Using Draft Data for Editing Pilots

With those building blocks set, we can now begin updating our `<PilotDetails>` form to work with the new editing logic and data.

Displaying the Copied Pilot Entry

Since we're copying edited items from `entities` to `editingEntities`, our `<PilotDetails>` form will need to be able to switch which slice of state it's reading the pilot data from. We'll add some additional logic to the `mapState` function so that it can determine what slice it should use to load the data. We also need to update the action creators to actually dispatch the `EDIT_ITEM_EXISTING` and `EDIT_ITEM_STOP` actions at the same time as we dispatch the `PILOT_EDIT_START` and `PILOT_EDIT_STOP` actions.

> **Commit db9b7f7: Add ability to display pilot entry from "draft" editingEntities slice**

features/editing/editingSelectors.js

```
import {createSelector} from "reselect";

import orm from "app/orm";

export const selectEditingEntities = state => state.editingEntities;

export const getEditingEntitiesSession = createSelector(
    selectEditingEntities,
    editingEntities => orm.session(editingEntities)
);
```

As with our `entities` slice, we'll create a memoized selector that ensures we only have a single Redux-ORM Session instance per update to the `editingEntities` slice.

features/pilots/pilotsActions.js

```
+import {
+   editExistingItem,
+   stopEditingItem
+} from "features/editing/editingActions";


-export function startEditingPilot() {
-   return {
-       type : PILOT_EDIT_START,
-   };
-}
+export function startEditingPilot(pilotID) {
+   return (dispatch, getState) => {
+       dispatch(editExistingItem("Pilot", pilotID));
+       dispatch({type : PILOT_EDIT_START});
+   }
+}

-export function stopEditingPilot() {
-   return {
-       type : PILOT_EDIT_STOP,
-   };
-}
+export function stopEditingPilot(pilotID) {
+   return (dispatch, getState) => {
+       dispatch({type : PILOT_EDIT_STOP});
+       dispatch(stopEditingItem("Pilot", pilotID));
+   }
+}
```

We need to dispatch two different actions each time the “Start Editing” and “Stop Editing” buttons are clicked. **We’ll change the existing plain action creators into thunks**, pass in the pilot IDs as arguments, and **dispatch both the pilot-related action and the editing-related action in each thunk**.

features/pilots/PilotDetails.jsx

```
import {getEntitiesSession} from "features/entities/entitySelectors";
+import {getEditingEntitiesSession} from "features/editing/editingSelector
s";

// Skip to mapState
```

```
    const currentPilot = selectCurrentPilot(state);


+   const pilotIsSelected = Boolean(currentPilot);
+   const isEditingPilot = selectIsEditingPilot(state);
+
+   if(pilotIsSelected) {
+       const session = isEditingPilot ?
+           getEditingEntitiesSession(state) :
+           getEntitiesSession(state);

        const {Pilot} = session;

        if(Pilot.hasId(currentPilot)) {
            pilot = Pilot.withId(currentPilot).ref;
        }
+    }

export class PilotDetails  extends Component {
+   onStartEditingClicked = () => {
+       const {id} = this.props.pilot;
+       this.props.startEditingPilot(id);
+   }

+   onStopEditingClicked = () => {
+       const {id} = this.props.pilot;
+       this.props.stopEditingPilot(id);
=   }

// Omit rendering code
                    <Button
                        primary
                        disabled={!canStartEditing}
                        type="button"
-                       onClick={actions.startEditingPilot}
+                       onClick={this.onStartEditingClicked}
                    >
                        Start Editing
                    </Button>
                    <Button
                        secondary
                        disabled={!canStopEditing}
                        type="button"
-                       onClick={actions.stopEditingPilot}
+                       onClick={this.onStopEditingClicked}
                    >
```

```
                    Stop Editing
                </Button>
```

We rework the `mapState` function to determine whether we're currently editing a pilot or just displaying it. Based on that, we retrieve either the editing Session or the "current data" Session, and read the Pilot out of there.

