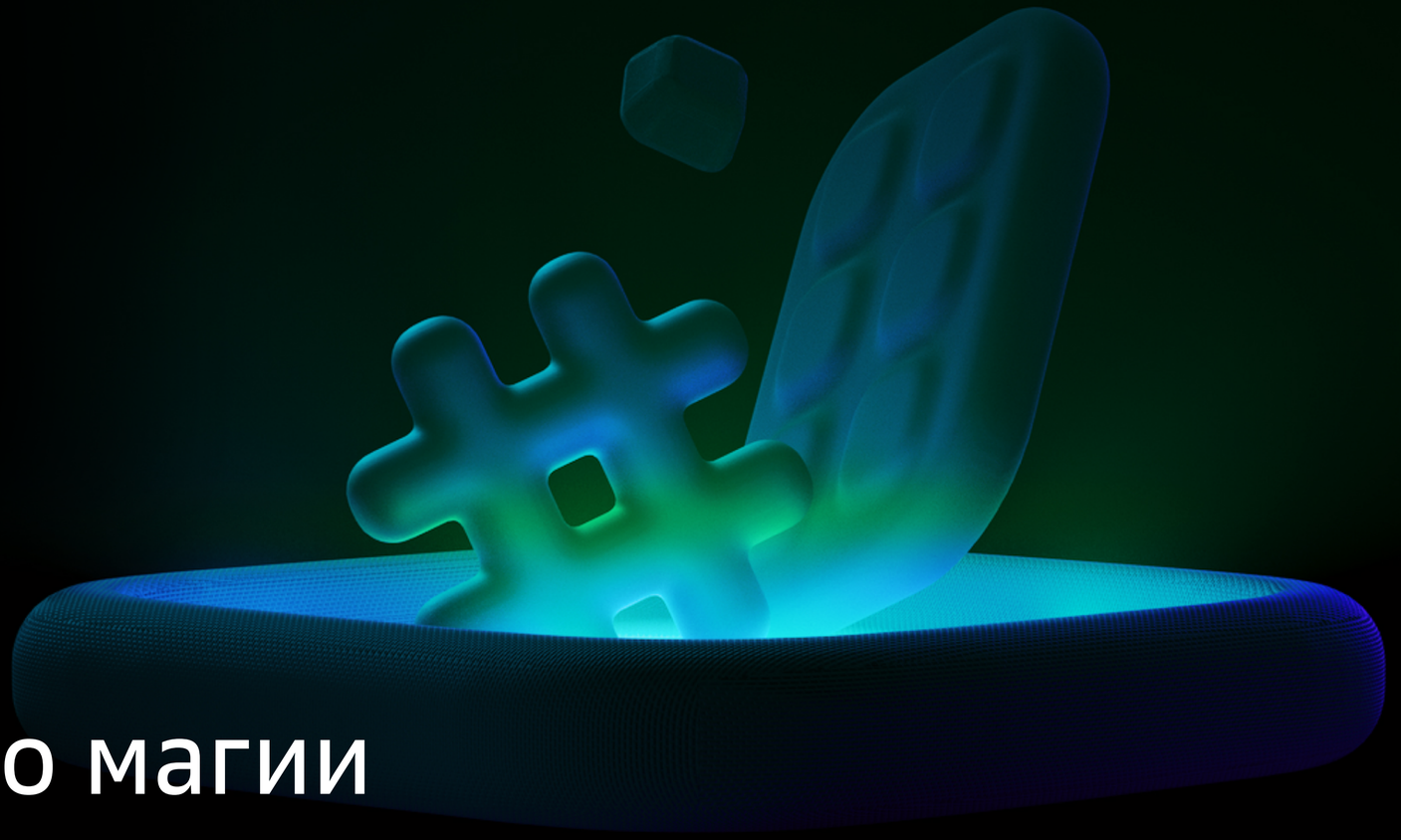
СБЕР «Цифровой Корпоративный Банк»



SwiftUI
Scroll, Text и немного магии

Благообразова Татьяна
Руководитель направления, мобильная разработка iOS
2025

SwiftUI
Scroll, Text и немного магии

- ScrollView: два примера из жизни

  ScrollView с прозрачным отступом для жестов;

  Управление ScrollView через интроспект, проблемы и решения, новое ScrollApi iOS18;

- Text

  Что надо знать про Text, делая социалку и не только;

  Чего ждать от TextRenderer (iOS18) и облегчит ли он нам жизнь;

Благообразова Татьяна

SwiftUI
Scroll, Text и немного магии

- ScrollView: два примера из жизни

  ScrollView с прозрачным отступом для жестов;

  Управление ScrollView через интроспект, проблемы и решения, новое ScrollApi iOS18;

- Text

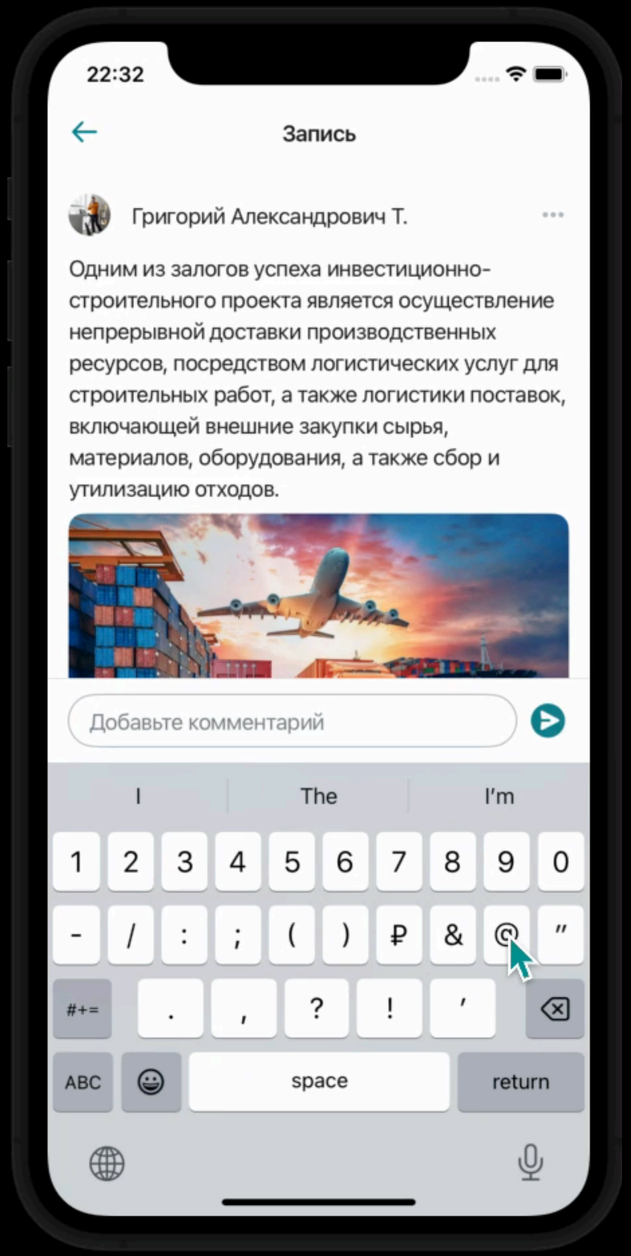
  Что надо знать про Text, делая социалку и не только;

  Чего ждать от TextRenderer (iOS18) и облегчит ли он нам жизнь;

#3 Ссылки на GitHub в примерах (и в конце доклада)

Благообразова Татьяна

ScrollView с прозрачным отступом для жестов

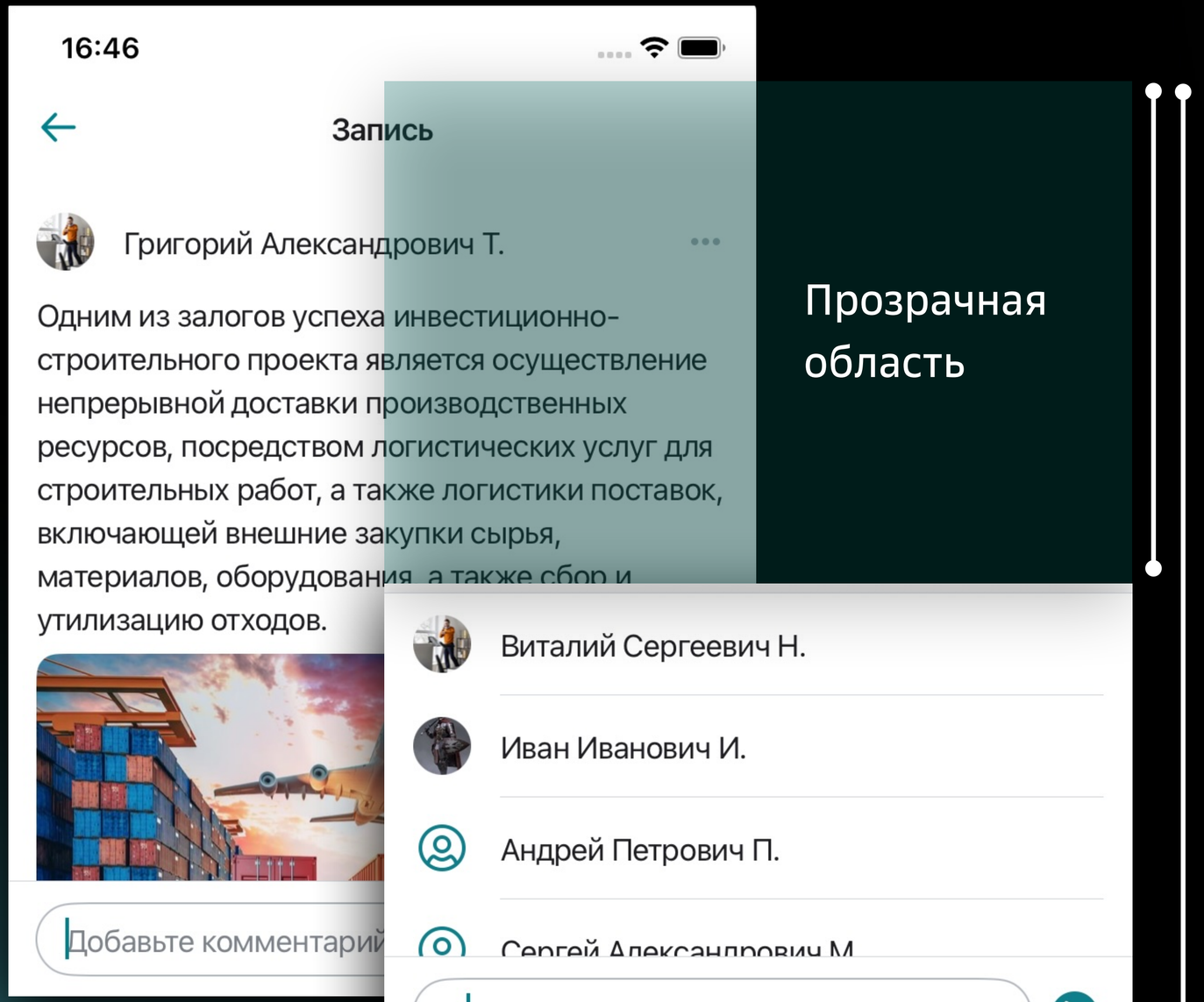


ScrollView пример 1:

- Сделать (растущий) список пользователей над клавиатурой для упоминаний в комментариях.
- Контент за списком доступен для жестов.

Изображения сгенерированы GigaChat

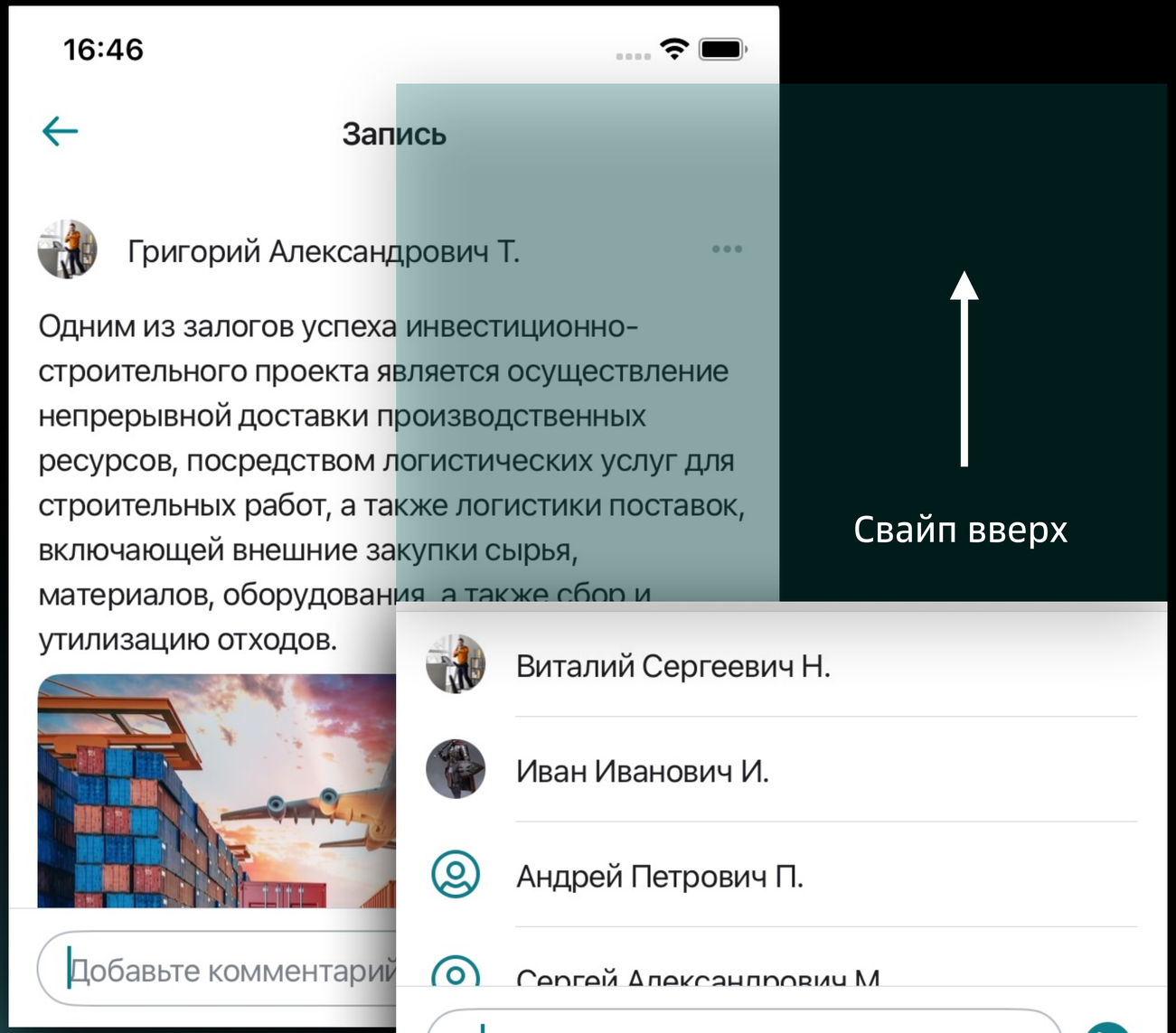
ScrollView с прозрачным отступом для жестов



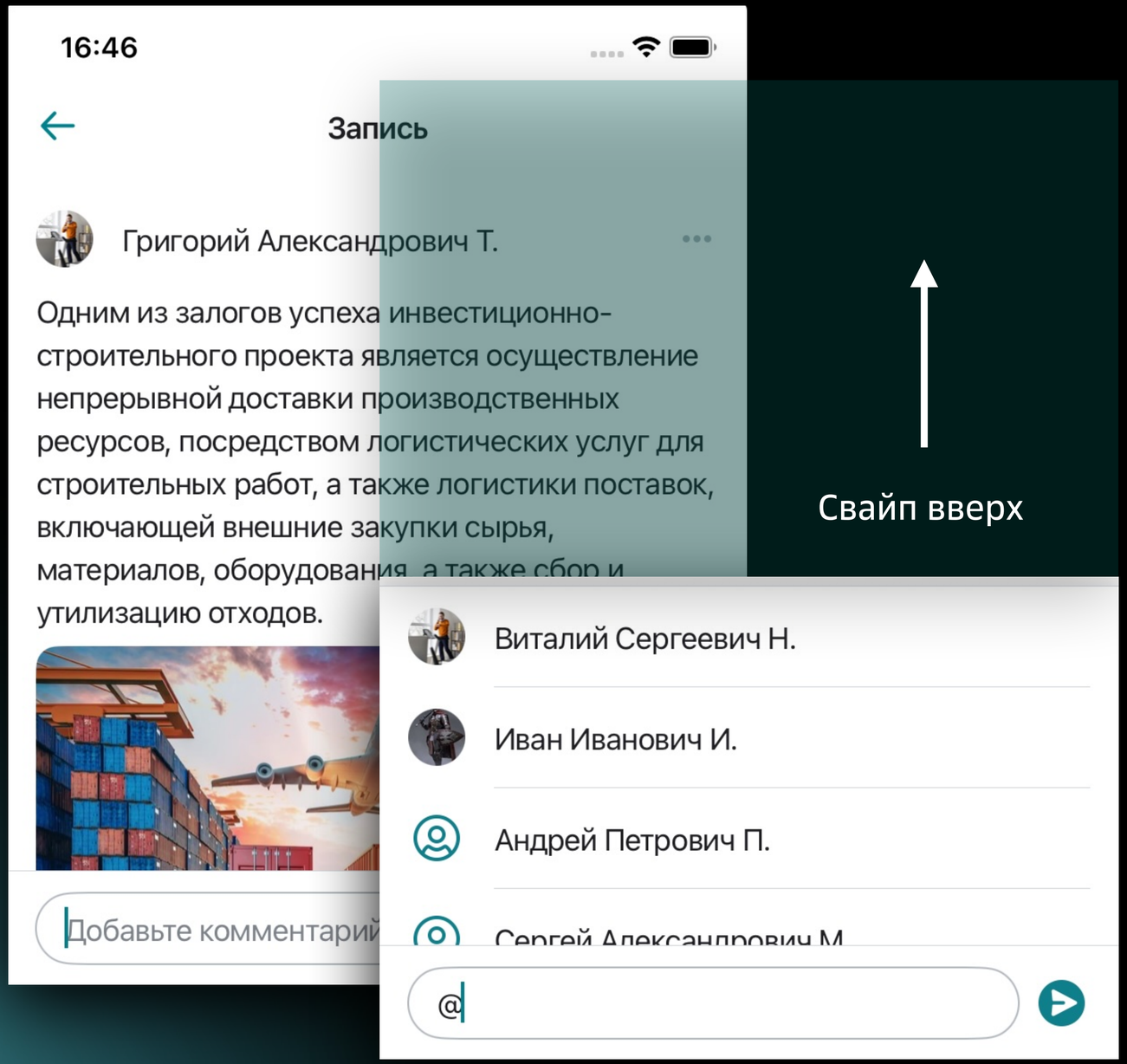
Прозрачная область

ScrollView

ScrollView с прозрачным отступом для жестов



ScrollView с прозрачным отступом для жестов



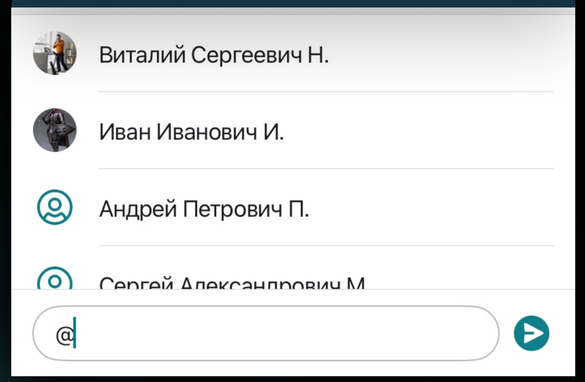
Свайп вверх

Как сделать эту область прозрачной для жестов?

ScrollView

ScrollView с прозрачным отступом для жестов

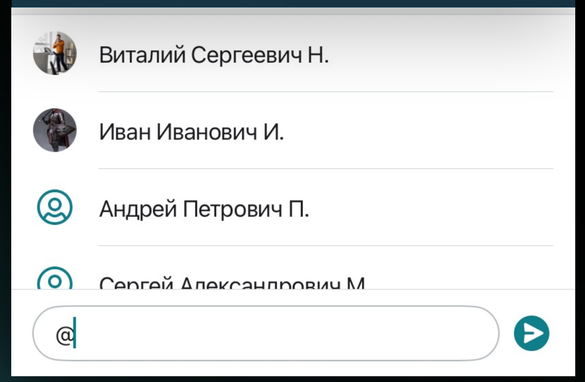
Как сделать эту область прозрачной для жестов?




```
var userList: some View {

    ScrollView {
        LazyVStack(spacing: 0) {

            Color.clear
                .frame(height: scrollViewInset, alignment: .center)

            ForEach(users) { user in
                UsersView(user)
            }
        }
    }
```

ScrollView с прозрачным отступом для жестов

Как сделать эту область прозрачной для жестов?

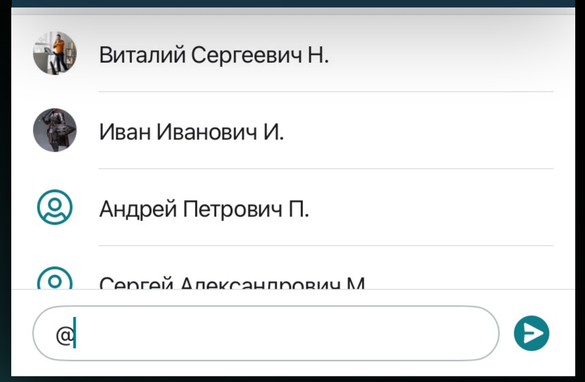



```
var userList: some View {

    ScrollView {
        LazyVStack(spacing: 0) {

            Color.clear
                .frame(height: scrollViewInset, alignment: .center)

            ForEach(users) { user in
                UsersView(user)
            }
        }
    }
```

```
.simultaneousGesture(…)
.gesture(…)
.highPriorityGesture(…)
.allowsHitTesting(false)
.defersSystemGestures(…)
```

ScrollView с прозрачным отступом для жестов

Как сделать эту область прозрачной для жестов?

