Aleksandr Filimonov
Senior iOS Dev · alexfilimon.dev

What Disney Taught Us About iOS Animations

SwiftUI is

full of implicit behavior and edge cases

Disney's Principles of Animation

Why animations are important

SwiftUI Animations

Applying everything together

Disney's Principles of Animation

Why animations are important

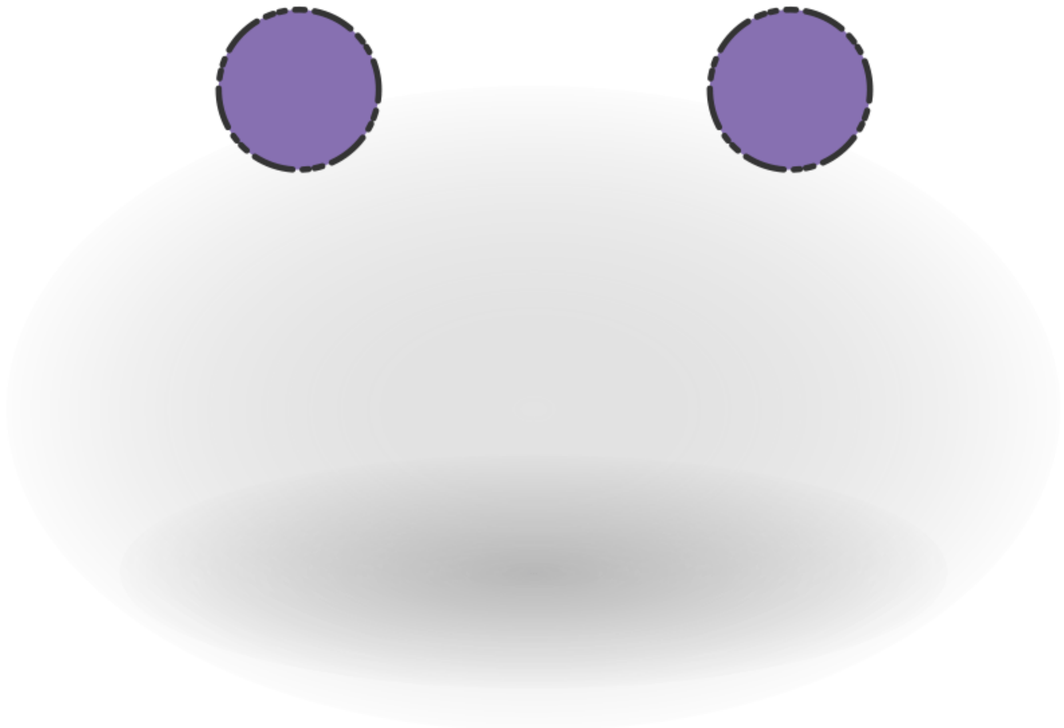
SwiftUI Animations

Applying everything together

Disney's Principles of Animation

1. Squash and Stretch
2. Anticipation
3. Staging
4. Straight Ahead Action and Pose to Pose
5. Follow Through and Overlapping Action
6. Slow In and Slow Out

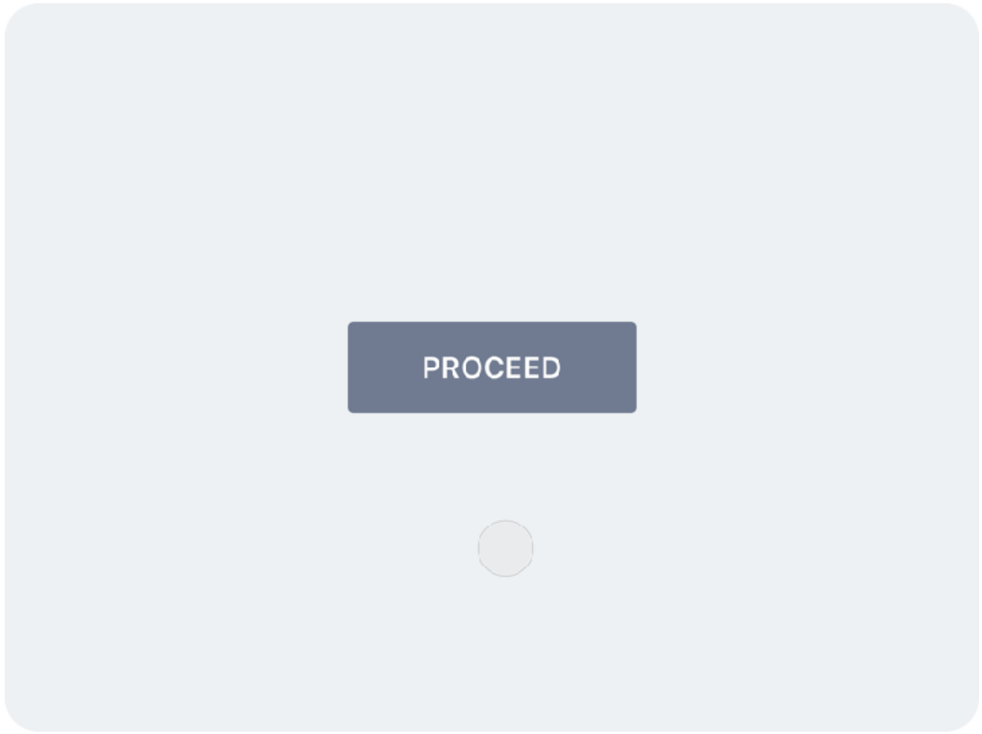
- …



Disney's Principles of Animation

1. Squash and Stretch
2. Anticipation
3. Staging
4. Straight Ahead Action and Pose to Pose
5. Follow Through and Overlapping Action
6. Slow In and Slow Out

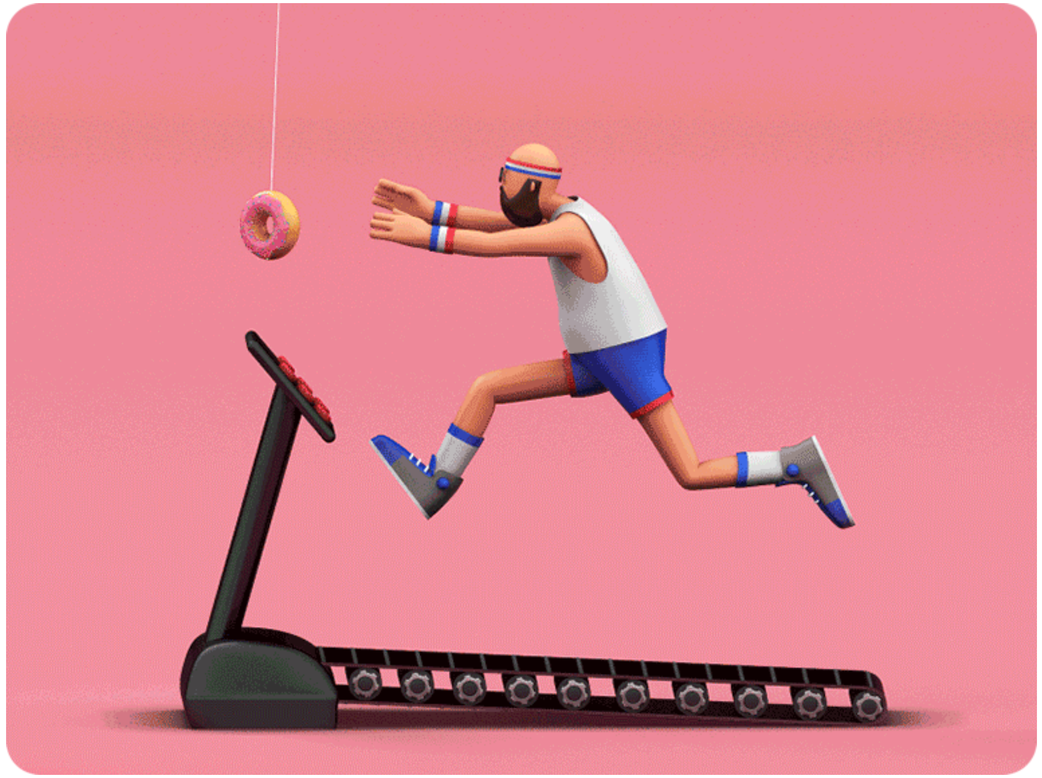
- …



Disney's Principles of Animation

1. Squash and Stretch
2. Anticipation
3. Staging
4. Straight Ahead Action and Pose to Pose
5. Follow Through and Overlapping Action
6. Slow In and Slow Out

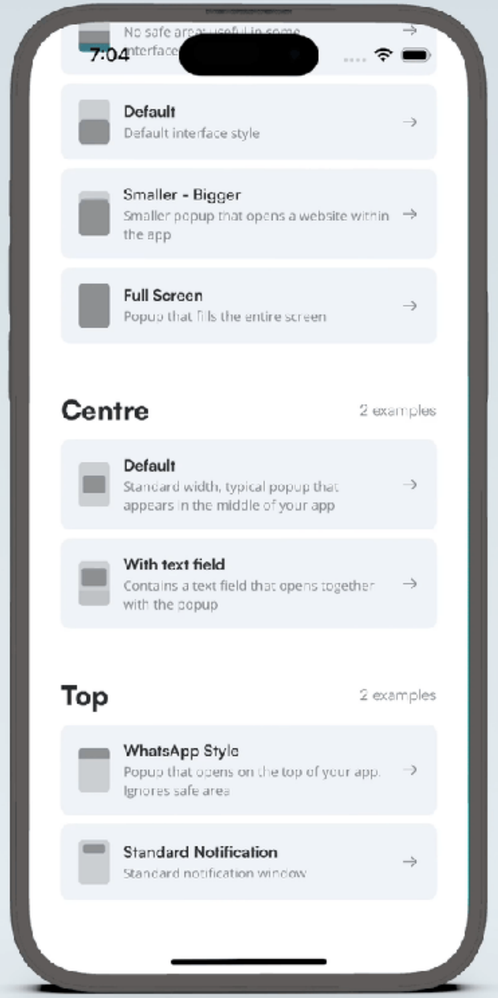
- …



Disney's Principles of Animation

1. Squash and Stretch
2. Anticipation
3. Staging
4. Straight Ahead Action and Pose to Pose
5. Follow Through and Overlapping Action
6. Slow In and Slow Out

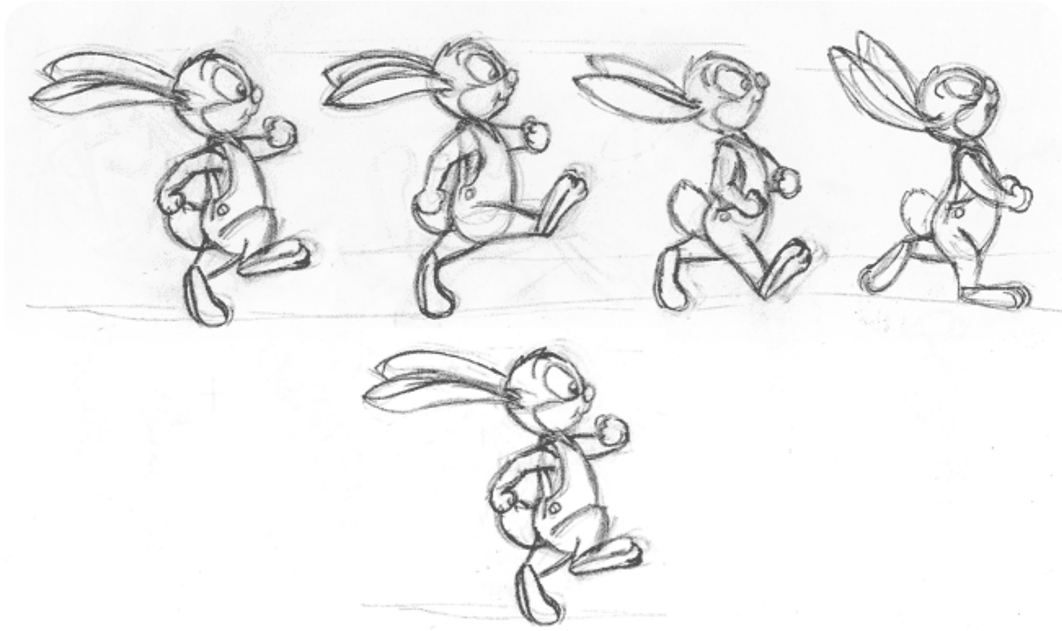
- …



Disney's Principles of Animation

1. Squash and Stretch
2. Anticipation
3. Staging
4. Straight Ahead Action and Pose to Pose
5. Follow Through and Overlapping Action
6. Slow In and Slow Out

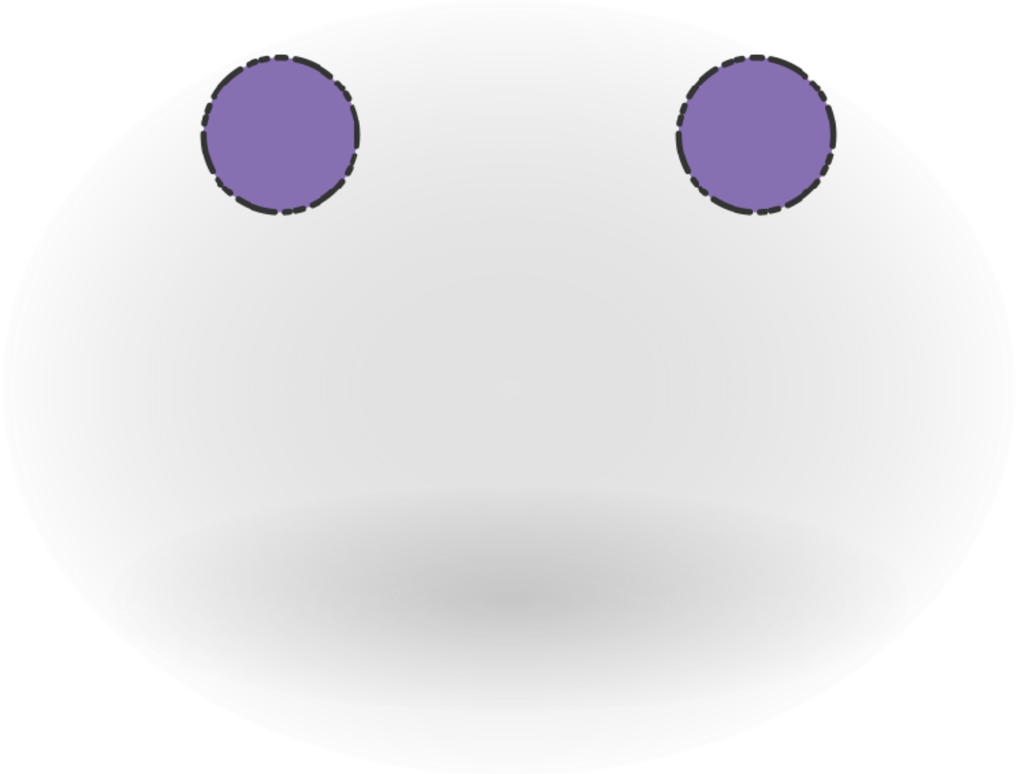
- …



Disney's Principles of Animation

1. Squash and Stretch
2. Anticipation
3. Staging
4. Straight Ahead Action and Pose to Pose
5. Follow Through and Overlapping Action
6. Slow In and Slow Out

- …



Disney's Principles of Animation

1. Squash and Stretch
2. Anticipation
3. Staging
4. Straight Ahead Action and Pose to Pose
5. Follow Through and Overlapping Action
6. Slow In and Slow Out

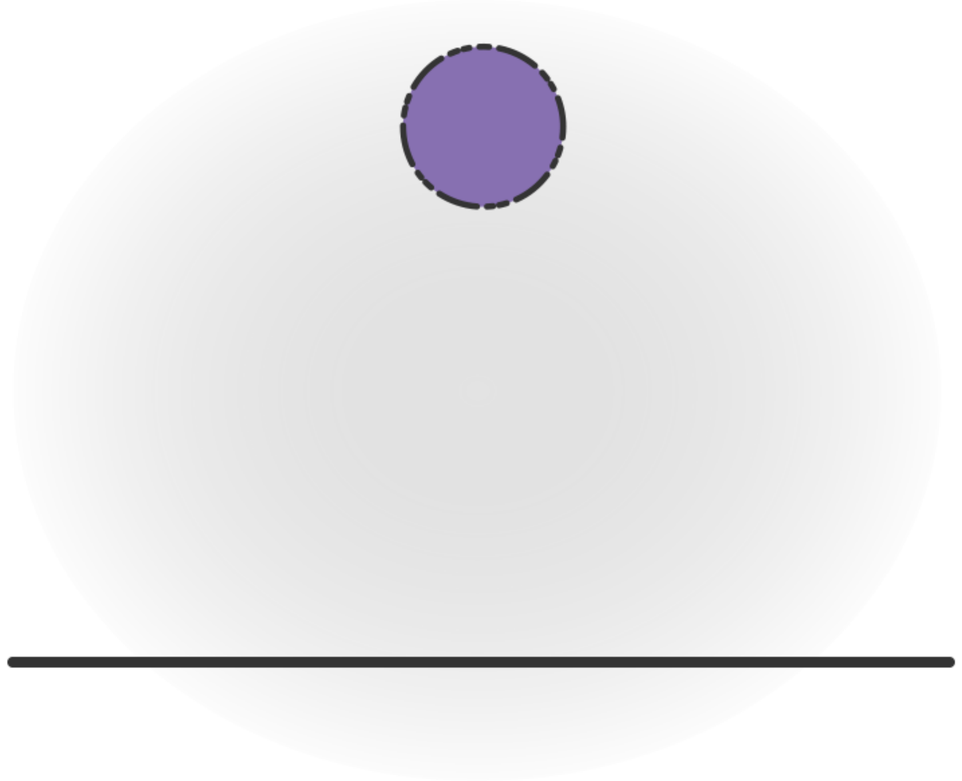
- …

Linear EaseOut

Disney's Principles of Animation

- …

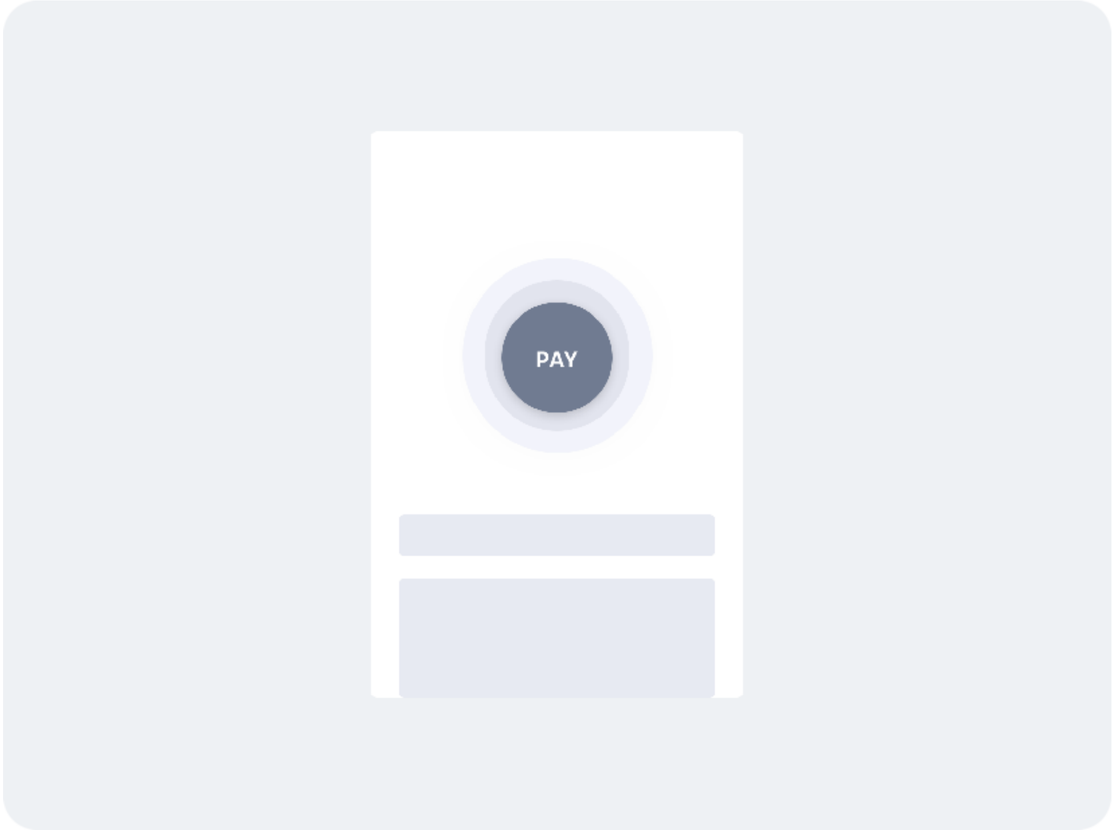
7. Arcs
8. Secondary Action
9. Timing
10. Exaggeration
11. Solid drawing
12. Appeal of Character



