ПРО

ДОСТУПНОСТЬ iOS

Михаил Рубанов

Что такое доступность?

Доступность приложения — это адаптация интерфейса для разных групп людей. Если человек хуже видит (или совсем потерял зрение), не слышит, не двигается, но при этом полноценно пользуется приложением, то оно доступно.

— Не знаете английский, но хочется посмотреть новый сериал на Netflix? Есть субтитры, а часто и полноценный перевод на русский.
— Не можете перехватить телефон одной рукой, чтобы нажать кнопку назад, потому что в другой руке держите ребенка? Можно свайпнуть от края экрана.
— Перед сном читаете книжку? Темная тема не будет слепить.

Это некая «стандартная» доступность для всех людей. Вроде можно прожить и без этих фич, но с ними удобнее. Если каждый из этих примеров усилить, то он поможет людям с проблемами со здоровьем, для кого эти «фичи» становятся жизненно необходимыми.

— Пожилые люди могут пользоваться телефоном без очков, увеличив шрифт, или включив экранный зум.
— Аудиодескрипция вместо субтитров расскажет незрячему, что происходит на экране, а вернуться на экран назад поможет специальный жест в VoiceOver.
— Темная тема поможет людям со светобоязнью.
— Вибрация на часах помогает узнавать о звонке тем, кто не слышит.

А для тех, кто совсем неподвижен, одно лишь движение щеки позволит контактировать с миром: отдавать телефону команды «выбери» и «следующий элемент». Возможно, это поможет кому-то написать книгу об устройстве вселенной.

Я расскажу о самых сложных нарушениях, в которых даже не сразу понятно, как вообще можно пользоваться телефоном с сенсорным экраном. Может показаться, что незрячему удобней пользоваться телефоном с кнопками, но на самом деле мир изменился после выхода iPhone 3GS. Подробней про это можно прочитать в статье «Доступность на iOS началась с 36 секунд»

Зачем заниматься доступностью?

Я, как мобильный разработчик, делаю интерфейсы для людей: удобные, красивые и простые. В этой работе много знаний: и законы взаимодействий с интерфейсами, и восприятие информации, и влияние на эмоции. При этом вся индустрия пытается максимально охватить пользователей — многие приложения работают на всю страну или весь мир.

Какие свойства у хорошего интерфейса? Он понятный, отзывчивый, красивый, быстрый и… доступный для всех.

Можно ли назвать качественным недоступный интерфейс? Только в редких случаях. Например, если вашим продуктом пользуется узкий круг людей и вы точно уверены, что они здоровы «как космонавты».

Для популярных приложений картина другая: из миллионов пользователей очень много людей с нарушениями. Иногда ограничения небольшие, иногда серьезные, но все меняет образ жизни. Раньше такие особенности могли создавать непреодолимые препятствия для жизни человека, но сегодня она может быть комфортной, насыщенной и интересной, благодаря технологиям, что есть у каждого человека в кармане.

Самое крутое, что для этого не нужно делать очень много. iPhone уже содержит все нужные инструменты: синтезатор речи, мультитач-дисплей, трекер головы и несколько модулей iOS, обеспечивающих доступность. Вам нужно им только немного помочь.

Почему это полезно для разработчиков?

Адаптация приложения для незрячих помогает понять что такое «интерфейс». Она показывает сложность мира и путей взаимодействия: люди не только жмут на экран, но и могут управлять с клавиатуры, голосом, жестами. Качественный интерфейс всё это обеспечивает.

Превращая графический интерфейс в звуковой лучше понимаешь, как работает наше восприятие. Мы быстро считываем ментальную модель с экрана, а дальше уже работает наше понимание этой модели. Тот, кто умеет работать с информацией на таком уровне может раньше найти проблемы и в графическом дизайне. Глубокое понимание взаимодействия учит делать лучше, толкает вас в правильном направлении и, в итоге, выигрывают все пользователи.

Почему это важно для компаний?

Если продуктом пользуются миллионы, то среди ее пользователей тысячи людей с ограничениями. Для них сервис может быть жизненно важным, но недоступным. Приложения влияют на всю жизнь человека и на то, что он в ней может сделать.

Не всё измеряется цифрами и прибылью. Любая крупная компания несёт ответственность за свое влияние на людей: она может помогать, мешать, создавать праздник, банить политиков. Всё IT сейчас отвечает за то, как мы живем.

Адаптируя приложения и сайты, мы даем незрячим людям способ взаимодействовать с миром: они так же заказывают еду на доставку, ездят на такси, оплачивают счета, общаются, читают новости и соцсети, ходят на работу. Не адаптируя приложения, мы забираем кусочек их жизни и функционала.

Приложения и сервис незрячим даже нужнее, чем всем остальным. Представьте, насколько сложно сходить в магазин: идёте с тростью до магазина, выбираете продукты (как?), возвращаетесь обратно. Можно в тысячу раз проще — заказать доставку через приложение. Если приложение адаптировано, то легко узнать состав, ассортимент, указать адрес, оплатить и купить всё, что хотите.

При этом, доступность — это не фича, её нельзя измерить сроком, сделать один раз и никогда больше не возвращаться. Доступность в головах: это набор правил, которые разработчики и дизайнеры должны учитывать, когда создают новый интерфейс.

Доступность — это навык и ему можно научиться

Поддержка доступности — хорошая метрика культуры компании. Все крупные иностранные приложения Эпла, Гугла, Фейсбука адаптированы и работают очень хорошо. Почти все крупные российские — не адаптированы совсем или содержат критичные ошибки.

Создавайте приложения для людей. Для всех.

Приступим?

Версия для незрячих? Нет

В вебе часто можно встретить отдельную версию для незрячих или слабовидящих, в мобиле я такого не встречал. Видимо, никому в голову не приходит написать второе приложение для особого случая, но тему обсудим.

Допустим, мы решили написать приложение специально для незрячего. Первое, что надо понять: а как такую версию писать?

Технически — это адаптация контролов с помощью `UIAccessibility`. Функционал нужен в том же объеме, что и у обычной версии, при этом неясно, как должен поменяться графический интерфейс. Получается, что есть две версии: одну мы не можем прочитать через VoiceOver, вторую не можем нарисовать. Но и там, и там, мы всё равно работаем с `UIKit`.

Крутость технологий доступности в том, что они берут много данных от существующего графического интерфейса. Они выступают над ним неким «вторым слоем», давая новые возможности работы с интерфейсом: воспринимать на слух, управлять голосом или иначе сообщать сигналы, если не можешь коснуться экрана.

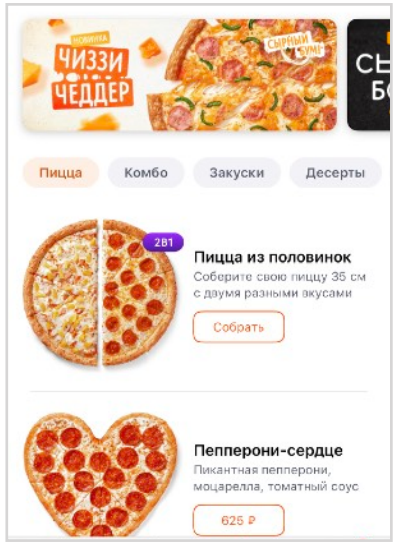
Иногда у доступности не получается правильно понять, что происходит на экране и нужно немного помочь. Это немного кода, зачастую, лишь несколько дополнительных меток, которые позволят работать VoiceOver с приложением как надо: сообщать о выбранном состоянии интерфейса, правильно называть все элементы на экране, поправить порядок чтения элементов.

При этом, крайне редко приходится писать какой-то специальный текст для незрячих. Надо лишь правильно оформить существующие контролы.

Ментальная модель

Не адаптируя доступность, мы работаем в режиме «анимированного макета с данными», копируя только внешние атрибуты, уверяя себя, что так и надо. Адаптируя VoiceOver, мы работаем с ментальной моделью восприятия информации: это другой уровень понимания интерфейса и его сложности.

Ментальная модель не меняется от способа взаимодействия — хорошая модель прекрасно адаптируется и для звукового интерфейса. У аудио-интерфейса может быть другая скорость считывания, другие жесты взаимодействия, но суть остаётся той же самой. Не делайте отдельные версии для людей с нарушениями — делайте нормально для всех.



