patientTable

/\* eslint-disable no-bitwise \*/

import React, { memo, useCallback, useEffect, useMemo, useState } from 'react';

import PropTypes from 'prop-types';

import {

useFlexLayout,

usePagination,

useResizeColumns,

useSortBy,

useTable,

} from 'react-table';

import { Box, Table, TableBody, TableSortLabel, Tooltip } from '@mui/material';

import { DateTime } from 'luxon';

import { useCookies } from 'react-cookie';

import { visuallyHidden } from '@mui/utils';

import { useTheme } from 'styled-components';

import {

TableCell,

TableCellLabel,

TableContainer,

TableHead,

TableRow,

TableWrapper,

} from './styles';

import PatientTableToolbar from './PatientTableToolbar';

import LoadingSpinner from '../LoadingSpinner/Loadable';

import Pagination from '../Pagination/Loadable';

import NoData from '../NoData/Loadable';

import {

getCamelCaseFormattedAddress,

zoneIgnoredDate,

} from '../../utils/helpers';

import { TABLE\_ROW\_HEIGHT } from '../../utils/constants';

import EditIcon from '../../assets/EditIcon';

import SettingIcon from '../../assets/SettingIcon';

import SelectTableColumns from '../SelectTableColumns';

import { DEFAULT\_COOKIES, PATIENT\_TABLE\_COL\_COOKIE\_NAME } from './constants';

import colDragGrey from '../../images/col-drag-grey.svg';

const renderCellData = (cellData = '') => (

<Tooltip title={cellData}>

<span>{cellData}</span>

</Tooltip>

);

function PatientTable({

patients = [],

onSearchPatient,

onSelectPatient,

clearPatientSelection,

patientSearchStates,

tableParams,

isLoading,

createModalVisibilityFunc,

onPageChange,

onRowsPerPageChange,

pageOptions,

bulkUpdatePatients,

total,

onSortParamChange,

handlePatientTypeChange,

rowSelection,

setRowSelectionState,

onRowClick,

uploadFileAction,

uploadButtonState,

setIsEditing,

}) {

const { page: currentPage = 1, searchValue, isActive, limit } = tableParams;

const [cookies, setCookie] = useCookies([PATIENT\_TABLE\_COL\_COOKIE\_NAME]);

const theme = useTheme();

const [selectColumnsDrawer, setSelectColumnsDrawer] = useState(false);

const handleColumnDrawer = (value) => {

setSelectColumnsDrawer(!!value);

};

const { exceptional, allSelected } = rowSelection;

useEffect(() => {

setRowSelectionState('exceptional', []);

}, [allSelected]);

const handleChangePage = (event, newPage) => {

gotoPage(newPage - 1);

onPageChange(newPage);

};

const handleChangeRowsPerPage = (page) => {

setPageSize(page);

onRowsPerPageChange(page);

};

const getRowId = useCallback((row) => row.uuid, []);

const defaultColumns = useMemo(

() => [

{

Header: 'Name',

id: `name`,

accessor: 'name',

isChecked: true,

minWidth: 30,

maxWidth: 400,

Cell: ({ value: { firstName, lastName } }) =>

renderCellData(`${firstName} ${lastName}`),

},

{

Header: 'Member ID',

accessor: 'enrollments',

id: `insurance`,

isChecked: true,

disableSortBy: true,

minWidth: 30,

maxWidth: 400,

Cell: ({ value: [coverage = {}] }) =>

renderCellData(coverage.memberId ?? ''),

},

{

Header: 'Date of Birth',

accessor: 'dateOfBirth',

isChecked: true,

minWidth: 30,

maxWidth: 400,

Cell: ({ value: date }) => {

const dob = zoneIgnoredDate(date);

return renderCellData(

dob instanceof DateTime ? dob.toFormat('MM/dd/yyyy') : '',

);

