

Niezależnie od tego, czy potrzebujesz dostosować regułę CSS dla pojedynczego komponentu, czy zmienić kolor etykiet w całej aplikacji, wszystko jest w zasięgu!

Nadpisywanie stylu komponentu

Każdy składnik React-admin uwidacznia `sx` właściwość z MUI poza dostarczaniem `className` właściwości, która jest zawsze stosowana do elementu głównego.

Oto przykład dostosowywania `EditButton` komponentu wewnątrz `Datagrid` przy użyciu `sx` właściwości z MUI:

```
import * as React from 'react';
import { NumberField, List, Datagrid, TextField, EditButton } from 'react-admin';

const MyEditButton = (props) => (
  <EditButton
    sx={{
      fontWeight: "bold",
      // This is CSS-in-JS syntax to target a deeper element using css selector, here the svg icon for this button
      "& svg": { color: "orange" },
    }}
    {...props}
  />
);

export const ProductList = () => (
  <List>
    <Datagrid>
      <TextField source="sku" />
      <TextField source="price" />
      <MyEditButton />
    </Datagrid>
  </List>
);
```

W przypadku niektórych komponentów możesz chcieć nadpisać nie tylko styl głównego komponentu, ale także styl komponentów wewnątrz głównego. W takim przypadku możesz skorzystać z `sx` właściwości, aby dostosować interfejs API CSS, którego komponent używa wewnętrznie.

Oto przykład wykorzystujący `sx` właściwość `<Datagrid>` komponentu:

```
import * as React from 'react';
import {
  BooleanField,
  Datagrid,
  DateField,
  EditButton,
  List,
  NumberField,
  TextField,
  ShowButton,
} from 'react-admin';
import Icon from '@mui/icons-material/Person';

export const VisitorIcon = Icon;

// The `Datagrid` component uses MUI System, and supports overriding styles through the `sx` property
export const PostList = () => (
  <List>
    <Datagrid
      sx={{
        "&.RaDatagrid-table": { // No space between & and .
          backgroundColor: "Lavender",
        },
        "& .RaDatagrid-headerCell": {
          backgroundColor: "MistyRose",
        },
      }}
    >
      <TextField source="id" />
      <TextField source="title" />
      <DateField source="published_at" sortByOrder="DESC" />
      <BooleanField source="commentable" sortable={false} />
      <NumberField source="views" sortByOrder="DESC" />
      <EditButton />
      <ShowButton />
    </Datagrid>
  </List>
);
```

Ten przykład daje w wyniku:

Zapoznaj się z dokumentacją komponentu i kodem źródłowym, aby dowiedzieć się, które klasy są dostępne do stylizacji. Na przykład możesz zajrzeć do dokumentacji [i Datagrid CSS](#).

Jeśli potrzebujesz większej kontroli nad kodem HTML, możesz również utworzyć własne komponenty [pól](#) i [danych wejściowych](#).

Formatowanie warunkowe

Czasami chcesz, aby format zależał od wartości. Poniższy przykład pokazuje, jak utworzyć nowy `NumberField` komponent niestandardowy, który podświetla swój tekst na czerwono, gdy jego wartość wynosi 100 lub więcej.

```
import * as React from 'react';
import { NumberField, List, Datagrid, TextField, EditButton } from 'react-admin';

const ColoredNumberFieldStyles = {
  small: { color: 'black' },
  big: { color: 'red' },
};

const ColoredNumberField = (props) => (
  <NumberField
    sx={{
      ...(props.record[props.source] < 100 &&
        ColoredNumberFieldStyles.small),
      ...(props.record[props.source] >= 100 &&
        ColoredNumberFieldStyles.big),
    }}
    {...props}
  />
);

// Ensure the original component defaultProps are still applied as they may be used by its parents (such as the `Show` component):
ColoredNumberField.defaultProps = NumberField.defaultProps;

export const PostList = () => (
  <List>
    <Datagrid>
      <TextField source="id" />
      ...
      <ColoredNumberField source="nb_views" />
      <EditButton />
    </Datagrid>
  </List>
);
```

