Crafting Responsive and Playful Interfaces

Adam Bell - @b3ll

Who am I?

- ya tu sabes, but now, Dynamic™
- @b3ll
- Software Engineer 🇨🇦
- Spend a lot of my personal time reverse engineering iOS / macOS
- I also enjoy making animations / interactions, synthesizers, and music-related apps
- Currently working at Netflix on the iOS UI team

What is a "responsive" interface?

What is a "playful" interface?

Responsive > [...] reacting quickly and positively.

— Oxford Dictionary

Responsiveness

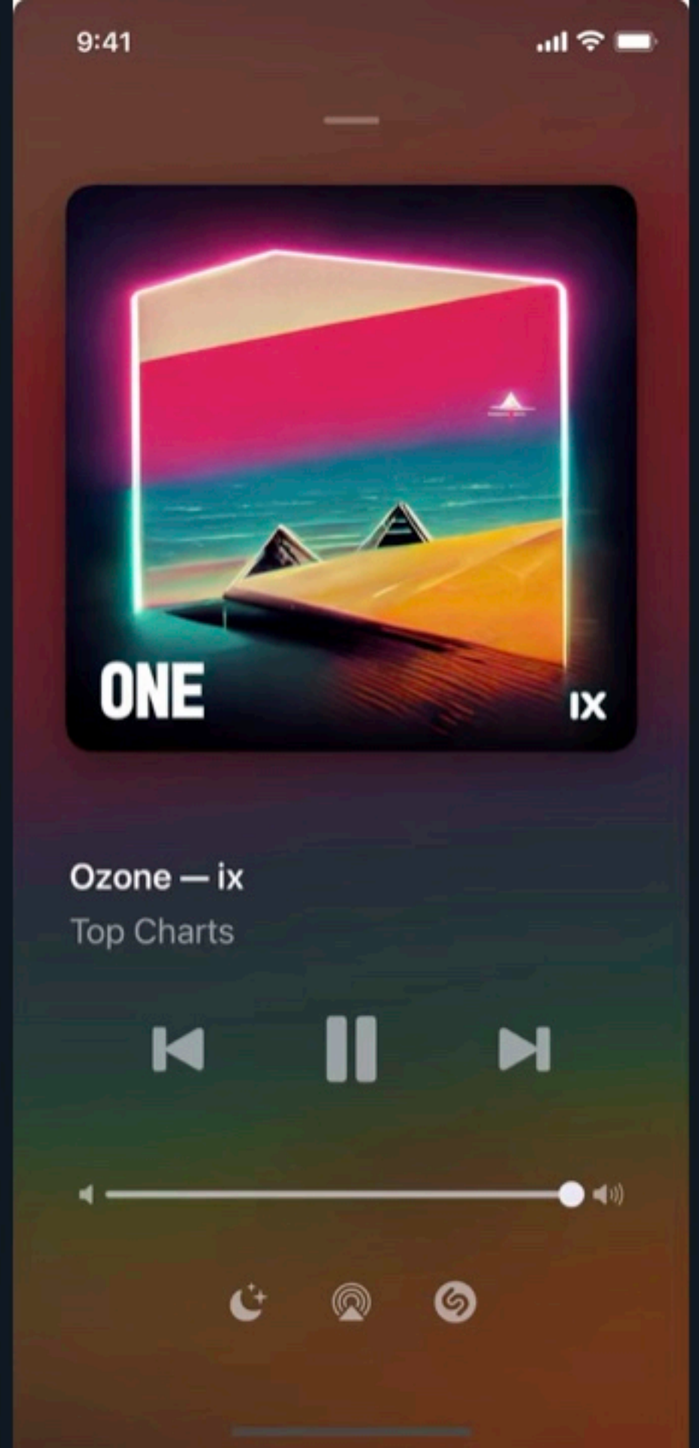
Taking that literally, a responsive app:

- Reacts to your touches
- Does so in a quick and / or positive manner
- Has animations that effectively communicate where you're going / how you got there

Broadcasts

by @stroughtonsmith

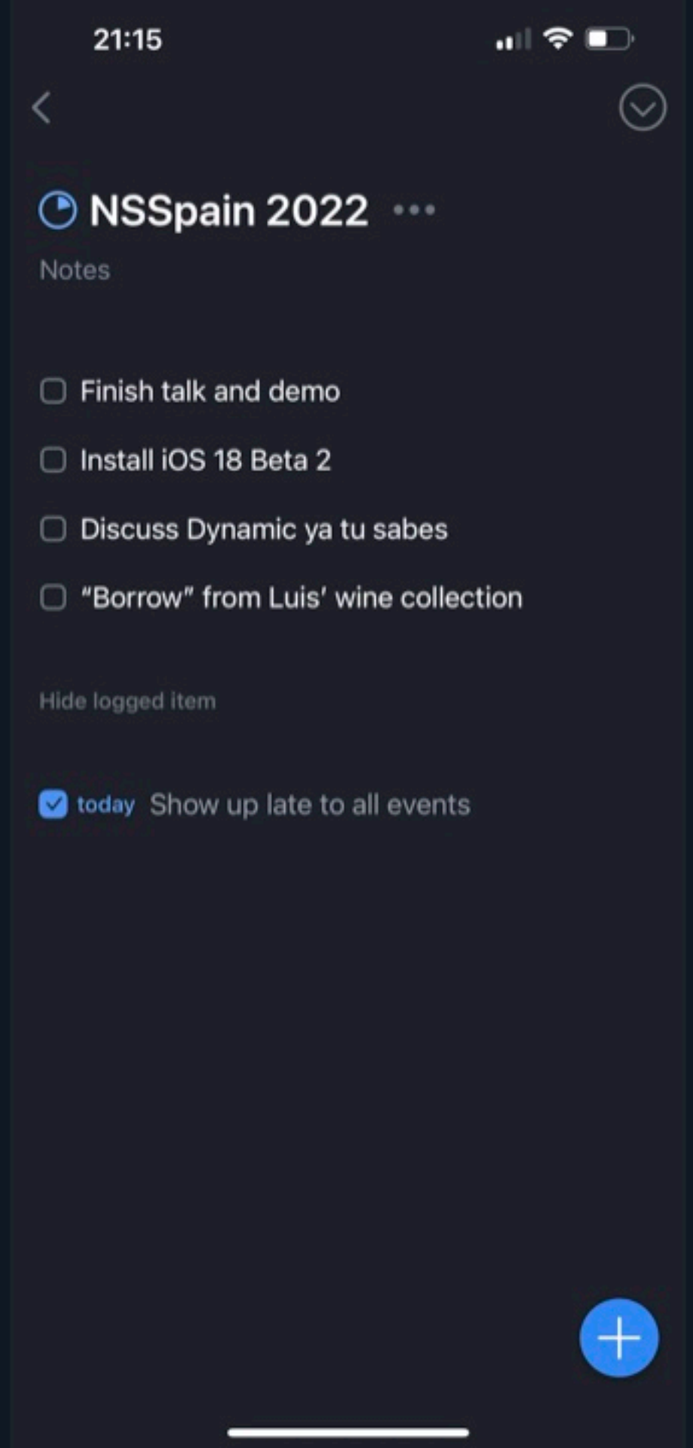
- Reacts to touches and updates state quickly
- Compliments your touches in a positive manner with a nice crossfade transition



Things

by Cultured Code

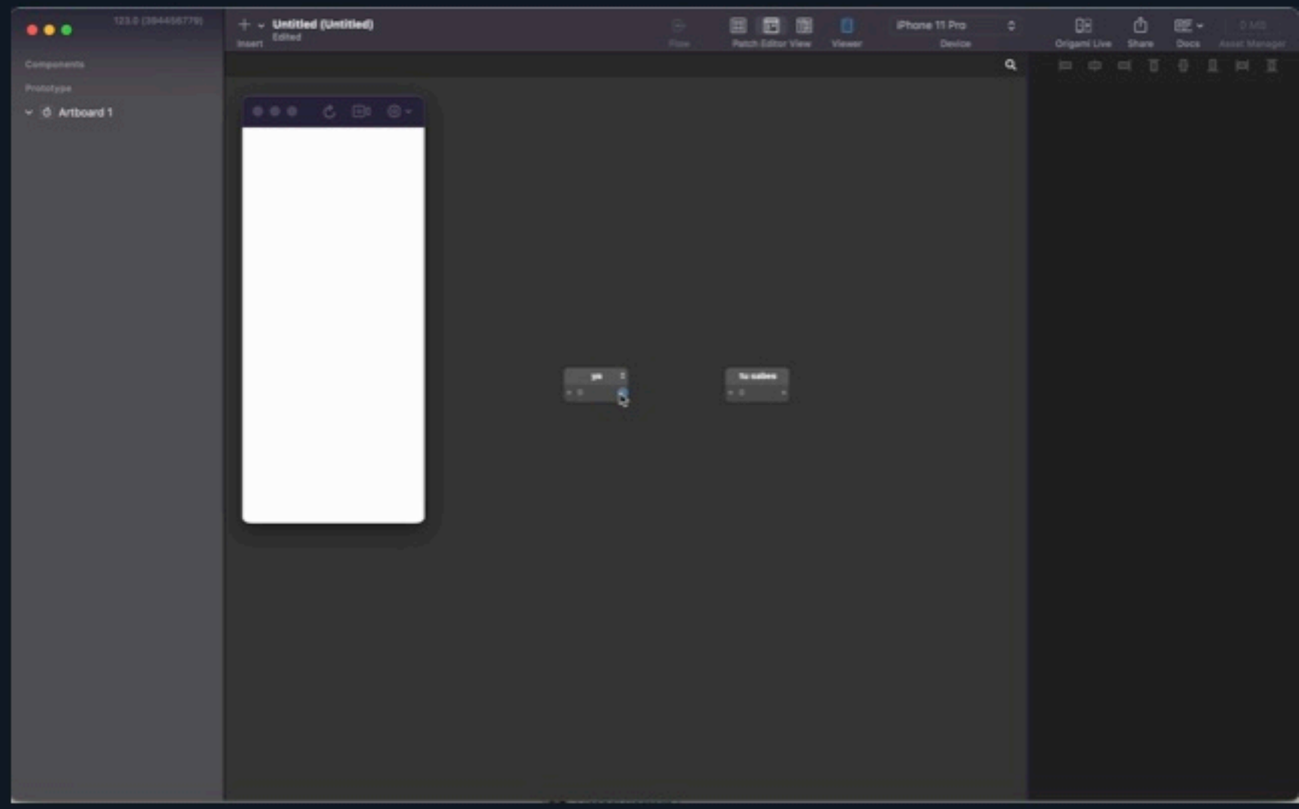
- Everything you tap on reacts to your touches immediately
  - percent completion of tasks
  - the task itself
  - completed tasks are moved shortly after
- Opening interactions are quick to compliment your actions
- Everything is interruptible



Origami Studio: Al Dente Noodles

by me!

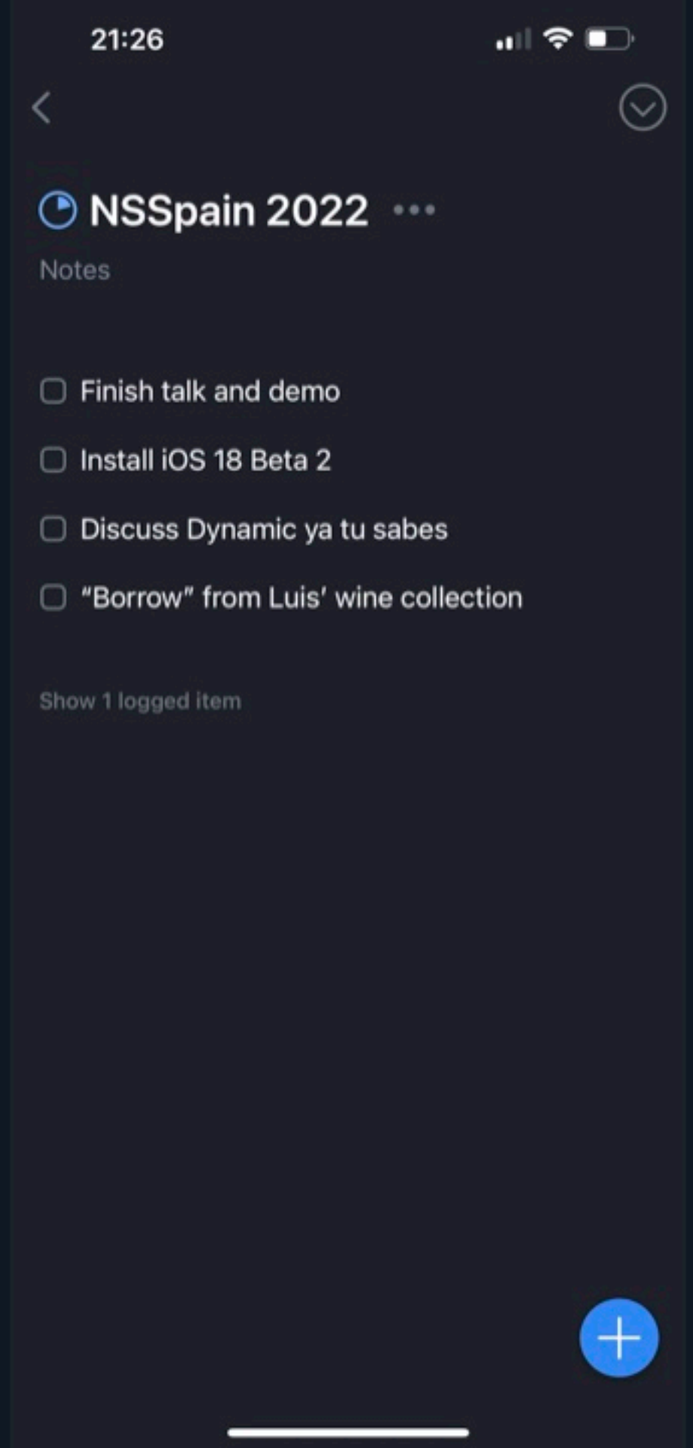
- Dragging lines to wire patches in Origami Studio is something you're doing all the time; calling them noodles instead of lines is just way more fun
- Making them all springy and stretchy added some delight to allow for playful creative prototyping, and then calling the noodles "al dente" was just the cherry on top



SpringBoard.app

Designed by Apple in California

- App dismissal compliment your gestures positively
- They seed the dismissing animation with velocity
- They communicate where you're going to quickly