},

},

{

Header: 'Insurance Name',

accessor: 'enrollments',

id: `insurance-name`,

isChecked: true,

disableSortBy: true,

minWidth: 30,

maxWidth: 400,

Cell: ({ value: [enrollment = {}] }) =>

renderCellData(enrollment.payer.name ?? ''),

},

{

Header: 'Phone (Mobile)',

id: `phone`,

accessor: 'contactInfo.mobilePhoneNumber',

isChecked: true,

disableSortBy: true,

minWidth: 30,

maxWidth: 400,

Cell: ({ value }) => renderCellData(value || ``),

},

// {

// Header: 'Payer',

// accessor: `enrollments`,

// id: `payer`,

// isChecked: true,

// disableSortBy: true,

// minWidth: 30,

// maxWidth: 400,

// Cell: ({ value: [coverage = {}] }) =>

// renderCellData(coverage?.payer?.name),

// },

// {

// Header: 'Is Active',

// accessor: `isActive`,

// id: `isActive`,

// isChecked: false,

// disableSortBy: true,

// minWidth: 30,

// maxWidth: 400,

// Cell: ({ value }) => (value ? `Y` : `N`),

// },

{

Header: 'Sex',

accessor: `sex`,

id: `sex`,

isChecked: false,

disableSortBy: true,

minWidth: 30,

maxWidth: 400,

Cell: ({ value }) => value || ``,

},

{

Header: 'Address',

accessor: `address.formattedAddress`,

id: `address`,

isChecked: false,

disableSortBy: true,

minWidth: 30,

maxWidth: 400,

Cell: ({ value }) =>

renderCellData(getCamelCaseFormattedAddress(value) || ``),

},

{

Header: 'Level of Service',

accessor: `levelOfService.displayText`,

id: `levelOfService`,

isChecked: false,

disableSortBy: true,

minWidth: 30,

maxWidth: 400,

Cell: ({ value }) => renderCellData(value || ``),

},

],

[],

);

const newColumns = useMemo(() => {

const headers = cookies[PATIENT\_TABLE\_COL\_COOKIE\_NAME] || DEFAULT\_COOKIES;

const columns = [[]];

headers[0].forEach((h, headIdx) => {

if (h.isChecked) {

columns[0][headIdx] = defaultColumns[h.idx];

}

});

return columns[0].filter((d) => d);

}, [cookies]);

const {

getTableProps,

headerGroups,

prepareRow,

page,

gotoPage,

setPageSize,

state: { pageSize, sortBy },

} = useTable(

{

columns: newColumns,

data: patients,

getRowId,

manualPagination: true,

pageCount: Math.ceil(total / pageSize),

initialState: { pageIndex: currentPage - 1 },

manualSortBy: true,

disableMultiSort: true,

autoResetSelectedRows: false,

},

useSortBy,

usePagination,

useResizeColumns,

useFlexLayout,

);

useEffect(() => {

const [sort] = sortBy;

if (sort) {

onSortParamChange(

sort.id === 'name' ? 'name.firstName' : sort.id,

sort.desc ? -1 : 1,

);

}

}, [sortBy]);

const renderTableData = () => {

if (isLoading)

return (

<TableBody>

<TableRow>

<TableCell

variant="body"

size="small"

colSpan={patients.length + 1}

style={{ display: 'table-cell', margin: `auto` }}

>

<LoadingSpinner />

</TableCell>

</TableRow>

</TableBody>

);

if (patients.length < 1)

return (

<TableBody>

<TableRow>

<TableCell

variant="body"

size="small"

padding="none"

colSpan={newColumns.length + 1}

>

<NoData />

</TableCell>

</TableRow>

</TableBody>

);

return (

<TableBody>

{page.map((row) => {

prepareRow(row);

const rowProps = row.getRowProps();

return (

<TableRow {...rowProps}>

{row.cells.map((cell) => (

<TableCell

onClick={() => onRowClick(row.original)}

variant="body"

size="small"

sx={{

color: theme.text.primary,

}}

// padding="none"

{...cell.getCellProps()}

>

{cell.render('Cell')}

</TableCell>

))}

<TableCell variant="body" size="small">

<EditIcon

id="edit"

cursor="pointer"

onClick={() => setIsEditing(row.original)}

/>

</TableCell>

</TableRow>

);

})}

</TableBody>

);

