Ajax Lab

In this lab we're going to make a major transition from simulated data to real data fetched from RESTful endpoints, processed through Redux, and displayed in the React Native components we've been building up. Don't worry, we've written most of the Redux code for you but if you can handle writing it yourself without relying on our code suggestions, please do it on your own.

A whole lot of prep we've been doing in prior labs will come to fruition in this lab. Are you excited? Let's get started!

Showtime!

In the ShowingTimes component we are giving the user a list of showing times but we've faked the showings data. Let's fetch real showings from the API server.

A *showing* is a particular film at a particular time, right? So every time the selected_date or the selected_film changes, we should refresh the showings. Make sense? Your mission in this section is to write a Redux middleware function that will read the showings and set those showings every time the user dispatches a "SET_SELECTED_DATE" or "SET_SELECTED_FILM".

1. Create a middleware. It should send a fetch to the API and in the callback, dispatch the showings it has retrieved. Something like this might work for you:

```
const fetchShowingsForDateMiddleware=({dispatch,getState})=>next=>action=> {
 // Complete the current action *first* so we have a good film and date!
 next(action);
 if (action.type==="SET_SELECTED_DATE"||action.type==="SET_SELECTED_FILM") {
  const selected_date = getState().selected_date.toISOString().split('T')[0]
  const film_id = getState().selected_film.id;
  fetch(`${host}/api/showings/${film_id}/${selected_date}`)
      .then(res => res.json())
      .then(showings => dispatch({ type: "SET_SHOWINGS", showings }))
      .catch(err => console.error("Couldn't fetch showings", err))
 }
}
```

2. Don't forget to register your middleware with the store.
3. Then, create a reducer case to handle "SET_SHOWINGS". It should do something like ...

```
case "SET_SHOWINGS":
  return { ...state, showings: action.showings };
```

4. You're probably setting your fake, hardcoded showings in FilmDetails.js and passing it into ShowingTimes as a prop. Remove wherever you're creating the fake showings currently.
5. In ShowingTimes.js, grab the showings in any way you like. (Hint: it could be through props drilling from App->Landing->FilmDetails->ShowingTimes or it could be with a useSelector() from react-redux).
6. Run and test. Once you have selected a film and a selected date on the Landing scene, your showings times should change in Film Details.

Fetching the tables and chairs

If you tap on a showing in FilmDetails, you'll navigate to the PickSeats scene. Currently this scene is using a hardcoded list of tables and seats. Let's grab real tables and seats from the API. The showing will tell us the theater_id, We can dispatch a request to fetch the list of tables and seats in that theater.

7. Add this reducer case:

```
case "SET_TABLES":
  return { ...state, tables: action.tables };
```

8. Add a middleware function called fetchTablesAndSeatsMiddleware to get the data from the /api/theaters/<theater_id>/tables/ endpoint. Attempt to write it yourself, but if you get stuck, something like this will do the trick:

```
const fetchTablesAndSeatsMiddleware = ({ dispatch }) => next => action => {
  const { theater_id } = action
  if (action.type === 'FETCH_TABLES_AND_SEATS') {
    fetch(`${host}/api/theaters/${theater_id}/tables`)
      .then(res => res.json())
      .then(tables => dispatch({ type: 'SET_TABLES', tables }))
      .catch(err => console.error("Couldn't fetch tables", err))
  }
  return next(action)
}
```

9. We're now ready to handle the return of data; we just have to fire off the request. In PickSeats, make sure you have a good showing and do this:

```
useEffect(() => {
  dispatch({type: "FETCH_TABLES_AND_SEATS", theater_id: showing.theater_id })
},[]);
const tables = useSelector(state => state.tables)
```

10. This will, when PickSeats loads, take the showing's theater_id and fetch all the tables in that theater. Then, when the tables are returned, the component will refresh with a whole new set of tables!

11. Bonus! Delete the tables.json file. We don't need it now that we're reading real data.

Fetching the reservations

As you test different showings, you'll see that the tables and chairs change. But the list of seats show them all to be available. In reality, some of those seats have been reserved by prior shoppers. To find out which, we need to fetch the reservations for this showing.

12. Add this reducer case

```
case "SET_RESERVATIONS":
  return { ...state, reservations: action.reservations };
```

13. Add a middleware function called fetchReservationsMiddleware to get the reservations.

```
const fetchReservationsMiddleware = ({ dispatch }) => next => action => {
 if (action.type === "FETCH_RESERVATIONS") {
  const id = action.showing_id;
  fetch(`${host}/api/showings/${id}/reservations/`)
   .then(res => res.json())
   .then(reservations => dispatch({ type:"SET_RESERVATIONS", reservations }))
   .catch(err => console.error("Couldn't fetch reservations", err))
 }
 next(action);
}
```

14. Edit PickSeats.js. In the useEffect, add another dispatch:

```
dispatch({ type: "FETCH_RESERVATIONS", showing_id: showing.id });
```

15. Run and test. You should be able to see all the reservations for this showing in the network traffic.

Now all that's left is to set the "status" property on each seat. What we'll do is iterate through the reservations, find that seat and set its status property to "seatIsTaken". Remember that in the styling lab, we changed the background color of "seatIsTaken" seats to red.

Showing taken seats

16. There's a dozen ways to add *status* to each seat. Try to write one without any help. But if you get stuck, here's one that is easy to implement.

17. Put this in PickSeats:

```
useEffect(() => {
 tables.forEach(table => {
  table.seats.forEach(seat => {
   if (reservations.some(res => res.seat_id === seat.id))
    seat.status = "seatIsTaken"
   })
 })
})
```

18. Run and test. You should be seeing all of the reserved seats as a different color. In fact, try several different showings. You should be seeing different seats reserved.

There's so much more we could do, but this will suffice for now. Enjoy a rest.