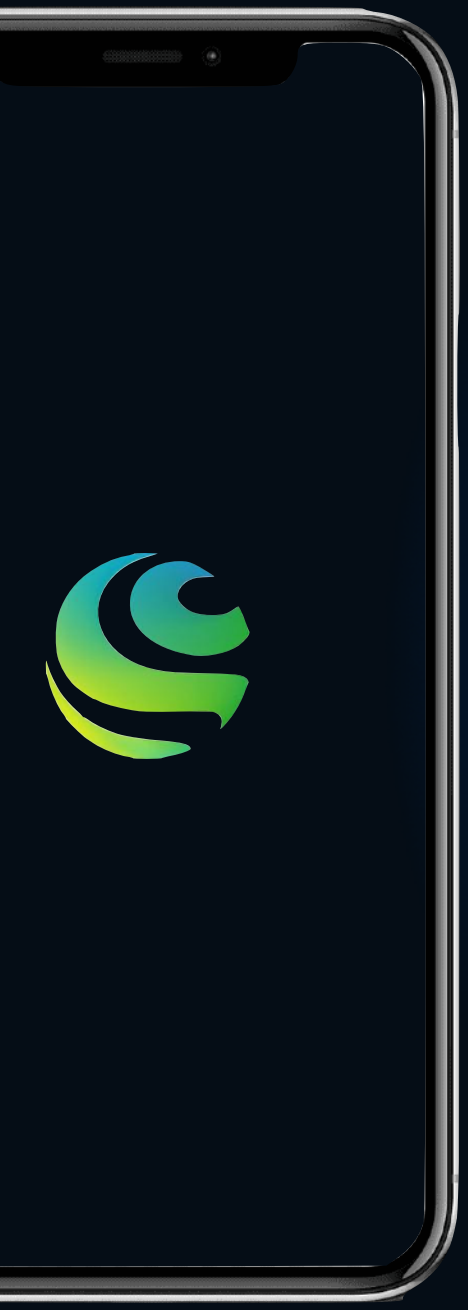


Вдохновившись SwiftUI, создаём дизайн систему на UIKit-е

SBER DEVICES

Анастасия Соколан
проект Smart Home
департамент SberDevices

План доклада

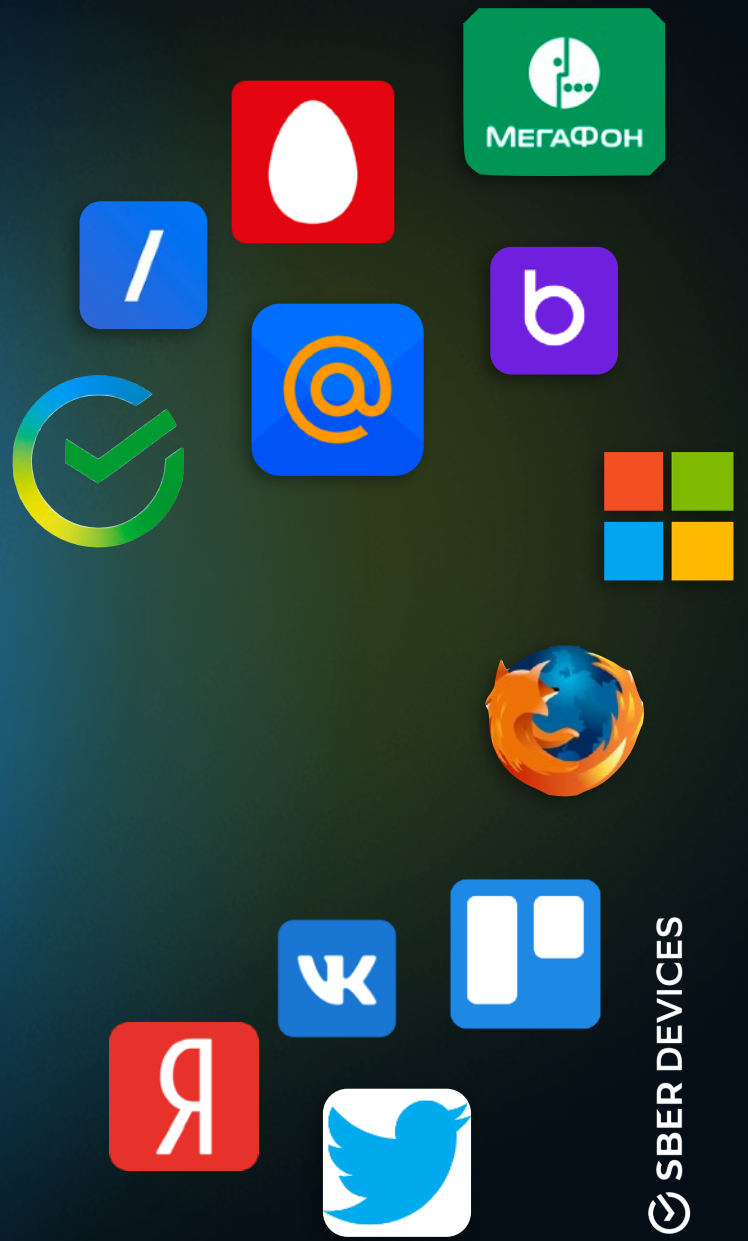
Вспомним: Design System и для чего она нужна

