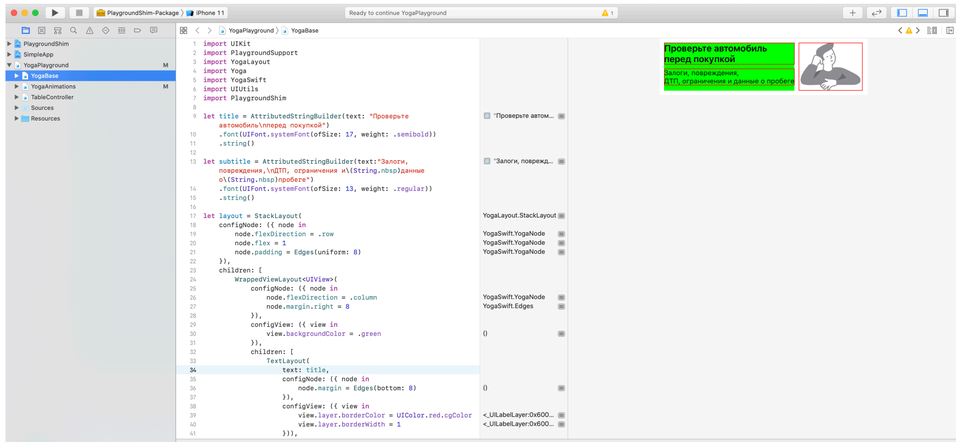
Подходы к построению UI

И как сделать свой декларативный фреймворк на Yoga

auto.ru

- DAU > 100k
- > 100 экранов
- Верстка в коде
- Лейаут на Yoga
- Дифф на основе алгоритма Майерса

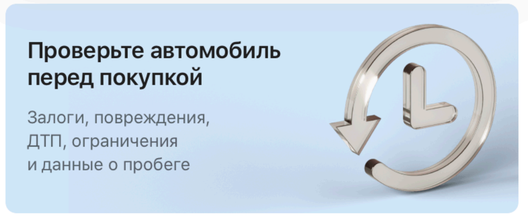
Код



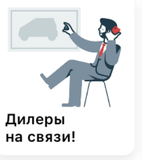
- https://github.com/fanruten/YogaPlayground
- Нужно собрать таргет PlaygroundShim или SimpleApp под симулятор
- Потом смотреть страницы в YogaPlaygorund

Классификация

Компоненты



BMW 5 Series G30 Autobahn
POV TEST Drive by AutoTopNL



Позвонить
с 9:00 до 21:00

Коллекции

Характеристики

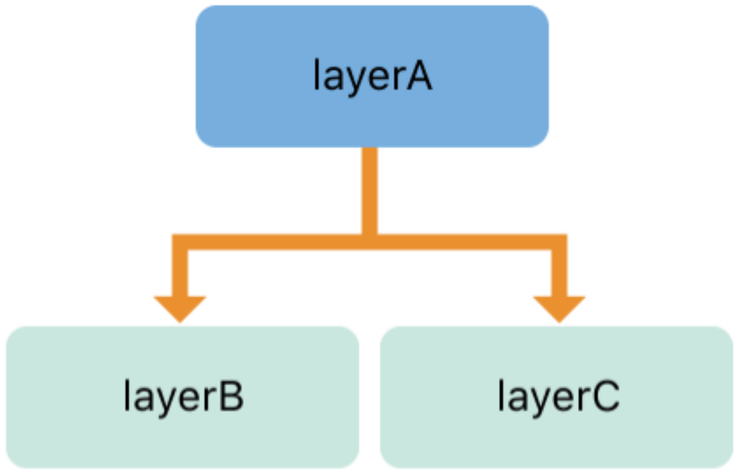
Статус	В наличии
Год выпуска	2019
Кузов	Седан
Цвет	Черный
Двигатель	2.0 л / 190 л.с. / Дизель
Налог	10 241 ₽
КПП	Автоматическая
Привод	Полный
Руль	Левый
ПТС	Оригинал
Таможня	Растаможен
Обмен	Возможен
VIN	X4X**********

Задачи при работе с UI

- Создание иерархии элементов
- Конфигурирование элементов
- Задание местоположения элементов (layout)
- Объединение элементов вместе для построения экранов или более сложных элементов (UITableView/UICollectionView/UIStackView/…)

Создание иерархии

- CALayer.addSublayer()
- UIView.addSubview()

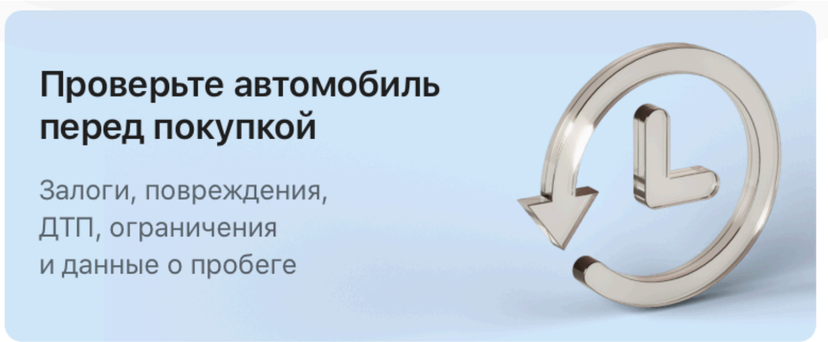


Создание иерархии

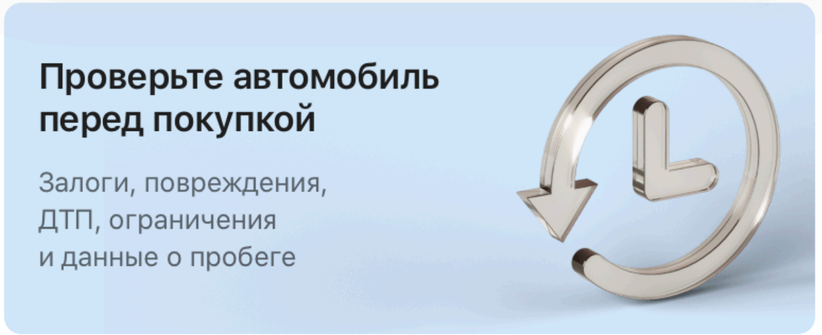
- Используем главный поток
- Создавать UIView дорого. Много лишнего вроде willMove(toSuperview:)/willRemoveSubview()
- CALayer где уместно

