Making Modals Stackable

Most of the time, you *probably* only need one modal open at once. But, what if you actually *do* need multiple modals open, stacked on top of each other? Fortunately, now that we've got the basic modal framework in place, this is pretty simple to add. All we really need to do have our reducer store an array of modal descriptions instead of just one, and have the `ModalManager` component render multiple modals in response.

While we're at it, we can update our `TestModal` component to actually make use of the new functionality.

> **Commit f20378c: Implement stackable modals**

features/modals/modalReducer.js

```
-const initialState = null;
+const initialState = [];

export function openModal(state, payload) {
    const {modalType, modalProps} = payload;
-   return {modalType, modalProps};
+   // Always pushing a new modal onto the stack
+   return state.concat({modalType, modalProps});
}

export function closeModal(state, payload) {
-   return null;
+   // Always popping the last modal off the stack
+   const newState = state.slice();
+   newState.pop();
+   return newState;
}
```

features/modals/ModalManager.jsx

features/modals/ModalManager.jsx

```
const mapState = (state) => ({currentModals : state.modals});

export class ModalManager extends Component {
    render() {
        const {currentModals} = this.props;

        const renderedModals = currentModals.map( (modalDescription, inde
x) => {
            const {modalType, modalProps = {}} = modalDescription;
            const ModalComponent = modalComponentLookupTable[modalType];

            return <ModalComponent {...modalProps}  key={modalType + inde
x}/>;
        });

        return <span>{renderedModals}</span>
    }
}
```

Our `modalReducer` cases switch to tracking a stack of modal descriptions in an array, and the `ModalManager` component changes to loop over that array and render an array of modal components.

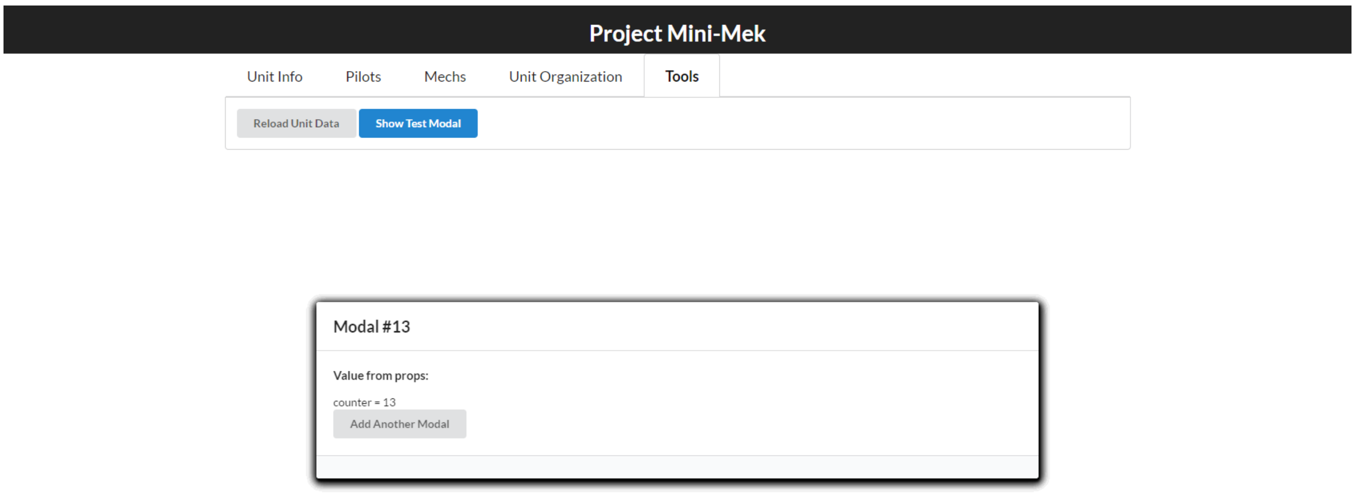
features/modals/TestModal.jsx

```
import {connect} from "react-redux";
import {
    Modal,
+   Button,
} from "semantic-ui-react";

-import {closeModal} from "features/modals/modalActions";
+import {openModal, closeModal} from "features/modals/modalActions";

-const actions = {closeModal};
+const actions = {openModal, closeModal};

export class TestModal extends Component {
+   onNextModalClick = () => {
+       const {counter} = this.props;
+       this.props.openModal("TestModal", {counter : counter + 1});
+   }
```

```
    render() {
+       const {counter, closeModal} = this.props;

        return (
            <Modal
                closeIcon="close"
                open={true}
-               onClose={this.props.closeModal}
+               onClose={closeModal}
            >
-               <Modal.Header>Modal #1</Modal.Header>
+               <Modal.Header>Modal #{counter}</Modal.Header>
                <Modal.Content image>
                    <Modal.Description>
-                       <p>This is a modal dialog.  Pretty neat, huh?</p>
+                       <h4>
+                           Value from props:
+                       </h4>
+                       <div>
+                           counter = {counter}
+                       </div>
+                       <div>
+                           <Button onClick={this.onNextModalClick}>Add An
other Modal</Button>
+                       </div>
                    </Modal.Description>
                </Modal.Content>
                <Modal.Actions>
```

Our `TestModal` gets changed in two ways. First, we'll take the `props.counter` value and display it as part of the header and the descriptive text. Then, we'll add a button and handle clicks on it by dispatching an action to show *another* modal on top of the current one. To illustrate that we're stacking them, we increase the counter value and pass it as a prop for the next modal in the stack.

Let's try it out! If we click the "Show Test Modal" button in the Tools menu, and then click "Add Another Modal" a few times, here's what we get (I temporarily added `dimmer={false}` to `TestModal` to keep the entire page from being blacked out):



Looks neat, and we’ve proven that we can show modals on the screen. Now, let’s move on to building something useful.