Графический интерфейс

Акции,
Новинка Чиззи-Чеддер,
1 из 12

Виды продуктов,
Пицца, 1 из 9

Пицца из половинок,
Кнопка

Пепперони-сердце,
625₽, Кнопка

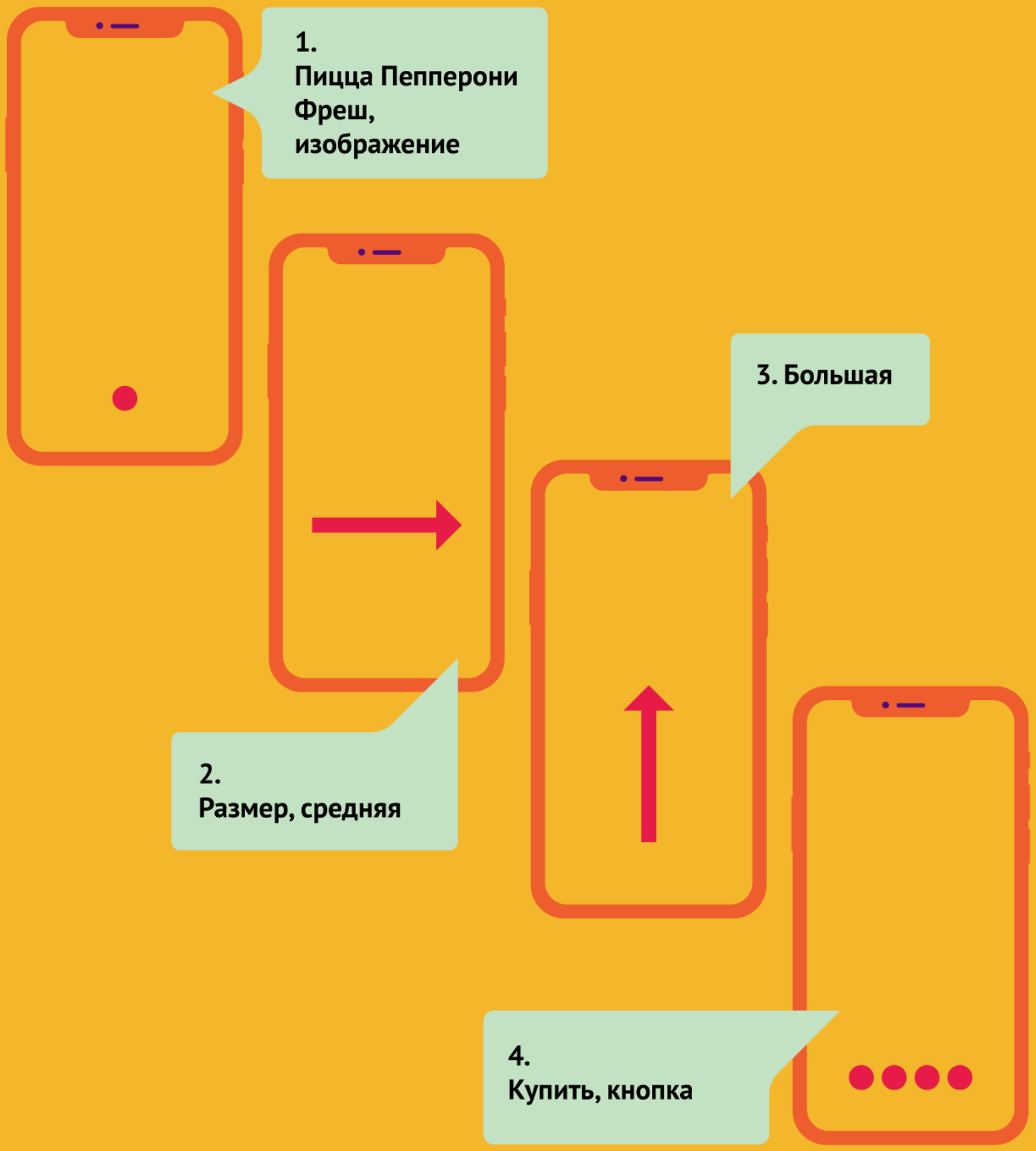
Аудио-интерфейс

Сверху акции,
потом виды продуктов,
в конце список

Ментальная модель

Человек

VoiceOver


1.
Пицца Пепперони Фреш, изображение
2.
Размер, средняя
3. Большая
4.
Купить, кнопка

Как работает VoiceOver

Перед тем как заняться адаптацией приложения, мы должны понять, как незрячие люди пользуются телефоном и что мы должны сделать, чтобы им стало удобней.

Для незрячего экран телефона превращается в сенсорную панель: по ней можно свайпать в разные стороны, тапать на нее одним или несколькими пальцами. В ответ на действия VoiceOver озвучивает текущее состояние интерфейса: на каком элементе мы сейчас находимся, что с ним можно сделать.

Чтобы лучше понять, попробуем пройти всю эволюцию звукового интерфейса с нуля: расскажем о содержимом экрана, отметим элементы, с которыми можно взаимодействовать, подумаем, как ускорить навигацию.

Звуковой интерфейс

Ситуация: вам нужно сделать новый экран для приложения, а макета у вас нет, интернета тоже нет. Вы звоните дизайнеру и он вам описывает картинку по телефону. Что он расскажет?

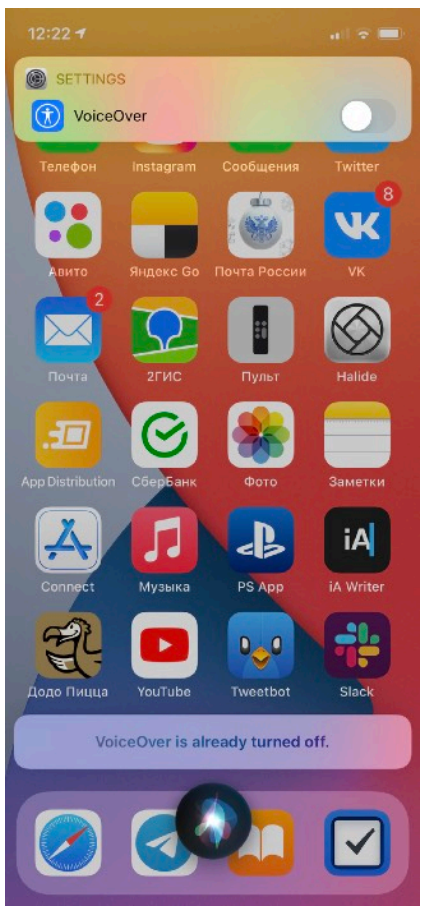
Скорее всего, он перечислит элементы по порядку, даст им название и расскажет об особенностях: заголовок «Пицца», надпись с размером теста, кнопка «купить». Из рассказа вам будет понятен порядок элементов, что они обозначают, в каком состоянии находятся, что с ними можно сделать. Вы готовы работать с этой информацией.

VoiceOver выступает в роли такого друга. Он берёт один элемент, читает его описание, рассказывает о состоянии (выбрано, отключено), говорит какой это тип и что с ним можно делать. В ответ вы можете попросить перейти к следующему элементу, нажать на кнопку или закрыть текущий экран. Команды отдаются разными свайпами на экране.

VoiceOver всеми силами старается уменьшить количество дополнительной работы. Не делайте специальных версий для незрячих, ваша задача лишь помочь VoiceOver назвать всё правильно и обработать ввод. Крутость технологии в том, что работы не так и много: синтез речи уже встроен в iOS, мультитач экрана распознает сложные жесты. Просто подправьте читаемый текст, тип элементов и обработайте несколько дополнительных функций.

Перед адаптацией приложения надо разобраться в деталях работы VoiceOver, научиться ей управлять, посмотреть, как она работает на примере стандартных приложений. Дальше, с пониманием звукового интерфейса, мы приступаем к адаптации своего приложения.

Включение



Чтобы понять, как адаптировать интерфейсы, научитесь работать с ними в таком же режиме, как это делают незрячие. Это просто, включите на телефоне VoiceOver и попользуйтесь.

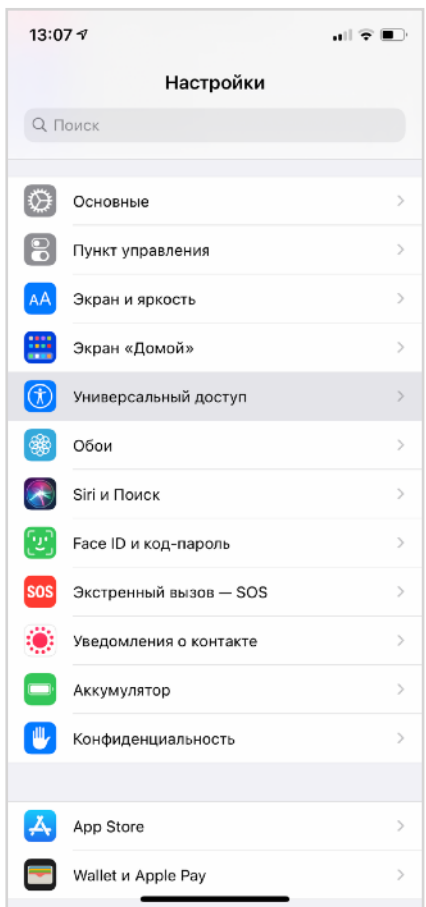
Перед включением важно **научиться отключать.** В самой сложной ситуации вас точно выручит Siri: «Привет Сири, выключи VoiceOver». Чтобы не натолкнуться на выключенную Сири, первое время включайте VoiceOver только через неё.

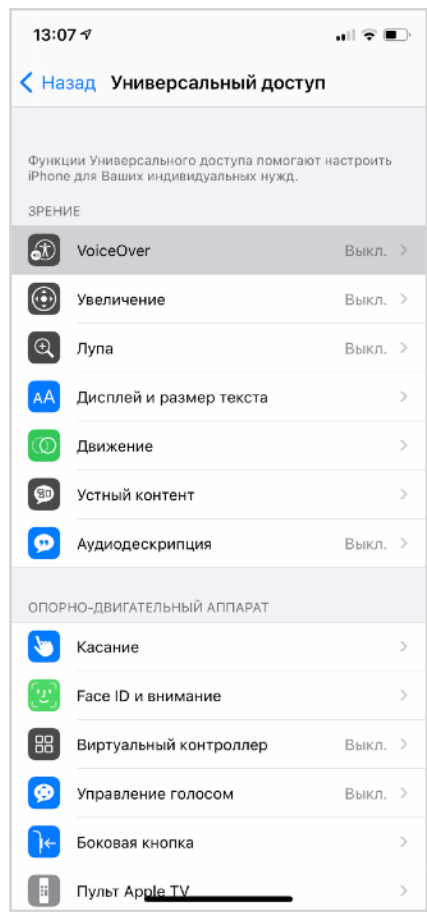
Включить VoiceOver можно еще несколькими способами:

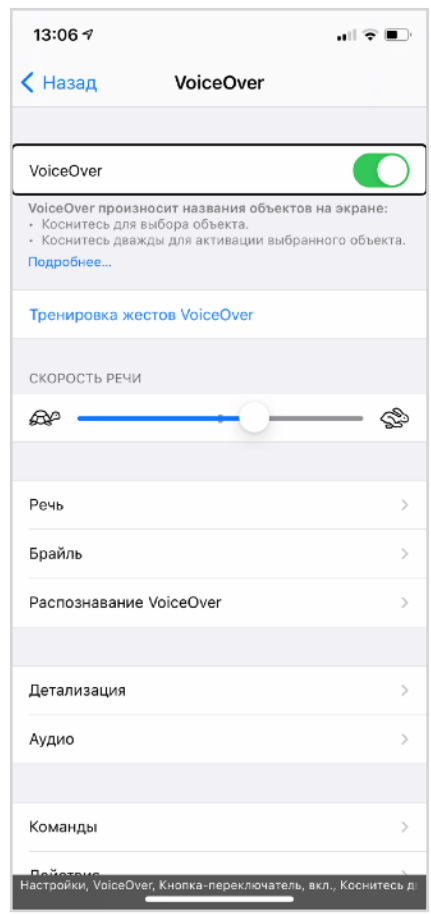
— через Настройки,

— шорткат для доступности,

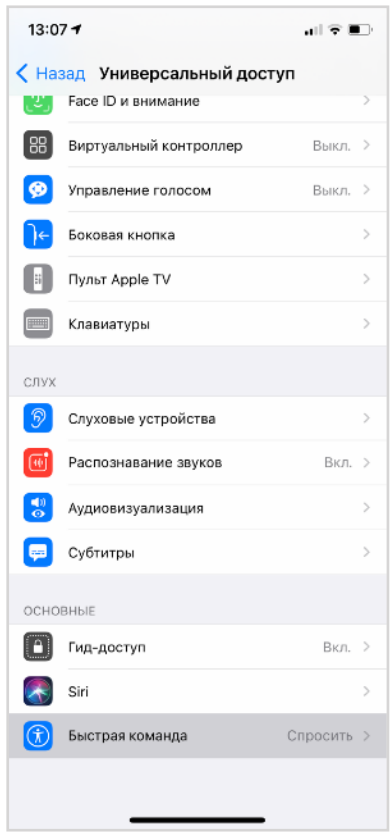
— бэк-тап по спинке телефона.

