



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение эвм и информационные технологии»

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ
НА ТЕМУ:**

***«Классификация методов верстки
элементов интерфейса в разработке
мобильных приложений под iOS»***

Студент ИУ7-54Б

_____ Егорова П. А.

Руководитель

_____ Барышникова М. Ю.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-7
(Индекс)

И. В. Рудаков
(И.О.Фамилия)

«16» сентября 2022 г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме

**«Классификация методов верстки элементов интерфейса
в разработке мобильных приложений под iOS»**

Студент группы **ИУ7-54Б**

Егорова Полина Александровна

Направленность НИР

учебная

Источник тематики

НИР кафедры

График выполнения НИР: 25% к 6 нед., 50% к 9 нед., 75% к 12 нед., 100% к 15 нед.

Техническое задание

Определить критерии, по которым осуществляется анализ эффективности использования методов верстки элементов интерфейса. Провести обзор существующих методов верстки мобильных приложений под iOS и выделить наиболее популярные. Провести классификацию методов на основе выделенных критериев.

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на **12-20** листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т. п.)

Презентация на **6-10** слайдах.

Дата выдачи задания «16» сентября 2022 г.

Руководитель НИР

(Подпись, дата)

М. Ю. Барышникова
(И.О.Фамилия)

Студент

(Подпись, дата)

П. А. Егорова
(И.О.Фамилия)

Оглавление

Введение	3
1 Обзор предметной области	4
1.1 Основные понятия	4
1.2 UIKit	5
2 Методы верстки	7
2.1 Ручная верстка	7
2.1.1 Frame	7
2.2 Автоматическая верстка	9
2.2.1 Auto Layout программно	9
2.2.2 Interface Builder	11
3 Результат анализа	14
Заключение	18
Список использованных источников	19

Введение

iOS [1] – мобильная операционная система, разработанная и выпущенная компанией Apple [2] в 2007 году: инновационными в первых iPhone [3] стали возможность просмотра видео на YouTube [4] и поиск нескольких интернет-игр. За 15 лет система претерпела массу изменений: были разработаны новые технологии, создан язык программирования Swift [5], который пришел на смену первоначально использовавшемуся Objective-C [6], были выпущены гайдлайны [7], и всё это – чтобы вырасти в многофункциональную платформу и превратить iPhone в некий «ПК в кармане». Это позволило Apple занять большое количество пользователей по всему миру, а, как следствие, и IT-специалистов, желающих создавать мобильные продукты под iOS.

Неотъемлемой частью iOS-разработки является создание интерфейса мобильного приложения. Каждый iOS-разработчик сталкивается с проблемой выбора метода разметки экранов продукта: Apple предоставляет несколько вариантов верстки. Возникает вопрос: по каким критериям осуществлять выбор и какая технология является в конкретном случае наиболее подходящей?

Целью данной работы является классификация методов верстки элементов интерфейса в разработке мобильных приложений под iOS. Для достижения поставленной цели необходимо решить следующие задачи:

- определить критерии, по которым осуществляется анализ эффективности использования методов верстки элементов интерфейса;
- провести обзор существующих методов верстки мобильных приложений под iOS и выделить наиболее популярные;
- классифицировать методы на основе выделенных критериев.

1 Обзор предметной области

Чтобы достичь намеченных целей, необходимо понять, что из себя представляет экран мобильного приложения, а также из каких элементов может состоять его интерфейс.

1.1 Основные понятия

В iOS существует различие между координатами, указываемыми в коде, и пикселями устройства [8] – наименьшими дискретными элементами двумерного цифрового изображения. Для большинства задач фактический размер точек [9] не имеет значения, их цель – обеспечить согласованный масштаб, который может использоваться в коде для указания размера и положения представлений и отображаемого содержимого. Например, если пиксели в два раза меньше изначальной высоты или ширины, можно использовать квадрат 2×2 пикселя для каждой точки (это называется масштаб @2x) – рисунок 1.1. Таким образом, измерение в точках позволяет корректно масштабировать изображение на экранах с высоким разрешением [7].

