9B State management libraries Lab

Remember that your app runs thick-client on your user's machine. Every widget is encapsulated -- 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?

If there is a parent-child relationship, we can pass data between them. But in our app, we have data in Landing that is needed in FilmDetails, and in Checkout, and in PickSeats. And none of them are parents or children to the others.

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. All of the libraries do essentially the same thing but in different ways.

In this class we are going to implement the simplest possible state container, raw_state.

Setting up the state container

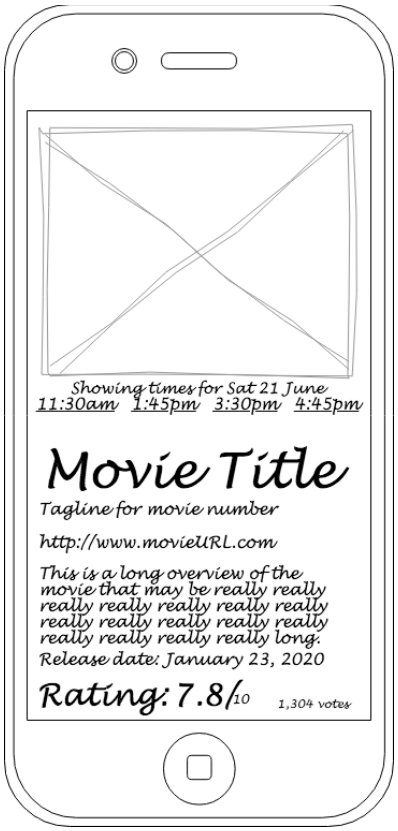
1. Either flutter pub add raw_state or edit pubspec.yaml, add raw_state to the dependencies section. This will download the raw_state library package.

You'll include this library everywhere that we read from or write to rawState. Don't forget to import it at the top of every file where it's needed as we write our code. In fact, we're going to stop reminding you to import because you're surely familiar with importing by now.

2. To begin, we need to set some default values. Add a couple of lines to main.dart:

```
void main() {
  rawState.set("selectedDate", DateTime.now()); // <-- Add this line
  rawState.set("cart", <String, dynamic>{"seats": []}); // <-- And this one
  runApp(const MyApp());
}
```

Getting the film and date in FilmDetails



FilmDetails is the scene that shows the user more details about the film (hence the name). It has a bigger movie poster, a link to the film's website, a synopsis, and other viewers' votes. It also lists the showing times for the selected date.

Clearly FilmDetails needs to know the selectedFilm and the selectedDate. Let's get these from our rawState state container.

3. First we should write the selected film into rawState. Let's do that when they tap the FilmBrief. Look in film_brief.dart for the GestureDetector. In its onTap handler, add this:

```
rawState.set("selectedFilm", film);
```

4. Next we should write the selectedDate into rawState when the user taps a date. Edit date_picker.dart. Find where the user taps a date and add this:

```
rawState.set("selectedDate", date);
```

Now they're both ready for FilmBrief to read them out of Global state.

5. Edit film_details.dart. At the very top of the build() method, add this:

```
Film selectedFilm = global.get<Film>("selectedFilm");
DateTime selectedDate = global.get<DateTime>("selectedDate");
```

Let's prove that FilmDetails is actually reading the stuff.

6. Start with a Column() widget in the build method.
7. Add a Image.network() pointing to the movie poster.
8. Add a ShowingTimes() widget
9. Add a Text() with the movie title.
10. Add a Text() with the movie tagline.
11. Add a Text() with the movie homepage.
12. Add a Text() with the movie overview.
13. Add a Text() with "Rating: ${selectedFilm.voteAverage}/10 ${selectedFilm.voteCount} votes".
14. Run and test. There's bound to be way too much stuff on the screen. Feel free to wrap it in a SingleChildScrollView so you can see it all. And don't worry about the styling. We'll fix that stuff later.

15. When you can click on different FilmBriefs on the Landing scene and different dates in DatePicker, and have the FilmDetails show the proper data, you've got this one right!