Co więcej, możesz wyodrębnić tę strategię wyróżniania do komponentu wyższego rzędu, jeśli chcesz ją ponownie wykorzystać również dla innych komponentów:

```
import * as React from 'react';
import { NumberField, List, Datagrid, TextField, EditButton } from 'react-admin';

const ColoredNumberFieldStyles = {
  small: { color: 'black' },
  big: { color: 'red' },
};

const colored = (WrappedComponent) => (props) =>
(
  <WrappedComponent
    sx={{
      ...(props.record[props.source] < 100 &&
        ColoredNumberFieldStyles.small),
      ...(props.record[props.source] >= 100 &&
        ColoredNumberFieldStyles.big),
    }}
    {...props}
  />
);

const ColoredNumberField = colored(NumberField);
// Ensure the original component defaultProps are still applied as they may be used by its parents (such as the `Show` component):
ColoredNumberField.defaultProps = NumberField.defaultProps;

export const PostList = () => (
  <List>
    <Datagrid>
      <TextField source="id" />
      ...
      <ColoredNumberField source="nb_views" />
      <EditButton />
    </Datagrid>
  </List>
);
```

Jeśli chcesz dowiedzieć się więcej o komponentach wyższego rzędu, zapoznaj się z samouczkiem SitePoint: Komponenty wyższego rzędu: wzorec projektowy aplikacji React

useMediaQueryHak

Aby zapewnić zoptymalizowane działanie na urządzeniach mobilnych, tabletach i komputerach stacjonarnych, często trzeba wyświetlać różne komponenty w zależności od rozmiaru ekranu. MUI udostępnia hook dedykowany do pomocy w takich responsywnych układach: [useMediaQuery](#).

Oczekuje funkcji otrzymującej motyw MUI jako parametr i zwracającej zapytanie o media. Użyj punktów przerwania motywu, aby sprawdzić typowe rozmiary ekranu. Przechwycenie zwraca wartość logiczną wskazującą, czy bieżący ekran pasuje do zapytania o media, czy nie.

```
const isXSmall = useMediaQuery(theme => theme.breakpoints.down('xs'));
const isSmall = useMediaQuery(theme => theme.breakpoints.down('sm'));
const isDesktop = useMediaQuery(theme => theme.breakpoints.up('md'));
```

Możesz także przekazać niestandardowe zapytanie o media jako ekran.

```
const isSmall = useMediaQuery('(min-width:600px)');
```

Oto przykład responsywnej listy postów, wyświetlanej SimpleListna urządzeniu mobilnym i w Datagridinny sposób:

```
// in src/posts.js
import * as React from 'react';
import { useMediaQuery } from '@mui/material';
import { List, SimpleList, Datagrid, TextField, ReferenceField, EditButton } from 'react-admin';

export const PostList = () => {
  const isSmall = useMediaQuery(theme => theme.breakpoints.down('sm'));
  return (
    <List>
      {isSmall ? (
        <SimpleList
          primaryText={record => record.title}
          secondaryText={record => `${record.views} views`}
          tertiaryText={record => new Date(record.published_at).toLocaleDateString()}
        />
      ) : (
        <Datagrid>
          <TextField source="id" />
          <ReferenceField label="User" source="userId" reference="users">
            <TextField source="name" />
          </ReferenceField>
          <TextField source="title" />
          <TextField source="body" />
          <EditButton />
        </Datagrid>
      )}
    </List>
  );
};
```

Wskazówka : poprzednie wersje React-admin udostępniały <Responsive>komponent do wykonywania zapytań o media. Ten składnik jest teraz przestarzały. Użyj `useMediaQuery` zamiast.

Korzystanie z predefiniowanego motywu

MUI obsługuje również kompletne motywy po wyjęciu z pudełka. MUI dostarcza dwa podstawowe motywy: jasny i ciemny. React-admin domyślnie używa jasnego. Aby użyć ciemnego, przekaz go <Admin>komponentowi w `themerekwizycie`.

```
const theme = {
  palette: {
    mode: 'dark', // Switching the dark mode on is a single property value change.
  },
};

const App = () => (
  <Admin theme={theme} dataProvider={simpleRestProvider('http://path.to.my.api')}>
    // ...
  </Admin>
);
```

Pisanie niestandardowego motywu

Jeśli potrzebujesz dokładniejszego dostrojenia, będziesz musiał napisać własny `theme` obiekt, postępując zgodnie z [dokumentacją motywów MUI](#).

