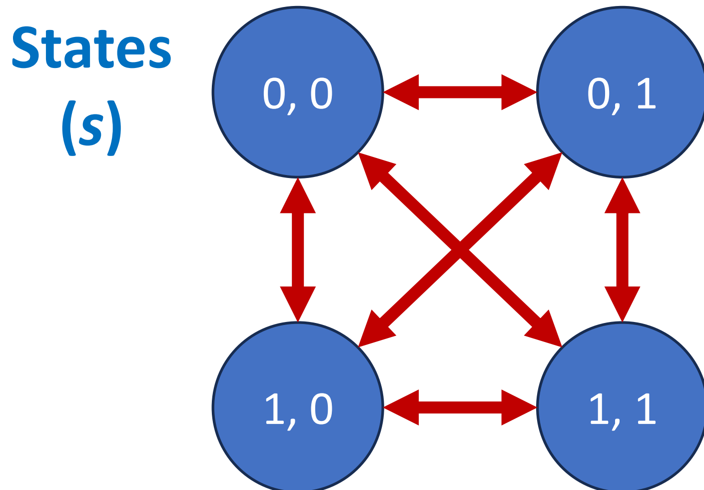
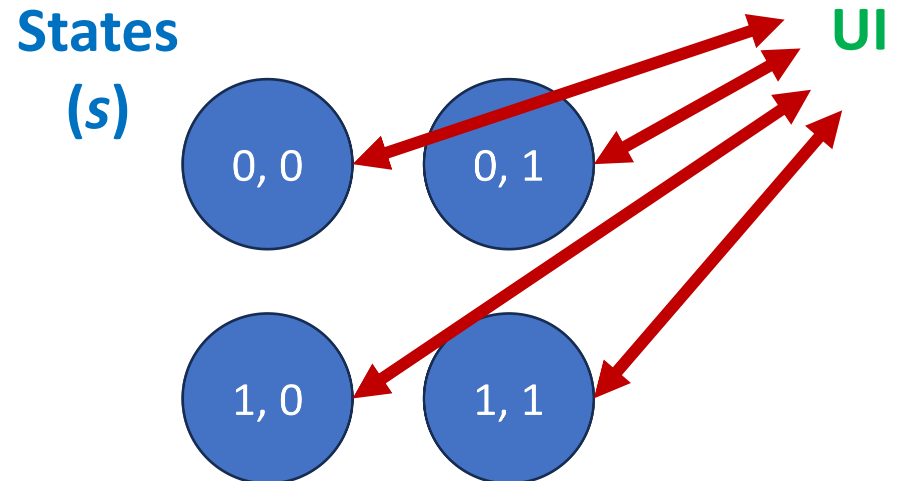
Widgets & Theming

Shan-Hung Wu
CS, NTHU

Declarative vs. Imperative UI

- Imperative UI: ***change*** UI in response to state changes
- Declarative UI: ***rebuild*** UI as a function of state





Imperative `UserProfile`

- JavaScript with direct DOM manipulation
- O(s^2) code paths to states, and then UI
- Hard to trance & debug

```
// states
let isUserLoggedIn = false;
let isProfileComplete = false;

// UI
...
document.getElementById('loginButton')
    .addEventListener('click', toggleLogin);
document.getElementById('editProfileButton')
    .addEventListener('click', editProfile);

function toggleLogin() {
  if (!isUserLoggedIn) {
    ... // change states & UI
  } else {
    ... // change states & UI
  }
}

function editProfile() {
  if (!isUserLoggedIn) {
    ... // change states & UI
  } else if (!isProfileComplete) {
    ... // change states & UI
  } else {
    ... // change states & UI
  }
}
```

Declarative `UserProfile`

- Single code path defined in `build()`
- Easy to trace
- Flutter optimizes “rebuilding” of UI

```
@Override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(...),
    body: Center(
      child: Column(
        children: <Widget>[
          if (!isUserLoggedIn) {
            ... // UI & future state changes
          } else if (!isProfileComplete) {
            ... // UI & future state changes
          } else  {
            ... // UI & future state changes
          },
        ],
      ),
    ),
  );
}
```