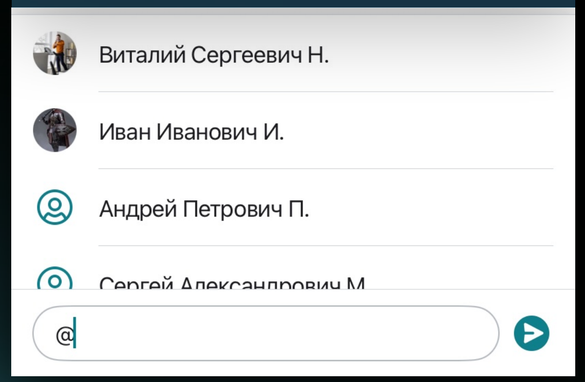



```
var userList: some View {

    ScrollView {
        LazyVStack(spacing: 0) {

            Color.clear
                .frame(height: scrollViewInset, alignment: .center)

            ForEach(users) { user in
                UsersView(user)
            }
        }
    }
```

```
Text("А если попробовать .contentShape?!")
    .padding(.all, 16)
    .contentShape(.rect)
```

ScrollView и ContentShape с динамическим фреймом

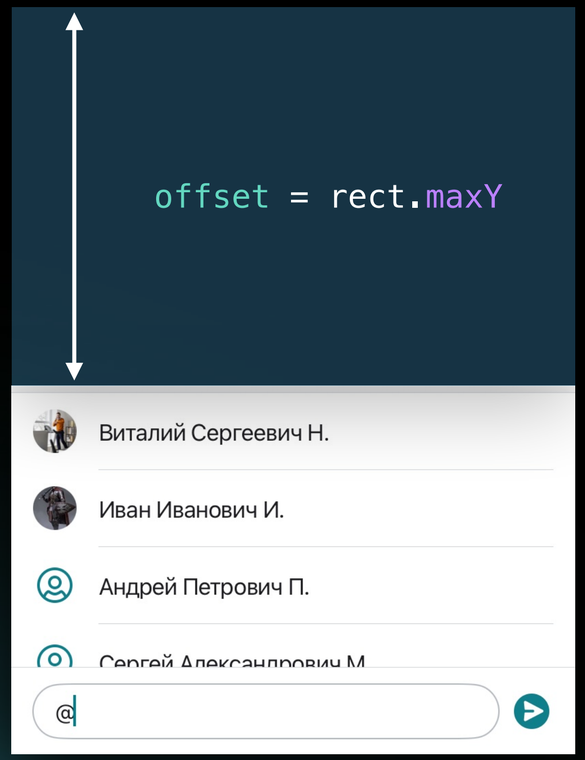
Как сделать эту область прозрачной для жестов?




```
var userList: some View {

    ScrollView {
        LazyVStack(spacing: 0) {

            Color.clear
                .frame(height: scrollViewInset, alignment: .center)

            ForEach(users) { user in
                UsersView(user)
            }
        }
    }
```

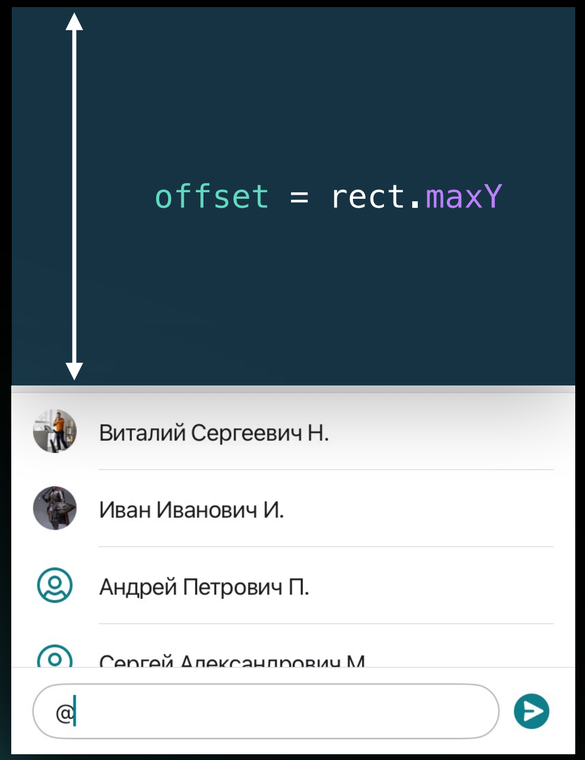
```
    .contentShape(
        /// Делаем отступ прозрачным для жестов
        Rectangle()
            .size(...)
            /// Взять frame ScrollView и исключить прозрачную область
    )
}
```

ScrollView и ContentShape с динамическим фреймом



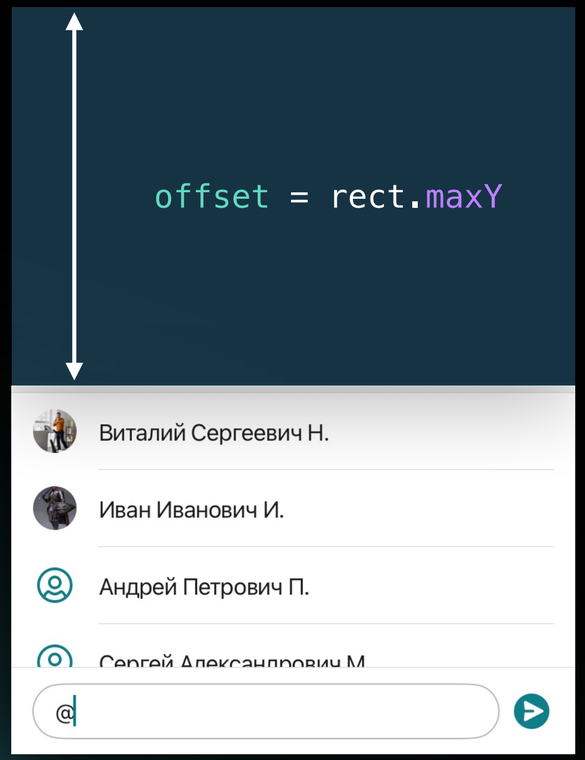

```
var userList: some View {

    ScrollView {
        LazyVStack(spacing: 0) {

            Color.clear
                .frame(height: scrollViewInset, alignment: .center)
                offset

            ForEach(users) { user in
                UsersView(user)
            }
        }
    }

    .coordinateSpace(.named(scrollViewSpace))

    .contentShape(
          /// Делаем отступ прозрачным для жестов
        Rectangle()
            .size(...)
            /// Взять rect ScrollView и исключить прозрачную область
    )
}
```

ScrollView и ContentShape с динамическим фреймом



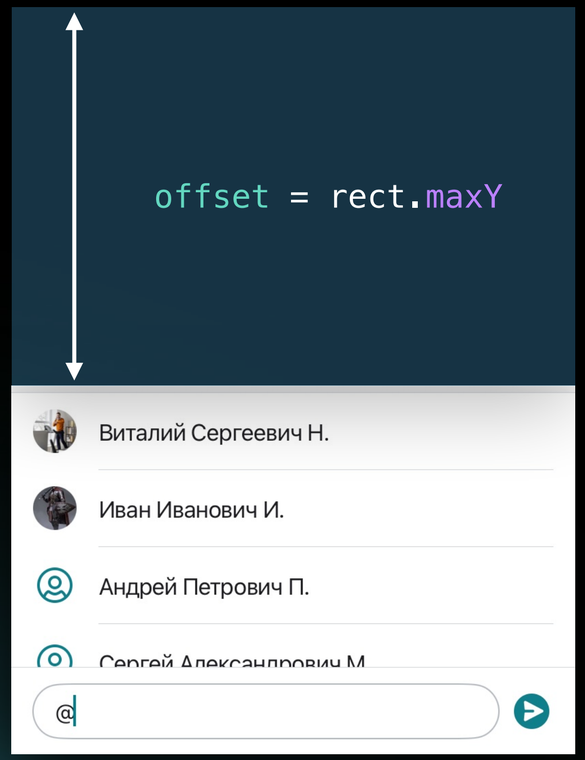

```
var userList: some View {

    ScrollView {
        LazyVStack(spacing: 0) {

            Color.clear
                .frame(height: scrollViewInset, alignment: .center)
                .onFrameChanged(coordinateSpace: .named(scrollViewSpace)) { rect in
                    offset = max(rect.maxY, 0)
                }

             ForEach(users) { user in
                 UsersView(user)
             }
        }
    }

    .coordinateSpace(.named(scrollViewSpace))


    .contentShape(
          /// Делаем отступ прозрачным для жестов
        Rectangle()
            .size(...)
            /// Взять rect ScrollView и исключить прозрачную область
    )
}
```

ScrollView и ContentShape с динамическим фреймом




```
var userList: some View {

    ScrollView {
        LazyVStack(spacing: 0) {

            Color.clear
                .frame(height: scrollViewInset, alignment: .center)
                .onFrameChanged(coordinateSpace: .named(scrollViewSpace)) { rect in
                    offset = max(rect.maxY, 0)
                }

            ForEach(users) { user in
                UsersView(user)
            }
        }
    }

    .coordinateSpace(.named(scrollViewSpace))
    .onFrameChanged { scrollViewRect = $0 }

    .contentShape(
        /// Делаем отступ прозрачным для жестов
        Rectangle()
            .size(width: scrollViewRect.width, height: scrollViewRect.height - offset)
            .offset(x: 0, y: offset)
    )
}
```

ScrollView и ContentShape с динамическим фреймом




```
var userList: some View {

    ScrollView {
        LazyVStack(spacing: 0) {

            Color.clear
                .frame(height: scrollViewInset, alignment: .center)
                .onFrameChanged(coordinateSpace: .named(scrollViewSpace)) { rect in
                    offset = max(rect.maxY, 0)
                }

             ForEach(users) { user in
                 UsersView(user)
             }
        }
    }

    .coordinateSpace(.named(scrollViewSpace))
    .onFrameChanged { scrollViewRect = $0 }

    .contentShape(
          /// Делаем отступ прозрачным для жестов
        Rectangle()
            .size(width: scrollViewRect.width, height: scrollViewRect.height - offset)
            .offset(x: 0, y: offset)
    )
}
```

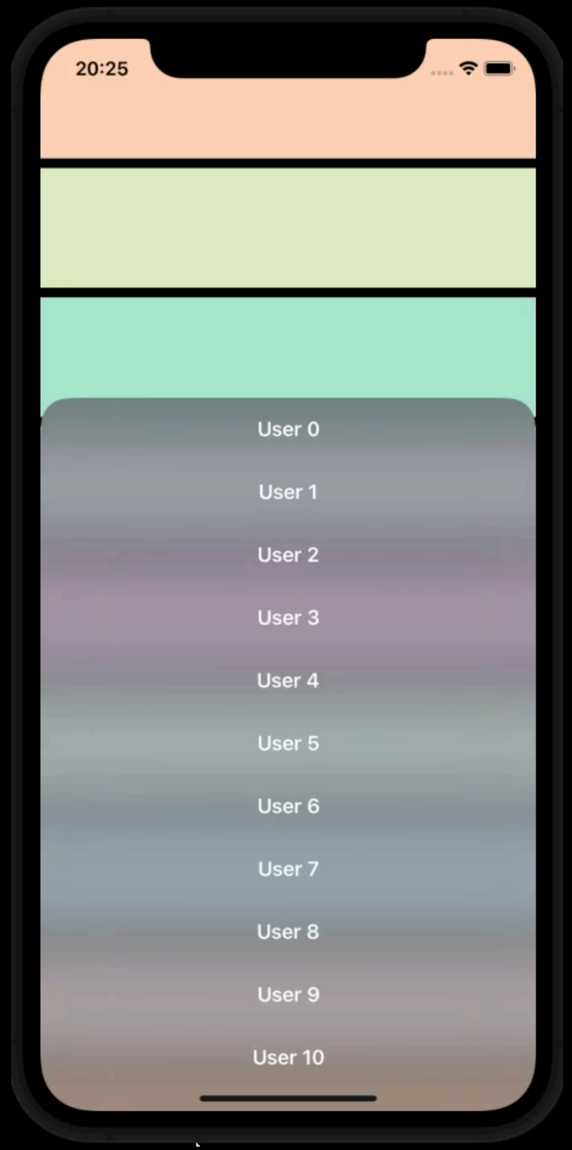
ContentShape с динамическим фреймом

Модификатор .contentShape - полезный инструмент, который позволяет не только увеличивать область жеста, но и менять ее разными способами, в том числе динамически.

Действие этого модификатора можно сравнить с функциями в UIKit:

```
func point(inside point: CGPoint, with event: UIEvent?) -> Bool
func hitTest(_ point: CGPoint, with event: UIEvent?) -> UIView?
```

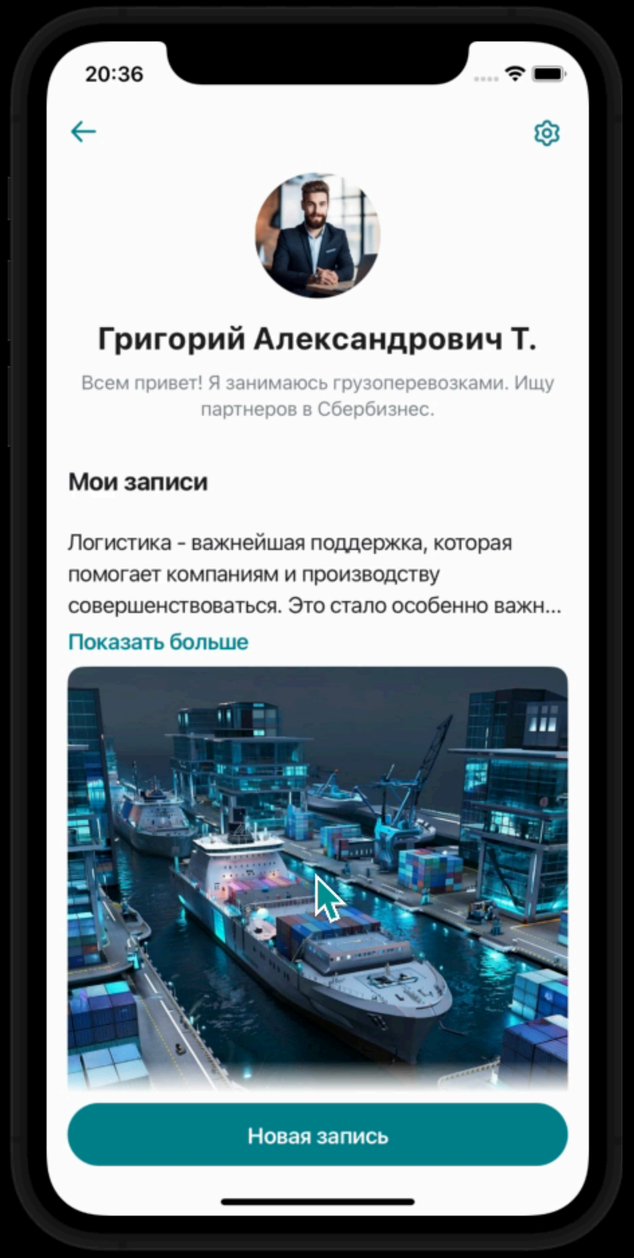
ScrollView и ContentShape с динамическим фреймом



На Github краткий пример:

Два scrollView: и через contentInset первого scrollView вы можете взаимодействовать со вторым.

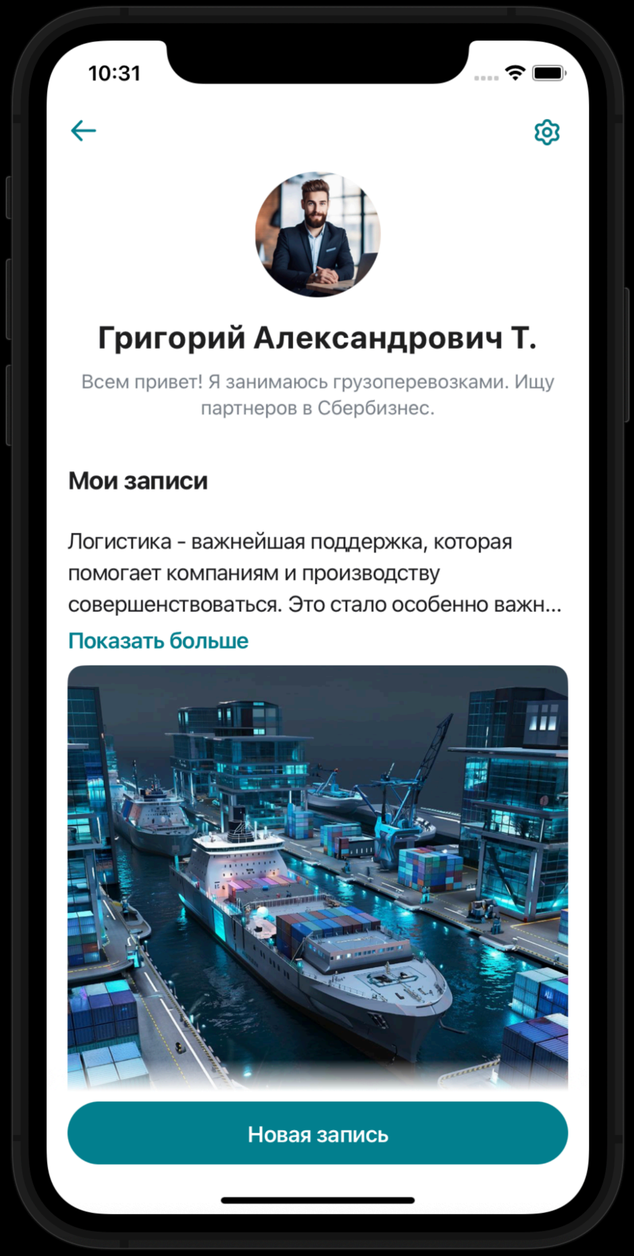
ScrollView: Анимация профиля



ScrollView пример 2: анимация профиля

Изображения сгенерированы GigaChat

ScrollView: Анимация профиля



ScrollView пример 2: анимация профиля

- Как сделать анимацию профиля в SwiftUI
- Управление ScrollView через интроспект, проблемы и решения
- Как выглядит решение с новым ScrollApi iOS18

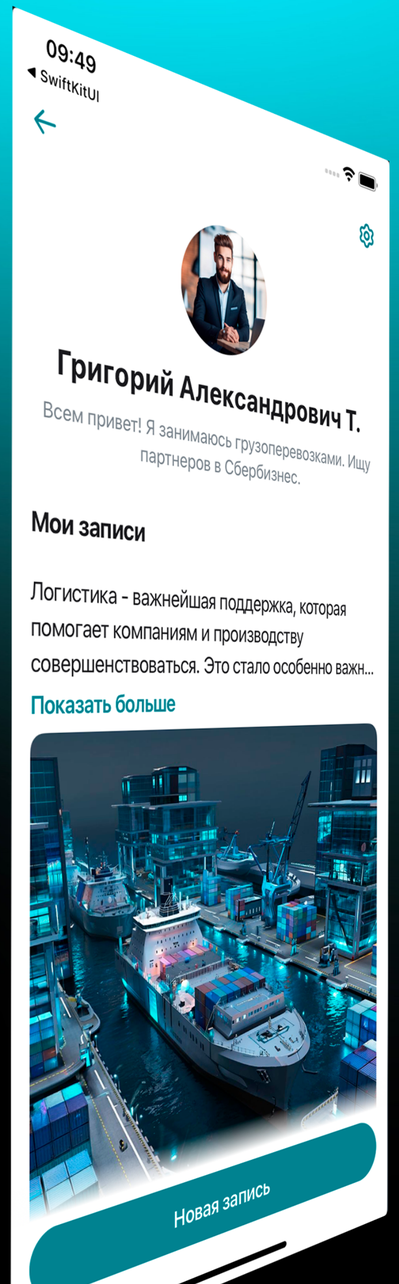
ScrollView: Анимация профиля


```
let profileModel: ProfileView.Model = .init(
    username: "Григорий Александрович Т.",
    description: "Всем привет! ...",
    getImage: { ... }
)
```

ScrollView: Анимация профиля

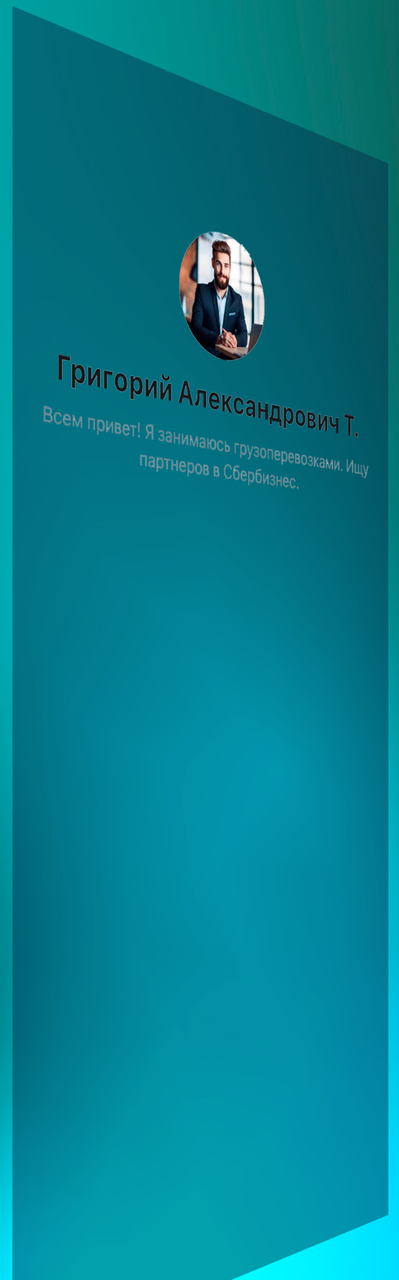

```
let profileModel: ProfileView.Model = .init(
    username: "Григорий Александрович Т.",
    description: "Всем привет! ...",
    getImage: { ... }
)

var body: some View {

    mainContent
        .wrappedInLazyVStack(profileModel: profileModel)

}
```