Создание иерархии

```
let title = UILabel(frame: .zero)
let subtitle = UILabel(frame: .zero)
let imageView = UIImageView(frame: .zero)

let component = UIView(frame: .zero)
component.addSubview(title)
component.addSubview(subtitle)
component.addSubview(imageView)
```



Конфигурирование элементов

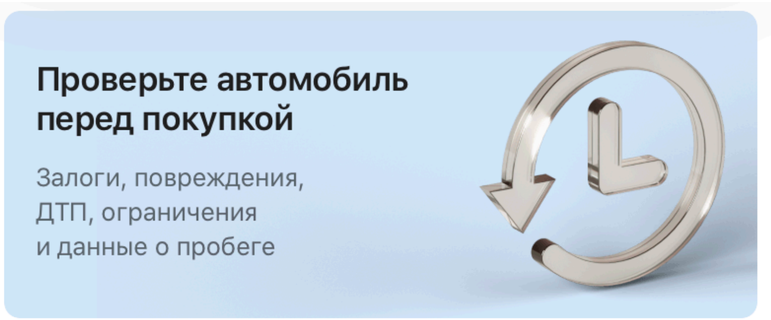



```
title.attributedText = NSAttributedString(
    string: "Проверьте автомобиль\nперед покупкой",
    attributes: [NSAttributedString.Key.font: UIFont.systemFont(ofSize: 17, weight: .semibold)])

subtitle.attributedText = NSAttributedString(
    string: "Залоги, повреждения,\nДТП, ограничения и\(String.nbsp)данные о\(String.nbsp)пробеге",
    attributes: [NSAttributedString.Key.font: UIFont.systemFont(ofSize: 13, weight: .regular)])

imageView.image = UIImage(named: "banner_history")
```

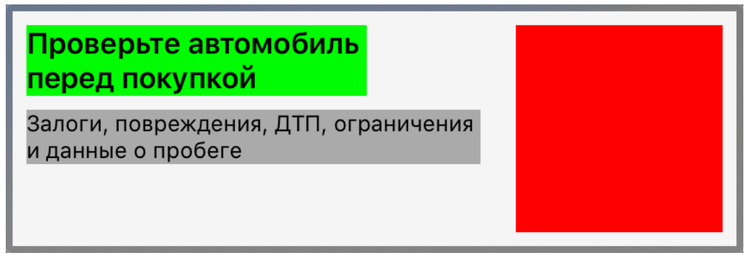
Расположение элементов

- Задаем положение элементов руками (ручной лейаут)
- Используем абстракцию (например autolayout)