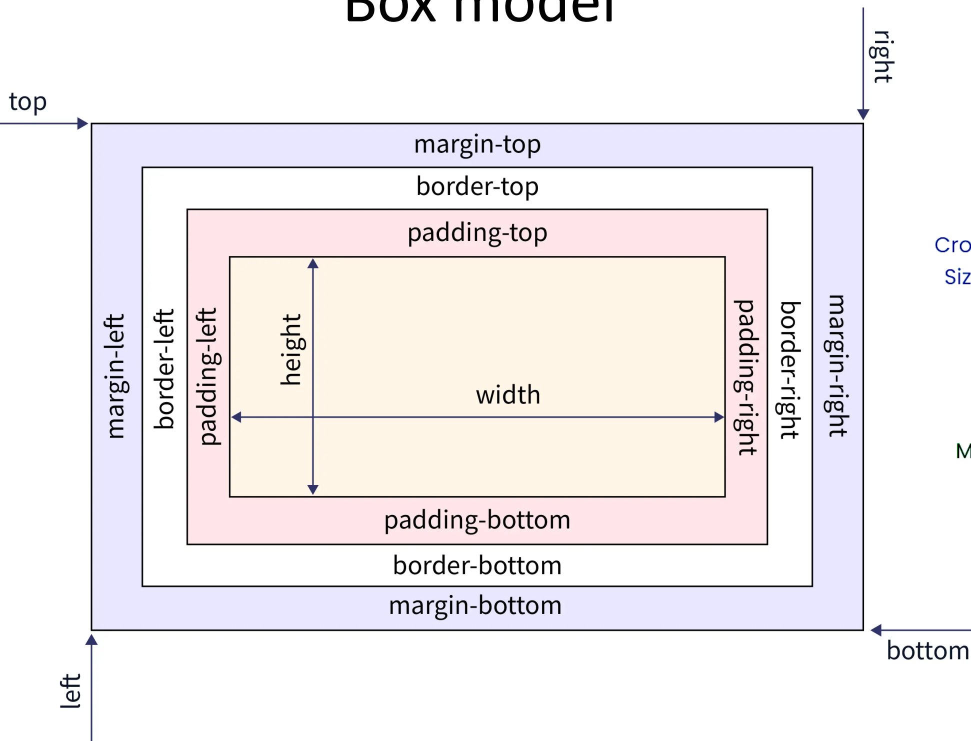
Declarative Widget Tree

- Widgets are more than just UI components
- Functional roles:
  - Interaction & gestures
  - ***Layout & positioning***
  - ***Theming & styling***
  - Navigation & routing
  - Dependency injection & state management
  - Animation
  - Integration with platforms & services
  - Accessibility & internationalization