ScrollView: Анимация профиля



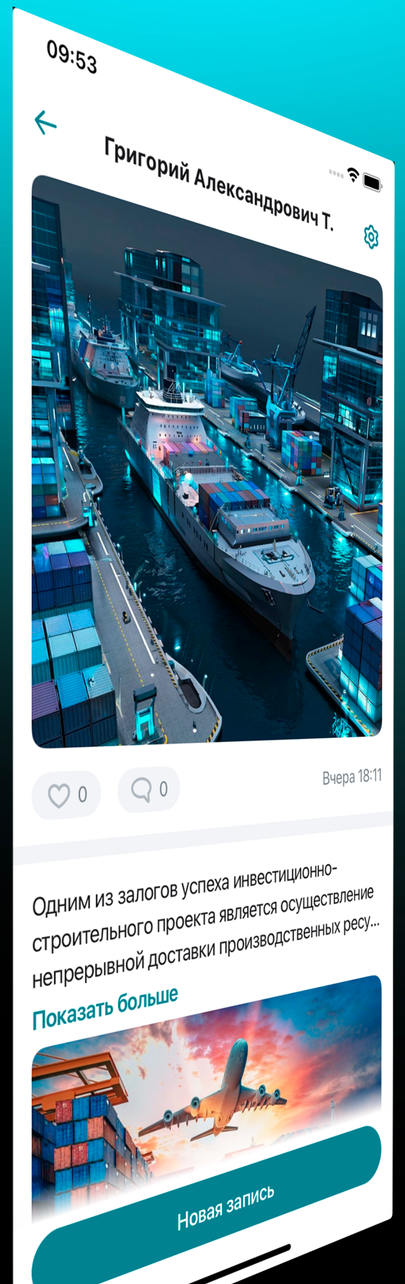
Spacer

ScrollView



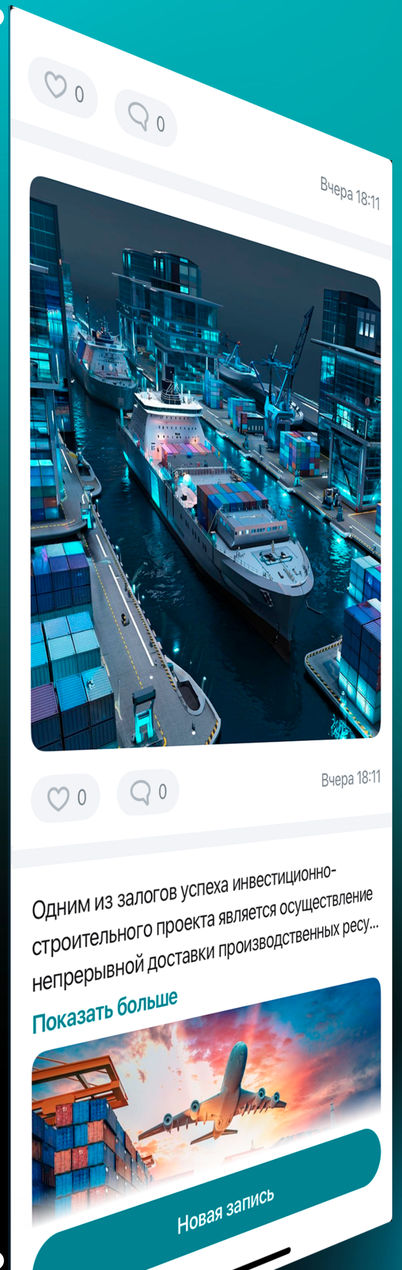
ProfileView

ScrollView: Анимация профиля



Spacer

ScrollView



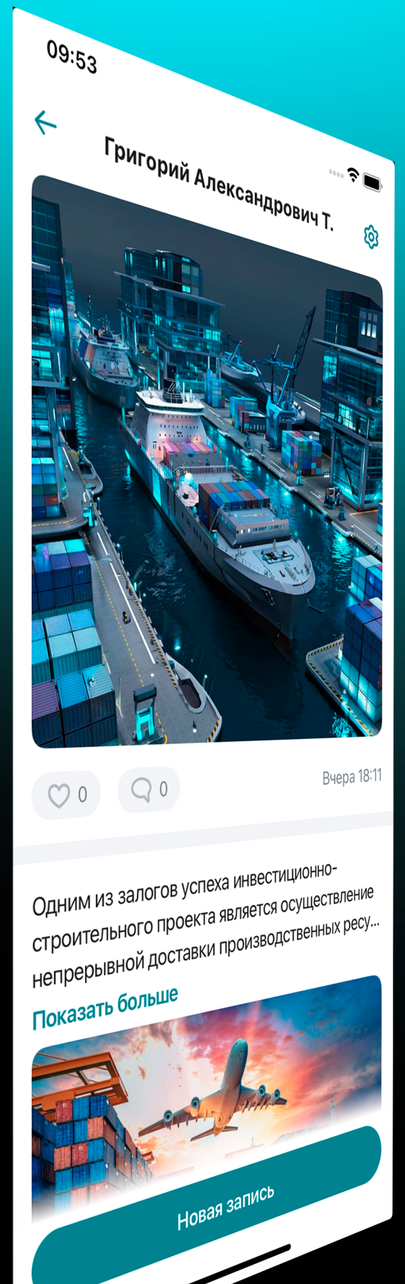
ProfileView

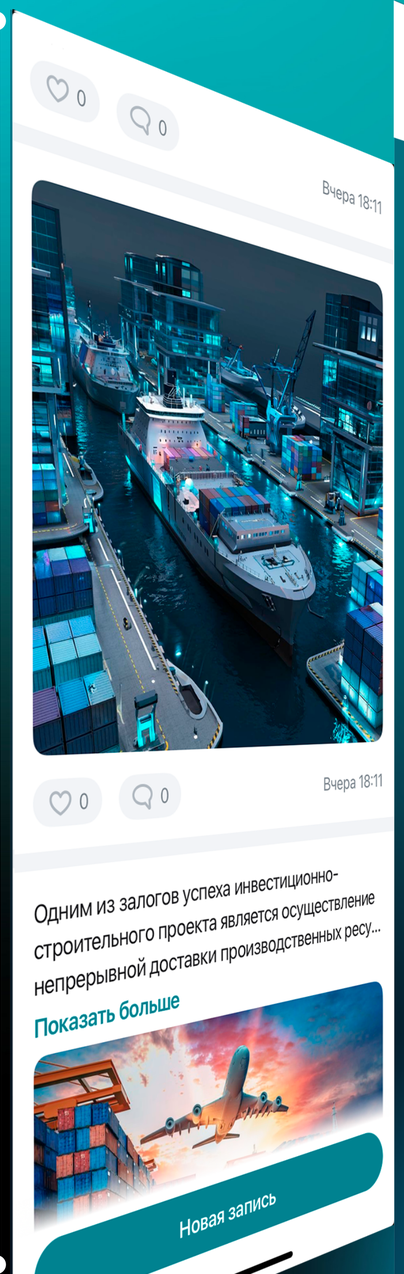
ScrollView: Анимация профиля

Spacer

ScrollView

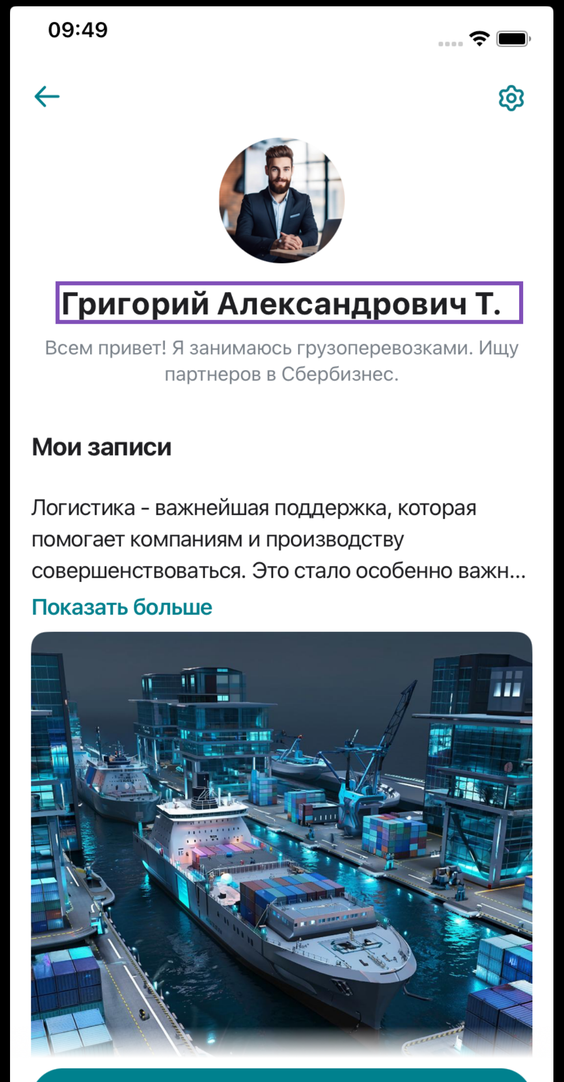
ProfileView





ScrollView: Анимация профиля

ProfileView



Задача: совместить заголовок ProfileView с заголовком NavigationBar.

ScrollView: Анимация профиля

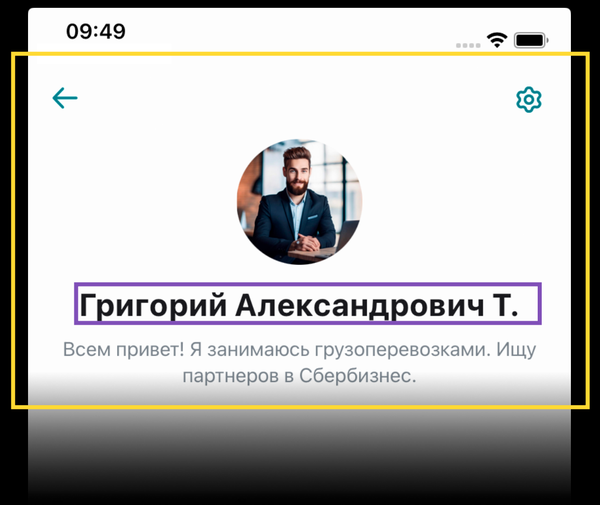
ProfileView

Задача: совместить заголовок ProfileView с заголовком NavigationBar.

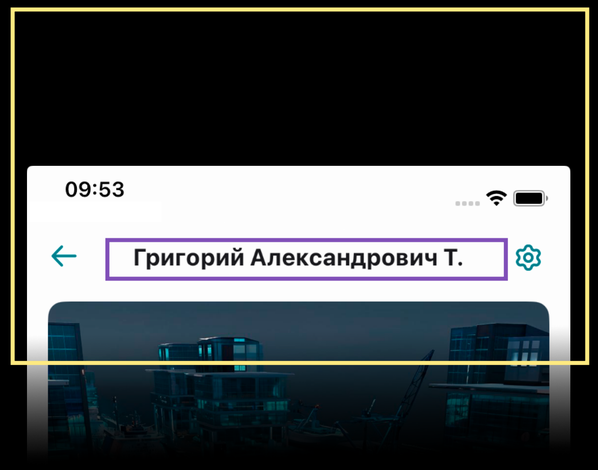
ProfileView имеет фиксированный размер и два крайних положения.

ScrollView: Анимация профиля

ProfileView



progress = 0



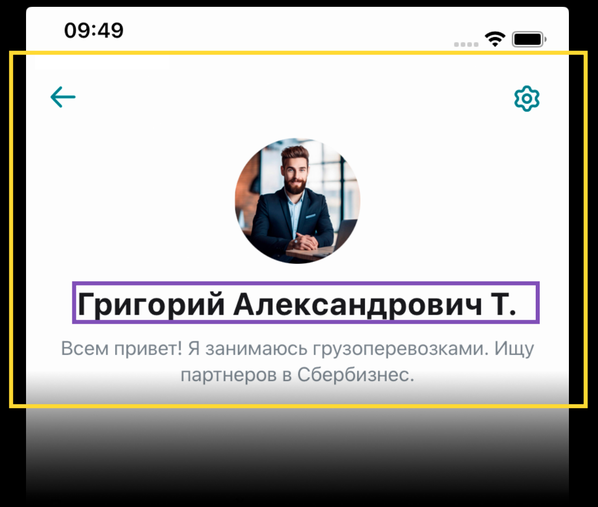
progress = 1

ProfileView имеет фиксированный размер и два крайних положения: progress [0...1]

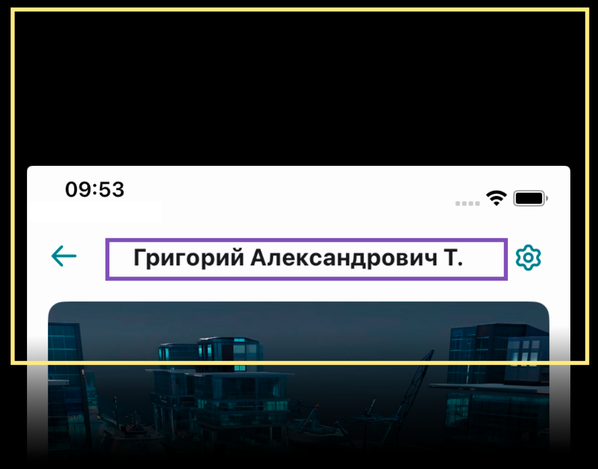
Картинка и описание уменьшаются и становятся прозрачными в зависимости от progress.

ScrollView: Анимация профиля

ProfileView



progress = 0



progress = 1

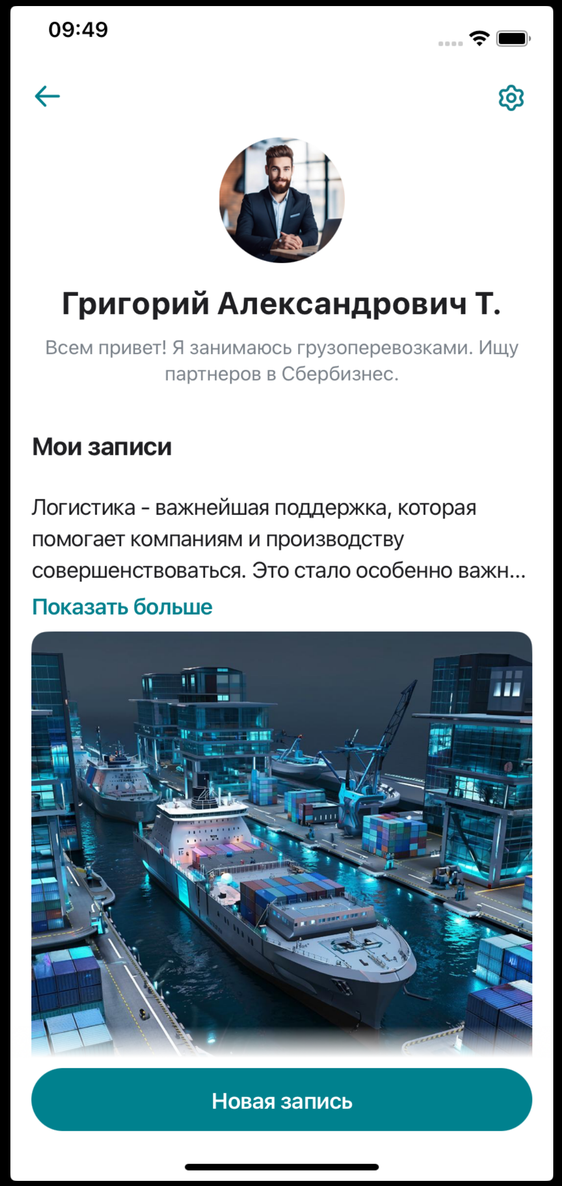
```
description
    .scaleEffect(1.0 - progress)
    .opacity(1.0 - progress)
```

ProfileView имеет фиксированный размер и два крайних положения: progress [0...1]

Картинка и описание уменьшаются и становятся прозрачными в зависимости от progress.

В SwiftUI удобно управлять View, через изменение параметра

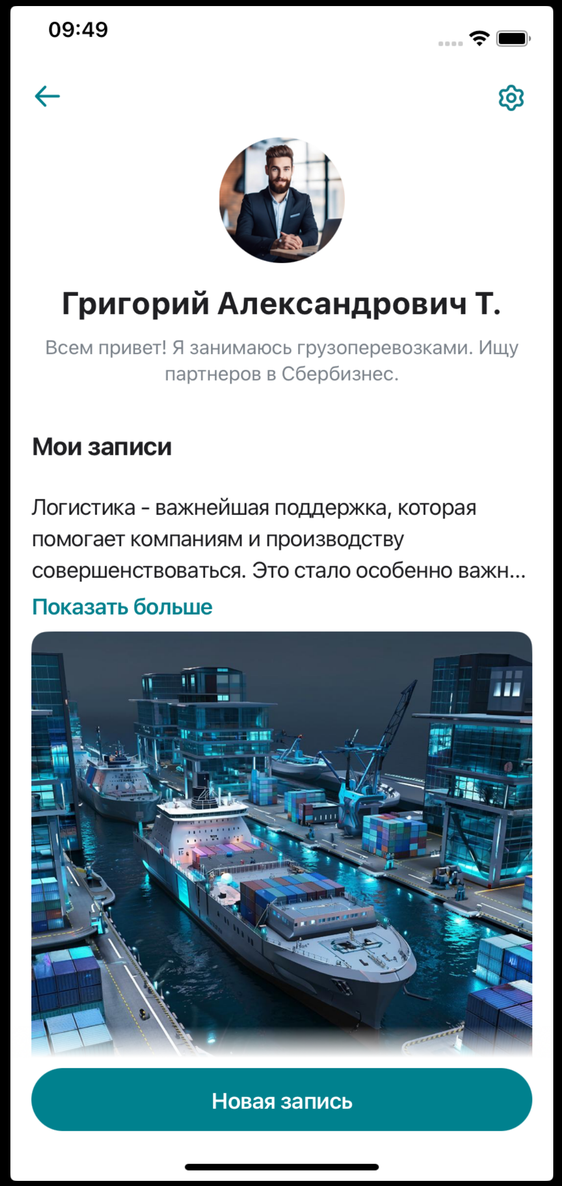
ScrollView: Анимация профиля



Нужно:

- ✳ отслеживать изменение ScrollView Offset
- ✳ отслеживать завершение скрола (доводчики анимации)

ScrollView: Анимация профиля



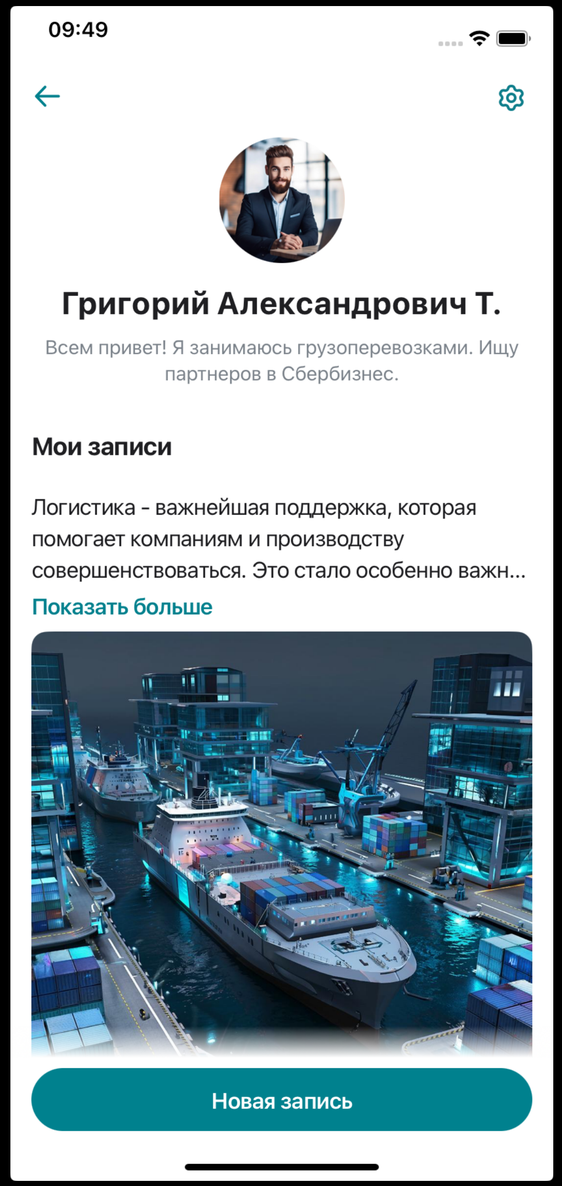
Нужно:

- отслеживать изменение ScrollView Offset
- отслеживать завершение скрола (доводчики анимации)

Для iOS 16-17 используем интроспект + observing

```
scrollView. observe(\.contentOffset,...) { _, offset in
    ... offset ...
}
```

ScrollView: Анимация профиля

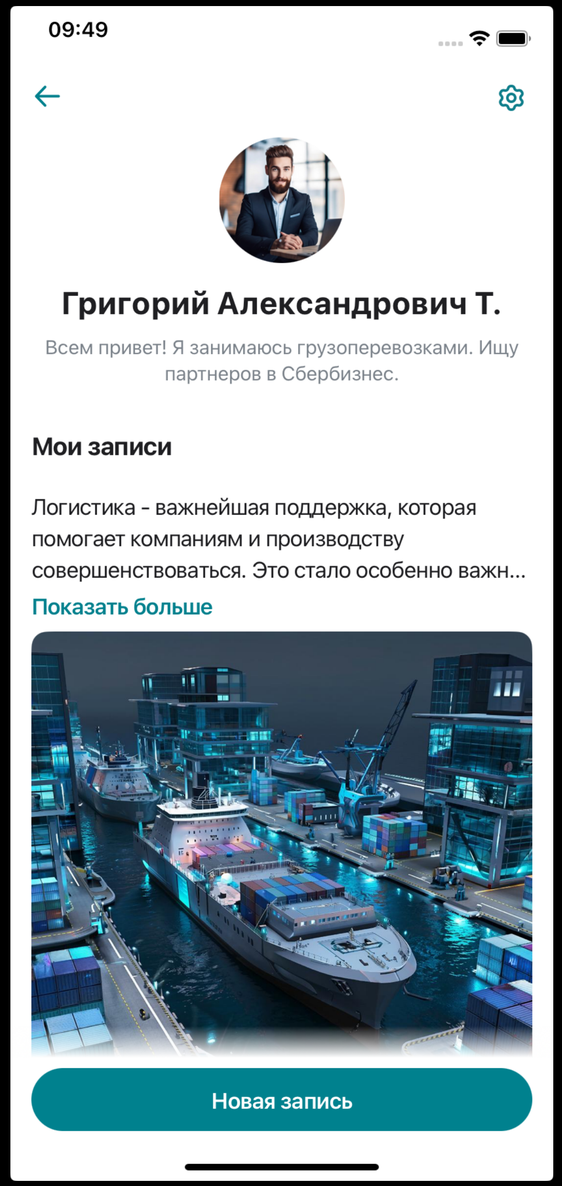


Нужно:

- отслеживать изменение ScrollView Offset
- отслеживать завершение скрола (доводчики анимации)

```
...
scrollView?.delegate = self
...
```

ScrollView: Как отслеживать завершение скролла?