Наше кастомное решение: этапы, положения, ремарки

Coding session

Какие задачи решает DS?

- унификация подходов
- переиспользуемость
- снижение количества ошибок
- сокращение срока создания компонента



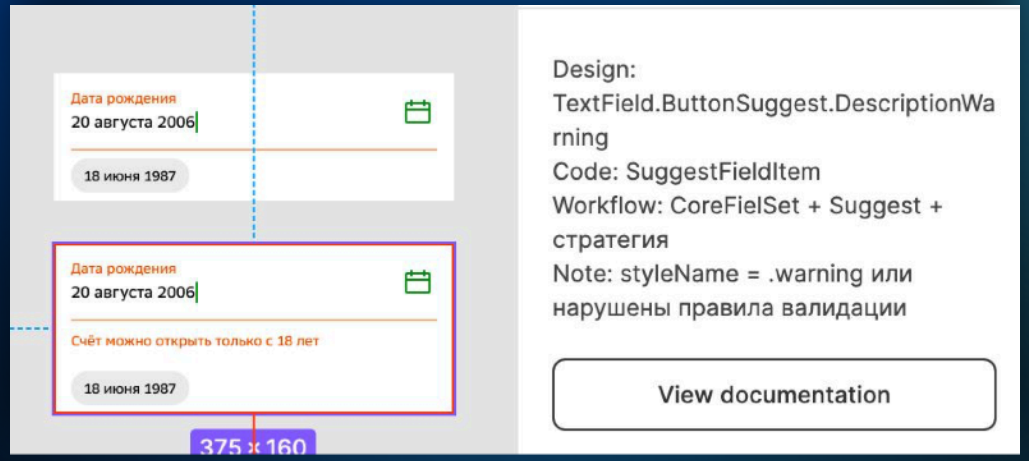
Что на практике?

- must have для крупного проекта
- снижение стоимости разработки
- унификация дизайна с другими поверхностями

Опыт формирования дизайн системы Сбер Онлайн

Начали разработку в 2017 году

- Совместимость кода и дизайна и наоборот
- Подробная документация
- Регламент для создания новой компоненты
- Синхронизация между Android и iOS
- Плагин для конвертации макетов iOS в Android



SBER DEVICES

Опыт формирования дизайн системы Сбер Онлайн

Советы от ребят

- на старте использовать snapshot-тесты
- регулярное проведение брейнштормов
- согласованность действий участников команды в разных частях приложения

Но в 2020 началась наша собственная история…

SBER DEVICES

Знакомство с Салютом

* картинка системы Салюта

* основные факты о приложении

Краткий обзор фреймворка UIKit

Возможности

- инструмент IB, позволяющий верстать UI без написания кода
- возможность комбинировать визуальную и кодовую вёрстку
- комбинирование архитектурных подходов

Краткий обзор фреймворка UIKit

Ограничения

- ориентирован под iOS, tvOS
- много кода, описывающего вёрстку UI компонента

Краткий обзор фреймворка SwiftUI

Возможности

- кроссплатформенность UI
- интегрированный режим просмотра
- декларативный подход
- меньше кода
- новые UI элементы и функции
- удобные обёртки свойств

