



Mobile Apps 2: Screens, Navigation and State

How users move through an app, and how an app remembers and reacts

NAME _____

DATE _____

CLASS _____

Screens and Navigation

A mobile app is built from **screens** (also called **views**). Each screen has one job, like showing a list or editing one item. **Navigation** is how the user moves between screens. Three patterns are used in almost every app, on iOS and Android and in tools like SwiftUI, Kotlin, Flutter, React Native, App Inventor and Thunkable:

Tab bar

A row of 3 to 5 buttons, usually at the bottom, that switches between the app's main sections. It stays visible, so the user can jump to any section at any time.

Stack (back) navigation

Tapping an item **pushes** a new screen on top, like adding a card to a pile. **Back pops** it off and returns to the screen underneath.

Modal

A temporary screen that slides over the current one for a short, focused task. The user must finish it (Save, Done) or cancel it before returning.

State, Events and the Lifecycle

State is the data an app is remembering right now, such as a score, the items in a cart, or whether a heart is liked. The screen should always show the current state. An **event** is something that happens, like a tap. An **event handler** is the code that runs when that event happens. A good handler follows a pattern: **update the state, then update the UI** (user interface) so the screen matches the new state.

The operating system (OS) controls each app's **lifecycle**. An app is in the **foreground** when it is on screen and the user is using it. When the user switches apps or locks the phone, the app moves to the **background**, where it gets little or no time to run, and is soon **suspended** (kept in memory but not running). To free memory, the OS may stop a background or suspended app completely, without warning. When that happens, any state that was not saved is lost. So apps should **save important data when they move to the background**, not only when the user taps Save.

1 Pick the Pattern

For each situation, write the navigation pattern that fits best: **tab bar**, **stack**, or **modal**.

Situation	Pattern
a. Switching between the Home, Search and Profile sections of a social app	
b. Tapping an email in the inbox to read it, then going back to the inbox	
c. Writing a new post that the user will either Post or Cancel	
d. Tapping "Notifications" in Settings to see the notification options	

**Mobile Apps 2: Screens, Navigation and State**

(page 2)

2 Map the Screen Flow

PantryPal is a recipe app. Draw its **screen flow diagram**: one box per screen, and an arrow for each move labeled with the tap and the pattern (tab, push, back or modal).

- A tab bar has three tabs: Recipes, Shopping List, Profile.
- Recipes shows a list. Tapping a recipe pushes Recipe Detail.
- Recipe Detail has an "Add to List" button that opens a modal, Choose Ingredients, with Cancel and Add buttons. Both close it.
- Profile has a "Settings" row that pushes the Settings screen.

a. On Recipe Detail, the user taps Back. Which screen appears?

c. Why is Choose Ingredients a modal instead of a fourth tab? Use the definitions from page 1.

b. On Choose Ingredients, the user taps Cancel. Which screen appears?

3 Lost Work

A notes app saves a note only when the user taps **Save**. Jordan types a long note, switches to the camera app for twenty minutes, then reopens the notes app. The app starts fresh and the note is gone.

a. Use the lifecycle terms from page 1 to explain what most likely happened to the app.

b. Describe a fix. When should the app save, and what event should trigger it?



Mobile Apps 2: Screens, Navigation and State

(page 3)

4 Trace the State

This pseudocode runs a counter screen. Each handler updates the state, then calls `render()`.

```
1 count = 0
2 liked = false
3
4 onTap(plusButton):
5     count = count + 1
6     render()
7
8 onTap(likeButton):
9     liked = not liked
10    render()
11
12 onTap(resetButton):
13     count = 0
14     render()
15
16 render():
17     label shows "Count: " + count
18     heart: filled if liked, else outline
```

Fill in the table after each tap, in order.

Tap	count	liked	Label shows	Heart
(start)	0	false	Count: 0	outline
a. plus				
b. plus				
c. like				
d. plus				
e. reset				
f. like				

5 Find the Bug

A teammate adds a minus button. Starting from `count = 0`, a user taps **plus** once, then **minus** once.

```
1 onTap(minusButton):
2     count = count - 1
```

a. After both taps, what is the value of `count`, and what does the label show?

count = _____

Label shows: _____

b. Explain the bug using the words **state** and **UI**.

c. Rewrite the minus handler so it fixes the bug **and** never lets `count` go below 0.