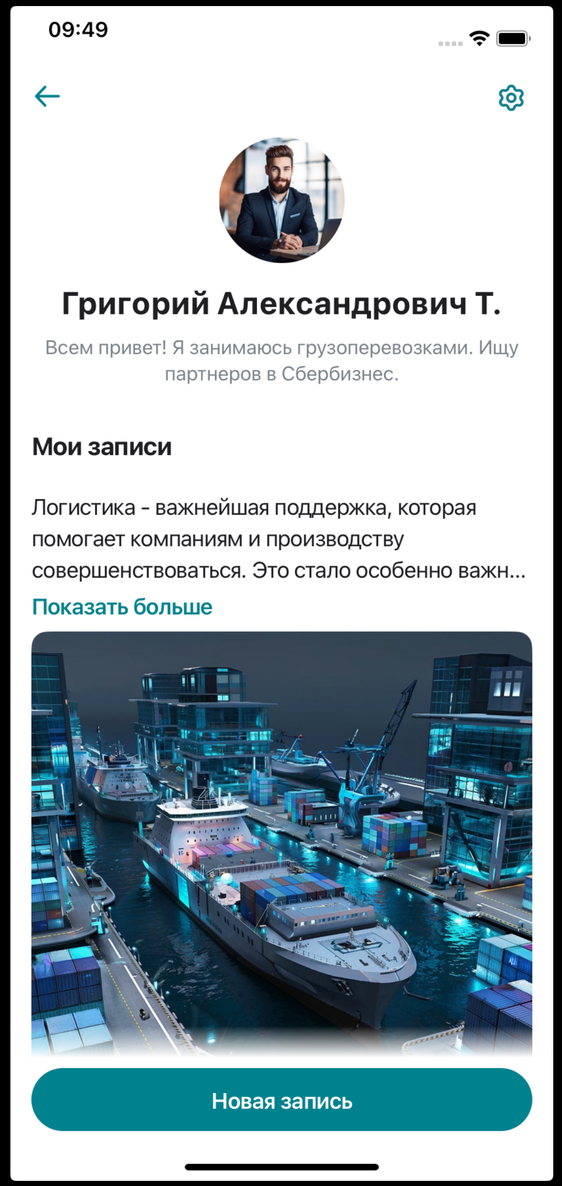



```
...
scrollView?.delegate = self
...
```

В SwiftUI нельзя использовать делегат для объектов, которые мы получаем интроспектом, поскольку SwiftUI использует делегат для своих целей.

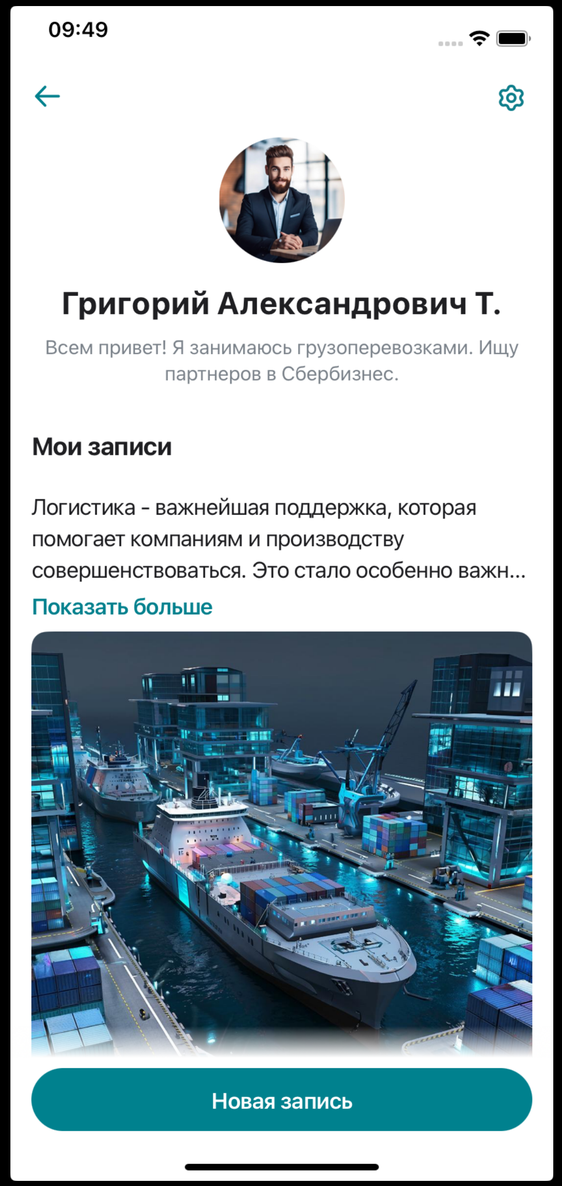
```
func scrollViewDidEndDragging(_ scrollView: UIScrollView, decelerate: Bool) {
    if !decelerate {
        finishScrolling?(scrollView)
    }
}

func scrollViewDidEndDecelerating(_ scrollView: UIScrollView) {
    finishScrolling?(scrollView)
}
```

ScrollView: Как отслеживать завершение скролла?




```
@Published var scrollViewOffset: CGFloat = 0
...
$scrollViewOffset
    .debounce(for: 0.05, scheduler: RunLoop.main)
    .sink { [weak self] _ in
        self?.finishScrolling()
    }
    .store(in: &cancellable)
```

ScrollView: Как отслеживать завершение скролла?

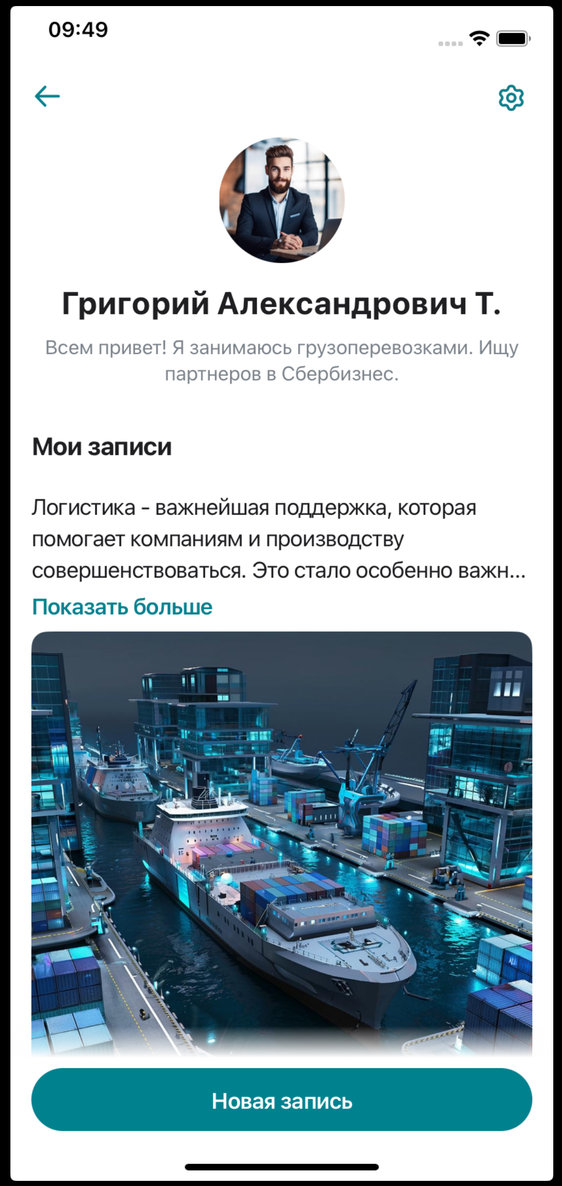



```
@Published var scrollViewOffset: CGFloat = 0
...
$scrollViewOffset
    .debounce(for: 0.05, scheduler: RunLoop.main)
    .sink { [weak self] _ in
        self?.finishScrolling()
    }
    .store(in: &cancellable)
```

Проблему решает модификатор .debounce для published переменной.
Debounce задерживает событие на 0.05 секунд =>
изменение offset будет откладываться, пока не произойдет остановка скола.

P.S. Даже если скролл замедляется или движется очень медленно, - события все равно приходят каждый фрейм.

ScrollView: Как отслеживать завершение скролла?




```
@Published var scrollViewOffset: CGFloat = 0
...
$scrollViewOffset
    .debounce(for: 0.05, scheduler: RunLoop.main) ~~DispatchQueue.main~~
    .sink { [weak self] _ in
        self?.finishScrolling()
    }
    .store(in: &cancellable)
```

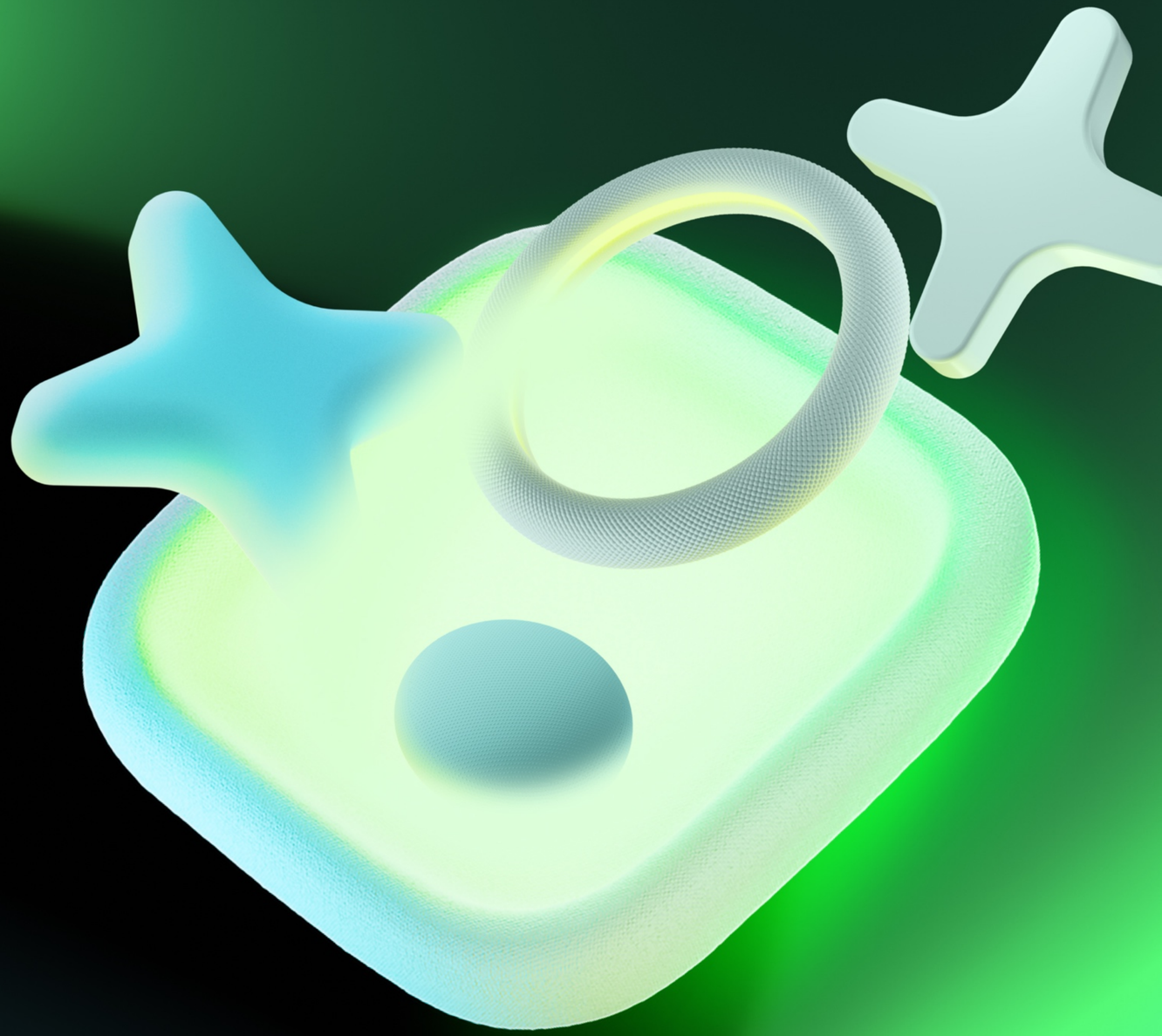
Если у ScrollView состояние isDragging, то есть пользователь находится в процессе касания или перетаскивания, то события также не будет приходить до завершения выполнения жеста из-за использования планировщика RunLoop.main.

Вызов для RunLoop.main выполнится при переходе в .default mode, то есть после завершения жестов касания и перетаскивания.

Анимация профиля + Scroll APIs iOS 18 🎉

Scroll APIs iOS 18

- ✳ позволяет полностью решить нашу задачу без использования интроспекта.
- ✳ решения выглядят проще и понятнее



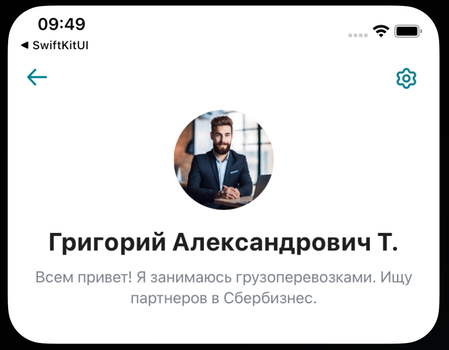
Пример на GitHub содержит отдельную ветку с iOS18 реализацией 🎉

Анимация профиля + Scroll APIs iOS 18 🎉

iOS 18

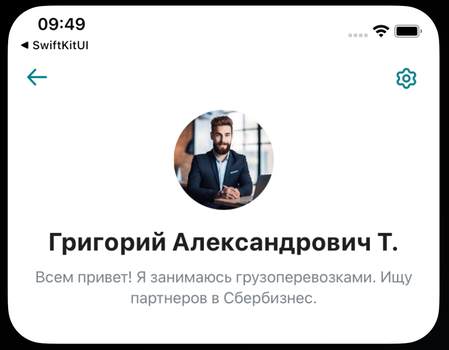
В данной задаче:

- ✳ отслеживать изменение позиции скрола.
- ✳ отслеживать завершение скрола (фазы скрола)
- ✳ выполнять перемещение к нужной позиции (доводчик анимации)



Анимация профиля + Scroll APIs iOS 18 🎉

iOS 18



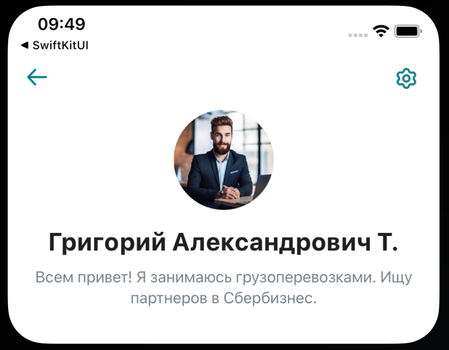
В данной задаче:

- отслеживать изменение позиции скрола.
- отслеживать завершение скрола (фазы скрола)
- выполнять перемещение к нужной позиции (доводчик анимации)

```
.onScrollGeometryChange(for: CGFloat.self) { geometry in
    geometry.contentOffset.y
} action: { _, newValue in
    ...updateScrollViewOffset(newValue)
}
```

Анимация профиля + Scroll APIs iOS 18 🎉

iOS 18



В данной задаче:

- отслеживать изменение позиции скрола.
- отслеживать завершение скрола (фазы скрола)
- выполнять перемещение к нужной позиции (доводчик анимации)

```
.onScrollGeometryChange(for: CGFloat.self) { geometry in
    geometry.contentOffset.y
} action: { _, newValue in
    ...updateScrollViewOffset(newValue)
}
```

Альтернатива до iOS 18: Интроспект + observing

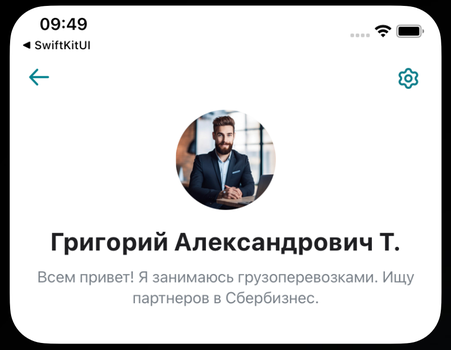
```
.onScrollGeometryChange(for: CGSize.self) { geometry in
    geometry.contentSize
} action: { _, newValue in
    print("ScrollView.contentSize = \(newValue)")
}
```

Анимация профиля + Scroll APIs iOS 18 🎉

iOS 18

В данной задаче:

- ✳ отслеживать изменение позиции скрола.
- ✳ отслеживать завершение скрола (фазы скрола)
- ✳ выполнять перемещение к нужной позиции (доводчик анимации)

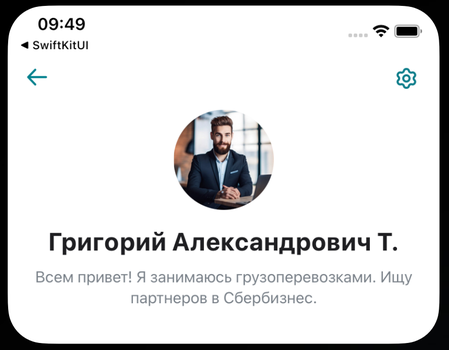



```
.onScrollPhaseChange { _, phase, context in

    /// анимация доводчика после остановки скрола
    if !phase.isScrolling {
        withAnimation {
            scrollPosition.scrollTo(y: offset)
        }
    }
}
```

Альтернатива до iOS 18:
@Publisher или PassthroughSubject +
.debounce(for: 0.05, scheduler: RunLoop.main)

Анимация профиля + Scroll APIs iOS 18 🎉

iOS 18



В данной задаче:

- ✳ отслеживать изменение позиции скрола.
- ✳ отслеживать завершение скрола (фазы скрола)
- ✳ выполнять перемещение к нужной позиции (доводчик анимации)

```
.onScrollPhaseChange { _, phase, context in

    /// анимация доводчика после остановки скрола
    if !phase.isScrolling {
        withAnimation {
            scrollPosition.scrollTo(y: offset)
        }
    }
}
```

Фазы скрола (ScrollPhase)

- ✓ interacting - движение пальцем
- ✓ decelerating - замедление скрола после того, как палец убрали с экрана
- ✓ idle - остановка скрола
- ✓ animating - вызовется при установке скрола к нужной позиции (аналог UIKit.setContentOffset)
- ✓ tracking - не вызывается в данном примере

Анимация профиля + Scroll APIs iOS 18 🎉

iOS 18

В данной задаче:

- ✳ отслеживать изменение позиции скрола.
- ✳ отслеживать завершение скрола (фазы скрола)
- ✳ выполнять перемещение к нужной позиции (доводчик анимации)

```
@State var scrollPosition = ScrollPosition(y: 0)
```

```
@State var scrollPosition = ScrollPosition(idType: MyItem.self)
```

Позиция скрола (ScrollPosition) может использовать

➡ координаты - CGPoint, CGFloat

id объекта списка - Hashable Type

Анимация профиля + Scroll APIs iOS 18 🎉

iOS 18

В данной задаче:

- ✳ отслеживать изменение позиции скрола.
- ✳ отслеживать завершение скрола (фазы скрола)
- ✳ выполнять перемещение к нужной позиции (доводчик анимации)

```
@State var scrollPosition = ScrollPosition(y: 0)

...
    .onScrollPhaseChange { _, phase, context in

        /// анимация доводчика после остановки скрола
        if !phase.isScrolling {
            withAnimation {
                scrollPosition.scrollTo(y: calcPosition)
            }
        }
    }
    .scrollPosition($scrollPosition)
```

Анимация профиля + Scroll APIs iOS 18 🎉

iOS 18

В данной задаче:

- ✳ отслеживать изменение позиции скрола.
- ✳ отслеживать завершение скрола (фазы скрола)
- ✳ выполнять перемещение к нужной позиции (доводчик анимации)

```
@State var scrollPosition = ScrollPosition(y: 0)

...
    .onScrollPhaseChange { _, phase, context in

        /// анимация доводчика после остановки скрола
        if !phase.isScrolling {
            withAnimation {
                scrollPosition.scrollTo(y: calcPosition)
            }
        }
    }
    .scrollPosition($scrollPosition)
```

Анимация профиля + Scroll APIs iOS 18 🎉

iOS 18

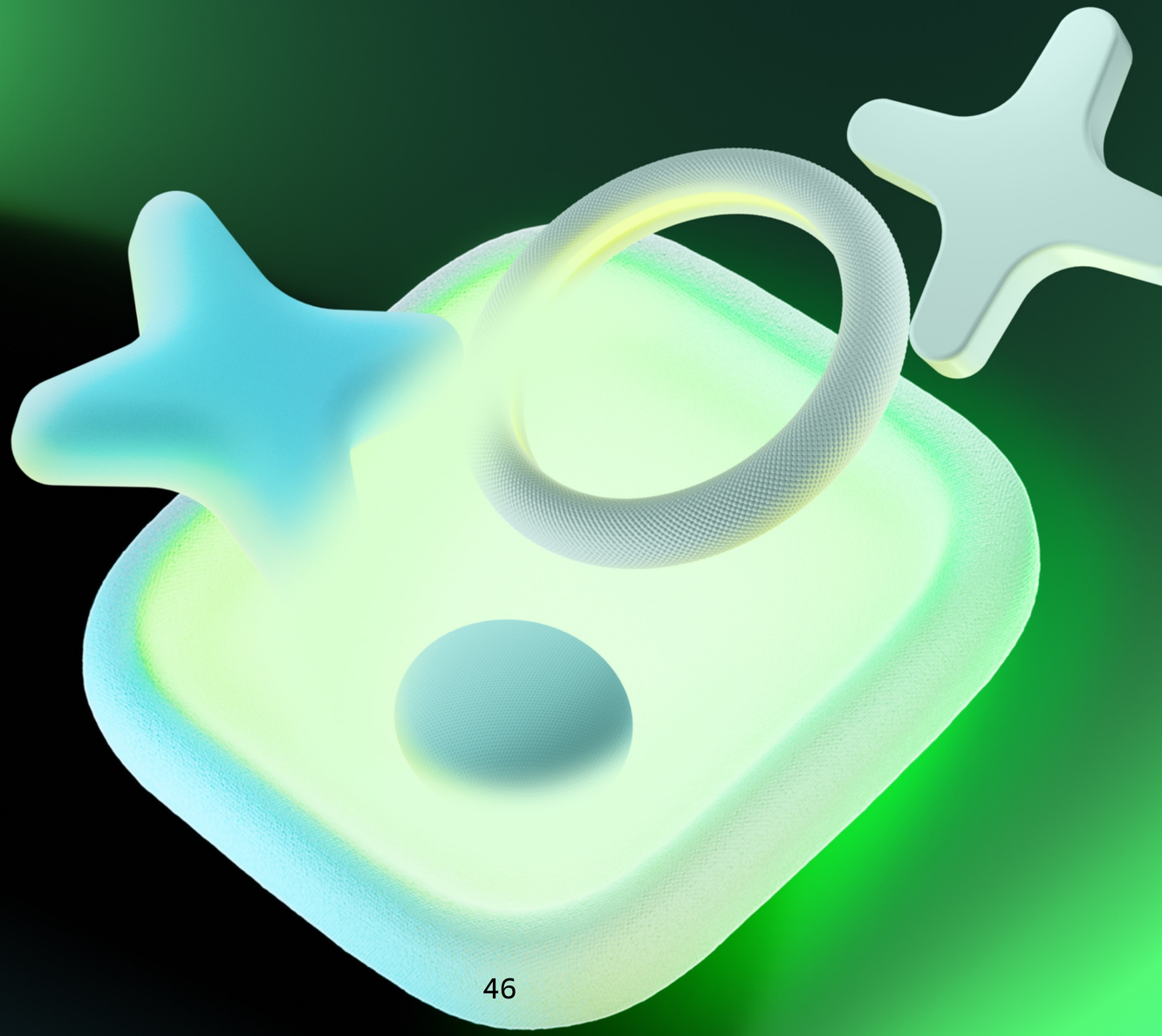
В данной задаче:

- ✳ отслеживать изменение позиции скрола.
- ✳ отслеживать завершение скрола (фазы скрола)
- ✳ выполнять перемещение к нужной позиции (доводчик анимации)

Scroll APIs iOS 18
⚠️ Не поддерживается для **List** ⚠️

Text в SwiftUI

- Что надо знать про Text, делая мобильное приложение;
- Чего ждать от TextRenderer (iOS18) и облегчит ли он нам жизнь;



Text в SwiftUI

Способы управления текстовым представлением, построения Rich Text:

- Объединение Text-ов с использованием модификаторов
- Создание текста через AttributedString (iOS 15)
- TextRenderer (iOS 18) модифицирует текст в процессе его отрисовки

Эти методы могут использоваться одновременно, дополняя друг друга.