Animations are Tools for Communication

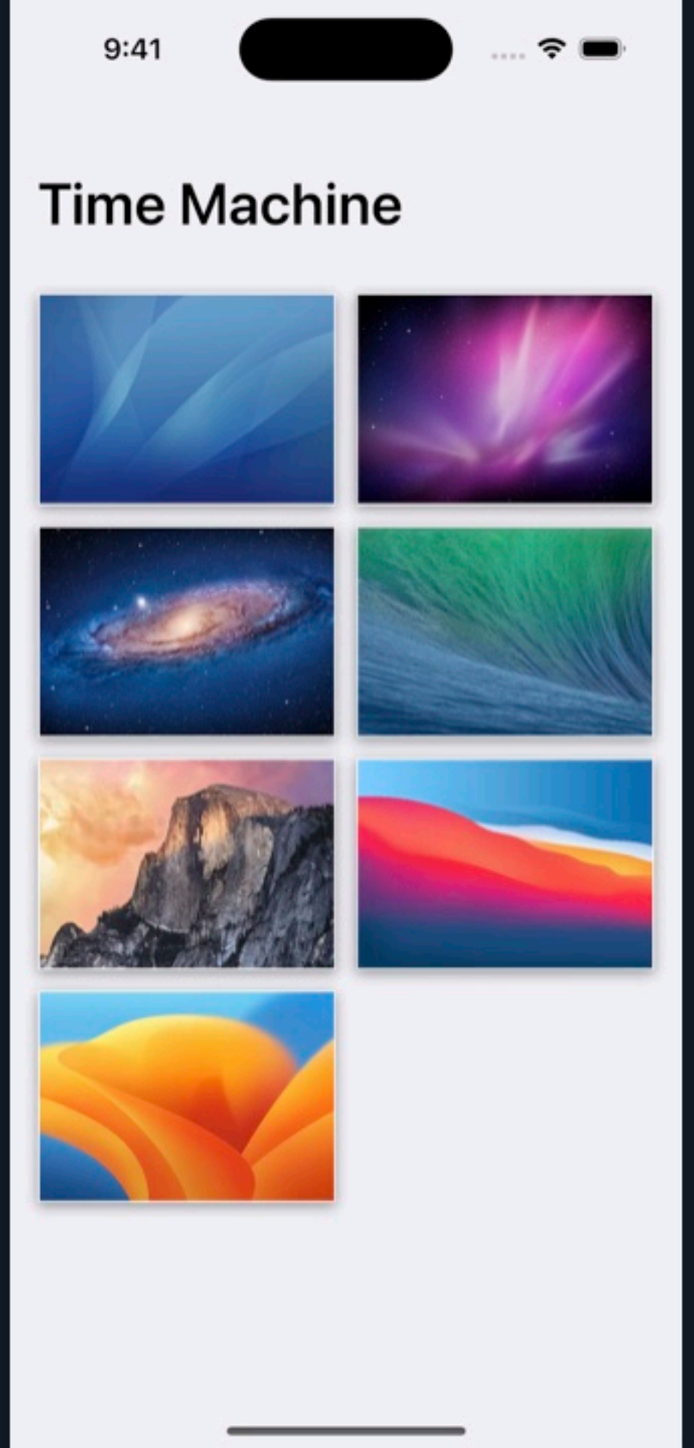
- All of these apps use animation as a means to compliment interactions and communicate where you're going / how you got there
- The animations never halt movement in or usage of the app
- They all respect your input and often act as an extension to any input or gestures you perform

Time Machine

by Adam Bell

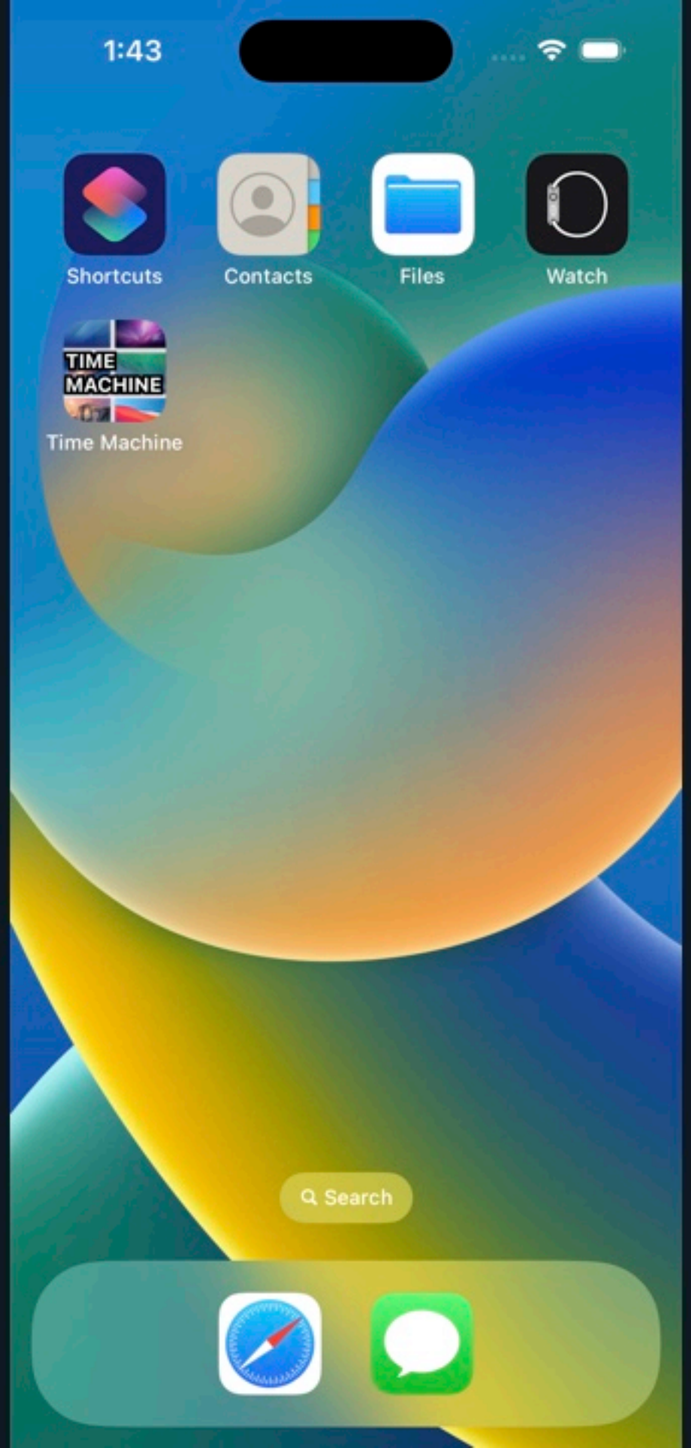
Time Machine

by Adam Bell




dies


TIME
MACHINE


1:43
Shortcuts
Contacts
Files
Watch
TIME MACHINE
Time Machine
Search

Michael Flarup

@flarup

That icon looks fantastic! Couldn't have done it better myself

September 16, 2022, 12:41 PM

Michael Flarup

@flarup

I made some alternates for you. Let me know what you think!

Yesterday, 2:25 PM

Anyways

Time Machine: Things to Fix

There's many issues with this to make it feel like a responsive application, including:

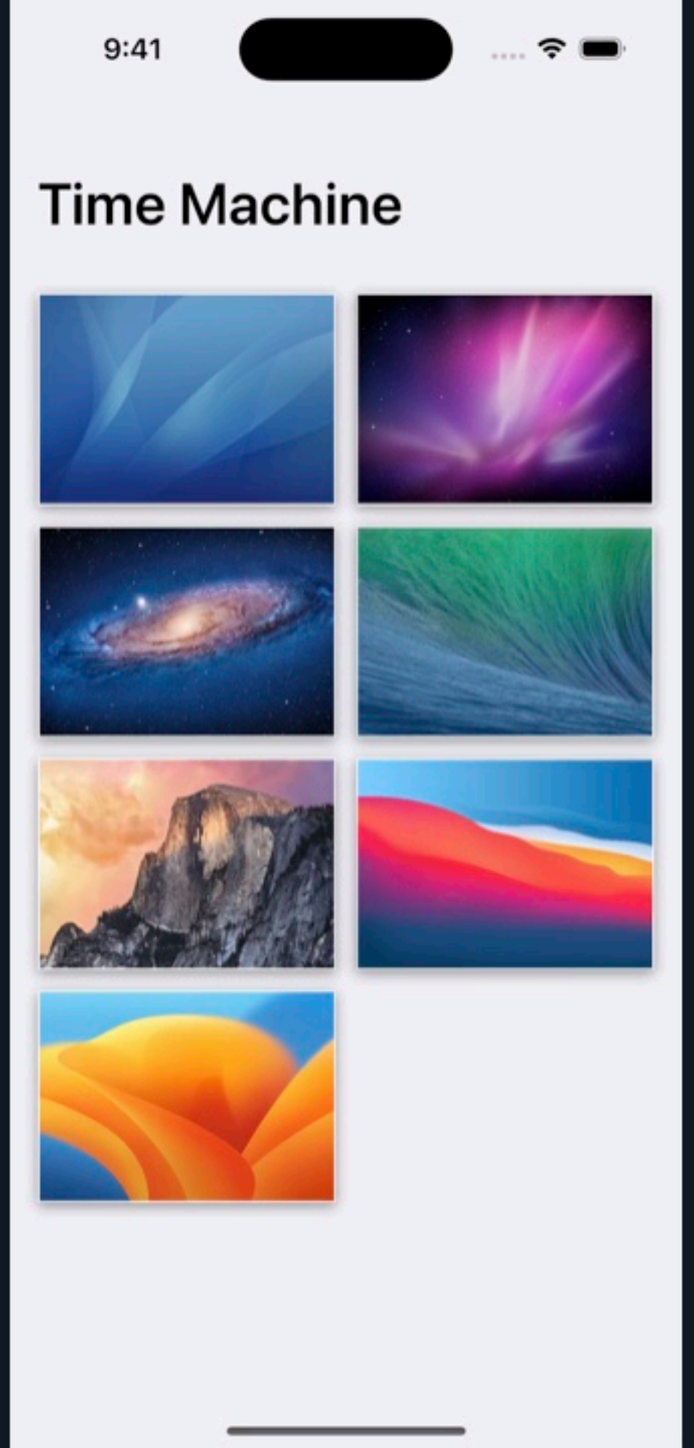
1. Slow and Broken Presentation Animation
2. Lack of Touch Acknowledgement
3. Animations Not Complimenting Gestures

Time Machine: Things to Fix

1. **Slow and Broken Presentation Animation** 🔨
2. Lack of Touch Acknowledgement
3. Animations Not Complimenting Gestures

Slow Presentation Animation: Blocking Input

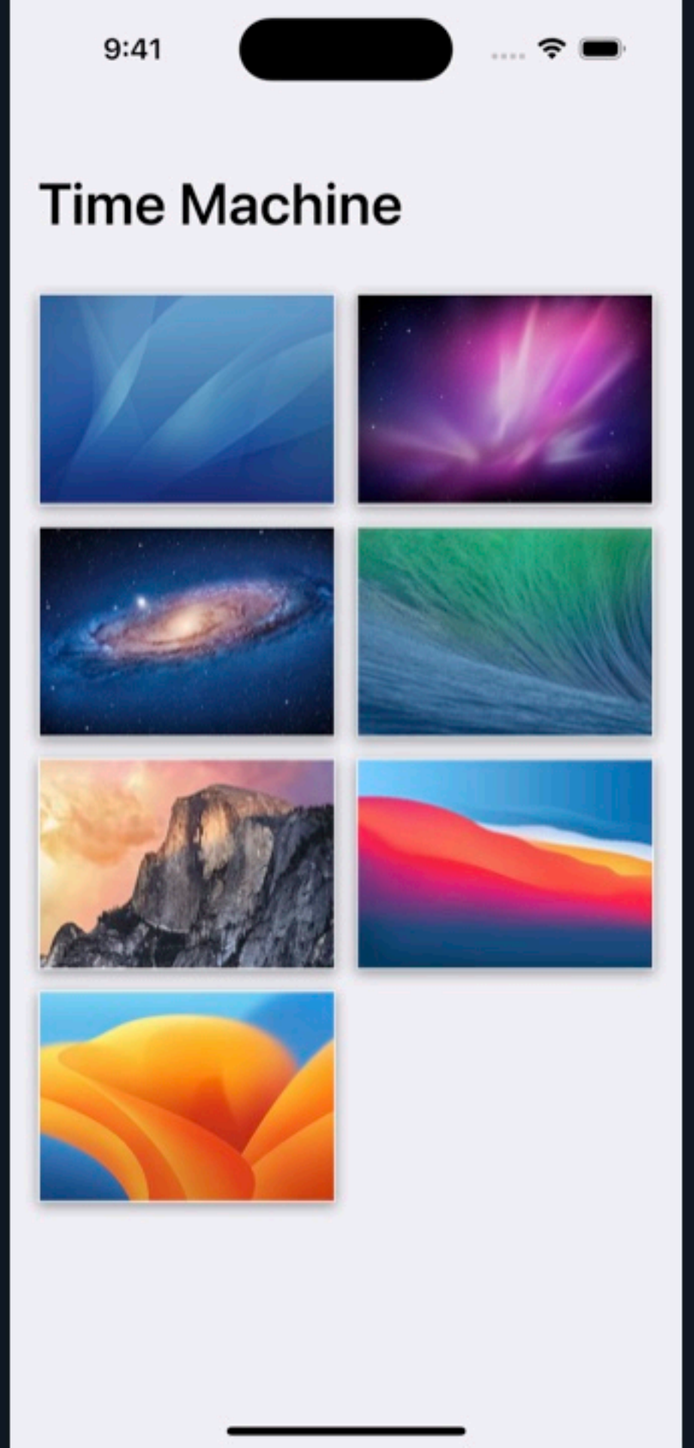
- In this app, all of my touches are ignored and the entire app is unresponsive while a wallpaper is loading
- This is a bad experience as I'm now pretty much stuck and have to either wait, or quit the app
- It's not responsive, at all