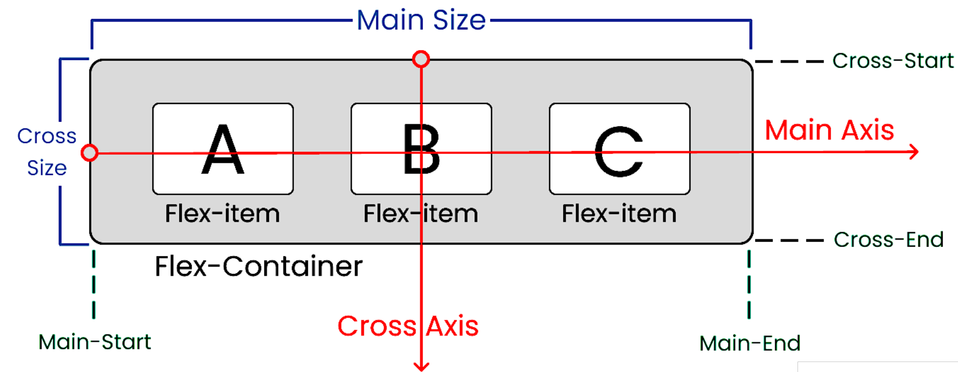
Layout Widgets

- Two main families: ***box*** vs. ***flexbox*** models

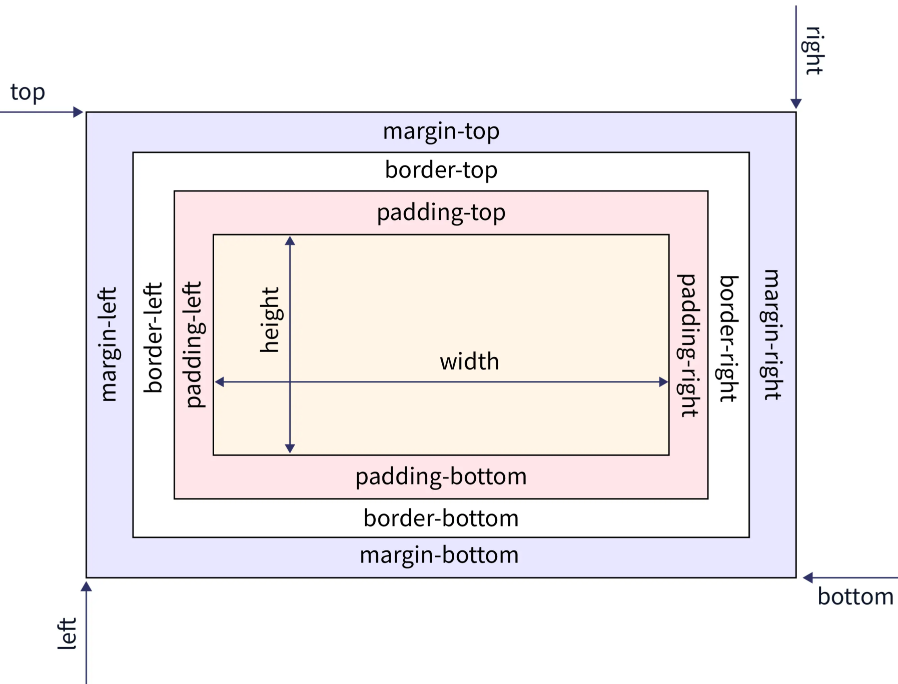
Box model



Flexbox model



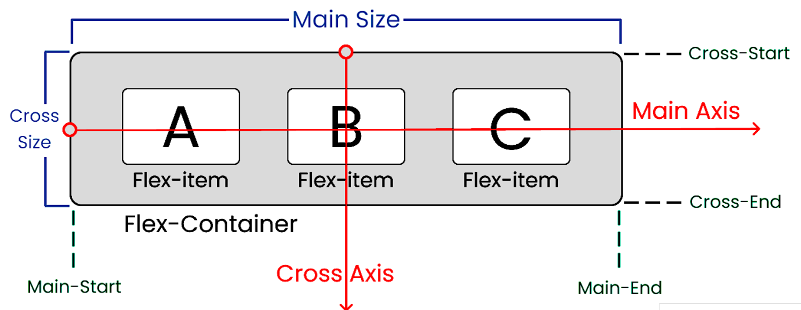
Box-model Layout Widgets



- Lightweight `Container`: `Padding`, `SizedBox`, `FittedBox`, `ConstrainedBox`, `ClipRRect`, etc.

```
Container(
  margin: EdgeInsets.all(20),
  padding: EdgeInsets.only(top: 8.0)
  decoration: BoxDecoration(
    color: Colors.blueAccent,
    border: Border.all(
      color: Colors.black,
      width: 3,
    ),
    borderRadius: BorderRadius.circular(10),
  ),
  child: ...,
),

SizedBox(
  width: 100, // Fixed width
  child: AspectRatio(
    aspectRatio: 3 / 2, // width / height
    child: ...,
  ),
),

Stack( // Along z-axis
  children: <Widget>[
    Positioned(
      top: 10,
      left: 10,
      child: ...,
    )
  ]
),
```

Flexbox Layout Widgets




```
Row( // Or use Column
  // occupy all available space from parent
  mainAxisSize: MainAxisSize.max,
  crossAxisAlignment: CrossAxisAlignment.center,
  children: <Widget>[
    Container(...), // Fixed size
    Spacer(flex: 1),
    Flexible(
      flex: 5,
      fit: FlexFit.tight, // Same as Expanded widget
      child: ...,
    ),
    Flexible(
      flex: 4,
      // Child's intrinsic size first
      fit: FlexFit.loose,
      child: ...,
    ), ],
)
```

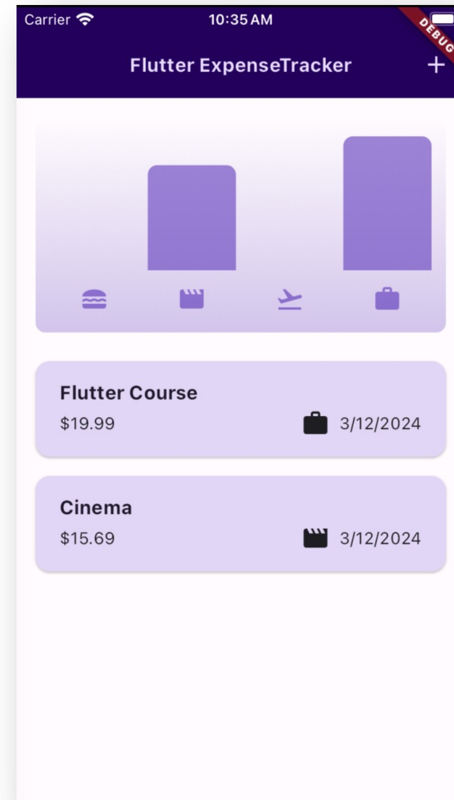
- Use `Align`, `Center`, to position individual child
- Use `FractionallySizedBox` for for sizing child to fraction of total available space

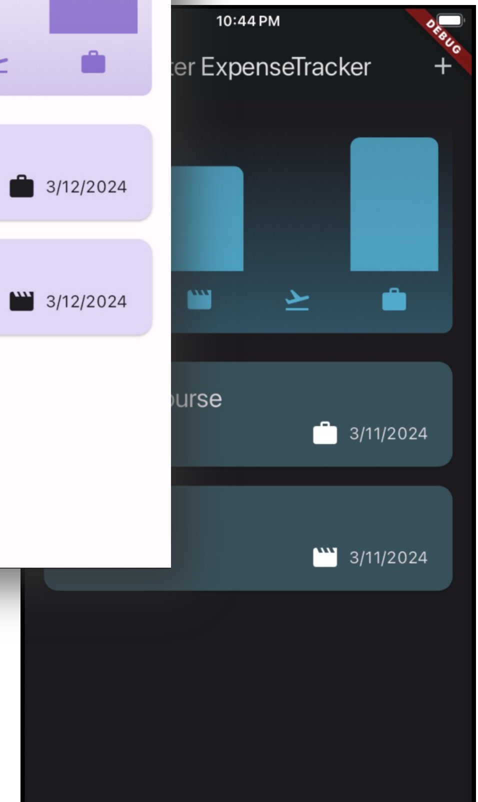
Next Step

- Widget Catelog
- Flutter Widgets of the Week

Expense Tracker App

- `ListView`
- Navigator & modals
- Async programming
- User input & validation
- `SnackBar`
- Theming





ListView

- To display expense items
- Prefer `builder` constructor whenever possible
  - Only items in screen are built and rendered
  - Works with infinite ccrolling

```
ListView(
  children: ...,
),
```

```
ListView.builder(
  itemCount: items.length,
  itemBuilder: (context, index) {
    return ...; // item Widget
  },
),
```

Keys for List Items

- `Key` required if item may be added, updated, or removed
  - To avoid bugs during rendering (to be discussed later)
- Also required by `Dismissable`
- Only needs be unique within parent Widget

```
ListView.builder(
  itemCount: items.length,
  itemBuilder: (context, index) {
    return ListTile(
      key: ValueKey(expenses[index]), // or ObjectKey(...)
    );
  },
),
```

