Building the Context Menu System

Looking back at our description of what a “context menu” is, we said that it needs to be absolutely positioned on the screen. We can put together a generic component to help render something at an absolute position.

> **Commit fdd8ba0: Add an AbsolutePosition component**

common/components/AbsolutePosition.jsx

```
import React from "react";
import PropTypes from "prop-types";

const AbsolutePosition = (props) => {
    const {children, nodeRef} = props;
    const style = {
        position: 'absolute',
        top: props.top,
        bottom : props.bottom,
        left: props.left,
        right : props.right,
        width: props.width,
    };

    return (
        <div style={style} className={props.className} ref={nodeRef}>
            {children}
        </div>
    );
}

AbsolutePosition.propTypes = {
    top: PropTypes.number,
    bottom : PropTypes.number,
    left: PropTypes.number,
    width: PropTypes.number,
```

```
    nodeRef : PropTypes.func,
};

export default AbsolutePosition;
```

All we really do here is set a `div`'s style to `position : "absolute"`, apply the provided positions, and insert the children inside the div. The only slightly unusual thing here is that we're taking a prop called `nodeRef`, and passing it down as a callback ref to the `div`. We'll see why that matters in a minute.

Now for the actual context menu behavior. First, we'll add the `react-portal` library to our app:

> **Commit 1602441: Add the React-Portal library**

Then, we'll implement the core of our context menu functionality, very similar to how we built the `ModalManager` component and reducer logic earlier.

> **Commit 5c3a3d9: Implement core context menu handling logic**

features/contextMenus/contextMenuReducer.js

```
import {createReducer} from "common/utils/reducerUtils";

import {
    CONTEXT_MENU_SHOW,
    CONTEXT_MENU_HIDE,
} from "./contextMenuConstants";

const contextMenuInitialState = {
    show : false,
    location : {
        x : null,
        y : null,
    },
    type : null,
    menuArgs : undefined,
}

function showContextMenu(state, payload) {
```

```
    return {
        ...state,

        show : true,
        ...payload
    };
}

function hideContextMenu(state, payload) {
    return {
        ...contextMenuInitialState
    }
};

export default createReducer(contextMenuInitialState, {
    [CONTEXT_MENU_SHOW] : showContextMenu,
    [CONTEXT_MENU_HIDE] : hideContextMenu
});
```

Our `contextMenuReducer` is fairly similar to the first iteration of the modal reducer. I probably could have done almost the same thing, where `null` represents no context menu and a valid object represents actually showing a menu, but wound up implementing this a bit differently in a couple ways. (Not entirely sure why, either, but I did :))

We're going to track a `show` flag that indicates whether we're showing a menu, and `type` and `menuArgs` represent the same concepts as with our modals. We also need to track the location on screen where the menu should be positioned.

features/contextMenus/ContextMenu.jsx

```
import React, {Component} from "react";
import {connect} from "react-redux";

import AbsolutePosition from "common/components/AbsolutePosition";

import {hideContextMenu} from "./contextMenuActions";

const actions = {hideContextMenu};

export class ContextMenu extends Component {
    componentDidMount() {
        document.addEventListener('click', this.handleClickOutside, true);
    }
```

```
    componentWillUnmount() {
        document.removeEventListener('click', this.handleClickOutside, tru
e);
    }

    handleClickOutside = (e) => {
        if (!this.node || !this.node.contains(e.target) ) {
            this.props.hideContextMenu();
        }
    }

    render() {
        const {location} = this.props;

        return (
            <AbsolutePosition
                left={location.x + 2}
                top={location.y}
                className="contextMenu"
                nodeRef={node => this.node = node}
            >
                {this.props.children}
            </AbsolutePosition>
        )
    }
}

export default connect(null, actions)(ContextMenu);
```