Let's do another one.

From ShowingTimes to PickSeats

When the user taps a showing in ShowingTimes(), we're navigating to PickSeats() but PickSeats() needs to know which showing was selected. It also needs to know what the theater looks like and the status of each seat (reserved or available). Let's read the theater information and save it to Global state as they're leaving ShowingTimes().

16. Edit ShowingTimes(). In preparation, add this enum to the bottom of the file (not inside a class):

```
enum SeatStatus {
  available, // Duh
  inCart, // Available, but you have it in your cart
  reserved // Somebody else has this seat reserved
}
```

17. Near the top of the _ShowingTimesState class, add a new cart property and read it from Global state:

```
Map<String, dynamic> cart = global.get<Map<String, dynamic>>("cart");
```

18. Next, add this method:

```
/// Returns a seat's status
///
/// seat: The seat whose status we're examining
/// reservations: All the reservations from other customers. These are unavailable
/// cart: The current shopping cart. You may already have this seat in your cart.
SeatStatus getSeatStatus(Map seat, List<Map<String, dynamic>> reservations,
    Map<String, dynamic> cart) {
  bool seatIsInCart = cart['seats'].any((heldSeat) => heldSeat == seat['id']);
  if (seatIsInCart) {
    return SeatStatus.inCart;
  }
  bool seatIsReserved =
      reservations.any((reservation) => reservation['seat_id'] == seat['id']);
  if (seatIsReserved) return SeatStatus.reserved;
  return SeatStatus.available;
}
```

19. Find the function that creates each showing widget. Make it look like this:

```
Widget _makeShowingWidget(dynamic showing) {
  DateTime showingTime = DateTime.parse(showing["showing_time"]);
  String timeString = DateFormat.jm().format(showingTime.toLocal());
  return TextButton(
      onPressed: () => chooseShowing(showing),
      child: Text(timeString));
```

```
}
```

Notice that the onPressed handler calls a new function called chooseShowing.

20. Create chooseShowing:

```
void chooseShowing(dynamic showing) {
  // Ask the API server for all of the tables and seats for this theater
  var theaterFuture = fetchTheater(theaterId: showing['theater_id']);
  // Ask the API server for all reservations for this showing
  var reservationsFuture =
      fetchReservationsForShowing(showingId: showing['id'])
          .then((reservations) => reservations as List)
          .then((reservations) => reservations.cast<Map<String, dynamic>>());
  // After the server fully responds, mark each seat as available, inCart, or reserved
  Future.wait([theaterFuture, reservationsFuture]).then((fa) {
    var theater = fa[0] as Map<String, dynamic>;
    var reservations = fa[1] as List<Map<String, dynamic>>;
    for (var table in theater['tables']) {
      for (var seat in table['seats']) {
        seat['table_number'] = table['table_number'];
        seat['status'] = getSeatStatus(seat, reservations, cart);
      }
    }
    //TODO: Set the theater in rawState here
    //TODO: Set the selectedShowing in rawState here
    Navigator.pushNamed(context, '/pickseats');
  });
}
```

21. See those two TODOs? Write the code to make them happen. (Hint: they're both one-liners).

At this point we've loaded up everything that PickSeats needs. Let's edit pick_seats.dart and read them from Global state.

22. Edit pick_seats.dart. Add these lines to the State class:

```
DateTime selectedDate = rawState.get<DateTime>("selectedDate");
Map<String, dynamic> selectedShowing =
    rawState.get<Map<String, dynamic>>("selectedShowing");
Map<String, dynamic> cart = rawState.get("cart");
Map<String, dynamic> theater = rawState.get<Map<String, dynamic>>("theater");
```

23. In the build() method, go ahead and print() selectedDate and selectedShowing.

24. Run and test. Do the values make sense?

25. Now print() theater. You should see a list of about 15 tables, each with 1-4 seats. Each seat will have a status of available or reserved. If different showings give you different statuses for the seats, you've got it right.

We can eventually allow users to add available seats to their cart and then purchase them.