Local v.s Global Keys

- Local keys: `ValueKey` or `ObjectKey`
  - Cheap
  - Commonly used in lists

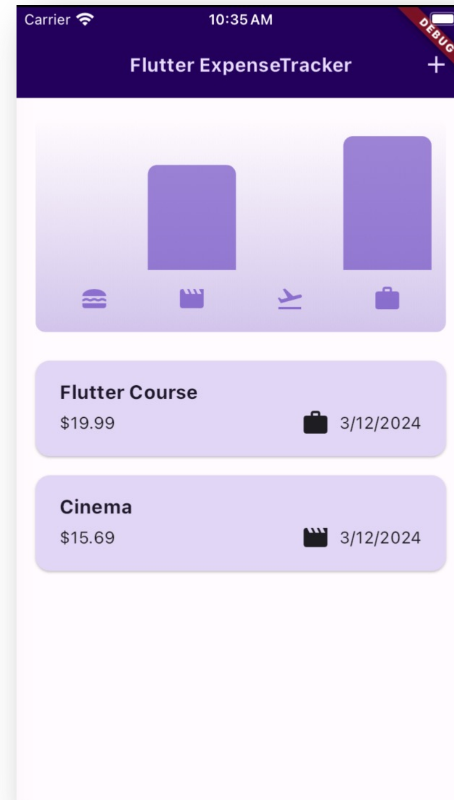
- Global keys: `GlobalKey`
  - Expensive
  - Allow non-widget code (form validation, animation, etc.) to access a widget in widget tree

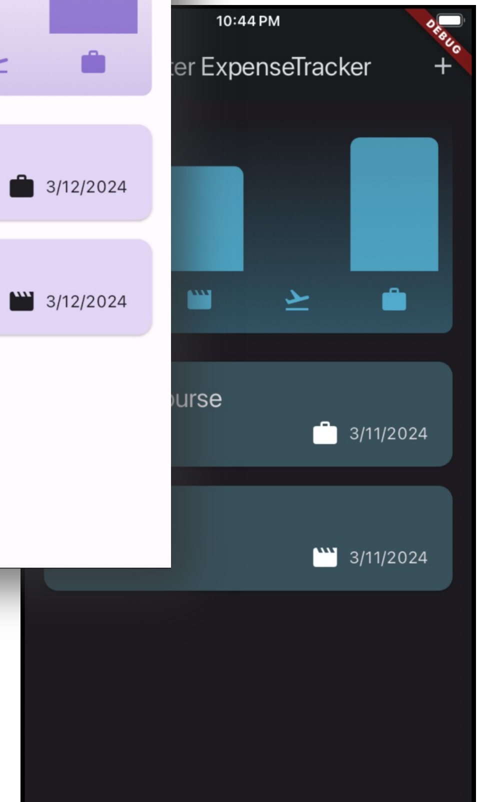
Watch Out Nested List

- Error when you place `ListView` directly under `Column` or `Row` in `expenses.dart`
- Reason:
    1. `ListView` tries to expand to fit all available space
    2. `Column` gives unbounded vertical space (so to be as big as children)
- Fix: wrap `ListView` with `Expanded`

Expense Tracker App

- ListView
- Navigator & modals
- Async programming
- User input & validation
- SnackBar
- Theming





Modals & Context

```
// expenses.dart
void _openAddExpenseOverlay() {
  showModalBottomSheet(
    isScrollControlled: true,
    context: context,
    builder: ...,
  );
}
```

- Action of `AppBar` call `showModalBottomSheet()` to display a full-screen, scrollable modal
- `BuildContext` is passed around. What is it?
  - Metadata on widget (incl., location) relative to entire widget tree
- So, modal knows "where to return" when closing

Navigation Stack

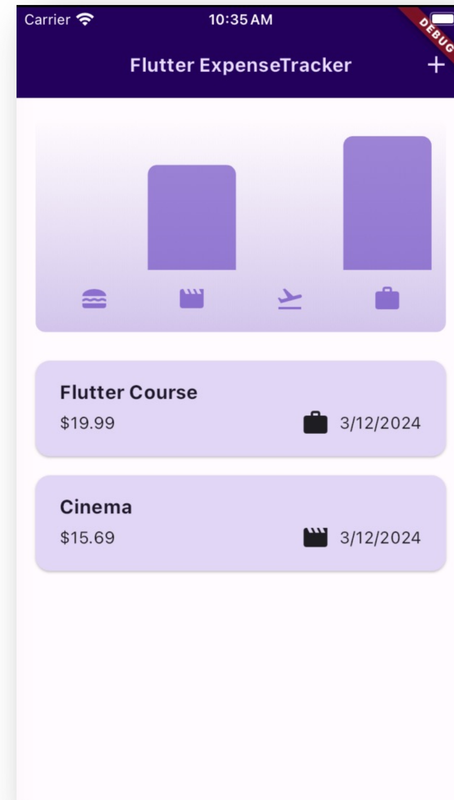
- `Navigator` allows screen push & pop
- Communication between screens
  - Push: callback functions
  - Pop: Return `Future`