Na przykład, oto jak zastąpić domyślny motyw React-admin:

```
import { defaultTheme } from 'react-admin';
import merge from 'lodash/merge';
import indigo from '@mui/material/colors/indigo';
import pink from '@mui/material/colors/pink';
import red from '@mui/material/colors/red';

const myTheme = merge({}, defaultTheme, {
  palette: {
    primary: indigo,
    secondary: pink,
    error: red,
    contrastThreshold: 3,
    tonalOffset: 0.2,
  },
  typography: {
    // Use the system font instead of the default Roboto font.
    fontFamily: ['-apple-system', 'BlinkMacSystemFont', '"Segoe UI"', 'Arial', 'sans-serif'].join(','),
  },
  components: {
    MuiButton: { // override the styles of all instances of this component
      styleOverrides: {
        root: { // Name of the rule
          color: 'white', // Some CSS
        },
      },
    },
  },
});
```

Obiekt theme może zawierać następujące klucze:

- breakpoints
- direction
- mixins
- components
- palette
- props
- shadows
- spacing
- transitions

- typography
- zIndex

Porada : Sprawdź [dokumentację domyślnego motywu MUI](#), aby zobaczyć wartości domyślne i znaczenie tych kluczy.

Po zdefiniowaniu motywu przekaż go do <Admin> komponentu w theme rekwizycie.

```
const App = () => (
  <Admin theme={myTheme} dataProvider={simpleRestProvider('http://path.to.my.api')}>
    // ...
  </Admin>
);
```

Programowa zmiana motywu

React-admin udostępnia useThemezaczep do programowego odczytywania i aktualizowania motywu. Używa tej samej składni co useState:

```
import { defaultTheme, useTheme } from 'react-admin';
import { Button } from '@mui/material';

const lightTheme = defaultTheme;
const darkTheme = {
  ...defaultTheme,
  palette: {
    mode: 'dark',
  },
};

const ThemeToggler = () => {
  const [theme, setTheme] = useTheme();

  return (
    <Button onClick={() => setTheme(theme.palette.mode === 'dark' ? lightTheme : darkTheme)}>
      {theme.palette.mode === 'dark' ? 'Switch to light theme' : 'Switch to dark theme'}
    </Button>
  );
}
```

Zamiast układu domyślnego możesz użyć własnego komponentu jako układu administratora. Wystarczy użyć `layout` podpory `<Admin>` komponentu:

```
// in src/App.js
import MyLayout from './MyLayout';

const App = () => (
  <Admin layout={MyLayout} dataProvider={simpleRestProvider('http://path.to.my.api')}>
    // ...
  </Admin>
);
```

Twój niestandardowy układ może rozszerzyć domyślny `<Layout>` komponent, jeśli chcesz tylko zastąpić pasek boczny, appBar, menu lub stronę błędu. Na przykład:

```
// in src/MyLayout.js
import { Layout } from 'react-admin';
import MyAppBar from './MyAppBar';
import MySidebar from './MySidebar';
import MyMenu from './MyMenu';

const MyLayout = props => <Layout
  {...props}
  appBar={MyAppBar}
  sidebar={MySidebar}
  menu={MyMenu}
/>;

export default MyLayout;
```

Dostosowywanie menu użytkownika

Możesz zastąpić domyślne menu użytkownika własnym, ustawiając `userMenu` właściwość `<AppBar>` komponentu. Na przykład, aby dodać niestandardowe elementy menu, możesz wyrenderować domyślne `<UserMenu>` i dodać do nich dzieci. Nie zapomnij dołączyć `<Logout>` komponentu, jeśli chcesz zachować pozycję menu wylogowania. Poza tym, aby poprawnie zamknąć menu po dodaniu elementu, wywołaj `onClose` metodę pobraną z `UserContext` przez `useUserMenu` hook. Jest to obsługiwane za Ciebie, jeśli używasz `<MenuItemLink>`:

```
import * as React from 'react';
import { AppBar, Logout, UserMenu, useUserMenu } from 'react-admin';
import { Link } from 'react-router-dom';
import MenuItem from '@mui/material/MenuItem';
import ListItemIcon from '@mui/material/ListItemIcon';
import ListItemText from '@mui/material/ListItemText';
import SettingsIcon from '@mui/icons-material/Settings';

// It's important to pass the ref to allow MUI to manage the keyboard navigation
const ConfigurationMenu = React.forwardRef((props, ref) => {
  return (
    <MenuItem
      ref={ref}
      component={Link}
      // It's important to pass the props to allow MUI to manage the keyboard navigation
      {...props}
      to="/configuration"
    >
      <ListItemIcon>
        <SettingsIcon />
      </ListItemIcon>
      <ListItemText>
        Configuration
      </ListItemText>
    </MenuItem>
  );
});

// It's important to pass the ref to allow MUI to manage the keyboard navigation
const SwitchLanguage = forwardRef((props, ref) => {
  const [locale, setLocale] = useLocaleState();
  // We are not using MenuItemLink so we retrieve the onClose function from the UserContext
  const { onClose } = useUserMenu();

  return (
    <MenuItem
      ref={ref}
      // It's important to pass the props to allow MUI to manage the keyboard navigation
      {...props}
      sx={{ color: 'text.secondary' }}
      onClick={event => {
        setLocale(locale === 'en' ? 'fr' : 'en');
        onClose(); // Close the menu
      }}
    >
```

```

      <ListItemIcon sx={{ minWidth: 5 }}>
        <Language />
      </ListItemIcon>
    <ListItemText>
      Switch Language
    </ListItemText>
  </MenuItem>
);
});

const MyUserMenu = props => (
  <UserMenu {...props}>
    <ConfigurationMenu />
    <SwitchLanguage />
    <Logout />
  </UserMenu>
);

const MyAppBar = props => <AppBar {...props} userMenu={<MyUserMenu />} />;

const MyLayout = props => <Layout {...props} appBar={MyAppBar} />;

