Adding Entity CRUD Reducers

In Form Change Handling, Data Editing, and Feature Reducers, we used a custom FormEditWrapper component to buffer form change events, wrote custom reducer logic for the "editing" feature, and added basic editing for Pilot entries. In this section, we'll **add the ability to delete Pilots, use generic entity reducer logic to implement "draft data" handling for forms, and implement form resetting and cancelation.**.

Thus far, we can load a list of Pilots from our fake API, display them in the UI, and edit the details of an individual pilot. That covers two of the four capabilities in the term "CRUD" ("Create", "Retrieve", "Update", and "Delete"). Let's go ahead and **add more reducer CRUD logic for other cases**.

Creating a Redux-ORM Session Selector

We'll do a bit of cleanup first, which will also give us a minor performance improvement. As mentioned in the "Performance" section, **many of our connected components create Redux-ORM Session instances in their `mapState` functions**. Creating these Session instances probably isn't *too* expensive, but there really isn't a good reason to keep creating many Session instances every time the store updates and then throw them all away. This is especially true because a Session instance wraps up a specific state "tables" object, and since all of our components will be doing reads and not writes, they could easily all do reads from the same Session instance with no problems.

In addition, our `mapState` functions are currently accessing `state.entities` to pass that slice to the schema. Explicitly accessing slices of state isn't *wrong*, but **it's good practice to use selector functions to encapsulate those lookups**.

We can fix both of these issues at the same time, by **creating a memoized selector function that will always return the same Session instance for a**

given version of the `entities` state slice. (As a reminder, "memoization"

means caching inputs and outputs, so that if you see the same inputs, you can skip the work and return the previously calculated output.) The Reselect library provides a function called `createSelector` that will create the memoized selectors we need. That way, we only create one Session instance per update to our Model data. Again, our application is small enough that we probably wouldn't even see a difference in performance benchmarks, but it's a useful step in the right direction.

> **Commit ec77c77: Use a selector to have only a single Session instance per state update**

features/entities/entitySelectors.js

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

import orm from "app/orm";

export const selectEntities = state => state.entities;

export const getEntitiesSession = createSelector(
    selectEntities,
    entities => orm.session(entities)
);
```

The selector creation is simple. We create an "input" selector that returns the `state.entities` slice, and pass that as the first argument to the `createSelector` function from Reselect. The "output" selector takes the returned entities slice, and calls `schema.from(entities)` to create the Session instance. **That output selector will only run when the `entities` object changes**.

From there, we update all the connected components to use that selector:

features/pilots/PilotDetails/PilotDetails.jsx

```
import {connect} from "react-redux";
import {Form, Dropdown, Grid, Button} from "semantic-ui-react";
```

```
-import orm from "app/orm";
+import {getEntitiesSession} from "features/entities/entitySelectors";


const mapState = (state) => {
    let pilot;

    const currentPilot = selectCurrentPilot(state);

-   const session = orm.session(state.entities);
+   const session = getEntitiesSession(state);
    const {Pilot} = session;
```

After updating all the components to use `getEntitiesSession()`, we should only be creating a single Session instance per entities update.

Adding Entity Creation and Deletion Reducers

We previously created a "feature reducer" to handle generic actions that apply to our `entities` slice. At the time, we only created a single case reducer, which responds to the `ENTITY_UPDATE` action. That allowed us to make updates to the Pilot entry that's being edited.

We're going to add two more case reducers to that section, one for creating new entities and one for deleting them. We need the deletion reducer now so that we can delete our Pilot entries, and the creation reducer will come into play a bit later.

> **Commit 779dbdc: Add generic entity creation and deletion handling**

features/entities/entityReducer.js

```
-import {ENTITY_UPDATE} from "./entityConstants";
+import {
+   ENTITY_UPDATE,
+   ENTITY_DELETE,
+   ENTITY_CREATE,
+} from "./entityConstants";

export function updateEntity(state, payload) {
    const {itemType, itemID, newItemAttributes} = payload;

    const session = orm.session(state);
```

```
    const ModelClass = session[itemType];

    let newState = state;

    if(ModelClass.hasId(itemID)) {
        const modelInstance = ModelClass.withId(itemID);

        modelInstance.update(newItemAttributes);

        newState = session.state;
    }

    return newState;
}

+export function deleteEntity(state, payload) {
+   const {itemID, itemType} = payload;
+
+   const session = orm.session(state);
+   const ModelClass = session[itemType];
+
+   let newState = state;
+
+   if(ModelClass.hasId(itemID)) {
+       const modelInstance = ModelClass.withId(itemID);
+
+       modelInstance.delete();
+
+       newState = session.state;
+   }
+
+   return newState;
+}
+
+export function createEntity(state, payload) {
+   const {itemType, newItemAttributes} = payload;
+
+   const session = orm.session(state);
+   const ModelClass = session[itemType];
+
+   ModelClass.parse(newItemAttributes);
+
+   return session.state;
+}
+
```

```
const entityHandlers = {
    [ENTITY_UPDATE] : updateEntity,

+   [ENTITY_CREATE] : createEntity,
+   [ENTITY_DELETE] : deleteEntity,
};

const entityCrudFeatureReducer = createConditionalSliceReducer("entitie
s", entityHandlers);
```

All of these reducers follow the same basic pattern: create a Redux-ORM session from the current state, look up a session-bound Model class by name, apply the updates, and return the updated state. (Yes, it would probably be possible to write some kind of wrapper function that de-duplicates some of the common behavior in these functions, but they're short enough it's not worth the effort.)

Note that the `createEntity()` reducer expects that each Model class will have a static `parse()` method. Per the Loading Data section, that's not something that's built in to Redux-ORM, but rather a convention that we're following by writing those functions in all our Model classes.