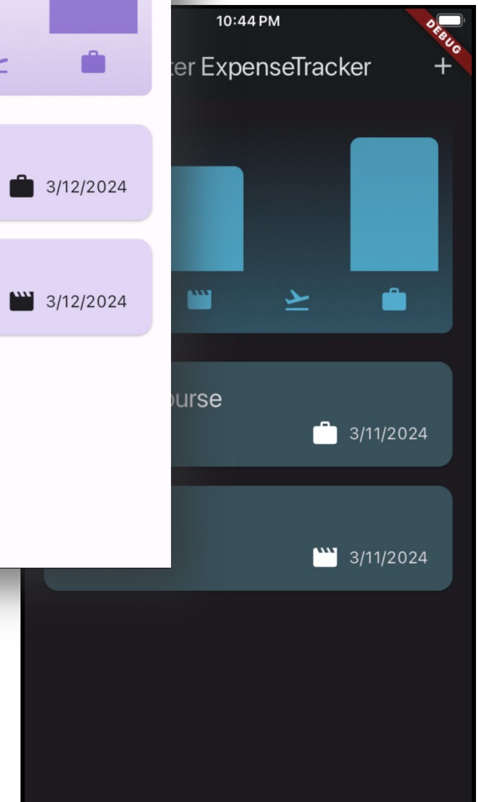
```
// expenses.dart
void _openAddExpenseOverlay() {
  showModalBottomSheet(
    isScrollControlled: true,
    context: context,
    builder: (ctx) => NewExpense(
      onAddExpense: _addExpense,
    ),
  );
}
// new_expense.dart
void _submitExpenseData() {
  ... // Call onAddExpense
  Navigator.pop(context);
}
```

```
// new_expense.dart
void _presentDatePicker() async {
  final DateTime? pickedDate = await showDatePicker(...);
  setState(() {
    _selectedDate = pickedDate;
  });
}
```

Expense Tracker App

- `ListView`
- Navigator & modals
- Async programming
- User input & validation
- `SnackBar`
- Theming





Async. Programming: `Future`

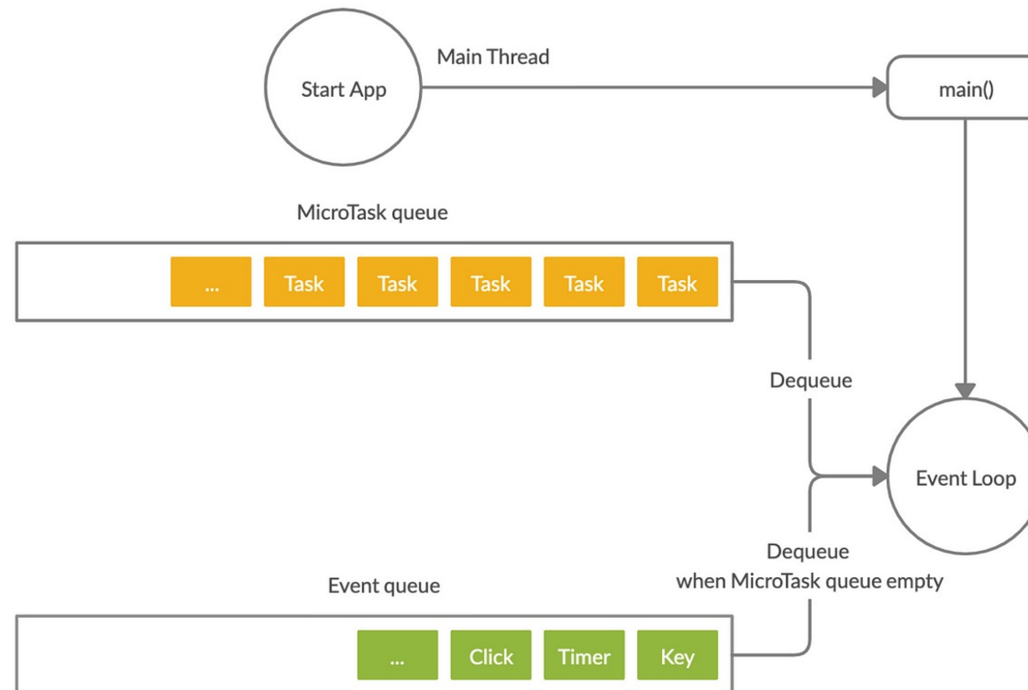
```
// new_expense.dart
void _presentDatePicker() async {
  final pickedDate = await showDatePicker(...);
  // executed later
  setState(() {
    _selectedDate = pickedDate;
  });
}
```

- `showDatePicker()` is an asynchronous function that returns **`Future`**`<DateTime?>`
  - A value that will be available in the future
- Handling:
  1. `async` + `await`
  2. `then()`

```
void _presentDatePicker() {
  showDatePicker().then((pickedDate) {
    // executed later
    setState(() {
      _selectedDate = pickedDate;
    });
  });
  // lines here executed immediately
}
```

Event Loop & Microtasks

- Flutter processes events and async functions (microtasks) using a ***single*** `main` thread
- Event loop:



- If a microtask takes too long, you UI janks!
  - Use [`Isolate`] to offload long task to another thread