```

Możesz również usunąć <UserMenu> z tego <AppBar>, przechodząc false do userMenu rek wizytu:

```

import * as React from 'react';
import { AppBar } from 'react-admin';

const MyAppBar = props => <AppBar {...props} userMenu={false} />;

const MyLayout = props => <Layout {...props} appBar={MyAppBar} />;

```

Możesz także dostosować domyślną ikonę, ustawiając icon rek wizyt na <UserMenu /> komponent.

```

import { AppBar, UserMenu } from 'react-admin';
import Avatar from '@mui/material/Avatar';

const MyCustomIcon = () => (
  <Avatar
    sx={{
      height: 30,
      width: 30,
    }}
    src="https://marmelab.com/images/avatars/adrien.jpg"
  />
);

const MyUserMenu = props => <UserMenu {...props} icon={<MyCustomIcon />} />;

const MyAppBar = props => <AppBar {...props} userMenu={<MyUserMenu />} />;

```

Dostosowywanie paska bocznego

Możesz określić sidebar szerokość, ustawiając właściwości width i closedWidth w niestandardowym motywie MUI:

```

import { defaultTheme } from 'react-admin';

const theme = {
  ...defaultTheme,
  sidebar: {
    width: 300, // The default value is 240
    closedWidth: 70, // The default value is 55
  },
};

const App = () => (
  <Admin theme={theme} dataProvider={simpleRestProvider('http://path.to.my.api')}>
    // ...
  </Admin>
);

```

Aby uzyskać bardziej zaawansowany motyw paska bocznego, przekaz własny Sidebar komponent do niestandardowego Layout:

```
import { Sidebar, Layout } from 'react-admin';

const MySidebar = (props) => (
  <Sidebar
    sx={{
      "& .RaSidebar-drawerPaper": {
        backgroundColor: "red",
      },
    }}
    {...props}
  />
);

const MyLayout = props => <Layout {...props} sidebar={MySidebar} />
```

Układ od podstaw

Aby uzyskać więcej niestandardowych układów, napisz komponent od podstaw. Musi zawierać {children} symbol zastępczy, w którym response-admin wyrenderuje zasoby. Użyj domyślnego układu jako punktu wyjścia. Oto uproszczona wersja (bez responsywnej obsługi):

```
// in src/MyLayout.js
import * as React from 'react';
import { useEffect } from 'react';
import PropTypes from 'prop-types';
import { styled } from '@mui/material';
import {
  AppBar,
  Menu,
  Sidebar,
  ComponentPropType,
  useSidebarState,
} from 'react-admin';

const Root = styled("div")(({ theme }) => ({
  display: "flex",
  flexDirection: "column",
  zIndex: 1,
  minHeight: "100vh",
  backgroundColor: theme.palette.background.default,
  position: "relative",
})));

const AppFrame = styled("div")(({ theme }) => ({
  display: "flex",
  flexDirection: "column",
  overflowX: "auto",
})));

const ContentWithSidebar = styled("main")(({ theme }) => ({
  display: "flex",
  flexGrow: 1,
})));

const Content = styled("div")(({ theme }) => ({
  display: "flex",
  flexDirection: "column",
  flexGrow: 2,
  padding: theme.spacing(3),
  marginTop: "4em",
  paddingLeft: 5,
})));

const MyLayout = ({
  children,
  dashboard,
```



```

    title,
  }) => {
    const [open] = useSidebarState();

    return (
      <Root>
        <AppFrame>
          <AppBar title={title} open={open} />
          <ContentWithSidebar>
            <Sidebar>
              <Menu hasDashboard={!!dashboard} />
            </Sidebar>
            <Content>
              {children}
            </Content>
          </ContentWithSidebar>
        </AppFrame>
      </Root>
    );
  };

  MyLayout.propTypes = {
    children: PropTypes.oneOfType([PropTypes.func, PropTypes.node]),
    dashboard: PropTypes.oneOfType([
      PropTypes.func,
      PropTypes.string,
    ]),
    title: PropTypes.string.isRequired,
  };

  export default MyLayout;