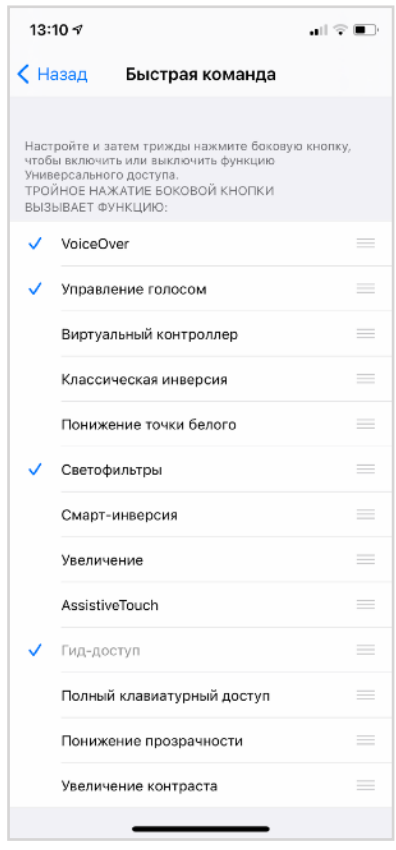


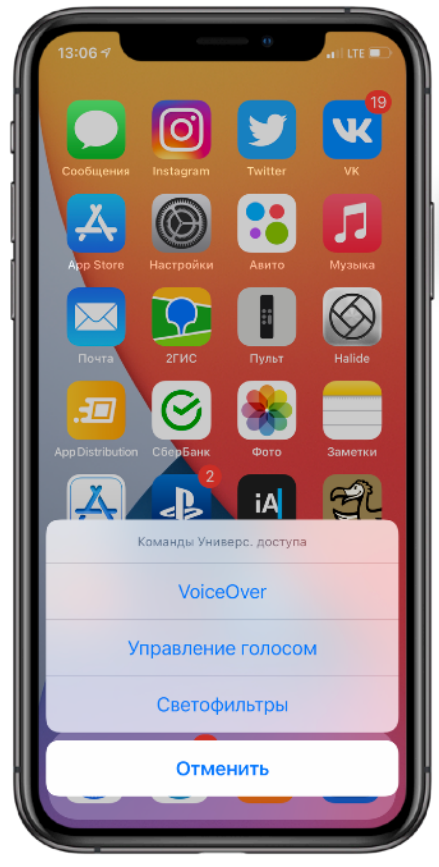




Параметры VoiceOver находятся в разделе «Универсальный доступ» в настройках телефона. Незрячие включают высокую скорость речи, им так удобней.

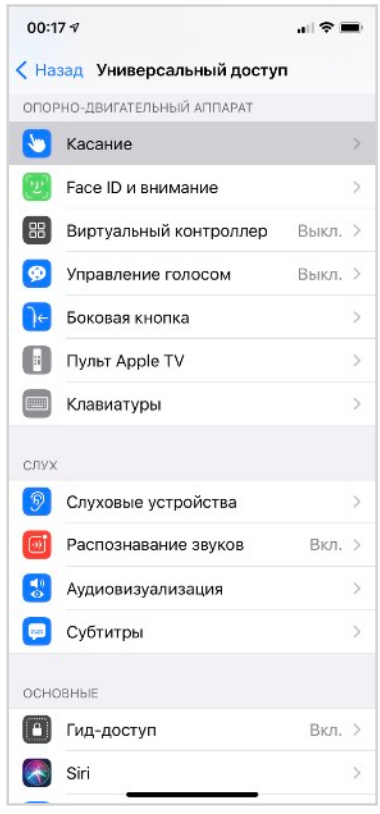


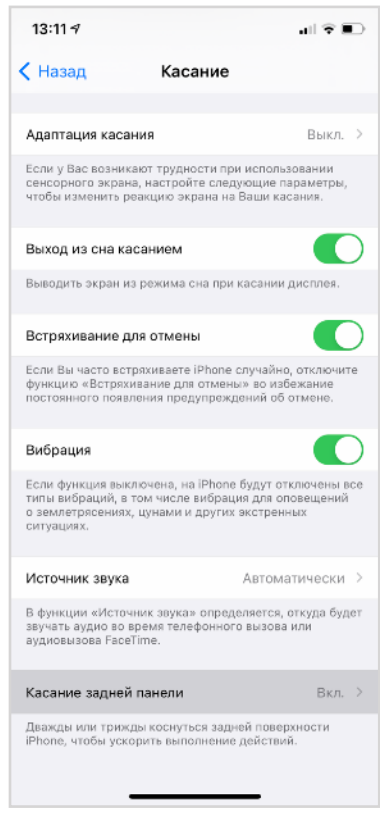


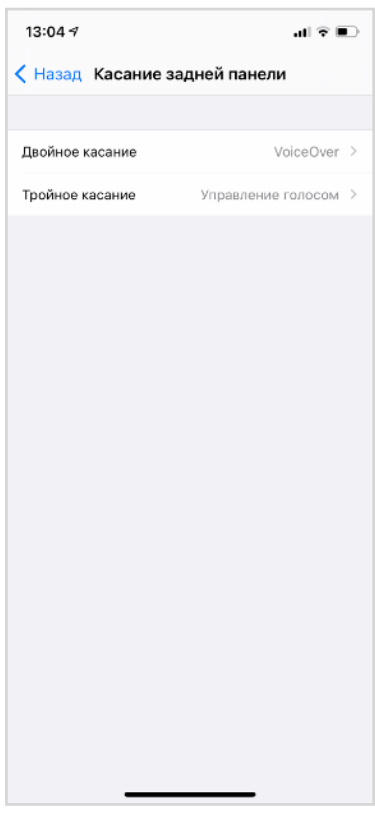


Разные варианты доступности можно активировать по тройному нажатию на кнопку включения. Ищите «Быстрые команды» в конце экрана «Универсальный доступ».

Удобно настроить включение VoiceOver на двойной (или тройной) тап по спинке телефона. Я пользуюсь этим способом чаще всего.

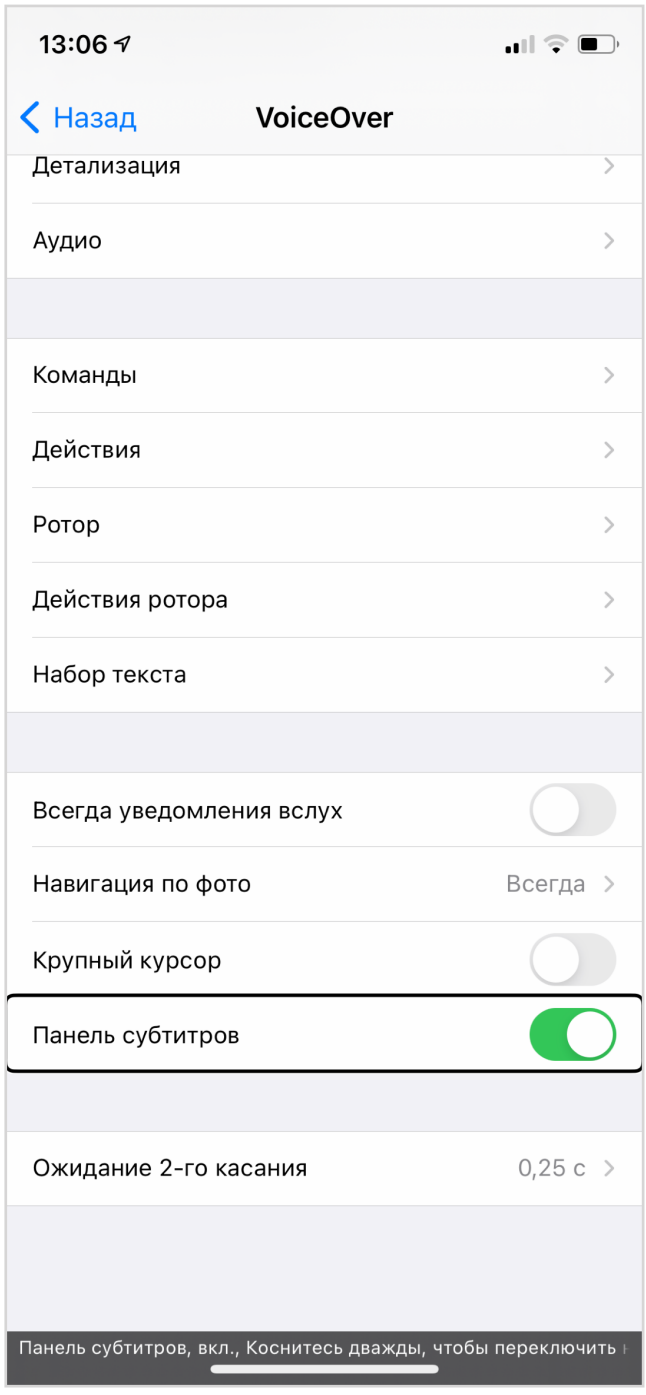






Для отладки удобно не слушать VoiceOver, а смотреть текст, который он читает. Включив субтитры, вы увидите описание текущего элемента.