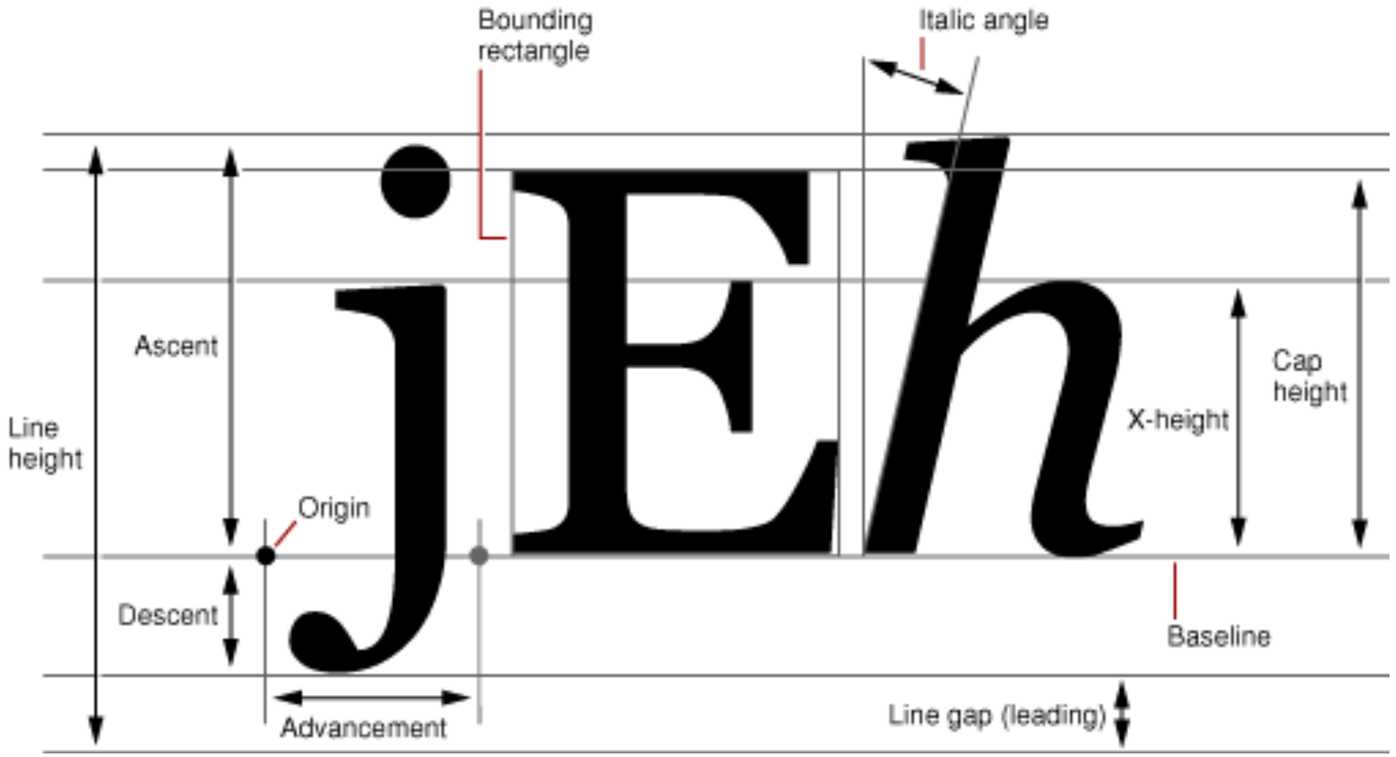
Сложности



- Надо считать размер текста

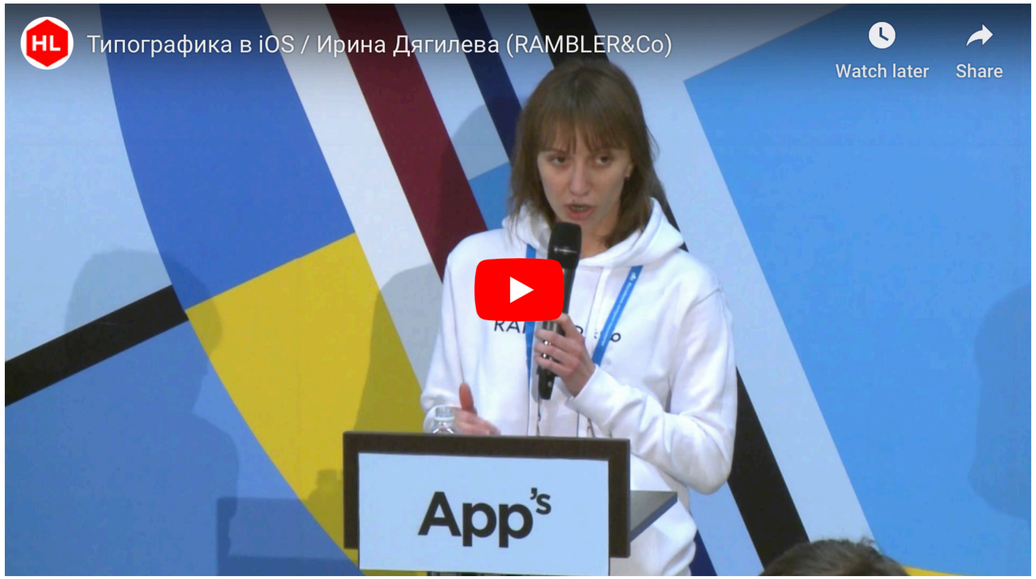


Рендер текста



Рендер текста

- Доклад Типографика в iOS от Ирины Дягилевой



https://habr.com/en/company/oleg-bunin/blog/354184/

Размер текста

```
extension NSAttributedString {
   open func boundingRect(with size: CGSize, options: NSStringDrawingOptions = [], context: NSStringDrawingContext?) -> CGRect
}

public struct NSStringDrawingOptions : OptionSet {
    public static var usesLineFragmentOrigin: NSStringDrawingOptions { get }
    public static var usesFontLeading: NSStringDrawingOptions { get }
    public static var usesDeviceMetrics: NSStringDrawingOptions { get }
    public static var truncatesLastVisibleLine: NSStringDrawingOptions { get }
}
```

Размер текста

- Расчет многострочного текста требует usesLineFragmentOrigin
- Результат boundingRect дробный и должен быть округлен вверх до ближайшего целого в пикселях
- UILabel округляет до (1 / UIScreen.main.scale)

```
let scale = UIScreen.main.scale
let width = ceil(width * scale) / scale
```

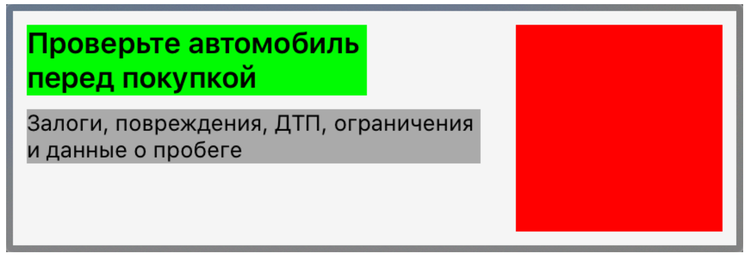
Рендер текста

- Рендер текста идет на CPU
- Можно попробовать CATextLayer
- Если критично, то выносим в фон

```
extension NSAttributedString {
    func render(_ size: CGSize) -> UIImage? {
        UIGraphicsBeginImageContextWithOptions(size, false, 0)

        guard UIGraphicsGetCurrentContext() != nil else {
            return nil
        }

        defer { UIGraphicsEndImageContext() }

        draw(with: CGRect(origin: .zero, size: size),
             options: .usesLineFragmentOrigin,
             context: nil)

        return UIGraphicsGetImageFromCurrentImageContext()
    }
}

class AsyncLabel: UIView {
    private let renderQueue = DispatchQueue(label: "render_queue")

    func renderAsync(text: NSAttributedString?) {
        renderQueue.async {
            let image = text?.render(self.bounds.size)

            DispatchQueue.main.async {
                self.layer.contents = image?.cgImage
            }
        }
    }
}
```

Ручной лейаут

```
let componentWidth: CGFloat = 420

let imageSize = CGSize(width: 120, height: 120)
let padding = UIEdgeInsets(top: 8, left: 8, bottom: 8, right: 8)

let maxContentWidth = componentWidth - padding.left - padding.right
let maxLabelWidth = maxContentWidth - imageSize.width

let titleSize = Text.attributed(title.attributedText!)
    .textSizeForLabel(within: CGSize(width: maxLabelWidth, height: .infinity))

let subtitleSize = Text.attributed(subtitle.attributedText!)
    .textSizeForLabel(within: CGSize(width: maxLabelWidth, height: .infinity))

title.frame = CGRect(x: padding.left, y: padding.top,
                     width: titleSize.width, height: titleSize.height)

subtitle.frame = CGRect(x: padding.left, y: title.frame.maxY + 8,
                        width: subtitleSize.width, height: subtitleSize.height)

imageView.frame = CGRect(x: padding.left + maxContentWidth - imageSize.width, y: padding.top,
                         width: imageSize.width, height: imageSize.height)

component.frame = CGRect(x: 0, y: 0,
                         width: componentWidth, height: imageSize.height + padding.top + padding.bottom)
```



Ручной лейаут

- Нормальная тема если делаем небольшие компоненты
- Можем посчитать в фоне

Готовые движки

- Autolayout
- SwiftUI
- LayoutKit
- ComponentKit
- Texture/AsyncDisplayKit
- Yoga

Готовые движки

- Autolayout
- SwiftUI
- LayoutKit
- ComponentKit
- Texture (AsyncDisplayKit)
- Yoga

Готовые движки

- Autolayout
- SwiftUI
- LayoutKit
- ComponentKit
- Texture (AsyncDisplayKit)
- Yoga

Декларативный лейаут

```
let image = SizeLayout<UIImageView>(width: 50, height: 50, config: { imageView in
    imageView.image = UIImage(named: "earth.jpg")
})

let label = LabelLayout(text: "Hello World!", alignment: .center)

let stack = StackLayout(
    axis: .horizontal,
    spacing: 4,
    sublayouts: [image, label])

let insets = UIEdgeInsets(top: 4, left: 4, bottom: 4, right: 8)
let helloWorld = InsetLayout(insets: insets, sublayout: stack)

helloWorld.arrangement().makeViews(in: rootView)
```

Hello World!