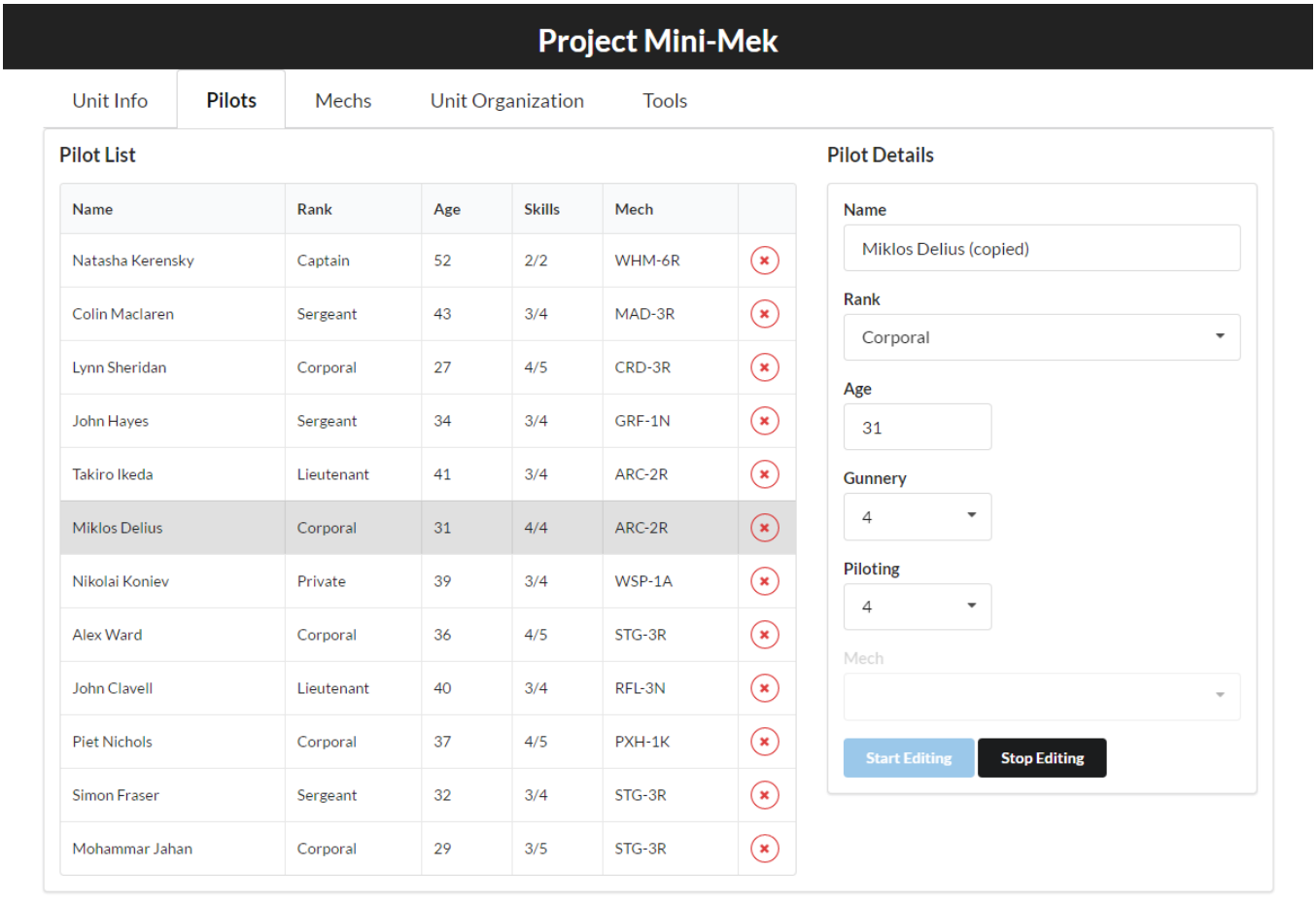
The `<PilotDetails>` component gets a couple new click handlers, which extract the pilot ID from props, and call the appropriate action creator.

features/editing/editingReducer.js

```
    const newItemAttributes = readEntityData(sourceEntities, itemType, ite
mID);
+   if(newItemAttributes.name) {
+       newItemAttributes.name += " (copied)";
+   }
    const creationPayload = {itemType, itemID, newItemAttributes}
```

As a visual indicator that we really are displaying the edited item, we'll temporarily modify our `copyEntity()` function to append the text " (copied)" to the end of the name field.

If we select a pilot and start editing it, we should now see:



Progress! We now have two different versions of the same pilot being displayed: the “current” entry in the list, and the “draft” entry in the form.

Updating the Edited Entry

If you try typing in the “Name” field, you’ll notice some very strange behavior. As soon as you type anything, the list item for the pilot will update as well. That’s because we’re still dispatching `ENTITY_UPDATE`, instead of `EDIT_ITEM_UPDATE`. We need to modify `<PilotDetails>` to actually update the “draft” version. While we’re at it, we’ll remove the little “(copied)” modification we made to the reducer.

> **Commit b90405e: Update pilot editing logic to apply to draft entry**

features/pilots/PilotDetails.jsx

```
-import {updateEntity} from "features/entities/entityActions";
+import {editItemAttributes} from "features/editing/editingActions";

const actions = {
    startEditingPilot,
    stopEditingPilot,
-   updateEntity,
+   editItemAttributes,
}


   onInputChanged = (e) => {
        const newValues = getValueFromEvent(e);
        const {id} = this.props.pilot;

-       this.props.updateEntity("Pilot", id, newValues);
+       this.props.editItemAttributes("Pilot", id, newValues);
    }

    onDropdownChanged = (e, result) => {
        const {name, value} = result;
        const newValues = { [name] : value};
        const {id} = this.props.pilot;
```

```
-       this.props.updateEntity("Pilot", id, newValues);


+       this.props.editItemAttributes("Pilot", id, newValues);
    }
```

Now, if you edit the pilot in the form, only that form should update. The corresponding entry in the list should stay as it was.