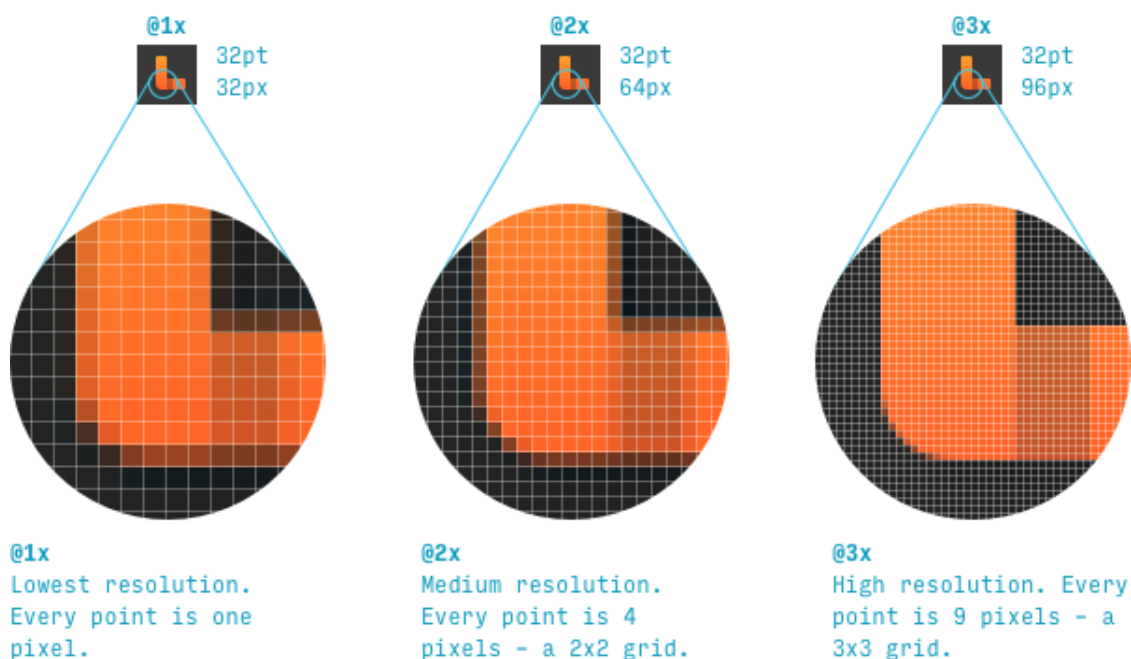


Рис. 1.1: Точки и пиксели

Устройства, работающие на операционной системе iOS, имеют различное разрешение экрана и соотношение сторон. Так, например, iPhone X имеет разрешение 1125 x 2436 пикселей и коэффициент масштабирования в точки, равный 3.0 (то есть 375 x 812 точек), а iPhone 6 – 750 x 1334 пикселей и коэффициент 2.0 соответственно (то есть 375 x 667) [10]. А также каждое устройство имеет горизонтальную и вертикальную ориентацию. Данные характеристики стоит учитывать в разработке, дабы создавать приложения, интерфейс которых адаптируется к устройствам разного размера, а также любой их ориентации.

1.2 UIKit

UIKit [11] – библиотека, предоставляющая архитектуру окон и представлений для реализации пользовательского интерфейса, включая компоненты, которые можно использовать для построения базовой инфраструктуры приложения. Альтернативой выступает фреймворк SwiftUI [12], предоставляющий не меньше возможностей, однако, будучи представленной в 2016 году, библиотека не смогла стать столь популярной в разработке. Поэтому целесообразным будет рассмотреть методы верстки UI-элементов, предоставляемых UIKit.

Рассмотрим базовые UI-элементы, предлагаемые фреймворком UIKit.

UIView (или представление) [13] – это фундаментальный блок пользовательского интерфейса приложения, а UIView-класс определяет поведение, общее для всех представлений. Объекты этого класса отображают содержимое в пределах своих границ и обрабатывают любые взаимодействия с этим содержимым. Для отображения надписей, изображений, кнопок и других элементов интерфейса, обычно встречающихся в приложениях, используют не определяемые самостоятельно подклассы view, а предоставляемые платформой UIKit.

