Pilot Form UI State

Now that we can edit the basic attributes for our combat unit, it's time to move on to the Pilots panel. We want to add the ability to edit the attributes for our individual Pilot entries. As part of that, it would be nice if we actually could toggle whether we're in "edit mode" or not. For now, let's implement logic to track "editing mode" for pilots, and hold off on actually connecting the inputs until next time.

We already have logic for tracking which pilot is selected. To add to that, we should only be able to start editing if a pilot is selected. If we're editing one pilot, and click to select another, editing mode should be turned off.

Tracking Editing State for the UI

Let's start by adding some logic to track whether we're editing a pilot or not. We'll create a couple new action types (`PILOT_EDIT_START` and `PILOT_EDIT_STOP`), and update our pilots reducer with a new flag and the logic to update it appropriately:

> **Commit ed3c460: Add logic to track if a pilot is being edited**

features/pilots/pilotsReducer.js

```
import {
    PILOT_SELECT,
+   PILOT_EDIT_START,
+   PILOT_EDIT_STOP,
} from "./pilotsConstants";

const initialState = {
    currentPilot : null,
+   isEditing : false,
};
```

```
export function selectPilot(state, payload) {

    const prevSelectedPilot = state.currentPilot;
    const newSelectedPilot = payload.currentPilot;

    const isSamePilot = prevSelectedPilot === newSelectedPilot;

    return {
        ...state,
        // Deselect entirely if it's a second click on the same pilot,
        // otherwise go ahead and select the one that was clicked
        currentPilot : isSamePilot ? null : newSelectedPilot,
+       // Any time we select a different pilot, we stop editing
+       isEditing : false,
    };
}

+export function startEditingPilot(state, payload) {
+   return {
+       ...state,
+       isEditing : true,
+   };
+}

+export function stopEditingPilot(state, payload) {
+   return {
+       ...state,
+       isEditing : false,
+   };
+}

export default createReducer(initialState, {
    [PILOT_SELECT] : selectPilot,
+   [PILOT_EDIT_START] : startEditingPilot,
+   [PILOT_EDIT_STOP] : stopEditingPilot,
});
```

The reducer logic is straightforward. We respond to “start” and “stop” by setting the `isEditing` flag appropriately, and also reset it to false whenever a pilots list entry is clicked.

Adding Edit Mode Toggles

Our last step for this section is adding a pair of “Start / Stop Editing” buttons to the `<PilotDetails>` form, and hooking them up. We also want to add some

conditional logic so that they're only enabled if appropriate.