Slow Presentation Animation: Blocking Input

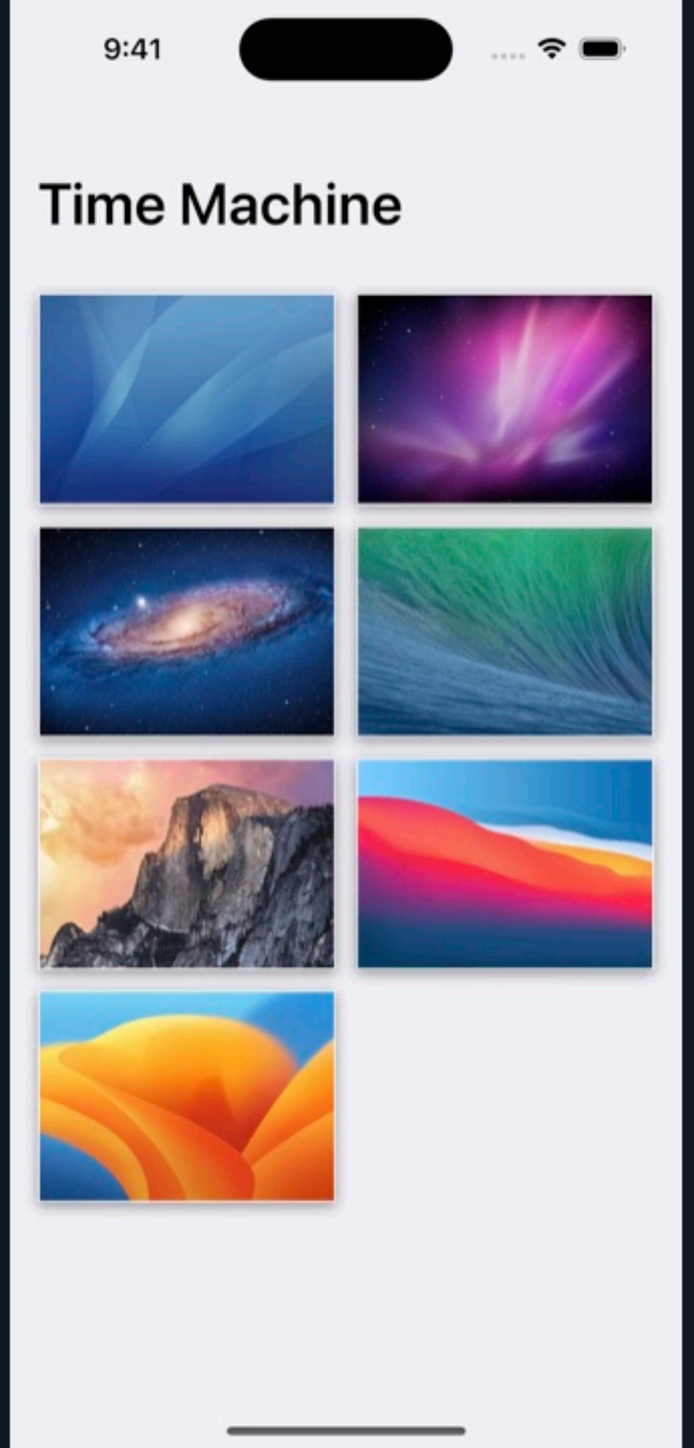
Fixed ✅

- Offload any and all network bound loading to be done asynchronously as the view controller is presenting
- While anything is loading **do not** disable user interaction
- While it may take the same amount of time to load, the experience overall feels like the app is still responsive



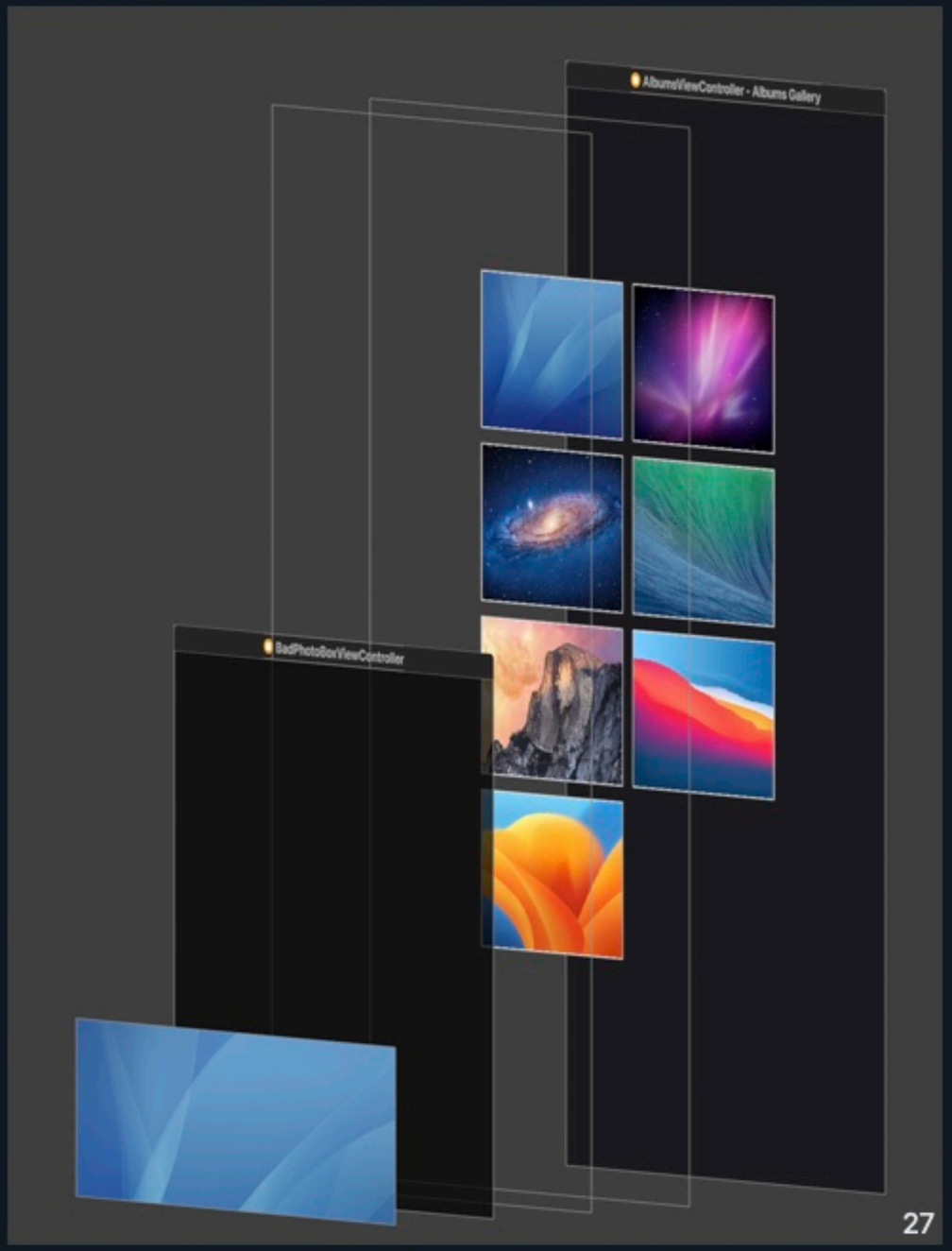
Broken Presentation Animation: Dimming

- If you watch closely, you'll see the dimming view of the wallpaper being presented is being presented along with the wallpaper
- Presenting the dimming view from the bottom of the screen doesn't make any sense
  - Shadows in the real world don't work this way
  - It makes your app feel broken



Broken Presentation Animation: Dimming

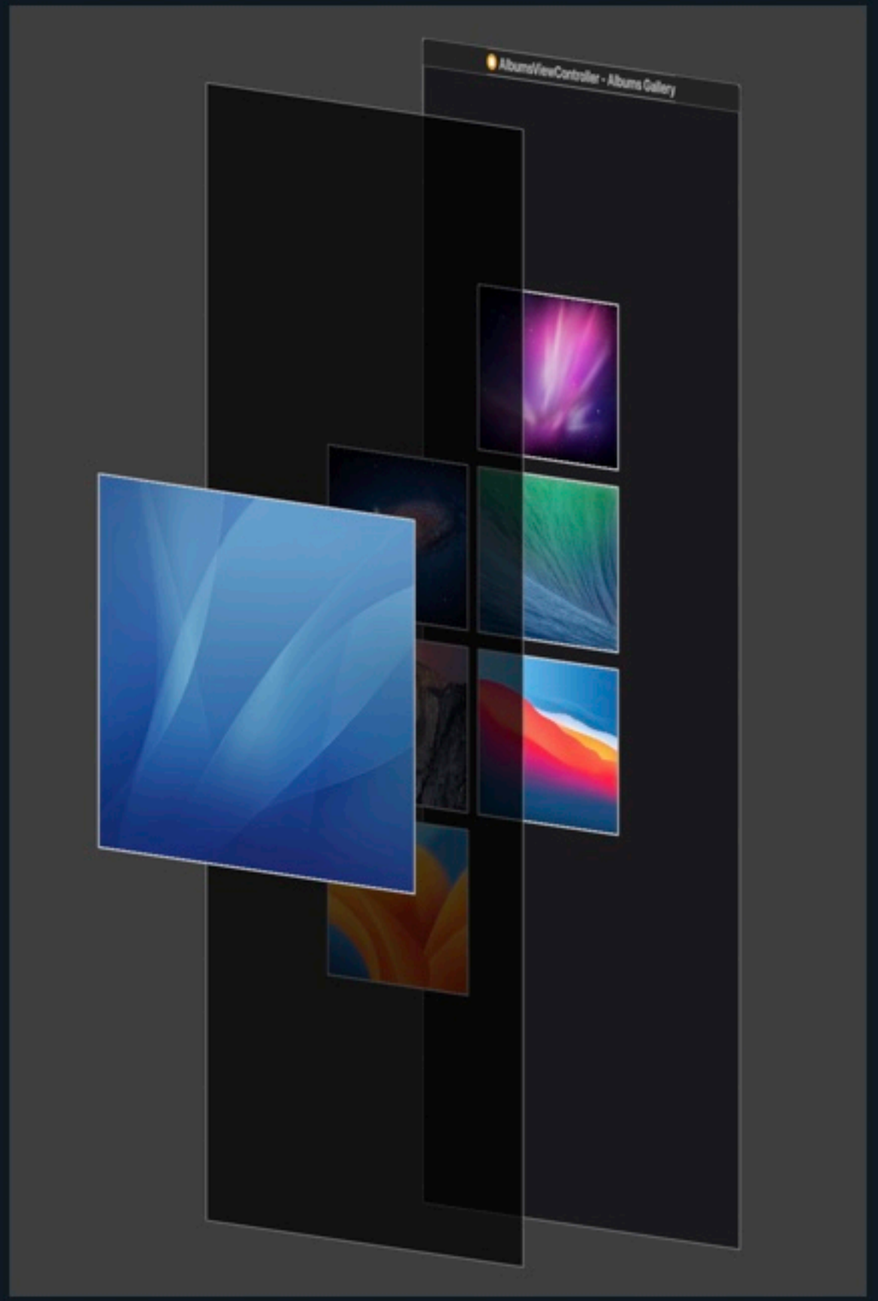
- When you present something, it should cast a shadow onto the contents below
- Here, the dimming view is attached to the same view controller being presented, if you present the whole view controller, the dimming view gets presented along with it



Broken Presentation Animation: Dimming

Fixed ✅

- Decouple dimming views from the presented view controller
- `UIPresentationController` can help with this
- They should fade in as the presenting view occludes the content behind
- This creates depth and gives hints as to how it can be dismissed, making your app more accessible to more people



Broken Presentation Animation: Dimming

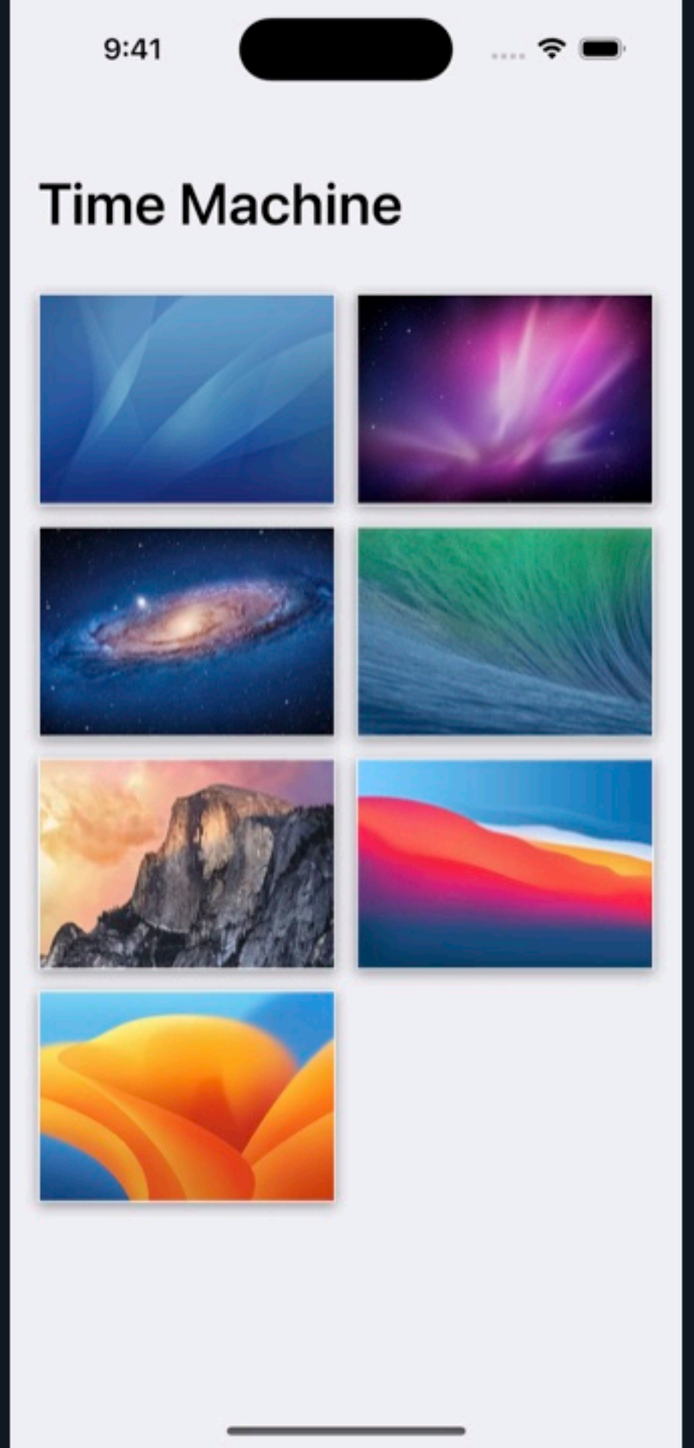
Fixed ✅

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)

    if dimmingView.superview != presentationController?.containerView {
        presentationController?.containerView?.addSubview(dimmingView)
    }

    transitionCoordinator?.animate { [weak self] _ in
        self?.dimmingView.alpha = 1.0
    }
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)

