Defining Models with Relations

Redux-ORM allows you to define relations between various model types, using standard database concepts. This is done by adding a `fields` entry on a Model class type itself, and using the relation operators provided by Redux-ORM. Also, as described previously, when a Model instance is created Redux-ORM will generate getter properties for all fields in the actual data object, as well as all of the relational fields.

With those ideas in mind, we're almost ready to define our next couple Model types, but first we need to review a bit more information about the ideas we're trying to represent. **In Battletech, there are many different Battlemech designs**. Each design has different statistics: weight, speed, armor, weapons, and so on. There may also be different variations on the same design, which would share the same basic characteristics (usually weight and speed), but maybe have some differences in weapons and armor. **There can be many different individual mechs of the same design**. Here's some examples of different Battlemech design variants to give you the idea:

- **Stinger STG-3R**: 20 tons; 6/9/6 movement points (walk/run/jump); 48 armor points; 1 Medium Laser and 2 Machine Guns
- **Stinger STG-3G**: 20 tons; 6/9/6 movement points; 69 armor points; 2 Medium Lasers
- **Warhammer WHM-6R**: 70 tons; 4/6 movement points; 160 armor points; 2 PPCs, 2 Medium Lasers, 2 Small Lasers, 1 SRM-6 launcher
- **Warhammer WHM-6D**: 70 tons; 4/6 movement points; 217 armor points; 2 PPCs, 2 Medium Lasers, 2 Small Lasers

As a real-life comparison, the F-15 Eagle has several variants: the F-15C is made for air-to-air combat, the F-15D is a training version, the F-15E is intended for ground attacks, and hundreds of individual F-15s have been manufactured.

For our data modeling, we're going to create two more Model classes. We're going to need to store information on different Battlemech designs, and we also need to track individual mechs. Rather than copy all the attributes of a design into each individual Mech entry, we can just store the design once, and add a relation between the individual Mech and its design entry. Meanwhile, since Pilots are going to be assigned to Mechs, we would also want to be able to relate Pilots and Mechs to each other.

Based on that, **we're going to create separate models for MechDesigns and Mechs. The `Mech` class will use "foreign key" relations to point to a `MechDesign` instance and a `Pilot` instance**. For now, we'll also add an FK relation from `Pilot` to `Mech` so that we can look up the `Mech` instance that a given Pilot is assigned to. We'll also go ahead and create `parse()` methods on them so that we can load them from data. Finally, we'll add the `Mech` and `MechDesign` classes to our `ORM` instance, the same way we did with `Pilot`.

Adding Model Relations

First up, we need to add our new Model classes.

Commit ef869ef: Add Mech and MechDesign model classes

features/mechs/MechDesign.js

```
import {Model, attr} from "redux-orm";

export default class MechDesign extends Model {
    static modelName = "MechDesign";

    static fields = {
        id : attr(),
        name : attr(),
        weight : attr(),
    }

    static parse(designData) {
        return this.create(designData);
    }
}
```

features/mechs/Mech.js

features/mechs/Mech.js

```
import {Model, fk, attr} from "redux-orm";

export default class Mech extends Model {
    static modelName = "Mech";

    static fields = {
        id : attr(),
        type : fk("MechDesign"),
        pilot : fk("Pilot"),
    };

    static parse(mechData) {
        return this.create(mechData);
    }
}
```

After making those changes, we need to make use of the new Model classes. First, we need to add entries for MechDesigns and Mechs to our sample data. Second, we need to modify our `entitiesReducer` to load in the new data entries for mechs and designs:

> **Commit dca21ea: Load Mechs and MechDesigns from sample data**

data/sampleData.js

```
+designs : [
+   {
+       id : "WHM-6R",
+       name : "Warhammer",
+       weight : 70,
+   },
+   // additional entries here
+],
+mechs : [
+   {
+       id : 1,
+       type : "WHM-6R",
+       pilot : 1,
+   },
+   // additional entries here
+]
```

app/reducers/entitiesReducer.js

```
export function loadData(state, payload) {
    // Create a Redux-ORM session from our entities "database tables" obje
ct
    const session = orm.session(state);
    // Get a reference to the correct version of model classes for this Se
ssion
    const {Pilot, MechDesign, Mech} = session;

    const {pilots, designs, mechs} = payload;

    // Insert the data entries into the Session
    pilots.forEach(pilot => Pilot.parse(pilot));
    designs.forEach(design => MechDesign.parse(design));
    mechs.forEach(mech => Mech.parse(mech));

    // return a new version of the entities state object with the inserte
d entries
    return session.state;
}
```

