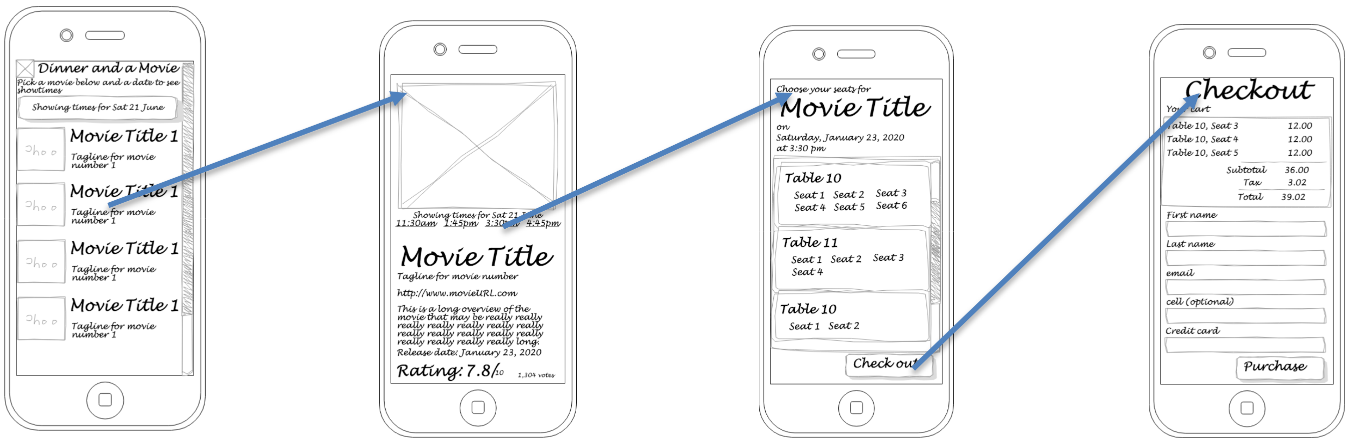
# Navigation Lab

Someday we may change how users get around our app but for now we're not seeing the need for tab navigation or drawer navigation. But it is crying out for stack navigation. You know how up until now you've been changing App.js, swapping out widgets in the MaterialApp widget? It felt pretty hokey, didn't it? Surely there's a better way to have the user navigate from one scene to the other! Well, there is and after this lab, your user will be able to start with the Landing widget, then ...

- Tapping a movie will navigate them to the FilmDetails scene.
- Tapping a showing time will navigate them to the PickSeats scene.
- Tapping the checkout button will navigate them to the Checkout scene.
- Tapping on the purchase button will navigate them to the Ticket scene.
- Tapping on the continue shopping button will navigate the user back to Landing.



Dinner and a Movie
Pick a movie below and a date to see showtimes
Showing times for Sat 21 June
Movie Title 1
Tagline for movie number 1
Showing times for Sat 21 June
11:30am 1:45pm 3:30pm 4:45pm
Movie Title
Tagline for movie number
http://www.movieURL.com
This is a long overview of the movie that may be really really really really really really really really really really really really really really really long.
Release date: January 23, 2020
Rating: 7.8/10
1,304 votes
Choose your seats for
Movie Title
on
Saturday, January 23, 2020
at 3:30 pm
Table 10
Seat 1 Seat 2 Seat 3 Seat 4 Seat 5 Seat 6
Table 11
Seat 1 Seat 2 Seat 3 Seat 4
Table 10
Seat 1 Seat 2
Check out
Checkout
Your cart
Table 10, Seat 3 12.00
Table 10, Seat 4 12.00
Table 10, Seat 5 12.00
Subtotal 36.00
Tax 3.02
Total 39.02
First name
Last name
email
cell (optional)
Credit card
Purchase

Remember, each type of navigation (stack, tab, drawer) share a common setup. Let's do that first.

