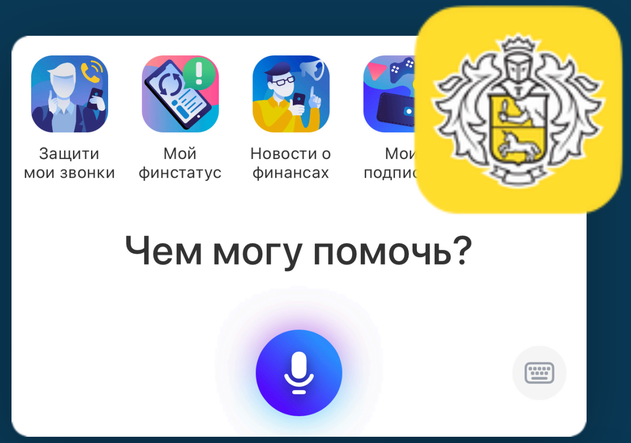
OZON tech

Выжимаем максимум из SwiftUI Preview

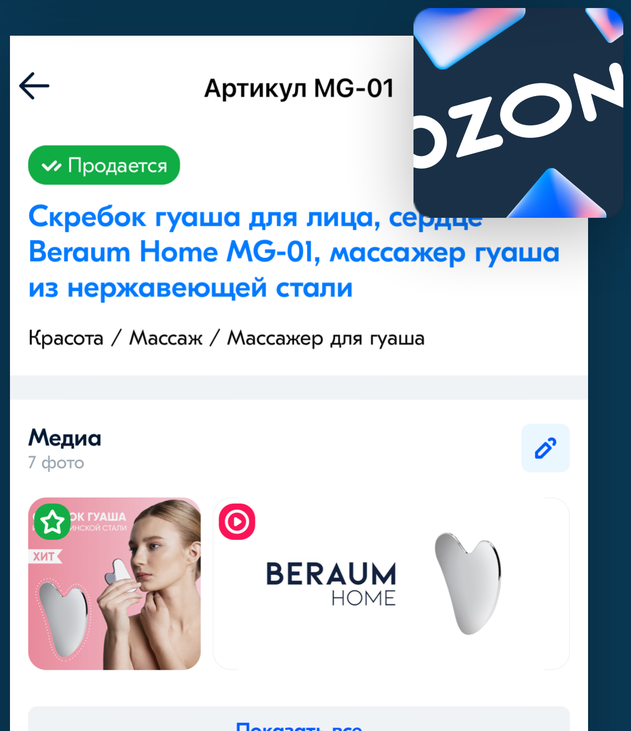
Максим Гришутин
Руководитель группы разработки мобильных приложений iOS
Ozon Tech

Немного обо мне

- Успел поработать над:
  - **White label** приложением для банков (ВТБ, ОТП, Абсолют и еще 100+)
  - UI/UX голосового помощника “**Олег**” в Тинькофф



- Я отвечаю за iOS приложение для селлеров в **Ozon**
- Преподаватель iOS курсов по **SwiftUI** в Ozon School Route 256
- Стараюсь внедрять и продвигать новые технологии 🚀



О чем мы поговорим

- Что мы будем 🍏 выжимать из **Preview**
- Про **SwiftUI Preview**
- Реализуем основу генерации
- Генерируем:
  - Playbook app
  - Performance анализ
  - Snapshot тесты
  - Accessibility тесты
- Project 🔥 **Prefire**
- Подведем итоги

Что мы будем  выжимать из Previews

OZON tech

Что будет, если использовать свой 🧠 Preview на 100%?

А не как сейчас, на 20%

Выжимаем максимум

Как использовать Preview?

Как обычно используют **Preview**:

- Просмотр верстки
- Взаимодействие с view

Как нужно использовать **Preview**:

- Генерация **Playbook app**
- Генерация **Performance** анализа
- Генерация **Snapshot** тестов
- Генерация **Accessibility** тестов

А зачем вообще выжимать Preview?

Какие проблемы решаем

SwiftUI Previews имеют уже готовые view

Куда еще нужны готовые view?

- **Playbook app**
- **Performance** анализ
- **Snapshot** тесты
- **Accessibility** тесты

Так какая проблема решается?

Одна из главных проблем

OZON tech

Ручное написание кода

Про SwiftUI Preview

Про SwiftUI Preview

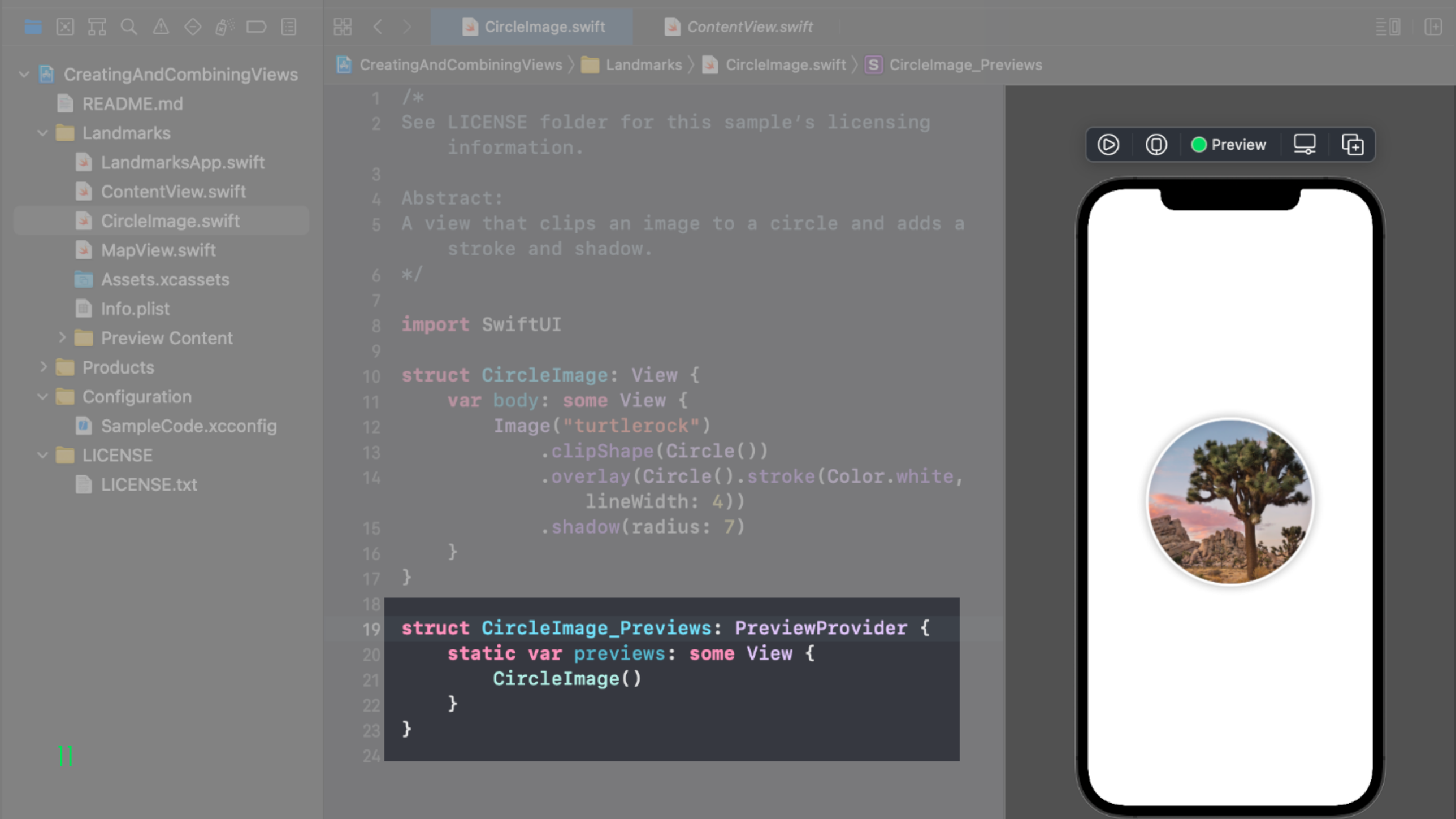
OZON{ech

Что это такое?