Краткий обзор фреймворка SwiftUI

Ограничения

- ограниченность документации и ресурсов
- линейка версий OS
- неполная коллекция UI компонент
- ограниченность некоторых UI компонент
- применимость архитектурных паттернов

Наш выбор

Критерии:
- поддержка Backend-Driven UI
- iOS 11+
- нестандартный UI/UX дизайн
- централизация навигационных переходов
- поддержка модульности

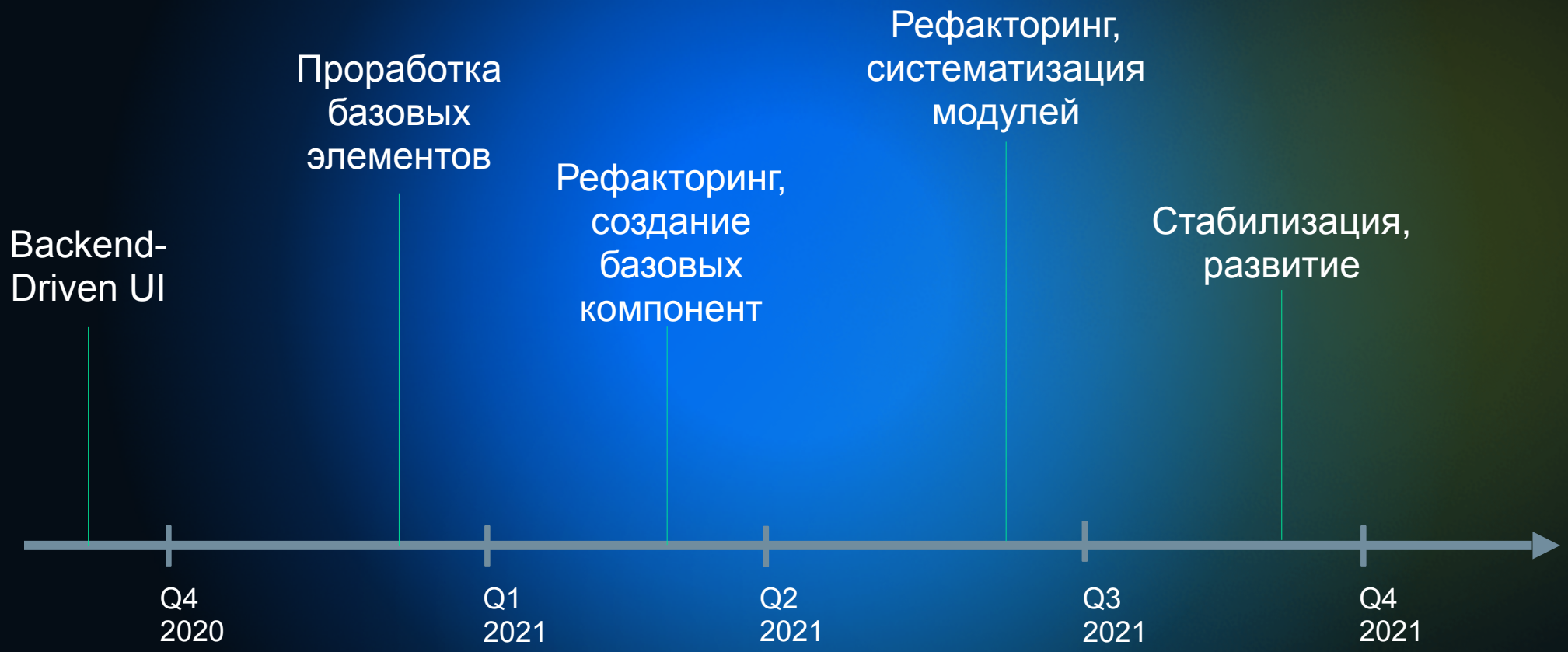
UIKit

Особенности Design System в SberDevices

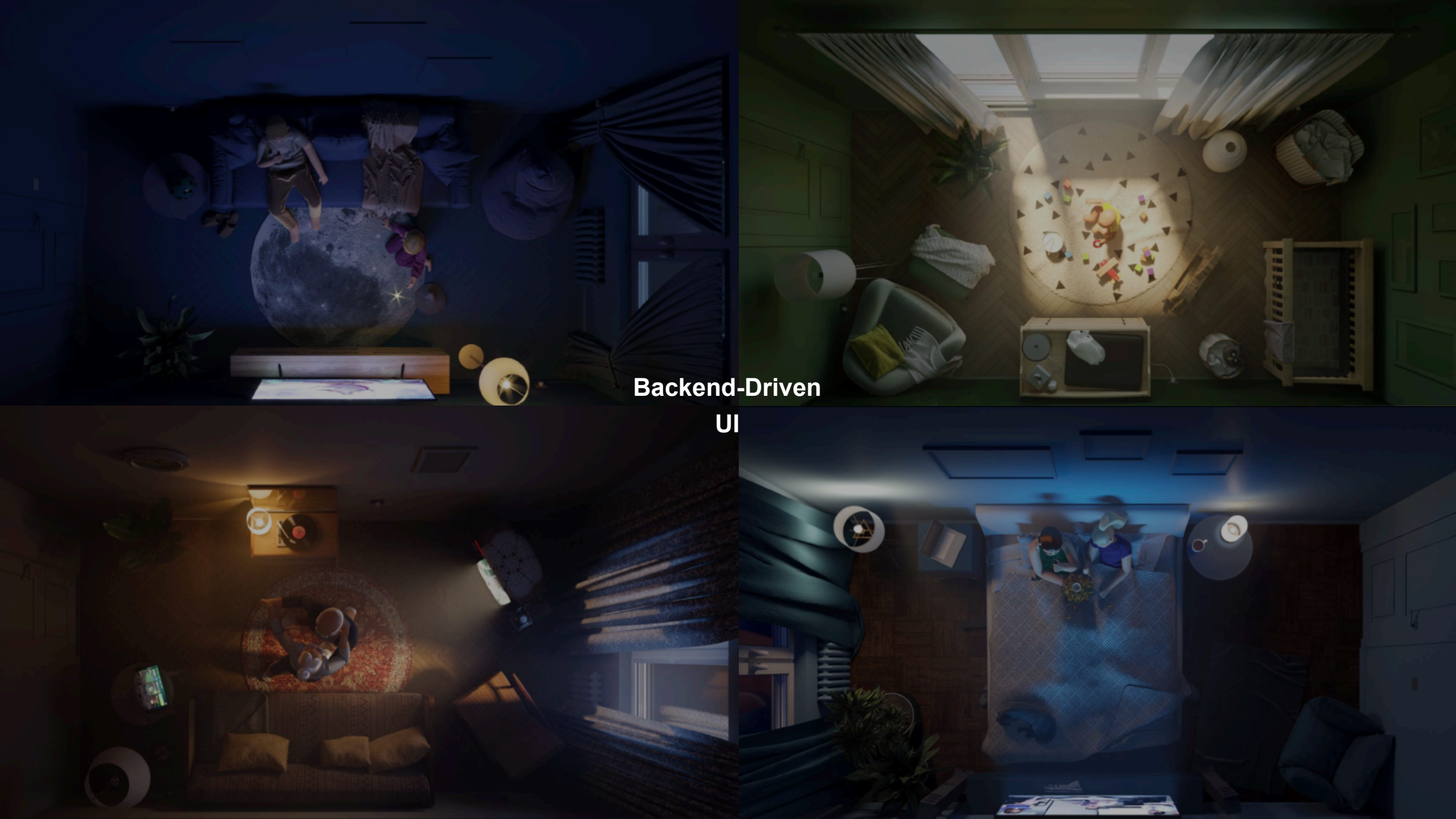
•картинка

SBER DEVICES

Наш путь



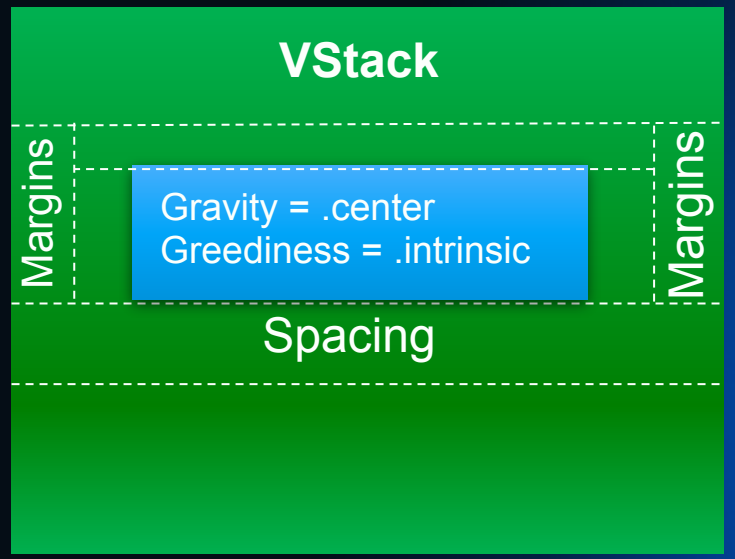