## Setting up common navigation

1. Before we can navigate we have to install it

```
expo install @react-navigation/native
```

2. After that, you'll need to install some peer dependencies. You can use expo install to pick the right version that's compatible with your expo SDK:

```
npx expo install react-native-screens react-native-safe-area-context
```

3. Open App.js in your IDE. You have a single root element. It may be a fragment or a <View>. Replace that with a NavigationContainer. (Hint: Don't forget to import it from @react-navigation/native).

## Creating the basic stack navigator

Now that the common setup is finished, we could implement any of the three types of navigation schemes. All three have their own installs. Since we're going to use the stack navigator, let's set it up next.

4. Do the install

```
npx expo install @react-navigation/native-stack
```

5. Edit App.js. import createNativeStackNavigator from @react-navigation/native-stack.

6. Create your stack navigator. Call it Nav.
```
const Nav = createNativeStackNavigator();
```

Your scenes (pages, screens, whatever you prefer to call them) will appear on the screen wherever your "navigator" is. If you want to put in a header View that never changes, that would be fine. If you desire, go ahead and put that in a View with a logo and big text or whatever. But it also makes sense to allow each scene to take up the entire screen. These instructions will be written that way.

7. Add a <Nav.Navigator> directly inside the <NavigationContainer>. It doesn't matter where, as long as it is directly inside. ie. No <View>s or anything surrounding the <Nav.Navigator>.

Let's give it one route to begin with, the landing route.

8. Add a <Nav.Screen> for the landing page. I might make the screen name "landing" or "home" but you can name it whatever you choose. Make sure you have an expression as a child of the <Nav.Screen>. It must be a function that returns the <Landing> component in JSX:
```
<Nav.Screen name="landing">
  {() => <Landing selectedDate={selectedDate} />}
</Nav.Screen>
```

9. Tell the <Nav.Navigator> that it should look at this route as the initialRoute:
```
<Nav.Navigator initialRouteName="landing">
```

10. Run and test. You should see "landing" at the top of your app at this point.

## Adding the other routes

We now have one route, which is cool I guess. But the point of this is to navigate around. Let's add more routes.

11. Add a <Nav.Screen> for the FilmDetails scene, the PickSeats scene, the Checkout scene, and the Ticket scene.

12. If you change the initialRouteName and re-run, you'll be able to see one and only one scene at a time. Give that a try with a few different initialRouteNames.

Caution!:A change to initialRouteName is one of those very few things that a refresh won't catch automatically. You may need to refresh manually after each change to initialRouteName.

13. Set landing as the initial Route. That's just common sense, right?

Okay, we have the routes set up. Are you ready to make the navigation work? Let's go!

## Landing/FilmBrief to FilmDetails

When the user is interested in one of the films on the Landing scene, they'll tap it and be navigated to its FilmDetails. But here's a problem, it is in a <View> which do not have an onPress event. So we're going to use a <Pressable>.

14. In FilmBrief.js, create a navigation object:
```
const nav = useNavigation();
```

(Hint: you'll want to import it from @react-navigation/native.)

15. Surround your main <View> with a Pressable. Put something like this before your View.
```
<Pressable onPress={() => nav.push('filmDetails')}>
```

16. And close off the Pressable after the <View>:
```
</Pressable>
```

17. Run and test.

## Passing values with the navigation

Notice that no matter which FilmBrief you tap, you always see the same FilmDetail. That's because we hardcoded the filmId. Instead, we should be setting the filmId on tap and passing it to FilmDetail. Let's do that now.

18. First, in FilmBrief, write the filmId as a second parameter to the push() method:
```
<Pressable onPress={() => nav.push('filmDetails', { filmId: film?.id })}>
```

Now, let's read that route parameter.

19. Edit FilmDetails.js. make a call to the useRoute() hook and grab the filmId property our of params:
```
const filmId = useRoute().params.filmId;
```
or more concisely:
```
const { params: { filmId } } = useRoute();
```

20. Obviously, you'll need to remove the hardcoded filmId.

21. Run and test by tapping on different FilmBriefs. Make sure you get different FilmDetails.

22. Bonus!! If you have extra time, eliminate the extra HTTP call. Note that FilmBrief can send the entire film to FilmDetails and not just the filmId. If you pass the entire film, you don't need to call fetchFilm(filmId) at all. Make that happen.