Нативная система для простого просмотра/взаимодействия с **View**

```
struct CircleImage: View {
    var body: some View {
        Image("nature")
            .clipShape(Circle())
            .overlay(Circle().stroke(Color.white, lineWidth: 4))
            .shadow(radius: 7)
    }
}
```

```
struct CircleImage_Previews: PreviewProvider {
    static var previews: some View {
        CircleImage()
            .previewLayout(.sizeThatFits)
    }
}
```


CircleImage.swift
ContentView.swift
CreatingAndCombiningViews
Landmarks
CircleImage.swift
CircleImage_Previews
CreatingAndCombiningViews
README.md
Landmarks
LandmarksApp.swift
ContentView.swift
CircleImage.swift
MapView.swift
Assets.xcassets
Info.plist
Preview Content
Products
Configuration
SampleCode.xcconfig
LICENSE
LICENSE.txt
/*
See LICENSE folder for this sample's licensing information.
Abstract:
A view that clips an image to a circle and adds a stroke and shadow.
*/
import SwiftUI
struct CircleImage: View {
var body: some View {
Image("turtlerock")
.clipShape(Circle())
.overlay(Circle().stroke(Color.white, lineWidth: 4))
.shadow(radius: 7)
}
}
struct CircleImage_Previews: PreviewProvider {
static var previews: some View {
CircleImage()
}
}
Preview

Преимущества SwiftUI Preview

OZON{ech

- Нативный Apple подход
- Простой и быстрый просмотр любой View
- Статичные view
- Гибкая настройка устройства для Preview

Deep dive into SwiftUI Preview

Это могло быть названием темы, но ...

Deep dive into SwiftUI Preview

OZON tech

Протокол **PreviewProvider**

```
public protocol PreviewProvider : _PreviewProvider {
    @ViewBuilder static var previews: Self.Previews { get }
}
```

```
struct CircleImage_Previews: PreviewProvider {
    static var previews: some View {
        CircleImage()
    }
}
```

Deep dive into SwiftUI Preview

OZON tech

Протокол **_PreviewProvider**

```
protocol _PreviewProvider {
    public static var platform: SwiftUI.PreviewPlatform? { get }
    public static var _previews: Any { get }
    public static var _platform: SwiftUI.PreviewPlatform? { get }
    public static var _allPreviews: [SwiftUI._Preview] { get }
}
```

Deep dive into SwiftUI Preview

ozon{tech

Протокол **_PreviewProvider**

```
protocol _PreviewProvider {
    public static var platform: SwiftUI.PreviewPlatform? { get }
    public static var _previews: Any { get }
    public static var _platform: SwiftUI.PreviewPlatform? { get }
    public static var _allPreviews: [SwiftUI._Preview] { get }
}
```

Deep dive into SwiftUI Preview

OZON{ech

Протокол **_PreviewProvider**

```
protocol _PreviewProvider {
    public static var _allPreviews: [SwiftUI._Preview] { get }
}
```

Deep dive into SwiftUI Preview

OZON{ech

Протокол **_PreviewProvider**

```
protocol _PreviewProvider {
    public static var _allPreviews: [SwiftUI._Preview] { get }
}
```

```
struct CircleImage_Previews: PreviewProvider {
    static var previews: some View {
        CircleImage()
        CircleImage()
    }
}
```

Deep dive into SwiftUI Preview

ozon{tech

Протокол **_PreviewProvider**

```
protocol _PreviewProvider {
    public static var _allPreviews: [SwiftUI._Preview] { get }
}
```

```
struct CircleImage_Previews: PreviewProvider {
    static var previews: some View {
        CircleImage()
        CircleImage()
    }
}
```

Deep dive into SwiftUI Preview

ozon tech

Весь код я брал из публичного интерфейса iOS SDK

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
public protocol _PreviewProviderLocator {
   static var previewProviders: [_PreviewProvider.Type] {
     get
   }
}

@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
public protocol _PreviewProvider {
  static var _previews: Any { get }
  static var _platform: SwiftUI.PreviewPlatform? { get }
}
```

Используем максимум от SwiftUI Preview

Используем максимум от SwiftUI Preview

OZON tech

Будем реализовывать

- Playbook (Demo) App
- Простой performance анализ
- Snapshot тесты
- Accessibility тесты

Сначала реализуем Core функциональность

Реализуем Core функциональность

OZON tech

Нам нужны определенные View из **Preview**

Реализуем Core функциональность

Нам нужны только определенные View

```
struct CircleImage: View {
    var body: some View {
        Image("nature")
            .clipShape(Circle())
            .overlay(Circle().stroke(Color.white, lineWidth: 4))
            .shadow(radius: 7)
    }
}

struct CircleImage_Previews: PreviewProvider {
    static var previews: some View {
        CircleImage()
    }
}
```

Реализуем Core функциональность

Добавим PrefireProvider

```
public protocol PrefireProvider: PreviewProvider {}

struct CircleImage: View {
    var body: some View {
        Image("nature")
            .clipShape(Circle())
            .overlay(Circle().stroke(Color.white, lineWidth: 4))
            .shadow(radius: 7)
    }
}

struct CircleImage_Previews: PreviewProvider {
    static var previews: some View {
        CircleImage()
    }
}
```

Реализуем Core функциональность

Добавим PrefireProvider

```
public protocol PrefireProvider: PreviewProvider {}

struct CircleImage: View {
    var body: some View {
        Image("nature")
            .clipShape(Circle())
            .overlay(Circle().stroke(Color.white, lineWidth: 4))
            .shadow(radius: 7)
    }
}

struct CircleImage_Previews: PreviewProvider, PrefireProvider {
    static var previews: some View {
        CircleImage()
    }
}
```

Отдыхаем 

Нам нужно хранить информацию о Preview

ozon{ech

Сделаем PreviewModel

```
public struct PreviewModel {
    /// Наша static view от Preview
    public let content: () -> AnyView

    /// Имя preview
    public let name: String

    /// Состояние preview (.default, loading и тд)
    public let state: String

    /// Тип preview (.screen, component)
    public var type: ViewType

    /// Устройство для отображения preview
    public var device: PreviewDevice?

    /// Наименование нашего Flow
    public var story: String?

    /// Время рендеринга
    public var renderTime: String?
}
```

Нам нужно хранить информацию о Preview

ozon{ech

Сделаем PreviewModel

```
public struct PreviewModel {
    /// Наша static view от Preview
    public let content: () -> AnyView

    /// Имя preview
    public let name: String

    /// Состояние preview (.default, loading и тд)
    public let state: String

    /// Тип preview (.screen, component)
    public var type: ViewType

    /// Устройство для отображения preview
    public var device: PreviewDevice?

    /// Наименование нашего Flow
    public var story: String?

    /// Время рендеринга
    public var renderTime: String?
}
```

Нам нужно хранить информацию о Preview

ozon{ech

Сделаем PreviewModel

```
public struct PreviewModel {
    /// Наша static view от Preview
    public let content: () -> AnyView

    /// Имя preview
    public let name: String

    /// Состояние preview (.default, loading и тд)
    public let state: String

    /// Тип preview (.screen, component)
    public var type: ViewType

    /// Устройство для отображения preview
    public var device: PreviewDevice?

    /// Наименование нашего Flow
    public var story: String?

    /// Время рендеринга
    public var renderTime: String?
}
```

Нам нужно хранить информацию о Preview

ozon{ech

Сделаем PreviewModel

```
public struct PreviewModel {
    /// Наша static view от Preview
    public let content: () -> AnyView

    /// Имя preview
    public let name: String

    /// Состояние preview (.default, loading и тд)
    public let state: String

    /// Тип preview (.screen, component)
    public var type: ViewType

    /// Устройство для отображения preview
    public var device: PreviewDevice?

    /// Наименование нашего Flow
    public var story: String?