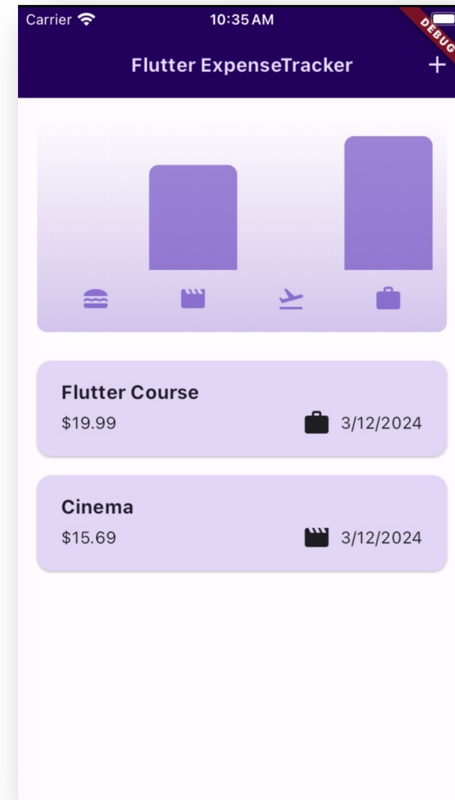
Async. Programming: `Stream`

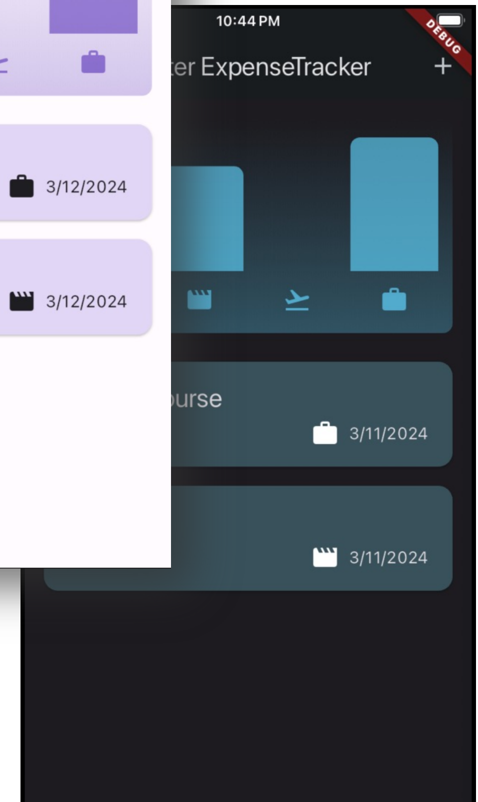
- A sequence of `Future` events
- Use `await for` to iterate over stream

```
Stream<int> intStream(int max) async* {
  for (int i = 1; i <= max; i++) {
    // Simulate some delay
    await Future.delayed(Duration(seconds: 1));
    yield i; // Emit an integer
  }
}

Future<void> listenToStream() async {
  Stream<int> stream = intStream(5);
  await for (int i in stream) {
    ... // Do something with each number
  }
  print('Stream completed');
}

void main() async {
  await listenToStream ();
}
```

Expense Tracker App

- ListView
- Navigator & modals
- Async programming
- User input & validation
- SnackBar
- Theming





Handling `TextField` Inputs

- Manual callback or `TextEditingCongtroller`

```dart
class _NewExpenseState extends State<NewExpense> {
  var _enteredTitle = '';

  void _saveTitleInput(String inputValue) {
    _enteredTitle = inputValue;
  }

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(16),
      child: Column(
        children: [
          TextField(
            onChanged: _saveTitleInput,
            maxLength: 50,
            decoration: const InputDecoration(
              label: Text('Title'),
            ),
          ),
          Row(
            children: [
              ElevatedButton(
                onPressed: () {
                  print(_enteredTitle);
                },
                child: const Text('Save Expense'),
              ),
            ],
          ),
        ],
      ),
```

```dart
12 class _NewExpenseState extends State<NewExpense> {
13+  final _titleController = TextEditingController();
14
15+  @override
16+  void dispose() {
17+    _titleController.dispose();
18+    super.dispose();
19   }
20
21   @override
22   Widget build(BuildContext context) {
23     return Padding(
24       padding: const EdgeInsets.all(16),
25       child: Column(
26         children: [
27           TextField(
28+            controller: _titleController,
29             maxLength: 50,
30             decoration: const InputDecoration(
31               label: Text('Title'),
32             ),
33           ),
34           Row(
35             children: [
36               ElevatedButton(
37                 onPressed: () {
38+                  print(_titleController.text);
39                 },
40                 child: const Text('Save Expense'),
41               ),
42             ],
43           ),
44         ],
45       ),
```

Stateful Widget Lifecycle

```
class TimerWidget extends StatefulWidget { ... }

class _TimerWidgetState extends State<TimerWidget> {
  int _counter = 0;
  late Timer _timer;

  @override
  void initState() { // Called when inserted into widget tree
    super.initState();
    _timer = Timer.periodic(Duration(seconds: 1), (timer) {
      setState(() { _counter++; });
    });
  }

  @override
  Widget build(BuildContext context) { ... }

  @override
  void dispose() { // Called when removed from widget tree
    _timer.cancel(); // Prevent memory leaks
    super.dispose();
  }
}
```