Пример с использованием LayoutKit

Что использовать

- Autolayout — может тормозить не декларативный UI
- SwiftUI — вроде ниче, но нужна iOS 13
- LayoutKit — вроде норм, но страшно
- ComponentKit — только для ObjectiveC++
- Texture (AsyncDisplayKit) — не декларативный UI
- Yoga — слишком простой и не декларативный UI

Делаем свой движок

- Не хотим тяжелых зависимостей
- Нельзя SwiftUI (iOS < 13)
- Нужно строить UI из ресурсов

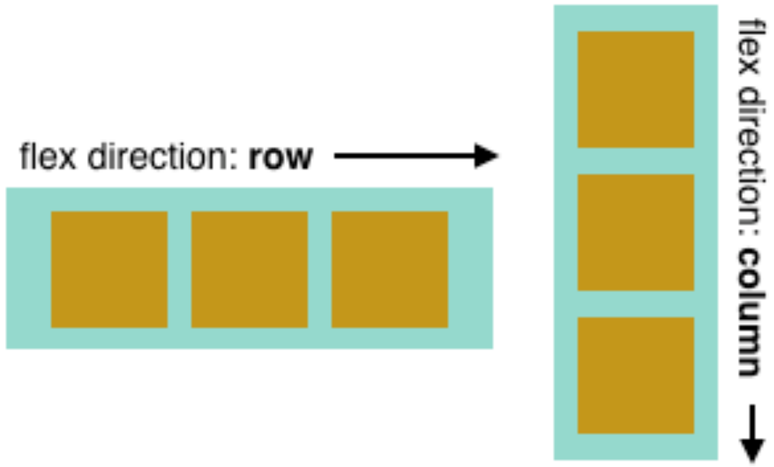
Yoga

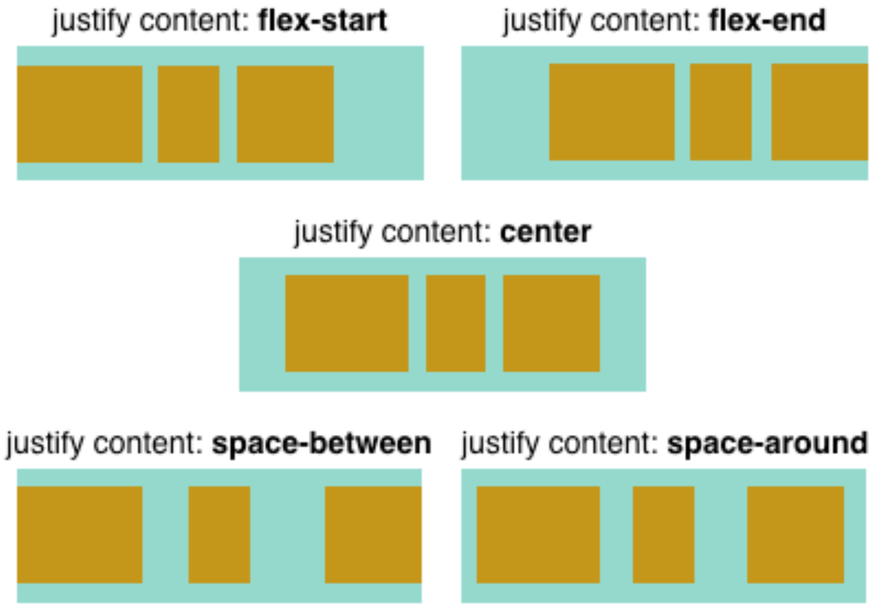
- Способ лейаута, для расположения элементов в строках и столбцах. Элементы растягиваются для того, чтобы заполнить дополнительное пространство и сжимаются, чтобы уместиться в в небольшом пространстве.

Yoga

- Открытый
- Простой
- Документированный
- Протестированный
- Клосс-платформенный