Disney's Principles of Animation

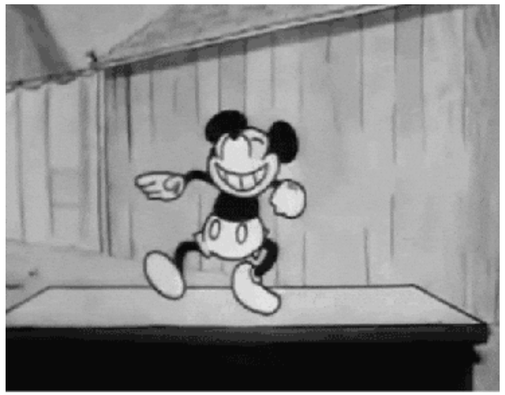
- …

7. Arcs
8. Secondary Action
9. Timing
10. Exaggeration
11. Solid drawing
12. Appeal of Character



Disney's Principles of Animation

Were developed in 1930s and formalised in a book in 1981



Disney's Principles of Animation

Why animations are important

SwiftUI Animations

Applying everything together

Disney's Principles of Animation

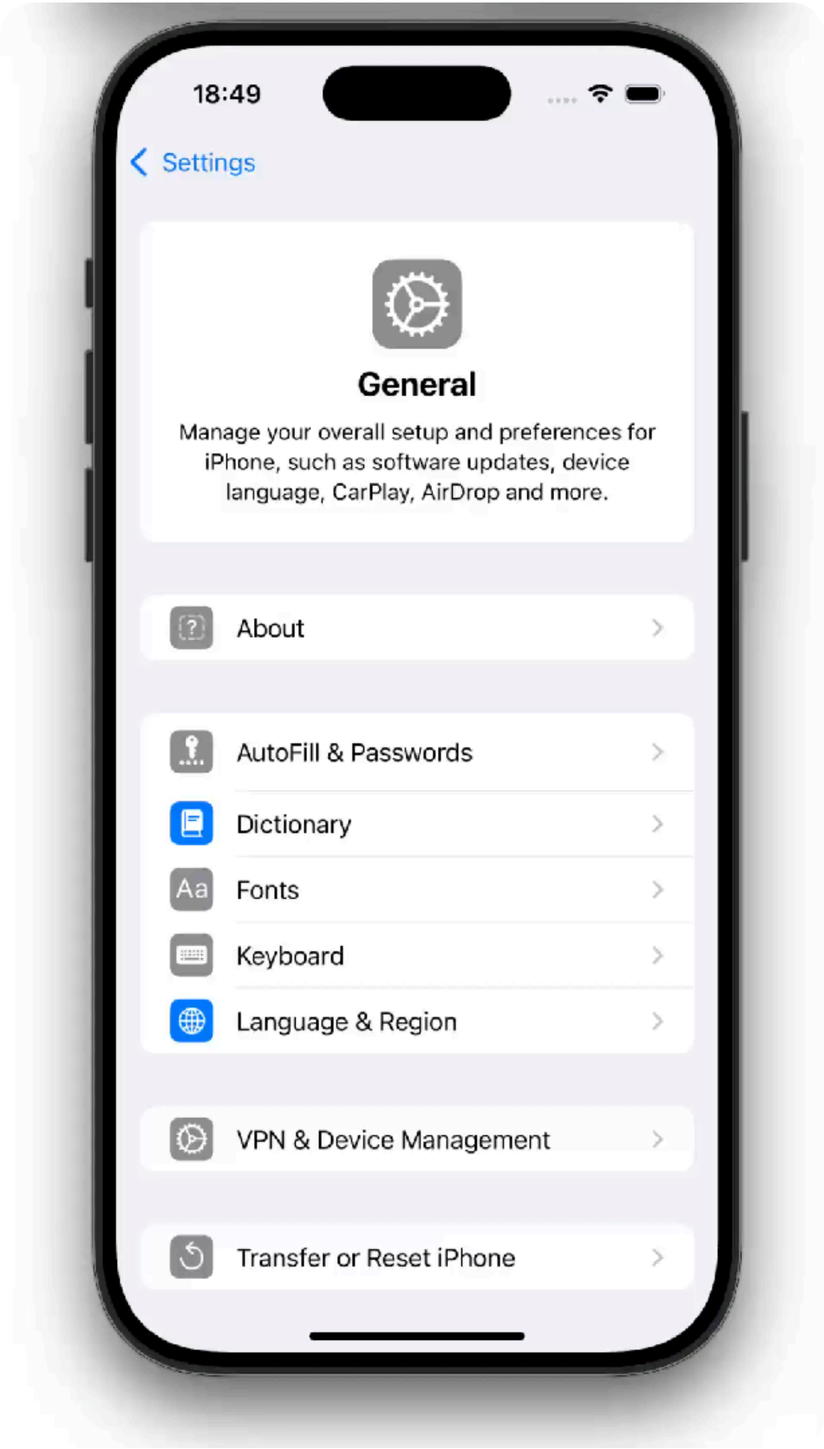
Why animations are important

SwiftUI Animations

Applying everything together

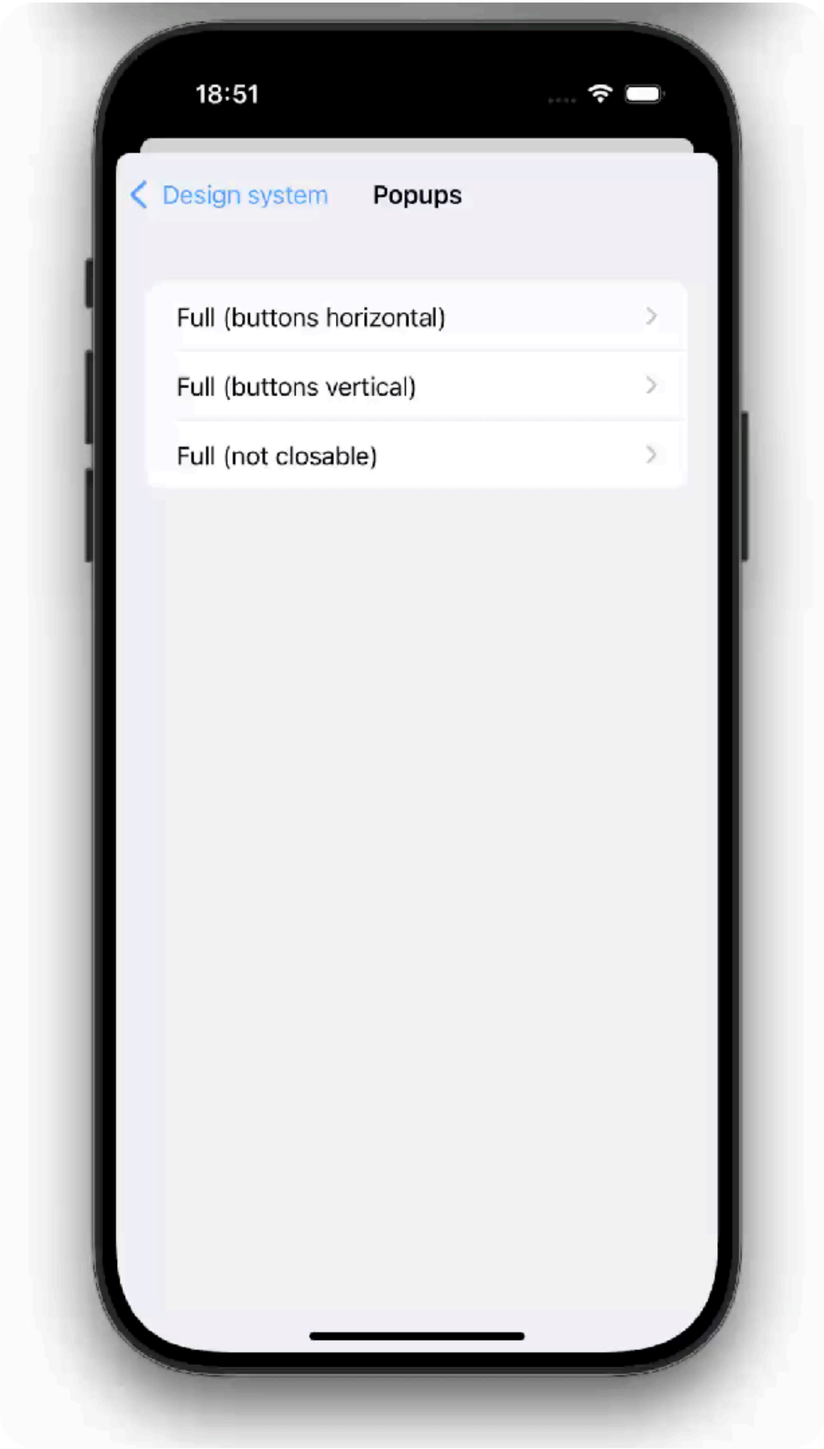
Animation types

- Navigation
- Status
- Feedback
- Instructional
- Delightful
- Emotional



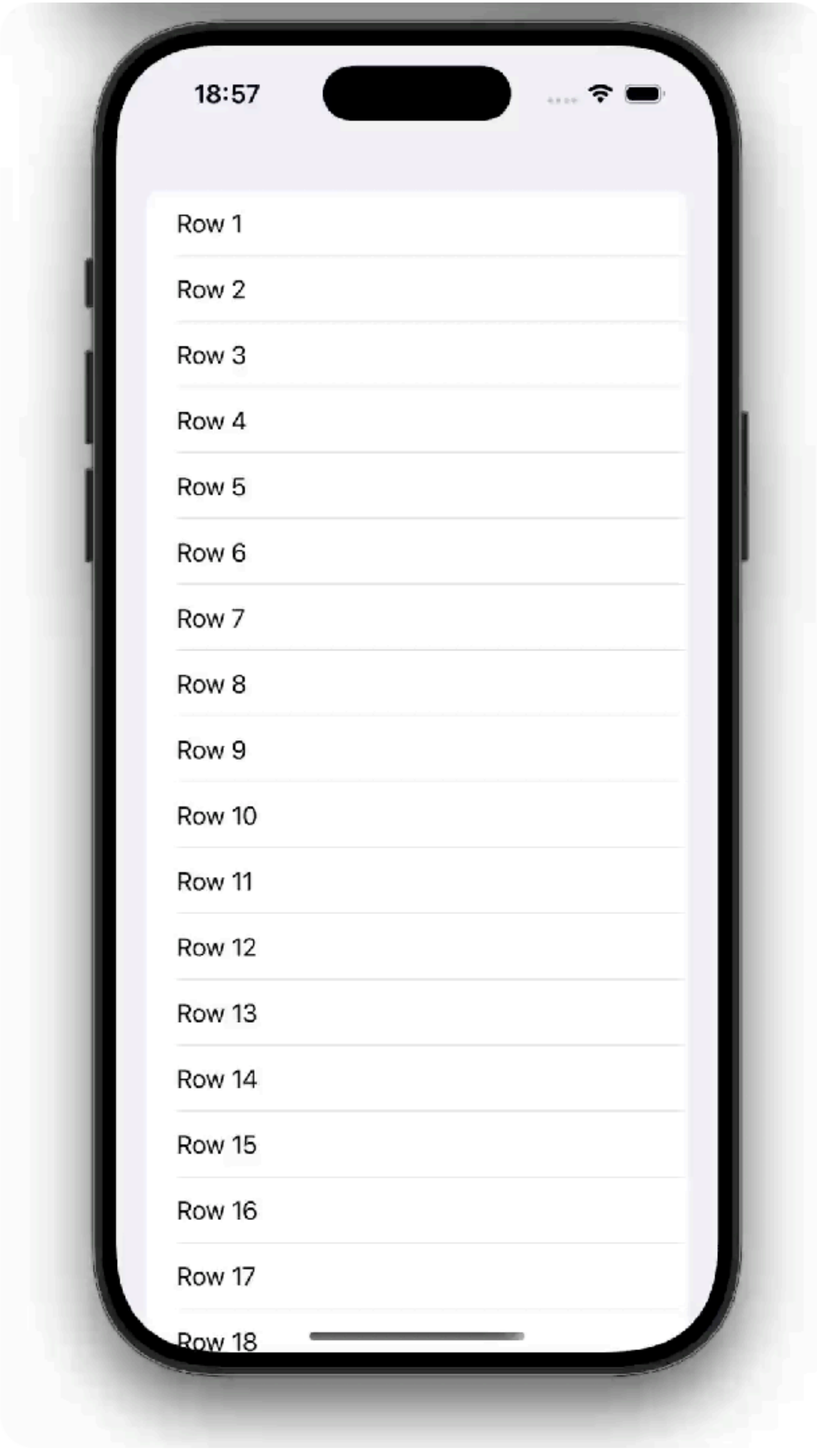
Animation types

- Navigation
- Status
- Feedback
- Instructional
- Delightful
- Emotional



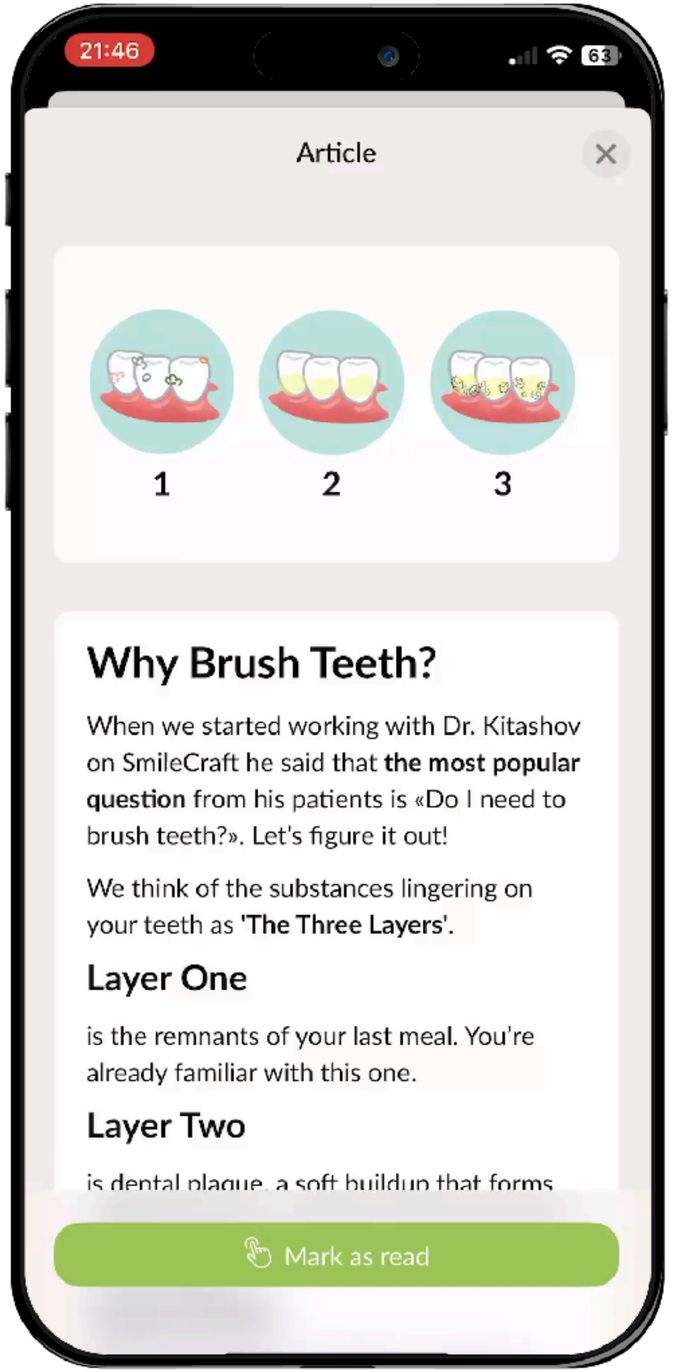
Animation types

- Navigation
- Status
- Feedback
- Instructional
- Delightful
- Emotional



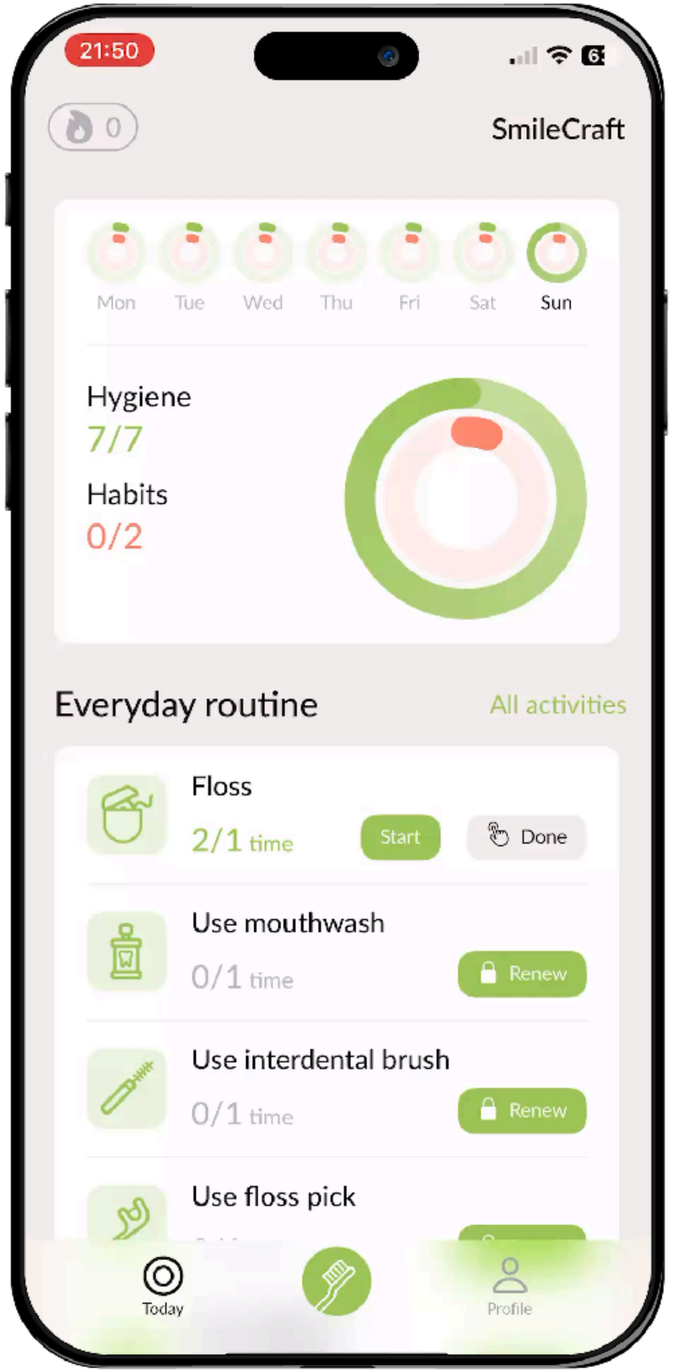
Animation types

- Navigation
- Status
- Feedback
- Instructional
- Delightful
- Emotional



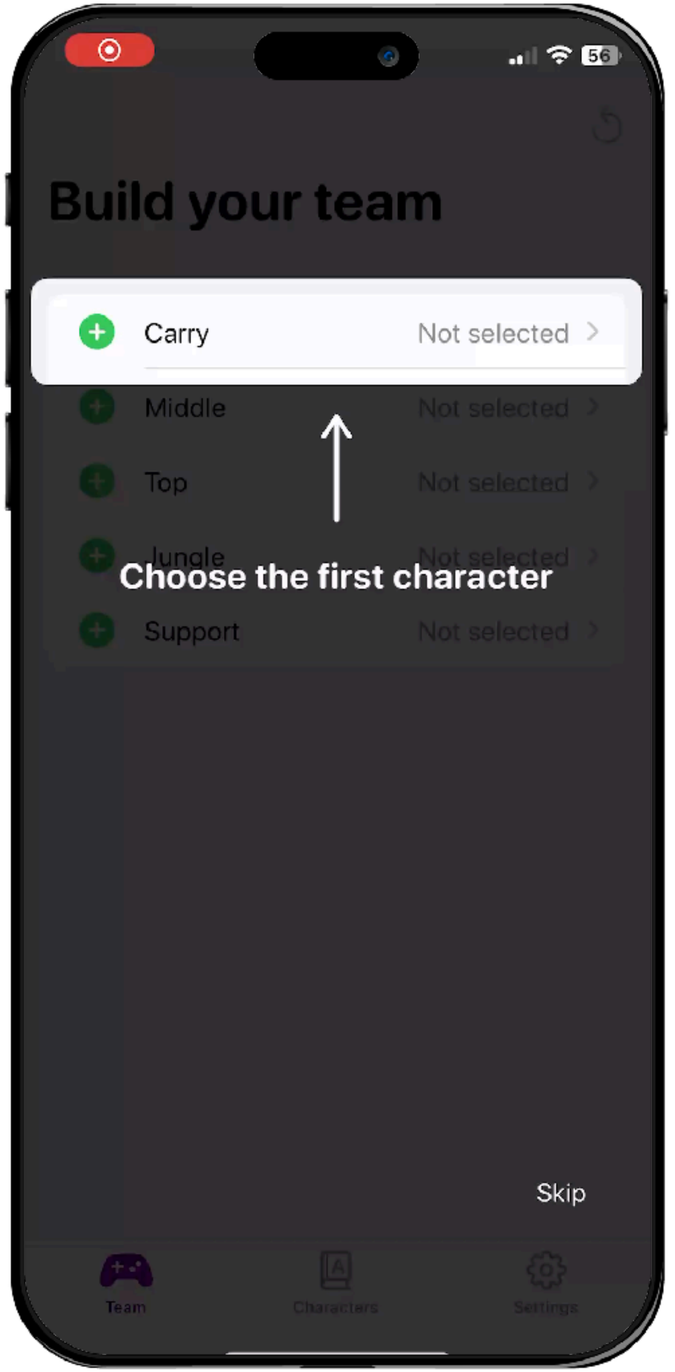
Animation types

- Navigation
- Status
- Feedback
- Instructional
- Delightful
- Emotional



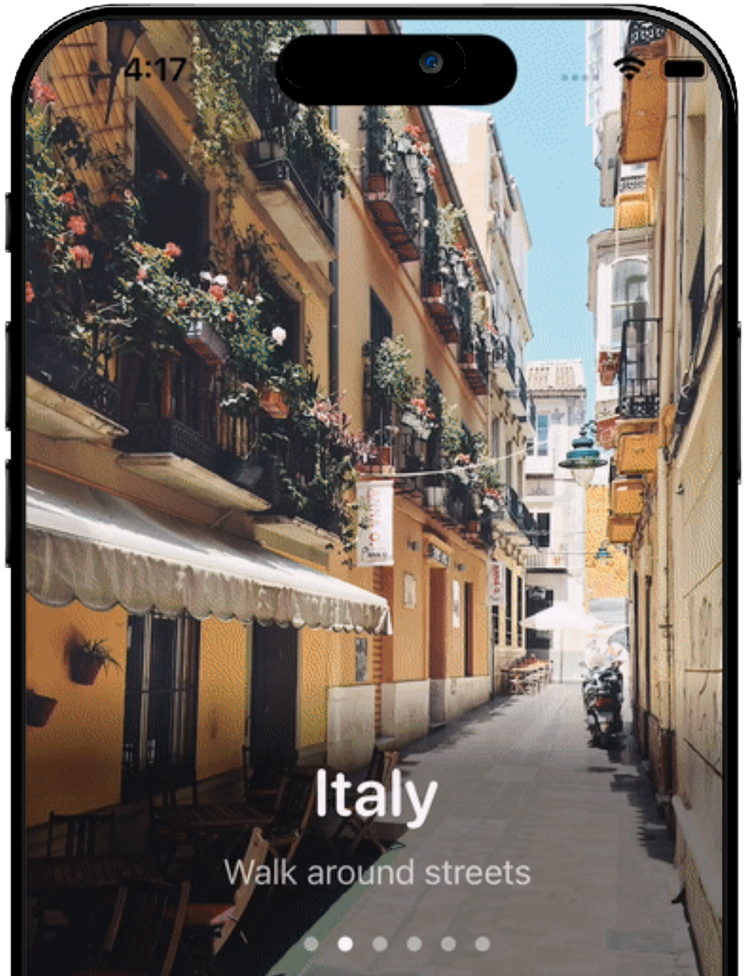
Animation types

- Navigation
- Status
- Feedback
- Instructional
- Delightful
- Emotional



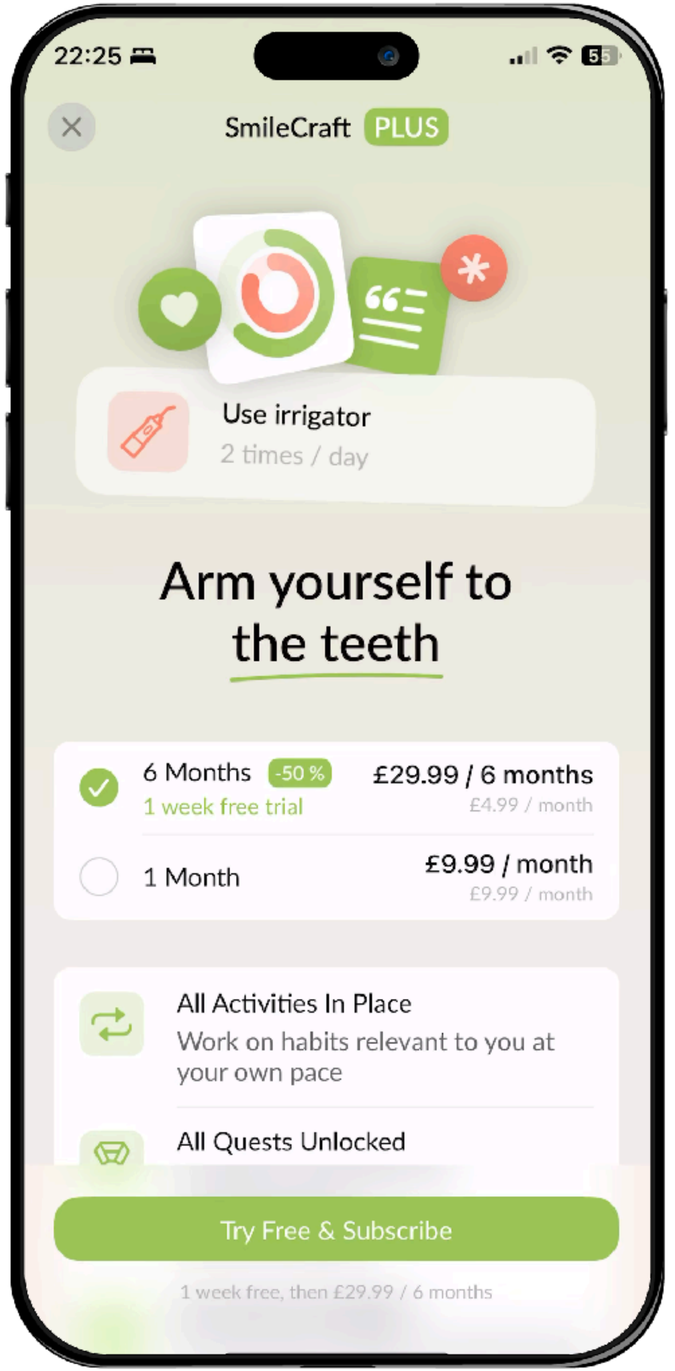
Animation types

- Navigation
- Status
- Feedback
- Instructional
- Delightful
- Emotional



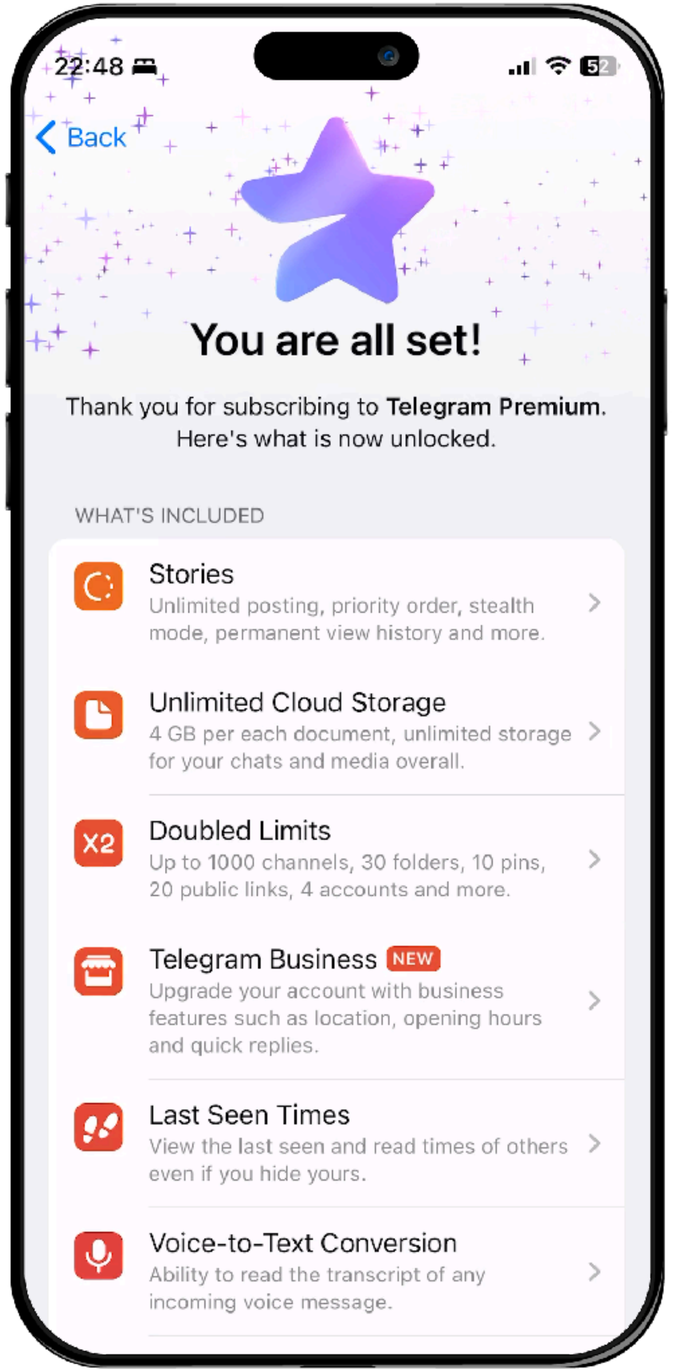
Animation types

- Navigation
- Status
- Feedback
- Instructional
- Delightful
- Emotional



Animation types

- Navigation
- Status
- Feedback
- Instructional
- Delightful
- Emotional



Metrics (for loading screen)

Metrics (for loading screen)

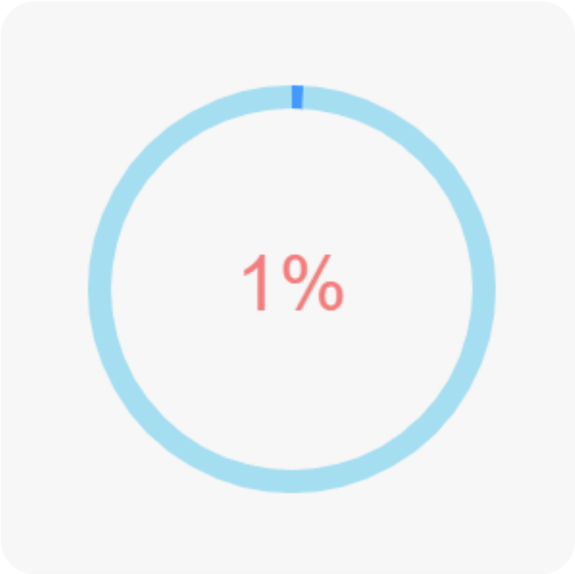
- Quantitative
  - Drop-off rate
  - Duration
- Qualitative
  - Perceived Performance
  - Emotional Response
  - Clarity

Metrics (for loading screen)

- Quantitative
  - Drop-off rate
  - Duration
- Qualitative
  - Perceived Performance
  - Emotional Response
  - Clarity

How to collect

- Analytics
- A/B tests
- User Research
- Survey



Researches (ex Facebook)

- Skeleton screen instead of spinner

	Change
Perceived loading time	+ ~30%
Retention/Engagement	+ ~10%
Satisfaction rate	+ ~20%

Disney's Principles of Animation

Why animations are important

SwiftUI Animations

Applying everything together

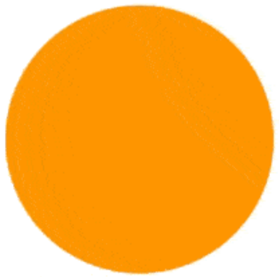
Disney's Principles of Animation

Why animations are important

SwiftUI Animations

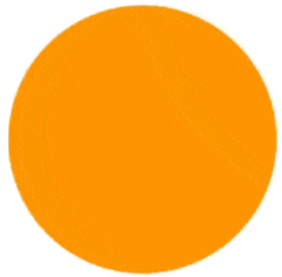
Applying everything together

```swift
struct KeynoteExampleScreen: View {
    @State private var isSquare = true

    var body: some View {
        VStack {
            Spacer()
            Rectangle()
                .fill(.orange)
                .clipShape(
                    RoundedRectangle(
                        cornerRadius: isSquare ? 30 : 50
                    )
                )
                .frame(
                    width: isSquare ? 140 : 100,
                    height: isSquare ? 140 : 100
                )
            Button("Toggle") {
                isSquare.toggle()
            }
        }
        .frame(
            maxWidth: .infinity,
            maxHeight: .infinity
        )
    }
}
```



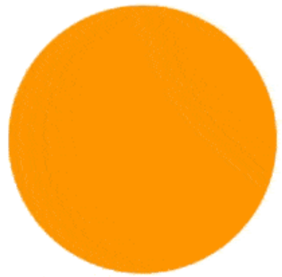
Toggle

```
struct KeynoteExampleScreen: View {
    @State private var isSquare = true

    var body: some View {
        VStack {
            Spacer()
            Rectangle()
                .fill(.orange)
                .clipShape(
                    RoundedRectangle(