};

const handleColumns = (newCookieColumns) => {

setCookie(PATIENT\_TABLE\_COL\_COOKIE\_NAME, newCookieColumns);

};

return (

<TableWrapper noData={!patients?.length}>

<SelectTableColumns

handleColumns={handleColumns}

open={selectColumnsDrawer}

toggleDrawer={handleColumnDrawer}

columnsCookies={

cookies[PATIENT\_TABLE\_COL\_COOKIE\_NAME] || DEFAULT\_COOKIES

}

columnsData={[defaultColumns]}

disabledColumns={[`Name`]}

/>

<PatientTableToolbar

numSelected={

allSelected ? total - exceptional.length : exceptional.length

}

archivePatients={() => bulkUpdatePatients({ isActive: !isActive })}

isActive={isActive}

searchValue={searchValue ?? ''}

createModalVisibilityFunc={createModalVisibilityFunc}

handlePatientTypeChange={handlePatientTypeChange}

uploadFileAction={(file) => uploadFileAction(file)}

uploadButtonState={uploadButtonState}

onSearchPatient={onSearchPatient}

onSelectPatient={onSelectPatient}

clearPatientSelection={clearPatientSelection}

patientSearchStates={patientSearchStates}

PatientSearchDropdownProps={{

width: '342px',

showNextLineAddress: true,

}}

/>

<TableContainer>

<Table

{...getTableProps()}

aria-labelledby="tableTitle"

size="small"

sx={{

width: `100%`,

placeItems: 'center',

}}

>

<TableHead elem="th">

{headerGroups.map((headerGroup) => {

const headerGroupProps = headerGroup.getHeaderGroupProps();

return (

<TableRow

sx={{

fontSize: 12,

borderBottom: '1px solid rgba(224, 224, 224, 1)',

height: TABLE\_ROW\_HEIGHT,

}}

{...headerGroupProps}

>

{headerGroup.headers.map((column) => {

const headerProps =

column.id === 'selection'

? column.getHeaderProps()

: column.getHeaderProps(column.getSortByToggleProps());

if (column.disableSortBy) headerProps.title = '';

return (

<TableCell

{...headerProps}

sx={{

display: `flex !important`,

alignItems: `center`,

justifyContent: `space-between`,

color: '#595C6A',

}}

padding="none"

size="small"

variant="head"

onClick={() => null}

>

<>

<TableCellLabel onClick={headerProps.onClick}>

{column.render('Header')}

{column.id !== 'selection' ? (

<TableSortLabel

active={column.isSorted}

direction={column.isSortedDesc ? 'desc' : 'asc'}

>

{column.id !== 'selection' ? (

<Box component="span" sx={visuallyHidden}>

{column.isSortedDesc

? 'sorted descending'

: 'sorted ascending'}

</Box>

) : null}

</TableSortLabel>

) : null}

</TableCellLabel>

<div {...column.getResizerProps()}>

<img draggable={false} src={colDragGrey} alt="" />

</div>

</>

</TableCell>

);

})}

<TableCell

onClick={handleColumnDrawer}

variant="head"

size="small"

>

<SettingIcon />

</TableCell>

</TableRow>

);

})}

</TableHead>

{renderTableData()}

</Table>

</TableContainer>

<Pagination

page={currentPage}

handlePageChange={handleChangePage}

handleChangeRowsPerPage={handleChangeRowsPerPage}

total={total}

rowsPerPage={limit}

pageOptions={pageOptions}

/>

</TableWrapper>

);

}

PatientTable.propTypes = {

tableParams: PropTypes.object.isRequired,

isLoading: PropTypes.bool.isRequired,

createModalVisibilityFunc: PropTypes.func.isRequired,

patients: PropTypes.array,

header: PropTypes.array,

onPageChange: PropTypes.func,

onRowsPerPageChange: PropTypes.func,

pageOptions: PropTypes.array,

currentPage: PropTypes.number,

total: PropTypes.number,

onSortParamChange: PropTypes.func,

onSearchPatient: PropTypes.func,

onSelectPatient: PropTypes.func,

clearPatientSelection: PropTypes.func,

patientSearchStates: PropTypes.object,

handlePatientTypeChange: PropTypes.func,

setRowSelectionState: PropTypes.func,

bulkUpdatePatients: PropTypes.func,

rowSelection: PropTypes.object,

onRowClick: PropTypes.func,

uploadFileAction: PropTypes.func,

uploadButtonState: PropTypes.bool,

setIsEditing: PropTypes.func,

};

export default memo(PatientTable);

