

Brief for react / minimalists / TS

Tech

- React
- Typescript
- MUI
- start with the minimalists MIT theme
- commit to repos; <https://github.com/tluyben/mui-minimal-free-ts>
 - all information is in that repository including the figma file

Choices

We require specific tag usage because this theme is set to replace something which we are building already. This means that we need to discuss what to call things in React and how these things are exposed.

Keep the names of the react components in the project IDENTICAL to the Minimalists/MUI ones, if they are present. Just put them in another namespace (directory).

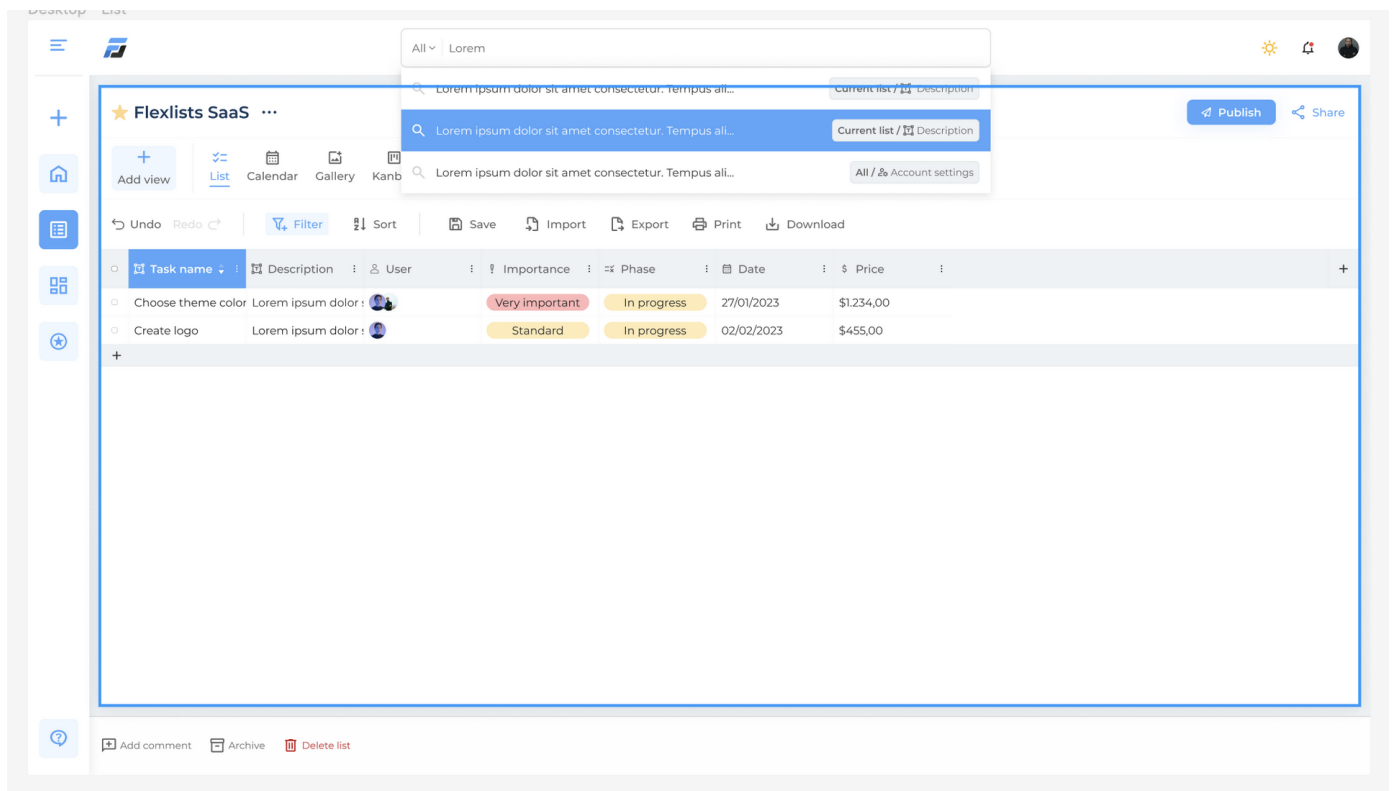
The below need to be transformed in order of appearance, not in another way.