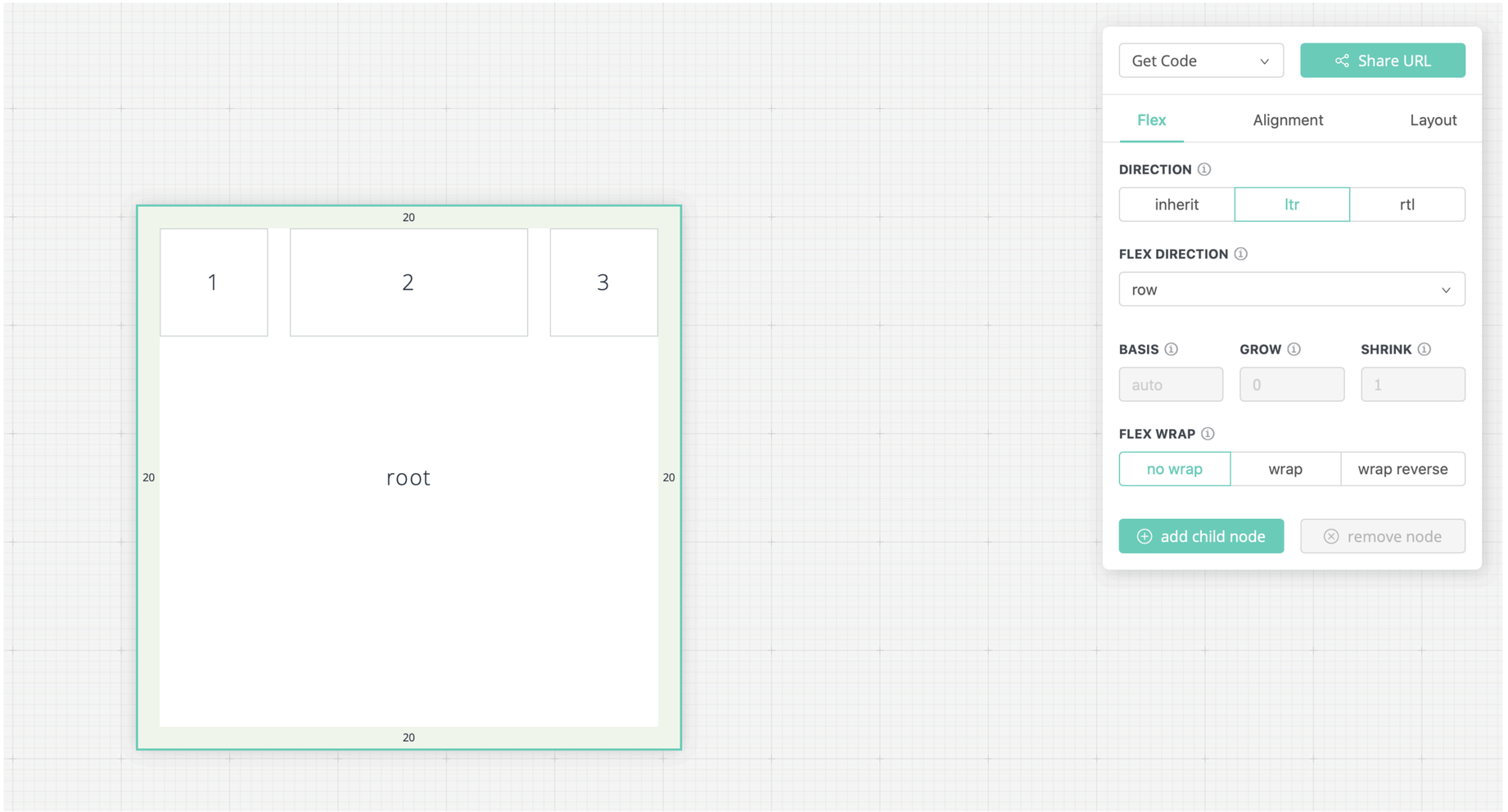
Yoga





https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Yoga

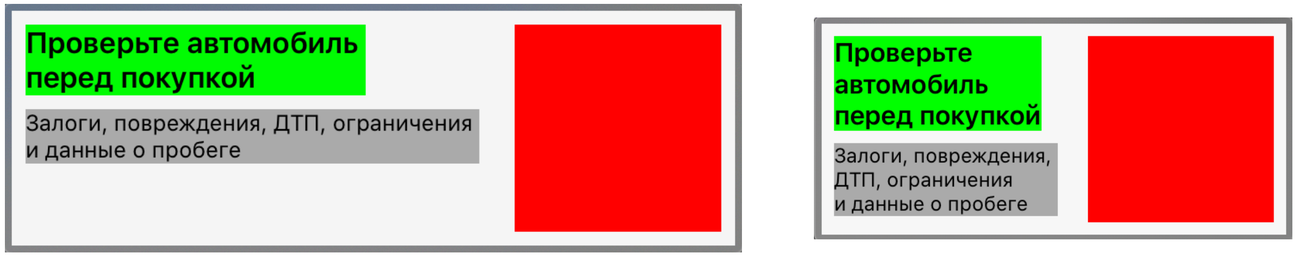


https://yogalayout.com

Yoga

- Контейнеры (row/column)
- Size-constraints (не может быть детей)

Yoga



- Текст это самый яркий пример Size-constraint элемента

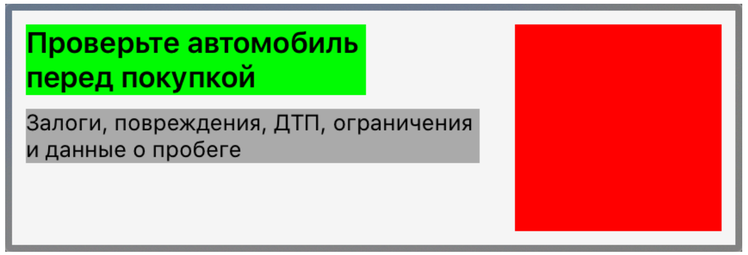
Yoga

```
enum YGMeasureMode {
    case undefined
    case exactly
    case atMost
}

typealias ContentSizeMeasureFunc = (Float, YGMeasureMode, Float, YGMeasureMode) -> CGSize
```

YogaNode

```
public class YogaNode {

    public var flexDirection: YGFlexDirection = .row
    public var flexWrap: YGWrap = .noWrap
    public var justifyContent: YGJustify = .flexStart
    ...
    public var tag: Int?

    public typealias ContentSizeMeasureFunc = (Float, YGMeasureMode, Float, YGMeasureMode) -> CGSize
    public var contentSize: ContentSizeMeasureFunc?

    public func addSubnode(_ subnode: YogaNode)

    @discardableResult public func calculateLayout(with size: CGSize) -> CGSize
    public var frame: CGRect
}
```

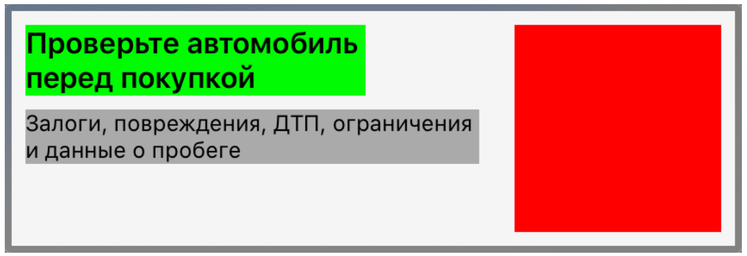
Считаем размеры




```
let title = YogaNode()
title.contentSize = ...
title.margin = Edges(bottom: 8)

let subtitle = YogaNode()
subtitle.contentSize = ...

let verticalStack = YogaNode()
verticalStack.flexDirection = .column
verticalStack.addSubnode(title)
verticalStack.addSubnode(subtitle)

let image = YogaNode()
image.size = YogaSize(width: 120, height: 120)

let stack = YogaNode()
node.flexDirection = .row
node.padding = Edges(uniform: 8)
stack.addSubnode(verticalStack)
stack.addSubnode(image)

stack.calculateLayout(CGSize(width: 320, height: .nan))
```