    transitionCoordinator?.animate { [weak self] _ in
        self?.dimmingView.alpha = 0.0
    }
}
```

Broken Presentation Animation: Dimming

Fixed ✅

- The dimming view now fades in along with the content being presented
- It makes a lot more sense from a mental model perspective



Time Machine: Things to Fix

1. **Slow and Broken Presentation Animation** 🔨
2. Lack of Touch Acknowledgement
3. Animations Not Complimenting Gestures

Time Machine: Things to Fix

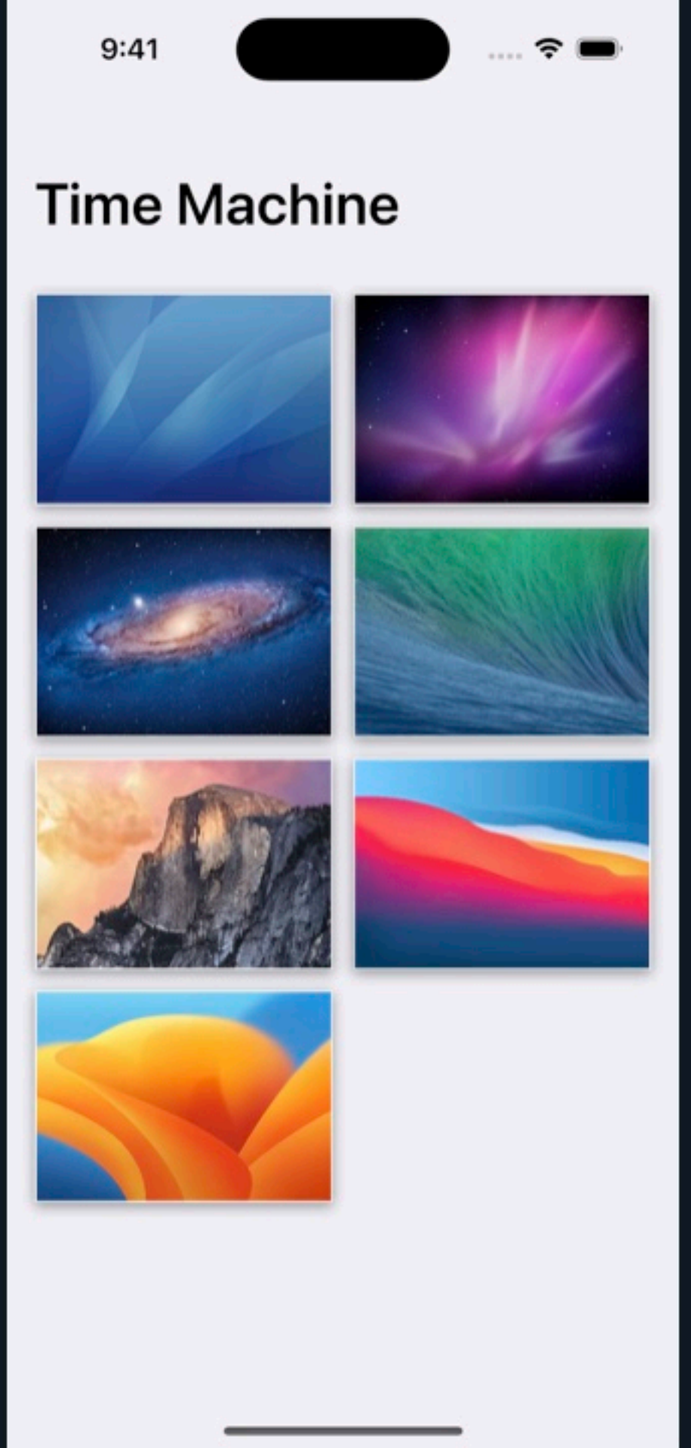
1. ~~Slow and Broken Presentation Animation~~ ✅
2. **Lack of Touch Acknowledgement** 🔨
3. Animations Not Complimenting Gestures

Touch Acknowledgement

- Anytime you touch something interactive, it should react to indicate it's interactive
- It needs to acknowledge your input
- This happens in the real world too: "every action has an equal and opposite reaction"

Touch Acknowledgement: Unresponsiveness

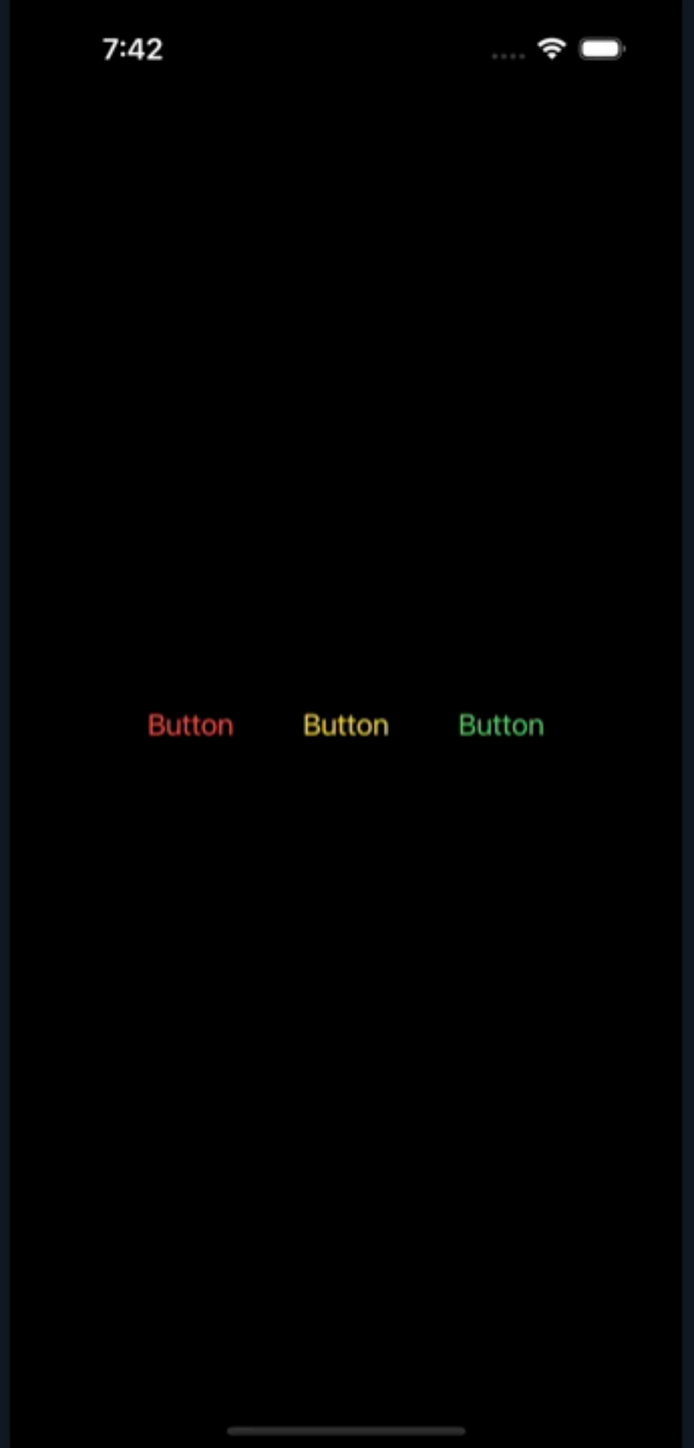
- When tapping on wallpapers, there's no indication that a tap occurred
- Without any sort of response to touches, there's no indication it's doing anything, especially with poor networks
- This could lead to frustration, or to several taps happening on the same thing and a really poor experience
- "Did I actually tap it?"
- "Did I tap the right one?"



So how can we fix this?

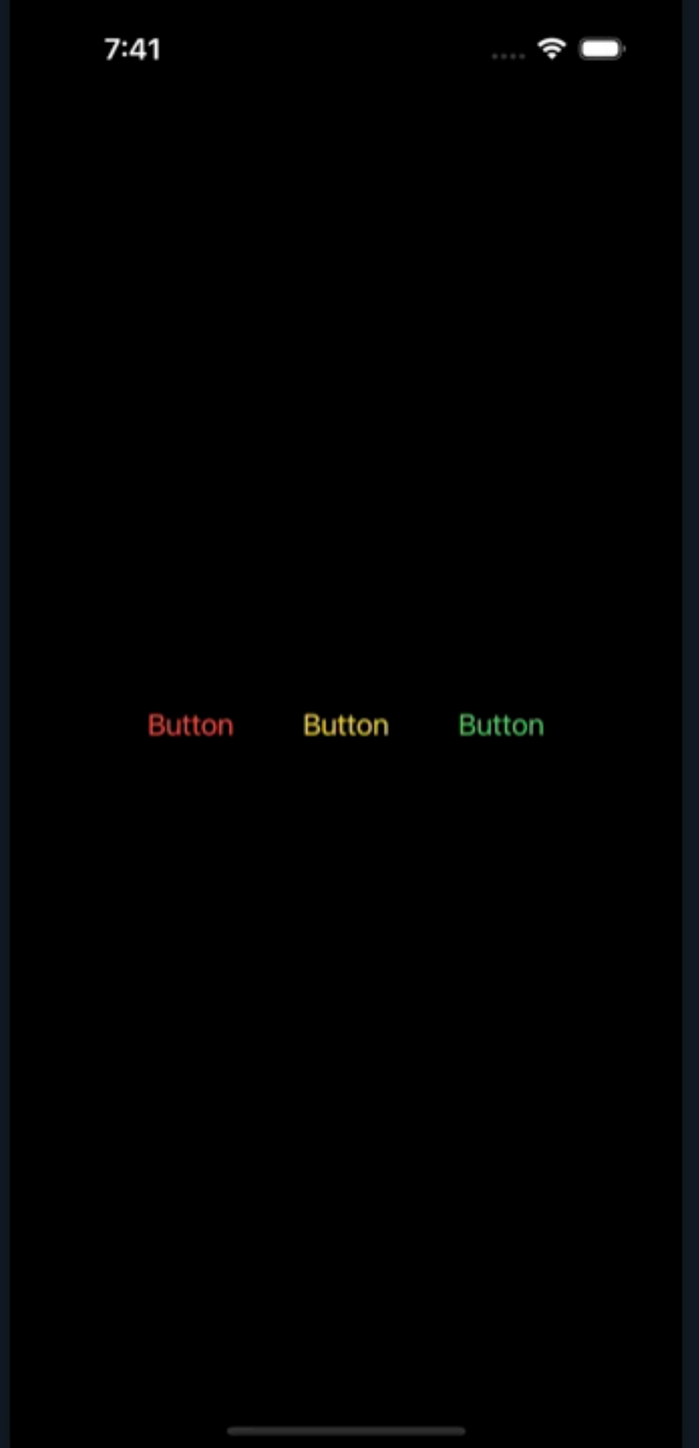
`view.alpha = 0.5?`

- It's pretty common practice to change the alpha of a button when you touch it, as a reaction to indicate you tapped on it



~~`view.alpha = 0.5`~~

- However, this fails when your finger covers up the button
- Since you can't see the button respond when you tap it, it makes it feel unresponsive
- Even if the button does change, if it's too quick or small to see, it won't be perceived as acknowledging your touch



What else can we do?

We can learn from synthesizers!

Synthesizers

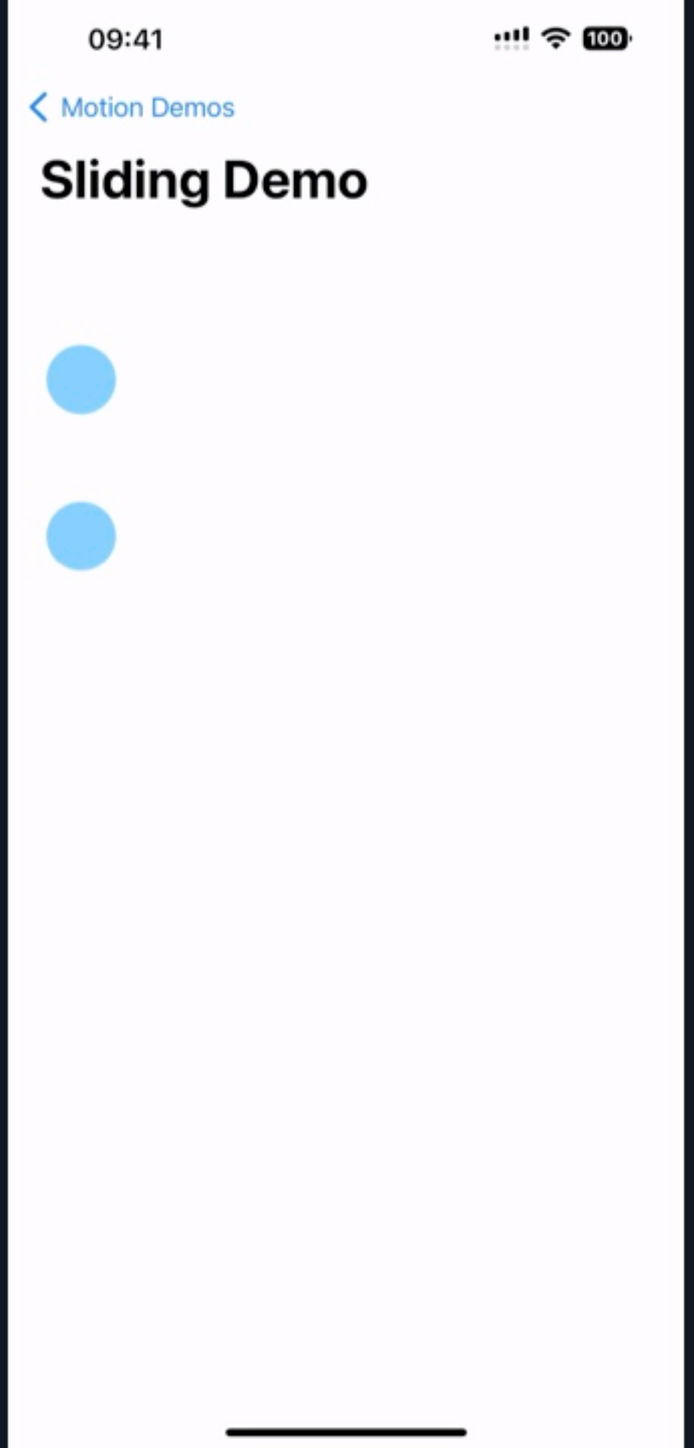
- Synthesizers have tons of buttons and keys, all with different shapes, sizes, and reactions to your touch
- They have tactility that respond to and acknowledge your touches

Mechanical Keyboards

- If you've ever wondered why people are so obsessed with mechanical keyboards...
- ...it's because of the tactility of their keys!
- Most have a **spring** inside that compresses / rebounds to your press, giving you feedback and acknowledging your input

Springs

- Springs are a great way to give feedback / acknowledge touches as they literally rebound with an "opposite and equal reaction" from whatever input they're given
- They're feel naturally responsive
- When you make a spring animation, you have something pulled towards a location (or value) in a way that would feel a lot more natural than simply a timing curve



Bouncy Buttons

- So now that we have the tools, we can add springs to our buttons to make them feel super responsive
- I like to call it a "bouncy behavior" as it sorta looks like it's bouncing on the screen
- You push down, and then it "bounces" back up, just like a keyboard key

Bouncy Buttons

```
class BouncyLabel: UILabel {

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        super.touchesBegan(touches, with: event)

        UIView.animate(withDuration: ???,
                       delay: 0.0,
                       usingSpringWithDamping: 0.8,
                       initialSpringVelocity: ???) {
            self?.layer.transform = CATransform3DMakeScale(0.8, 0.8, 1.0)
            self?.alpha = 0.8
        }

    }

    override func touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent?) {
        super.touchesCancelled(touches, with: event)

        // ???
    }

    override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
        super.touchesEnded(touches, with: event)