Note: the design is NOT meant to be a complete design!! It is meant to be a collection of components that can be used to implement the final product. While some parts look like what the end result should be, most parts like buttons etc are not as they will be finally - only the design, but text / icon can be different than the design. That's fine.

Table View

We start with the Table View first. It's the most important view.

Many parts (menu etc) are of course reusable so:



ColumnLayout

We have a simple layout system we use for most parts of the layout which is ColumnLayout;

```
<ColumnLayout count={6} columns={columns} rtl={false}/>
```

Where columns is an array:

```
[
  {
    component: <SomeComponent/>,
    width: 1,
    margin: [0,0,0,0],
    padding: [0,0,0,0],
    alignHorizontal: left/right/center,
    alignVertical: top/bottom/center,
    hide: ['xs', 'sm']
  },
  etc
]
```

The width needs explaining: width is the number of columns, so if your ColumnLayout is 800px and you have 8 columns and you then width:1 means every column is 100px. If a component has width:2 , it means it's 200px and if it's 0.5 it means it's 50px. This is already standard behavior in CSS anyway, but just so you know width is NOT absolute pixels or %.

Hide is for responsive obviously; for instance;

xs, extra-small: 0px
sm, small: 600px
md, medium: 900px
lg, large: 1200px
xl, extra-large: 1536px

This is also standard, so no real work needed.

RTL is right to left for, among other, Arabic language versions.

Although a `ColumnLayout` is called `ColumnLayout`, it of course has also rows. If you overflow the column count, it will show on a new row. Example:

```
<ColumnLayout count={6} columns={[ {      component: <SomeComponent1/>,
width: 2,      margin: [0,0,20,0],
  padding: [10,20,10,20],
  alignHorizontal: 'center',
  alignVertical: 'top',
  hide: ['xs', 'sm']
},
{
  component: <SomeComponent2/>,
  width: 4,
  margin: [0,0,20,0],
  padding: [10,20,10,20],
  alignHorizontal: 'center',
  alignVertical: 'middle',
  hide: ['xs', 'sm']
}
// add more column objects here for additional columns in the first row
,
{
  component: <SomeComponent3/>,
  width: 3,
  margin: [0,0,20,0],
  padding: [10,20,10,20],
  alignHorizontal: 'center',
  alignVertical: 'bottom',
  hide: ['xs', 'sm']
},
{
  component: <SomeComponent4/>,
  width: 3,
  margin: [0,0,20,0],
```

```

padding: [10,20,10,20],
alignHorizontal: 'center',
alignVertical: 'bottom',
hide: ['xs', 'sm']
}
// add more column objects here for additional columns in the second row
,
{
  component: <SomeComponent5/>,
  width: 3,
  margin: [0,0,0,0],
  padding: [10,20,10,20],
  alignHorizontal: 'center',
  alignVertical: 'center',
  hide: ['xs', 'sm']
},
{
  component: <SomeComponent6/>,
  width: 3,
  margin: [0,0,0,0],
  padding: [10,20,10,20],
  alignHorizontal: 'center',
  alignVertical: 'center',
  hide: ['xs', 'sm']
}
// add more column objects here for additional columns in the third row
]} rtl={false}/>

```

Menu

First we implement the menus; because these are special, we require them to be uniquely made for us.

LeftMenu

Create a React class LeftMenu.tsx that implements the LeftMenu. It must be possible to drop it into a page without other tags, so:

```
return (<LeftMenu menuItems={xxx} rtl={false}>)
```

Should simply work without further React or HTML tags.

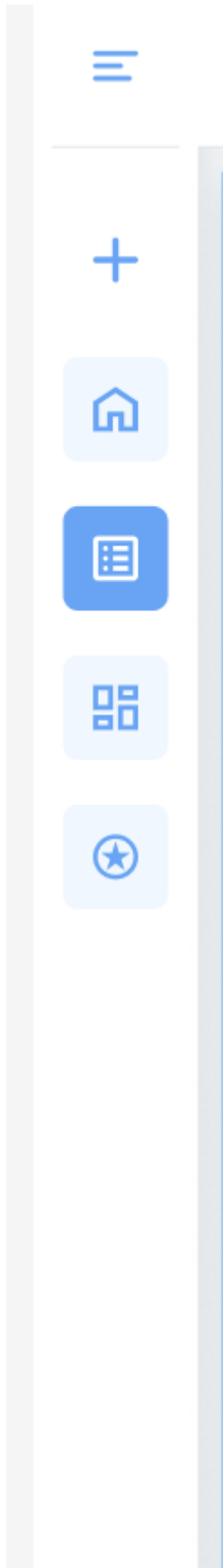
The left menu takes the following properties:

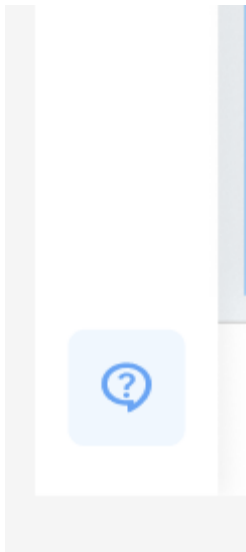
```
[
  { "icon": "someicon", "label": "Add Item", "onClick": ()=>{}, "align":
```

```
"top", "highlightMouseOver": true/false, "highlightClicked": true/false },  
]
```

Menu items can be nested; not in the current design, so we are skipping that feature, but that might be added in the future.

So the menu from the design, would be implemented as;

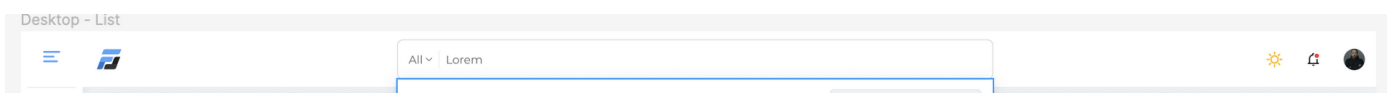




```
[
  { "icon": "plus", "label": "Add Item", "onClick": ()=>{}, "align": "top",
    "highlightMouseOver": true, "highlightClicked": false,
    selected: false },
  { "icon": "home", "label": "Dashboard", "onClick": ()=>{}, "align": "top",
    "highlightMouseOver": true, "highlightClicked": true,
    selected: false },
  { "icon": "table", "label": "List", "onClick": ()=>{}, "align": "top",
    "highlightMouseOver": true, "highlightClicked": true,
    selected: true },
  { "icon": "mondrian", "label": "Builder", "onClick": ()=>{}, "align":
    "top", "highlightMouseOver": true, "highlightClicked": true,
    selected: false },
  { "icon": "star", "label": "Star", "onClick": ()=>{}, "align": "top",
    "highlightMouseOver": true, "highlightClicked": true,
    selected: false },
  { "icon": "help", "label": "Help", "onClick": ()=>{}, "align": "bottom",
    "highlightMouseOver": true, "highlightClicked": true,
    selected: false }
]
```

as input for properties.