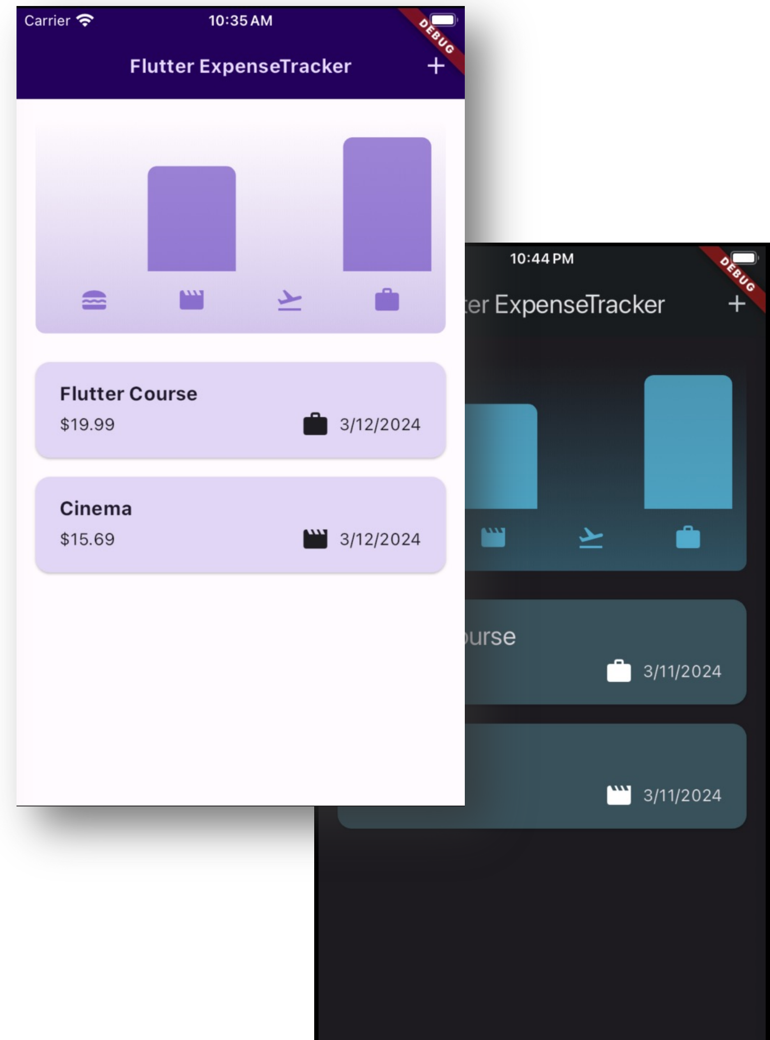
Input Validation

- Never trust input from user!
- Always validate user input
  - In `_submitExpenseData()` in `new_expense.dart`
  - Show `AlertDialog` if validation fails

- Alternatively, use [`Form` + `TextFormField` with `validator` property]()

Expense Tracker App

- `ListView`
- Navigator & modals
- Async programming
- User input & validation
- `SnackBar`
- Theming



Showing `SnackBar`

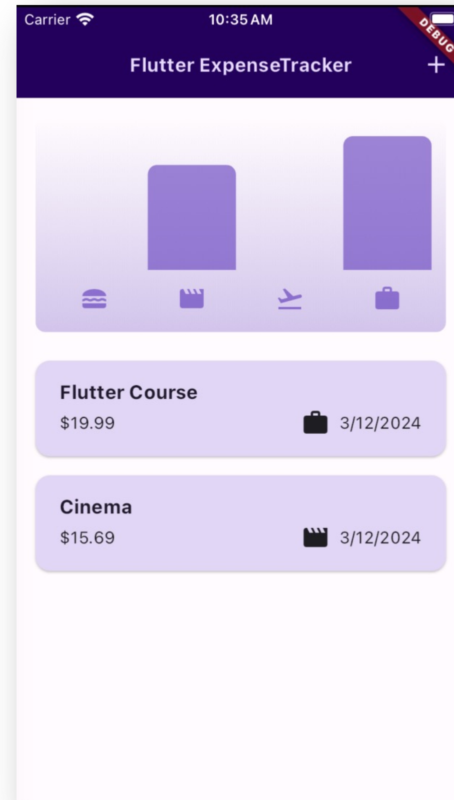
- `Dismissable` usually comes with `SnackBar` to allow action be undone

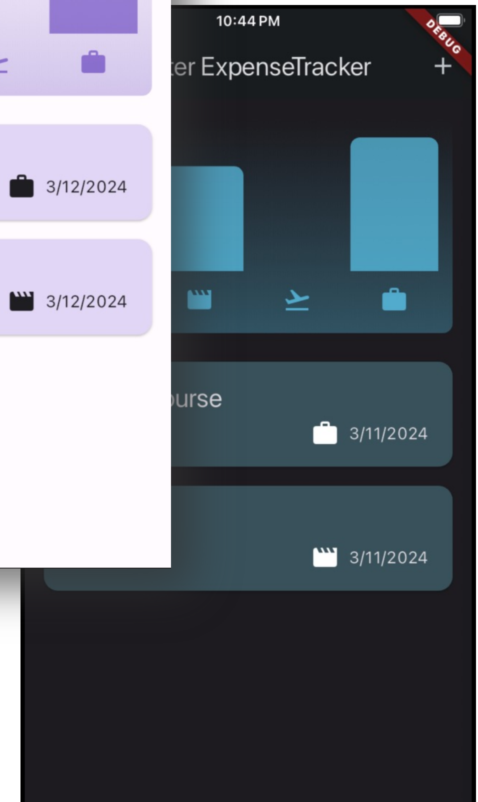
```
// expenses.dart
void _removeExpense(Expense expense) {
  ScaffoldMessenger.of(context).showSnackBar(...);
}
```