```

Dodawanie bułki tartej

Komponent `<Breadcrumb>` jest częścią `ra-navigation` modułu [Enterprise Edition](#) . Wyświetla menu nawigacyjne w oparciu o strukturę witryny, którą możesz dowolnie zmieniać.

```

import * as React from 'react';
import {
  AppLocationContext,
  Breadcrumb,
  ResourceBreadcrumbItems,
} from '@react-admin/ra-navigation';
import { Admin, Resource, Layout } from 'react-admin';

import PostList from './PostList';
import PostEdit from './PostEdit';
import PostShow from './PostShow';
import PostCreate from './PostCreate';

const MyLayout = ({ children, ...props }) => (
  <AppLocationContext>
    <Layout {...props}>
      <Breadcrumb {...props}>
        <ResourceBreadcrumbItems />
      </Breadcrumb>
      {children}
    </Layout>
  </AppLocationContext>
);

const App = () => (
  <Admin dataProvider={dataProvider} layout={MyLayout}>
    <Resource
      name="posts"
      list={PostList}
      edit={PostEdit}
      show={PostShow}
      create={PostCreate}
    />
  </Admin>
);

```

Więcej szczegółów znajdziesz w [ra-navigation dokumentacji](#).

Dostosowywanie zawartości AppBar

Domyślnie <AppBar>komponent React-admin wyświetla tytuł strony. Możesz zmienić to ustawienie domyślne, przekazując do nich dzieci <AppBar>— zastąpią one tytuł domyślny. A jeśli nadal chcesz dołączyć tytuł strony, upewnij się, że `react-admin-title` w górnym pasku znajduje się element o identyfikatorze (używa to [React Portals](#)).

Oto przykładowe dostosowanie umożliwiające <AppBar>umieszczenie logo firmy w środku nagłówek strony:

```
// in src/MyAppBar.js
import * as React from 'react';
import { AppBar } from 'react-admin';
import Typography from '@mui/material/Typography';

import Logo from './Logo';

const MyAppBar = (props) => (
  <AppBar
    sx={{
      "& .RaAppBar-title": {
        flex: 1,
        textOverflow: "ellipsis",
        whiteSpace: "nowrap",
        overflow: "hidden",
      },
    }}
    {...props}
  >
    <Typography
      variant="h6"
      color="inherit"
      className={classes.title}
      id="react-admin-title"
    />
    <Logo />
    <span className={classes.spacer} />
  </AppBar>
);

export default MyAppBar;
```

Aby użyć tego niestandardowego MyAppBarkomponentu, przekaz go jako podpowiedź do niestandardowego Layout, jak pokazano poniżej:

```
// in src/MyLayout.js
import * as React from 'react';
import { Layout } from 'react-admin';
import MyAppBar from './MyAppBar';

const MyLayout = (props) => <Layout {...props} appBar={MyAppBar} />;

export default MyLayout;
```

Następnie użyj tego układu w <Admin>z layoutrekwizytem:

```
// in src/App.js
import MyLayout from './MyLayout';

const App = () => (
  <Admin layout={MyLayout} dataProvider={simpleRestProvider('http://path.to.my.api')}>
    // ...
  </Admin>
);
```

Porada : Możesz zmienić kolor , <AppBar>ustawiając colorrekwizyt na default, inherit, lub . Wartość domyślna to .primary secondary transparent secondary

Wymiana paska aplikacji

Domyślnie React-admin używa komponentu MUI<AppBar> wraz z niestandardowym kontenerem, który wewnętrznie używa Slide do ukrywania AppBarprzewijania. Oto przykład, jak zmienić ten kontener za pomocą dowolnego komponentu:

```
// in src/MyAppBar.js
import * as React from 'react';
import { Fragment } from 'react';
import { AppBar } from 'react-admin';

const MyAppBar = props => (
  <AppBar {...props} container={Fragment} />
);

export default MyAppBar;
```

Aby wprowadzić bardziej drastyczne zmiany w górnym komponencie, prawdopodobnie będziesz chciał stworzyć <AppBar> od zera zamiast po prostu przekazywać dzieci do response-admin's <AppBar>. Oto przykładowy górny pasek przebudowany od podstaw:

```
// in src/MyAppBar.js
import * as React from 'react';
import AppBar from '@mui/material/AppBar';
import Toolbar from '@mui/material/Toolbar';
import Typography from '@mui/material/Typography';

const MyAppBar = props => (
  <AppBar {...props}>
    <Toolbar>
      <Typography variant="h6" id="react-admin-title" />
    </Toolbar>
  </AppBar>
);

export default MyAppBar;
```

Zwróć uwagę, że używa to *MUI* <AppBar> zamiast *React-admin* <AppBar>. Aby użyć tego AppBar komponentu niestandardowego, przekaz go jako właściwość do niestandardowego Layout, jak wyjaśniono w poprzedniej sekcji.

