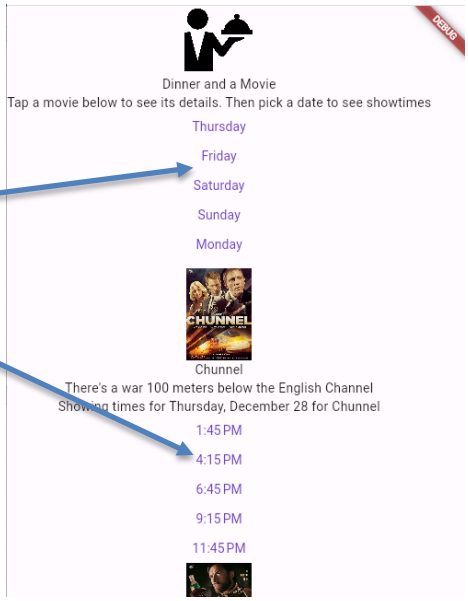
9A Stateful widgets deep dive Lab

Unfortunately, our app is only ever displaying the times for today's showings, never for tomorrow's or the next day's. In this lab you're going to allow the user to pick a date and refresh the list of showing times for the selected day.

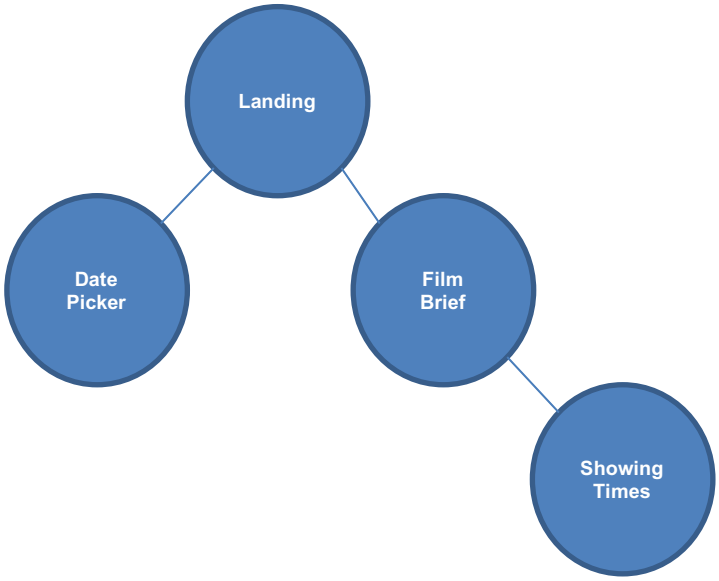


1. Try out the current situation. On the landing page notice the showing times. They're all for today. In the DatePicker widget, tap on a different day. Notice that the showing times have not changed. But they should; the showing times are different for each day.

The problem is that the FilmBrief widget always hardcodes the selectedDate as DateTime.now(). Our app should A) Change the selectedDate when a user taps in DatePicker and B) Use selectedDate in the ShowingTimes widget.

Oh no! DatePicker and ShowingTimes are encapsulated and can't share data.

Remember, when this happens we can lift state up to their common ancestor in the widget tree. In this case, the common ancestor is Landing.



Lift selectedDate up to Landing

2. Edit landing.dart. In Landing's State class, create a new DateTime variable called selectedDate. Initialize it to DateTime.now().

3. Send the selectedDate down from Landing to each FilmBrief.

4. Edit film_brief.dart. Modify FilmBrief's constructor to receive the selectedDate (along with the already-existing film).

5. Make sure you remove anywhere you've hardcoded selectedDate to be DateTime.now() and rely on the selectedDate that you just passed in.

6. Run and test. You shouldn't see anything different so far.

Passing a setter to DatePicker

7. Create a method in Landing called setSelectedDate(). Have it receive a DateTime and return nothing -- a void.

8. In this method, run setState() and set the selectedDate variable to whatever was passed in to setSelectedDate().

9. Find where you're instantiating DatePicker() and pass this method down in its constructor.

Obviously you'll have to edit the DatePicker widget to receive the new parameter.

10. Edit date_picker.dart, changing the constructor to receive a final void Function(DateTime) called setSelectedState.

11. Find the onPressed event handler. Call your setSelectedState method here.

12. Run and test. You will see a label change indicating the proper date. But the times still haven't changed yet. We'll fix that next.

Convincing ShowingTimes to re-draw itself

Flutter, in trying to be efficient will re-use widgets in memory. When Flutter re-draws, it saves old widgets and just plugs them back in to the widget tree. You must tell Flutter to re-render.

13. Edit ShowingTimes. Add this to the State class:

```
@override
void didUpdateWidget(oldWidget) {
  if (widget.selectedDate.day == oldWidget.selectedDate.day) {
    return;
  }
  fetchShowings(filmId: widget.film.id, date: widget.selectedDate)
      .then((showings) => setState(() => _showings = showings));
  super.didUpdateWidget(oldWidget);
}
```

Notice that it is exactly the same as initState() but initState() runs only one time; when the widget is first being rendered. We want to fetch more showings each time the widget is re-rendered and the date has changed. So we'll let the didUpdateWidget() lifecycle method do its work.

14. Run and test. You'll know you've got it working when you can tap on a day in the app and see the showing times change.

15. Bonus!! In order to keep your code DRY, take what's common between initState() and didUpdateWidget() and pull it into a third method that the others call.