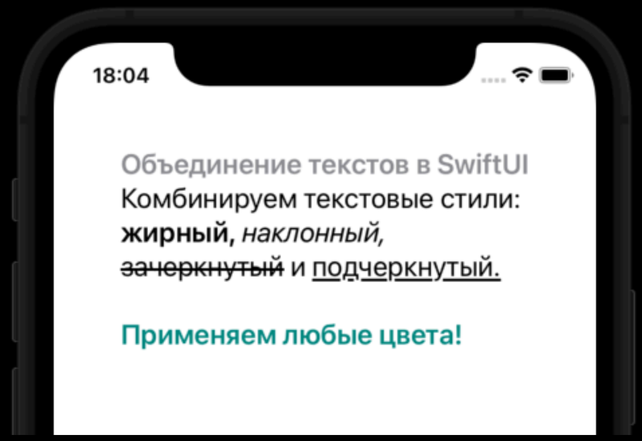
Text в SwiftUI

Способы управления текстовым представлением, построения Rich Text:

- Объединение Text-ов с использованием модификаторов
- Создание текста через AttributedString (iOS 15)
- TextRenderer (iOS 18) модифицирует текст в процессе его отрисовки

Эти методы могут использоваться одновременно, дополняя друг друга.

Объединение текстов

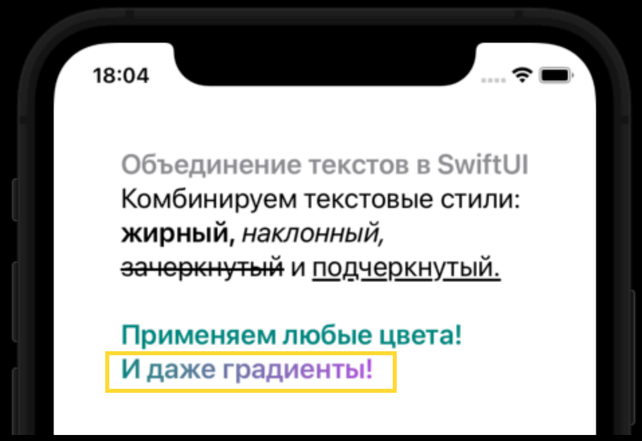



```
var texts: some View {

    Text("Объединение текстов в SwiftUI\n")
        .foregroundStyle(Color.gray).bold() +
    Text("Комбинируем текстовые стили:\n") +
    Text("жирный, ").bold() +
    Text("наклонный,\n").italic() +
    Text("зачеркнутый").strikethrough() + Text(" и ") +
    Text("подчеркнутый.\n\n").underline() +
    Text("Применяем любые цвета!\n")
        .foregroundStyle(Color.accent).bold()
}
```

Объединение текстов формирует один компонент с мультиформатным текстом.
В отладчике и профилере вы увидите единый компонент LazyView<TextUnion>.

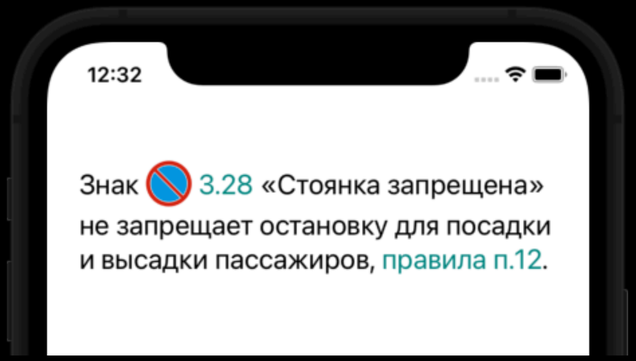
Объединение текстов

iOS 17




```swift
var texts: some View {

    Text("Объединение текстов в SwiftUI\n")
        .foregroundStyle(Color.gray).bold() +
    Text("Комбинируем текстовые стили:\n") +
    Text("жирный, ").bold() +
    Text("наклонный,\n").italic() +
    Text("зачеркнутый").strikethrough() + Text(" и ") +
    Text("подчеркнутый.\n\n").underline() +
    Text("Применяем любые цвета!\n")
        .foregroundStyle(Color.accent).bold() +
    Text("И даже градиенты!")
        .foregroundStyle(
            LinearGradient(
                colors: [.accent, .purple],
                startPoint: .leading,
                endPoint: .trailing
            )
        ).bold()
}
```

Объединение текстов: Text + Image




```swift
var comment: some View {

    Text("Знак ") +
    Text(Image(.sign328)).baselineOffset(-10) +
    Text("...") +
    Text(.init("[правила п.12](https://www.documents...ru)."))
}
```

⚠️ Вставить картинку в SwiftUI.Text можно только через объединение текстов.
AttributedString не поддерживает такой возможности.
NSTextAttachment не работает для SwiftUI.Text.

Text в SwiftUI

Способы управления текстовым представлением, построения Rich Text:

- Объединение Text-ов с использованием модификаторов
- Создание текста через AttributedString (iOS 15)
- TextRenderer (iOS 18) модифицирует текст в процессе его отрисовки

Эти методы могут использоваться одновременно, дополняя друг друга.

AttributedString

iOS 15

AttributedString (iOS 15) - аналог UIKit.NSAttributedString в SwiftUI.
AttributedString и NSAttributedString конвертируются друг в друга, но имеют ряд отличий, например:

- ➡ NSTextAttachment не поддерживается компонентом Text, но вставлять картинки можно через объединение текстов.
- ➡ ParagraphStyle не поддерживается компонентом Text, можно добавлять модификаторы на весь текст (не на части).

AttributedString

iOS 15

AttributedString (iOS 15) - аналог UIKit.NSAttributedString в SwiftUI.
AttributedString и NSAttributedString конвертируются друг в друга, но имеют ряд отличий, например:

- ➡ NSTextAttachment не поддерживается компонентом Text, но вставлять картинки можно через объединение текстов.
- ➡ ParagraphStyle не поддерживается компонентом Text, можно добавлять модификаторы на весь текст (не на части).
  - ➡ Нет инструмента для прекалькуляции размеров текста в SwiftUI
  - ➡ При этом, достаточно сложно привести тексты в UIKit и SwiftUI к одинаковому виду и одинаковому фрейму. Наиболее похожие результаты удается получить устанавливая lineHeightMultiple.

    UIKit.paragraphStyle.lineHeightMultiple
    SwiftUI.environment(\._lineHeightMultiple).

    Однако, UIKit и SwiftUI неодинаково работают с baseline: в UIKit она частично поглощается высотой линии, а в SwiftUI добавляется к ней. Использование других атрибутов шрифта или модификаторов текста может привести к еще большим различиям.

Text в SwiftUI

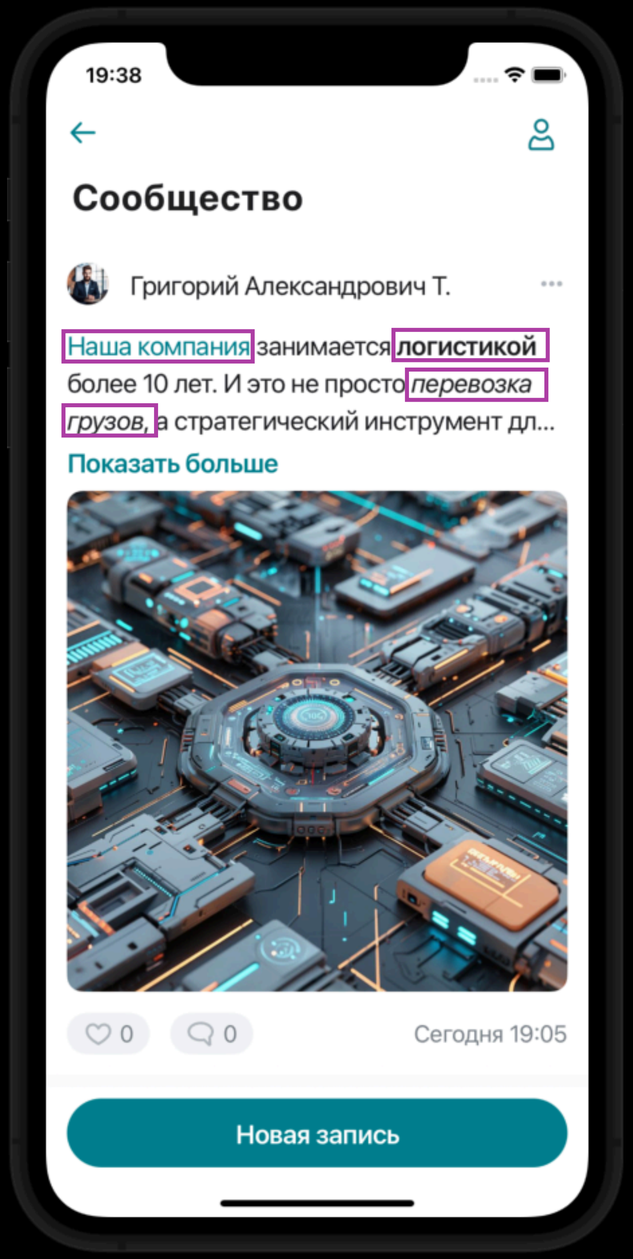
Способы управления текстовым представлением, построения Rich Text:

- Объединение Text-ов с использованием модификаторов
- Создание текста через AttributedString (iOS 15)
- TextRenderer (iOS 18) модифицирует текст в процессе его отрисовки

Эти методы могут использоваться одновременно, дополняя друг друга.

Text в SwiftUI, Markdown

iOS 15

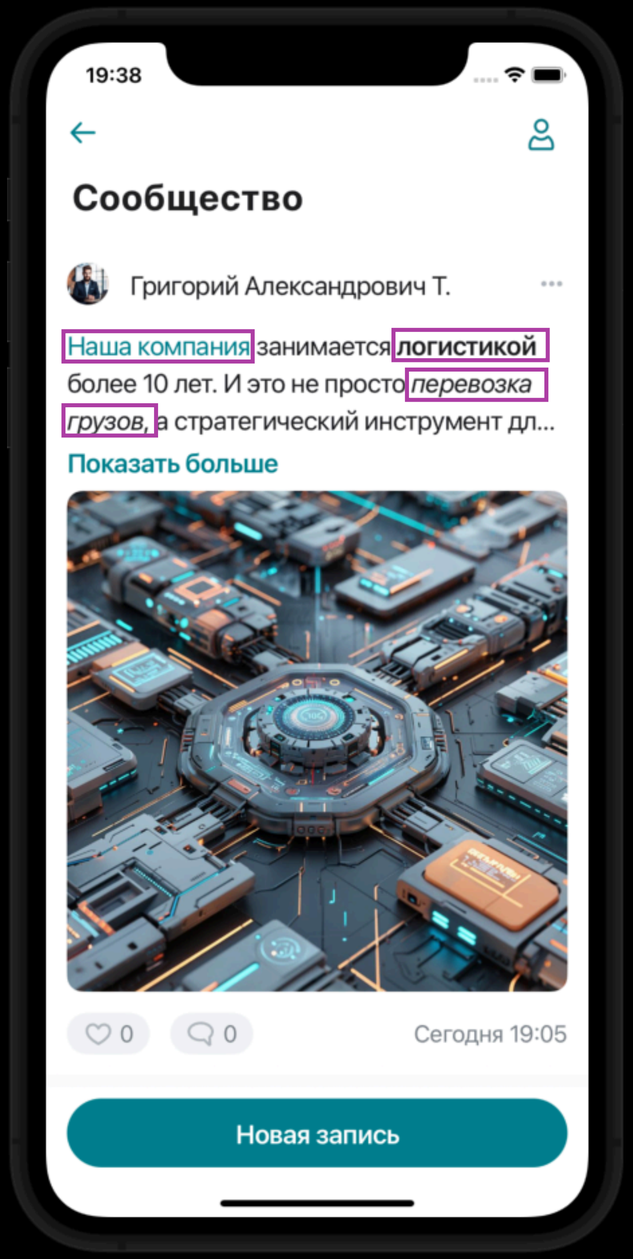


Поддержка Markdown.

- Объединение Text-ов с использованием модификаторов
- Создание текста через AttributedString (iOS 15)

Text в SwiftUI, Markdown

iOS 15



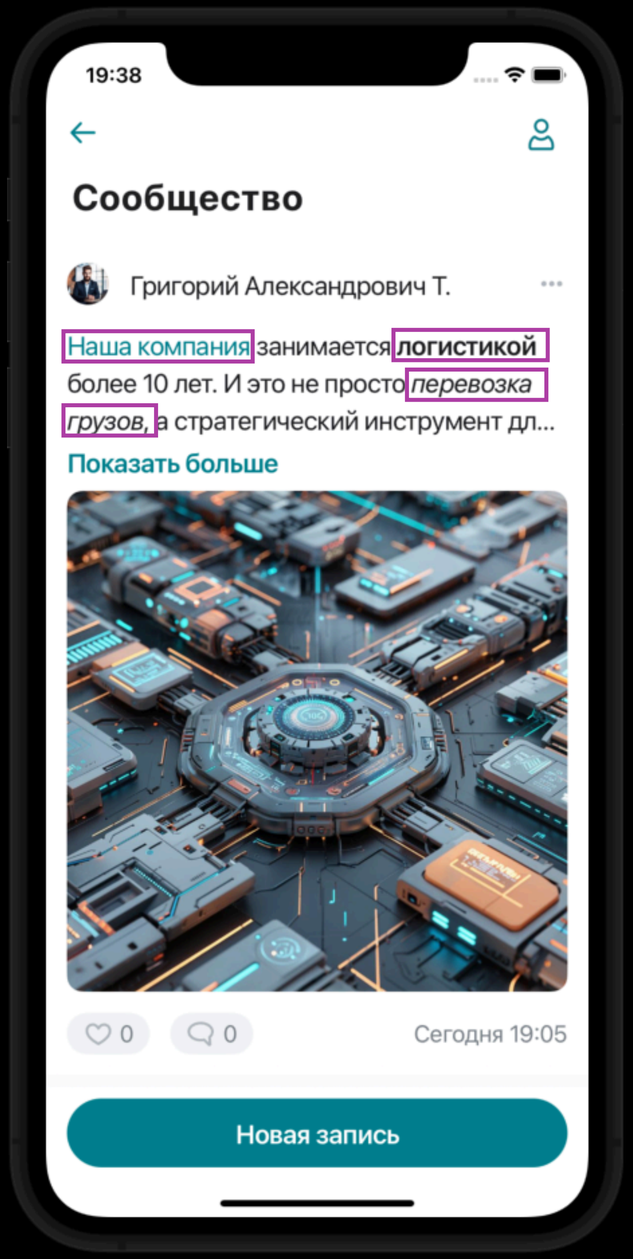
Поддержка Markdown.

- Объединение Text-ов с использованием модификаторов
- Создание текста через AttributedString (iOS 15)

- Markdown поддерживается также для NSAttributedString в UILabel.

Text в SwiftUI, Markdown

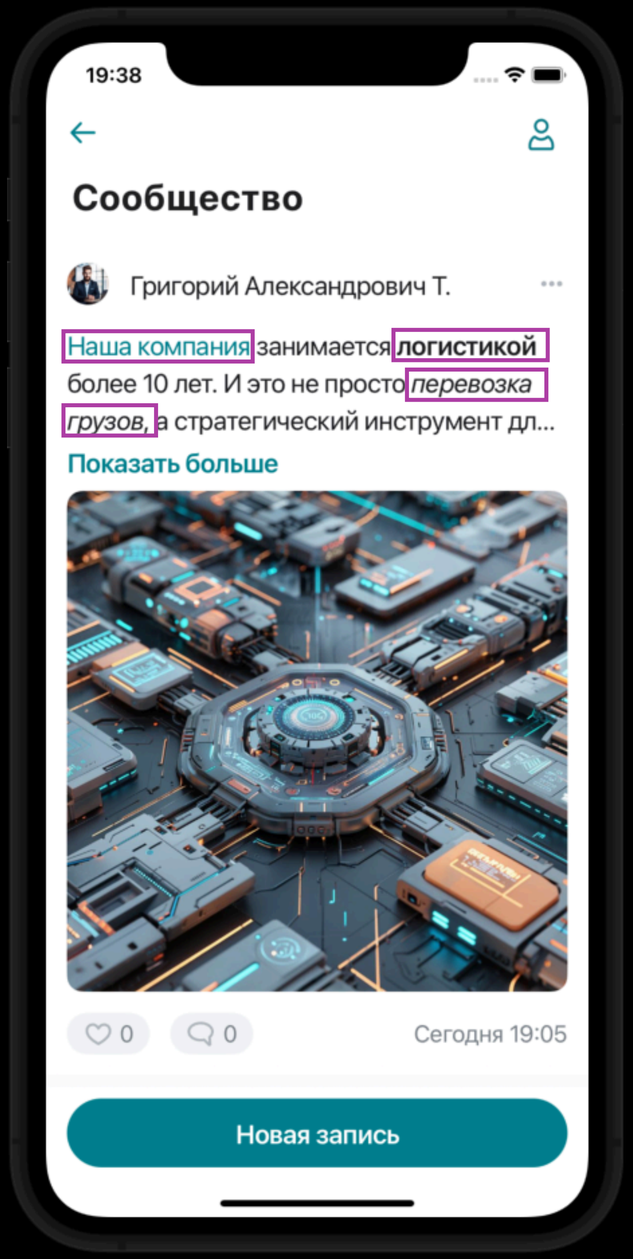
iOS 15



- Markdown - это упрощенный язык разметки для форматирования текстов.
- Часто используется в мессанджерах, соцсетях, при оформлении документации.
- Просто, удобно, не затратно для разработки.

Text в SwiftUI, Markdown

iOS 15



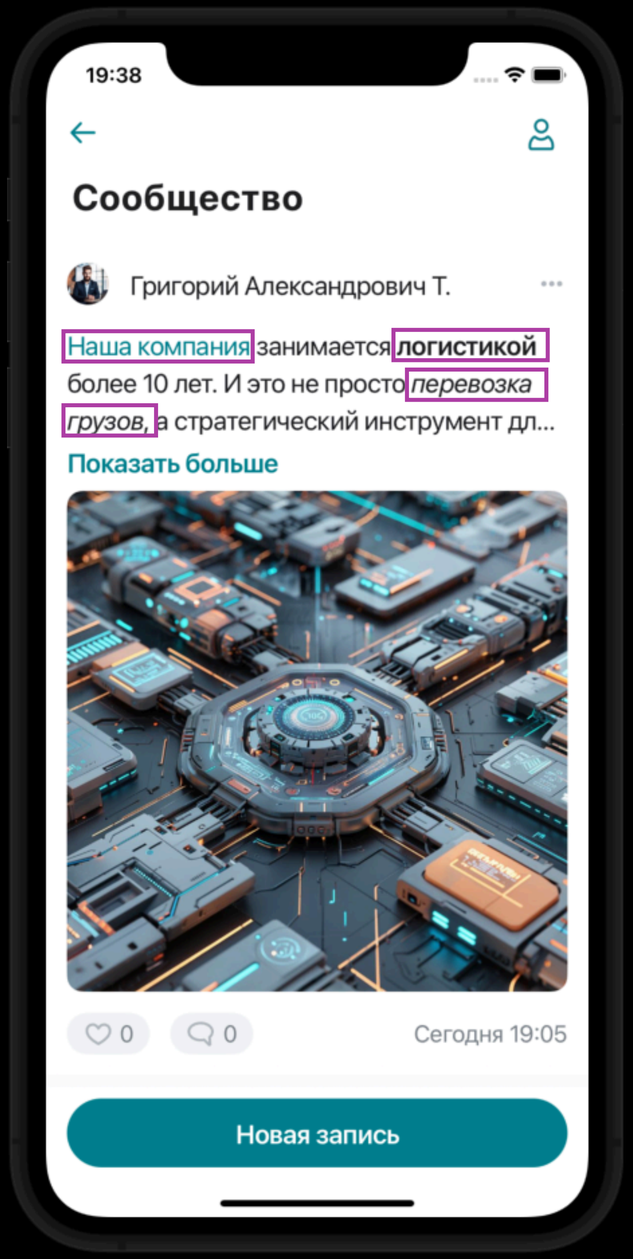
- Markdown - это упрощенный язык разметки для форматирования текстов.
- Часто используется в мессанджерах, соцсетях, при оформлении документации.
- Просто, удобно, не затратно для разработки.