        // ???
    }

}
```

UIView's Spring Animation

- `duration` is really hard to guess outright
- There's no way to redirect it smoothly (i.e. have it pop back up)
    - Doing another UIView Animation (even with `beginFromCurrentState`) will just start a new discontinuous animation
    - Supplying `initialVelocity` requires custom math as it wants velocity in a unit vector, which is also awkward to compute

`CASpringAnimation`?

- Unfortunately suffers from the same issues as UIView's spring animations
  - No way to rebound, starting a new animation is still discontinuous
- However, `velocity` is in points per second (exposed by gesture recognizers), which is nicer than `initialVelocity`'s unit vector requirements

... if only there was an open source library for making gesturally-driven interactions and animations

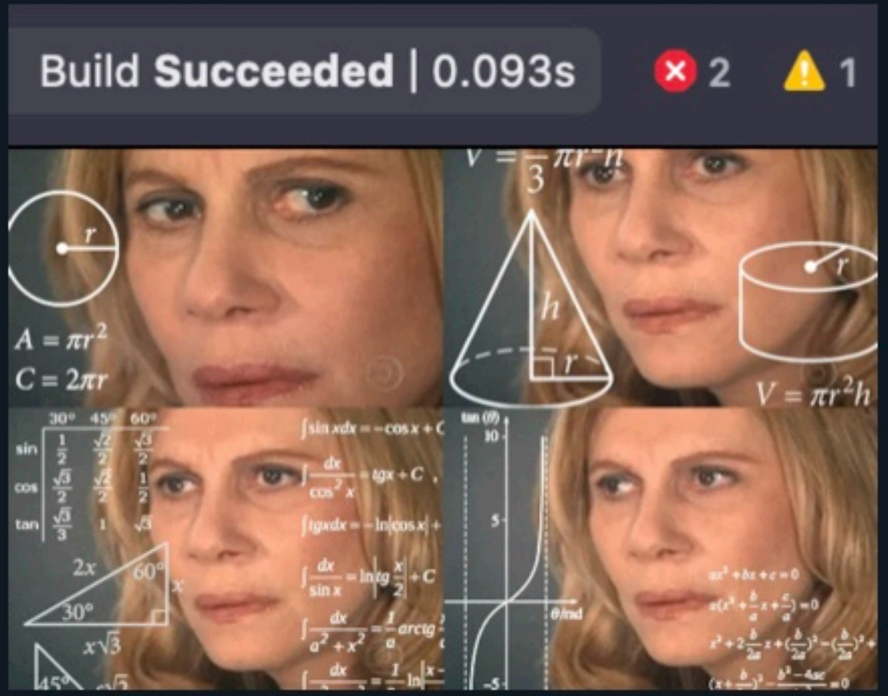

M TION

Motion

- Motion is an animation engine I wrote for building gesturally-driven user interfaces, animations, and interactions
- It does all of the animations **in-process** (think `CADisplayLink`)
- It allows for easily creating physically-modeled, interruptible animations (i.e. springs, decays, etc.) that work hand-in-hand with gesture recognizers to make really responsive interactions
- It's powered by SIMD

wait, what's SIMD?

SIMD Segue



Motion: SIMD

- Single Instruction, Multiple Data
- SIMD effectively lets you pack lots of values into a single register and do math on all the values simultaneously (instead of doing math for each value, sequentially)
- So all the animations done by Motion (be it on a `CGFloat` or a `CGRect` all animate equally as efficiently)

Motion: SIMD

For example, let's say you wanted to scale a `CGRect`

```
var rect = CGRect(x: 0.0, y: 0.0, width: 100.0, height: 200.0)
rect.origin.x    *= 2.0    // 4 "instructions" are done here
rect.origin.y    *= 2.0
rect.size.width  *= 2.0
rect.size.height *= 2.0
```

Motion: SIMD

Using SIMD, the multiplication is done on all values simultaneously.

When used properly, you can get some pretty large performance wins.

```
var simdRect = SIMD4(0.0, 0.0, 100.0, 200.0)
simdRect *= 2.0 // only 1 "instruction" is done here

// all 4 values are scaled at the same time

let rect = CGRect(x: simdRect[0], y: simdRect[1], width: simdRect[2], height: simdRect[3])
```

Motion: SIMD

- It can even do something like 5,000 spring animations of `SIMD64<Double>` in like `~130ms`
- That's **320,000** springs!!
- This means it's super efficient for running every frame, especially with Pro Motion at 120hz

tldr: Motion's animation code is ¡muy rápido!

Anyways

Bouncy Label

```
class BouncyLabel: UILabel {

    lazy private var bouncyAnimation = {
        let bouncyAnimation = SpringAnimation<CGPoint>(initialValue: CGPoint(x: 1.0, y: 1.0), response: 0.3, dampingRatio: 0.8)
        bouncyAnimation.onValueChanged(disableActions: true) { [weak self] newScale in
            self?.layer.scale = newScale
            self?.alpha = newScale.x
        }
        return bouncyAnimation
    }()

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        super.touchesBegan(touches, with: event)

        bouncyAnimation.velocity = CGPoint(x: -5.0, y: -5.0)
        bouncyAnimation.toValue = CGPoint(x: 0.8, y: 0.8)
        bouncyAnimation.start()
    }

    override func touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent?) {
        super.touchesCancelled(touches, with: event)

        bouncyAnimation.toValue = CGPoint(x: 1.0, y: 1.0)
        bouncyAnimation.start()
    }

    override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
        super.touchesEnded(touches, with: event)

        bouncyAnimation.toValue = CGPoint(x: 1.0, y: 1.0)
        bouncyAnimation.start()
    }

}
```

Motion: Some Notes on `SpringAnimation`

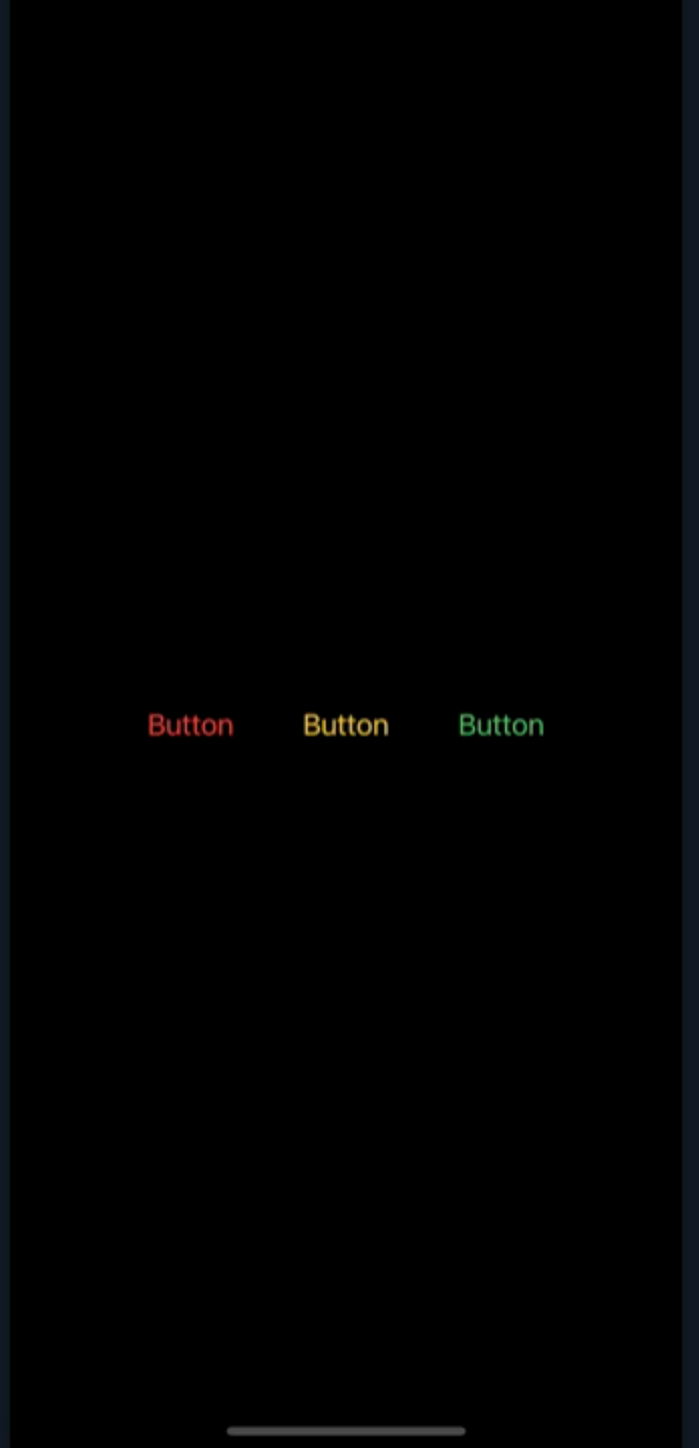
- **response**: how long it takes for the spring to reach its destination (in seconds)
- **dampingRatio**: how much it should bounce around the destination (0 means bounce forever)
- `start()` doesn't really mean "start a new animation", but means "continue or start again if needed"

What about `UIViewPropertyAnimator`?

- Totally valid API, and sometimes it's a great alternative to something like Motion, with some caveats:
  - Limited to standard animatable properties (i.e. `frame`, `transform`, etc.)
  - Redirecting spring animations isn't as easily done, and you do lose some control to the API
  - Doesn't allow you to make changes on a frame-by-frame basis, which you'll see is important later on

Bouncy Buttons

- Acknowledges touches
- Compliments input with a positive reaction
- Can be seen as soon as the touch is lifted
- Reinforces user intent; responsive



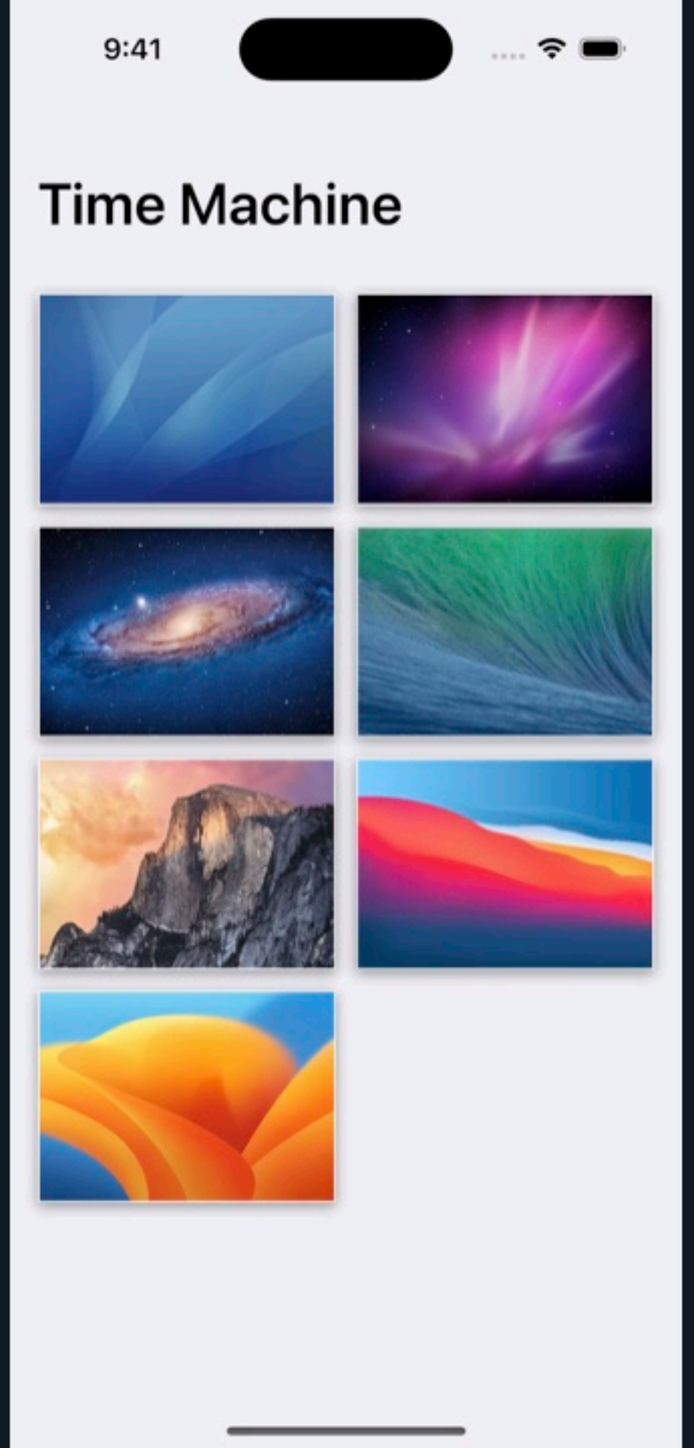
Bouncy Buttons: Accessibility

- They're very responsive and more visually distinguishing
- This makes them more accessible and more approachable to more people, regardless of vision impairment
- Adding borders / backgrounds to your buttons make this even better

Lack of Touch Acknowledgement

Fixed ✅

- Even if there's a slow network, the app acknowledges you interacting with the wallpapers
- You know you which one you tapped, and are less likely to think the touch was dropped / ignored



Time Machine: Things to Fix

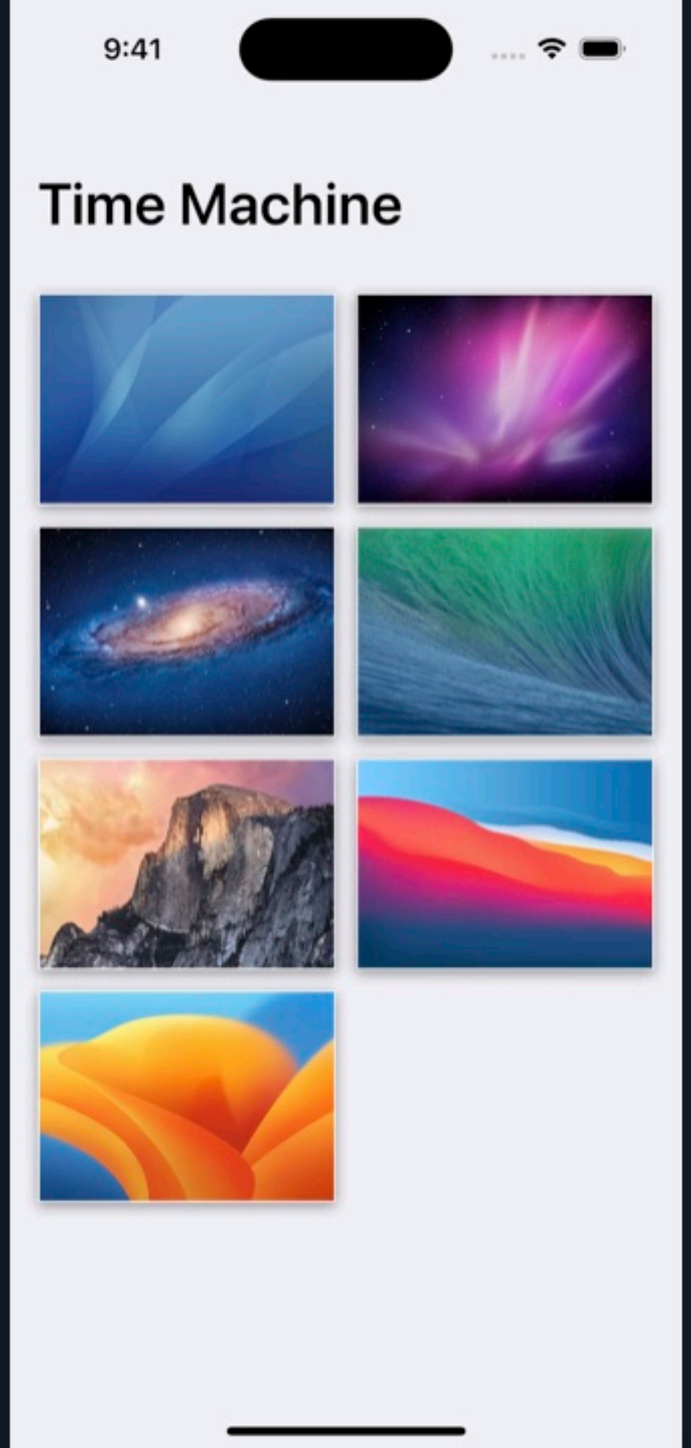
1. ~~Slow and Broken Presentation Animation~~ ✅
2. **Lack of Touch Acknowledgement** 🔨
3. Animations Not Complimenting Gestures

Time Machine: Things to Fix

1. ~~Slow and Broken Presentation Animation~~ ✅
2. ~~Lack of Touch Acknowledgement~~ ✅
3. **Animations Not Complimenting Gestures** 🔨

Dismissal not Complimenting Gesture

- When dismissing, the wallpaper begins to dismiss before we've even finished a proper swipe, which could lead to accidental dismissal
- It also dismisses at a velocity (speed) different than our swipe
- It's powered by `UISwipeGestureRecognizer`
- This makes it feel unresponsive, and disconnected from your input



Solution?

`UISwipeGestureRecognizer` is Bad.

Don't Use It.

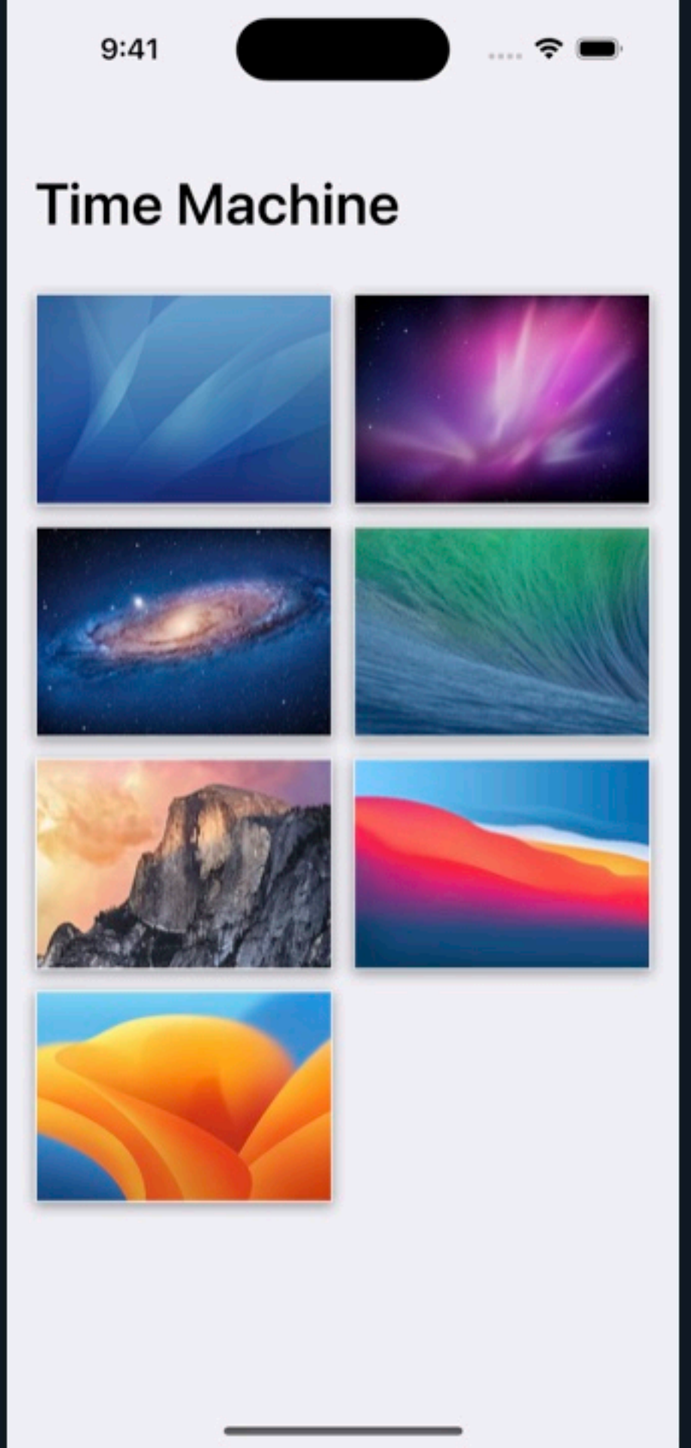
No.

`UIPanGestureRecognizer` is Better

- `UIPanGestureRecognizer` is a continuous gesture recognizer, which allows you to make changes that make your app feel responsive
- There is no delay before anything happens

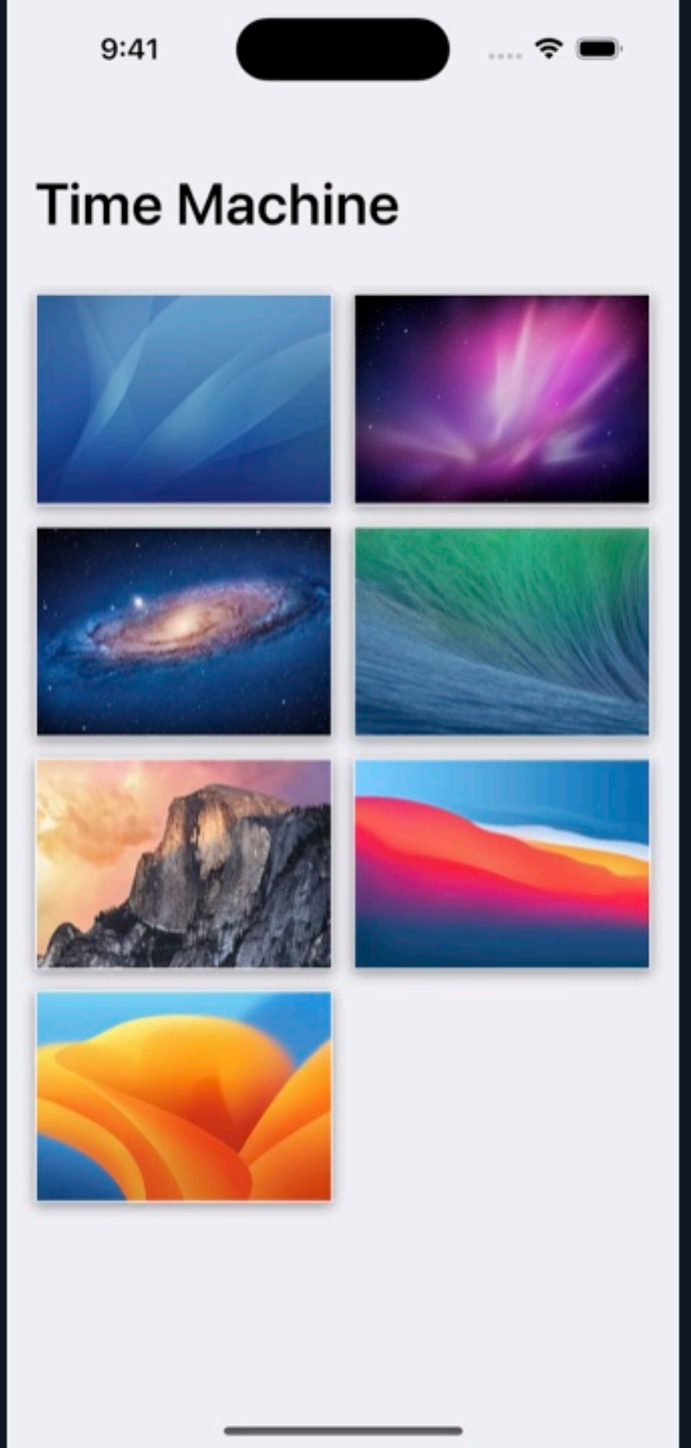
Time Machine:
`UIPanGestureRecognizer`

- Now when you drag, the app responds immediately
- After you let go, it dismisses as expected
- This is already better, but it's still a bit off...



Time Machine:
`UIPanGestureRecognizer`

- If you watch it closely, you'll see it still feels discontinuous, it dismisses with a different velocity
- It feels like two separate actions, vs. being a smooth handoff
  - Drag, and then Dismiss
- This discontinuity creates perceived unresponsiveness, as your input doesn't feel like it's being respected / complimented



Better Dismissal Animation