    /// Время рендеринга
    public var renderTime: String?
}
```

Нам нужно хранить информацию о Preview

ozon{ech

Сделаем PreviewModel

```
public struct PreviewModel {
    /// Наша static view от Preview
    public let content: () -> AnyView

    /// Имя preview
    public let name: String

    /// Состояние preview (.default, loading и тд)
    public let state: String

    /// Тип preview (.screen, component)
    public var type: ViewType

    /// Устройство для отображения preview
    public var device: PreviewDevice?

    /// Наименование нашего Flow
    public var story: String?

    /// Время рендеринга
    public var renderTime: String?
}
```

Нам нужно хранить информацию о Preview

ozon{ech

Сделаем PreviewModel

```
public struct PreviewModel {
    /// Наша static view от Preview
    public let content: () -> AnyView

    /// Имя preview
    public let name: String

    /// Состояние preview (.default, loading и тд)
    public let state: String

    /// Тип preview (.screen, component)
    public var type: ViewType

    /// Устройство для отображения preview
    public var device: PreviewDevice?

    /// Наименование нашего Flow
    public var story: String?

    /// Время рендеринга
    public var renderTime: String?
}
```

Нам нужно хранить информацию о Preview

ozon{ech

Сделаем PreviewModel

```
public struct PreviewModel {
    /// Наша static view от Preview
    public let content: () -> AnyView

    /// Имя preview
    public let name: String

    /// Состояние preview (.default, loading и тд)
    public let state: String

    /// Тип preview (.screen, component)
    public var type: ViewType

    /// Устройство для отображения preview
    public var device: PreviewDevice?

    /// Наименование нашего Flow
    public var story: String?

    /// Время рендеринга
    public var renderTime: String?
}
```

Нам нужно хранить информацию о Preview

ozon{ech

Сделаем PreviewModel

```
public struct PreviewModel {
    /// Наша static view от Preview
    public let content: () -> AnyView

    /// Имя preview
    public let name: String

    /// Состояние preview (.default, loading и тд)
    public let state: String

    /// Тип preview (.screen, component)
    public var type: ViewType

    /// Устройство для отображения preview
    public var device: PreviewDevice?

    /// Наименование нашего Flow
    public var story: String?

    /// Время рендеринга
    public var renderTime: String?
}
```

Нам нужно хранить информацию о Preview

ozon{ech

Сделаем PreviewModel

```
public struct PreviewModel {
    /// Наша static view от Preview
    public let content: () -> AnyView

    /// Имя preview
    public let name: String

    /// Состояние preview (.default, loading и тд)
    public let state: String

    /// Тип preview (.screen, component)
    public var type: ViewType

    /// Устройство для отображения preview
    public var device: PreviewDevice?

    /// Наименование нашего Flow
    public var story: String?

    /// Время рендеринга
    public var renderTime: String?
}
```

Реализуем Core функциональность

- Можем отличать нужные нам View
- Есть универсальная модель

Как можно пользоваться

ozon tech

```
struct CircleImage: View {
    var body: some View {
        Image("nature")
            .clipShape(Circle())
            .overlay(Circle().stroke(Color.white, lineWidth: 4))
            .shadow(radius: 7)
    }
}
```

```
struct CircleImage_Previews: PreviewProvider {
    static var previews: some View {
        CircleImage()
            .previewLayout(.sizeThatFits)
    }
}
```

Как можно пользоваться

ozon{tech

```
struct CircleImage_Previews: PreviewProvider, PrefireProvider {
    static var previews: some View {
        CircleImage()



        CircleImage(name: "loading")
            .previewLayout(.sizeThatFits)
            .previewState(.loading)
    }
}
```

Как можно пользоваться

ozon tech

```
struct CircleImage_Previews: PreviewProvider, PrefireProvider {
    static var previews: some View {
        CircleImage()



        CircleImage(name: "loading")
            .previewLayout(.sizeThatFits)
            .previewState(.loading)
    }
}
```

Как можно пользоваться

ozon tech

```
struct CircleImage_Previews: PreviewProvider, PrefireProvider {
    static var previews: some View {
        CircleImage()



        CircleImage(name: "loading")
            .previewLayout(.sizeThatFits)
            .previewState(.loading)
    }
}
```

Как можно пользоваться

ozon tech

```
struct CircleImage_Previews: PreviewProvider, PrefireProvider {
    static var previews: some View {
        CircleImage()



        CircleImage(name: "loading")
            .previewLayout(.sizeThatFits)
            .previewState(.loading)
    }
}
```

Как можно пользоваться

```
struct CircleImage_Previews: PreviewProvider, PrefireProvider {
    static var previews: some View {
        CircleImage()



        CircleImage(name: "loading")
            .previewLayout(.sizeThatFits)
            .previewUserStory(.loading)
    }
}
```

Используем кодогенерацию

Sourcery

Используем кодогенерацию

Делаем .stencil шаблон

```
enum PreviewModels {
   static var models: [PreviewModel] = {
       var views: [PreviewModel] = []

       {% for type in types.types where type.implements.PrefireProvider %}
       for state in {{ type.name }}.State.allCases {
          views.append(
             PreviewModel(
                 content: { AnyView({{ type.name }}.previews) },
                 name: "{{ type.name|replace:"_Previews", "" }}",
                 type: {{ type.name }}._allPreviews.first.layout == .sizeThatFits ? .component : .screen,
                 device: {{ type.name }}._allPreviews.first?.device,
                 state: String(String(reflecting: state).split(separator: ".").last)
             )
          )
       }
       {% endfor %}

       return views
   }()
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
enum PreviewModels {
   static var models: [PreviewModel] = {
       var views: [PreviewModel] = []

       {% for type in types.types where type.implements.PrefireProvider %}
       for state in {{ type.name }}.State.allCases {
          views.append(
             PreviewModel(
                 content: { AnyView({{ type.name }}.previews) },
                 name: "{{ type.name|replace:"_Previews", "" }}",
                 type: {{ type.name }}._allPreviews.first.layout == .sizeThatFits ? .component : .screen,
                 device: {{ type.name }}._allPreviews.first?.device,
                 state: String(String(reflecting: state).split(separator: ".").last)
             )
          )
       }
       {% endfor %}

       return views
   }()
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
enum PreviewModels {
   static var models: [PreviewModel] = {
       var views: [PreviewModel] = []

       {% for type in types.types where type.implements.PrefireProvider %}
       for state in {{ type.name }}.State.allCases {
          views.append(
             PreviewModel(
                 content: { AnyView({{ type.name }}.previews) },
                 name: "{{ type.name|replace:"_Previews", "" }}",
                 type: {{ type.name }}._allPreviews.first.layout == .sizeThatFits ? .component : .screen,
                 device: {{ type.name }}._allPreviews.first?.device,
                 state: String(String(reflecting: state).split(separator: ".").last)
             )
          )
       }
       {% endfor %}

       return views
   }()
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
enum PreviewModels {
   static var models: [PreviewModel] = {
       var views: [PreviewModel] = []

       {% for type in types.types where type.implements.PrefireProvider %}
       for state in {{ type.name }}.State.allCases {
          views.append(
             PreviewModel(
                 content: { AnyView({{ type.name }}.previews) },
                 name: "{{ type.name|replace:"_Previews", "" }}",
                 type: {{ type.name }}._allPreviews.first.layout == .sizeThatFits ? .component : .screen,
                 device: {{ type.name }}._allPreviews.first?.device,
                 state: String(String(reflecting: state).split(separator: ".").last)
             )
          )
       }
       {% endfor %}

       return views
   }()
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
enum PreviewModels {
   static var models: [PreviewModel] = {
       var views: [PreviewModel] = []

       {% for type in types.types where type.implements.PrefireProvider %}
       for state in {{ type.name }}.State.allCases {
          views.append(
             PreviewModel(
                 content: { AnyView({{ type.name }}.previews) },
                 name: "{{ type.name|replace:"_Previews", "" }}",
                 type: {{ type.name }}._allPreviews.first.layout == .sizeThatFits ? .component : .screen,
                 device: {{ type.name }}._allPreviews.first?.device,
                 state: String(String(reflecting: state).split(separator: ".").last)
             )
          )
       }
       {% endfor %}

       return views
   }()
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
enum PreviewModels {
   static var models: [PreviewModel] = {
       var views: [PreviewModel] = []