UIViewController (или контроллер) [14] – объект, который управляет иерархией представлений приложения. Основные обязанности контроллера включают следующее:

- обновление содержимого представлений;

- реагирование на взаимодействие пользователя с представлениями;
- изменение размеров представлений и управление макетом общего интерфейса;
- координация с другими объектами, включая другие контроллеры представления.

Итак, экран в мобильной разработке представляет `UIViewController`, являющийся контейнером для других `UIView`. Труд команды разработки интерфейса мобильного приложения сводится к задаче корректного отображения элементов на экране – расположения `UIView` на `UIViewController`. Чтобы работа была эффективной, целесообразно каждому члену команды дать индивидуальную подзадачу – таким образом используемая технология верстки должна предполагать возможность работы нескольких участников над одним проектом и минимизировать количество ошибок, возникающих при изменении параметров UI-элементов.

На основе рассмотренных теоретических сведений можно выделить критерии, по которым необходимо провести анализ эффективности использования методов верстки:

- возможность создания интерфейса, масштабируемого для устройств разного размера и двух ориентаций экрана, и относительная сложность его создания;
- сложность командной разработки интерфейса; (сложность?)
- сложность внесения изменений в интерфейс;
- возможность обработки всех параметров UI-элемента;
- скорость работы метода.

2 Методы верстки

Под понятием разметки подразумевается расчет необходимых координат и размеров UI-элементов. Существует два основных подхода к созданию пользовательского интерфейса: можно программно компоновать элементы, задавая координаты и размеры каждого в коде, или же использовать автоматическую компоновку.

2.1 Ручная верстка

2.1.1 Frame

Frame [15] – свойство view, описывающее прямоугольник, определяющий местоположение и размер представления в системе координат его родительского view [16].

View, изображенную на рисунке 2.1, можно описать кодом, представленным в листинге 2.1.

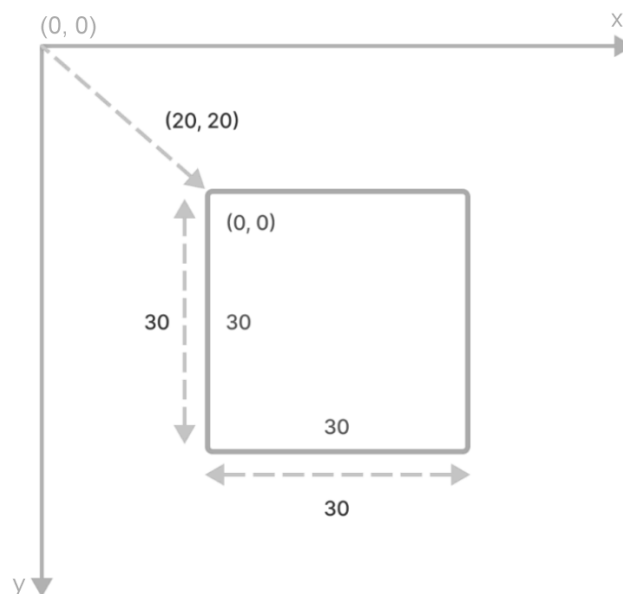


Рис. 2.1: Frame

```
1  var view = UIView()  
2  view.frame = CGRect(x: 20, y: 20, width: 30, height: 30)
```

Листинг 2.1: Задание свойства frame для UIView

Ручная верстка предполагает задание кодом координат и размеров каждого элемента экрана. Указание координат идет относительно левого верхнего угла экрана – именно там располагается точка (0, 0). При желании запустить будущее приложение на двух устройствах, размеры экрана которых различны, результат также будет разниться. То же будет происходить и при попытке сравнения экранов горизонтальной и вертикальной ориентации. Таким образом, возникает необходимость вручную обрабатывать каждый из возможных размеров дисплея в двух вариациях ориентации. На данный момент существует 10 различных размеров экрана iPhone и 7 – iPad. Нетрудно представить, во сколько раз увеличивается количество строк кода и повышается трудоемкость задачи при обработке каждого случая.

Однако данный подход позволяет разбить задачу создания интерфейса на несколько подзадач, поскольку в пределах одной `Superview` разработчик взаимодействует лишь с этим родительским представлением и конкретной `view`. Внесение изменений в написанный ранее код также не составит большого труда, так как параметры, задаваемые в свойстве `frame`, независимы друг от друга, и их значения являются константами.

