Ajax Lab

We've created a React Native app with components and we know how to debug it. In future labs we'll be learning how to draw things for the user to see so it is time to read real data so we have something to show them.

Remember back to our first lab. In it we stood up a RESTful API server. If that server is running, you'll be able to send requests to its URL and get responses. In this lab we're going to read that data.

Are you excited? Let's get started!

Preparing for all the fetches

Because we're assuming you know JavaScript and React very well already, we won't ask you to write the HTTP fetches from scratch. We've written the fetch requests for you.

1. Look in the starters folder and you'll see a file called repository.js. Copy that file into your project in a folder called utils.

2. Read through that file for a minute or two to see what it's doing. It's just some vanilla JavaScript to make HTTP requests to a server. We'll use the functions in this file to save you some writing and allow you to focus on the React Native.

App

Because certain variables will need to be synchronized in multiple child components, we should lift them up to the top component, which is App.

3. Edit App.js. Import useState at the top:

```
import { useState, useEffect } from 'react';
```

4. Inside the function, create state variables for cart, and selectedDate:

```
const [ cart, setCart ] = useState({});
const [ selectedDate, setSelectedDate ] = useState(new Date());
```

These can be passed down to child components as props.

5. In fact, we'll start off with the Landing component. Find where you're including <Landing /> and make it look like this:

```
<Landing selectedDate={selectedDate} />
```

This is us passing a prop called selectedDate from the parent component (App) down into a child component (Landing). Now, in the next section let's read that prop in in Landing.js...

Landing

6. Edit Landing.js. Read selectedDate as a prop:

```
export function Landing({selectedDate}) {
```

When the user visits the Landing scene, we will show them a list of all films that our theater is showing. We will GET those films in a useEffect() and put them in a state variable called films.

7. Import the useEffect and useState React hooks. Also import the fetchFilms library function we provided for you.

```
import { useEffect, useState } from 'react';
import { fetchFilms } from '../utils/repository';
```

8. Create a state variable called films.

```
const [ films, setFilms ] = useState([]);
```

9. Put your fetch in a useEffect that runs only on load:

```
useEffect(() => {
  fetchFilms()
  .then(newFilms => setFilms(newFilms));
}, []);
```

10. Let's add a console.log() just to show that the fetch has occurred properly:

```
console.log({films, selectedDate});
```

11. Alright. Run and test. If you see a list of films in the console, you've got it right.

Note: beginning now, we will not give you low-level instructions. As JavaScript and React experts, you already know about small details like how to import and how to use React hooks. Instead of writing the code for you, we'll be giving you more real-world requirements. If you're ever stuck, you can ask your pair programming partner or your instructor. Googling is fine, too.

FilmDetails

This component will try to sell the user on a particular movie so our app should show them all the details about it.

12. First, receive selectedDate via React props. (Hint: Remember to supply that prop like this <FilmDetails selectedDate={selectedDate} />).

13. Hardcode a filmId. Pick any number between 1 and 6:

```
const filmId = 1;
```

Later on this number will be supplied dynamically using route parameters.

14. Make a call to fetchFilm. (Hint: Put that in a useEffect() to avoid an endless loop).

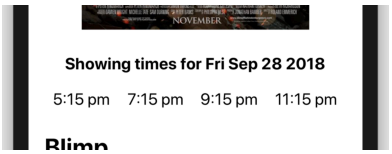
15. Get the film from it and put it in a state variable called film. (Hint: use useState to create a setFilms() function).

16. console.log() the film.

When you see a film in the console, you're done.

ShowingTimes

One of the great things about React is reusability of components. Our app will eventually show a list of showing times for a given film on a given selectedDate. We could duplicate this code in Landing and FilmDetails, but it is much cleaner to keep our code DRY by putting that in a separate component called ShowingTimes. Write the code once and use it in multiple places.



17. Create the new ShowingTimes component.

18. Have it receive a film and a selectedDate as props.

19. In a useEffect(), call the fetchShowings() library function. (Hint: fetchShowings() receives a film.id and selectedDate.)

20. After the promise resolves, put the showings in a state variable.

Don't worry about testing this one just yet. It'll be easier to test after the next chapter.

PickSeats

The PickSeats scene will eventually show the user which seats are available to purchase and which are already reserved for a given showing.

21. Make PickSeats receive two props: selectedDate and the cart. (Hint: Again, don't forget to pass those in the JSX in App.js).

22. In the PickSeats function, hardcode a showing:

```
const showing = {
  "id": 1,
  "film_id": 1,
  "theater_id": 1,
  "showing_time": "2028-04-19T17:30:00.000Z"
};
```

This will eventually be dynamic.

23. Fetch the theater given the showing.theater_id and it in a state variable called theater. Also fetch the reservations for this showing. Put them in a state variable called reservations. Since doing two setState()s can be tricky, here's the code for that:

```
const [theater, setTheater] = useState({})
const [reservations, setReservations] = useState([]);
useEffect(() => {
  fetchTheater(showing.theater_id)
    .then(t => setTheater(() => t));
  fetchReservationsForShowing(showing.id)
    .then(r => setReservations(() => r));
}, []);
console.log({ theater, reservations });
```

24. Run and test. See your theater and all the reservations for that showing?

Alright. That's enough fun for now. We'll do more of this later, but for now, if you've made it here, you can be finished.