Описания можно смотреть и через Voice Control — он покажет все элементы за раз, но про него в отдельной главе.



Панель с субтитрами

Voice Control

Пример субтитров

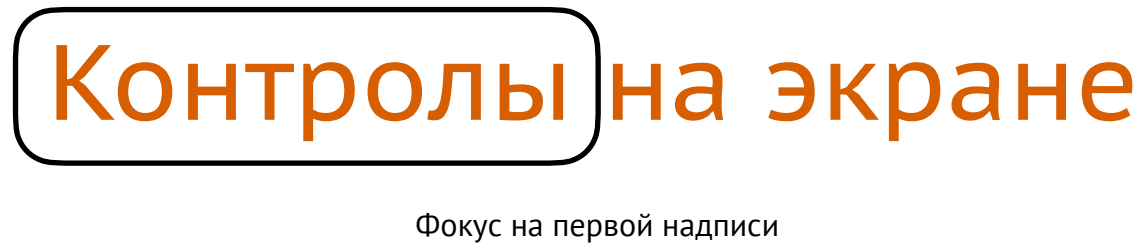
Навигация

При включении VoiceOver на экране появится черная рамка — это фокус. Фокус перемещается по элементам интерфейса от свайпов по экрану и читает описание нового элемента сразу после перемещения.

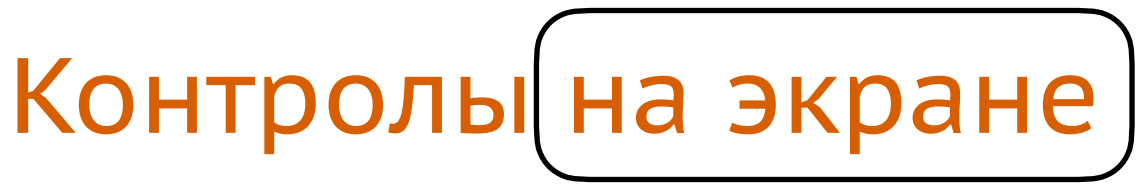
У такого интерфейса сразу несколько следствий:

— Работать можно только с одним элементом за раз.
— Все действия с экрана передаются элементу в фокусе.
— Не надо прицеливаться на элемент. Свайп в любом месте экрана будет влиять на элемент в фокусе.
— Если хотите прочитать описание ещё раз — тапните по экрану.

Свайпните вправо, чтобы перейти к следующему контролу, элементы переключаются в порядке чтения. Свайп влево переключит фокус на предыдущий элемент.



Фокус на первой надписи



После свайпа вправо фокус переключился в порядке чтения

Если фокус дошел до последнего элемента в строке, то по свайпу вправо он сам переключится на следующую строку. Всё ровно так, как мы читаем текст.

Контролы на экране в две строки

Эта система логична, но в мобилке есть один нюанс: чаще всего в строку помещается только один элемент, поэтому при свайпе вправо VoiceOver переключается на следующую строчку.

Контролы в мобиле

Начальное положение фокуса

Выходит, что чаще всего свайп вправо переключает на следующий элемент *вниз*, а свайп влево — на предыдущий элемент *вверх*. Непривычно, но теперь вы знаете почему так.

Контролы в мобиле

По свайпу вправо фокус переходит на следующую строчку

Перемещать фокус можно не только свайпами, но и «касанием»: водите палец по экрану, а VoiceOver будет читать элемент, который находится под вашим пальцем. Свайпами пользуются намного чаще, но изучение касанием — важный помощник в повседневных задачах.

Оба способа будут упоминаться дальше в книге, поэтому запомните их официальное название: **навигация смахиванием** и **изучение касанием**.

Действия

Одиночный тап по экрану выполняет самое важное действие — **читает описание элемента.** Используется он не часто, ведь описание читается при смене фокуса автоматически.

Нажмите на кнопку тапнув дважды в любом месте экрана — действие передастся элементу в фокусе.

Попробуйте двойным тапом открыть программу. Если у вас айфон «с челкой», то она закрывается привычным свайпом снизу, только чуть медленнее: почувствуйте лёгкую вибрацию, после этого отпустите палец и программа свернётся.

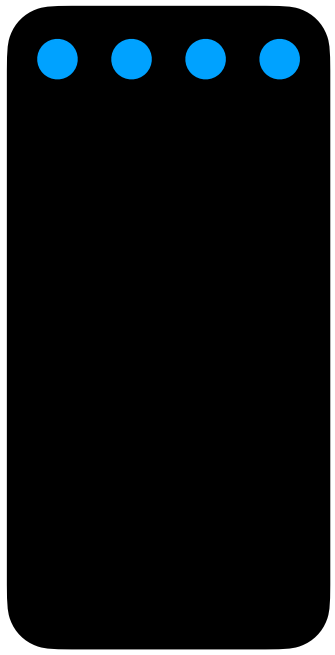
Проскрольте экран свайпнув тремя пальцами в нужном направлении. Полистайте главный экран айфона со значками.

Если хотите погрузиться во взаимодействие незрячих людей с интерфейсами, то **отключите экран,** тапнув трижды тремя пальцами. Включается тоже так.

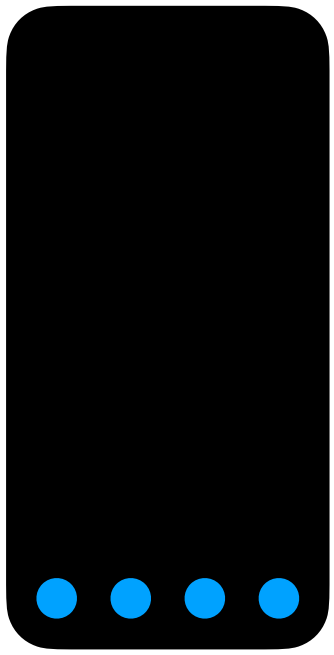
Примечание. В некоторых случаях ещё работает вертикальный свайп, но про это расскажу позже в главах «Элемент регулировки», «Контекстные действия» и «Ротор».

При знакомстве с новой программой, часто нужно **прочитать все элементы** на экране. Если свайпнуть двумя пальцами вверх, то VoiceOver прочитает все элементы с начала экрана, а если свайпнуть вниз — то начиная с текущего.

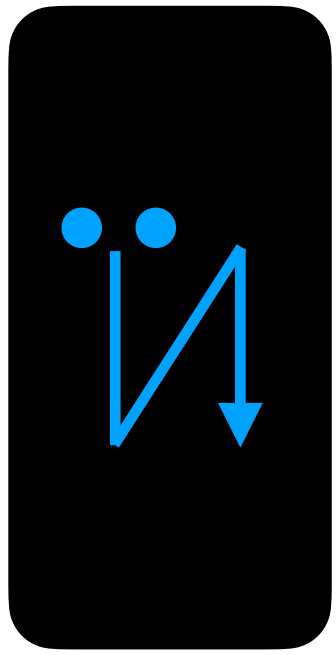
Мы познакомились с несколькими базовыми жестами, но у VoiceOver их намного больше. Несколько популярных можно посмотреть на следующей странице, а полный список в гайде.