TopMenu

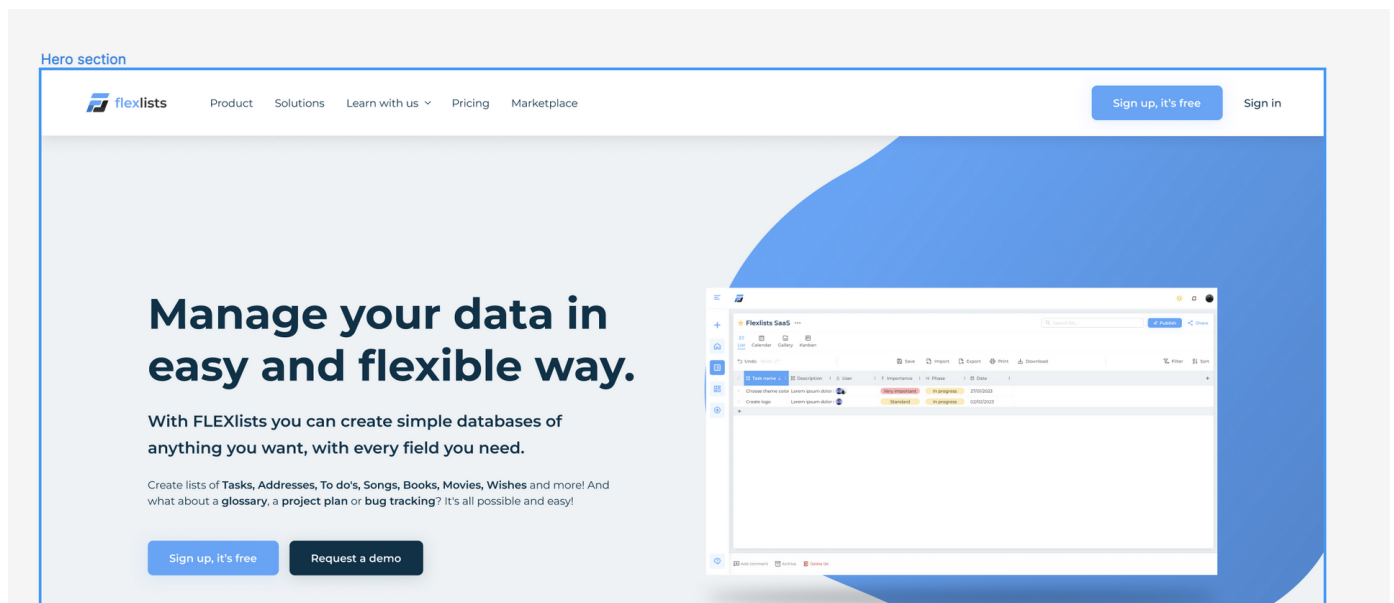


React TopMenu class is more flexible than the LeftMenu one because it's by definition more flexible.

If there is a LeftMenu on the page, there is always a hamburger for expand contract the menu. Or, in other words, the hamburger icon:



does actually not belong to the top menu but looks like it does. It can be that TopMenu is on a page WITHOUT the LeftMenu, in that case, of course, it takes the entire span (see the landing page design as example).



The top menu is flexible, so it is configured like this:

```
[
  {
    component: <ColumnLayout columns={xxx} />,
      alignHorizontal: left/right/center,
      alignVertical: top/bottom/center,
      padding: [],
      maring: [],
      hide: ['xs']
    }
  ]
```

The individual components handle the rest.

So the top menu in the Table design is implemented by putting in a ColumnLayout:

```
[
  {
    component: <ColumnLayout columns=[image with logo] />,
      alignHorizontal: left,
```

```

        alignVertical: center,
        padding: [],
        maring: [],
        hide: []
    },
    {
        component: <ColumnLayout columns=[search bar component] />,
        alignHorizontal: center,
        alignVertical: center,
        padding: [],
        maring: [],
        hide: []
    },
    {
        component: <ColumnLayout columns=[Dark mode button, notifications
button, profile button] />,
        alignHorizontal: right,
        alignVertical: center,
        padding: [],
        maring: [],
        hide: []
    },
]

```

BottomMenu

A BottomMenu is NOT a footer menu; it is simply the same as the left menu but then fixed to the bottom (so it doesn't scroll with the page unlike a footer menu). It's configuration is the same as the LeftMenu component, so check there.

SearchBar

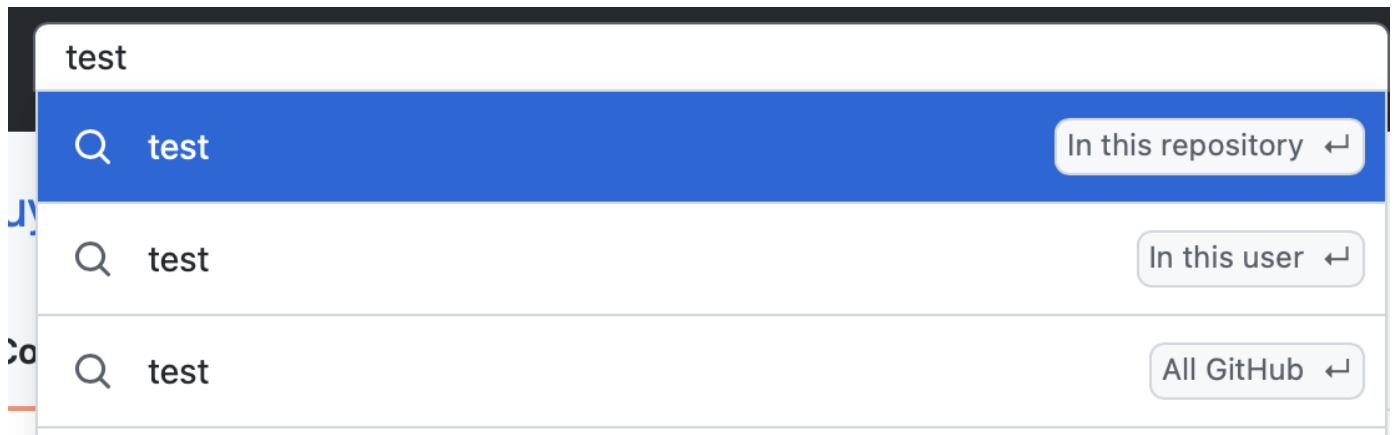
A searchbar is an input with the following configuration:

```

<SearchBar autoComplete?={ (s)=>... } onSearch={ (s, location?)=>{}} locations?
={locations}>

