This is documentation for what? Be more specific.

Documen

Let’s call these projects - “playground” is cute, but not very informative and might even be confusing given the popularity of Scratch. Maybe there’s another good term to distinguish the editing environment from the project itself, but maybe it’s not necessary to distinguish the two.

Could this be editable/revisable documentation in github? I think markdown would look even better than this and would be easier to deal with.

Section 1: Drag And Drop PlayGround

Give some overview. What are you going to tell us about? What’s the high level structure?

-reorder function and move function:

Source refers to the column from where the object has been dragged.
Destination refers to the column to which the object has been dropped

If the object is dragged to a destination which is one of either playground columns, only then will any action take place.
The source object can be either from the playground itself or from the columns on the left(the icons on the left side of the screen). If the source is the latter, then we must preserve the ordering of it. However, if the source is the playground, then we will have to reorder the arrangement of the objects in the list.

What does this mean? Why?

In what way? Are you saying you *instantiate* items in the first column but *move* items in the other two? If so, just say so.

-addBlock function:

define block define store

It dispatches an action that adds a new block in the store based on the param typeName.

-NavigationBar Class

This is a stateful class storing the information about the items present in the sidebars and playground.

Already playground is confusing and ambiguous - I thought the playground was the whole interface seen by the user, but it’s separate from sidebars? What’s in a side bar? Again tell us what you are going to tell us about and what’s the high level structure?

It is driven by two important functions:

- onDragEnd:
  This function is triggered once an object has been dropped to some part of the screen.
  It handles cases on how the objects are supposed to float in the UI. It makes use of the reorder and move function defined in ./extra.js
- handleDelete
  This function identifies the list from which the object is removed and updates the state of the instance accordingly. This function is passed on to <WithHeader/> where it is integrated with its onClick event handler.

<DragDropContext /> - Wraps the part of your application you want to have drag and drop enabled for
<Droppable /> - An area that can be dropped into. Contains <Draggable />s
<Draggable /> - What can be dragged around

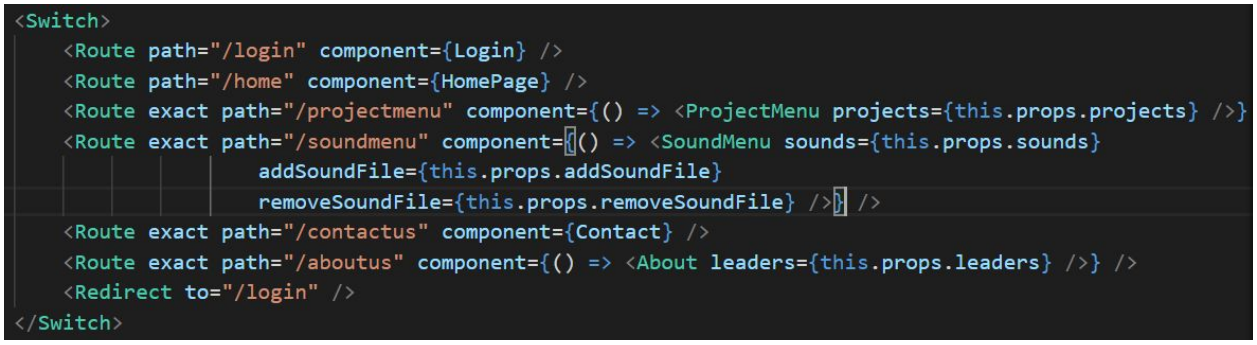
Innerref- Our <Draggable /> and <Droppable /> components both require a HTMLElement to be provided to them. This is done using the innerRef property on the DraggableProvided and DroppableProvided objects.

Section 2: UI

Isn't section 1 also about UI? Not clear how things are organized. Again tell us what you are going to tell us about. Maybe "playground" refers to one part of the UI. Again, what's the structure?

For the UI, we use React style of Single Page Application in which different components are rendered.
The shared folder stores the items that are shared globally across different components in the application which is made possible by the use of redux architecture.

There driver of this code is the MainComponent which calls the other components and makes use of Routing to render different components based on the path. Here we have <Switch> ... </Switch> which decides which of the component is rendered similar to switch in other programming languages (like C++). Once the path matches, the component requested is rendered. Functions can also passed as props here as shown below.



```
<Switch>
    <Route path="/login" component={Login} />
    <Route path="/home" component={HomePage} />
    <Route exact path="/projectmenu" component={() => <ProjectMenu projects={this.props.projects} />} />
    <Route exact path="/soundmenu" component={() => <SoundMenu sounds={this.props.sounds}
                addSoundFile={this.props.addSoundFile}
                removeSoundFile={this.props.removeSoundFile} />} />
    <Route exact path="/contactus" component={Contact} />
    <Route exact path="/aboutus" component={() => <About leaders={this.props.leaders} />} />
    <Redirect to="/login" />
</Switch>
```

Many of the components such as the Header, Footer, AboutUs, Home and Contact Component are pretty self-explanatory as they just include bootstrap styling.

-LoginComponent

It renders a login form and makes uses of the <RegisterForm> Component declared in the file 'LoginComponent.js' if a user has not signedUp Yet. Once the user registers, an action must be dispatched that adds the user credentials to the store which will later be used for verification purposes. The backend of the registration and verification are yet to be implemented.

What does that mean? Password? If we don't know the design or specs, we can't understand the implementation.

-SoundComponent
The user has the option to add sounds to his media files that can be later accessed by the sound components in the project later. The sound files are added by dispatching an action which stores the information of the new media in the store itself.

OK, so now we know Kristin has "secret" documentation. Can that be shared and a reference included here (or integrate all into a larger document)?

/redux folder

To understand the usecase of mapStateToProps, connect, mapDispatchToProps, please refer to this article. Kristin's documentaion on redux explains how redux works. However, if anything seems confusing, please feel free to contact me.

-ActionCreators
Each Action is declared here. Each action has a type and payload. The type helps identify the action whereas the payload is the information that is used to perform the action itself.
-actionTypes
It is a list of all the types of actions
-configureStore
It creates the stores by combining the reducers.