       {% for type in types.types where type.implements.PrefireProvider %}
       for state in {{ type.name }}.State.allCases {
          views.append(
             PreviewModel(
                 content: { AnyView({{ type.name }}.previews) },
                 name: "{{ type.name|replace:"_Previews", "" }}",
                 type: {{ type.name }}._allPreviews.first.layout == .sizeThatFits ? .component : .screen,
                 device: {{ type.name }}._allPreviews.first?.device,
                 state: String(String(reflecting: state).split(separator: ".").last)
             )
          )
       }
       {% endfor %}

       return views
   }()
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
enum PreviewModels {
   static var models: [PreviewModel] = {
       var views: [PreviewModel] = []

       {% for type in types.types where type.implements.PrefireProvider %}
       for state in {{ type.name }}.State.allCases {
          views.append(
             PreviewModel(
                 content: { AnyView({{ type.name }}.previews) },
                 name: "{{ type.name|replace:"_Previews", "" }}",
                 type: {{ type.name }}._allPreviews.first.layout == .sizeThatFits ? .component : .screen,
                 device: {{ type.name }}._allPreviews.first?.device,
                 state: String(String(reflecting: state).split(separator: ".").last)
             )
          )
       }
       {% endfor %}

       return views
   }()
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
enum PreviewModels {
   static var models: [PreviewModel] = {
       var views: [PreviewModel] = []

       {% for type in types.types where type.implements.PrefireProvider %}
       for state in {{ type.name }}.State.allCases {
          views.append(
             PreviewModel(
                 content: { AnyView({{ type.name }}.previews) },
                 name: "{{ type.name|replace:"_Previews", "" }}",
                 type: {{ type.name }}._allPreviews.first.layout == .sizeThatFits ? .component : .screen,
                 device: {{ type.name }}._allPreviews.first?.device,
                 state: String(String(reflecting: state).split(separator: ".").last)
             )
          )
       }
       {% endfor %}

       return views
   }()
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
enum PreviewModels {
   static var models: [PreviewModel] = {
       var views: [PreviewModel] = []

       {% for type in types.types where type.implements.PrefireProvider %}
       for state in {{ type.name }}.State.allCases {
          views.append(
             PreviewModel(
                 content: { AnyView({{ type.name }}.previews) },
                 name: "{{ type.name|replace:"_Previews", "" }}",
                 type: {{ type.name }}._allPreviews.first.layout == .sizeThatFits ? .component : .screen,
                 device: {{ type.name }}._allPreviews.first?.device,
                 state: String(String(reflecting: state).split(separator: ".").last)
             )
          )
       }
       {% endfor %}

       return views
   }()
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
enum PreviewModels {
   static var models: [PreviewModel] = {
       var views: [PreviewModel] = []

       {% for type in types.types where type.implements.PrefireProvider %}
       for state in {{ type.name }}.State.allCases {
          views.append(
             PreviewModel(
                 content: { AnyView({{ type.name }}.previews) },
                 name: "{{ type.name|replace:"_Previews", "" }}",
                 type: {{ type.name }}._allPreviews.first.layout == .sizeThatFits ? .component : .screen,
                 device: {{ type.name }}._allPreviews.first?.device,
                 state: String(String(reflecting: state).split(separator: ".").last)
             )
          )
       }
       {% endfor %}

       return views
   }()
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
enum PreviewModels {
   static var models: [PreviewModel] = {
       var views: [PreviewModel] = []

       {% for type in types.types where type.implements.PrefireProvider %}
       for state in {{ type.name }}.State.allCases {
          views.append(
             PreviewModel(
                 content: { AnyView({{ type.name }}.previews) },
                 name: "{{ type.name|replace:"_Previews", "" }}",
                 type: {{ type.name }}._allPreviews.first.layout == .sizeThatFits ? .component : .screen,
                 device: {{ type.name }}._allPreviews.first?.device,
                 state: String(String(reflecting: state).split(separator: ".").last)
             )
          )
       }
       {% endfor %}

       return views
   }()
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
enum PreviewModels {
   static var models: [PreviewModel] = {
       var views: [PreviewModel] = []

       {% for type in types.types where type.implements.PrefireProvider %}
       for state in {{ type.name }}.State.allCases {
          views.append(
             PreviewModel(
                 content: { AnyView({{ type.name }}.previews) },
                 name: "{{ type.name|replace:"_Previews", "" }}",
                 type: {{ type.name }}._allPreviews.first.layout == .sizeThatFits ? .component : .screen,
                 device: {{ type.name }}._allPreviews.first?.device,
                 state: String(String(reflecting: state).split(separator: ".").last)
             )
          )
       }
       {% endfor %}

       return views
   }()
}
```

Используем кодогенерацию

Запускаем кодогенерацию **Sourcery** в **Build Phase**

```
./bin/sourcery --sources <sources path> --templates <templates path> --output <output path>
```

Теперь у нас есть массив моделей Previews

```
public enum PreviewModels {
    public static var models: [PreviewModel] = {
        var views: [PreviewModel] = []

        for state in CircleImage_Previews.State.allCases {
            views.append(
                PreviewModel(
                    content: {
                        CircleImage_Previews.state = state
                        return AnyView(CircleImage_Previews.previews)
                    },
                    name: "CircleImage",
                    type: CircleImage_Previews._allPreviews.first?.layout == .sizeThatFits ? .component : .screen,
                    device: CircleImage_Previews._allPreviews.first?.device,
                    state: String(String(reflecting: state).split(separator: ".").last!)
                )
            )
        }

