Identifying the Relationships

As we develop a better understanding of the different components of our app, the design of our state object improves.

In the last lesson, we designed a basic implementation of our app's state:

```
const state = {
  user: [
    {
      contact1: 'Alex',
      messages: [
        'msg1',
        'msg2',
        'msg3'
      ]
    },
    {
      contact2: 'john',
      messages: [
        'msg1',
        'msg2',
        'msg3'
      ]
    }
  ]
}
```

This is a pretty good representation of our data. It seems like it shows the relationship between each entity, but in terms of the state of your frontend application, this is a bad idea. Bad is a strong word. Let's just say, there's a better way to do this.

Here's how I see it.

If you had to manage a football team, a good plan will be to pick out the best scorers in the team, and put them in the front to get you goals.

You can argue that good players can score from wherever - yes. I bet they'll be more effective when they are well positioned in front of the opposition's goal post.

The same goes for the state object.

Pick out the front runners within the state object, and place them in "front".

When I say "front runners," I mean the fields of the state object you'll be performing more CRUD actions on. The parts of the state you'll be Creating, Reading, Updating and Deleting more often than others. The parts of the state that are core to the application.

This is not an iron-clad rule, but it is a good metric to go by.

Looking at the current state object and the needs of our application, we can pick out the 'front runners' together.

For one, we'll be reading the "Messages" field quite often - for each user's contact.

There's also the need to edit and delete a user's message. Now, that's a front runner right there. The same goes for "Contacts" too.

Now, let's place them "in front."

Here's how.

Instead of having the "Messages" and "Contacts" fields nested, pick them out, and make them primary keys within the state object. Like this:

```
const state = {
  user: [],
  messages: [
    'msg1',
    'msg2' ],
  contacts: ['Contact1', 'Contact2']
}
```

This is still an incomplete representation, but we have greatly improved the representation of the app's state object.

Now let's keep going.

Remember that a user can message any of their contacts. Right now, the messages and contact field within the state object are independent.

After making these fields primary keys within the state object, there's nothing that shows the relationship between a certain message, and the associated contact. They are independent, and that's not good because we need to know which list of messages belongs to which user.

Without knowing that, how do we render the correct messages when a contact is clicked?

In the next lesson, we will restructure our object to help solve this problem.