При использовании данного метода разметки разработчик получает возможность создания любого кастомного элемента, поскольку имеет доступ к обработке каждого визуального параметра UI-элемента. Так, например, можно задавать радиус угла `view`, цвет и прозрачность элемента, что представлено в листинге 2.2.

```
1  var view = UIView()  
2  view.frame = CGRect(x: 20, y: 20, width: 30, height: 30)  
3  view.layer.cornerRadius = 2.0  
4  view.backgroundColor = UIColor.red.withAlphaComponent(0.5)
```

Листинг 2.2: Задание параметров `view`

Для того, чтобы оценить время работы метода, необходимо упомянуть, что любые модификации UI-элементов производятся на главном потоке многопоточного приложения – `main thread` [17]. Поскольку параметрами свойства `frame` выступают константы, а, соответственно, никакие вычисления не производятся, скорость работы метода будет соответствовать скорости размещения элементов на экране.

2.2 Автоматическая верстка

Существует альтернативный способ расчета необходимых координат и размеров.

Автоматическая компоновка (или Auto Layout) [18] динамически вычисляет размер и положение всех видов в иерархии на основе ограничений, наложенных на эти view. Например, существует возможность ограничить кнопку так, чтобы она была центрирована по горизонтали в соответствии с видом изображения и ее верхний край всегда оставался на 8 точек ниже нижнего края изображения.

Правила (или constraints), задаваемые Auto Layout, представляют собой обычное линейное уравнение вида

$$y = a \cdot x + b, \quad (2.1)$$

где y – необходимая координата или размер; a – произвольный множитель; x – параметр, от которого зависит итоговый результат; b – константа. Для того чтобы корректно посчитать координаты и размер элемента, необходимо минимум 3 правила, которые сложатся в систему уравнений, результатом решения которой будут координаты и размер элемента.

Существует два основных способа задать ограничения для автоматической компоновки: программный, когда каждое ограничение представлено строкой кода, и автоматический, когда ограничения заданы с помощью инструмента работы с пользовательским интерфейсом Interface Builder и Storyboard [19].

2.2.1 Auto Layout программно

Автоматическая компоновка предоставляет возможность создавать масштабируемый для устройств разных размеров и ориентации экрана интерфейс, поскольку значения ограничений вычисляются динамически при внутренних или внешних изменениях.

К внешним относятся изменение размера окна и вращение устройства. К внутренним – изменение содержимого, отображаемого приложением.

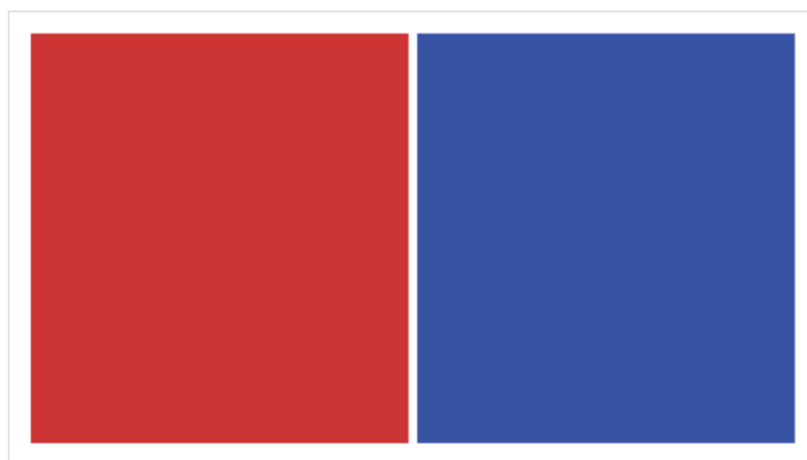
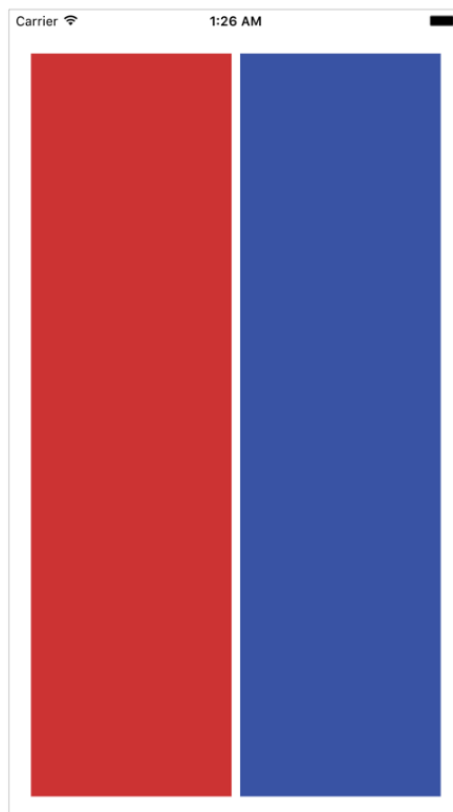