                        cornerRadius: isSquare ? 30 : 50
                    )
                )
                .frame(
                    width: isSquare ? 140 : 100,
                    height: isSquare ? 140 : 100
                )
            Button("Toggle") {
                isSquare.toggle()
            }
        }
        .frame(
            maxWidth: .infinity,
            maxHeight: .infinity
        )
    }
}
```



Toggle

```swift
struct KeynoteExampleScreen: View {
    @State private var isSquare = true

    var body: some View {
        VStack {
            Spacer()
            Rectangle()
                .fill(.orange)
                .clipShape(
                    RoundedRectangle(
                        cornerRadius: isSquare ? 30 : 50
                    )
                )
                .frame(
                    width: isSquare ? 140 : 100,
                    height: isSquare ? 140 : 100
                )
            Button("Toggle") {
                isSquare.toggle()
            }
        }
        .frame(
            maxWidth: .infinity,
            maxHeight: .infinity
        )
    }
}
```



Toggle

1. Run animation

```
struct KeynoteExampleScreen: View {
    @State private var isSquare = true

    var body: some View {
        VStack {
            Spacer()
            Rectangle()
                .fill(.orange)
                .clipShape(...)
                .frame(
                    width: isSquare ? 140 : 100,
                    height: isSquare ? 140 : 100
                )
            Button("Toggle") {
                isSquare.toggle()
            }
        }
        .frame(...)
        .animation(.spring(...), value: isSquare)
    }
}
```


```
struct KeynoteExampleScreen: View {
    @State private var isSquare = true

    var body: some View {
        VStack {
            Spacer()
            Rectangle()
                .fill(.orange)
                .clipShape(...)
                .frame(
                    width: isSquare ? 140 : 100,
                    height: isSquare ? 140 : 100
                )
            Button("Toggle") {
                withAnimation(.spring(...)) {
                    isSquare.toggle()
                }
            }
        }
        .frame(...)
    }
}
```

```
struct KeynoteExampleScreen: View {
    @State private var isVisible = true

    var body: some View {
        VStack {
            Spacer()
            if isVisible {
                Rectangle()
                    .fill(.orange)
                    .clipShape(
                        RoundedRectangle(
                            cornerRadius: isVisible ? 30 : 50
                        )
                    )
                    .frame(
                        width: isVisible ? 140 : 100,
                        height: isVisible ? 140 : 100
                    )
            }
            Button("Toggle") {
                isVisible.toggle()
            }
        }
        .frame(
            maxWidth: .infinity,
            maxHeight: .infinity
        )
    }
}
```

Toggle

2. Add transition

```
struct KeynoteExampleScreen: View {
    @State private var isVisible = true

    var body: some View {
        VStack {
            Spacer()
            if isVisible {
                Rectangle()
                    .fill(.orange)
                    .clipShape(...)
                    .frame(...)
                    .transition(.slide.combined(with: .opacity))
            }
            Button("Toggle") {
                isVisible.toggle()
            }
        }
        .frame(...)
        .animation(.spring(...), value: isSquare)
    }
}
```

2.

1.