Обычный
Наклонный
Наклонный
Жирный
~~Зачеркнутый~~
`Code`
Link

```
VStack {
    Text("Обычный")
    Text("_Наклонный_")
    Text("*Наклонный*")
    Text("**Жирный**")
    Text("~Зачеркнутый~")
    Text("`Code`")
    Text("[Link](https://www.apple.com)")
}
```

Text в SwiftUI, Markdown

iOS 15



- Markdown - это упрощенный язык разметки для форматирования текстов.
- Часто используется в мессанджерах, соцсетях, при оформлении документации.
- Просто, удобно, не затратно для разработки.
- Ограниченный набор маркдаунов в iOS.

Обычный
Наклонный
Наклонный
Жирный
~~Зачеркнутый~~
`Code`
Link

```
VStack {
    Text("Обычный")
    Text("_Наклонный_")
    Text("*Наклонный*")
    Text("**Жирный**")
    Text("~Зачеркнутый~")
    Text("`Code`")
    Text("[Link](https://www.apple.com)")
}
```

Text в SwiftUI

Способы управления текстовым представлением, построения Rich Text:

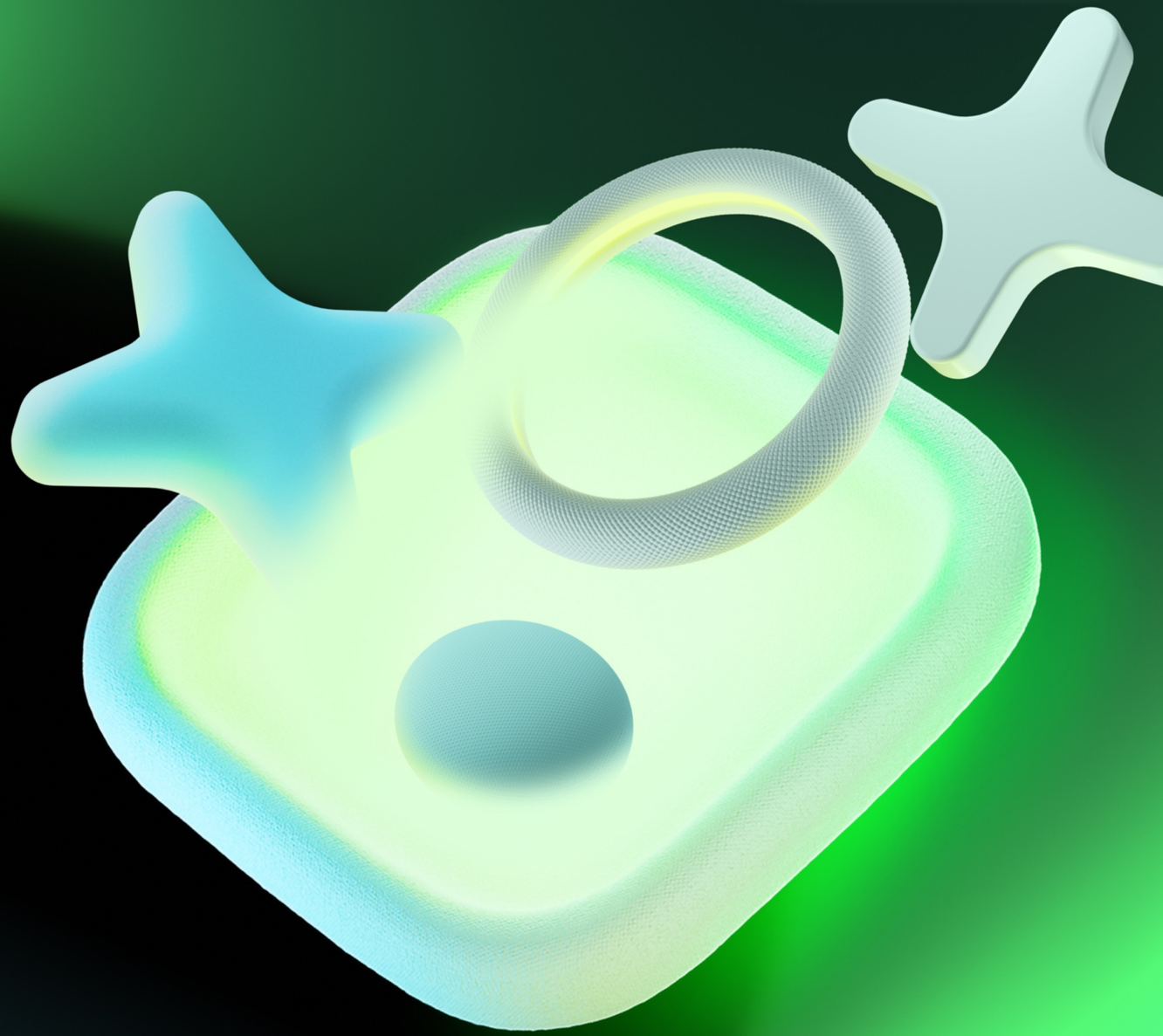
1. Объединение Text-ов с использованием модификаторов
2. Создание текста через AttributedString (iOS 15)
3. TextRenderer (iOS 18) модифицирует текст в процессе его отрисовки

Эти методы могут использоваться одновременно, дополняя друг друга.

iOS 18

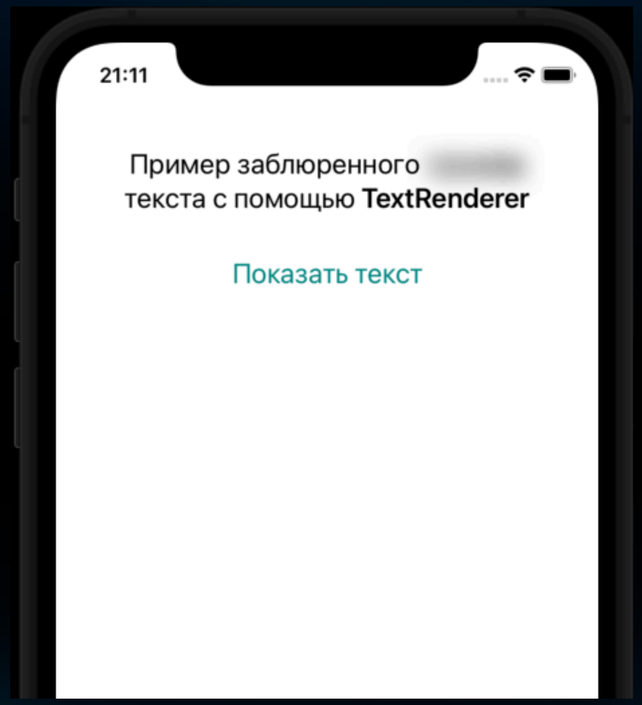
TextRenderer в SwiftUI

Инструмент для работы с Text,
доступный с iOS18



TextRenderer

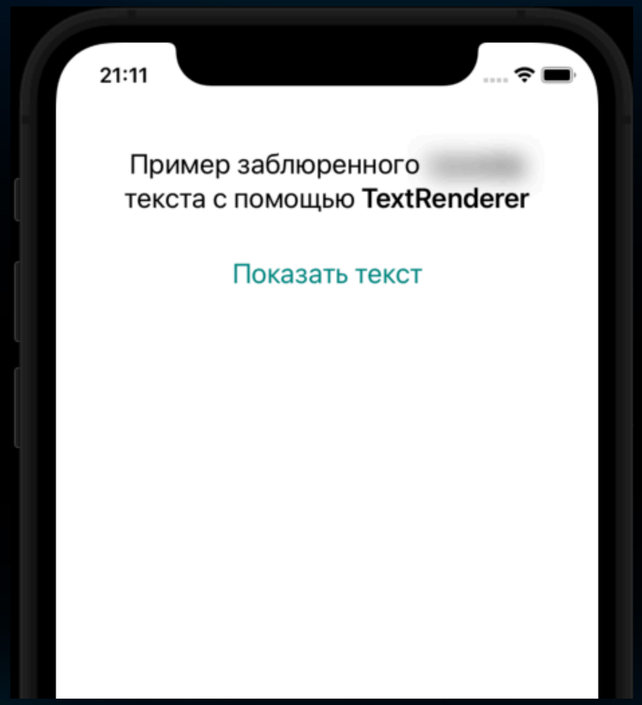
iOS 18



- Как работает TextRenderer
- Какие задачи можно решить с помощью TextRenderer

Как работает TextRenderer

iOS 18

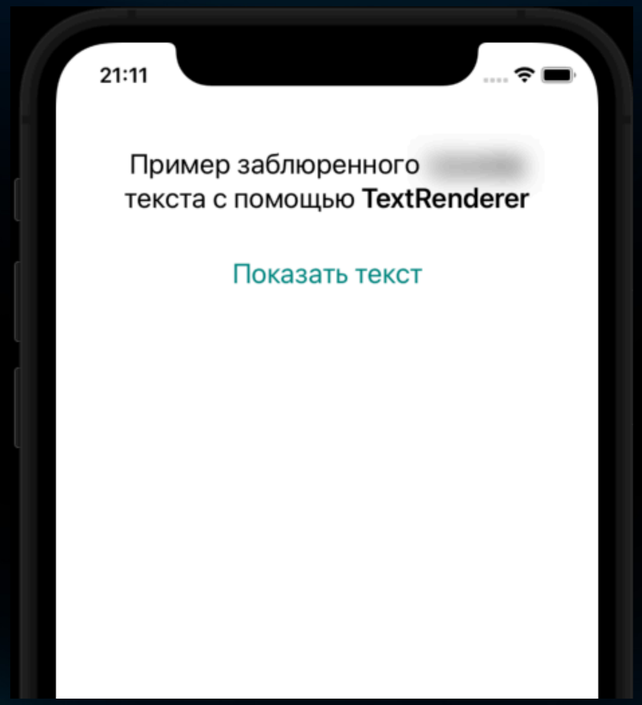



```
let blurredText = Text("123456")

Text("Пример заблюренного \(blurredText) текста с помощью **TextRenderer**")
```

1. Делаем текст с интерполяцией
2. Помечаем текст для блюра текстовым атрибутом BlurAttribute
3. Добавляем обработчик TextRenderer

Как работает TextRenderer

iOS 18



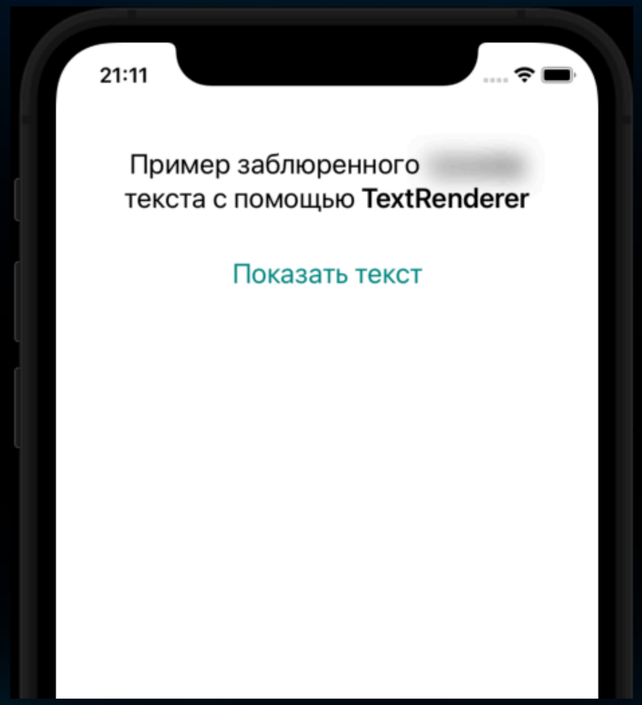

```
let blurredText = Text("123456")
    .customAttribute(BlurAttribute())

Text("Пример заблюренного \(blurredText) текста с помощью **TextRenderer**")
```

1. Делаем текст с интерполяцией
2. Помечаем текст для блюра текстовым атрибутом BlurAttribute
3. Добавляем обработчик TextRenderer

```
struct BlurAttribute: TextAttribute { }
```

Как работает TextRenderer

iOS 18




```
let blurredText = Text("123456")
    .customAttribute(BlurAttribute())

Text("Пример заблюренного \(blurredText) текста с помощью **TextRenderer**")
    .textRenderer(BlurEffect())
```

1. Делаем текст с интерполяцией
2. Помечаем текст для блюра текстовым атрибутом BlurAttribute
3. Добавляем обработчик TextRenderer

```
struct BlurAttribute: TextAttribute { }
struct BlurEffect: TextRenderer {

    func draw(layout: Text.Layout, in context: inout GraphicsContext) {
        ...
    }
}
```

Как работает TextRenderer

iOS 18

```
func draw(layout: Text.Layout, in context: inout GraphicsContext)
```

Позволяет модифицировать текст уже в процессе его отрисовки.
Через метод draw можем получить:

- геометрии составляющих текста через Text.Layout:
  - Строки в тексте
    - Runs - наборы символов с одинаковыми атрибутами
      - Glyphs - символы

- контекст для рисования, через который можно сделать фон, обводку, эффекты (например, блюр)

- Свойства layout.isTruncated

Модификатор TextRenderer

iOS 18

```
func draw(layout: Text.Layout, in context: inout GraphicsContext)
```

Позволяет модифицировать текст уже в процессе его отрисовки.
Через метод draw можем получить:

- геометрии составляющих текста через Text.Layout:
  - Строки в тексте
    - Runs - наборы символов с одинаковыми атрибутами
      - Glyphs - символы
- контекст для рисования, через который можно сделать фон, обводку, эффекты (например, блюр)
- Свойства layout.isTruncated

Модификатор TextRenderer

iOS 18

```
func draw(layout: Text.Layout, in context: inout GraphicsContext)
```

Позволяет модифицировать текст уже в процессе его отрисовки.
Через метод draw можем получить:

- ✳ геометрии составляющих текста через Text.Layout:
  - ▸ Строки в тексте
    - ▸ Runs - наборы символов с одинаковыми атрибутами
      - ▸ Glyphs - символы
- ✳ контекст для рисования, через который можно сделать фон, обводку, эффекты (например, блюр)
- ✳ Свойства layout.isTruncated

```
func draw(
    layout: Text.Layout,
    in context: inout GraphicsContext
) {

    for line in layout {
        for run in line {
            for glyph in run {
                ...
                let rect = glyph.typographicBounds.rect
                context.addFilter(...)
                context.draw(glyph)
            }
        }
    }
}
```

Модификатор TextRenderer

iOS 18

```
func draw(layout: Text.Layout, in context: inout GraphicsContext)
```

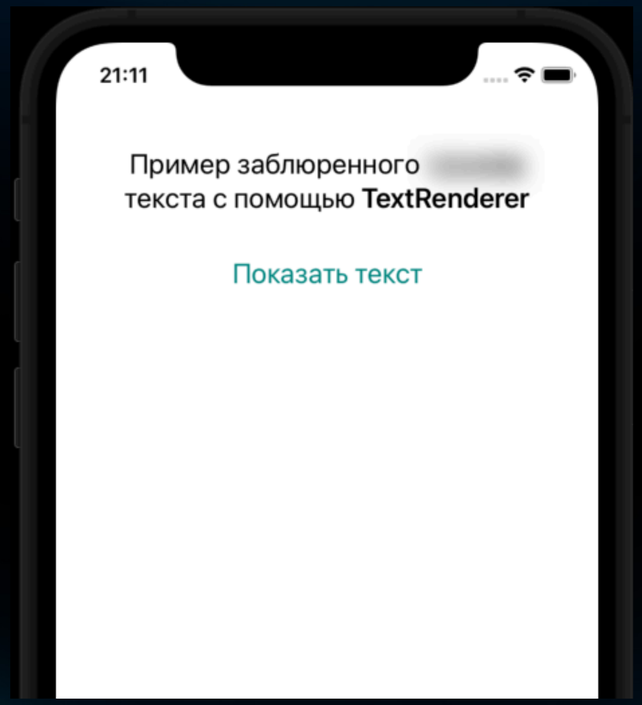
Позволяет модифицировать текст уже в процессе его отрисовки.
Через метод draw можем получить:

- ✳ геометрии составляющих текста через Text.Layout:
  - ‣ Строки в тексте
    - ‣ Runs - наборы символов с одинаковыми атрибутами
      - ‣ Glyphs - символы
- ✳ контекст для рисования, через который можно сделать фон, обводку, эффекты (например, блюр)
- ✳ Свойства layout.isTruncated

```
func draw(
    layout: Text.Layout,
    in context: inout GraphicsContext
) {

    for line in layout {
        for run in line {
            for glyph in run {
                ...
                let rect = glyph.typographicBounds.rect
                context.addFilter(...)
                context.draw(glyph)
            }
        }
    }
}
```

Модификатор TextRenderer

iOS 18

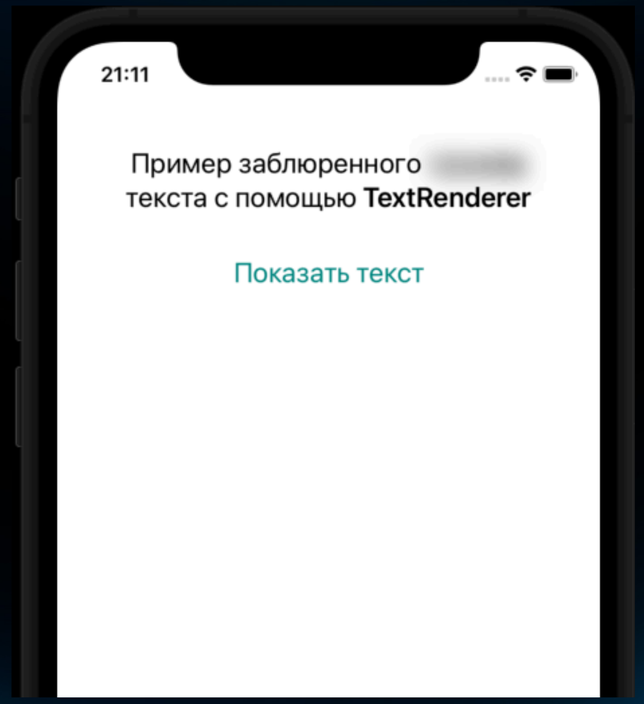



```
let blurredText = Text("123456")
    .customAttribute(BlurAttribute())

Text("Пример заблюренного \(blurredText) текста с помощью **TextRenderer**")
    .textRenderer(BlurEffect())
```

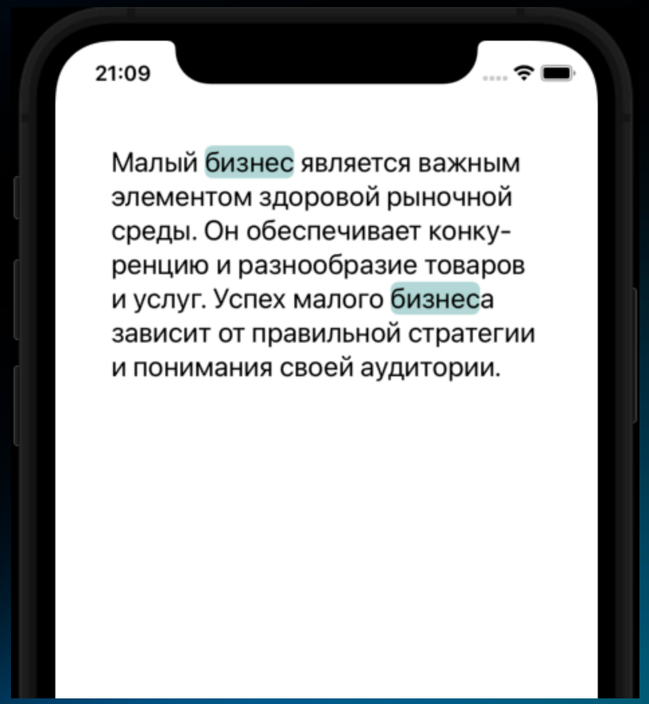
```
struct BlurAttribute: TextAttribute { }
struct BlurEffect: TextRenderer {

func draw(layout: Text.Layout, in context: inout GraphicsContext) {
    for run in layout.lines {
        if run[BlurAttribute.self] != nil {
            var copyContext = context
            copyContext.addFilter(.blur(radius: radius))
            copyContext.draw(run)
        }
        ...
    }
}
```

С помощью TextRenderer можно:

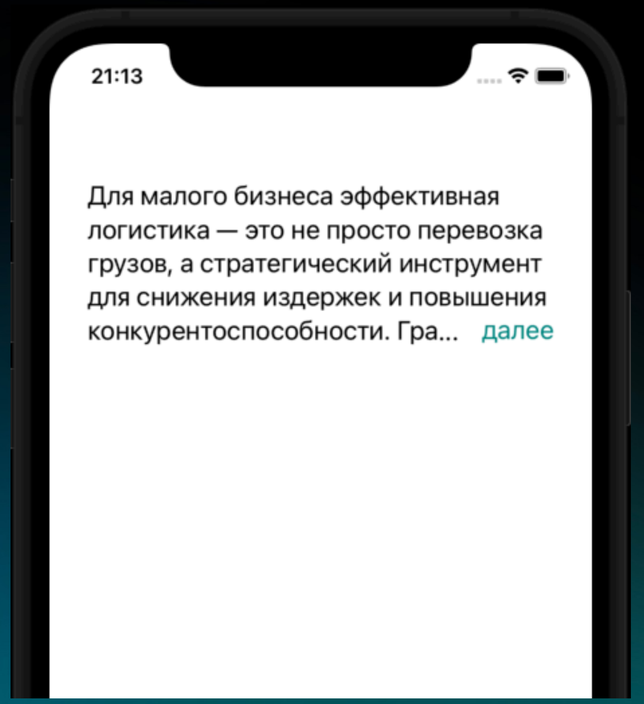
iOS 18



Использовать эффекты



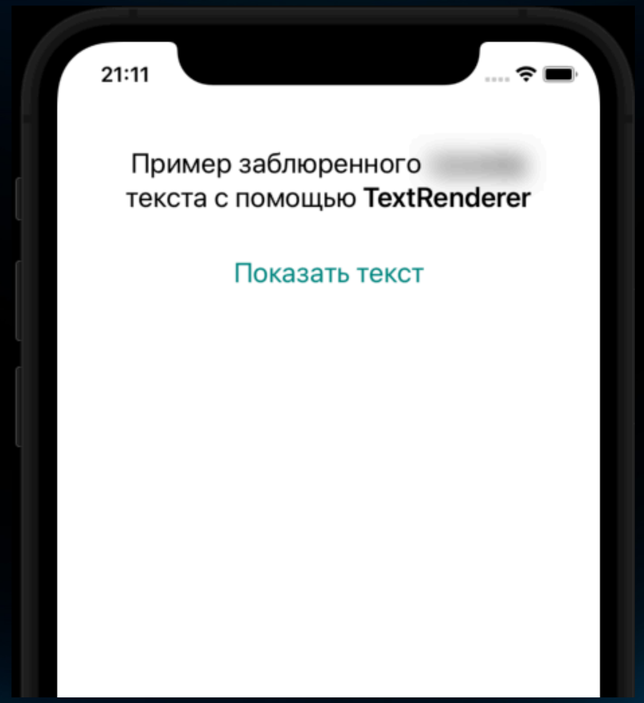
Получать размеры текста, учитывать их в верстке. Рисовать фон, обводку и тп.