```

Locations are basically where to search, for instance like on github where the user can search in the current repository, or in all repositories of the user or in the entire site:



If no locations are provided, it will just act like a normal search; so there are different options:

- autoComplete provided will request the function onChange with the search bar content and show the completions
 - if not provided it won't show them (duh)
- locations & autoComplete provided, it will show the typed word with the locations and underneath the autoComplete; if the user clicks on another autocompleted word, it will replace it in the search bar and show the location choices for that word

See the below examples:

SearchBar with autoComplete

javascript

Copy code

```
<SearchBar autoComplete={(searchText) => {  
  // logic to get autoComplete suggestions based on the searchText  
  return [  
    "autoComplete 1",  
    "autoComplete 2",  
    "autoComplete 3"  
  ];  
}} onSearch={(searchText, location) => {  
  // logic to perform search based on the searchText and location  
}} />
```

In this example, the `autoComplete` prop is provided with a function that returns an array of autoComplete suggestions based on the searchText entered by the user in the search bar. The `onSearch` prop is also provided with a function that will be called when the user performs a search, passing the searchText and location (if provided) as arguments.

SearchBar with locations and autoComplete

SCSS

 Copy code


```
const locations = [ { label: "Repositories", value: "repositories" },

<SearchBar autoComplete={(searchText) => {
  // logic to get autoComplete suggestions based on the searchText
  return [ "autoComplete 1", "autoComplete 2", "autoComplete 3"
]} onSearch={(searchText, location) => {
  // logic to perform search based on the searchText and location
}} locations={locations} />
```

In this example, the `locations` prop is provided with an array of options for where to search, and the `autoComplete` function is also provided to generate autoComplete suggestions based on the searchText. When the user types in the search bar, the component will show the typed word with the locations underneath, and if the user clicks on another autocompleted word, it will replace it in the search bar and show the location choices for that word.

SearchBar with locations only

SCSS

 Copy code

```
const locations = [ { label: "Repositories", value: "repositories" },

<SearchBar onSearch={(searchText, location) => {
  // logic to perform search based on the searchText and location
}} locations={locations} />
```

In this example, the `locations` prop is provided with an array of options for where to search, but the `autoComplete` prop is not provided. When the user types in the search bar, the component will show the typed word with the locations underneath, and the user can select a location to search in.

SearchBar without locations or autoComplete

javascript

Copy code

```
<SearchBar onSearch={({searchText}) => {  
  // logic to perform search based on the searchText  
}} />
```

In this example, neither the `autoComplete` nor the `locations` prop is provided. The user can simply type in the search bar and perform a search by pressing enter or clicking a search button.

Button

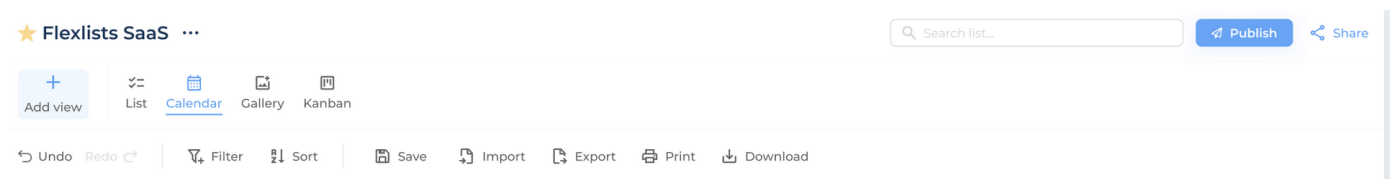
We need a Button component that can take the following input:

```
{  
  "label": 'Save',  
  "onClick": ()=>{},  
  "icon": "save",  
  "iconLocation": left/right/top/bottom ,  
  "highlightMouseOver": true/false, "highlightClicked": true/false  
}
```

So the IconLocation also determines the label location. If the label is empty, it won't show the label, if icon is empty it won't show the icon.

