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 try to 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!

At a couple of places we are going to want to show the user a list of showing times that they can select. Fortunately we've extracted that functionality into the ShowingTimes component so we only have to change it in one place. We'll start by fetching the showings.

Remember that as of right now, you have a component called DatePicker.js. When the user chooses a date, we're dispatching a "SET_SELECTED_DATE" action. What would be cool is if we had something that would also fetch showing times when SET_SELECTED_DATE is dispatched. But you can't do that in the reducer since reducers must be pure functions. Sounds like a job for middleware!

1. Create a middleware to handle the fetch. 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 fetchShowingsMiddleware= ({dispatch,getState}) => next => action => {
 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(`http://localhost:5000/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 handler for "SET_SHOWINGS". It should do something like ...

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

4. Run and test. Once you have a selected film and a selected date, your showings times should change on both the Landing scene and in Film Details.

5. Bonus! You should no longer have a need for showings.json. Feel free to delete that file and change your code to no longer read from it.

Fetching the tables and chairs

If you tap on a showing, you're navigating 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.

6. First, PickSeats is going to need to be a class-based component because it'll need to have state. If it isn't already one, convert it. Make sure it has a constructor that receives props.
7. Make sure it is importing your Redux store.

8. In the constructor, it should be receiving a *showing* object as one of its props.navigation.state params. Make sure that's the case. If not, find where you're navigating to it, and pass the showing object in the navigation parameters.
9. In the constructor, connect this component with the store

```
this.state = store.getState();
store.subscribe(() => this.setState(store.getState()));
```

10. Find where you're loading the tables property with data from the hardcoded tables.json file. Get rid of that. Instead read the tables from this.state.tables.
11. Add a componentDidMount() method. In there, dispatch an action kind of like this:

```
store.dispatch({type:"FETCH_TABLES_AND_SEATS", theater_id });
```

Clearly this middleware doesn't exist and the reducer action to set the tables and seats doesn't exist so we should create them.

12. Create a new middleware that will fetch the tables and seats in a theater. The middleware might look like this:

```
const fetchTablesMiddleware = ({ dispatch, getState }) => next => action => {
  if (action.type === "FETCH_TABLES_AND_SEATS") {
    fetch(`http://localhost:5000/api/theaters/${action.theater_id}/tables/`)
      .then(res => res.json())
      .then(tables => dispatch({ type: "SET_TABLES", tables }))
      .catch(err => console.error("Couldn't fetch tables", err))
  }
  next(action);
}
```

13. The reducer case might look like this:

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

14. Run and test. If we've put everything together right, you should now be able to tap on any showing time and see the actual tables and actual seats in the theater. Try a few different movies and verify that the seats change from theater to theater.

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

Showing taken seats

Some of these seats have already been reserved. We should read those reservations from our API.

16. Create another middleware function for fetching reservations:

```
const fetchReservationsMiddleware = ({dispatch,getState}) =>next=>action => {
 if (action.type === "FETCH_RESERVATIONS") {
  const id = action.showing_id;
  fetch(`http://localhost:5000/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);
}
```

17. And the reducer case:

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

18. Edit PickSeats.js. In componentDidMount, add another dispatch:

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

19. Run and test. You should be able to see all the reservations for this showing.

Now all that's left is to set the "status" property on each seat. Here's what we want... If there is a reservation for this seat in this showing, the background color should be different. You should have already prepared for this in the styling lab.

20. Write a pure utility function called setSeatStatus(seat, reservations). This should take in a single seat and an array of reservations. It should return a seat. If that seat is found in the reservations, the seat returned should have an additional property called 'status' set to 'seatIsTaken'.
21. Use the function at the top of the render function like this:

```
const tables = this.state.tables.map(t => ({...t, seats: t.seats.map(s =>
setSeatStatus(s, this.state.reservations))}));
```

22. And then map through tables instead of this.state.tables.
23. Run and test. You should be seeing all of the reserved seats as a different color.

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