SBER DEVICES

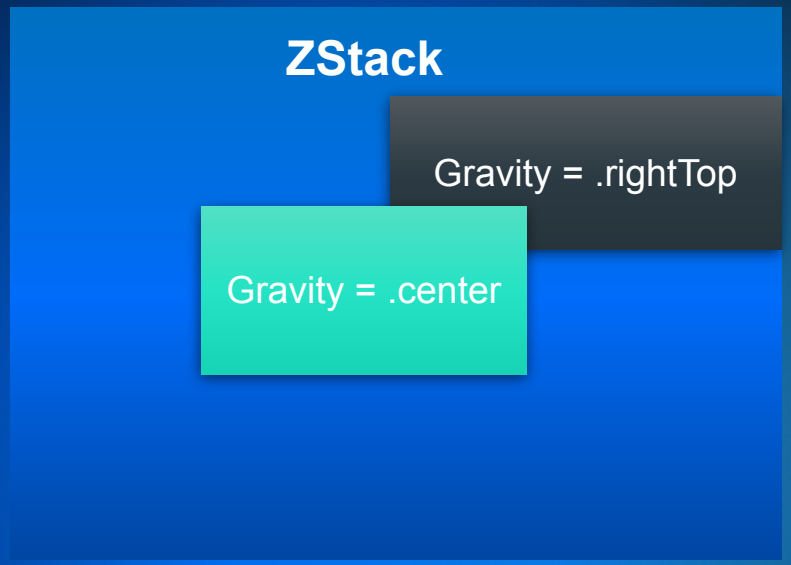

Backend-Driven
UI

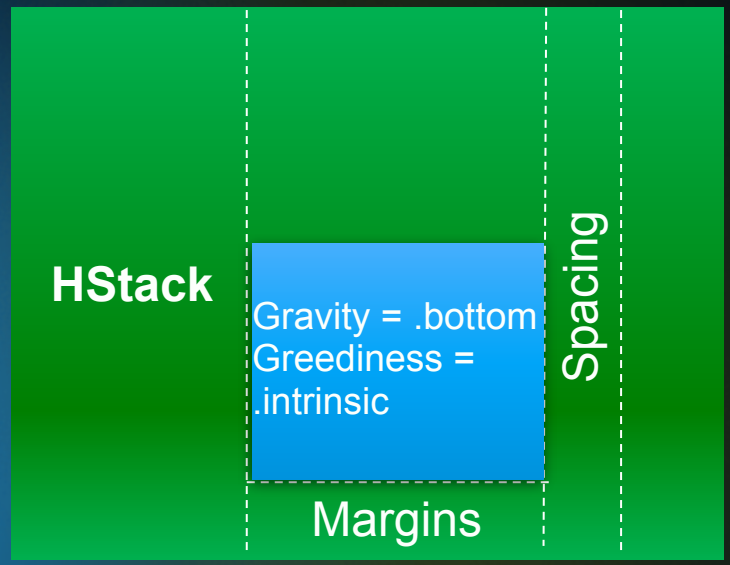
Проработка базовых элементов


```
struct UIViewContext {
    let imageProvider: ImageProvider
    let remoteImageProvider: ImageProvider
    let stringProvider: StringProvider
}
```