```
private lazy var dismissAnimation = {
    let dismissAnimation = SpringAnimation<CGFloat>(initialValue: 0.0, response: 0.3, dampingRatio: 1.0)
    dismissAnimation.onValueChanged(disableActions: true) { [weak self] newTranslationY in
        guard let self = self else { return }
        self.view.layer.transform = CATransform3DMakeTranslation(0.0, newTranslationY, 0.0)
        self.dimmingView.alpha = 1.0 - (newTranslationY / self.view.bounds.size.height)
    }
    dismissAnimation.completion = { [weak self] in
        self?.dismiss(animated: false)
    }
    dismissAnimation.resolvesUponReachingToValue = true
    return dismissAnimation
}()
```

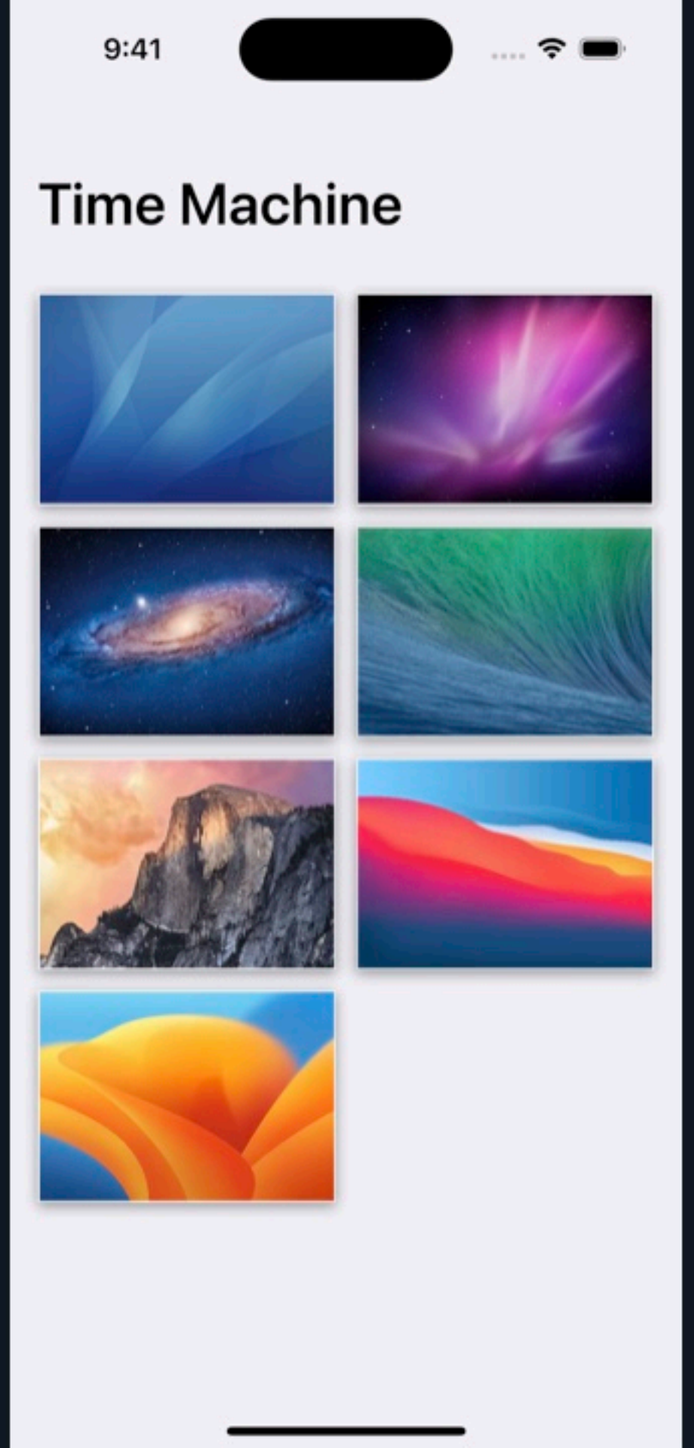
Better Gesture Recognition

```
private lazy var dismissAnimation = { /* ... */ }

@objc private func handleDismissGestureWithMotion(_ panGestureRecognizer: UIPanGestureRecognizer) {
    let translation = panGestureRecognizer.translation(in: self.view)
    let velocity = panGestureRecognizer.velocity(in: self.view)

    switch panGestureRecognizer.state {
        case .began:
            dismissAnimation.updateValue(to: translation.y, postValueChanged: true)
        case .changed:
            dismissAnimation.updateValue(to: translation.y, postValueChanged: true)
        case .cancelled, .ended:
            dismissAnimation.velocity = velocity.y
            dismissAnimation.toValue = view.bounds.size.height
            dismissAnimation.start()
        default:
            return
    }
}
```

Dismissal Animation

Fixed ✅

- Now it's a single continuous dismissal
- You drag to dismiss, and as soon as you let go it hands off the `velocity` to the dismissalAnimation
- The animation is an extension of the gesture
- This interaction responds positively to touch input



Time Machine: Things to Fix

1. ~~Slow and Broken Presentation Animation~~ ✅
2. ~~Lack of Touch Acknowledgement~~ ✅
3. Animations not Complimenting Gestures 🔨

Time Machine: Things to Fix

1. ~~Slow and Broken Presentation Animation~~ ✅
2. ~~Lack of Touch Acknowledgement~~ ✅
3. ~~Animations not Complimenting Gestures~~ ✅
4. One More Thing 🔨

Making the Interactions Even Better

- Currently, when you tap on a wallpaper, a larger copy presents from the bottom of the screen... but in the real world, things don't work that way
- Continuity is really important, especially with gestures, so why not pick up the wallpapers themselves and show them in the middle of the screen?
- Once you drag down the wallpaper, it would then go back to where it came from

Time Machine: Things to Fix

1. ~~Slow and Broken Presentation Animation~~ ✅
2. ~~Lack of Touch Acknowledgement~~ ✅
3. ~~Animations Not Complimenting Gestures~~ ✅
4. Make Presentations / Dismissals Better and Interruptible 🔨

Enhanced Presentation / Dismissal

- To adopt this, we'll need to simplify / change some things
- For now, we'll drop using standard view controller presentation and resort to manual view controller containment:

```
let photoViewController = /* ... */

addChild(photoViewController)

view.addSubview(photoViewController.view)

photoViewController.didMove(toParent: self)
```

Enhanced Presentation / Dismissal

- We'll also need to handle mounting the dimming view's separately, since we're no longer using the transitioning coordinator
- We'll need to change the animations to no longer present from / dismiss offscreen, since we're going to be picking up the wallpapers themselves
- Lastly, to make this feel really good we're going to make sure that all our presentation / dismissals are fully interruptible

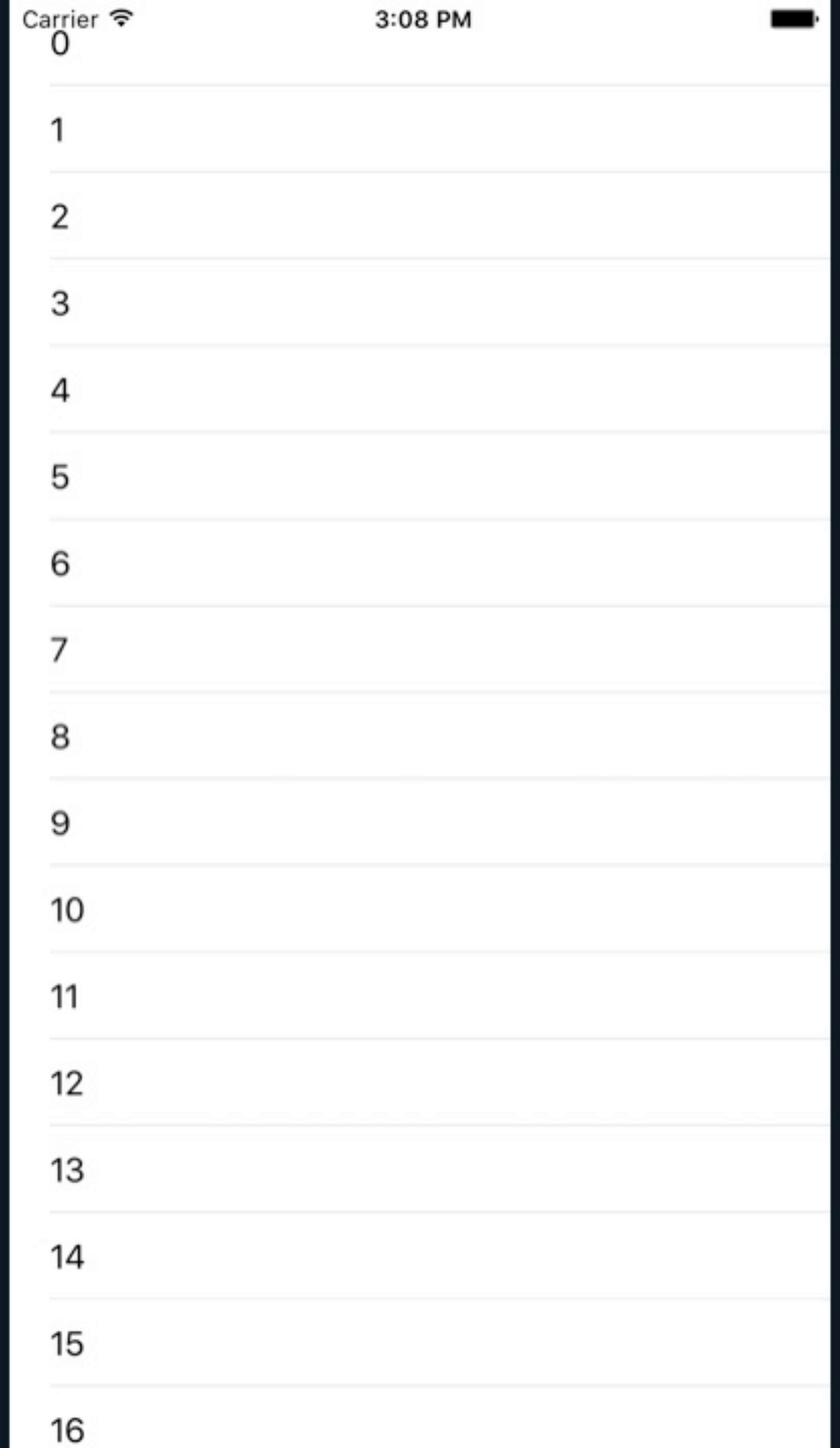
Interruptibility

- Interruptibility is when you can interrupt any animation and redirect it / cancel it
- In an ideal world, everything in an app that animates is interactive **always** (this is the default in SwiftUI!)
- Animations should never be cumbersome, nor should they block input, they should compliment input / touches
- This can be challenging to do everywhere, but for gesturally-driven interactions, it's very important

"Is it really that important?"

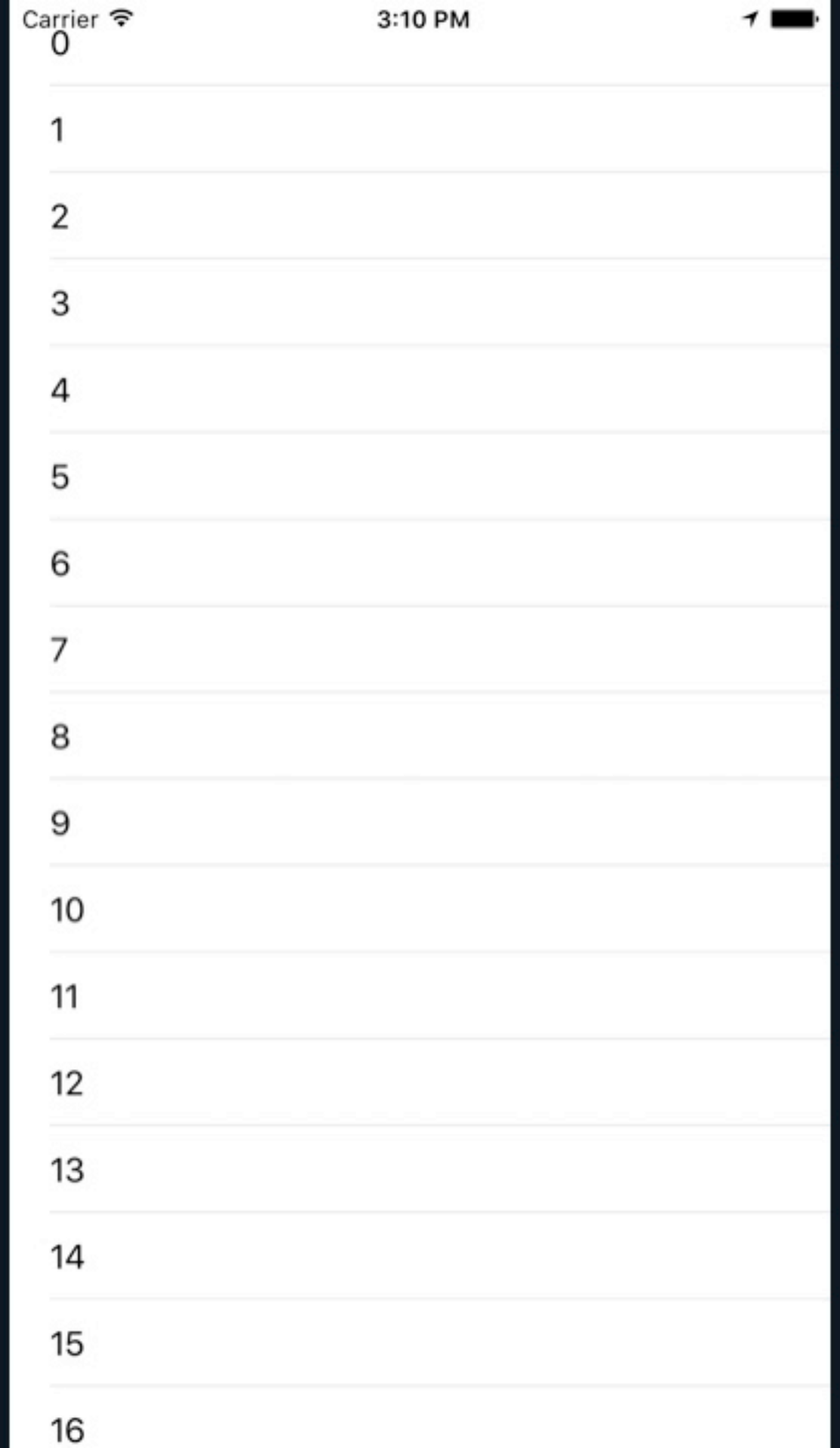
UIScrollView with Interruptibility

- Works as you expect



UIScrollView without Interruptibility

- You need to wait until it finishes scrolling before you can start scrolling again
- Could you imagine how frustrating that would be?



Interruptible Wallpaper Presentations

To start, we're going to have 3 primary dictionaries of information

Each maps a particular photo view to respective information

```
var positionAnimations = [PhotoView: SpringAnimation<CGPoint>]()
var boundsAnimations = [PhotoView: SpringAnimation<CGRect>]()
var initialPickupLocations = [PhotoView: CGPoint]()
```

Interruptible Wallpaper Presentation: Gestures

```
@objc func didPan(_ gestureRecognizer: UIPanGestureRecognizer) {
    let photoView = gestureRecognizer.view as! PhotoView

    let translation = panGestureRecognizer.translation(in: view)
    let velocity = panGestureRecognizer.velocity(in: view)

    switch panGestureRecognizer.state {
        case .began:
            initialPickupLocations[photoView] = photoView.layer.position
            positionAnimations[photoView]?.stop()

        case .changed:
            if let initialPickupLocation = initialPickupLocations[photoView] {
                let newPosition = CGPoint(x: initialPickupLocation.x + translation.x, y: initialPickupLocation.y + translation.y)
                positionAnimations[photoView]?.updateValue(to: newPosition, postValueChanged: true)
            }
        case .ended, .cancelled:
            if photoView.isOpened {
                if velocity.y < -400.0 {
                    close(photoView, velocity: velocity)
                } else {
                    open(photoView, velocity: velocity)
                }
            } else {
                if velocity.y > 400.0 {
                    open(photoView, velocity: velocity)
                } else {
                    close(photoView, velocity: velocity)
                }
            }
        default:
            break
    }
}
```

Time Machine: Enhanced Presentation Animation

```
private func open(_ photoView: PhotoView, velocity: CGPoint? = nil) {
    view.insertSubview(dimmingView, belowSubview: photoView)

    let boundsAnimation = boundsAnimations[photoView] ?? {
        let boundsAnimation = SpringAnimation<CGRect>(response: 0.4, dampingRatio: 0.8)
        boundsAnimation.updateValue(to: photoView.bounds)
        boundsAnimations[photoView] = boundsAnimation

        boundsAnimation.onValueChanged { [weak self] newBounds in
            photoView.bounds = newBounds
            dimmingView.alpha = self?.percentOpened(photoView, currentBounds: newBounds) ?? 0.0
        }

        return boundsAnimation
    }()

    boundsAnimation.toValue = expandedBounds(for: photoView)

    let positionAnimation = positionAnimations[photoView] ?? {
        let positionAnimation = SpringAnimation<CGPoint>(initialValue: .zero, response: 0.5, dampingRatio: 0.9)
        positionAnimation.onValueChanged(disableActions: true) { newPosition in
            photoView.layer.position = newPosition
        }
        positionAnimations[photoView] = positionAnimation
        return positionAnimation
    }()

    positionAnimation.toValue = CGPoint(x: view.bounds.midX, y: view.bounds.midY)

    if let velocity = velocity {
        positionAnimation.velocity = velocity
    }

    boundsAnimation.start()
    positionAnimation.start()

    photoView.opened = true
}
```

Time Machine: Enhanced Dismissal Animation

```
private func close(_ photoView: PhotoView, velocity: CGPoint? = nil) {
    view.insertSubview(dimmingView, belowSubview: photoView)

    let boundsAnimation = boundsAnimations[photoView]!
    boundsAnimation.toValue = collapsedBounds(for: photoView)

    let positionAnimation = positionAnimations[photoView]!
    positionAnimation.toValue = collapsedPosition(for: photoView)

    if let velocity = velocity {
        positionAnimation.velocity = velocity
    }

    boundsAnimation.start()
    positionAnimation.start()