Делаем свой движок

```
let layout = StackLayout(
    children: [
        StackLayout(
            children: [
                TextLayout(model: title, configNode: ({ node in
                    node.margin = Edges(bottom: 8)
                })),
                TextLayout(model: subtitle)
            ],
            configNode: ({ node in
                node.flexDirection = .column
            })
        ),
        ImageLayout(
            model: UIImage(named: "logo"),
            configNode: ({ node in
                node.size = YogaSize(width: 120, height: 120)
            }))
    ],
    configNode: ({ node in
        node.flexDirection = .row
        node.padding = Edges(uniform: 8)
    })
)

let view = layout.createView(width: 320, height: .nan)
```



Layout

```
public protocol ViewBuilder {
    func configView(_ view: UIView)
    func createView() -> UIView
}

public protocol Layout {
    func configNode(_ node: YogaNode)
    var viewBuilder: ViewBuilder? { get }
    var children: [Layout] { get }
}
```

StackLayout

```
public class StackLayout: Layout {
    private let configNodeBlock: ((YogaNode) -> Void)?

    public let viewBuilder: ViewBuilder? = nil
    public let children: [Layout]

    public init(configNode: ((YogaNode) -> Void)? = nil,
                optChildren: [Layout?] = []) {
        self.children = optChildren.compactMap { $0 }
        self.configNodeBlock = configNode
    }

    public init(configNode: ((YogaNode) -> Void)? = nil,
                children: [Layout] = []) {
        self.children = children
        self.configNodeBlock = configNode
    }

    public func configNode(_ node: YogaNode) {
        configNodeBlock?(node)
    }
}
```

WrappedViewLayout

```
public class WrappedViewLayout<View>: Layout, ViewBuilder where View: UIView {
    private let configNodeBlock: ((YogaNode) -> Void)?
    private let configViewBlock: ((View) -> Void)?
    private let createViewBlock: (() -> View)

    public var viewBuilder: ViewBuilder? {
        return self
    }
    public let children: [Layout]

    public init(createViewBlock: @escaping (() -> View) = { View(frame: .zero) },
                configNode: ((YogaNode) -> Void)? = nil,
                configView: ((View) -> Void)? = nil,
                children: [Layout] = []) {
        self.children = children
        self.createViewBlock = createViewBlock
        self.configNodeBlock = configNode
        self.configViewBlock = configView
    }

    open func configNode(_ node: YogaNode) {
        configNodeBlock?(node)
    }

    open func configView(_ view: UIView) {
        guard let typedView = view as? View else {
            return
        }

        configViewBlock?(typedView)
    }

    open func createView() -> UIView {
        return createViewBlock()
    }
}
```

TextLayout

```
public class TextLayout: WrappedViewLayout<UILabel> {
    public init(text: NSAttributedString, configNode: ((YogaNode) -> Void)? = nil, configView: ((UILabel) -> Void)? = nil) {
        super.init(
            configNode: ({ node in
                node.contentSize = DefaultTextSizeMeasureFunc(for: text)
                configNode?(node)
            }),
            configView: ({ view in
                view.numberOfLines = 0
                view.lineBreakMode = .byClipping
                view.attributedText = text
                configView?(view)
            }))
    }
}
```

Пример

```
StackLayout(
   children: [
        TextLayout(text: title),
        WrappedViewLayout<UIImageView>(
            configNode: ({ node in
                node.size = YogaSize(squareSize: 24)
            }),
            configView: ({ view in
                view.image = UIImage(named: “logo”)
            }))
    ],
    configNode: ({ node in
        node.flexDirection = .row
    }))
```

Создаем UIView

```
public class BasicViewHierarchyCreator {

    public init(rootComponent: Layout, boundingSize: CGSize)

    public func createView() -> UIView {
        calculateLayoutIfNeeded()

        let rootView = UIView(frame: CGRect(size: viewSize))
        ...
        return rootView
    }

    public lazy var viewSize: CGSize = {
        guard let node = layoutWithNodes.first()?.node else {
            return .zero
        }
        let size = node.calculateLayout(with: boundingSize)
        return size
    }()

    public func calculateLayoutIfNeeded() {
        _ = viewSize
    }
}
```

Что с анимациями

- Из коробки можно удобно анимировать только простые свойства
- Нужно научится переиспользовать UIView и задавать новые фреймы
- Не критично, если используем UICollectionView

Что с анимациями

```
var views = [String:UIView]()

let layout = StackLayout(
    ...
                TextLayout(
                    text: subtitle,
                    configNode: ({ node in
                        node.alignSelf = .center
                    }),
                    configView: ({ view in
                        view.aru_addTapHandler {
                            guard let imageView = views["image"] else {
                                return
                            }

                            UIView.animate(withDuration: 0.4) {
                                imageView.transform = imageView.transform.rotated(by: CGFloat.pi / 2)
                            }
                        }
                    }))
            ]
        ),
        ImageLayout(
            image: UIImage(named: "dude"),
            configView: ({ view in
                views["image"] = view
            }))
    ...
])
```

Коллекции

- Контент помещается на экран UIStackView, UIView.addSubview()
- Много контена, динамика UITableView/UICollectionView

UICollectionView

- Предоставить модель (список элемнтов)
- Предоставить изменения в модели (insert/delete/update)
- Предоставить информацию о расположении элементов (UICollectionViewLayout)

Модель

```
protocol CellAdapter {
    func createView() -> UIView
}

let model: [CellAdapter] = [
    ImageCell(imageNamed: "logo"),
    TextCell(text: "Row"),
    TextCell(text: "Row")
]
```

Изменения (Diff)

- Нужны для того, чтобы была анимация
- Происходят в UICollectionView.performBatchUpdates

Изменения (Diff)

Deletes are processed before inserts in batch operations.

This means the indexes for the deletions are processed relative to the indexes of the collection view’s state before the batch operation, and the indexes for the insertions are processed relative to the indexes of the state after all the deletions in the batch operation.