Первые шаги к модификаторам



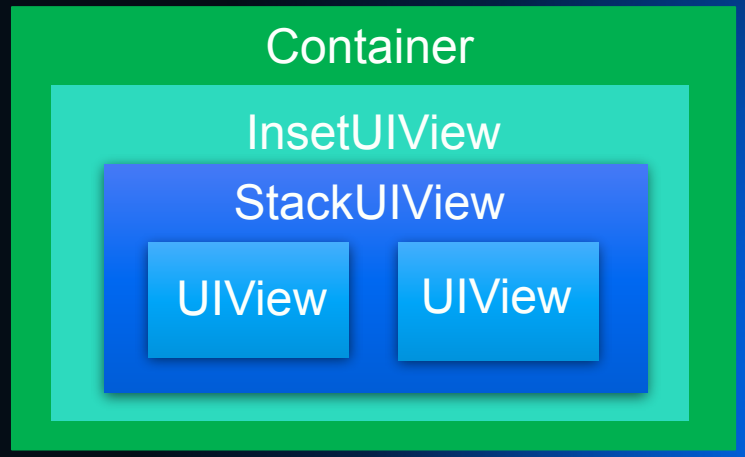




- Gravity – позиционирование
- Greediness – пропорция размера

Наши первые шаги: базовые элементы

С чего начали



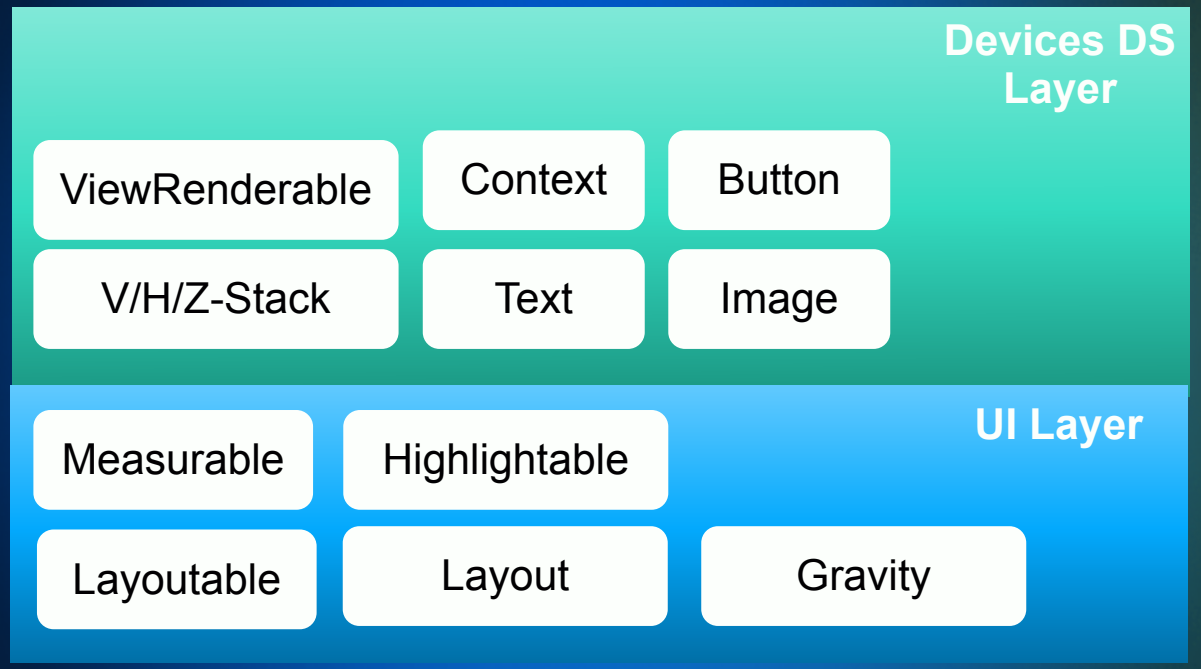

```
protocol UIViewRenderable {
   func makeView(context: UIViewContext)
   -> UIView
}
```

порождает UIView
context – элементы стилизации, провайдеры ресурсов

Почему двинулись дальше

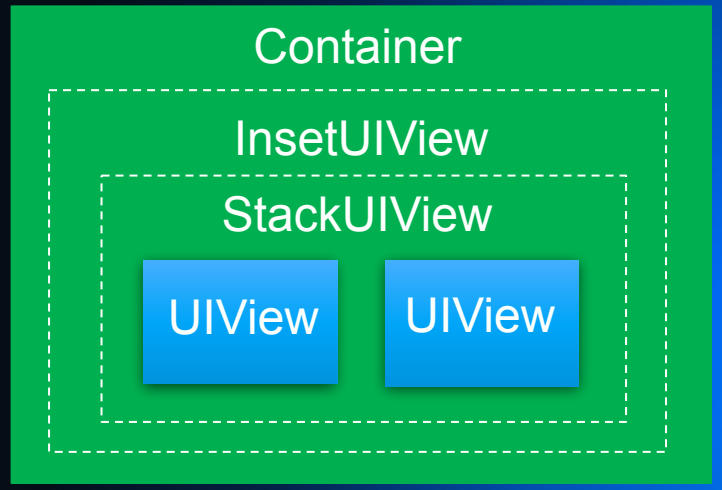
performance hit на сложных композициях
нельзя обновить