Toggle

withAnimation vs animation

	withAnimation {}	.animation()
Scope	**Global**	Specific View
Syntax	Imperative (Explicit)	Declarative (Implicit)

```
struct KeynoteExampleScreen: View {
    @State private var isSquare = true

    var body: some View {
        VStack {
            Spacer()
            Rectangle()
                .fill(.orange)
                .clipShape(...)
                .frame(
                    width: isSquare ? 140 : 100,
                    height: isSquare ? 140 : 100
                )
            Button("Toggle") {
                withAnimation(.spring(...)) {
                    isSquare.toggle()
                }
            }
        }
        .frame(...)
    }
}
```

withAnimation vs animation

	withAnimation {}	.animation()
Scope	Global	Specific View
Syntax	**Imperative (Explicit)**	Declarative (Implicit)

```
struct KeynoteExampleScreen: View {
    @State private var isSquare = true

    var body: some View {
        VStack {
            Spacer()
            Rectangle()
                .fill(.orange)
                .clipShape(...)
                .frame(
                    width: isSquare ? 140 : 100,
                    height: isSquare ? 140 : 100
                )
            Button("Toggle") {
                withAnimation(.spring(...)) {
                    isSquare.toggle()
                }
            }
        }
        .frame(...)
    }
}
```

withAnimation vs animation

	withAnimation {}	**.animation()**
Scope	Global	**Specific View**
Syntax	Imperative (Explicit)	Declarative (Implicit)

```
struct KeynoteExampleScreen: View {
    @State private var isSquare = true

    var body: some View {
        VStack {
            Spacer()
            Rectangle()
                .fill(.orange)
                .clipShape(...)
                .frame(...)
            Button("Toggle") {
                isSquare.toggle()
            }
        }
        .frame(...)
        .animation(.spring(...), value: isSquare)
    }
}
```

withAnimation vs animation

	withAnimation {}	**.animation()**
Scope	Global	**Specific View**
Syntax	Imperative (Explicit)	Declarative (Implicit)

```
struct KeynoteExampleScreen: View {
    @State private var isSquare = true

    var body: some View {
        VStack {
            Spacer()
            Rectangle()
                .fill(.orange)
                .clipShape(...)
                .frame(...)
                .animation(
                    .spring(...),
                    value: isSquare
                )
            Button("Toggle") {
                isSquare.toggle()
            }
        }
        .frame(...)
    }
}
```

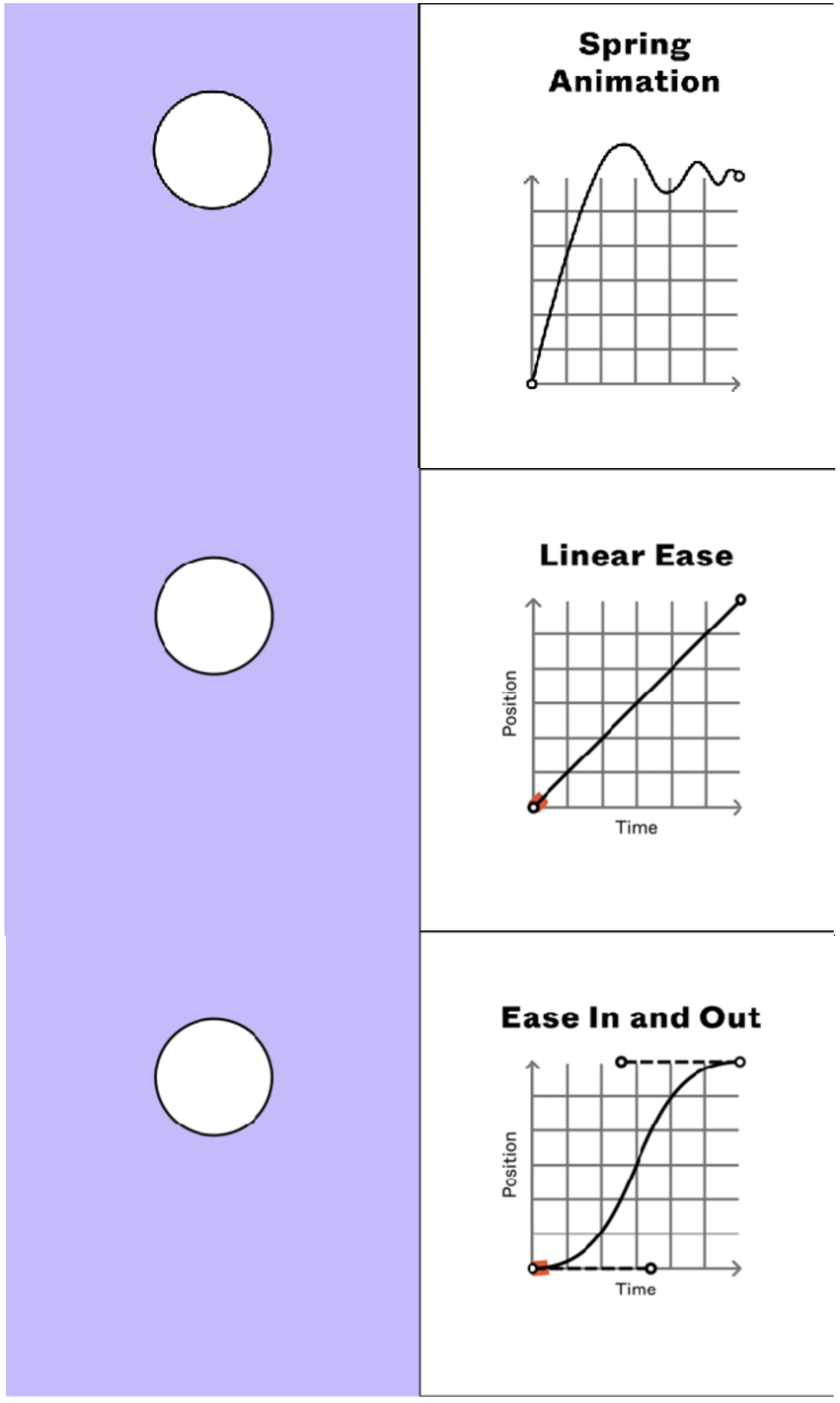
withAnimation vs animation

	withAnimation {}	**.animation()**
Scope	Global	Specific View
Syntax	Imperative (Explicit)	**Declarative (Implicit)**

```
struct KeynoteExampleScreen: View {
    @State private var isSquare = true

    var body: some View {
        VStack {
            Spacer()
            Rectangle()
                .fill(.orange)
                .clipShape(...)
                .frame(...)
            Button("Toggle") {
                isSquare.toggle()
            }
        }
        .frame(...)
        .animation(.spring(...), value: isSquare)
    }
}
```

Timing

- Spring
- EaseInOut/In/Out
- Linear



Specifics

- Position of animation method matters (sometimes)
- Some containers animating their content very poor (ScrollView/LazyStack/LazyGrid)
- Simulator/Previews could have different behaviour

Disney's Principles of Animation

Why animations are important

SwiftUI Animations

Applying all together

Disney's Principles of Animation

Why animations are important

SwiftUI Animations

Applying all together

Demo

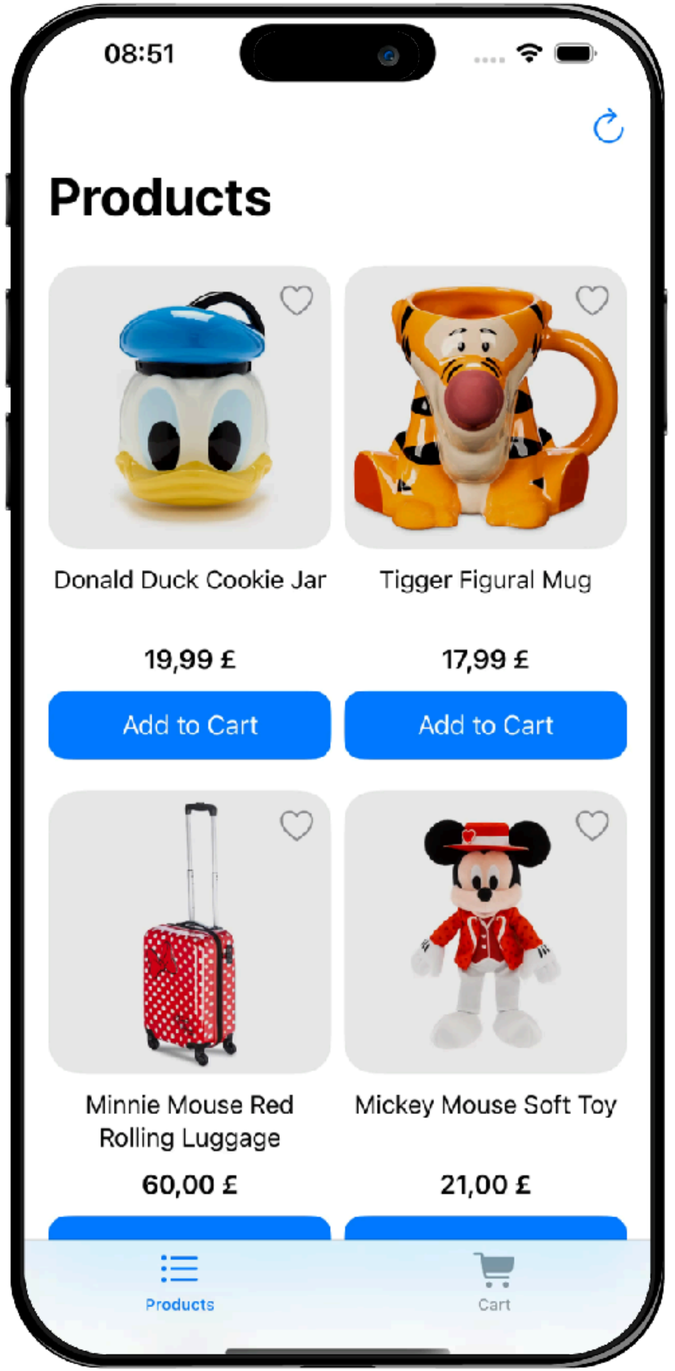
- Waiting on the Launch Screen
- Adding items to the basket
- Adding items to favourites
- Starting checkout process in the basket tab

Demo. Visual Issues

- Long startup time
- Hard to tell if an item was added to cart
- No visual feedback during checkout flow
- **Lacks visual coherence and smoothness**

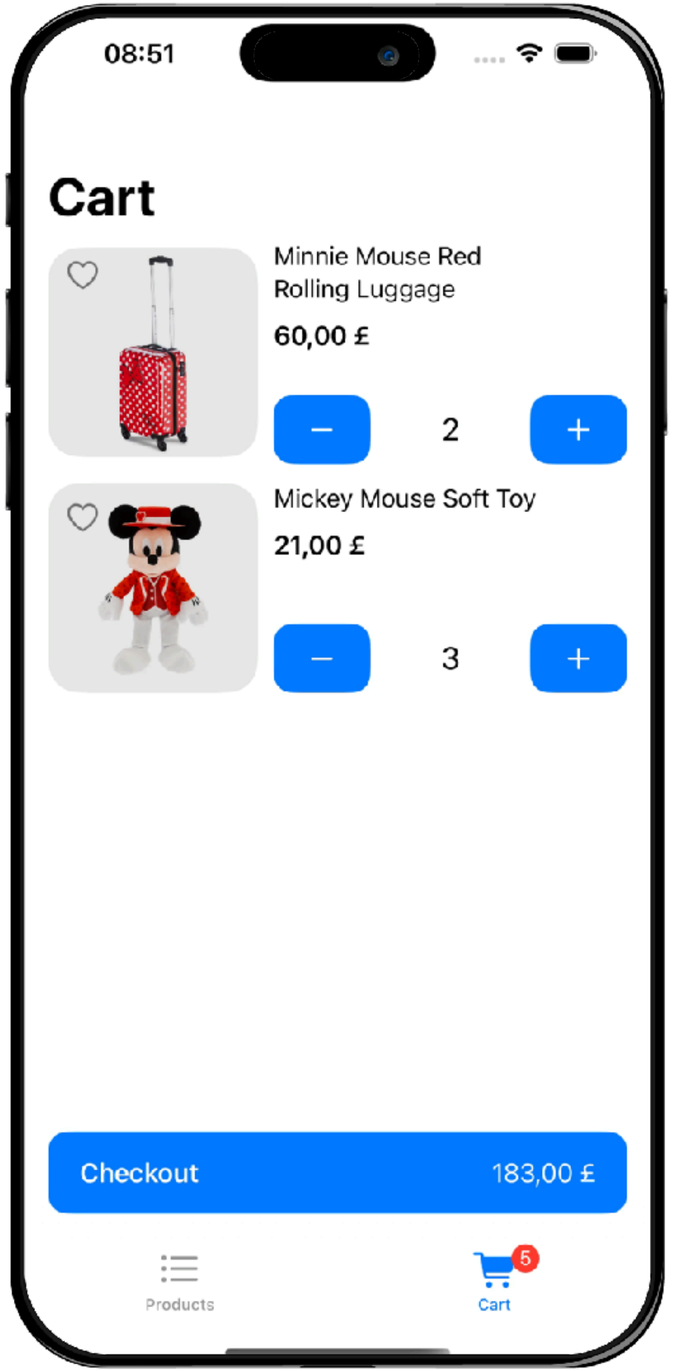
Demo. Visual Issues

- Long startup time
- Hard to tell if an item was added to cart
- No visual feedback during checkout flow
- **Lacks visual coherence and smoothness**



Demo. Visual Issues

- Long startup time
- Hard to tell if an item was added to cart
- No visual feedback during checkout flow
- Lacks visual coherence and smoothness

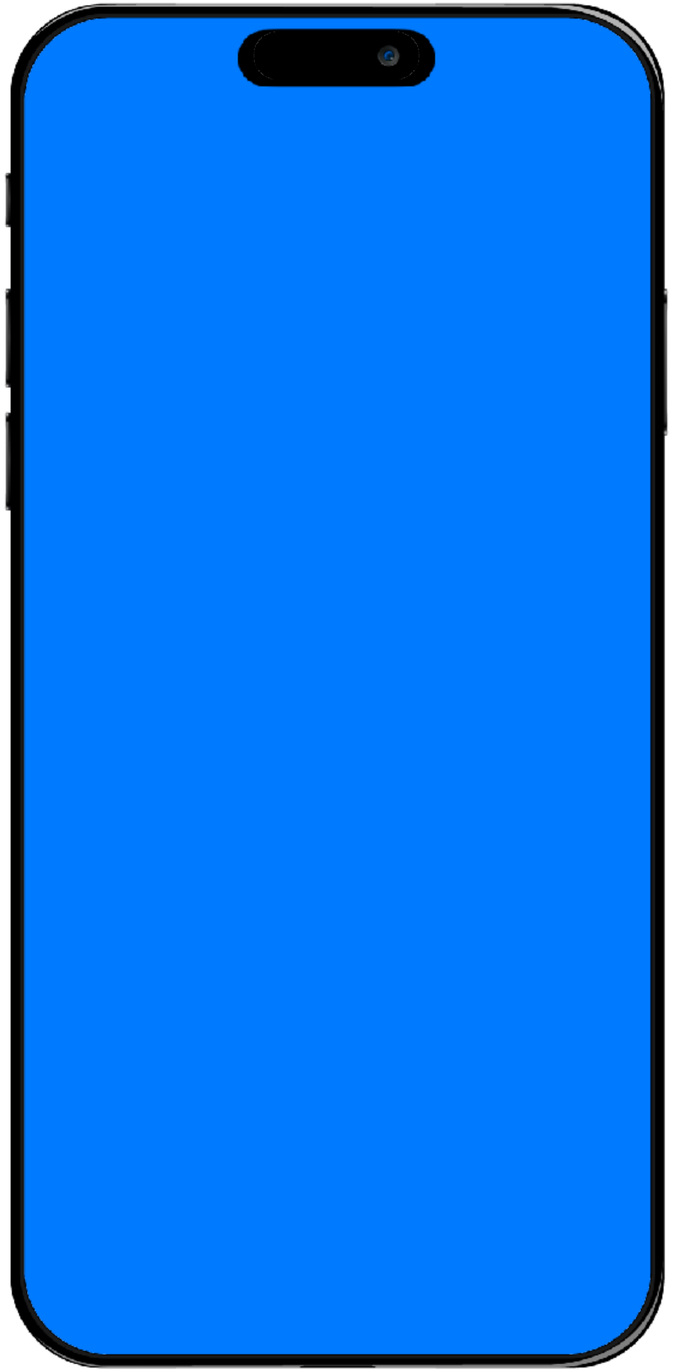


Demo. Visual Issues

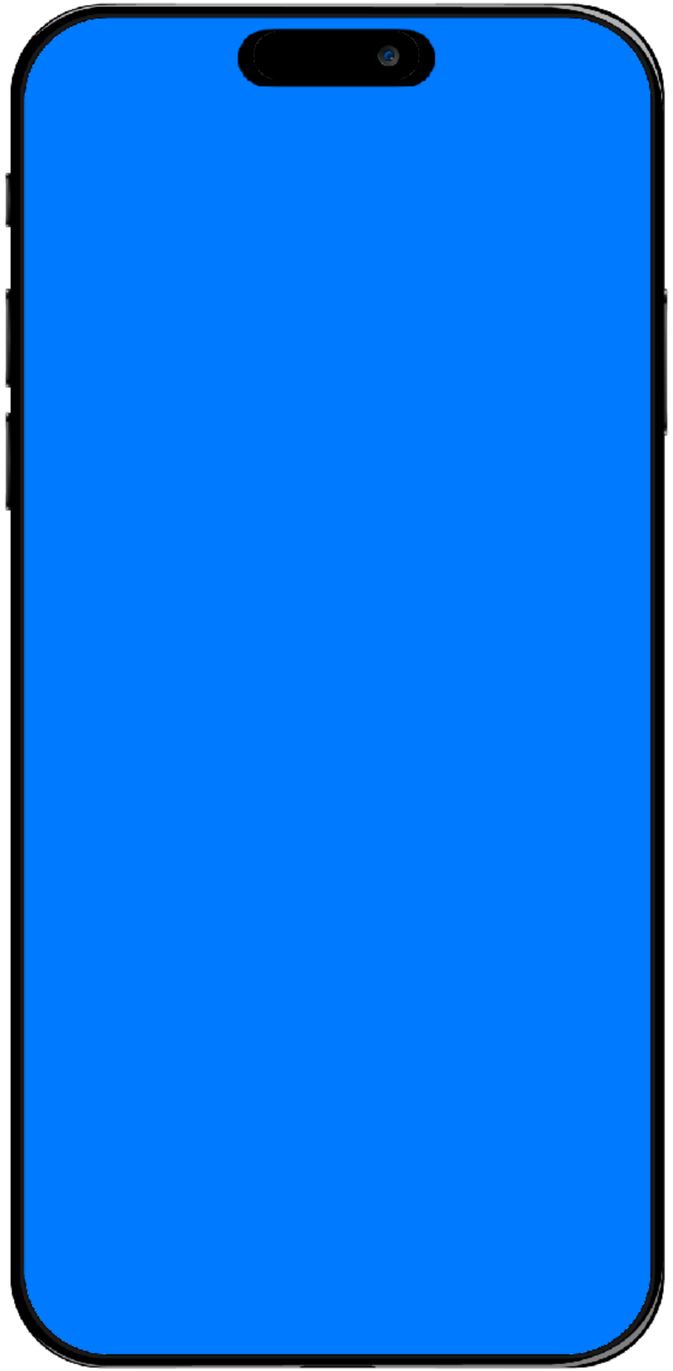
- Long startup time
- Hard to tell if an item was added to cart
- No visual feedback during checkout flow
- **Lacks visual coherence and smoothness**

Launch Screen

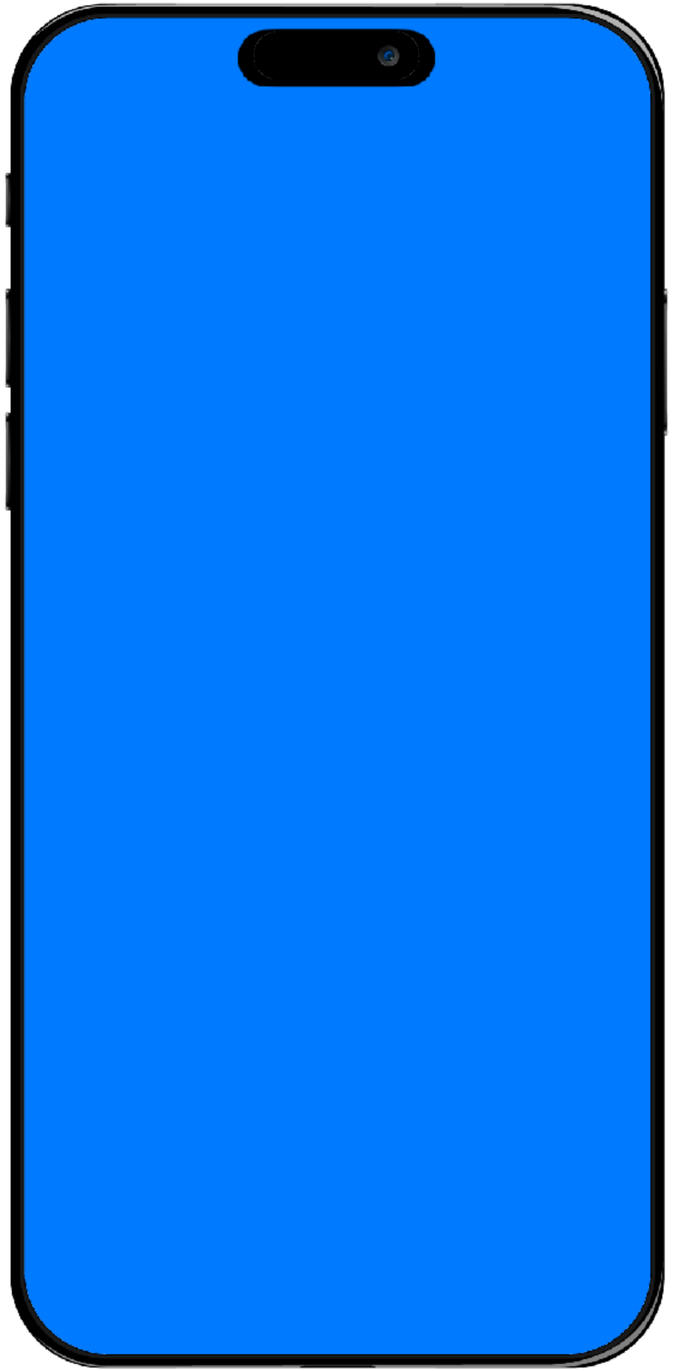
```
struct LaunchScreen<Content: View>: View {
    private let contentView: () -> Content
    @Binding var isFinished: Bool

    var body: some View {
        ZStack {
            contentView()
                .ignoresSafeArea()
            Rectangle()
                .fill(.blue)
                .opacity(isFinished ? 0 : 1)
                .ignoresSafeArea()
        }
    }
}
```