Patient Parent

import React, { memo, useEffect } from 'react';

import { compose } from 'redux';

import { batch, connect } from 'react-redux';

import PropTypes from 'prop-types';

import { createStructuredSelector } from 'reselect';

import { useLocation, useParams, useRouteMatch } from 'react-router-dom';

import { withTheme } from 'styled-components';

import { useInjectSaga } from 'utils/injectSaga';

import { useInjectReducer } from 'utils/injectReducer';

import CreatePatientModal from 'components/CreatePatientModal/Loadable';

import PatientTable from 'components/PatientTable/Loadable';

import { push } from 'connected-react-router';

import { DateTime } from 'luxon';

import { Helmet } from 'react-helmet';

import makeSelectPatientTable from './selectors';

import reducer from './reducer';

import saga from './saga';

import {

bulkUpdatePatients,

changeParams,

clearPatientFormData,

clearPatientSelection,

createPatientAction,

getPatients,

onAttemptFileUpload,

searchPayers,

selectPatientAction,

setCreateModalVisibility,

setIsEditing,

setRowSelectionState,

togglePatientCreationSuccessModalVisibility,

updatePatient,

} from './actions';

import PatientCreationSuccessModal from '../../components/StaticModals/PatientCreationSuccessModal';

import { fetchLocations, searchPatients, selectLocation } from '../App/actions';

import { removeEmptyKeys } from '../../utils/helpers';

// TODO: Rework this entire container

export function PatientTablePage({ dispatch, patientTableStore }) {

useInjectReducer({ key: 'patientTable', reducer });

useInjectSaga({ key: 'patientTable', saga });

const {

patients,

patientSearchStates,

isFileUploading,

params,

totalPatients = 0,

locationStates,

createModalVisibility,

createSuccessModalVisibility,

rowSelection,

loading,

editingPatientID,

editingPatient,

payerOptions,

} = patientTableStore;

const { selectedPatient } = patientSearchStates;

const { orgResourceId } = useParams();

const { state } = useLocation();

const { url } = useRouteMatch();

useEffect(() => {

const customParam = { ...params };

if (state) {

batch(() => {

dispatch(selectPatientAction(null));

dispatch(getPatients(orgResourceId, { ...customParam, ...state }));

});

window.history.replaceState({}, '', url);

} else if (selectedPatient) {

customParam.searchValue = null;

batch(() => {

dispatch(

getPatients(orgResourceId, {

...customParam,

...(selectedPatient && { uuid: selectedPatient.uuid }),

}),

);

});

} else {

dispatch(getPatients(orgResourceId, customParam));

}

}, [params, orgResourceId, selectedPatient, state]);

const onSearchPatient = (searchTerm) => {

if (searchTerm.trim().length) {

let customParam;

if (searchTerm.startsWith('!')) {

const dateString = searchTerm.substring(1).trim();

if (!DateTime.fromFormat(dateString, 'MM/dd/yyyy').isValid) return;

customParam = {

dateOfBirth: DateTime.fromFormat(dateString, 'MM/dd/yyyy', {

zone: 'utc',

})

.startOf('day')

.toISO(),

};

} else if (searchTerm.startsWith('#'))

customParam = { memberId: searchTerm.substring(1).trim() };

else if (searchTerm.startsWith('@'))

customParam = { name: searchTerm.substring(1).trim() };

else return;

dispatch(searchPatients(customParam, orgResourceId));

}

};

const handleBulkUpdatePatients = (updates) => {

const { exceptional, allSelected } = rowSelection;

const body = {

updates,

...(allSelected ? { excludes: exceptional } : { includes: exceptional }),

};

dispatch(bulkUpdatePatients(body, orgResourceId));

handlePatientTypeChange(!updates.isActive);