Now that we have the Mech and MechDesign data available, we need to modify the `Pilot` model class so that it refers to an individual mech by ID as a foreign key, rather than the ID of a mech type. We also need to update the sample data to match that change.

The final change we need to make here is how we prepare the data in the `mapState` function for `<Pilots>`. Previously, we just returned the plain JS objects directly. Now, though, we need to follow the relations from Pilot to Mech to MechDesign in order to display what type of mech this Pilot uses. The revised `mapState` function now looks like this:

> **Commit 6334f25: Use Pilot->Mech relations to look up Mech instances from Pilots**

data/sampleData.js

```
        {
            id : 1,
            name : "Natasha Kerensky",

            rank : "Captain",
            gunnery : 2,
            piloting : 2,
            age : 52,
-           mechType : "WHM-6R",
+           mech : 1,
        },
```

features/pilots/Pilot.js

```
-import {Model, attr} from "redux-orm";
+import {Model, attr, fk} from "redux-orm";

export default class Pilot extends Model {
    static modelName = "Pilot";

    static fields = {
        id : attr(),
        name : attr(),
        rank : attr(),
        gunnery : attr(),
        piloting : attr(),
        age : attr(),
-       mechType : attr()
+       mech : fk("Mech")
    }
```

features/pilots/Pilots.jsx

```
const mapState = (state) => {
    // Create a Redux-ORM Session from our "entities" slice, which
    // contains the "tables" for each model type
    const session =  orm.session(state.entities);

    // Retrieve the model class that we need.  Each Session
    // specifically "binds" model classes to itself, so that
    // updates to model instances are applied to that session.
    // These "bound classes" are available as fields in the sesssion.
    const {Pilot} = session;

    // Query the session for all Pilot instances.
    // The QuerySet that is returned from all() can be used to
    // retrieve instances of the Pilot class, or retrieve the
    // plain JS objects that are actually in the store.


    // The toModelArray() method will give us an array of the
    // Model class instances objects for each item in the QuerySet.
    const pilots = Pilot.all().toModelArray().map(pilotModel => {
        // Access the underlying plain JS object using the "ref" field,
        // and make a shallow copy of it
        const pilot = {
            ...pilotModel.ref
        };

        // We want to look up pilotModel.mech.mechType.  Just in case the
        // relational fields are null, we'll do a couple safety checks a
s we go.

        // Look up the associated Mech instance using the foreign-key
        // field that we defined in the Pilot Model class
        const {mech} = pilotModel;

        // If there actually is an associated mech, include the
        // mech type's ID as a field in the data passed to the component
        if(mech && mech.type) {
            pilot.mechType = mech.type.id;
        }

        return pilot;
    });

    // Now that we have an array of all pilot objects, return it as a prop
    return {pilots};
}
```

