9B State management libraries Lab

Remember that your app runs thick-client on your user's machine. Every widget is encapsulated, even ignorant of other widgets except children. Heck, they're even unaware of who their parents are! This is a wonderful thing but one of the tradeoffs is sharing data among all these independent widgets. How do we set data in one widget and read it in another?

This is one of the problems that state management attempts to solve. It usually works this way ... you have a state container that is separate from every widget. When you change state data, it is saved to the state container. And when you open any widget that needs state data, you'll read it from that state container.

There are lots of really good state management tools available. We've decided to use the Riverpod library for this project.

You've already implemented several things from Riverpod to this point but now that we know some more about it, let's put it in all the other places its needed.

main.dart

The first step with Riverpod is to set the bounds of the state container with a ProviderScope. We decided that Riverpod should have control of our state throughout the entire app rather than limiting it to a certain area.

1. Edit main.dart in the IDE. Look at the main() function. Remember that we wrapped everything in a ProviderScope(). That gives Riverpod freedom throughout our app.

Step two is to define what gets re-rendered when something changes. You do that by giving that section a Consumer() widget. To keep it simple, we decided to just redraw everything when state changes.

2. Look at the build() method. Remember that we wrapped everything in a Consumer(). This is us saying to redraw everything when the state container sees a new value.

Step three is to create one or more watches. Not every shard of data is important. We have to say what data is significant enough to trigger a re-render. You mark data as significant by watching it.

3. The next few lines say to create a watch for films and showings:

```
var films = watch(filmsProvider).state;
var showings = watch(showingsProvider).state;
```

These signal the Consumer to redraw when films and/or showings are changed anywhere else.

From now on, when you need to change state and want to app to refresh itself in response, you can add a watch for that data. Feel free to try this out.

Landing

4. Edit Landing.dart. See if you can find this line:

```
List<Film> films = context.read(filmsProvider).state;
```

This says "Go reach into state and read all the films that are in there." On the first render, there are none. So how do the films ever get into state? Good question. Read on...

5. Look elsewhere in this same file for these lines:

```
@override
void initState() {
  super.initState();
  fetchFilms().then((films) => context.read(filmsProvider).state = films);
}
```

Aha! There's the read; We're fetching the films and once they're read from the API server, we're setting state to be whatever the server responded.

And let me remind you that we told main.dart to be watching for a change of films state and to redraw itself when films are altered. Thus, Landing initially draws itself with no films, then it fetches the films, then it sets the films, and is redrawn once those films are set.

Is this all coming into focus? (Pun intended. Please forgive the dad joke).

FilmBrief to FilmDetails

Now let's do a new one. Remember that tapping FilmBrief navigates to FilmDetails but if they tap on Movie #1, they should get the details for Movie #1 and if they tap on Moviee #2, they should get the details for Movie #2.

Doesn't it sound like we should save the film chosen into state before we travel from filmBrief/Landing to FilmDetails?

6. Edit FilmBrief. Add these lines to the top if they're not already there:

```
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'state.dart';
```

7. In the button's onPressed, before you navigate away, save the current film to state:

```
onPressed: () {
  context.read(selectedFilmProvider).state = film;
  Navigator.pushNamed(context, '/film');
},
```

Now let's go read that selected film in FilmDetails. Remember that in the Value Widgets Lab, we told it to set the film by fetching film #1. It's time to get that film data directly from state.

8. Edit FilmDetails. Find where we're fetching the film and remove the fetch. Remember that we're reading the film at the top of the build method. Go find that read line to remind yourself.

9. Run and test. When you can tap on any FilmBrief and see the FilmDetails for that film, it's working as it should.

Now that you've got an idea generally how state management works, let's enable it for some other scenes.

ShowingTimes to PickSeats

We can successfully navigate to FilmDetails. Remember that FilmDetails has a ShowingTimes widget in it. ShowingTimes is given a list of times to display. When the user taps one, they should be navigated to PickSeats where they'll ... well ... pick their seats. So clearly we need to save the chosen showing into state when they're leaving ShowingTimes and read it again when entering PickSeats.

10. Edit ShowingTimes.dart. Add these lines to the top if they're not already there:

```
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'state.dart';
```

11. Find where you're handling the onPressed event. In that handler, before you navigate away, save the showing to state just like you've done several times now. (Hint: it should be a single line of code and it should have 'selectedShowingProvider' somewhere in it.)

That's the saving of the selected showing. Let's now do the reading.

12. Edit PickSeats.dart. Add these to the top:

```
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'state.dart';
import 'Film.dart';
```

13. Find the _PickSeatsState class and add this. But where there's the blank, fill it in with the read from the selectedShowingsProvider.

```
var _film = Film();
var _theater = {};
var _selected_showing = {};
var _reservations = [];
var _selected_date = DateTime.now();
List<Map<String, dynamic>> _cart = [];
@override
void initState() {
  super.initState();
  // Get the selected showing from state
  _selected_showing = context.read(selectedShowingProvider).state;
  // Ask the API server for all of the tables and seats for this theater.
  fetchTheater(theater_id: _selected_showing["theater_id"])
      .then((theater) => setState(() => _theater = theater));
  // Ask the API server for all the reservations for this showing.
  fetchReservationsForShowing(showing_id: _selected_showing["id"])
      .then((reservations) => setState(() => _reservations = reservations));
}
```

14. And lastly read your saved values from state in the build() method. Again, fill in the blanks with reads from state:

```
_film = ___________________________________ ?? Film();
```

15. Set a breakpoint or two, run and test. You should be able to tap on any showing and see that the showing, selected film, and theater with all tables are set properly.

PickSeats to Checkout

Awesome! You can move with real data through PickSeats. After we allow the user to pick some seats, they'll want to check out. But first we need to get some data to show up in the cart. We'll force the PickSeats FAB to put some seats in the shopping cart before we navigate to Checkout.

16. Locate your FAB in _CheckoutState and make its onPressed handler look like this. Fill the blank with saving the cartProvider to state.

```
onPressed: () {
  // Fake putting four seats in the cart.
  _cart.add({"table_number": 10, "seat_number": 1, "price": 10.75});
  _cart.add({"table_number": 10, "seat_number": 2, "price": 10.75});
  _cart.add({"table_number": 10, "seat_number": 3, "price": 10.75});
  _cart.add({"table_number": 10, "seat_number": 4, "price": 10.75});
  ____________________________________________________
  Navigator.pushNamed(context, '/checkout');
},
```

Now all of that is loading up the cart with fake seats so that when the user goes to Checkout they'll be able to read something from the cart. Here we go!

17. Edit Checkout. If these imports aren't already there, add them:

```
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'state.dart';
```

18. Create a _cart variable to keep track of picked seats. Add this to the top of the _CheckoutState class:

```
List<Map<String, dynamic>> _cart = [];
```

19. Read the _cart data from state in case there was something already in it. If you don't have an initState method, create one and read the cart from state.

20. Now our cart is loaded. Debug your app and make sure that the data is safely in the _cart in Checkout.