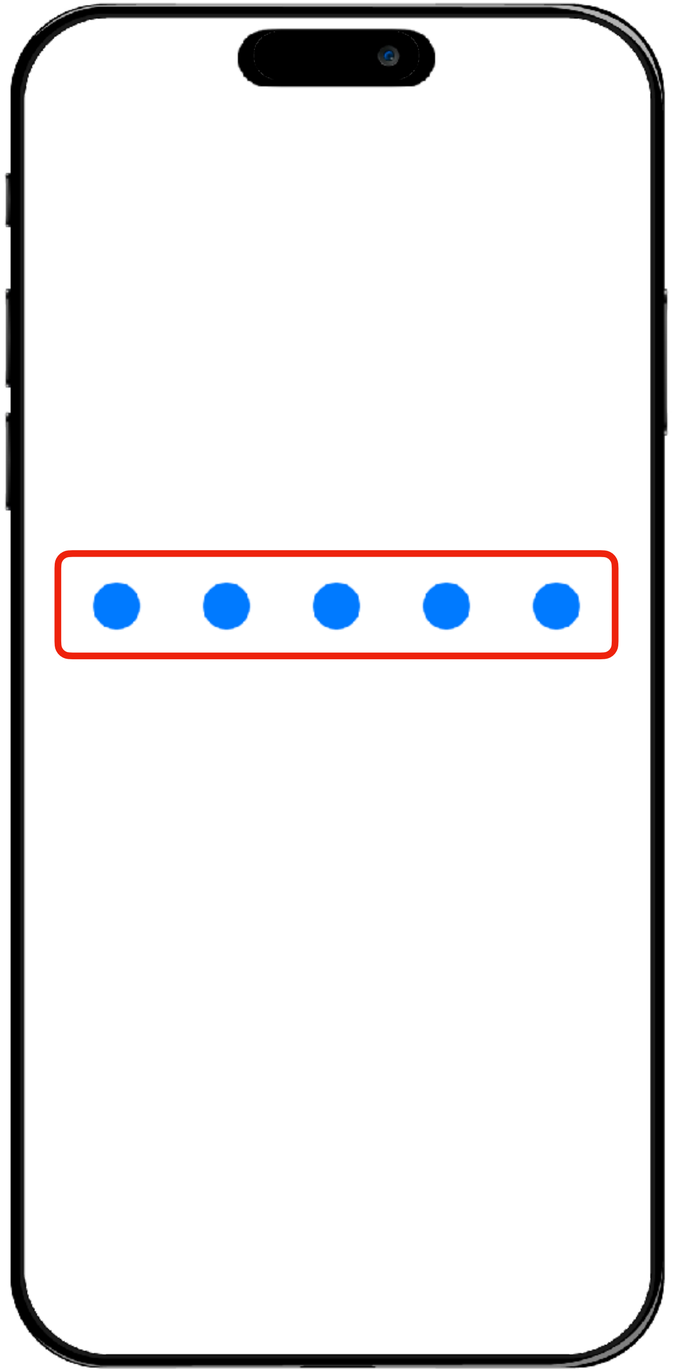
```
struct LaunchScreen<Content: View>: View {
    private let contentView: () -> Content
    @Binding var isFinished: Bool

    var body: some View {
        ZStack {
            contentView()
                .ignoresSafeArea()
            Rectangle()
                .fill(.blue)
                .opacity(isFinished ? 0 : 1)
                .ignoresSafeArea()
        }
    }
}
```



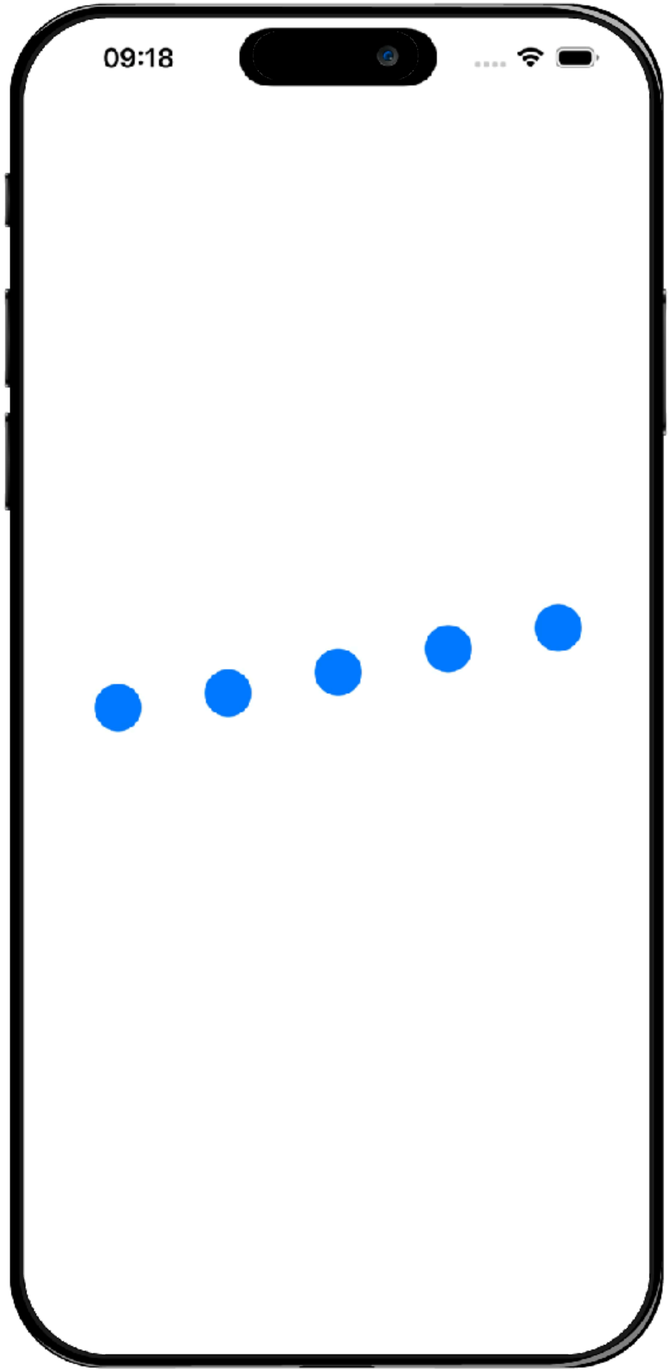
```
struct LaunchScreen<Content: View>: View {
    private let contentView: () -> Content
    @Binding var isFinished: Bool

    var body: some View {
        ZStack {
            contentView()
                .ignoresSafeArea()
            Rectangle()
                .fill(.blue)
                .opacity(isFinished ? 0 : 1)
                .ignoresSafeArea()
        }
    }
}
```



```swift
struct LaunchScreen<Content: View>: View {
    private let contentView: () -> Content
    @Binding var isFinished: Bool

    var body: some View {
        ZStack {
            contentView()
                .ignoresSafeArea()
            Rectangle()
                .fill(.blue)
                .opacity(isFinished ? 0 : 1)
                .ignoresSafeArea()
        }
        .mask {
            HStack(spacing: 40) {
                ForEach(0..<5) { index in
                    Circle()
                        .fill(Color.green)
                        .frame(width: 30, height: 30)
                }
            }
        }
    }
}
```



```swift
struct LaunchScreen<Content: View>: View {
    ...

    @State private var isAnimating = false

    var body: some View {
        ZStack { ... }
        .mask {
            HStack(spacing: 40) {
                ForEach(0..<5) { index in
                    Circle()
                        .fill(Color.green)
                        .frame(width: 30, height: 30)
                        .offset(y: isAnimating ? 0 : 70)

                        .animation(
                            Animation.easeInOut(duration: 0.8)
                                .repeatForever()
                                .delay(Double(index) * 0.1),
                            value: isAnimating
                        )
                }
            }
        }
        .onAppear {
            isAnimating = true
        }
    }
}
```

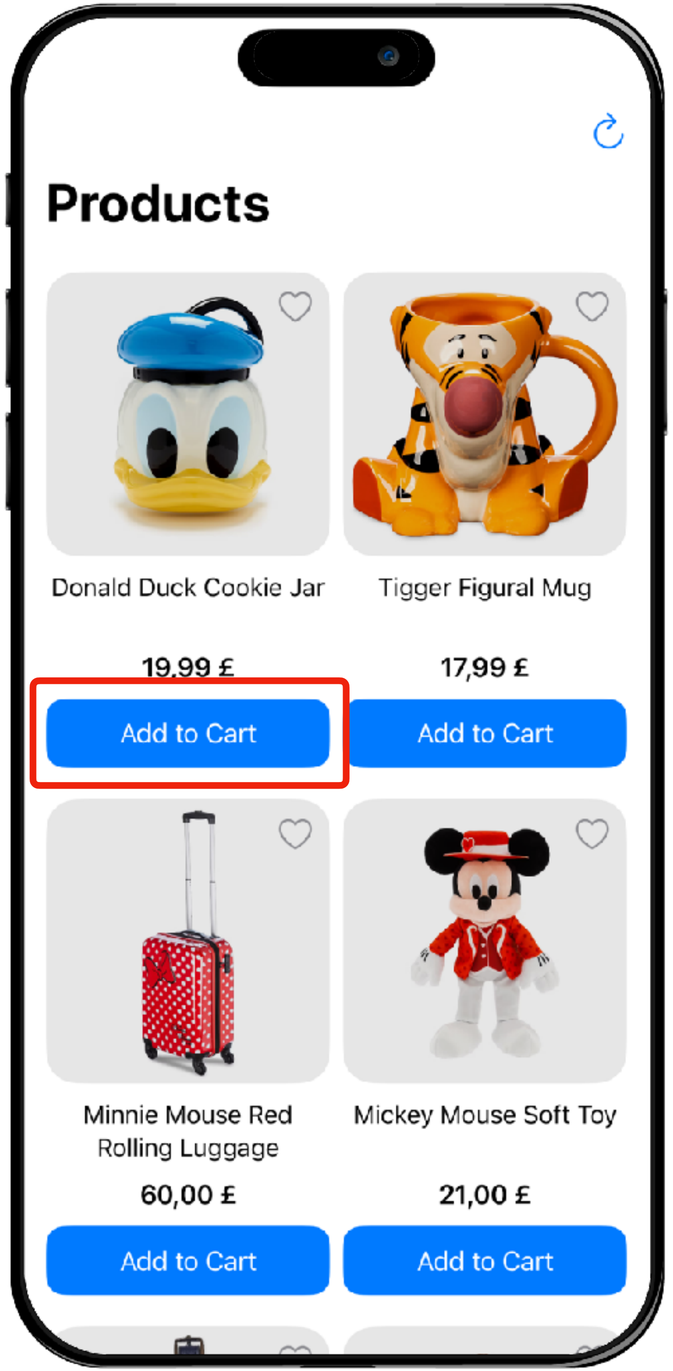


```
struct LaunchScreen<Content: View>: View {
    ...
    @Namespace var namespace

    var body: some View {
        ZStack { ... }
        .mask {
            ZStack {
                if isFinished {
                    Circle()
                        .fill(Color.blue)
                        .matchedGeometryEffect(
                            id: "2-circle",
                            in: namespace
                        )
                        .frame(width: 1000, height: 1000)
                } else {
                    HStack(spacing: 40) {
                        ForEach(0..<5) { index in
                            Circle()
                                .matchedGeometryEffect(
                                    id: "\(index)-circle",
                                    in: namespace
                                )
                                ...
                        }
                    }
                }
            }
        }
        .onAppear { ... }
        .animation(.easeInOut, value: isFinished)
    }
}
```

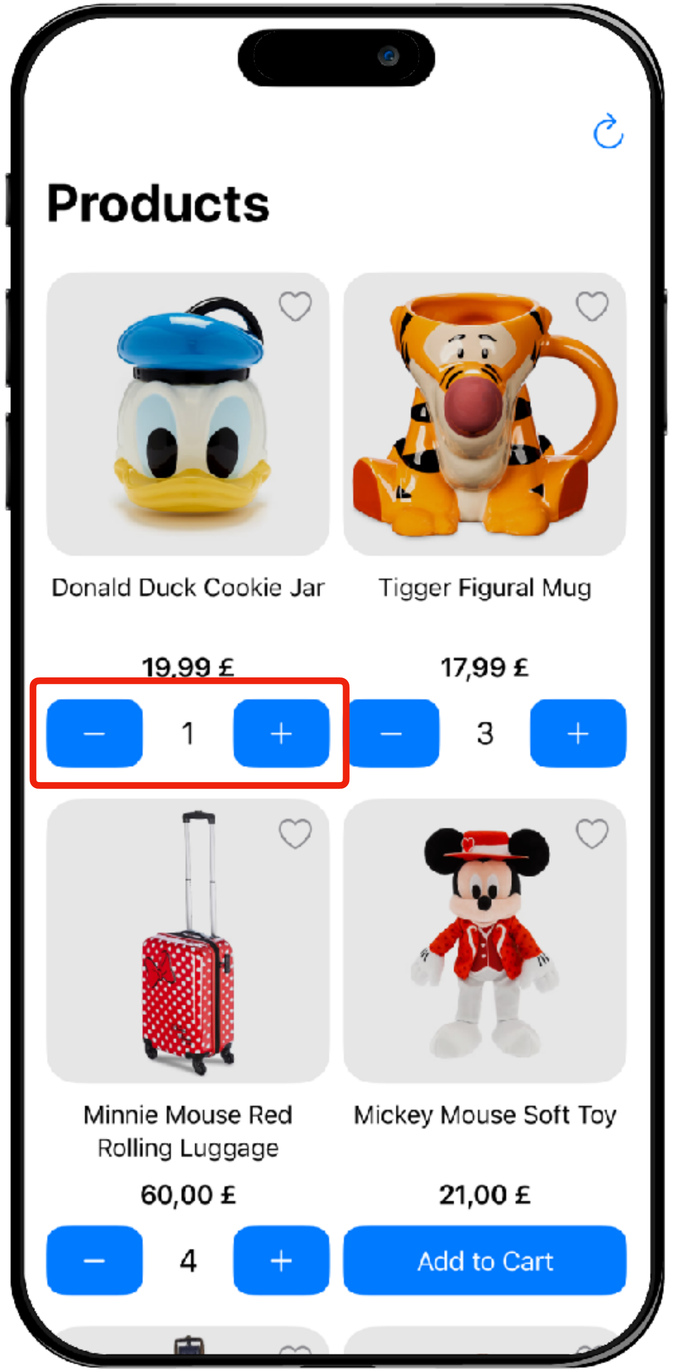
```
struct LaunchScreen<Content: View>: View {
    ...
    @Namespace var namespace

    var body: some View {
        ZStack { ... }
        .mask {
            ZStack {
                if isFinished {
                    Circle()
                        .fill(Color.blue)
                        .matchedGeometryEffect(
                            id: "2-circle",
                            in: namespace
                        )
                        .frame(width: 1000, height: 1000)
                } else {
                    HStack(spacing: 40) {
                        ForEach(0..<5) { index in
                            Circle()
                                .matchedGeometryEffect(
                                    id: "\(index)-circle",
                                    in: namespace
                                )
                                ...
                        }
                    }
                }
            }
        }
        .onAppear { ... }
        .animation(.easeInOut, value: isFinished)
    }
}
```

Cart Button

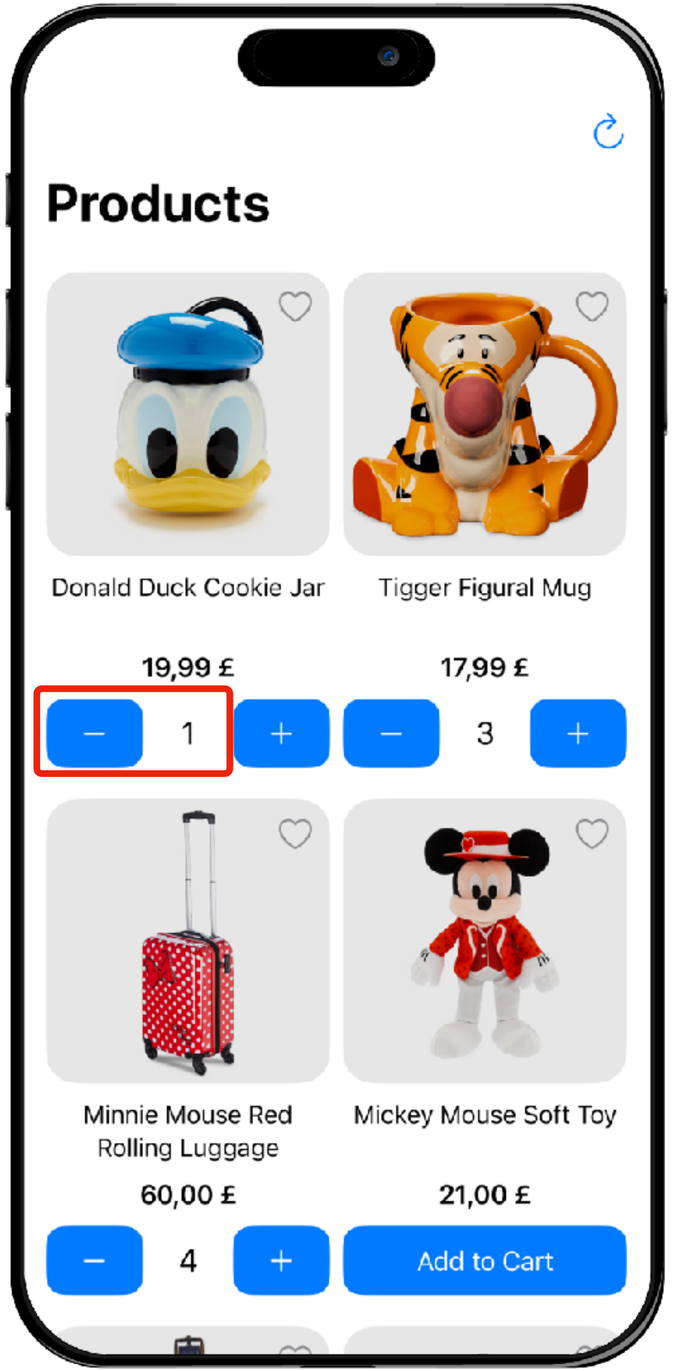
```
struct CartButtons: View {
    let onAction: () -> Void

    var body: some View {
        Button(action: onAction) {
            Text("Add to Cart")
                .frame(maxWidth: .infinity)
                .frame(height: 30)
        }
        .buttonStyle(.borderedProminent)
        .buttonBorderShape(.roundedRectangle(radius: 12))
    }
}
```