Рис. 2.2: UIView, скомпонованные с помощью Auto Layout

Например, для расположения на экране двух UIView, находящихся справа и слева друг от друга, как представлено на рисунке 2.2, можно использовать ограничения, представленные в листинге 2.3.

```

1 // Vertical Constraints
2 Red.top = 1.0 * Superview.top + 20.0
3 Superview.bottom = 1.0 * Red.bottom + 20.0
4 Blue.top = 1.0 * Superview.top + 20.0
5 Superview.bottom = 1.0 * Blue.bottom + 20.0
6
7 // Horizontal Constraints
8 Red.leading = 1.0 * Superview.leading + 20.0
9 Blue.leading = 1.0 * Red.trailing + 8.0
10 Superview.trailing = 1.0 * Blue.trailing + 20.0
11 Red.width = 1.0 * Blue.width + 0.0

```

Листинг 2.3: Задание параметров view

Строки кода позволяют проследить логику задания ограничений, поэтому внесение изменений в уже описанный интерфейс не является сложной задачей: необходимо лишь понимание того, какие модификации претерпевает UI-элемент на каждом этапе наложения на него ограничений. Из чего вытекает возможность и командной разработки с использованием метода Auto Layout.

Также сохраняется возможность создания любого кастомного элемента, поскольку доступ к обработке каждого визуального параметра UI-элемента все еще доступен при программной верстке.

Основной проблемой при автоматическом расчете разметки является скорость работы. При увеличении количества правил усложняется система уравнений, которая должна быть решена чтобы были получены все координаты и размеры. Так как этот механизм является закрытой системой, то программист не может повлиять на скорость работы алгоритмов. К тому же все расчёты производятся в главном потоке приложения: например, при наличии в интерфейсе списка со сложными элементами и быстром его прокручивании расчет всех координат и размеров будет занимать очень много времени, что в итоге скажется на плавности прокручивания.

2.2.2 Interface Builder

Interface Builder [20] представляет собой инструмент для создания пользовательского интерфейса: сцена, на которую можно перетаскивать view

из библиотеки видов – рисунок 2.3. Программист имеет возможность настраивать ограничения, выравнивать и закреплять каждый UI-элемент с помощью всплывающих окон.