Рефакторинг, добавление базовых компонент



Шаг 2: ViewRenderable и View

С чего начали




```
protocol View: Measurable,
Layoutable {
   func add(to container: View)
   func remove(from container: View)
}


 protocol ViewRenderable {
    func renderView(context) -> View
 }
```

- легковесная замена UIView
- позволяет избежать лишней вложенности

- неограниченное количество похожих модификаторов
- описание всей верстки без дополнительного подкласса UIView

Шаг 2: ViewRenderable и View

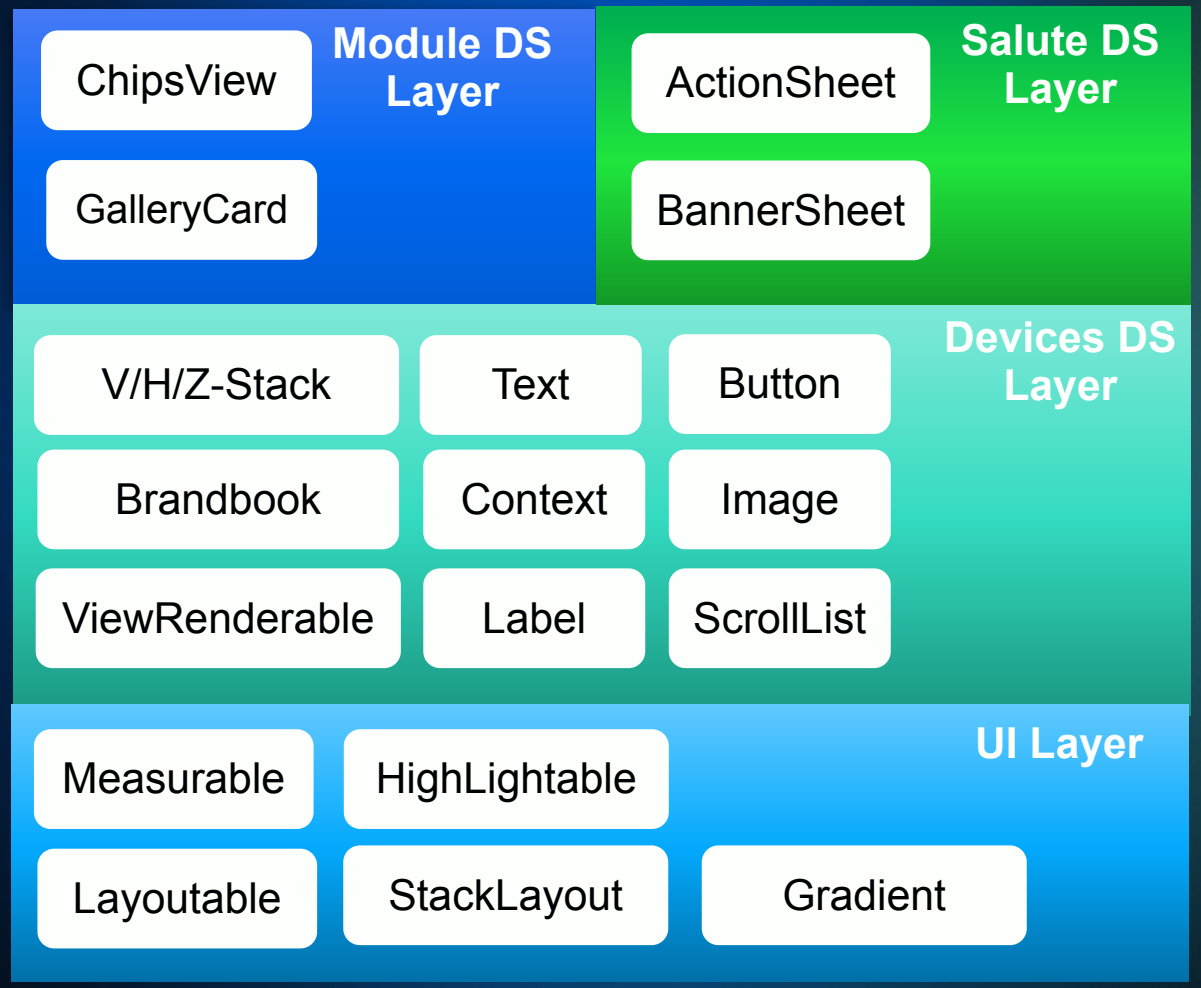
Почему двинулись дальше

- не предусмотрели reusing View
- «размашистая» архитектура для обновляемого компонента

Рефакторинг, систематизация модулей

- Ассоциативная картинка

Систематизация модулей



Поддерживаем модульность

С чего начали

```
struct ViewContext {
  struct Key<Value>: Hashable
  func register(_ value: Value, forKey key: Key<Value>)
  subscript(key: Key<Value>) -> Value
}

var context = ViewContext()
context.register(ImageProvider(), forKey: .imageProvider)

let imageProvider = context[.imageProvider]
```

более слабая система типов
можно определять зависимости в подмодулях
аналог SwiftUI @Environment

Почему двинулись дальше

ревизия провайдеров
ресурсов в рамках отдельных
модулей

Избавляемся от лишней сущности в обновляемом компоненте

Раньше

```
struct Model {
   let initialText: String
}

protocol Interactor {
   func setText(_ text: String)
}
```

- громоздкая архитектура для обновляемой компоненты
- логика обновления внутри лишней сущности

Сейчас

```
struct Model {
   @Property
   var initialText: String
}
```

- рычаги управления инжектятся в модель (коллбэки, property)

Создание модели

Сторона 1: Json -> Deserializable

```
struct ImageModel: Deserializable {
    let address: ImageAddress
    let scaleMode: ScaleMode