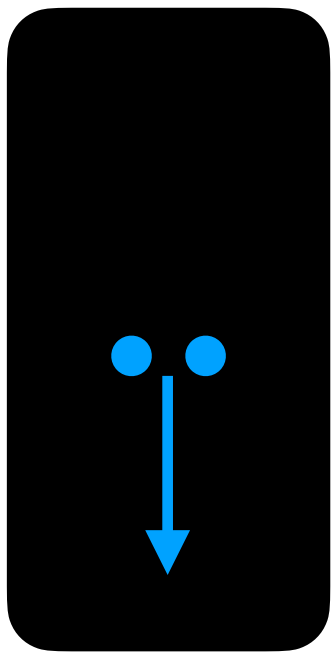
К первому элементу



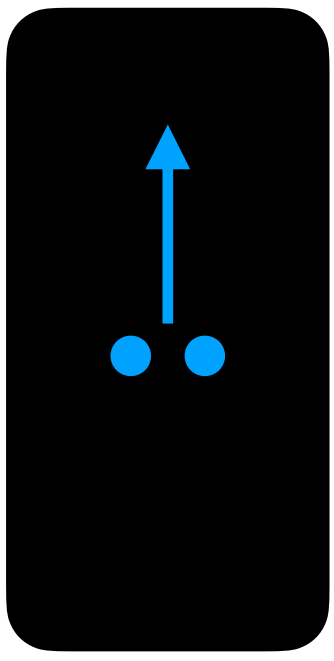
К последнему



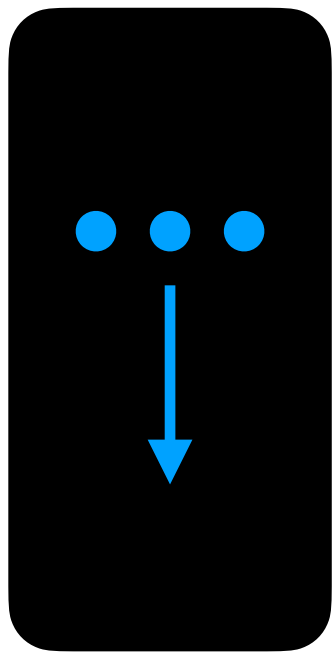
Выйти с экрана



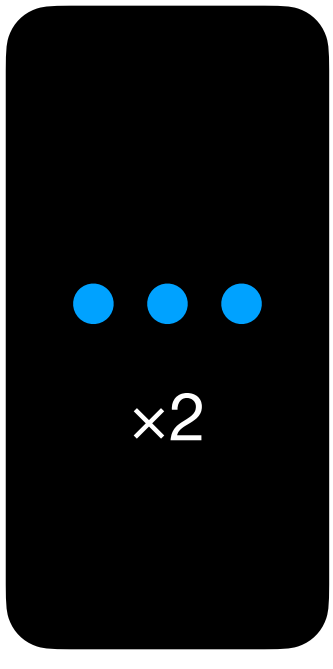
Читать все с текущего



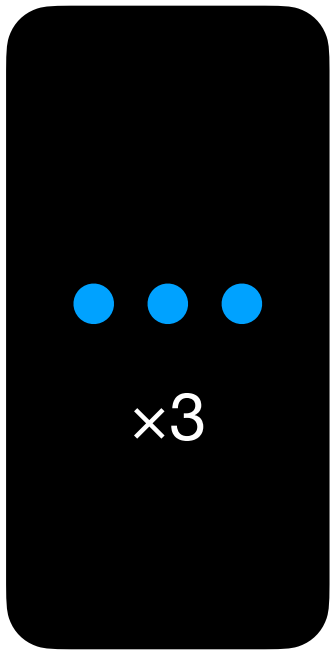
Читать все с первого



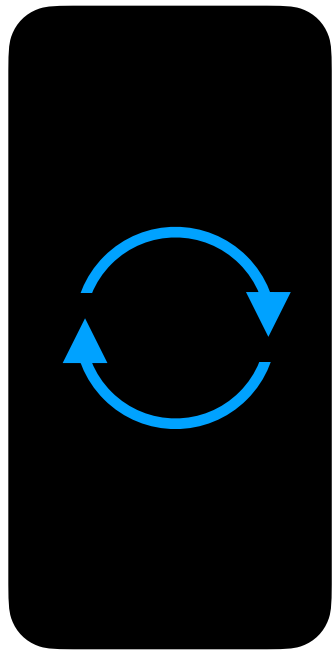
Свайп вниз



Заглушить
VoiceOver



Отключить экран



Ротор

До и после

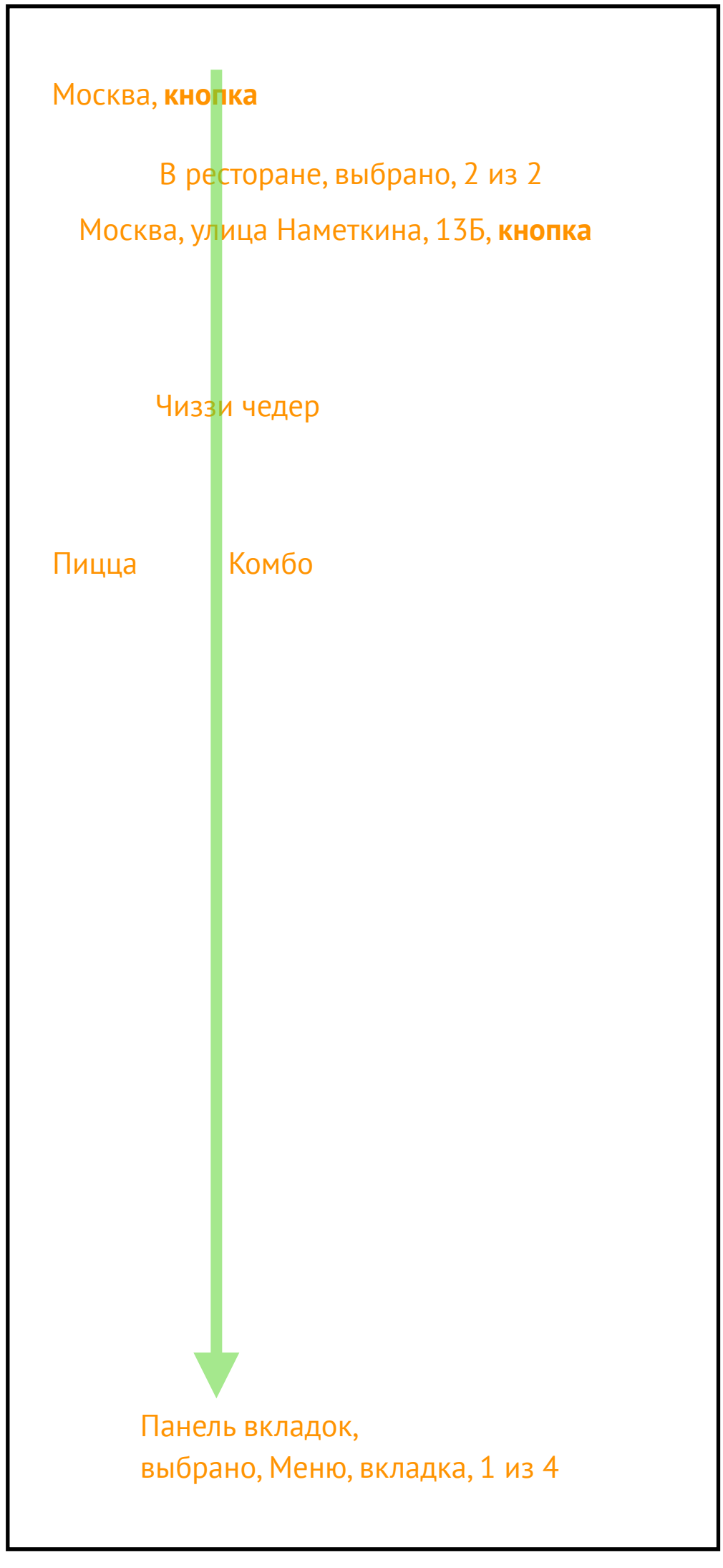
Чтобы понять разницу в значимости адаптации, посмотрим на интерфейс так, как им пользуются незрячие люди. Возьмём экран, который разработчик просто сверстал и ничего не делал для доступности, а потом сравним с тем, как этот же экран воспринимается после адаптации.

Я запускаю приложение и оказываюсь на каком-то экране. Чтобы понять, что на нём, свайпаю двумя пальцами вверх и VoiceOver читает все элементы на экране. Я могу пройтись по ним вручную, но это долго, ведь на экране их больше 22.

Прочитайте. Понимаете на каком экране находитесь? На какие элементы можно нажать?

Москва, кнопка
В ресторане, выбрано, 2 из 2.
Москва, улица Наметкина, 13Б, кнопка
Чиззи чедер
Сырный бортик
Какао с маршмеллоу
3 за 999 ₽
Втройне сырная
Работать с нами
Пицца
Комбо
Закуски
Десерты
Напитки
Другие товары
Пицца из половинок
Соберите свою пиццу за 35 см с двумя разными вкусами
Собрать, кнопка
Пепперони-сердце
Пикантная пепперони, моцарелла, томатный соус
625 знак рубля, кнопка
Панель вкладок, выбрано Меню, вкладка 1 из 4

Элементов очень много. Если я проведу пальцем по левой половине экрана, то тоже получится странно: элементов вдруг стало очень мало.



Если провести по правой половине экрана, то найдутся новые элементы. Вопросов к экрану всё равно много: что такое чиззи чедер и сырный бортик? Что такое пицца-комбо-закуски-десерты? Я могу на них нажать? Что произойдет?

Москва,
кнопка

Тип заказа,
В ресторане, 2 из 2
элемент регулировки

Адрес
Москва, улица Наметкина, 13Б,
кнопка

Акции,
Чиззи чедер, 1 из 7,
***элемент* регулировки**

Раздел меню
Пицца, 1 из 7,
элемент регулировки

Меню

Пицца из половинок,
Соберите свою пиццу за 35 см с двумя разными вкусами,
кнопка

Пепперони-сердце, 625 рублей
Пикантная пепперони, моцарелла, томатный соус
кнопка

Панель вкладок,
выбрано, Меню, вкладка, 1 из 4

Теперь сравним с адаптированной версией этого экрана.

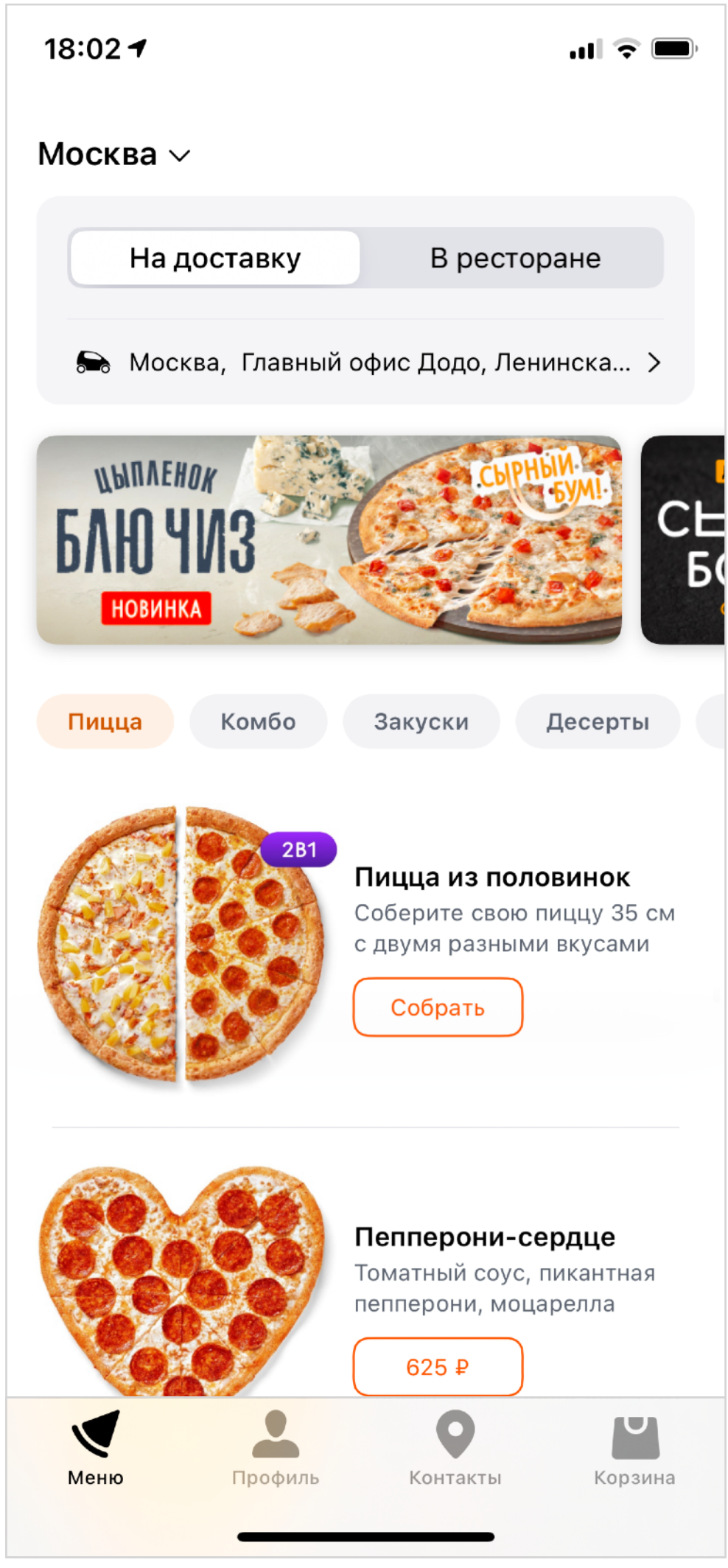
Контролов немного, у всех указан тип, понятно как взаимодействовать с ними, описание сгруппировано и интонация разная (выделил курсивом и жирным шрифтом).

