Editing Pilot Entries

Now that we have our generic entity update reducer in place, we can implement the ability to edit Pilot entries. We already set up the “start/stop editing” toggle last time, so we just need to hook up the event handlers and dispatch the right actions.

Hooking Up Pilot Inputs

We’ll start with the Pilot’s `name` field:

> **Commit efe8854: Hook up editing of pilot name field**

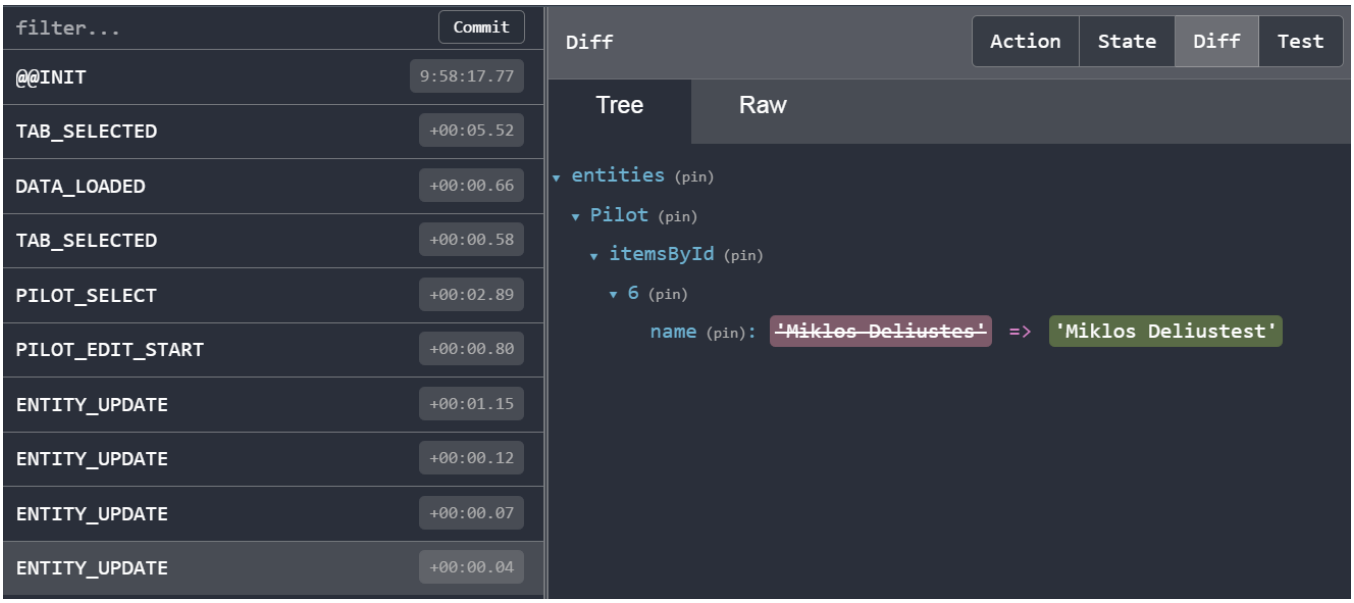
features/pilots/PilotDetails.jsx

```
+import {updateEntity} from "features/entities/entityActions";
+import {getValueFromEvent} from "common/utils/clientUtils";

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

export class PilotDetails  extends Component {
+   onNameChanged = (e) => {
+       const newValues = getValueFromEvent(e);
+       const {id} = this.props.pilot;
+
+       this.props.updateEntity("Pilot", id, newValues);
+   }

// Omit most rendering code

                <Form.Field
```