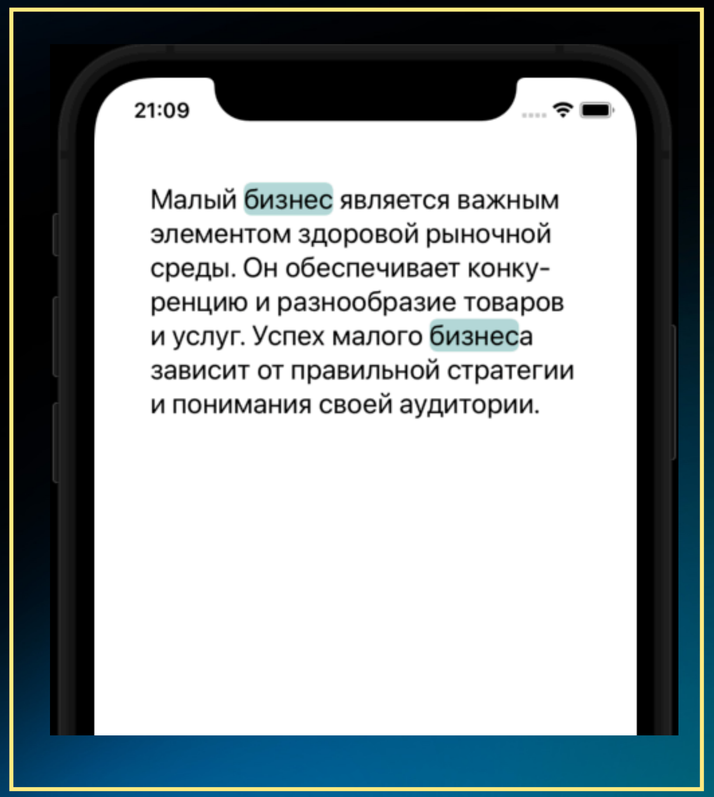
Отслеживать и обрабатывать обрезанный текст lineLimit

С помощью TextRenderer можно:

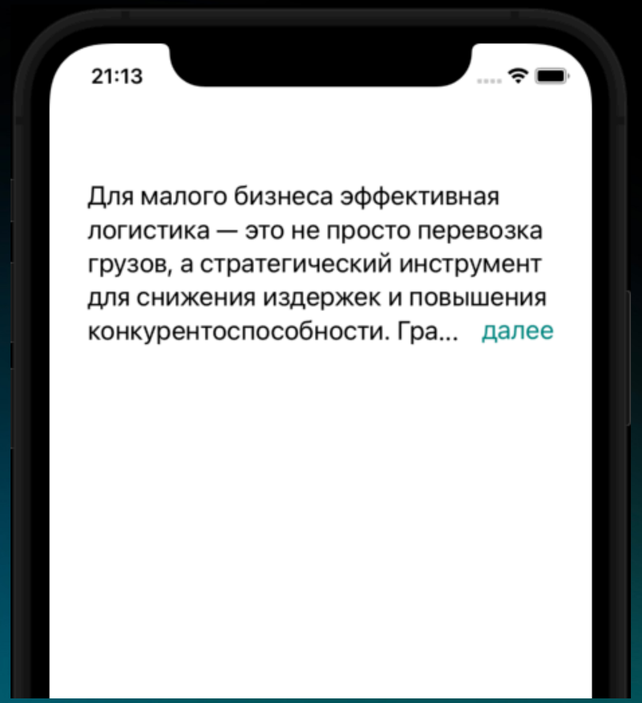
iOS 18



Использовать эффекты



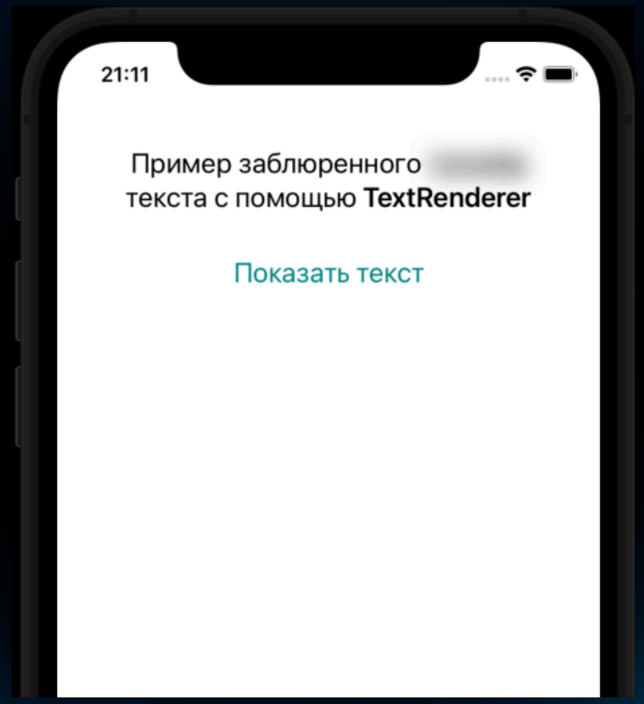
Получать размеры текста, учитывать их в верстке. Рисовать фон, обводку и тп.



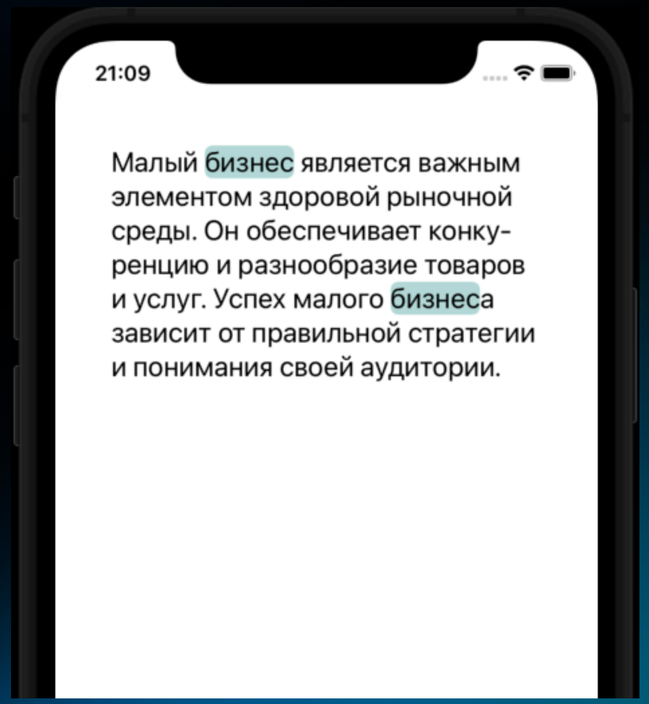
Отслеживать и обрабатывать обрезанный текст lineLimit

С помощью TextRenderer можно:

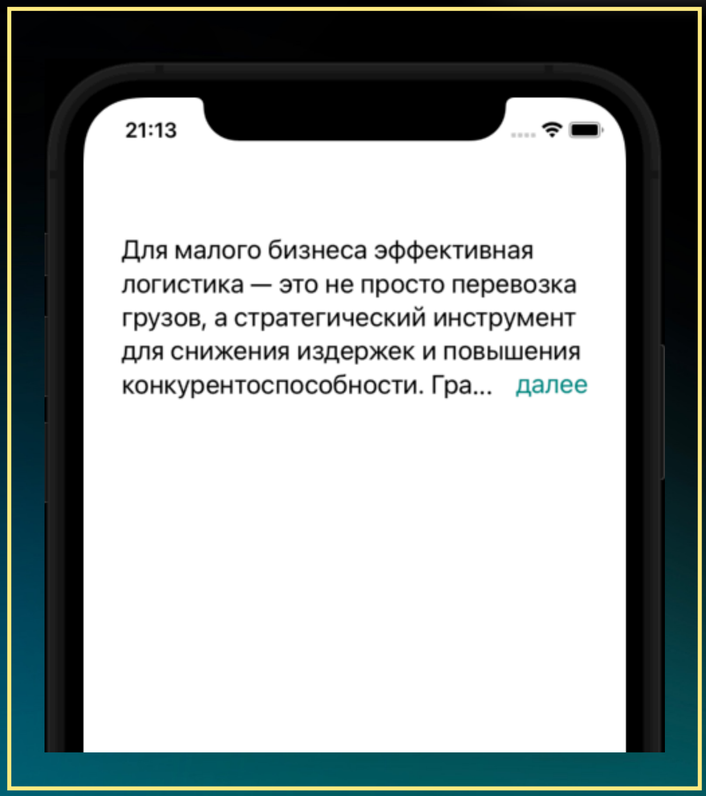
iOS 18



Использовать эффекты



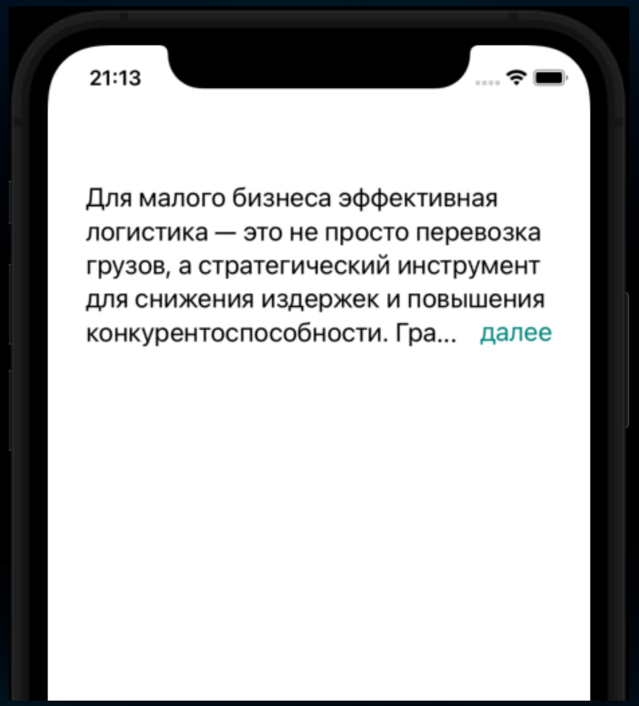
Получать размеры текста, учитывать их в верстке. Рисовать фон, обводку и тп.



Отслеживать и обрабатывать обрезанный текст lineLimit

Модификатор TextRenderer

iOS 18




```
func draw(layout: Text.Layout, in context: inout GraphicsContext)
```

Позволяет модифицировать текст уже в процессе его отрисовки.
Через метод draw можем получить:

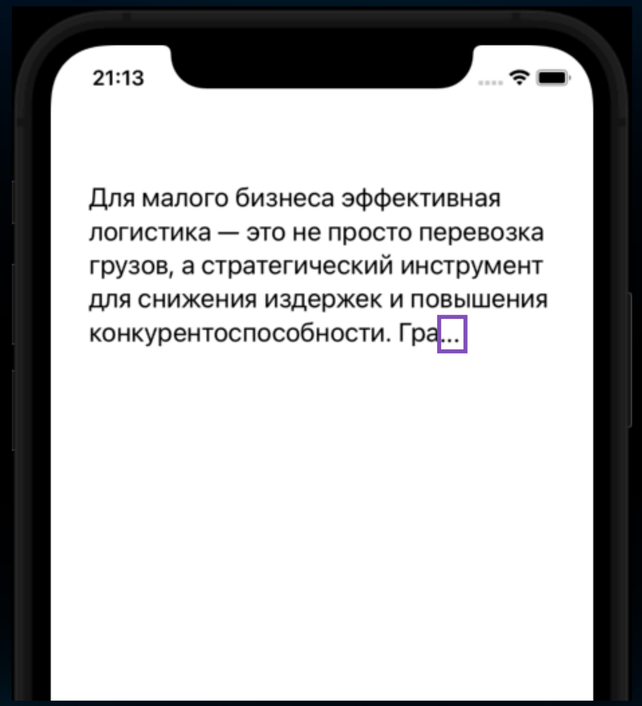
- геометрии составляющих текста через Text.Layout:
  - Строки в тексте
    - Runs - наборы символов с одинаковыми атрибутами
      - Glyphs - символы

- контекст для рисования, через который можно сделать фон, обводку, эффекты (например, блюр)

- Свойство layout.isTruncated позволяет узнать обрезан ли текст через ограничение lineLimit.

Модификатор TextRenderer

iOS 18

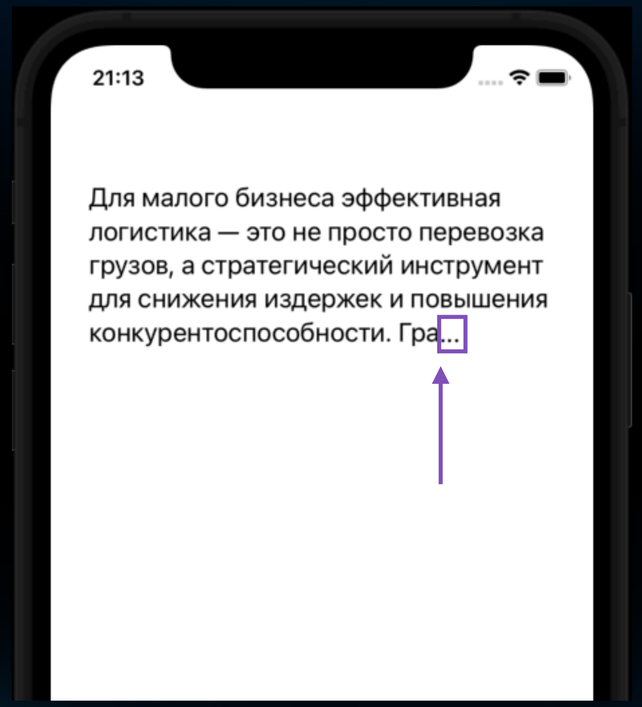


Свойства layout.isTruncated

Задача: нужно обрезать текст, чтобы троеточие было не в конце строки и оставалось место для дополнительного элемента.

Модификатор TextRenderer

iOS 18



Свойства layout.isTruncated

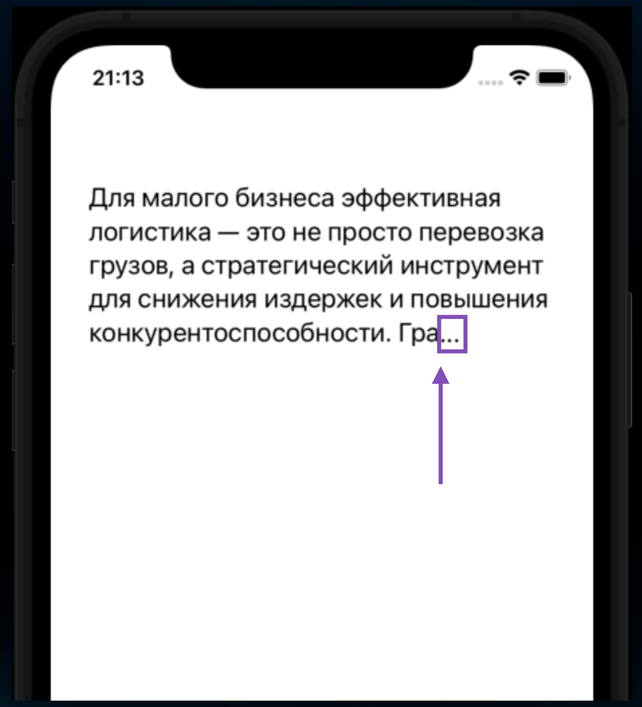
Решение: при отрисовке текста (в методе draw) нужно:

- Проверить, что текст обрезан =>
  - В последней строке по фрейму символа определяем, нужно ли его рисовать
  - Добавляем троеточие

```
func draw(layout: Text.Layout, in context: inout GraphicsContext) {

    if layout.isTruncated {
        ...
        /// Перебираем символы последней линии
        for glyph in lastLine.flatMap(\.self) {

            let rect = glyph.typographicBounds.rect
            if rect.maxX > maxX {

                context.draw(Text("..."), in: rect.width(50))
                ...
            }
        } ...
}
```

Модификатор TextRenderer

iOS 18



Свойства layout.isTruncated

Решение: при отрисовке текста (в методе draw) нужно:

- Проверить, что текст обрезан =>
  - В последней строке по фрейму символа определяем, нужно ли его рисовать
  - Добавляем троеточие

```
func draw(layout: Text.Layout, in context: inout GraphicsContext) {

    if layout.isTruncated {
        ...
        /// Перебираем символы последней линии
        for glyph in lastLine.flatMap(\.self) {

            let rect = glyph.typographicBounds.rect
            if rect.maxX > maxX {

                context.draw(Text("..."), in: rect.width(50))
                ...
            }
        } ...
}
```

Модификатор TextRenderer

iOS 18

```
public protocol TextRenderer: Animatable {
    func draw(layout: Text.Layout, in context: inout GraphicsContext)
    func sizeThatFits(proposal: ProposedViewSize, text: TextProxy) -> CGSize
    var displayPadding: EdgeInsets { get }
}
```

- Основная магия происходит в методе draw, который позволяет модифицировать текст уже в процессе его отрисовки;
- Дает возможность изменить размеры текста и отступы;
- TextRenderer конформится к Animatable, т.е. изменение параметров можно анимировать.

Модификатор TextRenderer

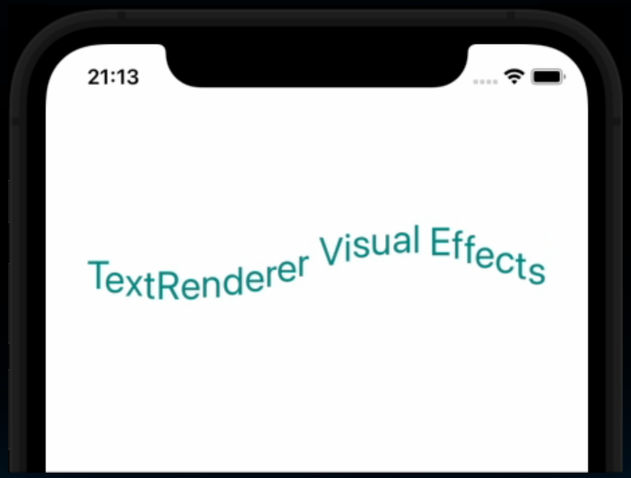
iOS 18

```
public protocol TextRenderer: Animatable {
    func draw(layout: Text.Layout, in context: inout GraphicsContext)
    func sizeThatFits(proposal: ProposedViewSize, text: TextProxy) -> CGSize
    var displayPadding: EdgeInsets { get }
}
```

- Основная магия происходит в методе draw, который позволяет модифицировать текст уже в процессе его отрисовки;
- Дает возможность изменить размеры текста и отступы;
- TextRenderer конформится к Animatable, т.е. изменение параметров можно анимировать.

Модификатор TextRenderer

iOS 18



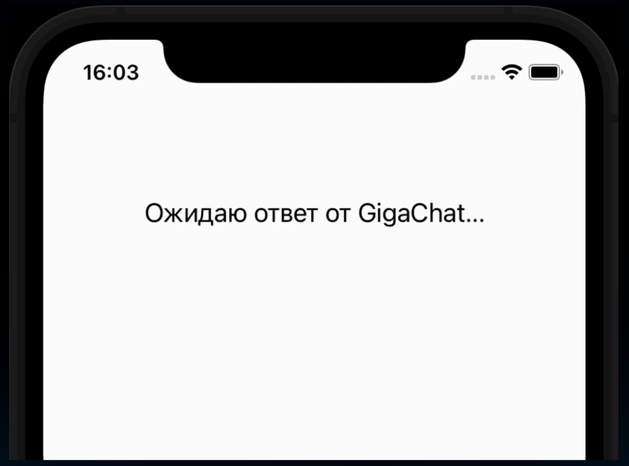
Существует много примеров с анимацией, которую можно построить с помощью TextRenderer. Такая анимация основана:

- на применении модификаторов translate, rotate и тп к контексту составляющих текста: line, run, glyph.