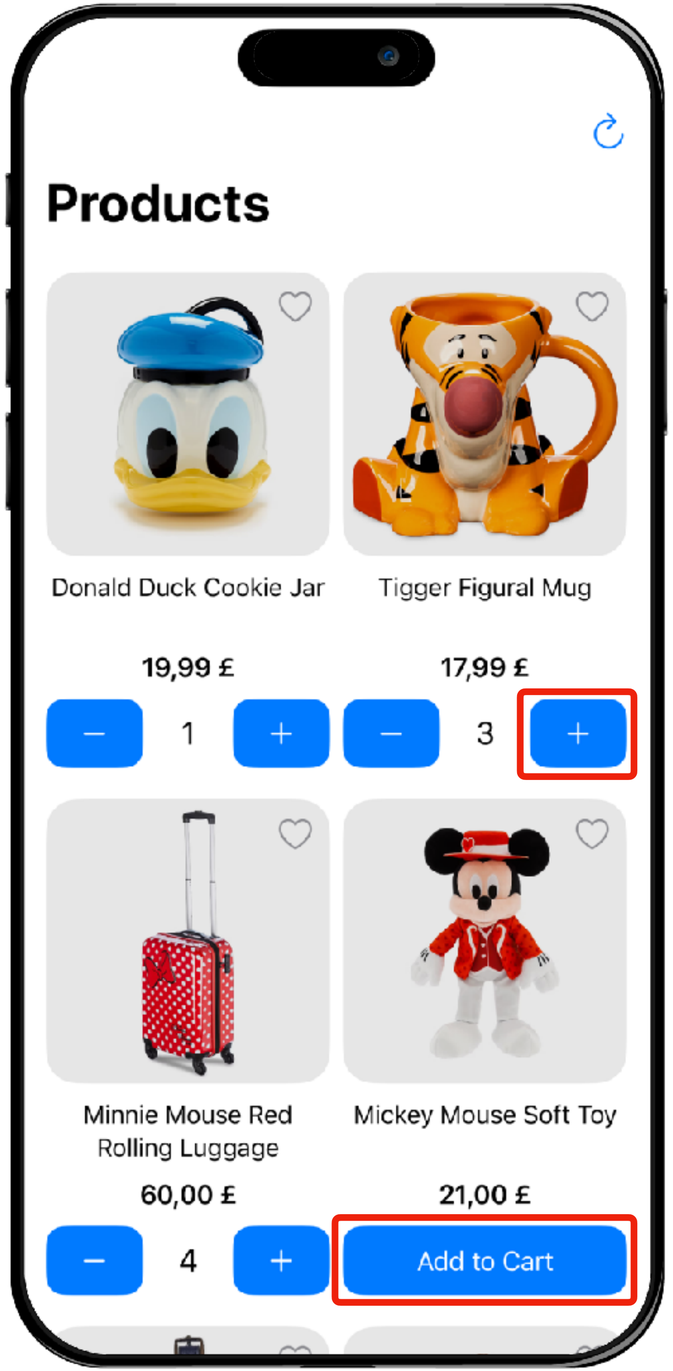
```swift
struct CartButtonsView: View {
    enum Action {
        case increase, decrease, toCart
    }
    let onAction: (Action) -> Void
    let count: Int

    var body: some View {
        HStack {
            if count > 0 {
                Button {
                    onAction(.decrease)
                } label: {
                    Image(systemName: "minus")
                }
                Text(count.description)
            }

            Button {
                if count > 0 { onAction(.increase) }
                else { onAction(.toCart) }
            } label: {
                if count > 0 {
                    Image(systemName: "plus")
                } else {
                    Text("Add to Cart")
                }
            }
        }
    }
}
```



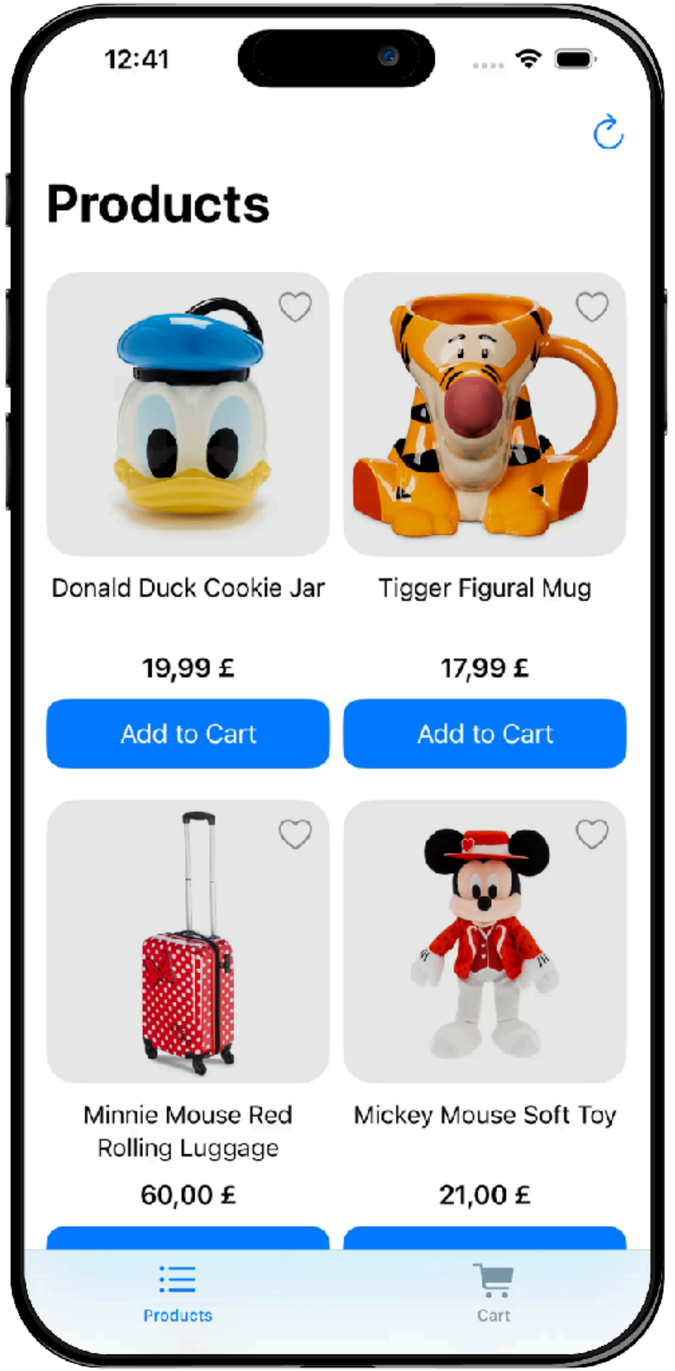
```swift
struct CartButtonsView: View {
    enum Action {
        case increase, decrease, toCart
    }
    let onAction: (Action) -> Void
    let count: Int

    var body: some View {
        HStack {
            if count > 0 {
                Button {
                    onAction(.decrease)
                } label: {
                    Image(systemName: "minus")
                }
                Text(count.description)
            }

            Button {
                if count > 0 { onAction(.increase) }
                else { onAction(.toCart) }
            } label: {
                if count > 0 {
                    Image(systemName: "plus")
                } else {
                    Text("Add to Cart")
                }
            }
        }
    }
}
```



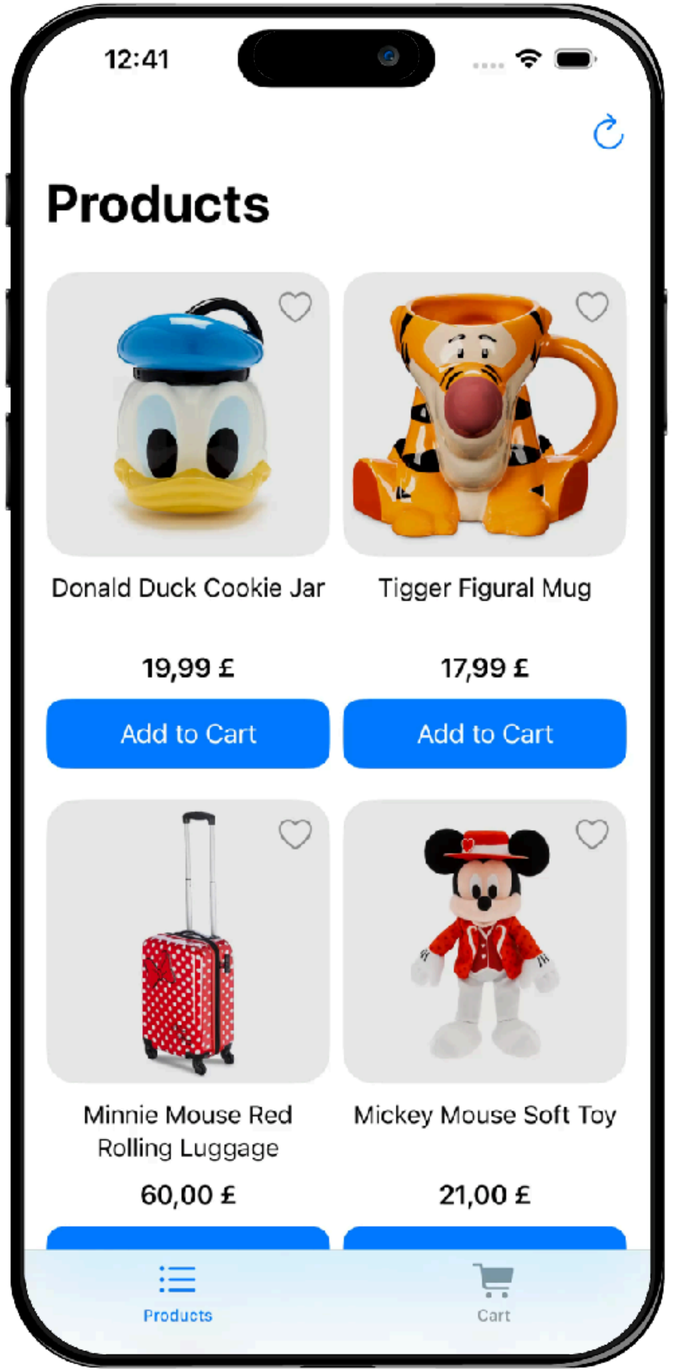
```
struct CartButtonsView: View {
    enum Action {
        case increase, decrease, toCart
    }
    let onAction: (Action) -> Void
    let count: Int

    var body: some View {
        HStack {
            if count > 0 {
                Button {
                    onAction(.decrease)
                } label: {
                    Image(systemName: "minus")
                }
                Text(count.description)
            }

            Button {
                if count > 0 { onAction(.increase) }
                else { onAction(.toCart) }
            } label: {
                if count > 0 {
                    Image(systemName: "plus")
                } else {
                    Text("Add to Cart")
                }
            }
        }
    }
}
```



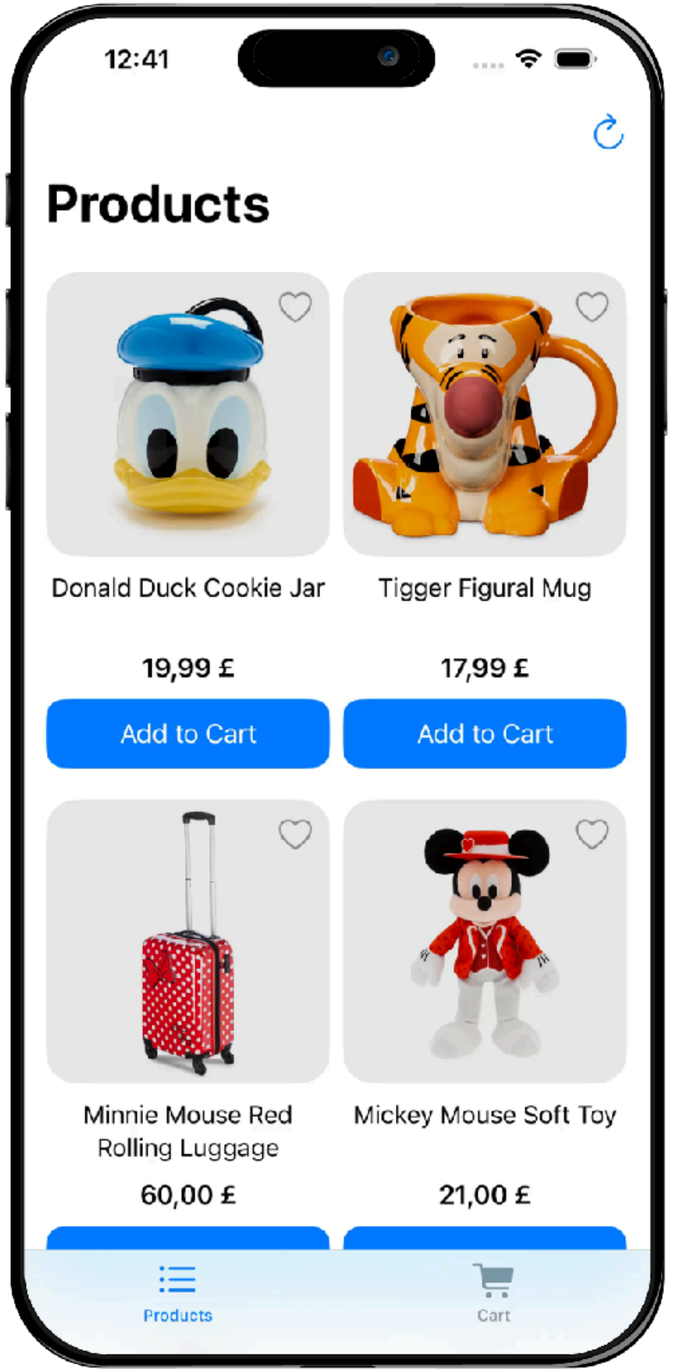
```
struct CartButtonsView: View {
    var body: some View {
        HStack {
            if count > 0 {
                Button {
                    onAction(.decrease)
                } label: {
                    Image(systemName: "minus")
                }
                .transition(...)
                Text(count.description)
                    .contentTransition(.numericText())
                    .transition(...)
            }

            Button {
                if count > 0 { onAction(.increase) }
                else { onAction(.toCart) }
            } label: {
                if count > 0 {
                    Image(systemName: «plus")
                        .transition(...)
                } else {
                    Text("Add to Cart»)
                        .transition(…)
                }
            }
        }
        .animation(.spring(…), value: count)
    }
}
```



```
struct CartButtonsView: View {
    var body: some View {
        HStack {
            if count > 0 {
                Button {
                    onAction(.decrease)
                } label: {
                    Image(systemName: "minus")
                }
                .transition(...)
                Text(count.description)
                    .contentTransition(.numericText())
                    .transition(...)
            }

            Button {
                if count > 0 { onAction(.increase) }
                else { onAction(.toCart) }
            } label: {
                if count > 0 {
                    Image(systemName: «plus")
                        .transition(...)
                } else {
                    Text("Add to Cart»)
                        .transition(…)
                }
            }
        }
        .animation(.spring(…), value: count)
    }
}
```

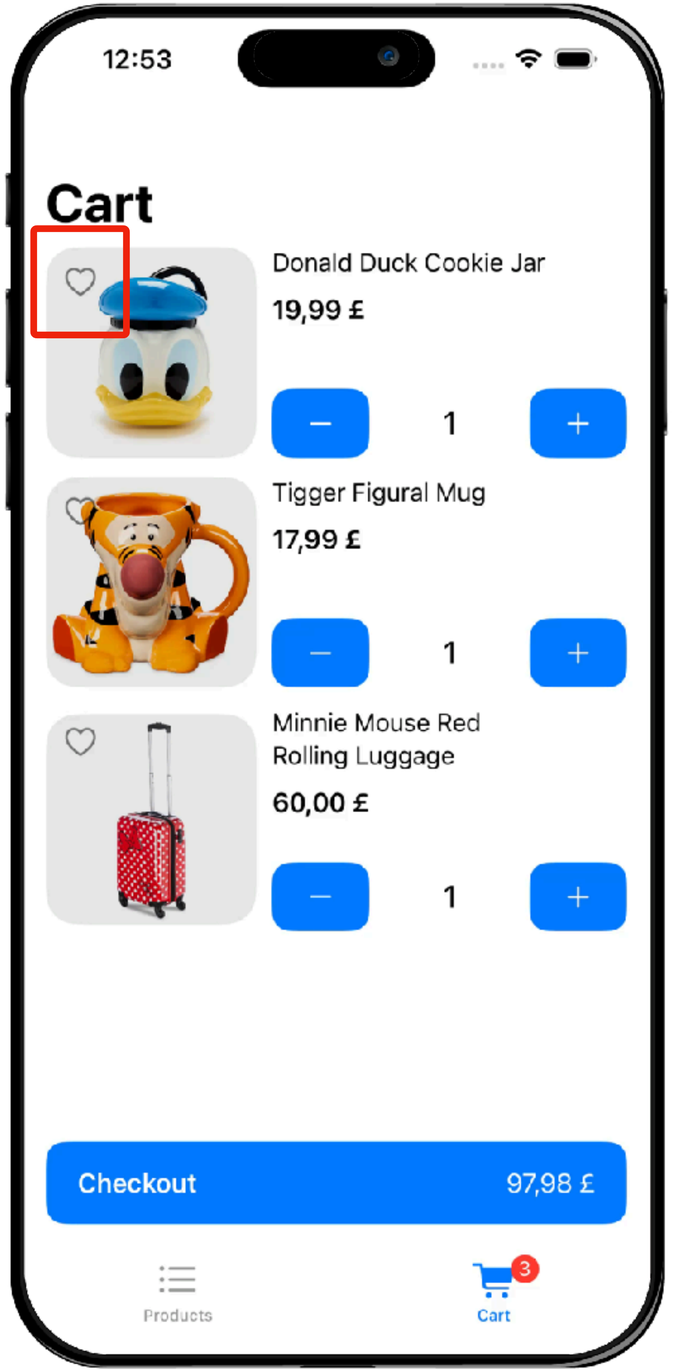


```
struct CartButtonsView: View {
    var body: some View {
        HStack {
            if count > 0 {
                Button {
                    onAction(.decrease)
                } label: {
                    Image(systemName: "minus")
                }
                .transition(...)
                Text(count.description)
                    .contentTransition(.numericText())
                    .transition(...)
            }

            Button {
                if count > 0 { onAction(.increase) }
                else { onAction(.toCart) }
            } label: {
                if count > 0 {
                    Image(systemName: «plus")
                        .transition(...)
                } else {
                    Text("Add to Cart»)
                        .transition(…)
                }
            }
        }
        .animation(.spring(…), value: count)
    }
}
```