        return views.sorted(by: { $0.name > $1.name || $0.story ?? "" > $1.story ?? "" })
    }()
}
```

Core слой готов

Давайте выжмем максимум из SwiftUI Preview 🚀

Используем максимум от SwiftUI Preview

ozon tech

Будем реализовывать

- Playbook (Demo) App
- Простой performance анализ
- Snapshot тесты
- Accessibility тесты

Используем максимум от SwiftUI Preview

ozon tech

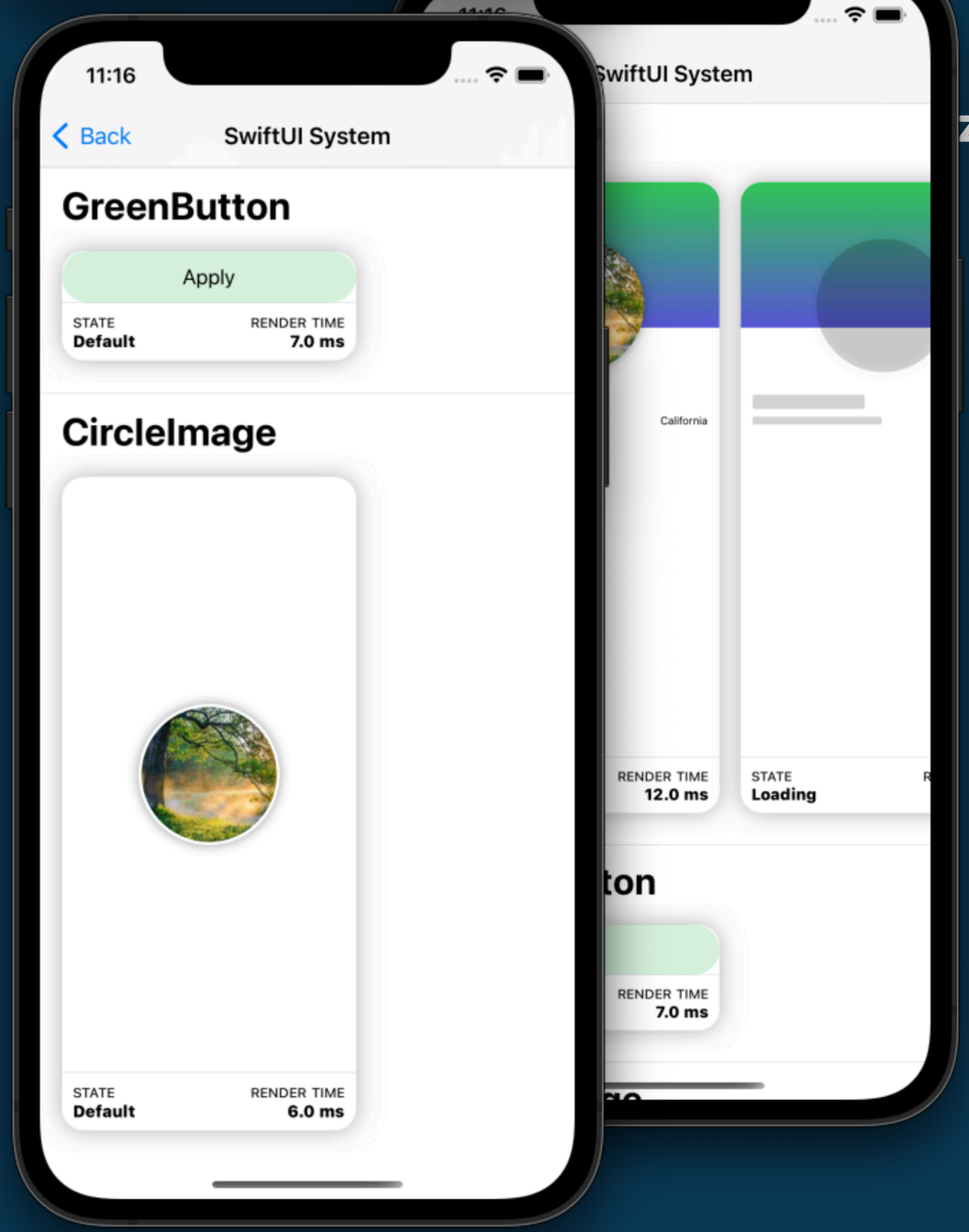
Будем реализовывать

- Playbook (Demo) App
- Простой performance анализ
- Snapshot тесты
- Accessibility тесты

Playbook App

Что такое Playbook App

Демо приложение, содержащее все компоненты и экраны из приложения



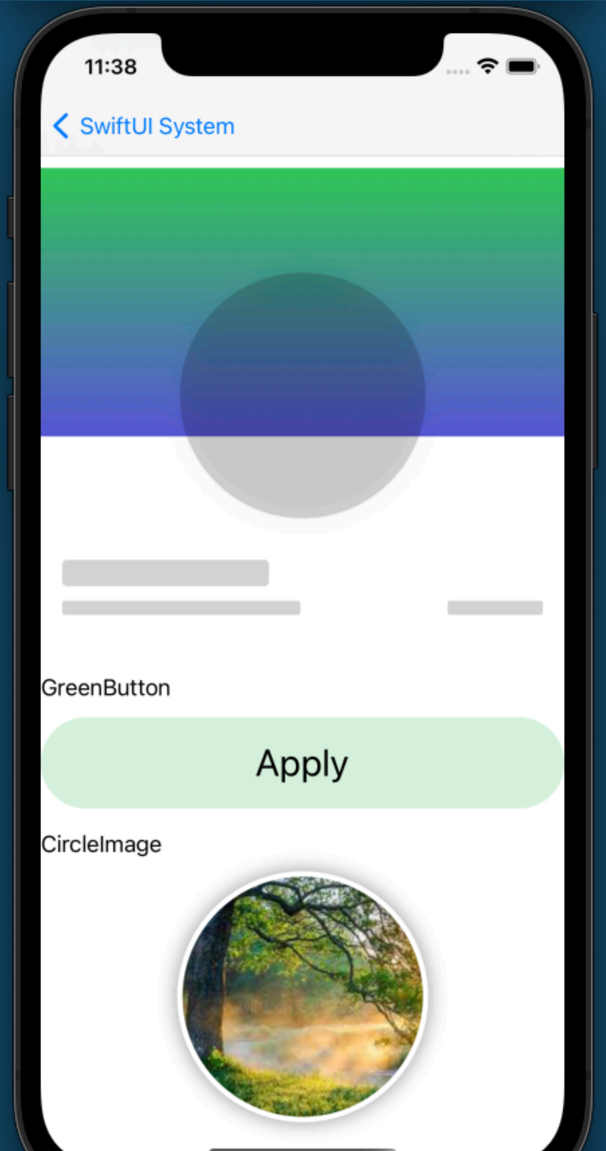
Зачем нужно Playbook App

- Знакомиться с приложением новичкам
- Проводить дизайн ревью
- Решать проблему с дублированием компонентов
- Отдельно тестировать UI

Playbook view

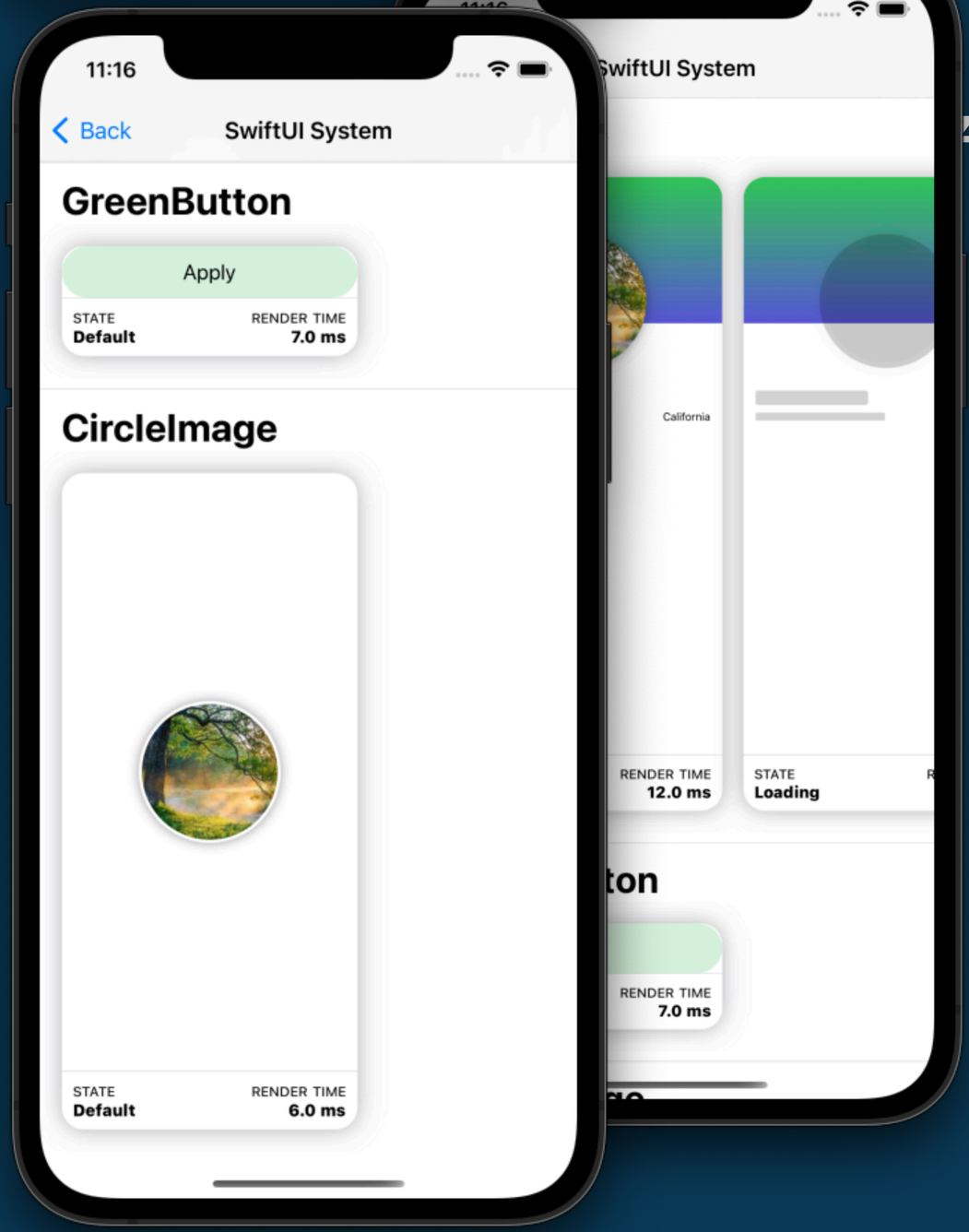
Простая реализация