Кнопки можно нажать, тапнув по экрану дважды, элементы регулировки можно менять вертикальными свайпами, подписаны области на экране (тип заказа, акции, меню, панель вкладок).

По такому тексту можно понять, что перед нами экран меню. При этом, мы не сделали ничего сложного: только дали всем элементам тип, немного подправили описание и сгруппировали элементы. Понятность и удобство для незрячего выросло в разы.

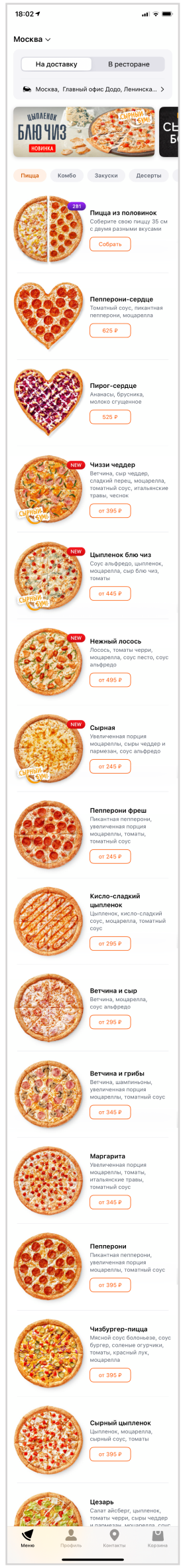
Количество элементов сильно сократилось с 22 до 8, при этом перемещаться между ними можно как свайпами, так и касанием, в любом случае, это будет удобно.

Так графически выглядит экран меню. Много элементов и видов контролов, но легко понять из каких блоков он состоит. Сравните, насколько этот экран похож на текстовую версию по восприятию. Ментальная модель одна, а представления разные.



Помните, в неадаптированной версии было 22 элемента?

После адаптации 22 элемента — это **четыре с половиной экрана.**



Анатомия фокуса

Для работы VoiceOver нужен контрол, на котором можно сфокусироваться. Как он его находит?

В первую очередь, мы обращаемся к корневому вью и спрашиваем: есть ли у него **доступные элементы**? Чтобы ответить на вопрос, `UIView` реализует протокол `UIAccessibilityContainer` и может пройтись по всем своим дочерним элементам, чтобы спросить у них, доступны ли они. Маркер простой: `isAccessibilityElement` должен быть true.

Если элемент недоступен, то, может быть, он работает как **контейнер** для других доступных контролов? Тогда надо у всех по очереди спросить, есть ли у них элементы. Вложенность может быть большой, но в конце только два варианта: либо сама `UIView` доступна, либо дочерних элементов больше нет совсем.

Представим, что мы нашли доступный элемент и решили вокруг него нарисовать рамку.

Для рамки нужны **координаты и размер элемента**, причем в координатах экрана, чтобы можно было по касанию на дисплей попробовать найти этот элемент среди иерархии.

Доступный контрол мы нашли и обвели. Теперь расскажем о нём.

Описание хранится в `accessibilityLabel`. Если у элемента есть какое-то **значение**, то оно хранится в `accessibilityValue`, оно читается после короткой паузы и немного другим голосом. Это создает динамику в речи VoiceOver.

У контрола может быть одно из стандартных свойств:

— **тип** — например, надпись или кнопка;

— **состояние:** обычное, выбранное, отключенное;

— или **особое свойство:** часто обновляется, начинает проигрывать медиа и т.п.

Обычно, свойство добавит текст к описанию элемента и как-то меняет поведение работы VoiceOver.

Вы поняли, что это за элемент, что это кнопка и вы хотите на неё **нажать**. Нажать кнопку можно двойным тапом, он вызовет специальный метод `accessibilityActivate()`.

Для кнопки этот метод эмулирует касание, при этом нажимает **на точку активации,** которая указана в `accessibilityActivationPoint`. Обычно это центр кнопки.

Если после нажатия открывается новый экран, то он **оповещает** VoiceOver о своем появлении и ставит фокус на свой первый элемент. Цикл повторяется.

Алгоритм простой, но его сила в том, что он может работать со всеми интерфейсами приложений и только в играх нужно придумывать что-то особенное. Другие технологии работают похоже: Voice Control возьмет ту же информацию, чтобы показать какие кнопки можно активировать голосом, Switch Control использует тот же фокус, просто дает другой способ управления фокусом. Если появится интерфейс управления через нервные импульсы, то работать он тоже будет через `UIAccessibility`, зуб даю.

Ваша задача — дополнить интерфейс данными, чтобы помочь алгоритму правильно сработать и превратить ваш графический интерфейс в звуковой.

Когда ломается доступность

Доступность ломается от любого действия.

— Убрали текст у кнопки? Теперь VoiceOver не знает как её назвать.
— Сделали горизонтальную карусель, чтобы уменьшить количество элементов на экране? Получили кучу элементов в VoiceOver.
— Разместили 3 надписи в ячейке? Теперь фокус встает на каждой по отдельности.
— Уменьшили .alpha у кнопки, чтобы показать что она отключена? VoiceOver вас не понял и не говорит, что кнопка недоступна.

Нет такого способа верстать интерфейс, который бы не ломал доступность, потому что доступность — это обогащение графического интерфейса информацией о его структуре.

Но есть много свойств у стандартных элементов, про которые мы можем не знать. Например, стандартные ячейки таблицы адаптированы для VoiceOver, но если вы решили сделать свои, или ячейку для `UICollectionView`, то доступность потеряется.

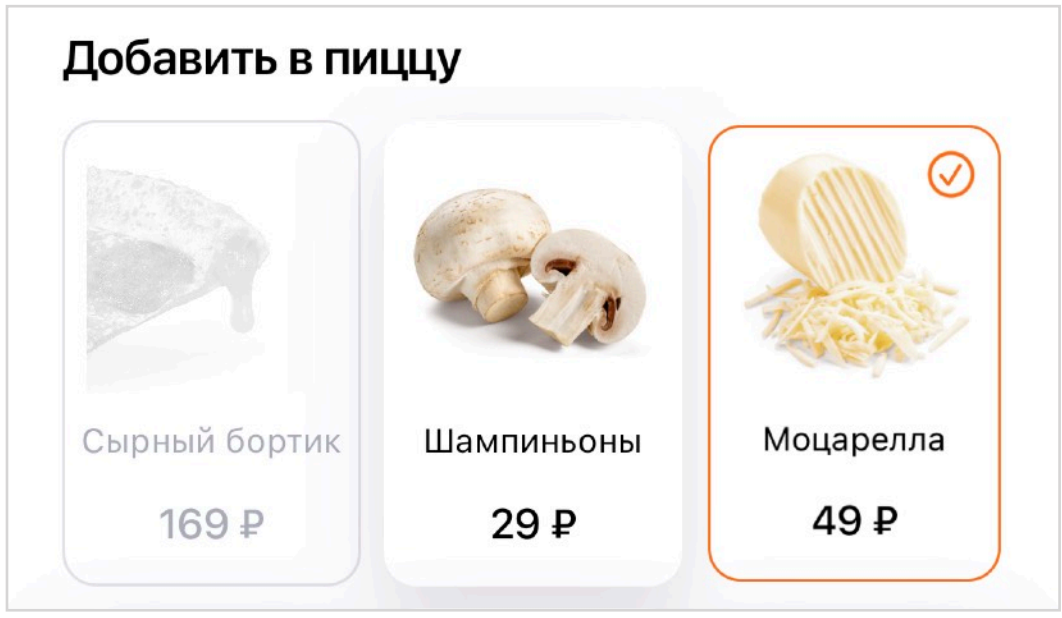
Это не значит, что нельзя делать самодельные контролы — делайте. Просто адаптируйте. Для адаптации сделайте 4 шага, для каждого будет целый раздел:

— подписать,
— упростить,
— поправить навигацию,
— проверить сценарий.