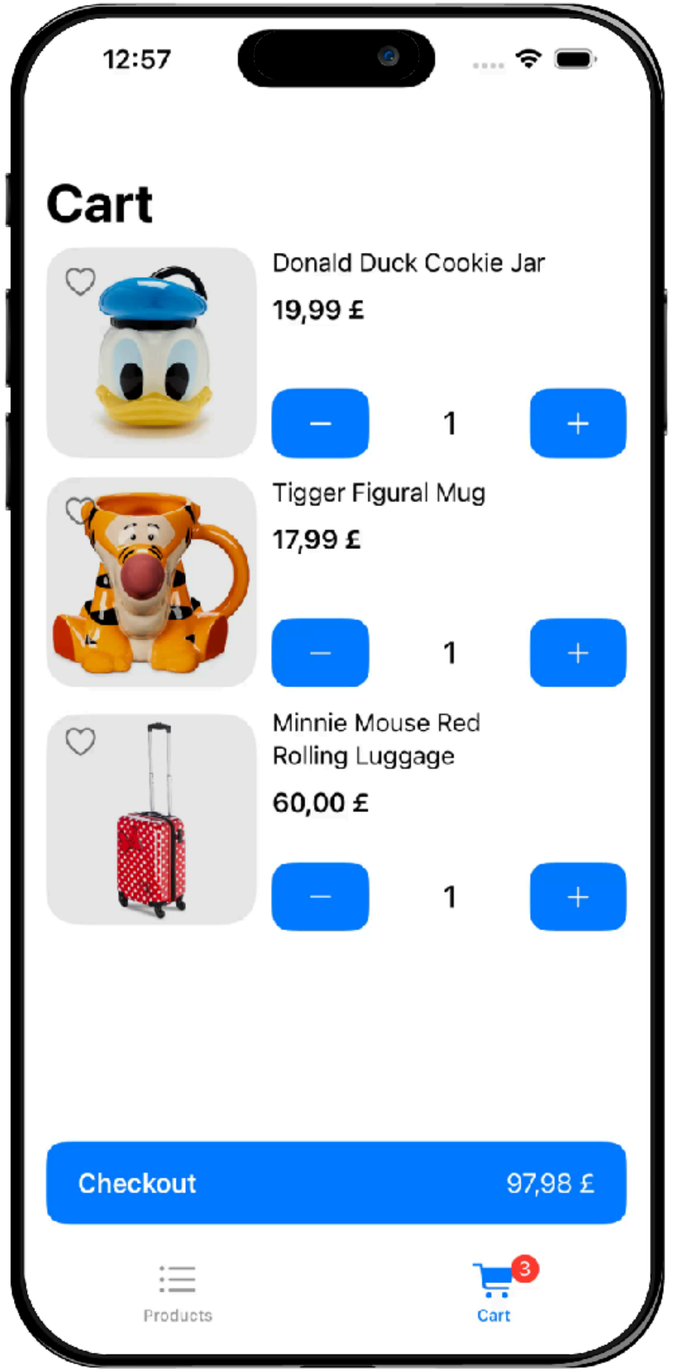


Like Button

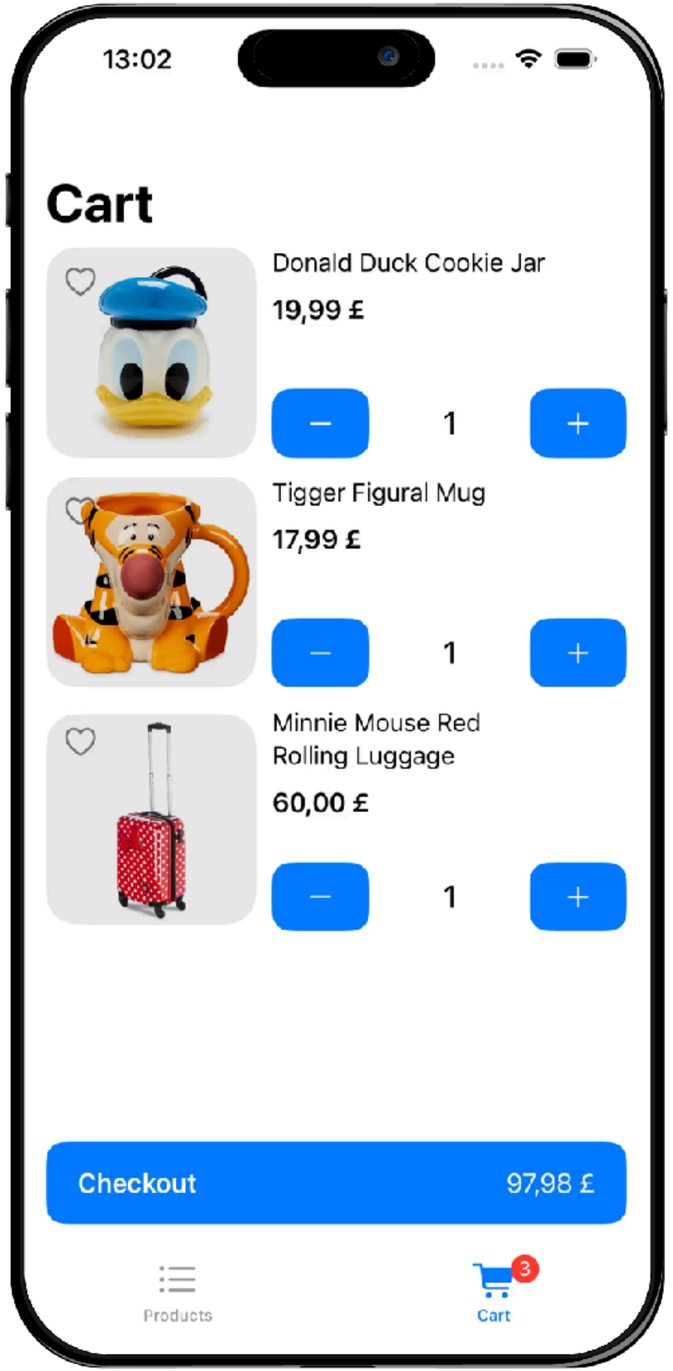
```
struct LikeButton: View {
    let onTap: () -> Void
    let status: Bool

    var body: some View {
        Button(action: onTap) {
            Image(
                systemName: status
                    ? "heart.fill"
                    : "heart"
            )
            .font(.title3)
            .foregroundColor(
                status
                    ? .red
                    : .black.opacity(0.5)
            )
            .padding(.horizontal, 18)
            .padding(.vertical, 8)
        }
        .buttonStyle(.plain)
    }
}
```



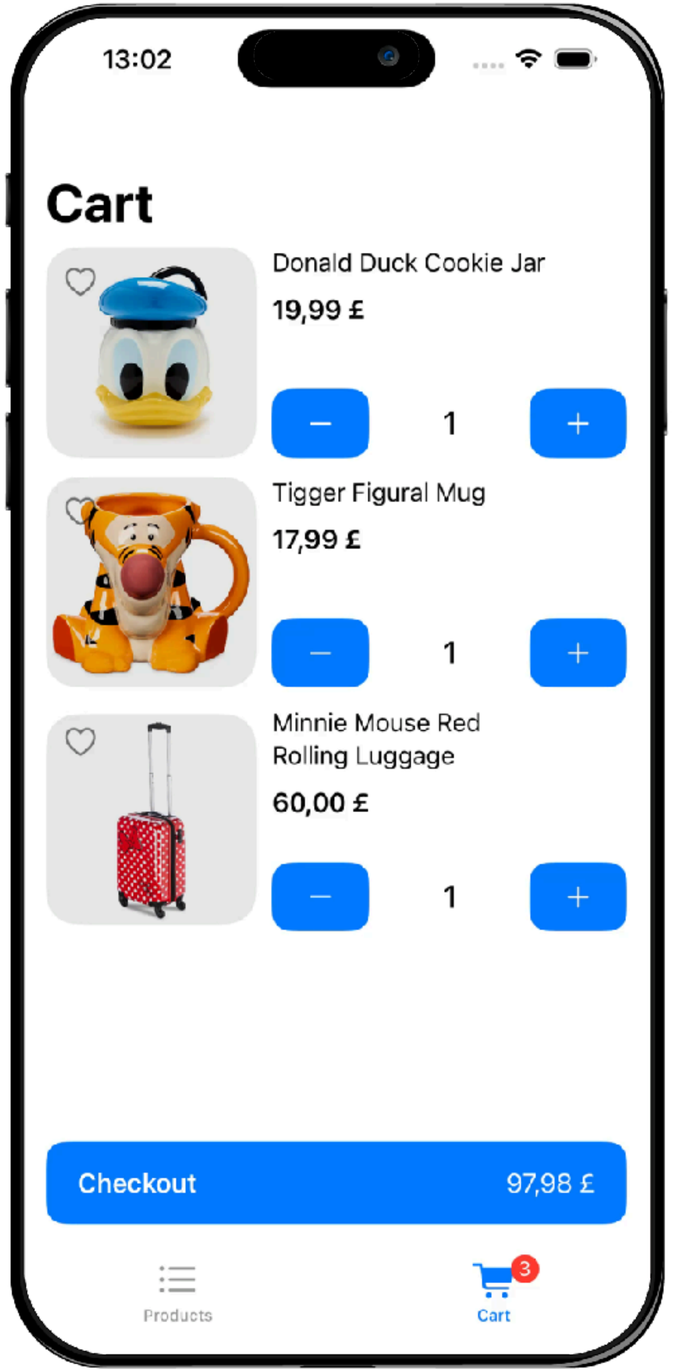
```swift
struct LikeButton: View {
    let onTap: () -> Void
    let status: Bool

    var body: some View {
        Button(action: onTap) {
            Image(
                systemName: status
                    ? "heart.fill"
                    : "heart"
            )
            .font(.title3)
            .foregroundColor(
                status
                    ? .red
                    : .black.opacity(0.5)
            )
            .padding(.horizontal, 18)
            .padding(.vertical, 8)
            .contentTransition(.symbolEffect(.replace))
            .animation(.easeInOut, value: status)
        }
        .buttonStyle(.plain)
    }
}
```



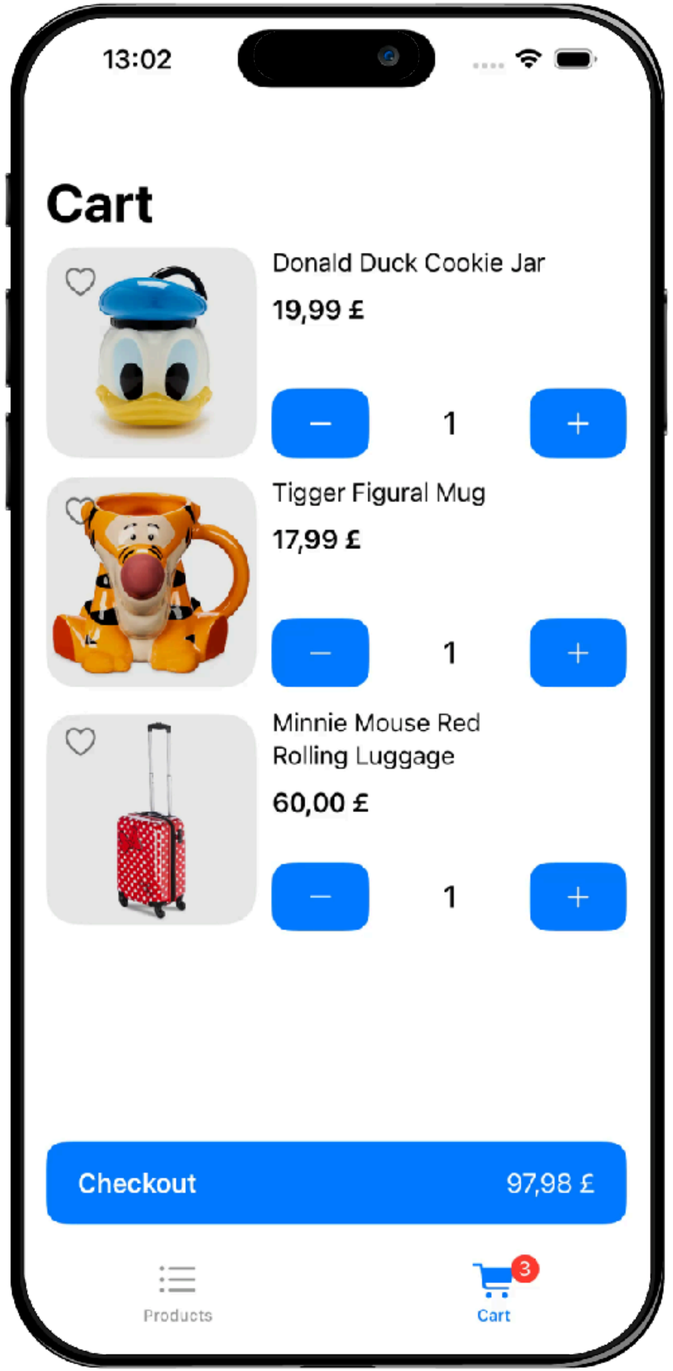
```swift
struct LikeButton: View {
    var body: some View {
        Button(action: onTap) {
            Image(
                systemName: status
                    ? "heart.fill"
                    : "heart"
            )
            .font(.title3)
            .foregroundColor(
                status
                    ? .red
                    : .black.opacity(0.5)
            )
            .particleEffect(
                systemImage: "heart.fill",
                font: .title3,
                status: status,
                activeTint: .red,
                inActiveTint: .black.opacity(0.5)
            )
            .padding(.horizontal, 18)
            .padding(.vertical, 8)
            .contentTransition(.symbolEffect(.replace))
            .animation(.easeInOut, value: status)
        }
        .buttonStyle(.plain)
    }
}
```



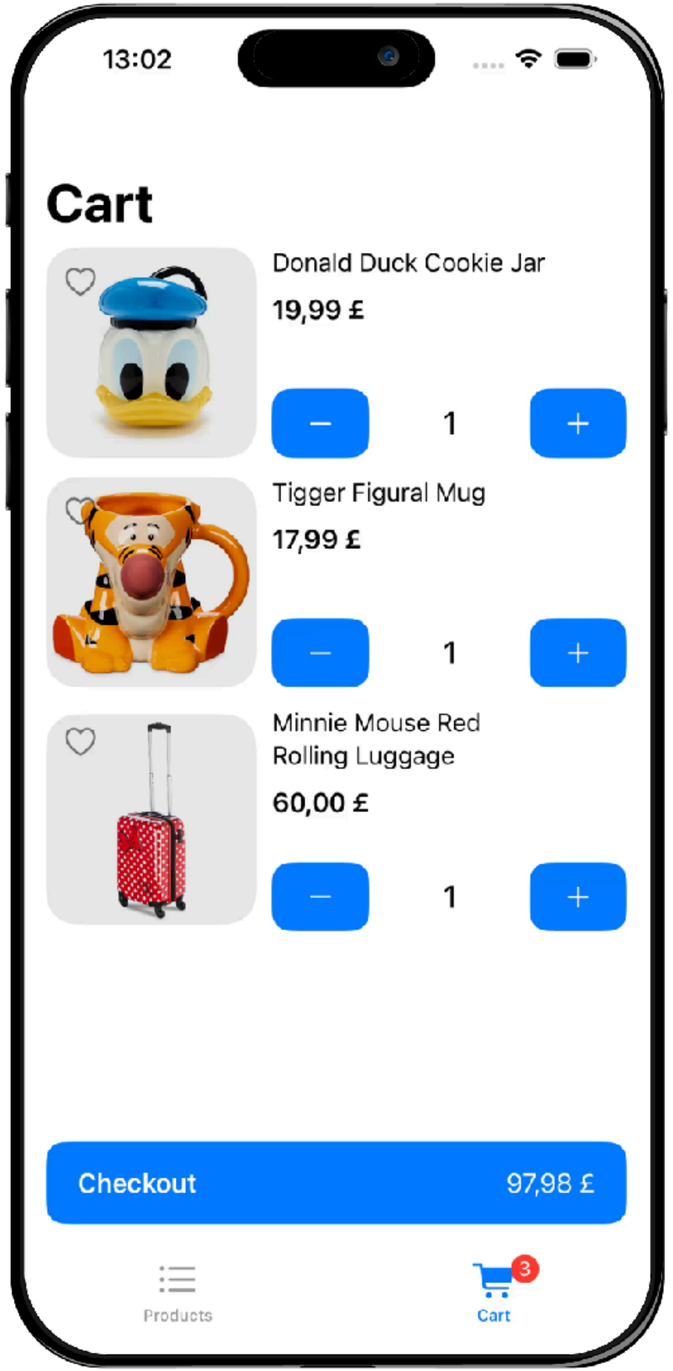
```swift
private struct ParticleEffectModifier: ViewModifier {
    @State private var particles: [Particle] = []

    func body(content: Content) -> some View {
        content.overlay(alignment: .top) {
            ZStack {
                ForEach(particles, id: \.id) { particle in
                    Image(systemName: systemImage)
                        .foregroundColor(...)
                        .scaleEffect(particle.scale)
                        .offset(x: particle.x, y: particle.y)
                        .opacity(particle.opacity)

                        .opacity(status ? 1 : 0)
                        .animation(.none, value: status)
                }
            }
            .onAppear {
                if particles.isEmpty {
                    particles = (0..<10).map { _ in Particle() }
                }
            }
            .onChange(of: status) {
                // run animations
            }
        }
    }
}
```



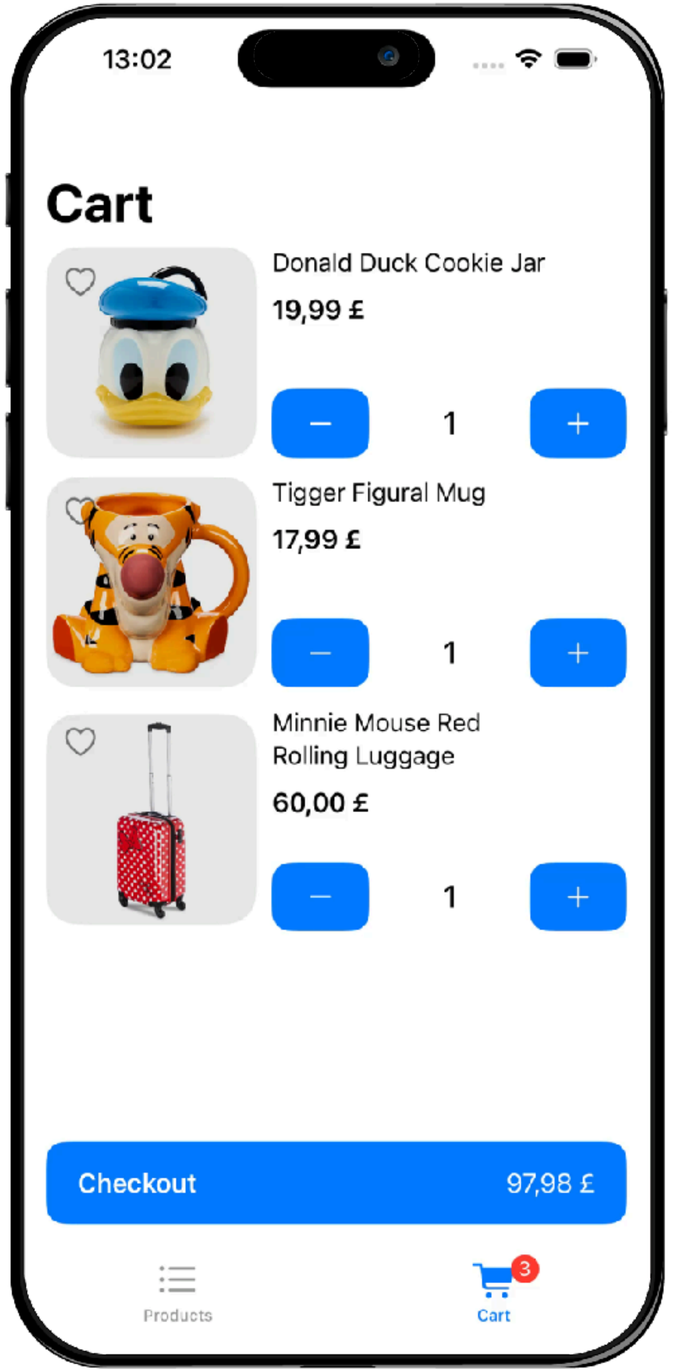
```swift
private struct ParticleEffectModifier: ViewModifier {
    @State private var particles: [Particle] = []

    func body(content: Content) -> some View {
        content.overlay(alignment: .top) {
            ZStack {
                ForEach(particles, id: \.id) { particle in
                    Image(systemName: systemImage)
                        .foregroundColor(...)
                        .scaleEffect(particle.scale)
                        .offset(x: particle.x, y: particle.y)
                        .opacity(particle.opacity)

                        .opacity(status ? 1 : 0)
                        .animation(.none, value: status)
                }
            }
            .onAppear {
                if particles.isEmpty {
                    particles = (0..<10).map { _ in Particle() }
                }
            }
            .onChange(of: status) {
                // run animations
            }
        }
    }
}
```



```swift
private struct ParticleEffectModifier: ViewModifier {
    @State private var particles: [Particle] = []

    func body(content: Content) -> some View {
        content.overlay(alignment: .top) {
            ZStack {
                ForEach(particles, id: \.id) { particle in
                    Image(systemName: systemImage)
                        .foregroundColor(...)
                        .scaleEffect(particle.scale)
                        .offset(x: particle.x, y: particle.y)
                        .opacity(particle.opacity)

                        .opacity(status ? 1 : 0)
                        .animation(.none, value: status)
                }
            }
            .onAppear {
                if particles.isEmpty {
                    particles = (0..<10).map { _ in Particle() }
                }
            }
            .onChange(of: status) {
                // run animations
            }
        }
    }
}
```