```
public struct PlaybookView: View {
    @State private var viewModels: [PreviewModel]

    public var body: some View {
        ScrollView(.horizontal, showsIndicators: false) {
            LazyHStack(alignment: .top, spacing: 16) {
                ForEach($viewModels) { $viewModel in
                    viewModel.content()
                }
            }
            .padding(16)
        }
    }
}
```



Playbook App

Немного доработав, можем получить такой вид



Используем максимум от SwiftUI Preview

ozon tech

Будем реализовывать

- Playbook (Demo) App
- Простой performance анализ
- Snapshot тесты
- Accessibility тесты

Используем максимум от SwiftUI Preview

ozon tech

Будем реализовывать

- Playbook (Demo) App
- Простой performance анализ
- Snapshot тесты
- Accessibility тесты

Performance

Performance

OZON tech

Можем замерить время загрузки View

Performance

Модификатор замера времени загрузки View

```
struct LoadingTimeModifier: ViewModifier {
    @State private var createDate: Date? = Date()
    var completion: (String) -> Void

    func body(content: Content) -> some View {
        content
            .onAppear {
                guard let createDate = createDate else { return }

                let diffTime = Date().timeIntervalSince(createDate) * 1000
                completion(String(diffTime.truncatingRemainder(dividingBy: 1000).rounded()) + " ms")

                self.createDate = nil
            }
    }
}
```

Performance

Модификатор замера времени загрузки View

```
struct LoadingTimeModifier: ViewModifier {
    @State private var createDate: Date? = Date()
    var completion: (String) -> Void

    func body(content: Content) -> some View {
        content
            .onAppear {
                guard let createDate = createDate else { return }

                let diffTime = Date().timeIntervalSince(createDate) * 1000
                completion(String(diffTime.truncatingRemainder(dividingBy: 1000).rounded()) + " ms")

                self.createDate = nil
            }
    }
}
```

Performance

Модификатор замера времени загрузки View

```
struct LoadingTimeModifier: ViewModifier {
    @State private var createDate: Date? = Date()
    var completion: (String) -> Void

    func body(content: Content) -> some View {
        content
            .onAppear {
                guard let createDate = createDate else { return }

                let diffTime = Date().timeIntervalSince(createDate) * 1000
                completion(String(diffTime.truncatingRemainder(dividingBy: 1000).rounded()) + " ms")

                self.createDate = nil
            }
    }
}
```

Performance

Модификатор замера времени загрузки View

```
struct LoadingTimeModifier: ViewModifier {
    @State private var createDate: Date? = Date()
    var completion: (String) -> Void

    func body(content: Content) -> some View {
        content
            .onAppear {
                guard let createDate = createDate else { return }

                let diffTime = Date().timeIntervalSince(createDate) * 1000
                completion(String(diffTime.truncatingRemainder(dividingBy: 1000).rounded()) + " ms")

                self.createDate = nil
            }
    }
}
```

Performance

Модификатор замера времени загрузки View

```
struct LoadingTimeModifier: ViewModifier {
    @State private var createDate: Date? = Date()
    var completion: (String) -> Void

    func body(content: Content) -> some View {
        content
            .onAppear {
                guard let createDate = createDate else { return }

                let diffTime = Date().timeIntervalSince(createDate) * 1000
                completion(String(diffTime.truncatingRemainder(dividingBy: 1000).rounded()) + " ms")

                self.createDate = nil
            }
    }
}
```

Performance

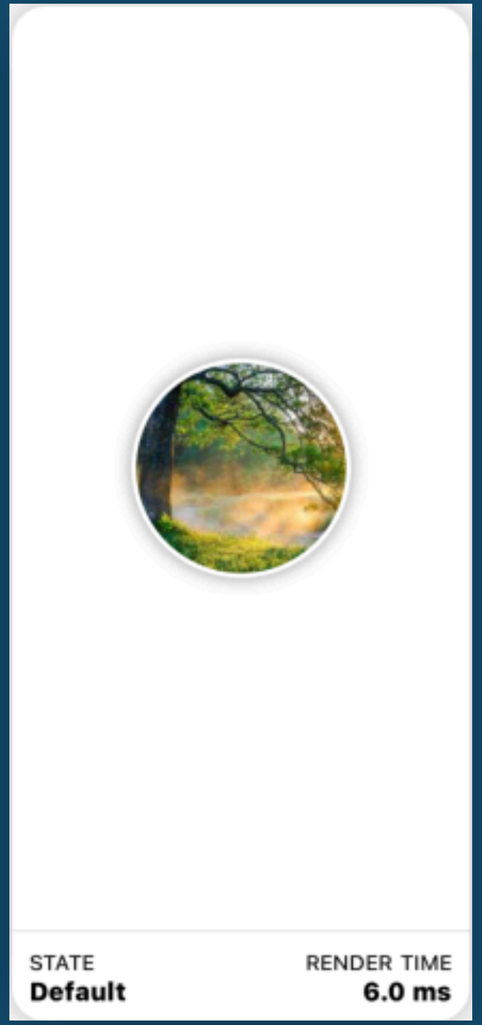
Модификатор замера времени загрузки View

```
struct LoadingTimeModifier: ViewModifier {
    @State private var createDate: Date? = Date()
    var completion: (String) -> Void

    func body(content: Content) -> some View {
        content
            .onAppear {
                guard let createDate = createDate else { return }

                let diffTime = Date().timeIntervalSince(createDate) * 1000
                completion(String(diffTime.truncatingRemainder(dividingBy: 1000).rounded()) + " ms")

                self.createDate = nil
            }
    }
}
```

Performance

Использование модификатора

```
viewModel.content()
    .modifier(LoadingTimeModifier { loadingTime in
        viewModel.renderTime = loadingTime
    })
```



Используем максимум от SwiftUI Preview

ozon tech

Будем реализовывать

- Playbook (Demo) App
- Простой performance анализ
- Snapshot тесты
- Accessibility тесты

Используем максимум от SwiftUI Preview

OZON tech

Будем реализовывать

- Playbook (Demo) App
- Простой performance анализ
- Snapshot тесты
- Accessibility тесты

Snapshot тесты

Snapshot тесты

OZON tech

Возьмем готовую библиотеку

Бибилиотека от *pointfreeco*

SnapshotTesnig

POINT·FREE

Snapshot тесты

Как выглядит Snapshot тест

```
class OnboardingViewTests: XCTestCase {
    func testOnboardingView() {
        // Given
        let view = OnboardingView()

        // Then
        assertSnapshot(matching: view, as: .image())
    }
}
```

Snapshot тесты

Как выглядит Snapshot тест

```
class OnboardingViewTests: XCTestCase {
    func testOnboardingView() {
        // Given
        let view = OnboardingView()

        // Then
        assertSnapshot(matching: view, as: .image())
    }
}
```

Snapshot тесты

Как выглядит Snapshot тест

```
class OnboardingViewTests: XCTestCase {
    func testOnboardingView() {
        // Given
        let view = OnboardingView()

        // Then
        assertSnapshot(matching: view, as: .image())
    }
}
```

Snapshot тесты

Более подробно о Snapshot тестировании

Snapshot тесты

ozon{ech

Что нам нужно для генерации тестов?

- Stencil шаблон
- SwiftUI Preview

Используем кодогенерацию

Делаем .stencil шаблон