Пример анимации взят с сайта:
https://fatbobman.com/en/posts/creating-stunning-dynamic-text-effects-with-textrender

Модификатор TextRenderer

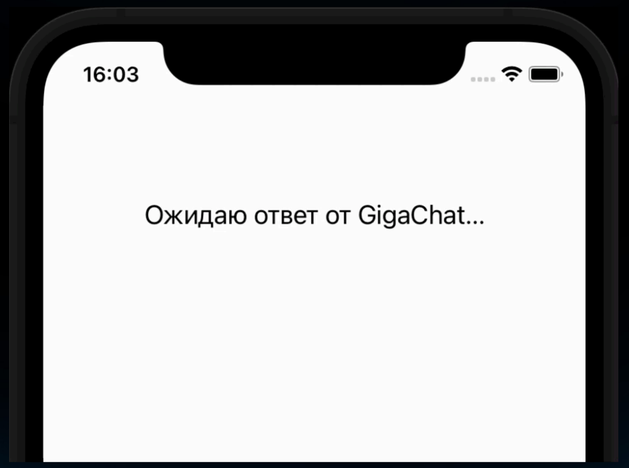
iOS 18

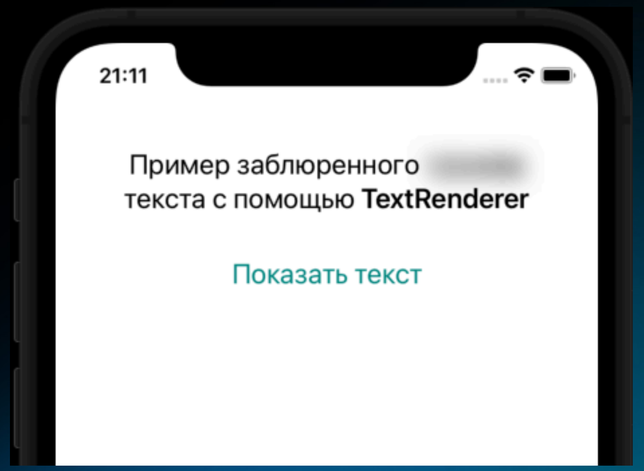


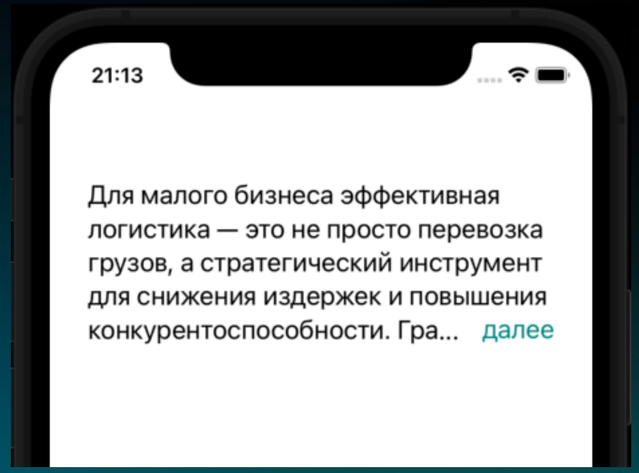

```
func draw(layout: Text.Layout, in context: inout GraphicsContext) {

    for (index, glyph) in layout.glyphs.enumerated() {

        /// анимируем 3 последних символа
        if index >= layout.glyphs.count - 3 {
            ...
            let offset = calcOffset(index, count)
            context.translateBy(x: 0, y: offset)

        }
        context.draw(glyph)
    }
```

Модификатор TextRenderer

iOS 18







Примеры использования
TextRenderer на GitHub

Модификатор TextRenderer

iOS 18

- Позволяет делать много интересных анимаций;
- Различные эффекты для текста: блюр, обрезка, замена, фон, ободка, рисование;
- Изменение фрейма и отступов в тексте (например, резерв места для дополнительной строки); Учитывать что изменение параметров верстки в методе draw применится только в следующем кадре;
- Использование как средство для отладки: через рендеринг, мы можем заглянуть в финальные атрибуты текста;
- Можем получать размеры области текста, помеченной TextAttribute;
  - Атрибуты AttributedString недоступны ;
  - Не может использоваться в фоновом режиме, в том числе, для прекалькуляции размеров текста.

Модификатор TextRenderer

iOS 18

- Позволяет делать много интересных анимаций;
- Различные эффекты для текста: блюр, обрезка, замена, фон, ободка, рисование;
- Изменение фрейма и отступов в тексте (например, резерв места для дополнительной строки); Учитывать что изменение параметров верстки в методе draw применится только в следующем кадре;
- Использование как средство для отладки: через рендеринг, мы можем заглянуть в финальные атрибуты текста;
- Можем получать размеры области текста, помеченной TextAttribute;
  - Атрибуты AttributedString недоступны ;
  - Не может использоваться в фоновом режиме, в том числе, для прекалькуляции размеров текста.

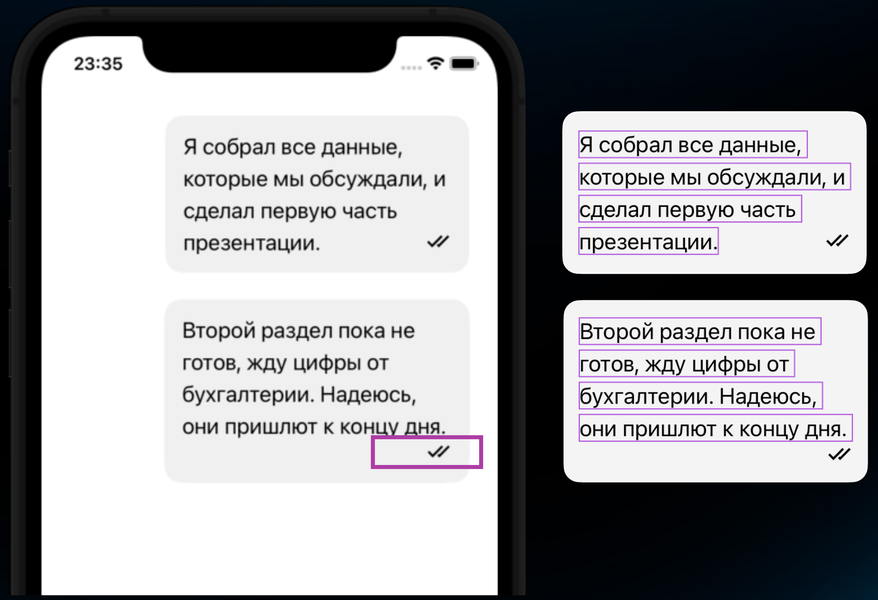
Модификатор TextRenderer

iOS 18

- Позволяет делать много интересных анимаций;
- Различные эффекты для текста: блюр, обрезка, замена, фон, ободка, рисование;
- Изменение фрейма и отступов в тексте (например, резерв места для дополнительной строки); Учитывать что изменение параметров верстки в методе draw применится только в следующем кадре;
- Использование как средство для отладки: через рендеринг, мы можем заглянуть в финальные атрибуты текста;
- Можем получать размеры области текста, помеченной TextAttribute;
  - Атрибуты AttributedString недоступны ;
  - Не может использоваться в фоновом режиме, в том числе, для прекалькуляции размеров текста.

Модификатор TextRenderer

iOS 18



⚠ Важно:

Если в методе TextRenderer draw мы захотим изменить верстку (размеры) нашего View?

Во втором сообщении нет места для checkmark и надо зарезервировать дополнительную строку.

Изменение верстки не обновит текущий кадр, и сработает только в следующем.

Модификатор TextRenderer

iOS 18

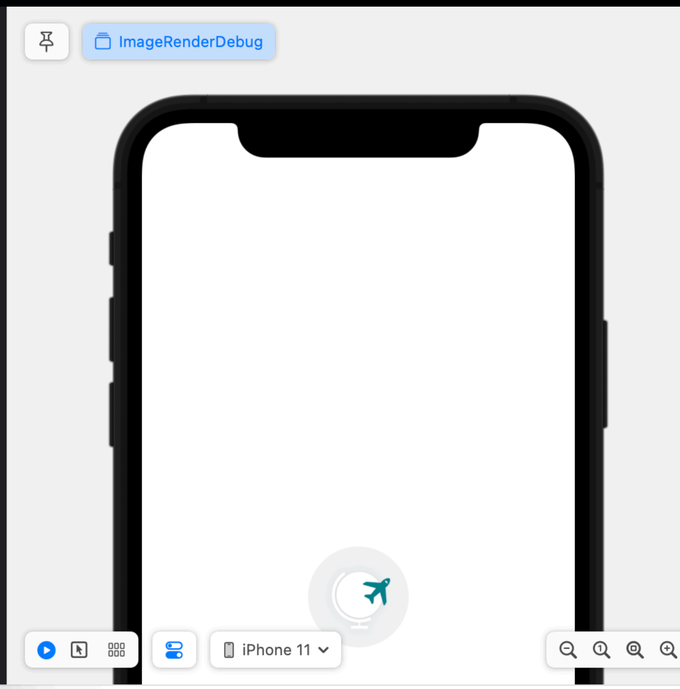
- Позволяет делать много интересных анимаций;
- Различные эффекты для текста: блюр, обрезка, замена, фон, ободка, рисование;
- Изменение фрейма и отступов в тексте (например, резерв места для дополнительной строки); Учитывать что изменение параметров верстки в методе draw применится только в следующем кадре;
- Использование как средство для отладки: через рендеринг, мы можем заглянуть в финальные атрибуты текста;
- Можем получать размеры области текста, помеченной TextAttribute;
  - Атрибуты AttributedString недоступны ;
  - Не может использоваться в фоновом режиме, в том числе, для прекалькуляции размеров текста.

Через TextRenderer, можно заглянуть в финальные атрибуты текста в момент отрисовки.

Вопрос: почему картинка, добавленная через NSAttributedString не отображается. А через объединение текстов - отображается.

Ответ: представления имеет разный внутренний тип: SwiftUITextAttachment и _UIImageTextAttachment.

```
@available(iOS 18.0, *)
struct ImageRenderDebug: View {

    var body: some View {
        Text(Image(.travel))
            .textRenderer(DebugRenderer())
    }
}

struct DebugRenderer: TextRenderer {

    func draw(layout: Text.Layout, in context: inout
        GraphicsContext) {
        for run in layout.flatMap(\.self) {
            print(run)
            context.draw(run)
        }
    }
}
```



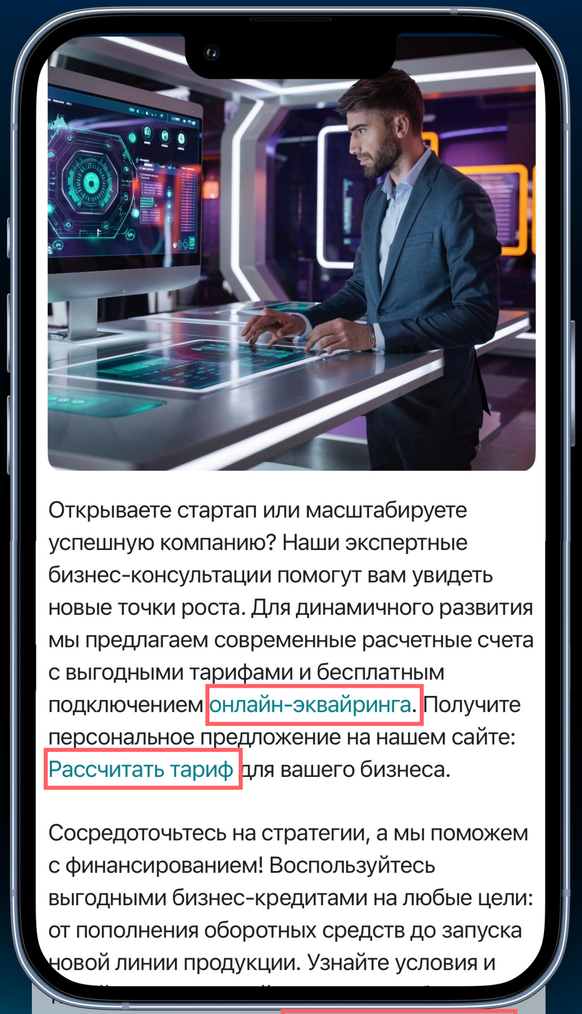
Line: 5 Col: 66

```
<CTRun: 0x103409e60>{string range = (0, 1), string = "\uFFFC", attributes = {
    CTRunDelegate = "<CTRunDelegate 0x60000261c300 [0x1e006f658]>";
    NSAttachment = "<SwiftUI.SwiftUITextAttachment: 0x600003d103c0>";
    NSColor = "UIExtendedSRGBColorSpace 1 1 1 1";
    NSFont = "<UICTFont: 0x103315890> font-family: \"UICTFontTextStyleBody\"; font-weight: normal;
font-style: normal; font-size: 17.00pt";
    NSParagraphStyle = "Alignment Left, LineSpacing 0, ParagraphSpacing 0, ParagraphSpacingBefore 0,
HeadIndent 0, TailIndent 0, FirstLineHeadIndent 0, LineHeight 0/0, LineHeightMultiple 0, LineBreakMode
WordWrapping, Tabs (\n    28L,\n    56L,\n    84L,\n    112L,\n    140L,\n    168L,\n    196L,\n    224L,\n
   252L,\n    280L,\n    308L,\n    336L\n), DefaultTabInterval 0, Blocks (\n), Lists (\n),
BaseWritingDirection Natural, HyphenationFactor 0, TighteningForTruncation NO, HeaderLevel 0
LineBreakStrategy 65535 PresentationIntents (\n) ListIntentOrdinal 0 CodeBlockIntentLanguageHint ''";
}, {width = 96, glyphs = (
```

Модификатор TextRenderer

iOS 18

- Позволяет делать много интересных анимаций;
- Различные эффекты для текста: блюр, обрезка, замена, фон, ободка, рисование;
- Изменение фрейма и отступов в тексте (например, резерв места для дополнительной строки); Учитывать что изменение параметров верстки в методе draw применится только в следующем кадре;
- Использование как средство для отладки: через рендеринг, мы можем заглянуть в финальные атрибуты текста;
- Можем получать размеры области текста, помеченной TextAttribute;
  - Атрибуты AttributedString недоступны ;
  - Не может использоваться в фоновом режиме, в том числе, для прекалькуляции размеров текста.



Открываете стартап или масштабируете успешную компанию? Наши экспертные бизнес-консультации помогут вам увидеть новые точки роста. Для динамичного развития мы предлагаем современные расчетные счета с выгодными тарифами и бесплатным подключением онлайн-эквайринга. Получите персональное предложение на нашем сайте: Рассчитать тариф для вашего бизнеса.

Сосредоточьтесь на стратегии, а мы поможем с финансированием! Воспользуйтесь выгодными бизнес-кредитами на любые цели: от пополнения оборотных средств до запуска новой линии продукции. Узнайте условия и

подайте заявку онлайн Кредит для бизнеса. Наши решения помогут вам легко пройти через кассовые разрывы и сезонные колебания.

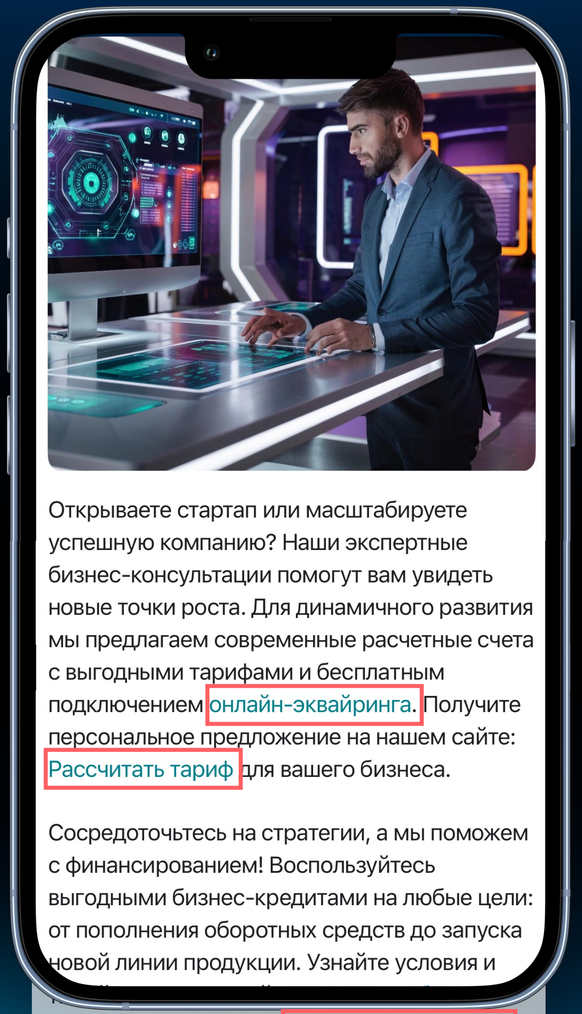
Мы ценим ваше время, поэтому все финансовые операции — от открытия счета до управления платежами — доступны в любое время в удобном мобильном приложении. Доверьте банковское обслуживание нам, чтобы вы могли уделять больше времени главному — развитию своего дела. Давайте строить успех вместе!

Как узнать какие ссылки в большом тексте показались на экране для аналитики?

```
func draw(layout: Text.Layout, in context: inout GraphicsContext) {

    for run in layout.flatMap(\.self) {

        /// Получили фрейм атрибута
        let rect = run.typographicBounds.rect

        ...

    }
}
```

Казалось бы простая задача.
Можно получить фрейм для run, но:

Атрибуты AttributedString недоступны в TextRenderer
Только TextAttribute можно использовать в TextRenderer



Такую задачу можно выполнить только через создание дополнительных атрибутов TextAttribute

```
struct LinkTextAttribute: TextAttribute {
    var link: String
}

func draw(layout: Text.Layout, in context: inout GraphicsContext) {

    for run in layout.flatMap(\.self) {

        if run[LinkTextAttribute.self] != nil {

            /// Получили фрейм атрибута
            let rect = run.typographicBounds.rect

            ...
        }
    }
}
```

Можно, но сложным путем.

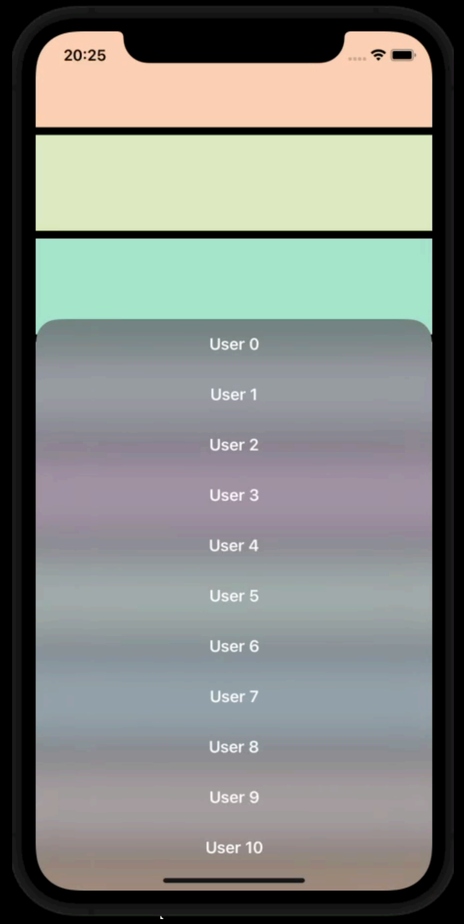
Атрибуты AttributedString недоступны в TextRenderer
Только TextAttribute можно использовать в TextRenderer

Модификатор TextRenderer

iOS 18

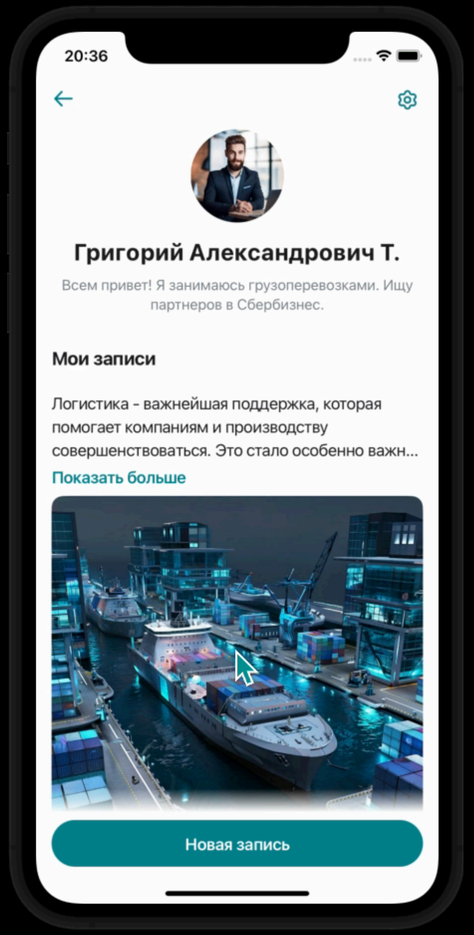
- ✳ Позволяет делать много интересных анимаций;
- ✳ Различные эффекты для текста: блюр, обрезка, замена, фон, ободка, рисование;
- ✳ Изменение фрейма и отступов в тексте (например, резерв места для дополнительной строки); Учитывать что изменение параметров верстки в методе draw применится только в следующем кадре;
- ✳ Использование как средство для отладки: через рендеринг, мы можем заглянуть в финальные атрибуты текста;
- ✳ Можем получать размеры области текста, помеченной TextAttribute;
- – Атрибуты AttributedString недоступны ;
- – Не может использоваться в фоновом режиме, в том числе, для прекалькуляции размеров текста.

ScrollView с прозрачным отступом для жестов.



iOS16+

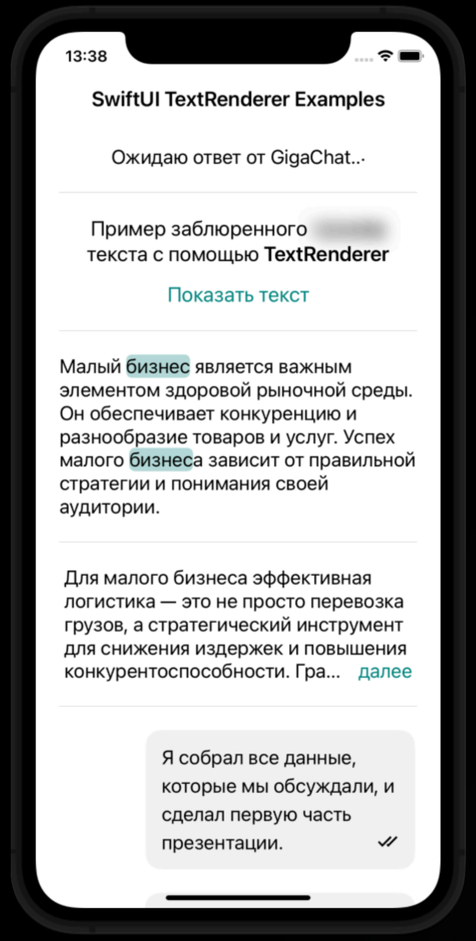
Анимация профиля.



В общей ветке пример iOS16+

В отдельной ветке для iOS18

Примеры с TextRenderer.



iOS18+

Примеры кода на GitHub

Благообразова Татьяна Александровна
Руководитель направления
Сбер «Цифровой Корпоративный Банк»