Got em? Let's do another with fewer instructions.

## ShowingTimes to PickSeats

Remember our goal: when the user presses a showing in ShowingTimes we want the user to be sent to PickSeats for the correct showing.

23. Open ShowingTimes and add an onPress event to each <Text>:
```
onPress={() => nav.push('pickSeats', { showing: showing })}
```
Hints:
- Remember that this will navigate to the 'pickSeats' scene and pass the current showing.
- Don't forget to import useNavigate, and call it to get the nav object.

That's the sender. Let's write the receiver.

24. Edit PickSeats.js. Import useRoute and use it to get the showing parameter.

25. Replace the hardcoded showing parameter with this one.

26. Run and test. Choosing different showing times should give you different details. Choosing a showing for one film should give you different seatmaps than a different film.

## Navigating to CheckOut

When the user presses PickSeat's "Check out" button, we should be navigating to the Checkout scene. This one doesn't need to pass any routing parameters because we're reading all we need from props.

27. Edit PickSeats.js. Find the check out button and make it navigate to Checkout.

28. Run and test. Are you getting to Checkout? Move on when you are.

## Checkout to Ticket

In Checkout, when the user hits the purchase button, let's send their purchase to the server where their credit card will be charged and will send back the ticket numbers. We'll then get the ticket numbers and navigate the user to the Ticket scene.

29. First, locate the Button in Checkout.js. In its onPress, call a function:

```
<Button title="Purchase" onPress={checkout} />
```

30. Next, we should write the function. See if you can do it without peeking at the solution below. It is supposed to:
    - Set up a fake, temporary cart with four seatIds:
      ```
      const cart = {
        showing_id: 1,
        seats: [1, 2, 3, 4],
      }
      ```
    - Call buyTickets() which is imported from repository.js and receives the cart.
    - buyTickets() returns a promise, so you'll need to await it or use a .then().
    - In the callback, get its response which will be an array of reservations.
    - Iterate through those reservations any way you want to. Add each reservationId to an array.
    - Navigate to the Ticket component, sending the array of reservationIds in its routing object.

    Here's a possible solution to all of that:

```
function checkout() {
  const cart = {
    showing_id: 1,
    seats: [1, 2, 3, 4],
  }
  buyTickets(cart)
    .then(reservations => nav.push('ticket', { ticketIds: reservations.map(r => r.id)
}))
}
```

31. If we've done this right, you should be able to tap the button and navigate to Ticket. You can also look in the database and see that you've purchased four reservations for showing_id 1, seats 1, 2, 3, and 4.

32. Now let's read those ticketIds. Edit Ticket.js. Instead of using hardcoded TicketIds, read them using the useRoute() hook like we've done a few times now.

33. Run and test. You'll know you've got it right when you can tap the button and see four tickets for whichever hardcoded seatIds you used. Look at the ticketId on the screen. You'll see those ticket numbers increasing each time you hit the button because you're actually hitting the database.

## Ticket back to Landing

Last one. This one is simple. Since the user has purchased their tickets, let's offer to send them back to the Landing page.

34. Add a button to Ticket.js that says "Find more movies" or something. When the user taps it, they should be navigated back to the Landing scene.

35. Run and test. When you can navigate through the app, You're finished.

Congratulations! That was a lot of good work!

## Bonus! Change the header

36. You can call yourself complete but if you have extra time, change the words in the header to be something other than the name. You'll do this by changing:

```
<Nav.Screen name="filmDetails">
```

to:

```
<Nav.Screen name="filmDetails" options={{ title: "Film Details" }}>
```

37. Do this for all the screens except the first. The Landing screen shouldn't have a header at all:

```
<Nav.Screen name="landing" options={{ headerShown: false }}>
```