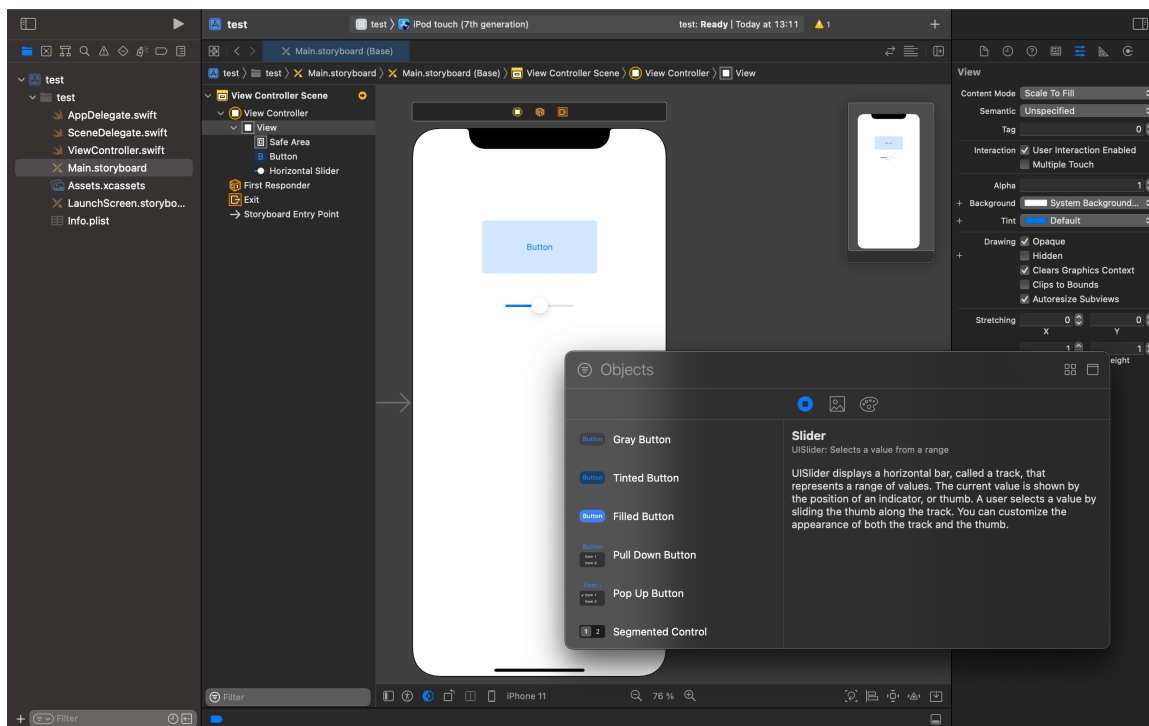


Рис. 2.3: Interface Builder

Таким образом, интерфейс может быть создан без написания кода. Однако если в одних случаях это является бесспорным преимуществом, в других – источником проблем или грозит отсутствием возможности реализации всех задумок.

Этот метод верстки имеет преимущество в разработке статического интерфейса небольшого приложения: при его создании существенно сократится время разработки, поскольку больше не нужно каждое ограничение для UI-элемента описывать строкой кода. Также имеется возможность легко изменять параметры какого-либо конкретного элемента: например, длину лэйбла. Однако Interface Builder не предоставляет возможности в динамике обрабатывать изменения интерфейса, а также при большом количестве частей интерфейса сцена разрастется настолько, что перестает быть наглядной. Из чего вытекает, что внесение изменений в уже готовые проекты является сложной задачей, поскольку разработчик не имеет наглядно представленных правил ограничения UI. Следствием является и сложность командной разработки: сцена Interface Builder неделима, невоз-

можно по отдельности обработать каждый UIView, закрепляемый на ней.

Невозможно обработать и каждый параметр UIView или слоя представления, так как рассматриваемый инструмент предоставляет лишь ограниченный список настраиваемых параметров. Так, например, разработчик не имеет возможности задать view, края которой скруглены, из Interface Builder.

Затраты по времени метода зависят от сложности сцены: чем объемнее сцена, тем больше времени требует процессор для обработки графических элементов, расположенных на ней, и преобразования в код.

3 Результат анализа

Таблица 3.1: Методы верстки

Критерий	Метод	Заключение
Масштабируемость интерфейса	Frame	Ручная верстка предполагает задание констант в качестве координат и размеров элемента экрана. На устройствах различного размера и при разной ориентации экрана верстка будет различаться, поэтому появляется необходимость обрабатывать каждую вариацию размеров экрана отдельно.
	Autolayout	Значения ограничений вычисляются динамически при внутренних или внешних изменениях. При корректном составлении линейных уравнений, задающих ограничения, интерфейс будет масштабирован на все виды устройств.
	Interface Builder	Значения ограничений вычисляются динамически при внутренних или внешних изменениях. При корректном составлении линейных уравнений, задающих ограничения, интерфейс будет масштабирован на все виды устройств.