Accessibility tree

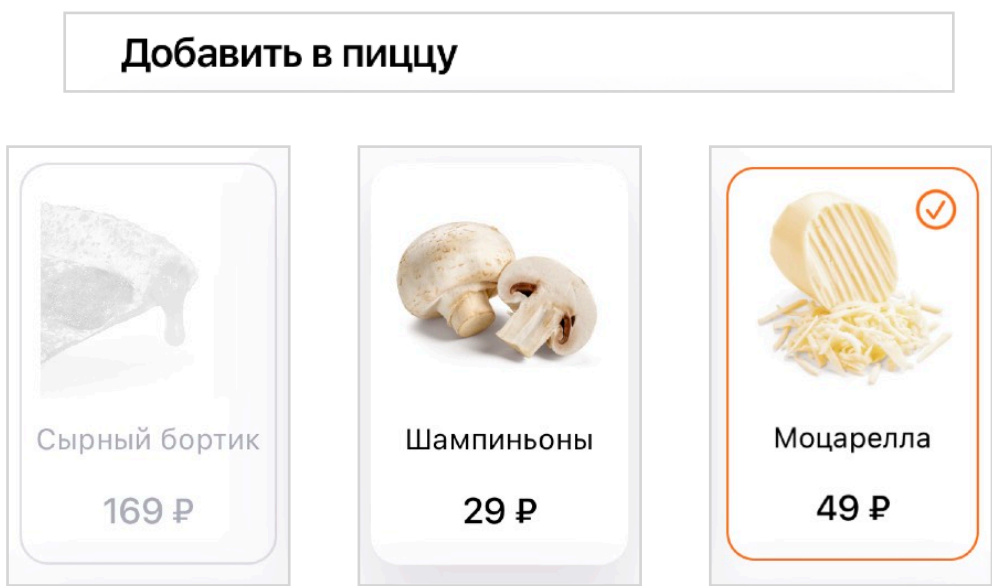
Доступность в iOS обеспечивается протоколом `UIAccessibility`. Он работает с любыми объектами, не обязательно, чтобы они были элементами интерфейса.

```
@interface NSObject (UIAccessibility)
```



Чаще всего VoiceOver запрашивает доступность элементов у иерархии `UIView`. Уже из этих данных он создает фокус и накладывает его поверх интерфейса.

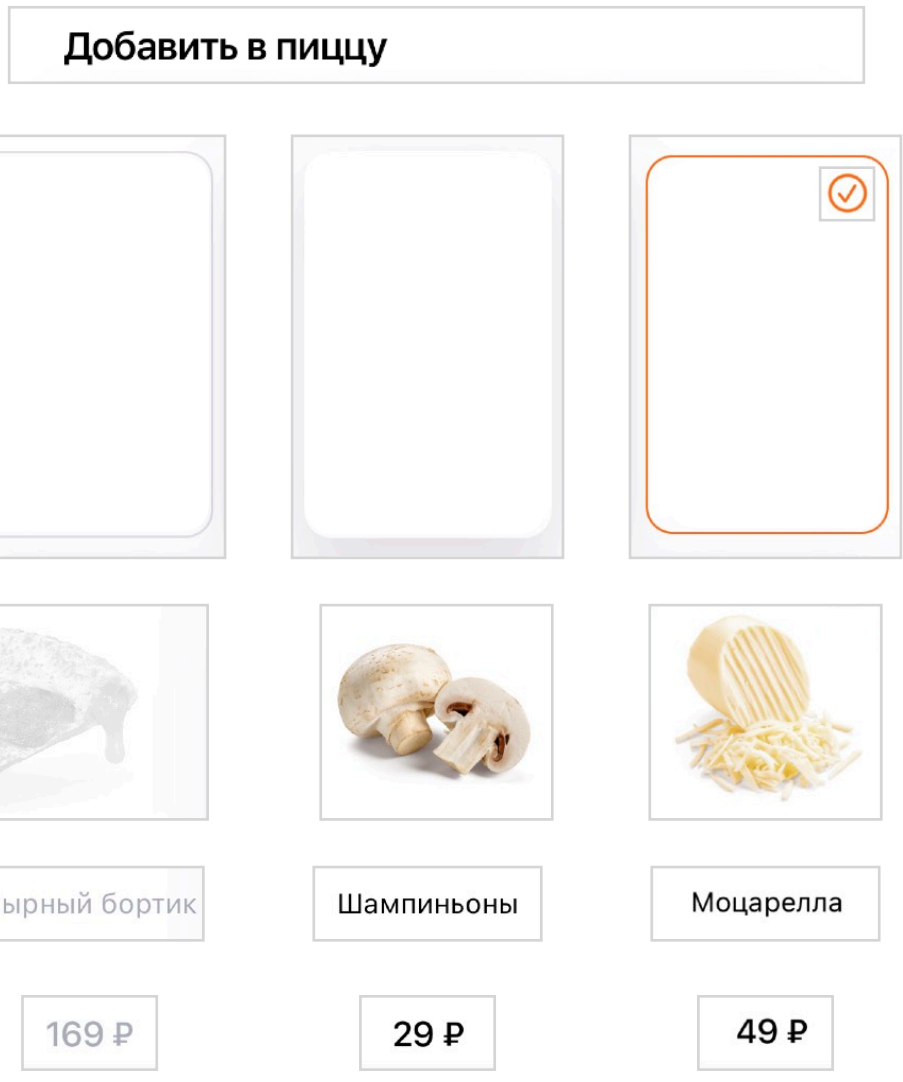
Разберем устройство VoiceOver на примере добавок к пицце. Посмотрим, как мы воспринимаем интерфейс, его техническую сторону и где помочь VoiceOver.



Для зрячего человека на экране 4 логических элемента: заголовок и три ячейки, на которые можно нажать.

Для iOS на экране 13 элементов:

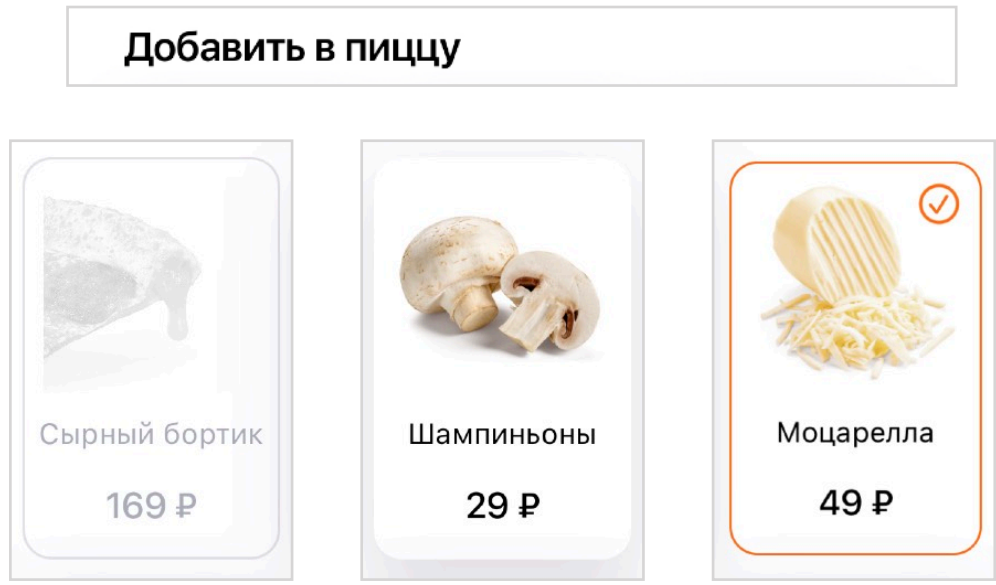
— заголовок,
— 3 ячейки-контейнера с разными состояниями,
— 3 картинки,
— 3 названия,
— 3 цены.



VoiceOver уже имеет ряд встроенных правил, по которым она пытается правильно прочитать интерфейс. Например, она не фокусируется на контейнерах других элементов, поэтому на ячейке она останавливаться не будет, картинки по умолчанию недоступны и VoiceOver их тоже не видит. А вот надписи и цены VoiceOver считает отдельными элементами и незрячему придется свайпать между ними раз за разом, чтобы прочитать.

Ещё VoiceOver ничего не расскажет про структуру экрана: он не знает, что «добавить в пиццу» это заголовок, что недоступно или уже добавлено в пиццу, и что на ячейку можно нажать. Мы понимаем это визуально, VoiceOver — нет.

Задача разработчика — подсказать VoiceOver как правильно читать элементы, какие у них свойства, как сгруппировать и что с контролами можно делать. По сути, нужно вернуться к первой ментальной модели.



В процессе мы будем разбирать, как применить все правила так, чтобы VoiceOver не был многословен, но рассказывал о всём нужном.

Адаптированная версия состоит из четырех элементов, их описание будет такое:

Добавить в пиццу, заголовок
Сырный бортик, 169 рублей, недоступно, кнопка
Шампиньоны, 29 рублей, кнопка
Выбрано, Моцарелла, 49 рублей, кнопка

После добавления ингредиента, VoiceOver ещё и скажет о новой цене всей пиццы.

Всё одновременно кратко и понятно. Незрячие слушают VoiceOver на очень большой скорости, поэтому текст похож на звуковые маячки, а не на поставленную речь диктора.

Практика

- Включи VoiceOver. Для выключения можно дать команду Сири, проверь заранее, что она работает.
- Настрой шорткат на быстрый доступ через кнопку включения или на тап по спинке телефона.
- Попользуйся стандартными приложениям iOS, посмотри как они читают кнопки, их названия и типы.
- Научись сворачивать приложения, открывать панель с оповещениями.
- Попробуй отключить экран и ориентироваться только на звук.
- Попробуй попользоваться своим приложением, отметь непонятные места, где нет подписей, где много контролов и где сломался фокус.
- Получится ли выполнить основной сценарий в твоем приложении?

На этом этапе стоит записывать все непонятные места в вашем приложении, позже вы поймете, как их починить.

Если нет своего приложения, то возьми любое популярное российское. Пример хорошей адаптации можно посмотреть в приложении Додо Пиццы.

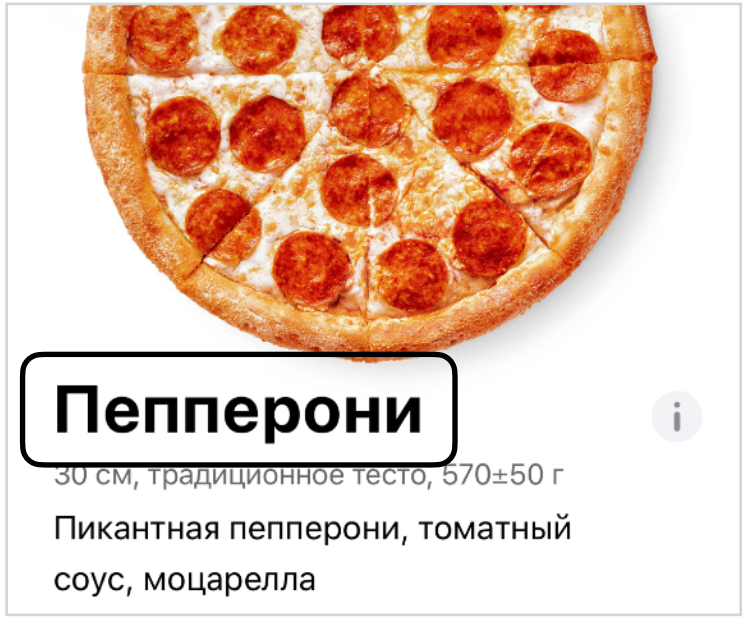
Подписать

Первая проблема доступности, с которой сталкиваются незрячие — элементы подписаны неправильно, или не подписаны совсем. Из-за этого непонятно, что с ними делать и что случится, если попробовать их нажать.