- `ScaffoldMessenger` keeps `SnackBar` displayed as user navigates away current screen

Expense Tracker App

- `ListView`
- Navigator & modals
- Async programming
- User input & validation
- `SnackBar`
- Theming





Theming

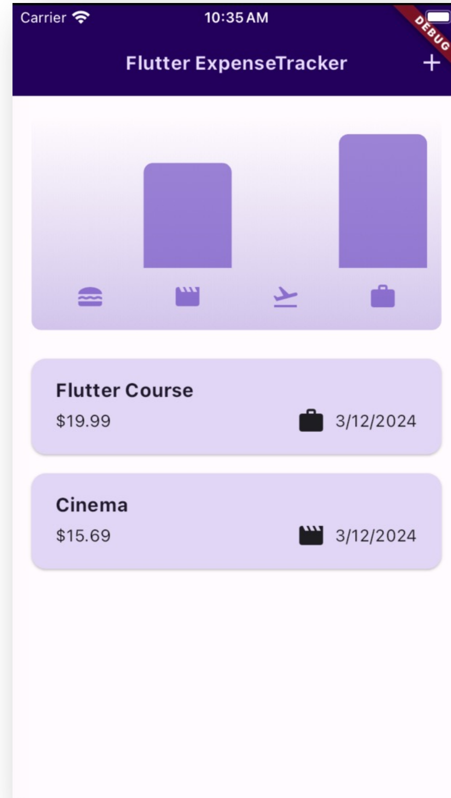
- Coherent way to customize visual aspects of app
  - Color scheme
  - Typography
  - Icons
  - Platform adaptation

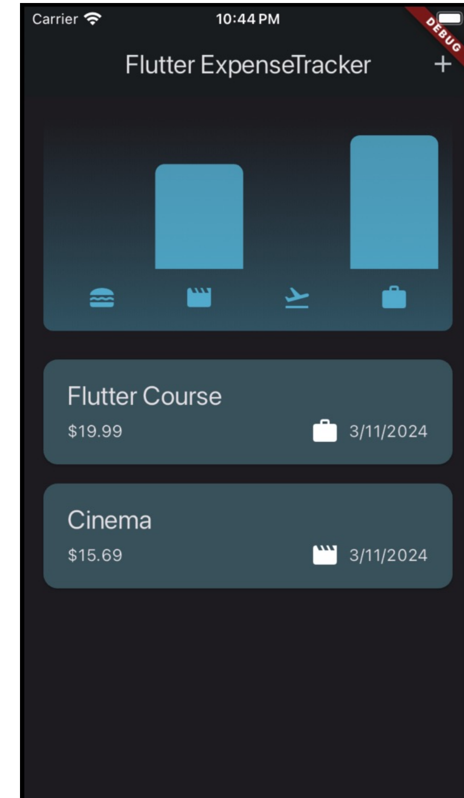
- Use `copyWith()` and `stytleForm()` to duplicate theme and style, respectively

Color Scheme

- Use Material Theme Builder to create your own
- Or obtained from single color via `ColorScheme.fromSeed()`
- Use `Theme.of(context).colorScheme` to accesse colors in your code

Color Mode





- Set mode via `themeMode` property of `MaterialApp`
- To determine light/dark mode:

```
final isDarkMode = MediaQuery.of(context).platformBrightness
                   == Brightness.dark;
```

Typography

- SF Font on iOS and Mac
- Roboto on other devices
- Accessed by `Theme.of(context).textTheme`

Display Large

Display Medium

Display Small

Headline Large

Headline Medium

Headline Small

Title Large

Title Medium

Title Small

Label Large

Label Medium

Label Small

Body Large

Body Medium

Body Small

Icons

- Use default icons: `Icon(Icons…)`
- Custom icons: https://www.fluttericon.com/

10k	10mp	11mp	123	12mp	13mp	14mp	15mp	16mp	17mp	18mp
19mp	1k	1k_plus	1x_mobiledata	20mp	21mp	22mp	23mp	24mp	2k	2k_plus
2mp	30fps	30fps_select	360	3d_rotation	3g_mobiledata	3k	3k_plus	3mp	3p	4g_mobiledata
4g_plus_mob...	4k	4k_plus	4mp	5g	5k	5k_plus	5mp	60fps	60fps_select	6_ft_apart
6k	6k_plus	6mp	7k	7k_plus	7mp	8k	8k_plus	8mp	9k	9k_plus
9mp	abc	ac_unit	access_alarm	access_alarms	access_time	access_time_...	accessibility	accessibility_...	accessible	accessible_fo...
account_bala...	account_bala...	account_box	account_circle	account_tree	ad_units	adb	add	add_a_photo	add_alarm	add_alert

Automatic Platform Adaptation

- Flutter automatically adjusts UI and behavior on different platforms

- Theming
- Platform-specific widgets
- Adaptive Constructors
  - E.g., `Icon.adaptive.share`, `AdaptiveDialog`
- Platform Checks
  - E.g., `Plaform.isIOS`, `Plaform.isAndroid`

Suggested Reading

- Widget Catelog
- Flutter Widgets of the Week
- Infinite Scrolling
- Concurrency & Isolates
- Forms & Validation
- Material Theme Builder
- Automatic Platform Adaptation