Таблица 3.2: Методы верстки – продолжение

Критерий	Метод	Заключение
Возможность командной разработки	Frame	Координаты view будут заданы константными значениями в коде и координаты одного представления не зависят от координат другого, возникает возможность разделить задачу верстки экрана на подзадачи. Командная разработка в таком случае возможна, причем можно выделить столько подзадач, сколько view будет принадлежать родительскому view.
	Autolayout	В пределах одного родительского view координаты и размеры дочерних могут находиться в зависимости друг от друга, поскольку размещение элементов на экране осуществляется посредством правил, задаваемых линейными уравнениями. Командная разработка возможна, но не в пределах одного родительского представления.
	Interface Builder	UI-элементы прикрепляются к view в процессе работы с инструментом создания интерфейсов. Наглядно увидеть правила задания ограничений не представляется возможным. Идет неделимая работа над созданием интерфейса экрана: отсутствует возможность извне добавить элементы на subview. Командная разработка возможная лишь в пределах одного экрана.

Таблица 3.3: Методы верстки – продолжение 2

Критерий	Метод	Заключение
Возможность внесения изменений	Frame	Координаты view заданы константными значениями в коде и координаты одного представления не зависят от координат другого. Внесение изменений сводится к изменению констант, поэтому не влечет за собой необходимость изменять координаты и размеры других view.
	Autolayout	Координаты и размеры view могут находиться в зависимости друг от друга, поскольку размещение элементов на экране осуществляется посредством правил, задаваемых линейными уравнениями. Изменение одного правила может повлечь за собой необходимость внесения изменений во все ограничения экрана, соответственно, сложность внесения изменений зависит от сложности сцены.
	Interface Builder	UI-элементы прикрепляются к view в процессе работы с инструментом создания интерфейсов. Наглядно увидеть правила задания ограничений не представляется возможным, в соответствии с чем внесение изменений в готовый проект возможно, однако является задачей, сложность которой зависит от объемности сцены: изменение одного правила может повлечь за собой необходимость редактирования всех ограничений экрана.

Таблица 3.4: Методы верстки – продолжение 3

Критерий	Метод	Заключение
Возможность обработки параметров UI-элементов	Frame	Верстка кодом: имеется возможность доступа к обработке всех параметров UI-элемента.
	Autolayout	Верстка кодом: имеется возможность доступа к обработке всех параметров UI-элемента.
	Interface Builder	Верстка через инструмент создания интерфейса: ограниченный набор параметров обработки UI-элемента.
Скорость работы метода	Frame	Параметрами свойства frame выступают константы, поэтому скорость работы метода будет соответствовать скорости размещения элементов на экране.
	Autolayout	Скорость работы напрямую зависит от сложности сцены: чем больше объектов содержит сцена, тем больше линейных уравнений необходимо решить для размещения элементов на экране.
	Interface Builder	Чем больше объектов содержит сцена, тем больше линейных уравнений необходимо решить для размещения элементов на экране и тем больше времени требует процессор для обработки графических элементов, расположенных на ней, и преобразования в код – скорость работы метода зависит от сложности сцены.

Заключение

В ходе выполнения научно-исследовательской работы были выполнены следующие задачи:

- определено понятие верстки под iOS;
- выявлены критерии классификации методов верстки;
- выделены наиболее популярные методы верстки;
- проведена классификация на основе выделенных критериев.

Были классифицированы такие методы верстки, как компоновка с помощью frame, Auto Layout программно и с помощью Interface Builder. Выявлено, что при желании создания интерфейса, масштабируемого для устройств разного размера, не рационально использовать Interface Builder – его преимущества быстроты создания и изменения конкретно взятого элемента проявляются при создании статического экрана, содержащего не много UI-элементов. Наиболее гибким для командной разработки и внесения изменений в написанный ранее код является верстка с помощью frame, хоть она и уступает Auto Layout в количестве строк кода: читаемо, но объемно. А также несомненное преимущество перед инструментом создания пользовательского интерфейса имеет программная компоновка в вопросе создания кастомных элементов – Interface Builder предоставляет ограниченный набор параметров обработки UI. В отношении скорости работы методов можно сказать однозначно: каждый из них зависит от объема информации проектируемого экрана и сложности каждого его элемента. Однако при равенстве этих параметров автоматическая компоновка будет уступать ручной по времени работы, поскольку потребуется еще и время на решение систем уравнений, используемых в Auto Layout для задания координат и размеров элементов.