    let verticalAlignment: VerticalGravity
    let size: ImageSize
}
```

Equatable модель с бекенд стилизацией

Сторона 2: UI реализация

```
extension ImageModel: ViewRenderable {
    func makeView(context: ViewContext,
                  reusing view: View?) -> View {
        if let imageView = view as? ImageView {
            imageView.update(self, context)
            return imageView
        }
        return ImageView(model: self, context)
    }
}
```

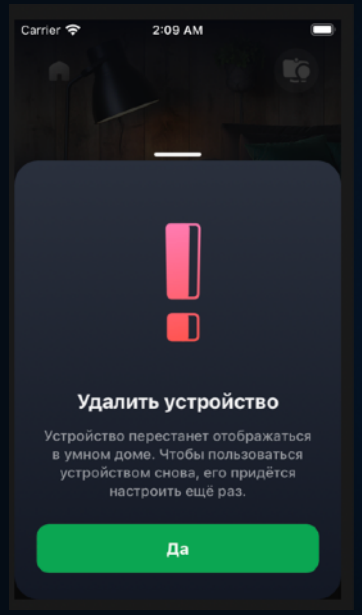
описывает вёрстку компонента и механизм переиспользования

Стабилизация и развитие

- улучшение производительности (кеширование layout-а, рендеринг бекграунда и т.п.)
- создание дебаг инструмента
- проработка снепшот тестов
- проработка документации

Coding session

Создадим UI компоненту по готовому макету вместе




```
11      func addWarningScreen() -> StackScreen {
12          let content: ViewRenderable
13
14          let gradient = Gradient(
15              style: Gradient.DefaultStyle(
16                  config: GradientStyleIDBackground(styleID: .saluteSheetBackground)
17                      .makeGradientConfig(context: context)
18              )
19          )
20
21          let sheetModel = SheetModel(
22              content: content,                    Constant 'content' used before being initialized
23              background: gradient,
24              hasBorderLine: true,
25              cornerRadius: 24
26          )
27
28          return SheetViewController(model: sheetModel, context: context)
29      }
30  }
31
```


SBER DEVICES

Ремарки по созданию дизайн системы

- проведение ревизий в дизайне и кодовой базе
- тестовое покрытие
- коммуникации, проведение брейнштормов
- согласованность действий участников команды на разных частях приложения

SBER DEVICES