    photoView.opened = false
}
```

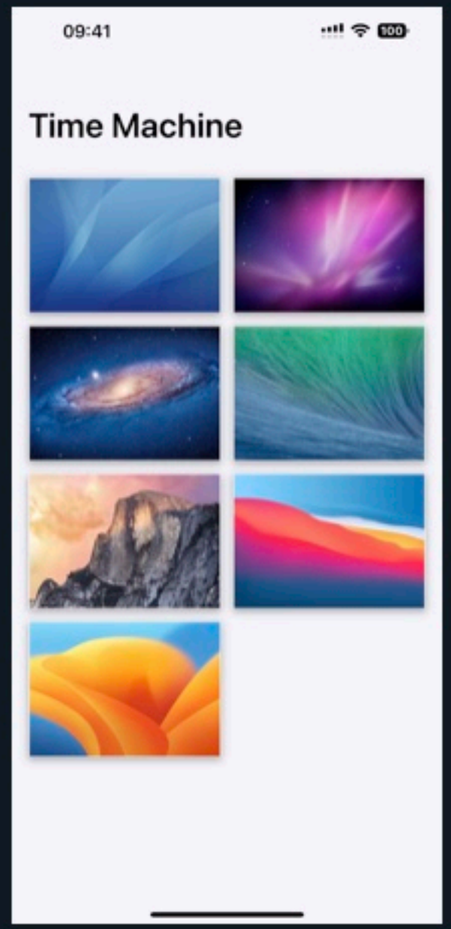
Interruptibility Notes

- Once you have a system of gestures working properly, you'll realize that you have a nice sequence of events that can happen
- Open/Close -> Interrupt? -> Open/Close
- You no longer need to worry about cancelling in-flight animations or waiting for them to finish
- Your app becomes much more dynamic

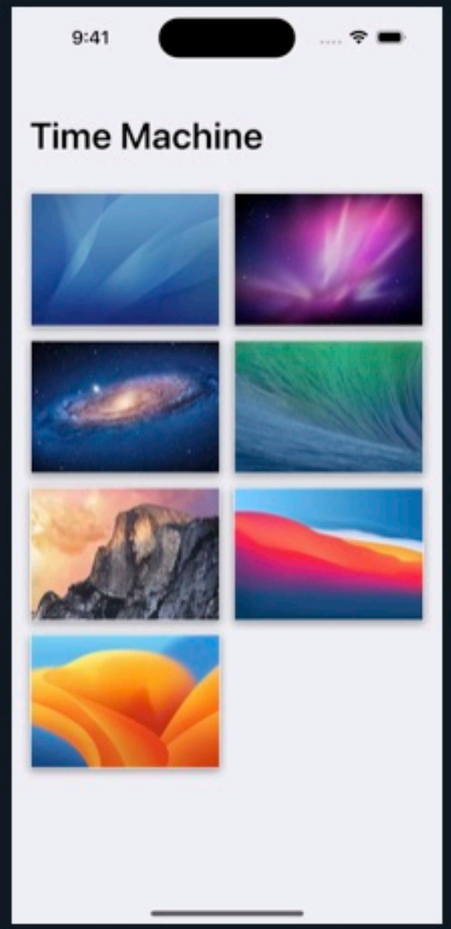
Things to Note

- One thing to keep in mind, when using an animation framework like Motion: it doesn't run out of process like CoreAnimation does
- It is **not** a silver bullet
- If you're doing lots of work on the main thread, even a top spec iPhone or iPad **will** drop frames, you need to be very careful with where you're doing work
- Consider doing asynchronous text measurement / layout

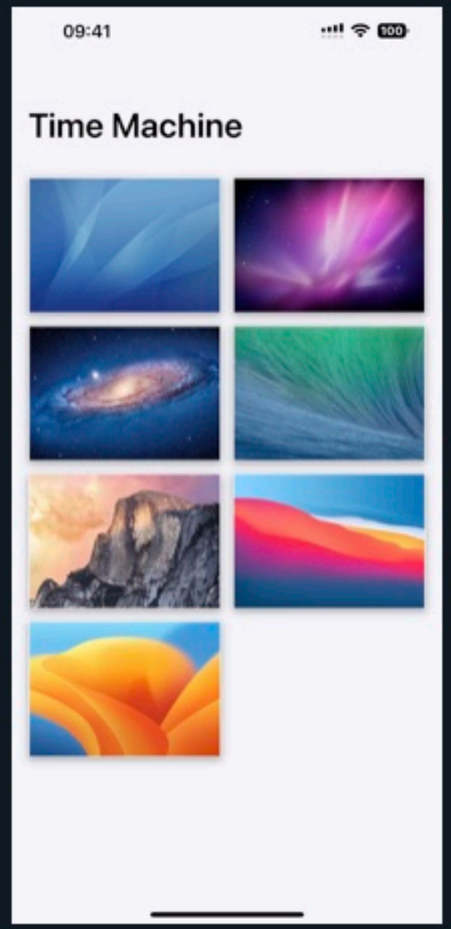
Time Machine: Fully Enhanced



Time Machine: Before



Time Machine: After



Some would say that interactions like these are "superfluous moving and jiggling" and / or "needless distractions"

However, that would be a really bad take because, as we've proven, these actually do make your app easier to use / understand as well as more accessible to a wider audience

They all work together to make your app feel **great** to use, it's why you'll find interactions like these all over iOS

Presentation / Dismissals and Accessibility

- Transitions like these that have great continuity are very effective at communicating what's happening / where you're going / how you got there
- There's less guesswork with how your app works, and as such, people spend more time using your app vs. figuring it out
- You end up creating apps that are more friendly, inviting, and welcoming for people

Speaking of Accessibility...

- Don't forget that for some people, animations like these can be a bit too much
- Always be sure to have fallbacks for `UIAccessibility.isReduceMotionEnabled`
- Crossfade transitions are an excellent starting point

Responsiveness -> Playfulness

- Positive reactions to input are what make an app feel responsive
- They encourage exploration and aid in developing muscle memory
- You feel good about using an app and it becomes something you can enjoyably rely on
- By building a responsive app, you set the groundwork for gestural experiences that elevate your app to being a playful and delightful one

To Increase Responsiveness, Focus on the Delays

- Any small delay or hitch in your app does harm its overall experience
- Use Instruments' Hitches and Time Profiler tools to survey your app for frame drops and expensive operations
- Focus on the overall experience of presenting and dismissing view controllers, they're often the first thing people notice
- Try shifting your interactions to being interruptible, the overall experience will get better significantly
- If you're using UISwipeGestureRecognizer, **throw it away**

To Increase Responsiveness, Focus on the Details

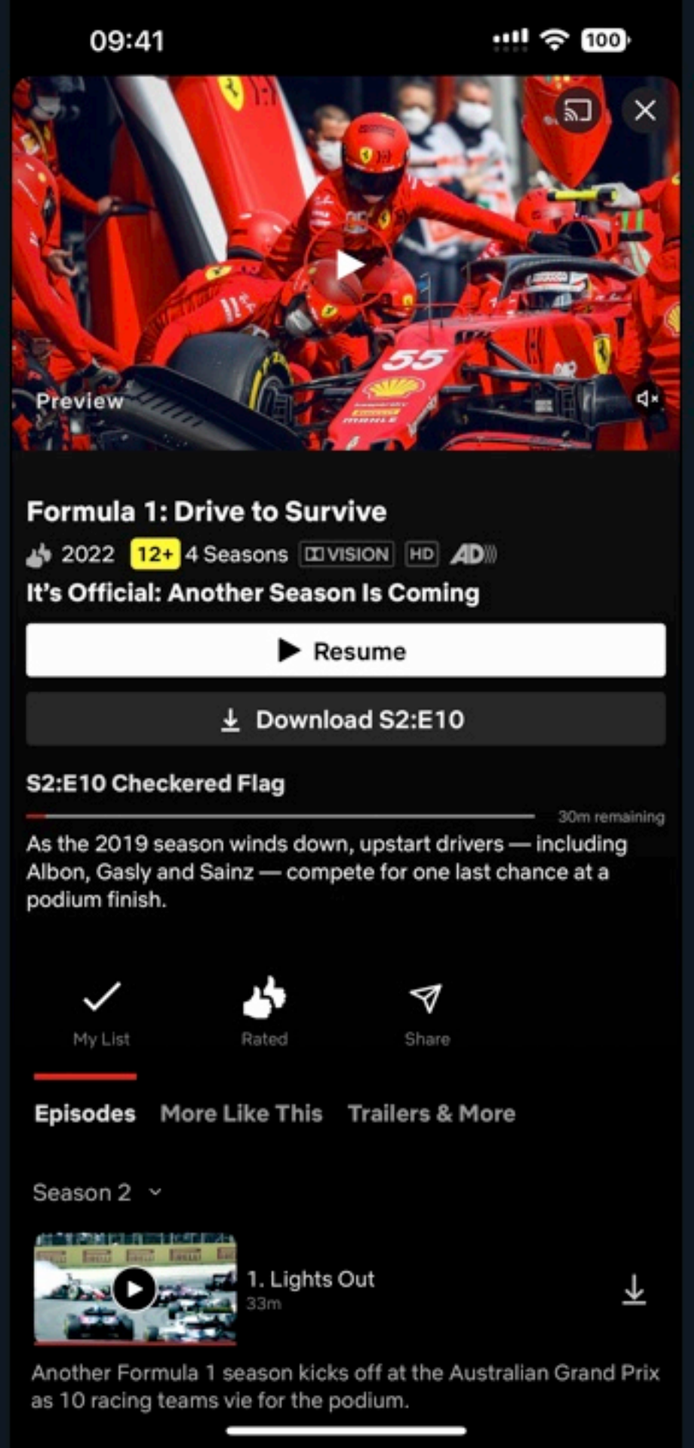
- A lot of this talk has probably seemed like we're picking over very small details, but these details matter
- Small overlooks can turn into small "UI paper-cuts", and if you have too many, it can ruin the experience
- Really focusing on all the details can elevate your app from a good app to a great app

The Netflix App

You know, some would say "it's just a video streaming app"

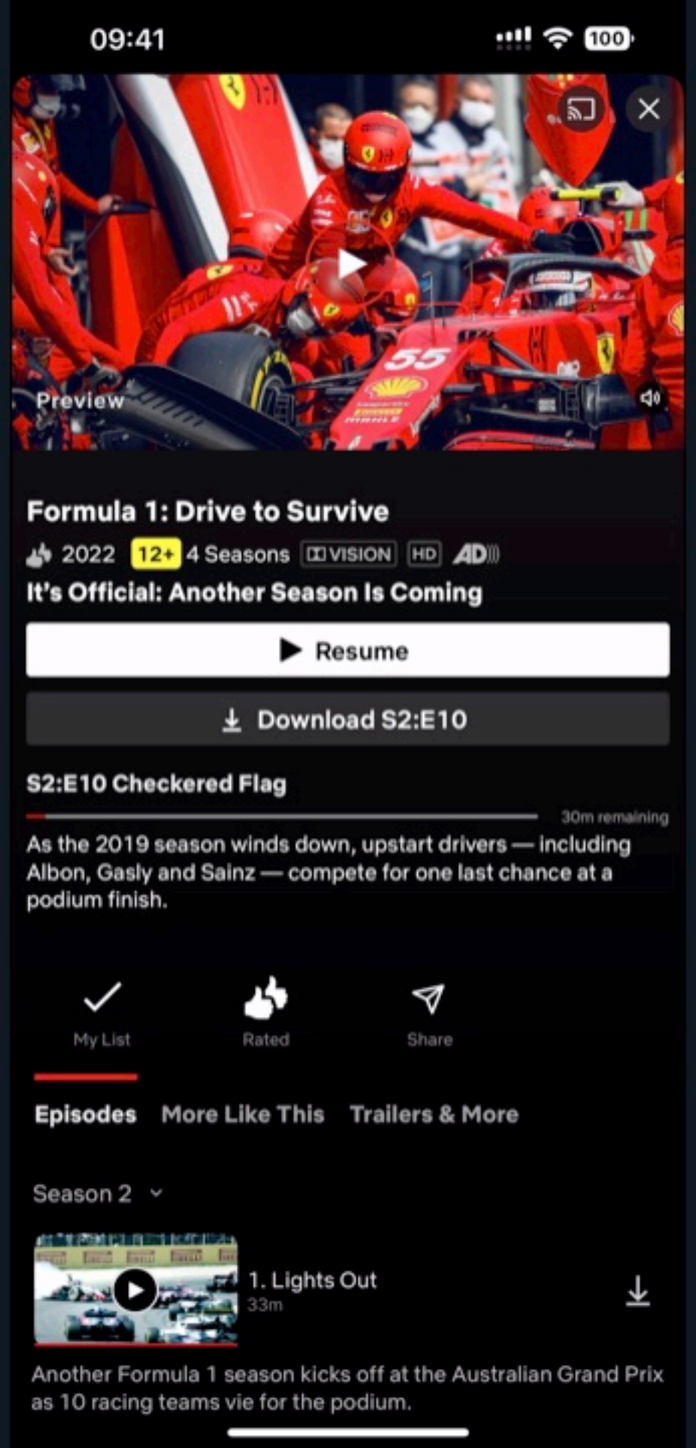
But for my team, we work really hard to make it feel like "the **best** streaming app"

The better the app becomes, the more enjoyable it becomes when finding the best stuff to watch or play



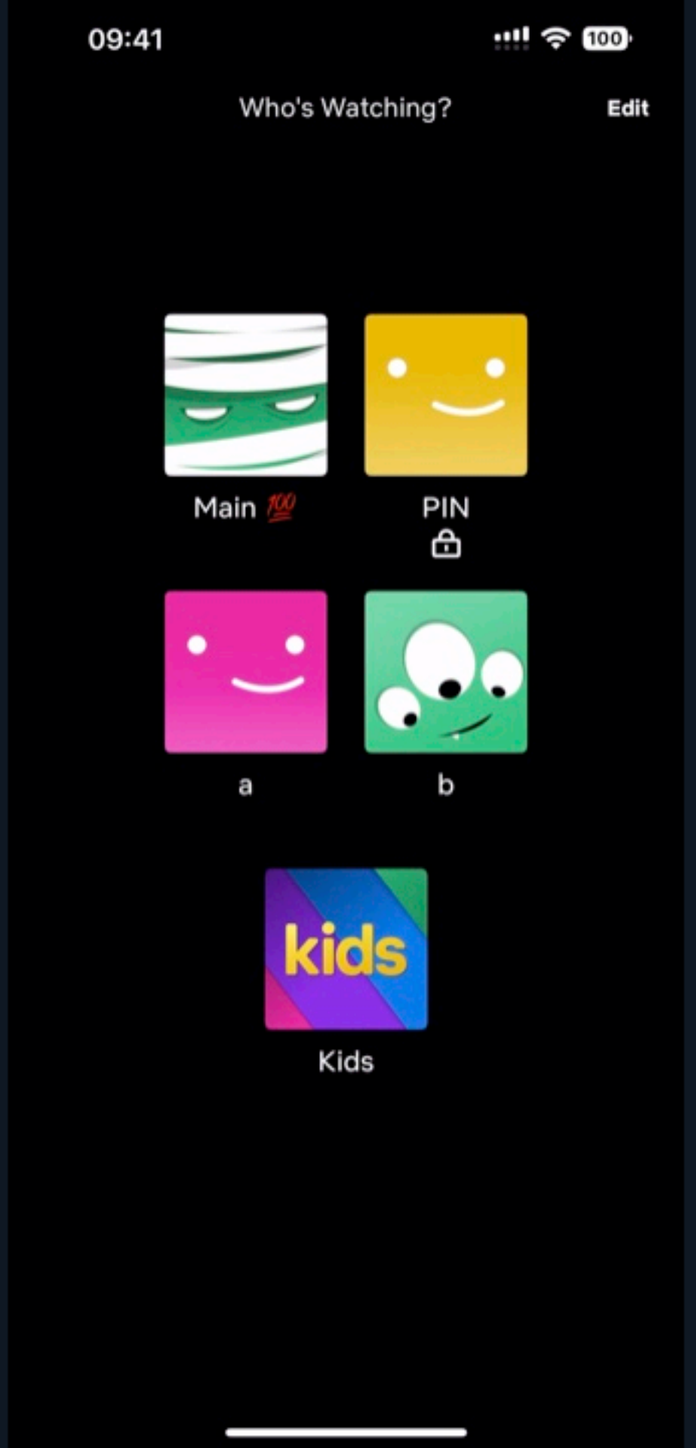
The Netflix App

- We spend a lot of time working on the quality, responsiveness, and accessibility of the app, so it's the best it can be for anyone who enjoys Netflix
- Recently, we spent a lot of time redoing all our navigation into lightweight and fully interruptible sheets, making exploration just that much more fun in the process



Closing Thoughts

- Creating responsive and playful apps are a great craft to practice, and they're the reason why apps become so memorable
- Building gestural interactions that reward interaction with positive responses really does make your app feel best-in-class
- There's nothing wrong with embracing the act of building fun apps with delightful expression



iOS has some of the best interaction and animation frameworks in the world, there's no reason not to use it to build some of the best experiences in the world! :D

Thanks!