List Header

Ignore the search bar here ; it's some testing to see if we want it on top or not; i think we do , but it doesn't matter; it's easy to do it either way; put it in the top menu, not here;



The list header contains:

- Row 1
 - list icon + list name on the left
 - Publish and Share button on the right
- Row 2
 - Add View (+) with a right padding of some pixels

- The current views in this List, For instance List, Calendar, Kanban, Gallery buttons with their respective icons
- on the left
- Row 3
 - Undo, Redo, separator
 - Filter, Sort, separator
 - Save, Import, etc
 - on the left

So like this (not totally correct but you get the idea); we don't want to specify Icon/Name etc for buttons manually of course, see above;

```
<ColumnLayout count={6} columns={[
  {
    component: (
      <Button
        label="List Name"
        icon="list"
        iconLocation="left"
        highlightMouseOver={true}
        highlightClicked={false}
        onClick={() => {
          // logic to handle button click
        }}
      />
    ),
    width: 3,
    margin: [0, 0, 0, 0],
    padding: [20, 0, 0, 20],
    alignHorizontal: "left",
    alignVertical: "middle",
    hide: []
  },
  {
    component: (
      <Button
        label="Publish"
        icon="publish"
        iconLocation="left"
        highlightMouseOver={true}
        highlightClicked={false}
        onClick={() => {
```

```

        // logic to handle button click
    }}
    />,
    <Button
    label="Share"
    icon="share"
    iconLocation="left"
    highlightMouseOver={true}
    highlightClicked={false}
    onClick={() => {
        // logic to handle button click
    }}
    />
),
width: 3,
margin: [0, 0, 0, 0],
padding: [20, 20, 0, 0],
alignHorizontal: "right",
alignVertical: "middle",
hide: []
}
// add more column objects here for additional columns in the first row
,
{
    component: <Button
        label="Add View"
        icon="Plus"
        iconLocation="top"
        highlightMouseOver={true}
        highlightClicked={false}
        onClick={() => {
            // logic to handle button click
        }}
    />,
    width: 1,
    margin: [0, 0, 0, 0],
    padding: [20, 10, 20, 20],
    alignHorizontal: "left",
    alignVertical: "middle",
    hide: []
},
{
    component: (

```

```

<Button
  label="List"
  icon="Plus"
  iconLocation="top"
  highlightMouseOver={true}
  highlightClicked={false}
  onClick={() => {
    // logic to handle button click
  }}
/>,
<Button
  label="My Calendar"
  icon="Plus"
  iconLocation="top"
  highlightMouseOver={true}
  highlightClicked={false}
  onClick={() => {
    // logic to handle button click
  }}
/>,
<Button
  label="Kanban"
  icon="Plus"
  iconLocation="top"
  highlightMouseOver={true}
  highlightClicked={false}
  onClick={() => {
    // logic to handle button click
  }}
/>
),
width: 5,
margin: [0, 0, 0, 0],
padding: [20, 0, 20, 0],
alignHorizontal: "left",
alignVertical: "middle",
hide: []
}
// add more column objects here for additional columns in the second row
,
{
  component: (
    <Button

```

```

        label="Undo"
        icon="undo"
        iconLocation="left"
        highlightMouseOver={true}
        highlightClicked={false}
        onClick={() => {
            // logic to handle button click
        }}
    />,
    <Button
        label="Redo"
        icon="redo"
        iconLocation="left"
        highlightMouseOver={true}
        highlightClicked={false}
        onClick={() => {
            // logic to handle button click
        }}
    />,
    <Separator/>
),
width: 2,
margin: [0, 0, 0, 0],
padding: [20, 20, 0, 0],
alignHorizontal: "left",
alignVertical: "middle",
hide: []
},
{
    component: (
        <div>
            <Button icon="filter">Filter</Button>
            <Button icon="sort">Sort</Button>
        </div>
    ),
    width: 2,
    margin: [0, 0, 0, 0],
    padding: [20, 20, 0, 0],
    alignHorizontal: "center",
    alignVertical: "middle",
    hide: []
},
{



