React Router Lab

This is the lab where our disjoint React components come together to become a Reach application. In it, we'll be creating routes and setting up navigation so the user can move from one component to another. Ready? Let's go!

1. Open index.js. Add a <BrowserRouter>.
2. Add a Route for each of LandingPage, PickSeats, Checkout, Login.
3. Run and test by typing in the URL for each. Make sure each route works.

Let's say we wanted the URL for the LandingPage to be "/".

4. Change the path for the LandingPage route to "/".
5. Test each again. Depending on how you wrote it, you may see more than one component for each URL. You may see the LandingPage **and** then another component under it.

But this isn't what we wanted.

6. Go ahead and discuss with your partner what needs to be done to make the LandingPage component show up only when the URL is "/". Then make that happen. (Hint: there is more than one way to solve this problem.)

Adding a 404 - Not Found route

7. Type in a URL of "/jslkdjkd". What happens? _____
   _____
8. Discuss with your partner; is this really what we want? Or do we want to tell the user when they've typed in a bad address?
9. Create a new component called NotFound. It should tell the user " That page doesn't seem to exist. Want to try one of these instead?" And then give them Links to some of the other pages.
10. Add a catch-all route for the NotFound page. Once you're finished, if the user types in a URL that is not one in our list, we should tell them by showing the NotFound component.
11. Run and test this. Make sure that any URL typed in that is not recognized will give us the NotFound component.

Linking components to others

FilmDetails has a list of showings. We're going to allow the user to click on one of those links to push them into the PickSeats component where they'll be able to choose their seats for that showing. Clearly we'll need to tell PickSeats which showing they want to see. So let's set that part up first.

12. Edit index.js and find the <Route> for PickSeats. Change this route to use a route parameter called showing_id. (Hint: use the colon (:) to mark this portion of the url as a parameter).
13. Run and test. At this point, you should only be able to get to PickSeats if your route now contains a second part, the showing_id. Put a fake showing_id in the url and you'll see your component.
14. Edit PickSeats. Have it read the Route Parameter called showing_id. Maybe something like this will do the trick:

```
const { showing_id } = props.match.params;
```

15. Run and test. Set a breakpoint or just console.log(showing_id) to make sure it is working. If you navigate to /pickseats/123, you should see *123* in the console. /pickseats/45678 should show *45678* in the console.

Last step; make FilmDetails pass the showing_id.

16. Edit FilmDetails. Change your list of showings to include a <Link> element that points to /PickSeats/123 or /PickSeats/456 or /PickSeats/789. You get the idea; any random hardcoded number for now.
17. Run and test. When you navigate to FilmDetails, you should now be able to click on any link for the showtimes and be sent to the PickSeats page for that particular showing. (Hint: Check the console to make sure everything aligns logically).

While we're in PickSeats, let's work with the checkout button. As of now, when the user clicks checkout, it just console.log()s.

18. After the console.log(), push the user to the Checkout route. (Hint: use props.history.push()).

Pushing the user to other routes

19. At this point, you have some simple event handlers for the button clicks in the Login and Checkout components. Let's simulate some authentication activities.
20. When the login button is clicked, in the login function, push the user to the checkout component.
21. And when the Checkout component is loaded, immediately push them to the Login component with a <Redirect> JSX tag.

If you've done this right, when the user visits '/checkout', they'll be immediately pushed to Login. When the user clicks login button, they'll be pushed to Checkout which auto-forwards them back to Login. (Be patient, we'll be using this in future labs).

22. Bonus!! You have a link on the Checkout component that says "Keep shopping". Make that link send the user to the LandingPage route when clicked. (Obviously you'll need to temporarily turn off your auto-forwarding to test this out).