Aby ułatwić dostosowywanie, eksportujemy niektóre komponenty i haki używane przez <AppBar>:

- <LoadingIndicator>: CircularProgress Powiązanie z działaniem dataProvider.
- <SidebarToggleButton>: IconButton Służy do przełączania <Sidebar>.
- useSidebarState: Hook, który zwraca stan otwarcia paska bocznego i funkcję do jego przełączania. Używane wewnętrznie przez <SidebarToggleButton>.

Dodawanie obsługi trybu ciemnego

Komponent <ToggleThemeButton> umożliwia użytkownikom przełączanie się z trybu jasnego na ciemny i utrzymuje ten wybór, wykorzystując [sklep](#).

Możesz dodać <ToggleThemeButton> do niestandardowego paska aplikacji:

```
import * as React from 'react';
import { defaultTheme, Layout, AppBar, ToggleThemeButton } from 'react-admin';
import { createTheme, Box, Typography } from '@mui/material';

const darkTheme = createTheme({
  palette: { mode: 'dark' },
});

const MyAppBar = props => (
  <AppBar {...props}>
    <Box flex="1">
      <Typography variant="h6" id="react-admin-title"></Typography>
    </Box>
    <ToggleThemeButton
      lightTheme={defaultTheme}
      darkTheme={darkTheme}
    />
  </AppBar>
);

const MyLayout = props => <Layout {...props} appBar={MyAppBar} />;
```

Korzystanie z menu niestandardowego

Domyślnie React-admin używa listy <Resource> komponentów przekazanych jako elementy podrzędne <Admin> do budowania menu dla każdego zasobu z *List* komponentem. Jeśli chcesz zmienić kolejność, dodać lub usunąć elementy menu, na przykład, aby połączyć się ze stronami bez zasobów, musisz dostarczyć niestandardowy <Menu> komponent do swojego Layout.

Przykład menu niestandardowego

```
// in src/Menu.js
import * as React from 'react';
import { DashboardMenuItem, Menu, MenuItemLink } from 'react-admin';
import BookIcon from '@mui/icons-material/Book';
import ChatBubbleIcon from '@mui/icons-material/ChatBubble';
import PeopleIcon from '@mui/icons-material/People';
import LabelIcon from '@mui/icons-material/Label';

export const Menu = (props) => (
  <Menu {...props}>
    <DashboardMenuItem />
    <MenuItemLink to="/posts" primaryText="Posts" leftIcon={BookIcon} />
    <MenuItemLink to="/comments" primaryText="Comments" leftIcon={ChatBubbleIcon} />
    <MenuItemLink to="/users" primaryText="Users" leftIcon={PeopleIcon} />
    <MenuItemLink to="/custom-route" primaryText="Miscellaneous" leftIcon={LabelIcon} />
  </Menu>
);
```

Aby użyć tego niestandardowego komponentu menu, przekaż go do niestandardowego układu, jak wyjaśniono powyżej:

```
// in src/Layout.js
import { Layout } from 'react-admin';
import { Menu } from './Menu';

export const Layout = (props) => <Layout {...props} menu={Menu} />;
```

Następnie użyj tego układu w <Admin> layout rekwizycie:


```
// in src/App.js
import { Layout } from './Layout';

const App = () => (
  <Admin layout={Layout} dataProvider={simpleRestProvider('http://path.to.my.api')}>
    // ...
  </Admin>
);
```