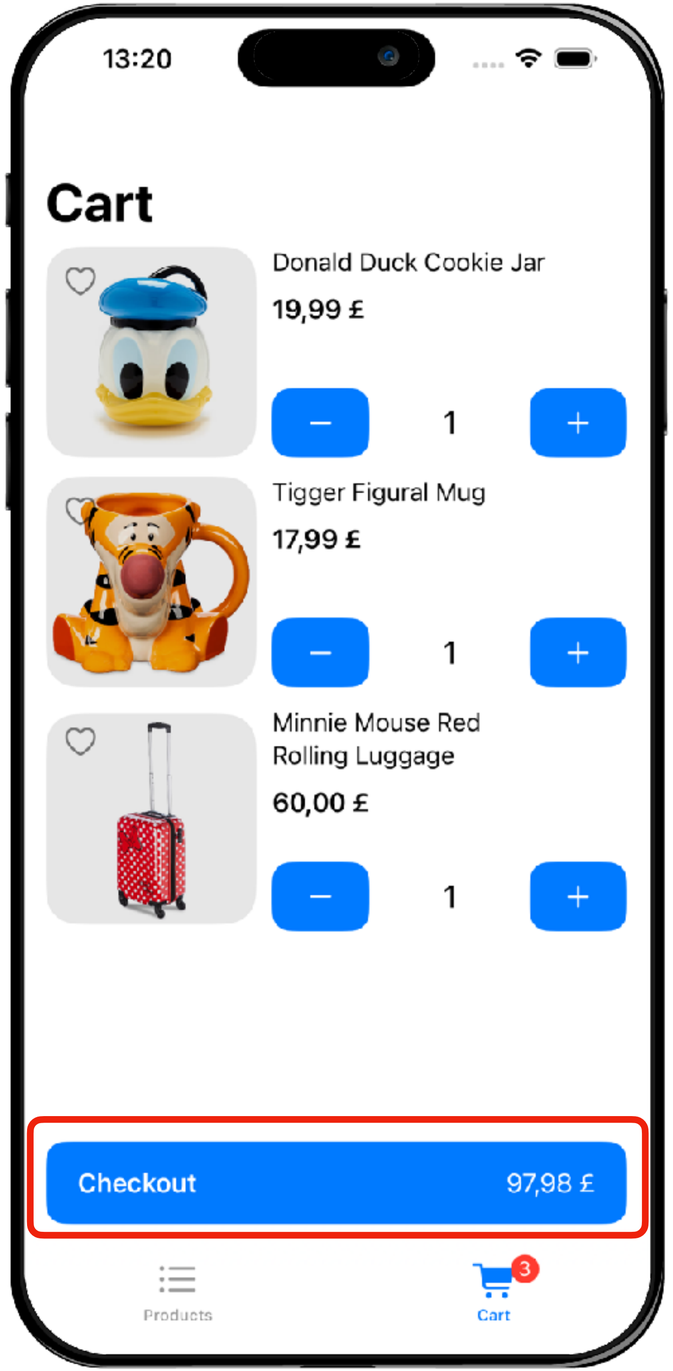


```swift
private struct ParticleEffectModifier: ViewModifier {
    @State private var particles: [Particle] = []

    func body(content: Content) -> some View {
        content.overlay(alignment: .top) {
            ZStack {
                ForEach(particles, id: \.id) { particle in
                    Image(systemName: systemImage)
                        .foregroundColor(...)
                        .scaleEffect(particle.scale)
                        .offset(x: particle.x, y: particle.y)
                        .opacity(particle.opacity)

                        .opacity(status ? 1 : 0)
                        .animation(.none, value: status)
                }
            }
            .onAppear {
                if particles.isEmpty {
                    particles = (0..<10).map { _ in Particle() }
                }
            }
            .onChange(of: status) {
                // run animations
            }
        }
    }
}
```

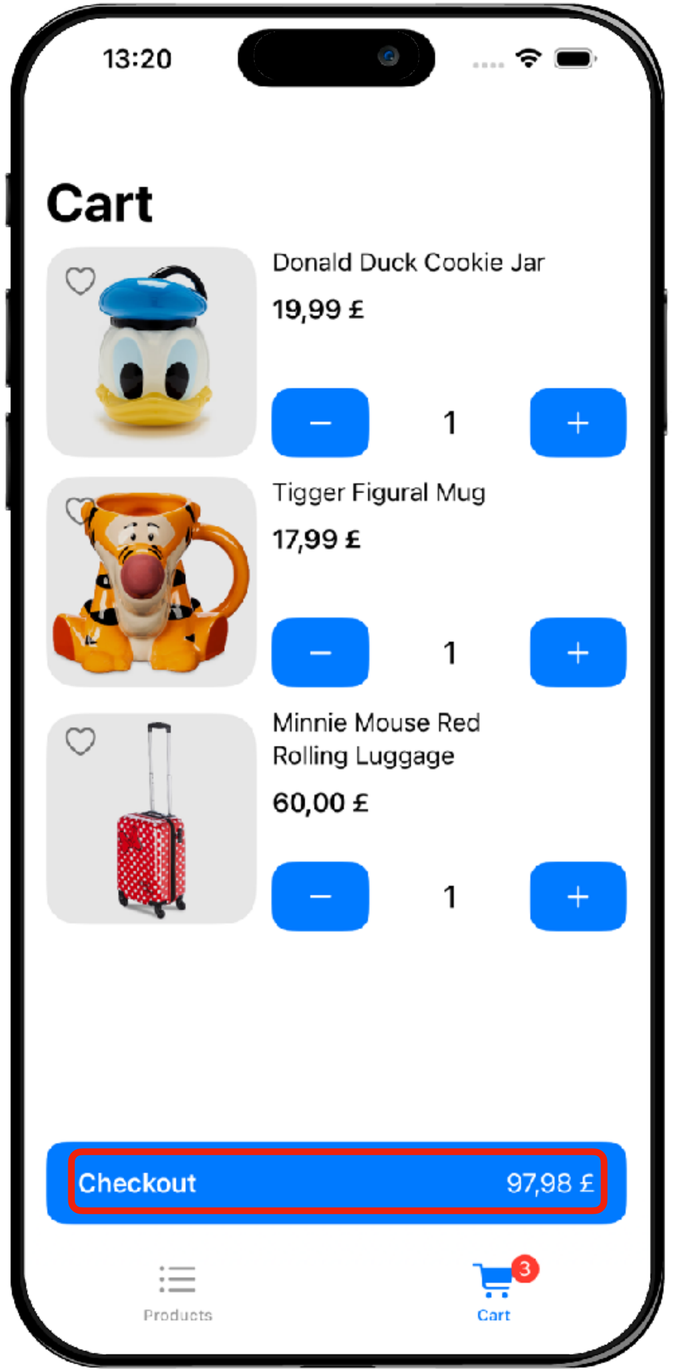


Checkout Button

```
struct CartButton: View {
    let sum: Double
    let onTap: () -> Void

    var body: some View {
        Button(action: onTap) {
            let priceStr: String = sum.formatted(.currency())
            HStack {
                Text("Checkout")
                    .font(.headline)
                Spacer()
                Text(priceStr)
            }
            .padding(...)
            .contentShape(Rectangle())
        }
        .background {
            Rectangle().fill(.tint)
                .clipShape(RoundedRectangle(cornerRadius: 12))
        }
        .foregroundStyle(.white)
        .buttonStyle(.plain)
        .padding(...)
    }
}
```



```swift
struct CartButton: View {
    let sum: Double
    let onTap: () -> Void

    var body: some View {
        Button(action: onTap) {
            let priceStr: String = sum.formatted(.currency())
            HStack {
                Text("Checkout")
                    .font(.headline)
                Spacer()
                Text(priceStr)
            }
            .padding(...)
            .contentShape(Rectangle())
        }
        .background {
            Rectangle().fill(.tint)
                .clipShape(RoundedRectangle(cornerRadius: 12))
        }
        .foregroundStyle(.white)
        .buttonStyle(.plain)
        .padding(...)
    }
}
```



13:20

Cart

Donald Duck Cookie Jar

19,99 £

− 1 +

Tigger Figural Mug

17,99 £

− 1 +

Minnie Mouse Red Rolling Luggage

60,00 £

− 1 +

Checkout 97,98 £

Products Cart 3

14:03

Cart

Donald Duck Cookie Jar

19,99 £

− 1 +

Tigger Figural Mug

17,99 £

− 1 +

Minnie Mouse Red Rolling Luggage

60,00 £

− 1 +

Products Cart 3

14:03

Cart

Donald Duck Cookie Jar

19,99 £

− 1 +

Tigger Figural Mug

17,99 £

− 1 +

Minnie Mouse Red Rolling Luggage

60,00 £

− 1 +

Products Cart 3

14:04

Cart

Donald Duck Cookie Jar

19,99 £

− 1 +

Tigger Figural Mug

17,99 £

− 1 +

Minnie Mouse Red Rolling Luggage

60,00 £

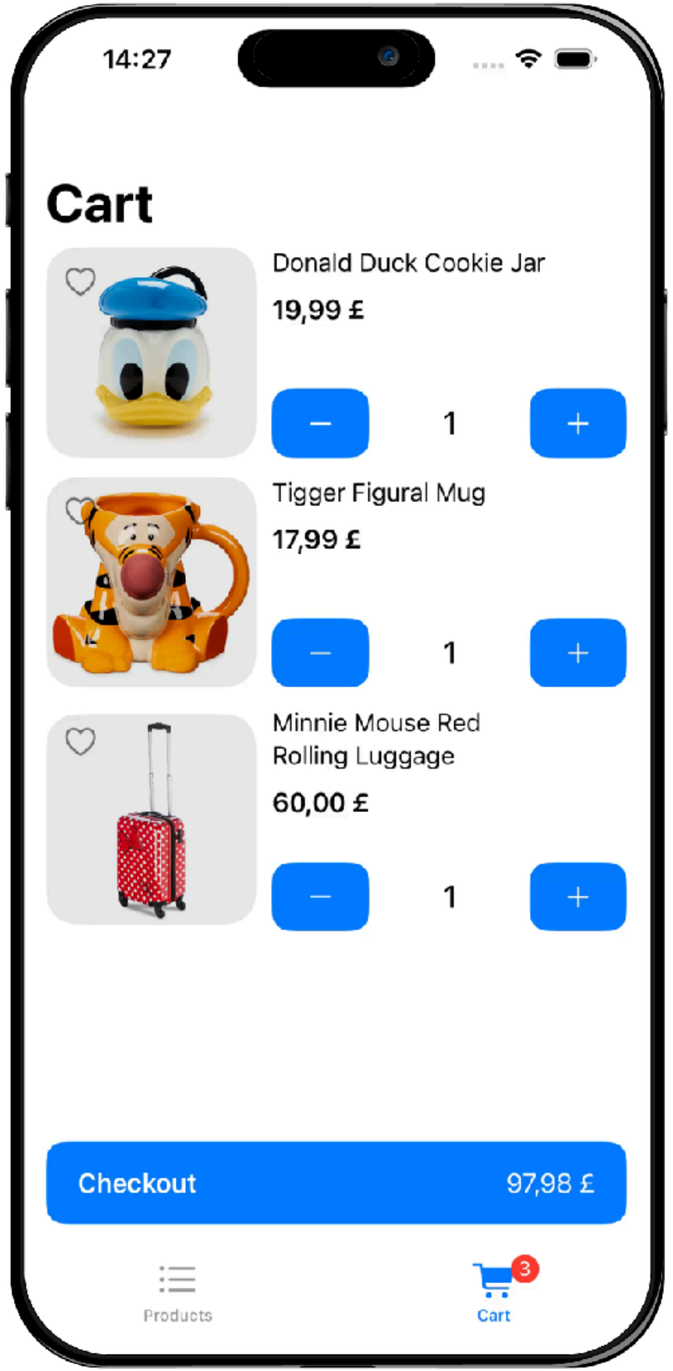
− 1 +

Products Cart 3

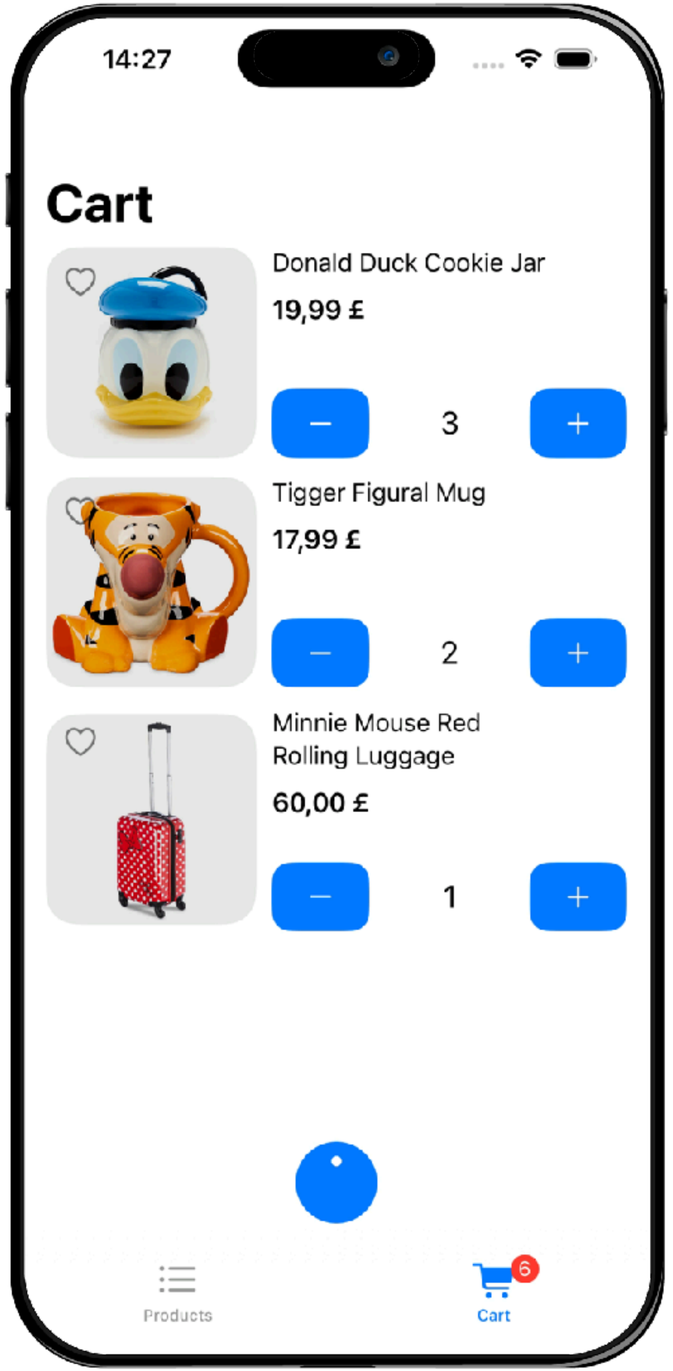
```
struct CartButton: View {
    enum State {
        case normal
        case loading
        case error
        case success
    }
    let state: State

    let sum: Double
    let onTap: () -> Void

    var body: some View {...}
}
```

```
struct CartButton: View {
    var body: some View {
        Button(action: onTap) {
            ZStack {
                switch state {
                case .normal:
                    let priceStr: String = sum.formatted(.currency(code: "GBP"))
                    HStack {
                        Text("Checkout")
                            .font(.headline)
                        Spacer()
                        Text(priceStr)
                    }
                case .loading:
                    ActivityIndicator()
                case .error:
                    Image(systemName: "exclamationmark")
                case .success:
                    Image(systemName: "checkmark")
                }
            }
        }
        .background {
            Rectangle()
                .fill(.tint)
                .clipShape(RoundedRectangle(cornerRadius: state == .normal ? 12 : 30))
        }
        .tint(...)
    }
}
```

```swift
struct CartButton: View {
    @State var isErrorAnimating = false

    var body: some View {
        Button(action: onTap) {
            // .transition() + contentTransition()
        }
        .onChange(of: state) { oldValue, newValue in
            if newValue == .error {
                isErrorAnimating = true
            }
        }
        .onChange(of: isErrorAnimating, { oldValue, newValue in
            if newValue {
                withAnimation(.spring()) {
                    isErrorAnimating = false
                }
            }
        })
        .offset(x: isErrorAnimating ? 30 : 0)
        .animation(
            .spring(),
            value: "\(state.rawValue)-\(sum)"
        )
    }
}
```

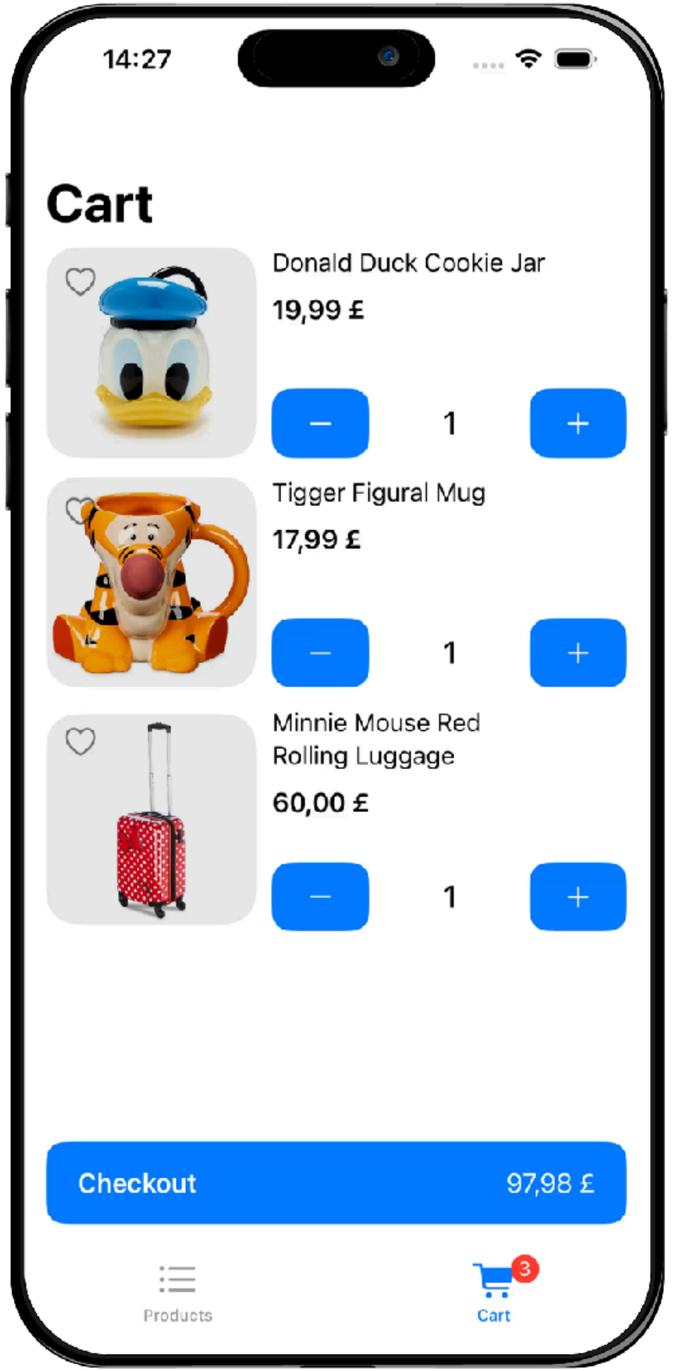


```swift
struct CartButton: View {
    @State var isErrorAnimating = false

    var body: some View {
        Button(action: onTap) {
            // .transition()
        }
        .onChange(of: state) { oldValue, newValue in
            if newValue == .error {
                isErrorAnimating = true
                withAnimation(.spring()) {
                    isErrorAnimating = false
                }
            }
        }
        .offset(x: isErrorAnimating ? 30 : 0)
        .animation(
            .spring(),
            value: "\(state.rawValue)-\(sum)"
        )
    }
}
```



Haptic feedback

```
struct CartButton: View {
    var body: some View {
        Button(action: onTap) {}
            .sensoryFeedback(trigger: state) { old, new in
                switch new {
                case .normal:
                    return .none
                case .loading:
                    return .impact(weight: .medium)
                case .error:
                    return .error
                case .success:
                    return .success
                }
            }
    }
}
```



Disney's Principles of Animation

Why animations are important

SwiftUI Animations

Applying everything together

Disney's Principles of Animation

Why animations are important

SwiftUI Animations

Applying everything together

Final Recipe

- Use all system instruments and API
- Animation should be
  - Clear and intuitive
  - Meaningful and purposeful
  - Should follow design principles and design style (HIG and App)

Aleksandr Filimonov
Senior iOS Dev · alexfilimon.dev

What Disney Taught Us About iOS Animations