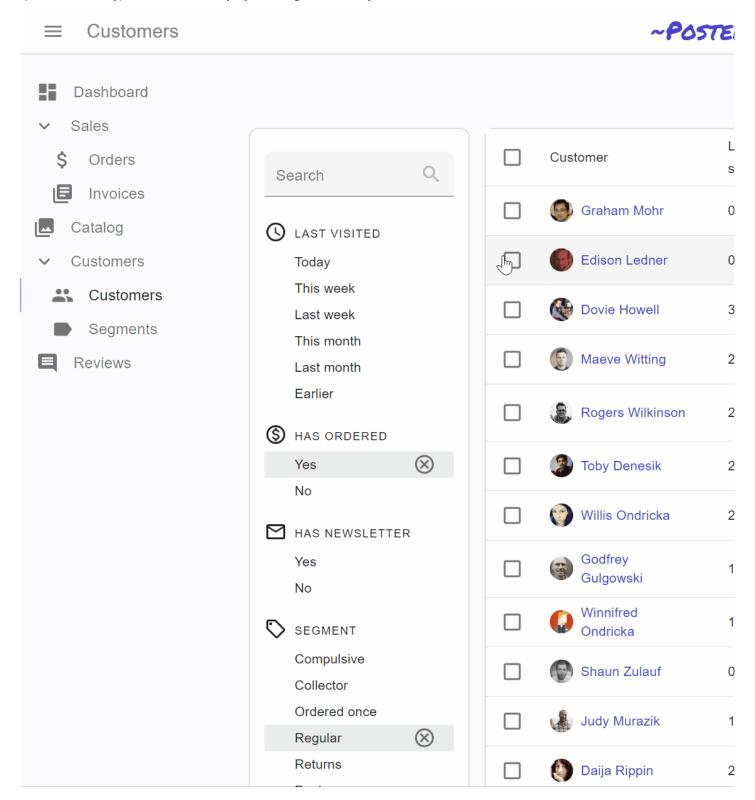
### Pasek <FilterList>boczny



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.



# **Operatorzy filtrów**

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

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

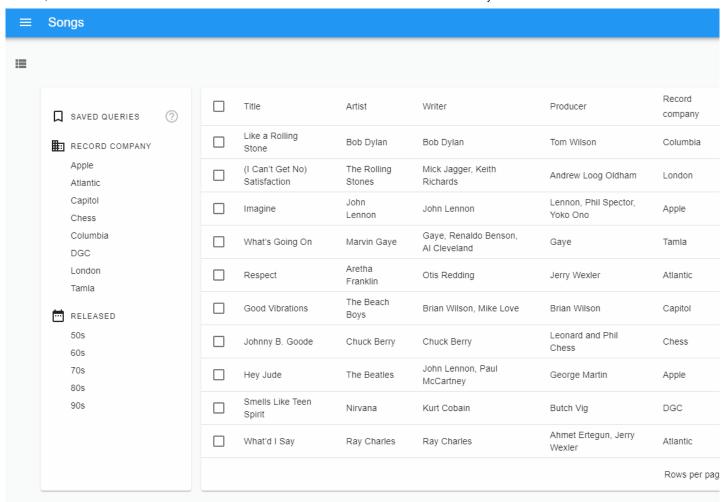
Ponieważ nie ma standardowego sposobu przekazywania tak złożonych filtrów do API, React-admin nie podejmuje w tej sprawie decyzji. Od Ciebie zależy, jak przechowywać je w obiekcie filtrującym.

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

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': '!=' };</pre>
         // 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)];
                 ? { 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



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.

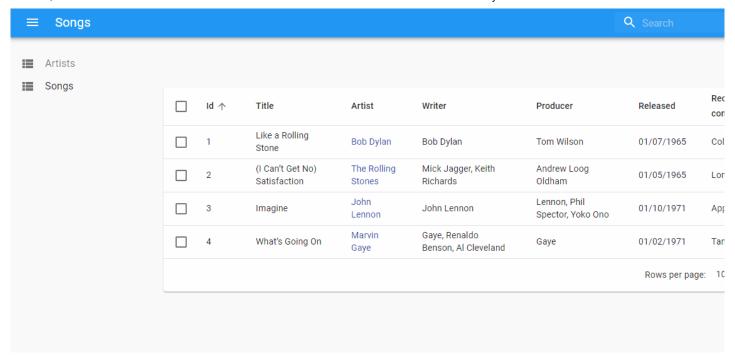
<SavedQueriesList>jest uzupełnieniem <FilterList>sekcji dla paska bocznego filtra

```
import { FilterList, FilterListItem, List, Datagrid } from 'react-admin';
import { Card, CardContent } from '@mui/material';
+import { SavedQueriesList } from 'react-admin';
const SongFilterSidebar = () => (
            <SavedQueriesList />
            <FilterList label="Record Company" icon={<BusinessIcon />}>
            </FilterList>
            <FilterList label="Released" icon={<DateRangeeIcon />}>
            </FilterList>
        </CardContent>
    </Card>
);
const SongList = props => (
    <List {...props} aside={<SongFilterSidebar />}>
        <Datagrid>
        </Datagrid>
    </List>
```