Wskazówka : możesz wygenerować elementy menu dla każdego z zasobów, czytając kontekst Konfiguracje zasobów:

```
// in src/Menu.js
import * as React from 'react';
import { createElement } from 'react';
import { useMediaQuery } from '@mui/material';
import { DashboardMenuItem, Menu, MenuItemLink, useResourceDefinitions, useSidebarState } from 'react-admin';
import DefaultIcon from '@mui/icons-material/ViewList';
import LabelIcon from '@mui/icons-material/Label';

export const Menu = (props) => {
  const resources = useResourceDefinitions();
  const [open] = useSidebarState();
  return (
    <Menu {...props}>
      <DashboardMenuItem />
      {Object.keys(resources).map(name => (
        <MenuItemLink
          key={name}
          to={`/${name}`}
          primaryText={
            (resources[name].options && resources[name].options.label) ||
            name
          }
          leftIcon={
            resources[name].icon ? <resource.icon /> : <DefaultIcon />
          }
          onClick={props.onMenuClick}
          sidebarIsOpen={open}
        />
      ))}
      {/* add your custom menus here */}
    </Menu>
  );
};
```

Wskazówka : jeśli potrzebujesz wielopoziomowego menu lub paneli otwierających Mega Menu z niestandardową zawartością, sprawdź [moduł \(ra-navigation\)](#)  część Enterprise Edition)

Komponent `<MenuItemLink>` wyświetla pozycję menu z etykietą i ikoną — lub tylko ikonę z podpowiedzią, gdy pasek boczny jest zminimalizowany. Obsługuje również automatyczne zamykanie menu w telefonie komórkowym.

Właściwość `primaryText` akceptuje ciąg znaków lub węzeł React. Możesz go użyć np. do wyświetlenia znacznika na górze pozycji menu:

```
import Badge from '@mui/material/Badge';

<MenuItemLink to="/custom-route" primaryText={
  <Badge badgeContent={4} color="primary">
    Notifications
  </Badge>
} />
```

`leftIcon` Rekwizyt pozwala ustawić lewą ikonę menu .

Dodatkowe właściwości są przekazywane do podrzędnego komponentu MUI `<MenuItem>` .

Wskazówka : `<MenuItemLink>` Komponent korzysta z komponentu React Router `NavLink` , co pozwala na dostosowanie stylu aktywnego menu. Na przykład, oto jak użyć niestandardowego motywu, aby wyświetlić lewą ramkę aktywnego menu:

```
export const theme = {
  palette: {
    // ...
  },
  overrides: {
    RaMenuItemLink: {
      active: {
        borderLeft: '3px solid #4f3cc9',
      },
      root: {
        borderLeft: '3px solid #fff', // invisible menu when not active, to avoid scrolling the text when selecting the menu
      },
    },
  },
};
```

Menu do listy filtrowanej

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

Na przykład, aby dołączyć menu do listy opublikowanych postów:

```
<MenuItemLink
  to={{
    pathname: '/posts',
    search: `filter=${JSON.stringify({ is_published: true })}`,
  }}
  primaryText="Posts"
  leftIcon={<BookIcon />}
/>
```

Menu do listy bez filtrów

Domyślnie kliknięcie `<MenuItemLink>` strony z listą otwiera listę z tymi samymi filtrami, które zostały zastosowane ostatnim razem, gdy użytkownik je widział. Zwykle jest to oczekiwane zachowanie, ale użytkownicy mogą preferować, aby kliknięcie elementu menu resetowało filtry listy.

Po prostu użyj pustego `filter` parametru zapytania, aby wymusić puste filtry:

```
<MenuItemLink
  to="/posts?filter=%7B%7D" // %7B%7D is JSON.stringify({})
  primaryText="Posts"
  leftIcon={<BookIcon />}
/>
```

Korzystanie z niestandardowej strony logowania

Domyślnie strona logowania wyświetla tło gradientowe. Jeśli chcesz zmienić tło, możesz użyć domyślnego komponentu strony logowania i przekazać URL obrazu jako `backgroundImage` rekwizyt.

```
import { Admin, Login } from 'react-admin';

const MyLoginPage = () => (
  <Login
    // A random image that changes everyday
    backgroundImage="https://source.unsplash.com/random/1600x900/daily"
  />
);

const App = () => (
  <Admin loginPage={MyLoginPage}>
    // ...
  </Admin>
);
```

Korzystanie z niestandardowego przycisku wylogowania

Możliwe jest użycie całkowicie niestandardowego przycisku wylogowania lub po prostu nadpisanie niektórych właściwości przycisku domyślnego. Jeśli chcesz zmienić ikonę, możesz użyć domyślnego <Logout> komponentu i przekazać inną ikonę jako `icon` rekvizyt.

```
import { Admin, AppBar, Layout, Logout, UserMenu } from 'react-admin';
import ExitToAppIcon from '@mui/icons-material/ExitToApp';

const MyLogoutButton = props => <Logout {...props} icon={<ExitToAppIcon/>} />;

const MyUserMenu = () => <UserMenu><MyLogoutButton /></UserMenu>;

const MyAppBar = () => <AppBar userMenu={<MyUserMenu />} />;

const MyLayout = () => <Layout appBar={MyAppBar} />;

const App = () => (
  <Admin layout={MyLayout}>
    // ...
  </Admin>
);
```

