Managing State Lab

At this point we have the ability to select a film by clicking on it and the ability to select a date but we're doing nothing with them. In this lab, we're going to change that. When you select one or the other, we'll show the film details and we'll read the showtimes.

1. First, since we'll be settting the internal state, we have to convert LandingPage from a stateless functional component into a class-based component. Go ahead and do that.
2. Run and test to make sure nothing has changed in its presentation and the events still work.

When the user clicks on a Film in FilmsList, we're firing a function called chooseFilm() which actually exists on the LandingPage and we're passing the entire film to it.

3. In the chooseFilm function, call this.setState() and pass it an object with a key of currentFilm and a value of the film that was just clicked on.
4. Look at the JSX of LandingPage.js. You currently have a <ShowingDate /> component and a <FilmsList /> component. Now add another component, your <FilmDetails /> component.
5. Edit FilmDetails.js. Make sure the JSX in FilmDetails says that if props.currentFilm is falsey, return a null. This will ensure that if the user hasn't picked a film, we see nothing.
6. Go ahead and test it out. You should see nothing rendered from FilmDetails because props.selectedFilm is undefined.
7. Change FilmDetails's JSX to display {props.currentFilm.whatever} in all the places where you've hardcoded data. (Hint: It still shouldn't show anything.)
8. Now let's bring all this together! Back in LandingPage, change the JSX for <FilmDetails /> to pass in this.state.currentFilm into the currentFilm prop of FilmDetails.
9. Run and test. If we've done all this right, you should be able to click on any film on the LandingPage and immediately see FilmDetails for that film appear.

Getting the showing times

10. Notice that we have another function called chooseDate() which receives in a date. Let's hook it up much like we did with chooseFilm above.
11. In that chooseDate function, call this.setState(), passing in an object with a key of selectedDate and a value of the date being passed in.
12. And since we've got the date and a film, let's show the user the showtimes!
13. Still in chooseDate(), call fetch("/api/showings/<film_id>/<showing_date>"). Of course, substitute real values for those placeholders in the angle brackets.
    Here are some examples:
    - /api/showings/1/2019-02-14
    - /api/showings/3/2019-04-15T12:00:00.000Z
    - /api/showings/4/2019-10-31T23:59:59.999Z
14. As you surely know, fetch() returns a promise. So chain a .then() to it and in the success callback, call this.setState(), passing in an object with a key of showings and the array that was returned from the RESTful API service.
15. Run and test. Open the developer tools in your browser and look at the network requests. When you choose a date, you should see the Ajax request being made and a response of an array of objects. Adjust unitl you do.
16. Last piece of this section ... Let's display the showings in <FilmsDetails />.
17. In LandingPage.js, find the JSX where you've included <FilmsDetails />. Add a prop to that tag; showings={this.state.showings}.
18. In FilmsDetails.js, you're displaying hardcoded showings. Remove that fake object from your code. Instead have FilmsDetails receive the showings in props. map through those instead.

19. Without any further coding changes, the user should be able to select a film, select a date and see a list of showings in FilmsDetails.

20. Bonus!! You're fetching showings when the user changes a date, but you might not be when they choose a film. Copy or extract that functionality Ajax functionality and make it happen when the user takes either action.