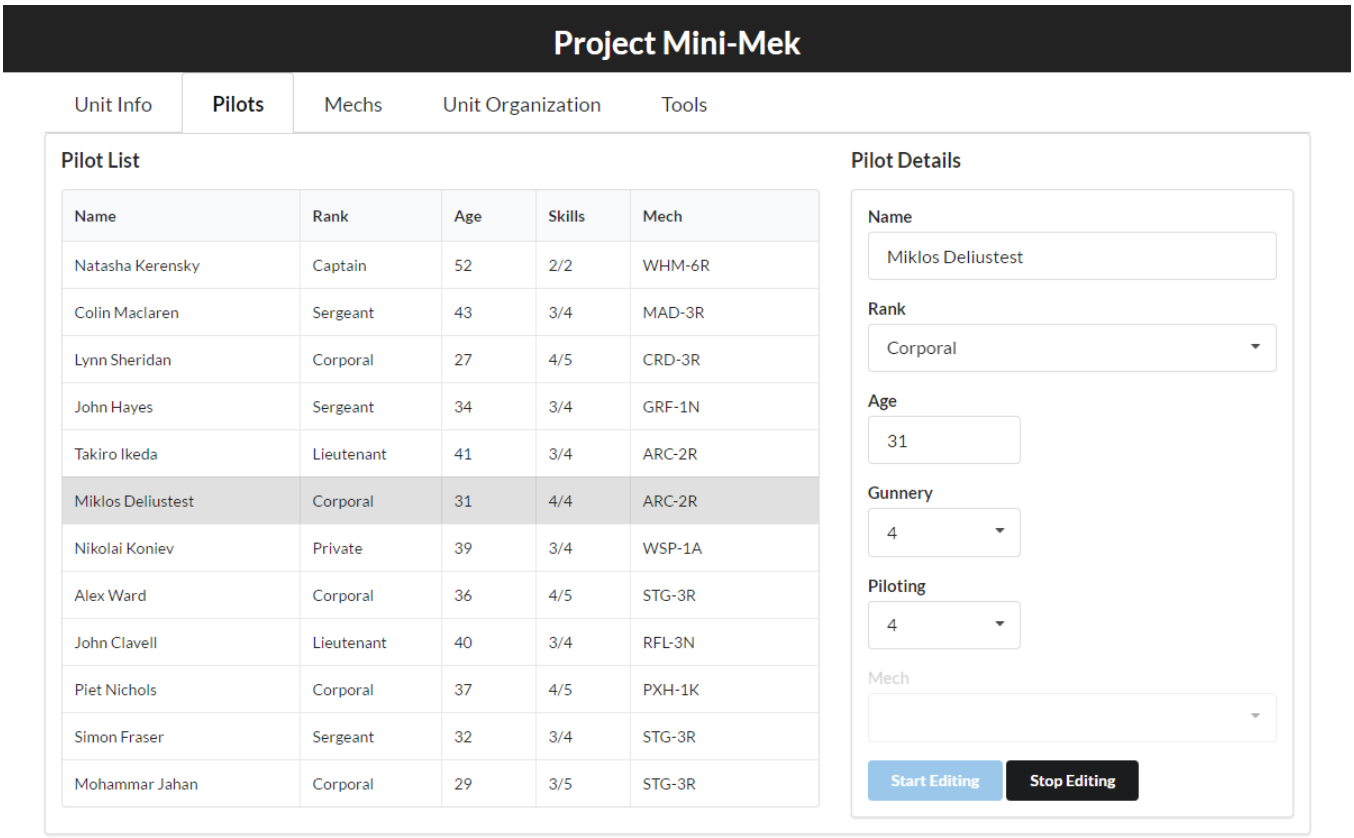
```
                    name="name"
                    label="Name"
                    width={16}
                    placeholder="Name"
                    value={name}
                    disabled={!canStopEditing}
+                   onChange={this.onNameChanged}
                    control="input"
                />
```

We import the `updateEntity` action creator we just wrote, and add it to the actions that will be bound up to auto-dispatch when called. We then add an `onNameChanged` handler, extract the new values from the change event, and dispatch `updateEntity()` by passing in the type of item (`"Pilot"`) and the pilot's ID.

Let's try this out by editing one of the pilots. If we select the pilot named "Miklos Delius", click "Start Editing", and type "test" at the end of his name, we should see some actions dispatched:



And there we go! The dispatched action reached our entities feature reducer, and the `updateEntity()` reducer applied the updates to the right data object in the state. If we look at the screen, we should now see:



Because both the form and the list item are displaying the values from the same item in state, both of them have been updated. It's important to note that **we are directly editing the values for this Pilot entry in our state**. Much of the time, that behavior is *not* something we want. It's very likely that some parts of the application would still need to display the original data, at the same time that we are making edits to an item. **We'll look at one way to handle the "draft/editing data" concept in the next section.**

Next up is the "Rank" dropdown:

> **Commit 7f74bc3: Hook up Pilot "rank" dropdown**

features/pilots/PilotDetails.jsx

```
+   onRankChanged = (e, result) => {
+       const newValues = {rank : result.value};
+       const {id} = this.props.pilot;
+
+       this.props.updateEntity("Pilot", id, newValues);
+   }
```

```
// Omit rendering code


                <Form.Field
                    name="rank"
                    label="Rank"
                    width={16}
                    control={Dropdown}
                    fluid
                    selection
                    options={RANKS}
                    value={rank}
+                   onChange={this.onRankChanged}
                    disabled={!canStopEditing}
                />
```

Easy enough, and now we can promote Corporal Delius to be a Sergeant instead.

Improving Input Handling

We've got three more fields that still need to be hooked up (we'll leave the "Mech" field alone for now). The "Age" field is another text input, and the "Gunnery" and "Piloting" fields are dropdowns. We *could* write three more individual change handlers for those three fields, but looking at the two we have already, things are pretty simple. In fact, `onNameChanged` doesn't actually refer to "name" anywhere specifically, and `onRankChanged` just references "rank" once as a key. We could turn those into a generic "text input" handler and a generic "SUI-React Dropdown" handler, and reuse them for all five inputs.