```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: view,
                as: isScreen ? .image(layout: .device(config: device)) : .image(layout: .sizeThatFits)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: view,
                as: isScreen ? .image(layout: .device(config: device)) : .image(layout: .sizeThatFits)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: view,
                as: isScreen ? .image(layout: .device(config: device)) : .image(layout: .sizeThatFits)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: view,
                as: isScreen ? .image(layout: .device(config: device)) : .image(layout: .sizeThatFits)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: view,
                as: isScreen ? .image(layout: .device(config: device)) : .image(layout: .sizeThatFits)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: view,
                as: isScreen ? .image(layout: .device(config: device)) : .image(layout: .sizeThatFits)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: view,
                as: isScreen ? .image(layout: .device(config: device)) : .image(layout: .sizeThatFits)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: view,
                as: isScreen ? .image(layout: .device(config: device)) : .image(layout: .sizeThatFits)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: view,
                as: isScreen ? .image(layout: .device(config: device)) : .image(layout: .sizeThatFits)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: view,
                as: isScreen ? .image(layout: .device(config: device)) : .image(layout: .sizeThatFits)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: view,
                as: isScreen ? .image(layout: .device(config: device)) : .image(layout: .sizeThatFits)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: view,
                as: isScreen ? .image(layout: .device(config: device)) : .image(layout: .sizeThatFits)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Делаем .stencil шаблон

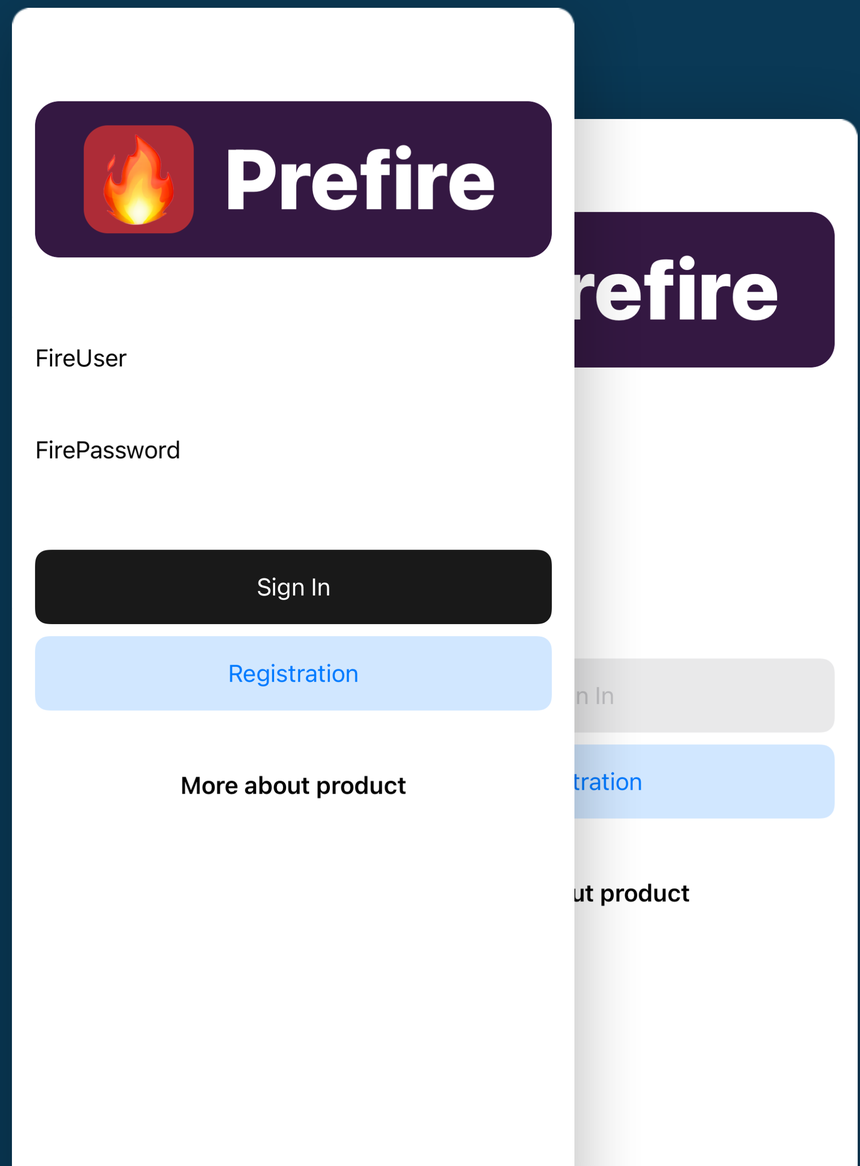
```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: view,
                as: isScreen ? .image(layout: .device(config: device)) : .image(layout: .sizeThatFits)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Результат

Запускаем и получаем результат 🚀

```
func test_prefireView_Preview() {
    for view in PrefireView_Preview._allPreviews {
        // Given
        let preview = PrefireView_Preview._allPreviews.first
        let isScreen = preview?.layout == .device
        let device = preview?.device?.snapshotDevice() ?? deviceConfig

        // When
        var view = view.content
        view = isScreen ? view : AnyView(view.frame(width: device.size?.width))

        // Then
        assertSnapshot(
            matching: view,
            as: isScreen ? .image(layout: .device(device)) : .image(.sizeThatFits)
        )
    }
}
```



Используем максимум от SwiftUI Preview

ozon tech

Будем реализовывать

- Playbook (Demo) App
- Простой performance анализ
- Snapshot тесты
- Accessibility тесты

Используем максимум от SwiftUI Preview

ozon tech

Будем реализовывать

- Playbook (Demo) App
- Простой performance анализ
- Snapshot тесты
- Accessibility тесты

Accessibility тесты

Accessibility тесты

Возьмем готовую библиотеку

Библиотека на основе предыдущей *Snapshot Testing*

AccessibilitySnapshot

Accessibility тесты

OZON tech

Все как и с Snapshot тестами

- Используем SwiftUI Preview
- Делаем stencil шаблон
  - Возьмем шаблон Snapshot тестов

Используем кодогенерацию

Делаем .stencil шаблон