```

```

component: (
  <div>
    <Button icon="save">Save</Button>
    <Button icon="import">Import</Button>
  </div>
),
width: 2,
margin: [0, 0, 0, 0],
padding: [20, 20, 0, 0],
alignHorizontal: "right",
alignVertical: "middle",
hide: []
}
// add more column objects here for additional columns in the third row
}} rtl={false} />

```

Table

<input type="checkbox"/>	Task name	Description	User	Importance	Phase	Date	Price
<input type="checkbox"/>	Choose theme color	Lorem ipsum dolor		Very important	In progress	27/01/2023	\$1,234,00
<input type="checkbox"/>	Create logo	Lorem ipsum dolor		Standard	In progress	02/02/2023	\$455,00

The last component is the list and that's the most difficult one (not really); it is a basic list which has the following properties:

```

{
  bulkSelect: true/false,
  headers:[
    {
      label: "xxx",
      icon: "icon",
      menuAction: (headerCell) => {},
      onClick: (headerCell) => {}
    },
    etc
  ]
  footers: [
    {
      label: "xxx",
      icon: "icon",
      menuAction: (footerCell) => {},
      onClick: (footerCell) => {}
    }
  ]
}

```



```

],
content: [
  ['hello', 'world'],
  ['some', 'test'],
  etc
],
headerRender: ()=>{},
footerRender: ()=>{},
contentRender: ()=>{},
headerCellRender: (headerCell)=>{},
footerCellRender: (footerCell)=>{},
contentCellRender: (contentCell)=>{}
headerCellOnClick: (headerCell)=>{},
footerCellOnClick: (footerCell)=>{},
contentCellOnClick: (contentCell)=>{},
headerCellMouseOver: (headerCell)=>{},
footerCellMouseOver: (footerCell)=>{}
contentCellMouseOver: (contentCell)=>{}
}

```

The Render functions:






```

headerRender: ()=>{},
footerRender: ()=>{},
contentRender: ()=>{},
headerCellRender: (headerCell)=>{},
footerCellRender: (footerCell)=>{},
contentCellRender: (contentCell)=>{}



```

Must return a react component and the input properties of these must be the current cell or total configuration.

If there are no Render functions defined, it will show the default rendering, which is the standard Table component in MUI minus the images and colors;

<input type="checkbox"/>	Name ↑	Company	Role	Verified	Status	
<input type="checkbox"/>	 Alfonso Dibbert	Hettinger LLC	Front End Developer	Yes	Active	⋮
<input type="checkbox"/>	 Angelica Rippin	Emard Group	UI Designer	Yes	Active	⋮
<input type="checkbox"/>	 Clarence Powlowski	Nienow and Sons	UI/UX Designer	Yes	Banned	⋮
<input type="checkbox"/>	 Elvira Johnston	Ortiz LLC	Front End Developer	Yes	Banned	⋮
<input type="checkbox"/>	 Gene Marks	Adams LLC	UI/UX Designer	Yes	Active	⋮

By using OnRender to return special renderings, we can create this:

<input type="checkbox"/>	Task name ↓	Description	User	Importance	Phase	Date	Price	
<input type="checkbox"/>	Choose theme color	Lorem ipsum dolor :		Very important	In progress	27/01/2023	\$1,234,00	
<input type="checkbox"/>	Create logo	Lorem ipsum dolor :		Standard	In progress	02/02/2023	\$455,00	
+								

The colors and images and icons can be handled by having contentCellRender return a custom component which knows about the current cell content and can handle it.

The + in the footer can be done by rendering a custom footerOnRender and just only returning a + button.

The + in the header can be done by rendering the header fields custom and when you are on headers.length-1 (so the last one), you add a plus align right.

Pagination can be handled in the same way; we have a special pagination way which is easy to implement in this framework; <https://www.awesomescreenshot.com/video/14962515?key=f190ed5e0b9061d0a58d9618ab64f904>