};

const handlePageChange = (newPage) => dispatch(changeParams('page', newPage));

const handleChangeRowsPerPage = (newRowsPerPage) => {

batch(() => {

dispatch(changeParams('limit', newRowsPerPage));

dispatch(changeParams('page', 1));

});

};

const handleSortParamChange = (sortBy, sortOrder) => {

batch(() => {

dispatch(changeParams('sortBy', sortBy));

dispatch(changeParams('sortOrder', sortOrder));

});

};

const handlePatientTypeChange = (type) => {

batch(() => {

dispatch(changeParams('isActive', type));

dispatch(changeParams('page', 1));

dispatch(setRowSelectionState('allSelected', false));

dispatch(setRowSelectionState('exceptional', []));

});

};

const openProfile = (patient) => {

dispatch(

push(`${url}/${patient.uuid}`, {

patientUuid: patient.uuid,

}),

);

};

const locationFunctions = {

fetchLocations: (key, searchTerm) => {

dispatch(fetchLocations(key, searchTerm));

},

selectLocation: (k, v, a) => dispatch(selectLocation(k, v, a)),

};

const submitPatientForm = (form, isEditForm, patientID = null) => {

const newValues = { ...form };

if (!isEditForm) {

dispatch(createPatientAction(removeEmptyKeys(newValues), orgResourceId));

} else {

if (newValues?.enrollments) delete newValues.enrollments;

if (newValues?.notes) delete newValues.notes;

dispatch(updatePatient(newValues, patientID, orgResourceId));

}

};

return (

<>

<Helmet>

<title>All Patients</title>

<meta name="Kinetik All Patient Page" content="Kinetik Patient List" />

</Helmet>

<CreatePatientModal

showModal={createModalVisibility}

handleClose={() => {

dispatch(setCreateModalVisibility(false));

dispatch(clearPatientFormData());

}}

selectedPatient={editingPatient}

selectedPatientID={editingPatientID}

locationStates={locationStates}

locationFunctions={locationFunctions}

isSaving={loading}

payerOptions={payerOptions}

searchPayers={(searchTerm) => dispatch(searchPayers(searchTerm))}

editMode={!!editingPatientID?.length}

onSubmitForm={submitPatientForm}

/>

<PatientCreationSuccessModal

visibility={createSuccessModalVisibility}

toggleVisibility={() =>

dispatch(togglePatientCreationSuccessModalVisibility())

}

/>

<PatientTable

patients={patients}

onSearchPatient={onSearchPatient}

onSelectPatient={(patient) => dispatch(selectPatientAction(patient))}

clearPatientSelection={() => dispatch(clearPatientSelection())}

patientSearchStates={patientSearchStates}

tableParams={params}

onPageChange={handlePageChange}

onRowsPerPageChange={handleChangeRowsPerPage}

bulkUpdatePatients={handleBulkUpdatePatients}

total={totalPatients}

isLoading={loading}

createModalVisibilityFunc={(visibility) =>

dispatch(setCreateModalVisibility(visibility))

}

onSortParamChange={handleSortParamChange}

handlePatientTypeChange={handlePatientTypeChange}

rowSelection={rowSelection}

onRowClick={openProfile}

setRowSelectionState={(key, value) =>

dispatch(setRowSelectionState(key, value))

}

uploadFileAction={(file) =>

dispatch(onAttemptFileUpload(file, orgResourceId))

}

uploadButtonState={isFileUploading}

setIsEditing={(patient) => {

dispatch(setIsEditing(patient, patient.uuid));

dispatch(setCreateModalVisibility(true));

}}

/>

</>

);

}

PatientTablePage.propTypes = {

patientTableStore: PropTypes.object,

dispatch: PropTypes.func,

};

const mapStateToProps = createStructuredSelector({

patientTableStore: makeSelectPatientTable(),

});

function mapDispatchToProps(dispatch) {

return {

dispatch,

};

}

const withConnect = connect(mapStateToProps, mapDispatchToProps);

export default compose(withConnect, withTheme, memo)(PatientTablePage);