Next up we have a generic wrapper component for context menus. This component takes care of listening for clicks outside the menu and calling a close function, as well as using an `<AbsolutePosition>` component to put the menu in the right spot. Note that we offset the `x` coordinate by a couple pixels just to have the menu appear slightly offset from underneath the cursor. Finally, note that we use the `nodeRef` prop for `AbsolutePosition`. That's because we need to do some DOM checks to see if a click on the document is inside or outside the menu. Since the `ContextMenu` component doesn't render any actual HTML itself, it needs to have the `AbsolutePosition` component "forward a ref" on down. This is a useful technique, and Dan Abramov wrote an example of the "forwarded refs" pattern a while back.

features/contextMenus/ContextMenuManager.jsx

```
import React, {Component} from "react";
import {connect} from "react-redux";
import Portal from 'react-portal';

import ContextMenu from "./ContextMenu";

import {selectContextMenu} from "./contextMenuSelectors";

const menuTypes = {
};

export function contextMenuManagerMapState(state) {
    return {
        contextMenu : selectContextMenu(state)
    };
}

export class ContextMenuManager extends Component {
    render() {
        const {contextMenu} = this.props;
        const {show, location, type, menuArgs = {}} = contextMenu;

        let menu = null;

        if(show) {
            let MenuComponent = menuTypes[type];

            if(MenuComponent) {
                menu = (
                    <Portal isOpened={true}>
                        <ContextMenu location={location}>
                            <MenuComponent {...menuArgs} />
                        </ContextMenu>
                    </Portal>
                )
            }
        }

        return menu;
    }
}

export default connect(contextMenuManagerMapState)(ContextMenuManager);
```

Similar to our `ModalManager` component, the `ContextMenuManager` uses the description in Redux to look up the right menu component if appropriate, and renders it. In this case, we also surround the menu component with our `<ContextMenu>` component to put it in the right position and handle clicks outside of it, and surround that with a `<Portal>` to ensure that it floats over the UI.

With those in place, we add the `contextMenuReducer` and the `ContextMenuManager` to our root reducer and the core application layout:

> **Commit e979a27: Add the context menu reducer and component to the app**

And we can now throw together a quick test menu component to verify that this is working (including adding it to the context menu lookup table):

Commit 67908aa: Add an initial test context menu component and hook it up

features/contextMenus/TestContextMenu.jsx

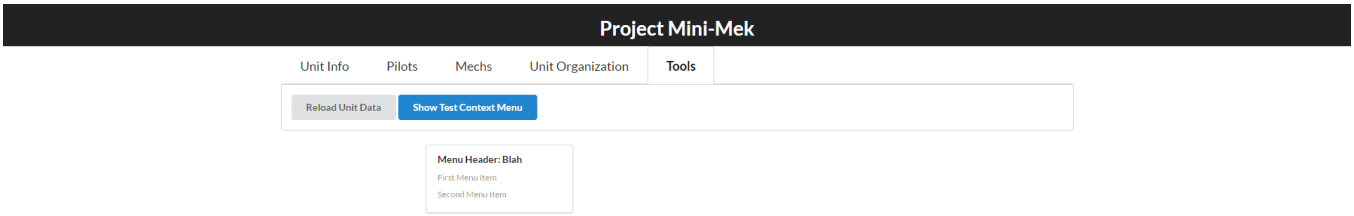
```
import React, { Component } from 'react'
import { Menu } from 'semantic-ui-react'

export default class TestContextMenu extends Component {
    render() {
        return (
            <Menu vertical>
                <Menu.Item>
                    <Menu.Header>Menu Header: {this.props.text} </Menu.Hea
der>

                    <Menu.Menu>
                        <Menu.Item>First Menu Item</Menu.Item>
                        <Menu.Item>Second Menu Item</Menu.Item>
                    </Menu.Menu>
                </Menu.Item>
            </Menu>
        )
    }
```

```
}
```

If we click our “Show Test Context Menu” button, here’s what we should see:



Yay, a menu! And it does nothing useful! Don’t worry, we’ll take care of that next.