```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: view,
                as: isScreen ? .image(layout: .device(config: device)) : .image(layout: .sizeThatFits)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Делаем .stencil шаблон

```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: view,
                as: isScreen ? .image(layout: .device(config: device)) : .image(layout: .sizeThatFits)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Делаем .stencil шаблон

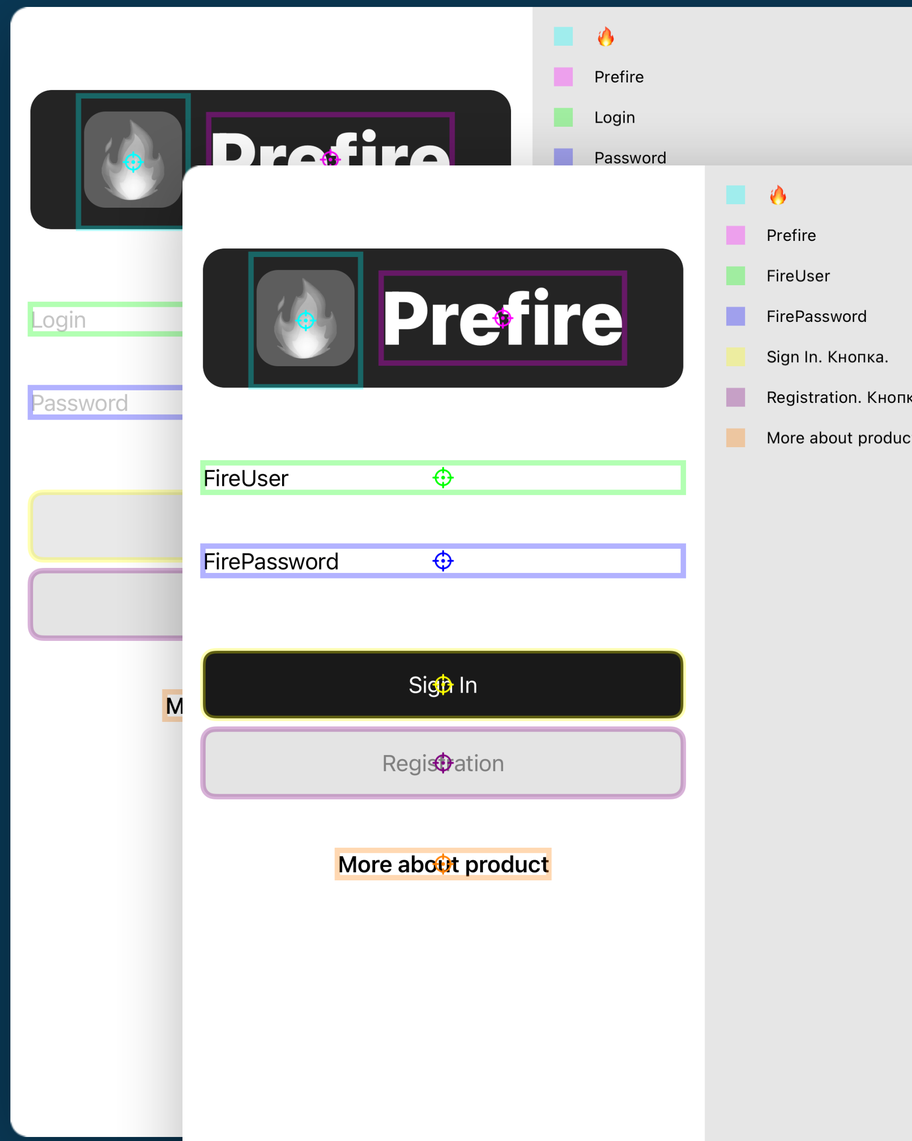
```
class PreviewTests: XCTestCase {
    {% for type in types.types where type.implements.PrefireProvider %}
    func test_{{ type.name|lowerFirstLetter|replace:"_Previews", "" }}() {
        for view in {{ type.name }}._allPreviews {
            // Given
            let preview = {{ type.name }}._allPreviews.first
            let isScreen = preview?.layout == .device
            let device = preview?.device

            // When
            var view = view.content
            view = isScreen ? view : AnyView(view.frame(width: device.size?.width).fixedSize(vertical: true))

            // Then
            assertSnapshot(
                matching: UIHostingController(rootView: view),
                as: .accessibilityImage(showActivationPoints: .always)
            )
        }
    }
    {% endfor %}
}
```

Используем кодогенерацию

Результат

Запускаем и получаем результат 🚀

```
func test_prefireView_Preview() {
    for view in PrefireView_Preview._allPreviews {
        // Given
        let preview = PrefireView_Preview._allPreviews.first
        let isScreen = preview?.layout == .device
        let device = preview?.device?.snapshotDevice() ?? deviceConfig

        // When
        var view = view.content
        view = isScreen ? view : AnyView(view.frame(width: device.size?.width))

        // Then
        assertSnapshot(
            matching: UIHostingController(rootView: view),
            as: .accessibilityImage(showActivationPoints: .always)
        )
    }
}
```



Используем максимум от SwiftUI Preview

ozon tech

Будем реализовывать

- Playbook (Demo) App
- Простой performance анализ
- Snapshot тесты
- Accessibility тесты

Используем максимум от SwiftUI Preview

ozon tech

Будем реализовывать

- Playbook (Demo) App
- Простой performance анализ
- Snapshot тесты
- Accessibility тесты

Project 🔥Prefire

Project Prefire

OZON tech

Проект, реализующий все, что мы обсудили

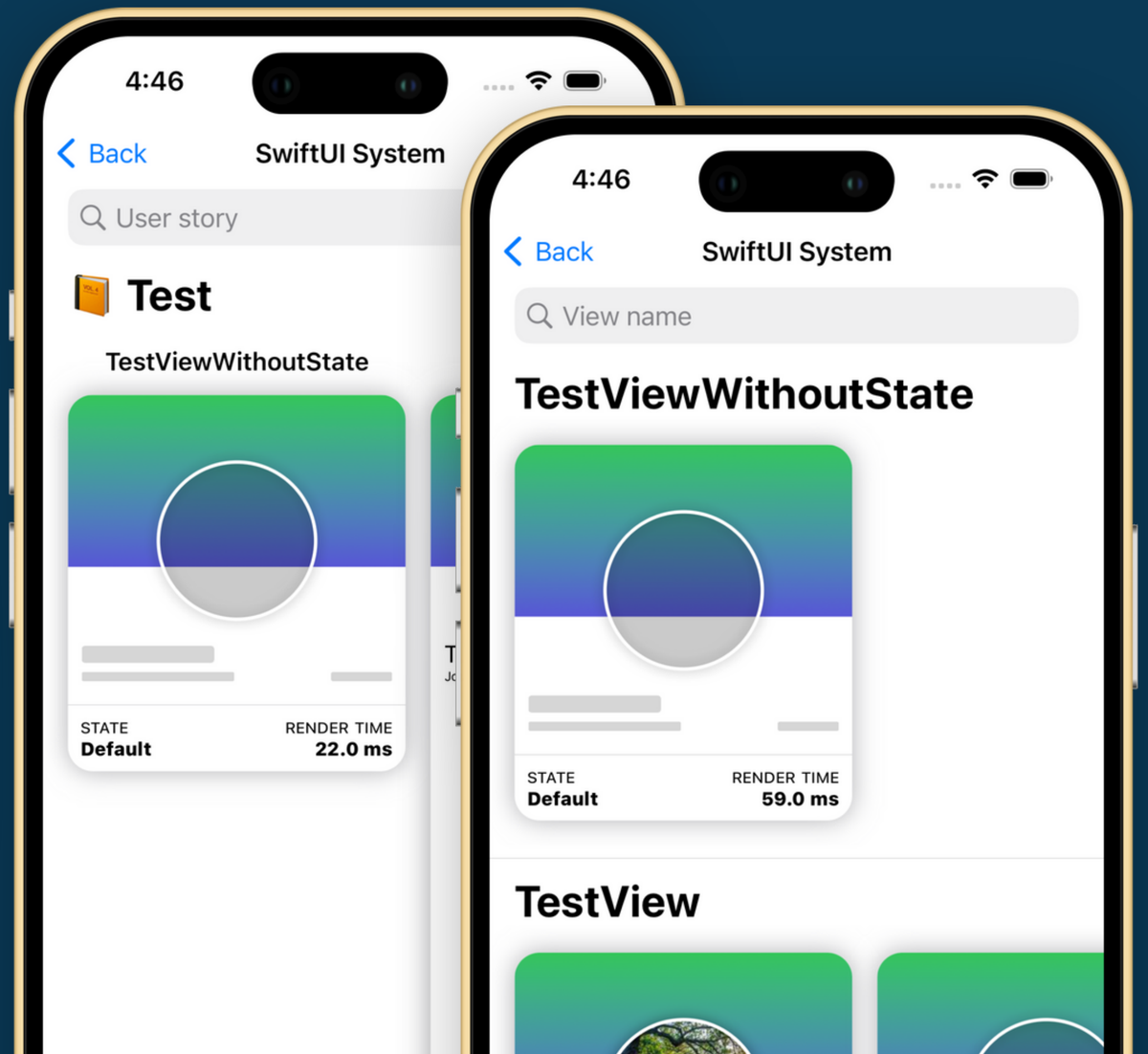
Включает в себя примеры и подробную инструкцию

https://github.com/BarredEwe/Prefire

Еще немного про проект

Буду рад любому фидбеку и звездочкам ⭐

🔥 Prefire



Итоги

- Выжили максимум из **SwiftUI Preview**
- Очень сильно уменьшили ручной код, генерацией
- **Playbook App**
- **Snapshot** тесты
- **Accessibility** Тесты

OZON tech

Спасибо за внимание

Максим. Руководитель группы разработки мобильных приложений iOS

GitHub: BarredEwe
Telegram: @BarredEwe
LinkedIn: BarredEwe

🔥 Prefire