Изменения (Diff)

```
Invalid update: invalid number of items in section 0. The number of items contained in an existing section after the update (1) must be equal to the number of items contained in that section before the update (1), plus or minus the number of items inserted or deleted from that section (1 inserted, 0 deleted) and plus or minus the number of items moved into or out of that section (0 moved in, 0 moved out).
```

Изменения (Diff)

```swift
struct RowPosition {
    var index: Int
}

struct MovedRowPosition {
    var initialIndex: Int
    var finalIndex: Int
}

enum RowChange {
    case insert(RowPosition)
    case remove(RowPosition)
    case refresh(RowPosition)
    case move(MovedRowPosition)
}
```

```swift
class ModelHolder {
    private (set) var model: [CellAdapter]
    private (set) var changes: [RowChange] = []

    init(_ initialModel: [CellAdapter]) {
        self.model = initialModel
    }

    func resetChanges() {
        changes = []
    }

    func insert(_ item: CellAdapter, at position: Int) {
        model.insert(item, at: position)
        changes.append(.insert(RowPosition(index: position)))
    }

    func remove(at position: Int) {
        model.remove(at: position)
        changes.append(.remove(RowPosition(index: position)))
    }

    func update(by item: CellAdapter, at position: Int) {
        model[position] = item
        changes.append(.refresh(RowPosition(index: position)))
    }

    func move(from: Int, to: Int) {
        model.insert(model[from], at: to)
        model.remove(at: from)
        changes.append(.move(MovedRowPosition(initialIndex: from, finalIndex: to)))
    }
}
```

Изменения (Diff)

```
let model1: [CellAdapter] = [
    ImageCell(imageNamed: "logo"),
    TextCell(text: "Row"),
    TextCell(text: "Row")
]

let model2: [CellAdapter] = [
    TextCell(text: "Other row"),
    TextCell(text: "Row")
]

let changes: [RowChange] = calculateDiff(from: model1, to: model2)
```

Изменения (Diff)

```
protocol CellAdapter {
    func createView() -> UIView

    var diffIdentifier: AnyHashable { get }
    func equalTo(_ other: CellAdapter) -> Bool
}
```

Алгоритмы

- Myers Algorithm. DiffUtils в Android и Swift CollectionDifference
- Paul Heckel Algorithm. IGListKit. Простой, O(N), не всегда лучший путь

Изменения (Diff)

- UICollectionViewDiffableDataSource. iOS 13 можно в фоне
- https://github.com/Instagram/IGListKit
- https://github.com/RxSwiftCommunity/RxDataSources
- https://github.com/ra1028/DifferenceKit
- https://github.com/onmyway133/DeepDiff

Модель

- Diffable и Equtable для TableSection/TableItem

```
public class TableModel {
    public let readyLoadMore: Bool
    public let lastLoadFinishedWithError: Bool
    public let sections: [TableSection]
}

public struct TableSection {
    public var identifier: String
    public var items: [TableItem]
}

public struct TableItemActions {
    public var onTap: (() -> Void)?
    public var onDelete: (() -> Void)?
    public var onTryEdit: (() -> Bool)?
    public var onDisplay: ((UIView) -> Void)?
}

public struct TableItem {
    public var identifier: String
    public var cellHelper: CellHelper
    public var actions: TableItemActions
}
```

```
extension TableItem: Diffable {
    public var diffIdentifier: AnyHashable {
        return identifier
    }
}

extension TableItem: Equatable {
    public static func ==(lhs: TableItem, rhs: TableItem) -> Bool {
        return lhs.identifier == rhs.identifier && lhs.cellHelper.isEqual(to: rhs.cellHelper)
    }
}
```

Модель

```
public class CollectionViewCellAttributes {
    var setupInitialCollectionAttributesForAppearing: ((_ attributes: UICollectionViewLayoutAttributes) -> Void)?
    var setupFinalCollectionAttributesForDisappearing: ((_ attributes: UICollectionViewLayoutAttributes) -> Void)?
    var setupDefaultCollectionAttributes: ((_ attributes: UICollectionViewLayoutAttributes) -> Void)?
}

public protocol CellHelper: class {
    /// Create view. It is assumed that `createCellView` can return same view for different calls
    func createCellView(width: CGFloat) -> UIView

    /// May be called on background. Should calcualte cell size and set `precalculatedLayoutSize` property
    func precalculateLayout(indexPath: IndexPath, width: CGFloat)

    /// Precalculated size of cell. Calculated by `precalculateLayout` call
    var precalculatedLayoutSize: CGSize? { get }

    /// Used by auto-dffing algorithm to understand necessity of cell update
    func isEqual(to cellHelper: CellHelper) -> Bool

    /// On refresh old cell replaced by new. This method allow leave old one
    /// May be used in cases when cell caches view returned by `createCellView`
    func updateBy(_ cellHelper: CellHelper) -> Bool

    /// `UICollectionViewLayoutAttributes` for customization cell position. Commonly used to set z-index
    var collectionViewCellAttributes: CollectionViewCellAttributes? { get }
}
```

AutoDiffViewController

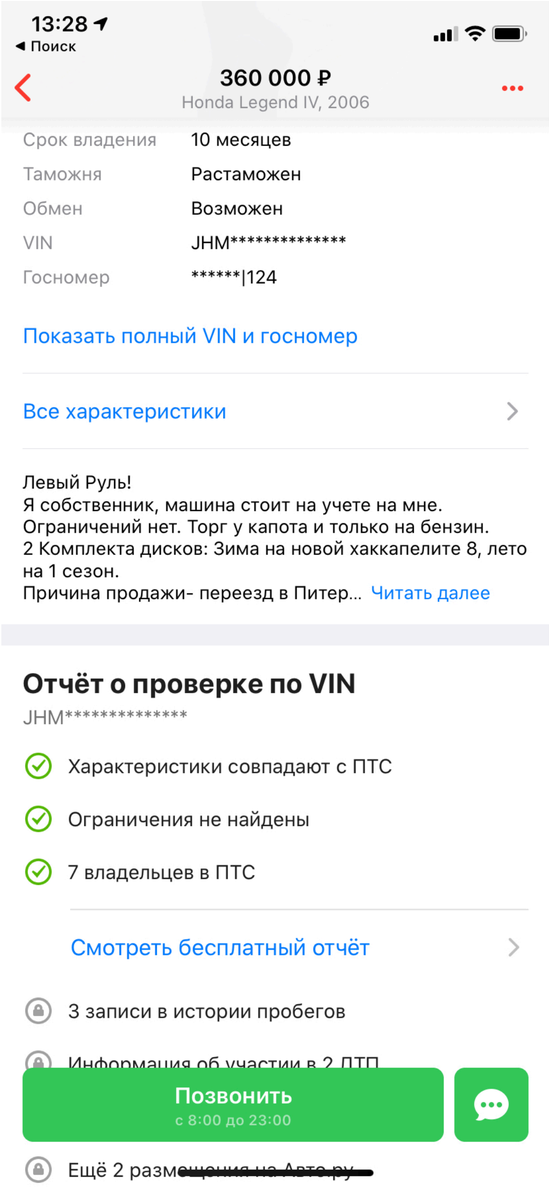
```
public class AutoDiffViewController: UIViewController {
    public var backgroundView: UIView?
    public var scrollView: UIScrollView

    public var didRequestLoadMore: ((_ lastCellVisible: Bool) -> Void)?
    public var willBeginDragging: (() -> Void)?

    public func updateTableModel(with newModel: TableModel)
    public func updateTableModel(with newModelCreator: @escaping (() -> TableModel)

    public func tableItemWithIdentifier(_ identifier: String) -> TableItem?
    public func scrollToItemWithIdentifier(_ itemId: String,
                                           scrollPosition: UICollectionView.ScrollPosition,
                                           animated: Bool)
    }
```