Displaying a List of Mechs

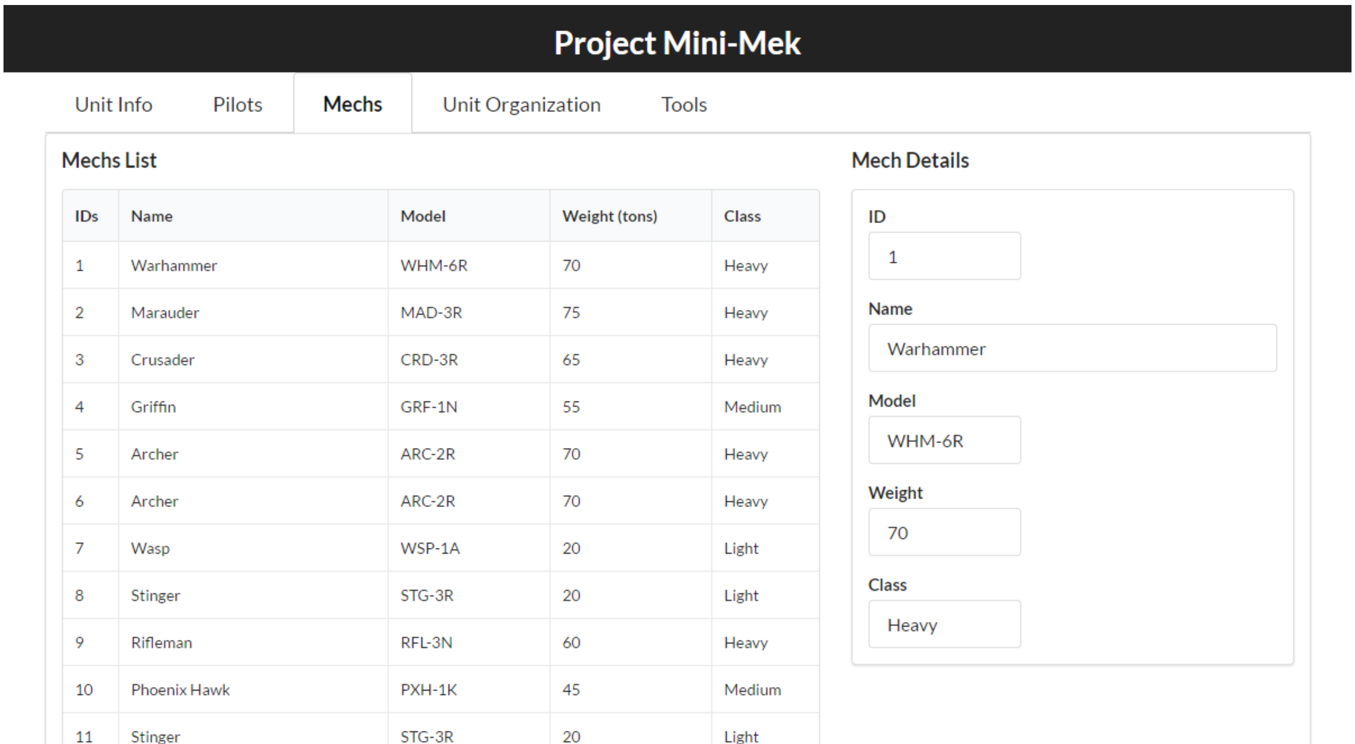
We can now connect our `<Mechs>` component the same way we did `<Pilots>`. This is pretty straightforward, and the only really interesting part is the `mapState` method:

> **Commit 02dcc9e: Connect the Mechs component to display a list of mechs from the store**

features/mechs/Mechs.jsx

```
// omit imports

const mapState = (state) => {
    const session = orm.session(state.entities);
    const {Mech} = session;

    const mechs = Mech.all().toModelArray().map(mechModel => {
        const mech = {
            // Copy the data from the plain JS object
            ...mechModel.ref,
            // Provide a default empty object for the relation
            mechType : {},
        };

        if(mechModel.type) {
            // Replace the default object with a copy of the relation's da
ta
            mech.mechType = {...mechModel.type.ref};
        }

        return mech;
    });

    return {mechs}
}

// omit component
```

And that gives us this update to our UI:



Mechs List

IDs	Name	Model	Weight (tons)	Class
1	Warhammer	WHM-6R	70	Heavy
2	Marauder	MAD-3R	75	Heavy
3	Crusader	CRD-3R	65	Heavy
4	Griffin	GRF-1N	55	Medium
5	Archer	ARC-2R	70	Heavy
6	Archer	ARC-2R	70	Heavy
7	Wasp	WSP-1A	20	Light
8	Stinger	STG-3R	20	Light
9	Rifleman	RFL-3N	60	Heavy
10	Phoenix Hawk	PXH-1K	45	Medium
11	Stinger	STG-3R	20	Light

Mech Details

ID: 1
Name: Warhammer
Model: WHM-6R
Weight: 70
Class: Heavy

12	Stinger	STG-3R	20	Light