Все методы имеют преимущества и недостатки, поэтому программисту полезно иметь в своем арсенале каждый, уметь анализировать плюсы и минусы способа в конкретном случае и уметь выбирать оптимальный или же сочетать несколько средств создания интерфейса.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] iOS | Apple Developer Documentation / [Электронный ресурс]. Режим доступа: <https://developer.apple.com/documentation/packagedescription/platform/ios/>
- [2] Apple (Россия) – Официальный сайт / [Электронный ресурс]. Режим доступа: <https://www.apple.com/ru/>
- [3] iPhone – Apple (RU) – Официальный сайт / [Электронный ресурс]. Режим доступа: <https://www.apple.com/ru/iphone/>
- [4] YouTube / [Электронный ресурс]. Режим доступа: <https://www.youtube.com/>
- [5] The Swift Programming Language (Swift 5.7) / [Электронный ресурс]. Режим доступа: <https://www.apple.com/swift/>
- [6] About Objective-C / [Электронный ресурс]. Режим доступа: <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html>
- [7] Human Interface Guidelines – Design – Apple Developer / [Электронный ресурс]. Режим доступа: <https://developer.apple.com/design/human-interface-guidelines/guidelines/overview/>
- [8] Pixel | Definition and Meaning / [Электронный ресурс]. Режим доступа: <https://www.merriam-webster.com/dictionary/pixel#h1>
- [9] Point | Apple Developer Documentation / [Электронный ресурс]. Режим доступа: <https://developer.apple.com/documentation/uikit/uiaccessibilitylocationdescriptor/2890956-point?changes>
- [10] Displays | Apple Developer Documentation / [Электронный ресурс]. Режим доступа: <https://developer.apple.com/library/archive/documentation/DeviceInformation/Reference/iOSDeviceCompatibility/Displays/Displays.html>

- [11] UIKit | Apple Developer Documentation / [Электронный ресурс]. Режим доступа: <https://developer.apple.com/documentation/uikit/uikit>
- [12] SwiftUI | Apple Developer Documentation / [Электронный ресурс]. Режим доступа: <https://developer.apple.com/documentation/swiftui>
- [13] UIView | Apple Developer Documentation / [Электронный ресурс]. Режим доступа: <https://developer.apple.com/documentation/uikit/UIView>
- [14] UIViewController | Apple Developer Documentation / [Электронный ресурс]. Режим доступа: <https://developer.apple.com/documentation/uikit/uiviewcontroller>
- [15] Frame | Apple Developer Documentation / [Электронный ресурс]. Режим доступа: <https://developer.apple.com/documentation/uikit/UIView/1622621-frame>
- [16] Superview | Apple Developer Documentation / [Электронный ресурс]. Режим доступа: <https://developer.apple.com/documentation/appkit/nview/1483737-superview>
- [17] Thread | Apple Developer Documentation / [Электронный ресурс]. Режим доступа: <https://developer.apple.com/documentation/foundation/thread?language=swift>
- [18] Auto Layout Guide: Understanding Auto Layout / [Электронный ресурс]. Режим доступа: <https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/AutolayoutPG/>
- [19] UIStoryboard | Apple Developer Documentation / [Электронный ресурс]. Режим доступа: <https://developer.apple.com/documentation/uikit/uistoryboard?language=objc>
- [20] Auto Layout Guide: Working with Constraints in Interface Builder / [Электронный ресурс]. Режим доступа: <https://developer.apple.com/documentation/uikit/uistoryboard?language=objc>

com/library/archive/documentation/UserExperience/Conceptual/
AutolayoutPG/WorkingwithConstraintsinInterfaceBuidler.html#
//apple_ref/doc/uid/TP40010853-CH10-SW1

- [21] Peculiarities of layout user interface elements on the iOS and MacOS platforms./ И.М. Булыга.// UNIVERSUM: ТЕХНИЧЕСКИЕ НАУКИ – 2022 – 2(95) – С. 45–48/// [Электронный ресурс]. Режим доступа: [https://7universum.com/pdf/tech/2\(95\)/2\(95_1\).pdf#page=45](https://7universum.com/pdf/tech/2(95)/2(95_1).pdf#page=45)