> **Commit 20f3339: Add "Start/Stop Editing" buttons to PilotDetails**

features/pilots/PilotDetails/PilotDetails.jsx

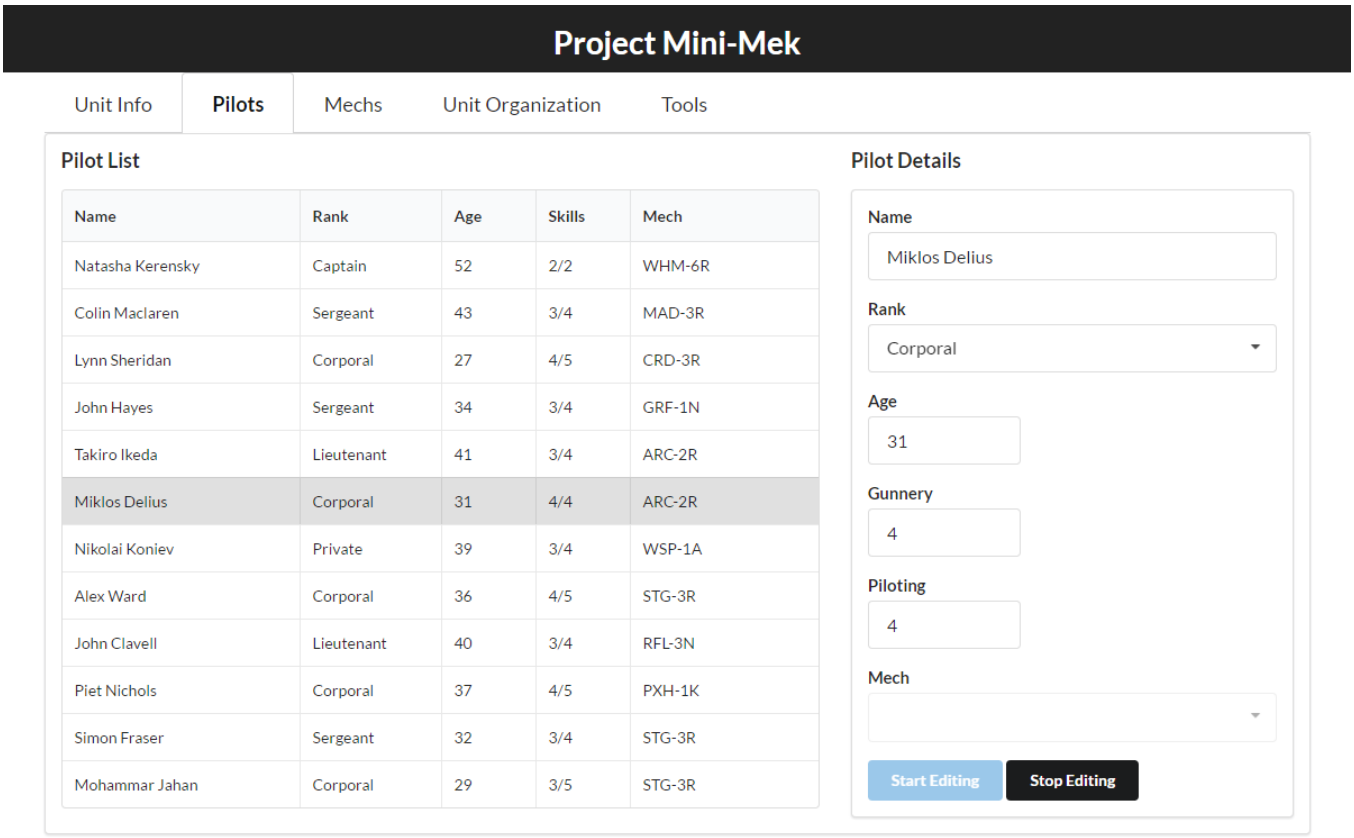
```
-import {selectCurrentPilot} from "./pilotsSelectors";
+import {selectCurrentPilot, selectIsEditingPilot} from "./pilotsSelector
s";

+import {
+   startEditingPilot,
+   stopEditingPilot,
+} from "./pilotsActions";


const mapState = (state) => {
    // Omit Pilot object lookup code

+   const pilotIsSelected = Boolean(currentPilot);
+   const isEditingPilot = selectIsEditingPilot(state);

-   return {pilot}
+   return {pilot, pilotIsSelected, isEditingPilot}
}

+const actions = {
+    startEditingPilot,
+    stopEditingPilot,
+}


-const PilotDetails = ({pilot={}}) =>{
+const PilotDetails = ({pilot={}, pilotIsSelected = false, isEditingPilo
t = false, ...actions }) =>{
// Omit attribute lookups

+    const canStartEditing = pilotIsSelected && !isEditingPilot;
+    const canStopEditing = pilotIsSelected && isEditingPilot;

    return (
        <Form size="large">
            <Form.Field name="name" width={16}>
                <label>Name</label>
                <input
```

```
                    placeholder="Name"
                    value={name}
-                   disabled={true}
+                   disabled={!canStopEditing}
                />
            </Form.Field>
// Omit other fields
+           <Grid.Row width={16}>
+               <Button
+                   primary
+                   disabled={!canStartEditing}
+                   type="button"
+                   onClick={actions.startEditingPilot}
+               >
+                   Start Editing
+               </Button>
+               <Button
+                   secondary
+                   disabled={!canStopEditing}
+                   type="button"
+                   onClick={actions.stopEditingPilot}
+               >
+                   Stop Editing
+               </Button>
+           </Grid.Row>
```

In our `mapState` function, we look at the `currentPilot` flag to determine if a pilot is selected or not, and pass that as a prop. In the component, we look at `isEditing` and `pilotIsSelected`, and derive two new flags to determine if the "Start" and "Stop" buttons should be enabled. We also use those to appropriately enable and disable the inputs.

One other useful note: by default, clicking an HTML `<button>` inside of a `<form>` will auto-submit the form. To avoid that, you have to give the button a `type="button"` attribute. Real pain in the neck, but now you know :)

Let's check out how the form looks now. If we have data loaded, select a pilot, and click "Start Editing", we should now see this:



And that's a good place to wrap up this section. We don't actually have the `<PilotDetails>` form hooked up yet, so those inputs don't do anything useful at the moment. We'll deal with those in the next section.

Here's the current progress with the app:

```
.App-header {
  background-color: #222;
  height: 70px;
  padding: 20px;
  color: white;
}
```