Connecting the Unit Info Tab

It's time to start hooking up some of our UI to the store. We'll start with the simplest part: the Unit Info tab. First, a bit of background info - it would be helpful to know what a "unit" is.

A Battletech "unit" is a military group, which could be part of the official armed forces of one of the various "star nations" in the Battletech universe, or an independent combat group like a mercenary unit. The five Great Houses in the Battletech universe all maintain vast armies, while hundreds of mercenary units large and small try to make a living through combat. Some examples of units from the game universe include:

- Great House army units:
  - 24th Dieron Regulars from the Draconis Combine
  - 2nd Free Worlds Guards from the Free Worlds League
  - 11th Avalon Hussars from the Federated Suns
- Mercenaries:
  - Wolf's Dragoons
  - 21st Centauri Lancers
  - Hansen's Roughriders

For now, we're just going to track two attributes for a unit: their name, and what larger group they are affiliated with.

We'll start by creating a simple reducer that stores the "Name" and "Affiliation" values, add that to our root reducer, and connect the `<UnitInfo>` component to use the data:

> **Commit 0c93284: Add initial unit info reducer and connection**

features/unitInfo/unitInfoReducer.js

features/unitInfo/unitInfoReducer.js

```
import {createReducer} from "common/utils/reducerUtils";

const initialState = {
    name : "Black Widow Company",
    affiliation : "wd",
};

export default createReducer(initialState, {
});
```

features/unitInfo/unitInfoSelectors.js

```
export const selectUnitInfo = state => state.unitInfo;
```

app/reducers/rootReducer.js

```
import {combineReducers} from "redux";

import tabReducer from "features/tabs/tabsReducer";
import unitInfoReducer from "features/unitInfo/unitInfoReducer";

const rootReducer = combineReducers({
    unitInfo : unitInfoReducer,
    tabs : tabReducer,
});

export default rootReducer;
```

features/unitInfo/UnitInfo.jsx

```
const mapState = (state) => ({
    unitInfo : selectUnitInfo(state),
});

const UnitInfo = ({unitInfo = {}}) => {
    const {name, affiliation} = unitInfo;

    return (
        <Segment attached="bottom">
            <Form size="large">
                <Form.Field name="name" width={6} >
                    <label>Unit Name</label>
```

```
                    <input placeholder="Name" value={name}/>
                </Form.Field>

                <Form.Field name="affiliation" width={6}>
                    <label>Affiliation</label>
                    <Dropdown
                        selection
                        options={FACTIONS}
                        value={affiliation}
                    />
                </Form.Field>
            </Form>
        </Segment>
    );
};

export default connect(mapState)(UnitInfo);
```

Now that the `<UnitInfo>` component is being driven by data from the store, we can do something with that sample data from the mock API. Let's update the unit info reducer to respond to the `DATA_LOADED` action and see what happens:

> **Commit 97bed75: Load unit details from sample data**

features/unitInfo/unitInfoReducer.js

```
import {createReducer} from "common/utils/reducerUtils";

import {DATA_LOADED} from "features/tools/toolConstants";

const initialState = {
    name : "N/A",
    affiliation : "",
};

function dataLoaded(state, payload) {
    const {unit} = payload;

    return unit;
}

export default createReducer(initialState, {
    [DATA_LOADED] : dataLoaded,
```

```
    [DATA_LOADED] : dataLoaded,
});
```

The reducer's very simple for now. We start with some empty data in our initial state, listen for `DATA_LOADED`, and return the `unit` data from the action payload.

If we load the page, we should see "Name: N/A". If we then click the "Reload Unit Data" button in the Tools tab, we should see it change to "Name: Black Widow Company". No point in showing a screenshot here, because the UI should still look exactly the same as it has up until now.