Powiadomienia

Możesz nadpisać komponent powiadomienia, na przykład, aby zmienić czas trwania powiadomienia. Domyślna wartość to 4000, czyli 4 sekundy, i możesz ją zmienić za pomocą właściwości `autoHideDuration`. Na przykład, aby utworzyć niestandardowy komponent powiadomień z domyślną wartością 5 sekund:

```
// in src/MyNotification.js
import { Notification } from 'react-admin';

const MyNotification = props => <Notification {...props} autoHideDuration={5000} />;

export default MyNotification;
```

Aby użyć tego niestandardowego komponentu powiadomień, przekaz go do <Admin> komponentu jako właściwość `notification`:

```
// in src/App.js
import MyNotification from './MyNotification';
import dataProvider from './dataProvider';

const App = () => (
  <Admin notification={MyNotification} dataProvider={dataProvider}>
    // ...
  </Admin>
);
```

Dostosowywanie strony błędu

Za każdym razem, gdy wystąpi błąd po stronie klienta w React-admin, użytkownik widzi domyślny komunikat o błędzie. Jeśli chcesz dostosować tę stronę lub zarejestrować błąd w usłudze innej firmy, utwórz własny <Error> komponent. Poniższy fragment jest uproszczoną wersją komponentu `Error-React-admin`, którego możesz użyć jako bazy dla swojego własnego:

```
// in src/MyError.js
import * as React from 'react';
import Button from '@mui/material/Button';
import ErrorIcon from '@mui/icons-material/Report';
import History from '@mui/icons-material/History';
import { Title, useTranslate } from 'react-admin';
import { useLocation } from 'react-router';

const MyError = ({
  error,
  resetErrorBoundary,
  ...rest
}) => {
  const { pathname } = useLocation();
  const originalPathname = useRef(pathname);

  // Effect that resets the error state whenever the location changes
  useEffect(() => {
    if (pathname !== originalPathname.current) {
      resetErrorBoundary();
    }
  }, [pathname, resetErrorBoundary]);

  const translate = useTranslate();
  return (
    <div>
      <Title title="Error" />
      <h1><ErrorIcon /> Something Went Wrong </h1>
      <div>A client error occurred and your request couldn't be completed.</div>
      {process.env.NODE_ENV !== 'production' && (
        <details>
          <h2>{translate(error.toString())}</h2>
          {errorInfo.componentStack}
        </details>
      )}
      <div>
        <Button
          variant="contained"
          startIcon={<History />}
          onClick={() => history.go(-1)}
        >
          Back
        </Button>
      </div>
    </div>
  );
};

export default MyError;
```

Aby użyć tego niestandardowego komponentu błędu, przekaz go do niestandardowego układu, jak wyjaśniono powyżej:

```
// in src/MyLayout.js
import { Layout } from 'react-admin';
import MyError from './MyError';

const MyLayout = (props) => <Layout {...props} error={MyError} />;

export default MyLayout;
```

Następnie użyj tego układu w <Admin> layout rekwizycie:

```
// in src/App.js
import MyLayout from './MyLayout';

const App = () => (
  <Admin layout={MyLayout} dataProvider={simpleRestProvider('http://path.to.my.api')}>
    // ...
  </Admin>
);
```

Ładowanie

Wyświetl okrągły komponent postępu z opcjonalnymi komunikatami. Wyświetlaj ten sam składnik ładowania, co `react-admin` na stronach niestandardowych, aby zapewnić spójność.

Obsługiwane rekwizyty:

Rekwizyt	Wymagany	Rodzaj	Domyślna	Opisy
loadingPrimary	Opcjonalny	string	ra.page.loading	Etykieta używana do podstawowej wiadomości o ładowaniu
loadingSecondary	Opcjonalny	string	ra.message.loading	Etykieta do użycia w przypadku dodatkowej wiadomości o ładowaniu

Stosowanie:

```
<Loading loadingPrimary="app.page.loading" loadingSecondary="app.message.loading" />
```

LinearProgress

Wyświetl składnik postępu liniowego. Wyświetlaj ten sam składnik ładowania, co react-adminw przypadku niestandardowych danych wejściowych, aby zapewnić spójność.

Stosowanie:

```
({ data, ...props }) => !data ?  
  <LinearProgress /> :  
  <MyInput data={data} />;
```