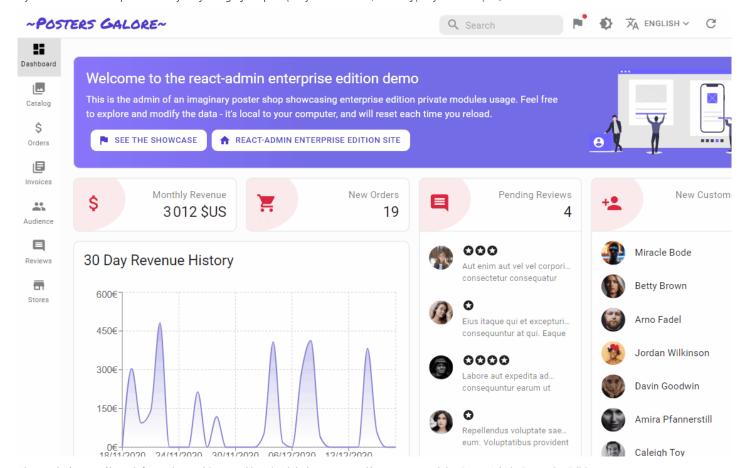
# Wyszukiwanie globalne

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 .

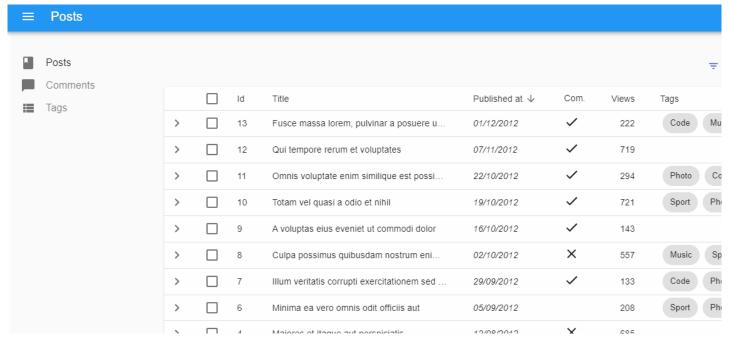


ra-searchmożna podłączyć do dowolnej istniejącej wyszukiwarki (ElasticSearch, Lucene lub własnej wyszukiwarki) i umożliwia dostosowanie wyników wyszukiwania w celu zapewnienia szybkiej nawigacji do powiązanych elementów, zmieniając wyszukiwarkę w "omniboks":



 $Aby\ uzyska\acute{c}\ szczeg\acute{o}lowe\ informacje\ o\ trybie\ wyszukiwania\ globalnego,\ sprawd\acute{z}\ ra-search moduł\ w\ React-Admin\ Enterprise\ Edition.$ 

### Budowanie niestandardowego filtra



Jeśli ani kombinacja przycisku/formularza Filtruj, ani <FilterList>paska bocznego nie odpowiadają Twoim potrzebom, zawsze możesz stworzyć własny. Reactadmin 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 useListContextlatwiejszej interakcji z filtrem listy. Hak zwraca następujące stałe:

- filterValues: Wartość filtrów na podstawie URI, np.{ "commentable": true, "q": "lorem" }
- $\bullet \quad \mathsf{setFilters()} \colon \mathsf{Oddzwo\acute{n}}, \mathsf{aby} \; \mathsf{ustawi\acute{c}} \; \mathsf{warto\acute{s}} \mathsf{ci} \; \mathsf{filtra}, \mathsf{np.setFilters(\{ \ "commentable": \mathsf{true} \ \})} \\$
- displayedFilters: Nazwy filtrów wyświetlanych w formularzu, np.['commentable', 'title']
- showFilter(): Callback w celu wyświetlenia dodatkowego filtru 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 showFilterfunkcji:

Zwykle showFilter() dodaje jedno wejście do displayedFilterslisty. 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 reacthook-formobsługi stanu formularza użyjemy:

000010

```
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 component="span" mb={1.5}>
                    <Button variant="outlined" onClick={resetFilter}>
                        Close
                    </Button>
                </Box>
            </Box>
       </form>
   );
};
```

#### Korzystanie z niestandardowych filtrów w działaniach listy

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

Wskazówka: nie trzeba już przekazywać żadnych filtersdo 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.