AutoDiffViewController

```
public func updateTableModel(with newModelCreator: @escaping (() -> TableModel),
                             completion: (() -> Void)? = nil,
                             animated: Bool) {
    operationQueue.cancelAllOperations()

    operationQueue.addOperation({ () -> Void in
        self.semaphore.wait()

        let newModel = newModelCreator()
        let transform = ModelTranformGenerator(model: self.tableModel).calculateTransformTo(newModel)

        // Calculates layout for new or updated items.
        for indexPath in transform.newItemsIndexes {
            let tableItem = transform.updatedModel.sections[indexPath.section][indexPath.row]
            tableItem.cellHelper.precalculateLayout(indexPath: indexPath, width: cellWidth)
        }

        let applyCompletion: (() -> Void) = {
            self.semaphore.signal()
            completion?()
        }

        DispatchQueue.main.sync(execute: { () -> Void in
            guard !transform.changes.isEmpty else {
                applyCompletion()
                return
            }

            self.tableModel = transform.updatedModel

            if let collectionView = self.collectionView {
                self.applyChanges(transform.changes, for: collectionView, completion: applyCompletion, animated: animated)
            } else {
                applyCompletion()
            }
        })
    })
}
```

Сепараторы

- Много динамического контента
- Разные блоки требуют разный сепаратор



TableModelBuilder

```
final class SimpleTableModelBuilder: BaseTableModelBuilder {
    func addHeader() {
        let cellHelper = LayoutCellHelper<String>(
            model: "",
            layoutCreator: ({ model in
                let layout = ...
                return layout
            }))

        addCellHelper(identifier: "header", cellHelper: cellHelper)
    }
}

let modelBuilder = SimpleTableModelBuilder()
modelBuilder.addHeader()
modelBuilder.addSeparatorItem(.medium)
modelBuilder.addTextButton(title: "OK", onTap: ({
    print("Ok")
}))
modelBuilder.addSeparatorItem(.small)
modelBuilder.addTextButton(title: "Отмена", onTap: ({
    print("Cancel")
}))


autoDiffViewController.updateTableModel(with: TableModel(readyLoadMore: false, lastLoadFinishedWithError: false, sections: modelBuilder.build()))
```

TableModelBuilder

```
public func cleanupSeparators() {
    var cleanedItems: [TableItem] = []
    let smallSeparatorIdPrefix = "separator_\(SeparatorType.small)"

    var prevItem: TableItem?
    for item in items {
        if let prevItem = prevItem {
            if item.identifier.starts(with: "separator_") {
                if prevItem.identifier.starts(with: smallSeparatorIdPrefix) {
                    cleanedItems.removeLast()
                } else if prevItem.identifier.starts(with: "separator_") {
                    continue
                }
            }
        }

        cleanedItems.append(item)
        prevItem = item
    }

    if let lastItem = cleanedItems.last {
        if lastItem.identifier.starts(with: "separator_") {
            cleanedItems.removeLast()
        }
    }

    self.items = cleanedItems
}
```

Все вместе

```
class SimpleViewController: UIViewController {
    private let autoDiffViewController = AutoDiffViewController(identifier: "SimpleViewController")

    override func viewDidLoad() {
        super.viewDidLoad()
        setupCollectionView()
    }

    private func setupCollectionView() {
        addChild(autoDiffViewController)
        view.addSubview(autoDiffViewController.view)
        autoDiffViewController.view.pinEdgesToSuperView()
        autoDiffViewController.didMove(toParent: self)

        autoDiffViewController.view.backgroundColor = .white
    }

    func update(by model: Model) {
        let modelBuilder = SimpleTableModelBuilder()

        modelBuilder.addHeader(title: model.title)
        modelBuilder.addSeparatorItem(.medium)
        modelBuilder.addTextButton(title: "OK", onTap: ({
            model.onOkTap?()
        }))
        modelBuilder.addSeparatorItem(.small)
        modelBuilder.addTextButton(title: "Отмена", onTap: ({ [weak self] in
            model.onCancelTap?()
        }))

        autoDiffViewController.updateTableModel(with: TableModel(readyLoadMore: false, lastLoadFinishedWithError: false, sections: modelBuilder.build()))
    }
}
```

Все вместе

```
class Interactor {
    var model: Observable<Model> { get }
}

class SimpleViewController: UIViewController {
    func update(by model: Model)
}

class Router {
    static func build() -> Router {
        let interactor = Interactor()
        let view = SimpleViewController()

        Observable
            .combineLatest(interactor.model, view.viewDidPreparedSubject)
            .observeOn(MainScheduler.instance)
            .subscribe(onNext: { [weak self] model, _ in
                view.update(by: model)
            })
            .disposed(by: disposeBag)

        return Router(view: view, interactor: interactor)
    }
}
```

Цена абстракции

- Время рантайма
- Время компилятора
- Время человека работающего с кодом (поддержка)

SwiftUI

- Нативный вид
- Preview в Xcode
- Черная тема из коробки

SwiftUI

- Поддержка на уровне компилятора
- Быстрый Diff (дерево элементов создается при компиляции)

Конец

- Будущее за декларативным лейаутом
- Для ускорения разработки нужны абстракции
- Нужно помнить, что бесплатных абстракций не бывает