We've got also one last improvement to make. The "Name" and "Age" fields are currently unbuffered, and dispatching an `ENTITY_UPDATE` action every time we type a key. We can use our `<FormEditWrapper>` component to buffer both of those inputs:

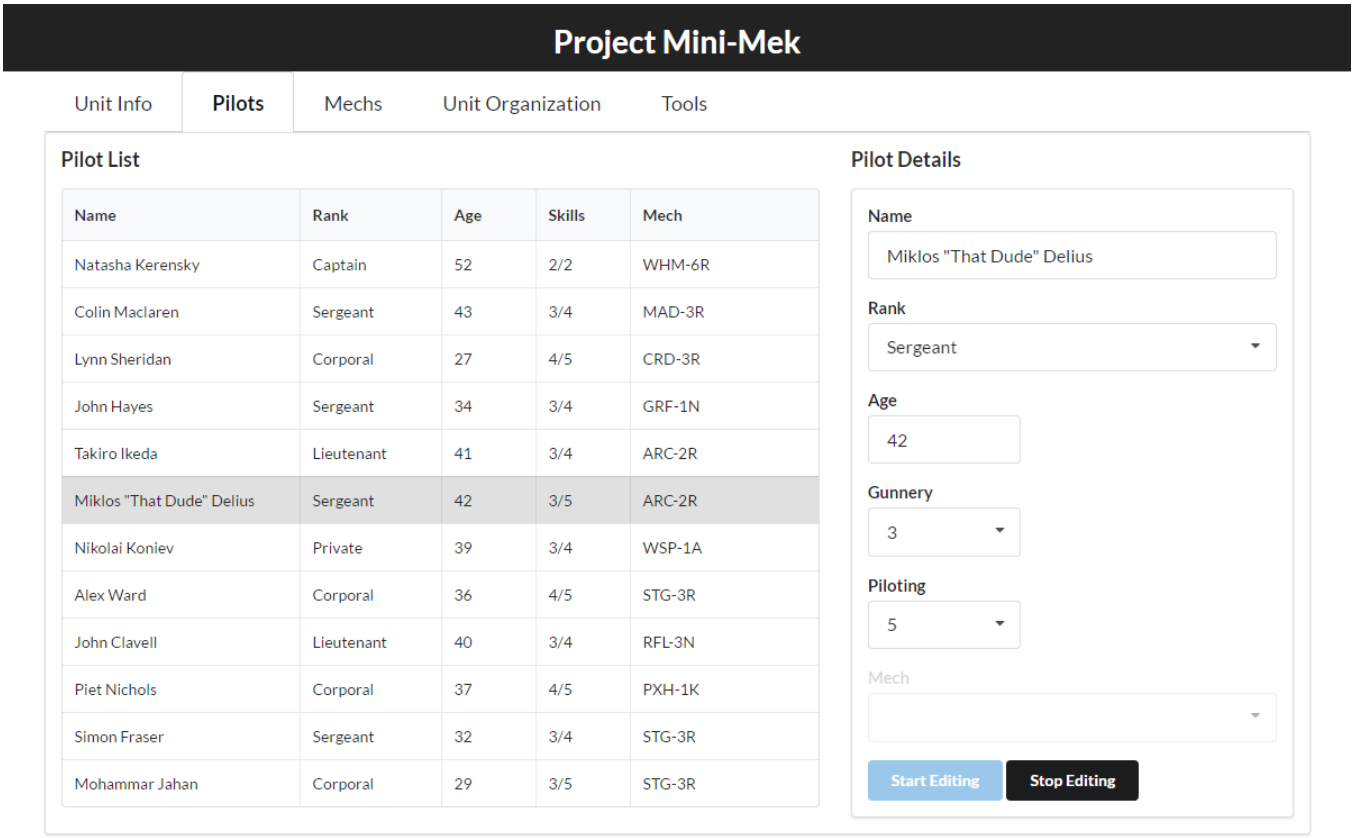
> **Commit b5f54fc: Use FormEditWrapper to buffer changes from Pilot inputs**

features/pilots/PilotDetails.jsx

```
+import FormEditWrapper from "common/components/FormEditWrapper";

// omit other imports

export class PilotDetails  extends Component {
-   onNameChanged = (e) => {
+   onInputChanged = (e) => {
        const newValues = getValueFromEvent(e);
        const {id} = this.props.pilot;

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

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

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


// Omit other component and rendering code

+               <FormEditWrapper
+                   singleValue={true}
+                   value={ {name} }
+                   onChange={this.onInputChanged}
+                   passIsEditing={false}
+               >
                    <Form.Field
                        name="name"
                        label="Name"
                        width={16}
                        placeholder="Name"
-                       value={name}
                        disabled={!canStopEditing}
-                       onChange={this.onInputChanged}
                        control="input"
                    />
+               </FormEditWrapper>
```

```
+                <FormEditWrapper
+                    singleValue={true}
+                    value={ {age} }
+                    onChange={this.onInputChanged}
+                    passIsEditing={false}
+                >
                     <Form.Field
                         name="age"
                         width={6}
                         label="Age"
                         placeholder="Age"
                         control="input"
-                    value={age}
-                    onChange={this.onInputChanged}
                         disabled={!canStopEditing}
                     />
+                </FormEditWrapper>
```

We then pass the appropriate change handler to each input field, and bam! All the inputs should now be editable:



Now, if we edit the name, we should only see a single `ENTITY_UPDATE` action get dispatched.