Это самая простая, но самая распространенная проблема. Подписав название элемента и указав его тип, можно исправить половину проблем доступности.

Подписать

Надписи



Текстовая надпись – это базовый элемент доступности.

Обычные подписи уже адаптированы для VoiceOver: их размер задаст фрейм фокусу, а текст в лейбле — это текст самой надписи.

Если вы рисуете текст через CATextLayer, то информация о доступности теряется. Восстановить её можно самостоятельно, указав параметры. Разберем такой случай и посмотрим, что UILabel делает за нас.

Для начала, отметим, что элемент доступен, а значит на него можно поставить фокус. Если значение будет false, то VoiceOver попытается найти доступный элемент среди дочерних.

```
isAccessibilityElement = true
```

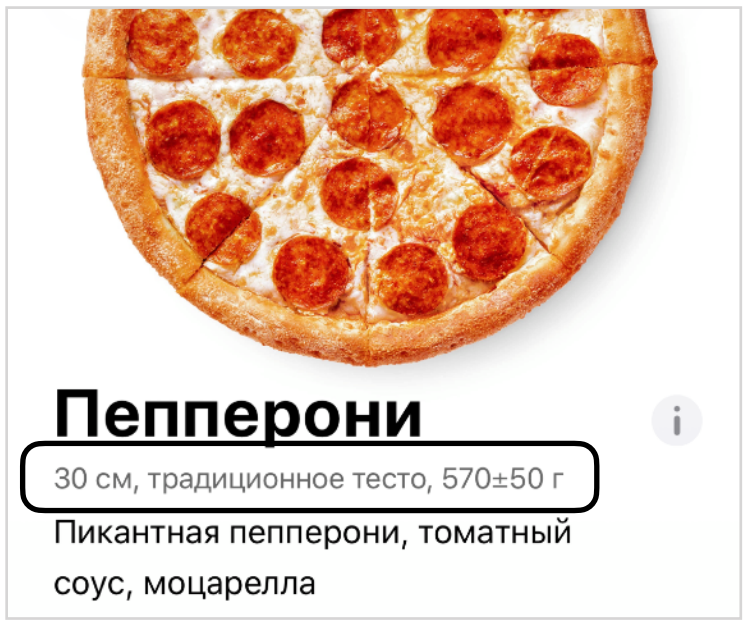
Затем подписать элемент, для описания возьмем текст в элементе. Этот текст VoiceOver прочитает, когда фокус попадет на контрол.

```
accessibilityLabel = text
```

В конце нужно задать фрейм элемента: так фокус будет виден на экране и можно будет навести касанием. Фокус нужен не только для незрячих и VoiceOver, но и для Switch Control.

```
accessibilityFrame = frameInScreenCoordinates
```

Готово, доступность в самом простом виде заработала.



Для хорошей доступности надписи нужно проверять, ведь VoiceOver читает ровно тот текст, который ему дали. Если на входе «30 см», то он и прочитает «тридцать сэмэ», а надо «тридцать сантиметров».

Чаще всего это встречается в ценах, когда красивое «30 ₽» превращается в прямолинейное «тридцать знак рубля».

Текст может быть длинным, просто соедините его через запятую, VoiceOver учтет это и даст правильную интонацию.

Починить можно дополнительным форматированием: например, создайте структуру, которая будет хранить и видимый текст, и специальный текст для VoiceOver.

```
struct AccessibleText {
    let visibleText: String
    let accessibleText: String
}
```

Использовать можно напрямую, или завернуть в какой-нибудь красивый экстеншен для `UILabel`.

```
titleLabel.text = title.visibleText
titleLabel.accessibilityLabel = title.accessibleText
```

Множественное число

Текст будет читаться намного лучше, если вы его просклоняете: 1 рубль, 2 рубля, 5 рублей. В каждом языке разные способы работы с числами. В английском их два, в русском 3. Все варианты можно описать в файле `Localizable.stringsdict`. Подробнее про формат файла в документации.

▾ %d рублей ⇕	Dictionary	(2 items)
NSStringLocalizedFormatKey	String	%#@Variable@
▾ Variable	Dictionary	(6 items)
NSStringFormatSpecTypeKey	String	NSStringPluralRuleType
NSStringFormatValueTypeKey	String	d
zero	String	%d рублей
one	String	%d рубль
few	String	%d рубля
other	String	%d рублей

В коде строку с локализацией нужно вызвать так:

```
String(format:
    NSLocalizedString("%d рублей", comment: ""),
    price)
```

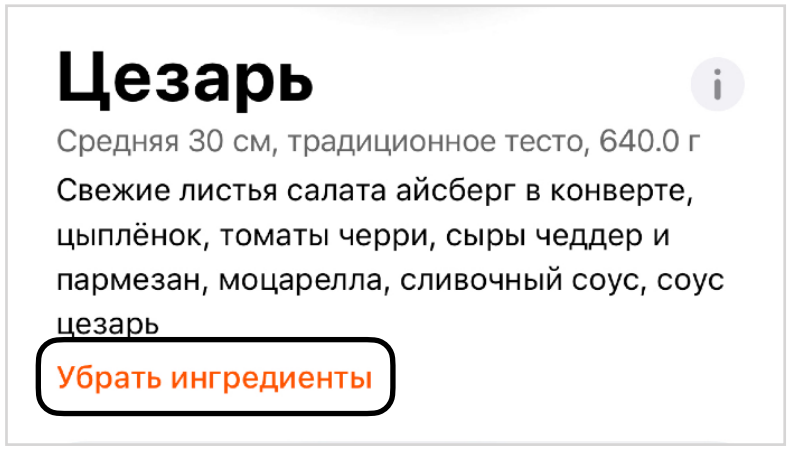
Еще в таком файле можно использовать не только ключ `NSStringLocalizedFormatKey`, но и `NSStringVariableWidthRuleType`. С его помощью можно задать отличающийся текст для разной ширины элементов, например, сокращать текст для маленьких размеров экрана.

▾ Телефон ⇕	Dictionary	(1 item)
▾ NSStringVariableWidthRuleType	Dictionary	(2 items)
320	String	Тел.
375	String	Телефон

```
let screenWidth = Int(UIScreen.main.bounds.width)
let bundle: Bundle = Bundle()
(NSLocalizedString("Телефон",
                   bundle: bundle,
                   comment: "Back button title") as NSString)
    .variantFittingPresentationWidth(screenWidth)
```

Подписать

Кнопки



Теперь разберем, как работает доступность кнопок. Кнопка с текстом читается так же как и надпись, но в конце добавляется подпись «кнопка»:

Убрать ингредиенты, кнопка

Описание типа элемента находится в конце текста, чтобы подсказать, что можно сделать после того, как услышите описание. Добавлять текст «кнопка» не нужно, за вас это сделает iOS, вам нужно лишь указать, что этот элемент является кнопкой, для этого поставьте трейт .button.

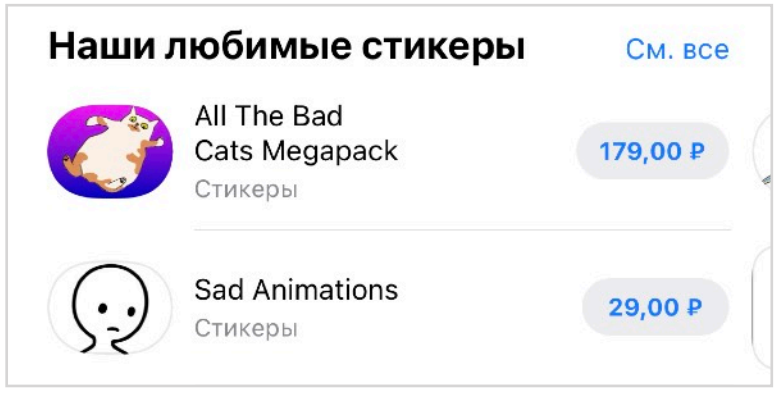
```
accessibilityTraits = .button
```

Действие от нажатия обрабатывается в функции `accessibilityActivate()`. Кнопка вызывает обычное нажатие на себя и возвращает true, если действие обработалось. Если вернется `false`, то VoiceOver попытается вызывать метод у следующего объекта в иерархии `UIView`.

```
override func accessibilityActivate() -> Bool {
    // sendEvent(.touchUpInside)
    // return true
}
```

`AccessibilityActivate()` можно вызывать у любого доступного объекта, лишь бы у него был выставлен `isAccessibleElement = true`.

Текст кнопки надо проверять так же, как и обычные надписи. Различные сокращения и знаки могут читаться не так, как вы ожидаете.



~~См все~~ Смотреть все

~~179 знак рубля~~ 179 рублей

~~29 знак рубля~~ 29 рубля

Кнопками могут быть не только элементы `UIButton`, но и, например, ячейки в списках. Если на ячейку можно нажать, то мы должны рассказать об этом. Самый простой способ — поставить трейт `.button` для ячейки. Про списки и ячейки мы еще подробно поговорим в следующей главе.

Подписать

Кнопки без текста

В угоду краткости, из многих кнопок убирают текст. Причин для этого много: неясно какой текст писать, кнопка неважная и не должна быть большой, ее действие очевидно и т.д. Доступность при этом сильно теряется, так как текста больше нет и VoiceOver нечего читать.

VoiceOver пытается хоть как-то помочь понять что на кнопке, поэтому читает название файла картинки!

Примеры с экрана: «айсиклоуз» (файл назывался icClose, сокращение для icon close), «айсикарт» и «айсиинфо». Что это означает догадаться можно, но не всегда.

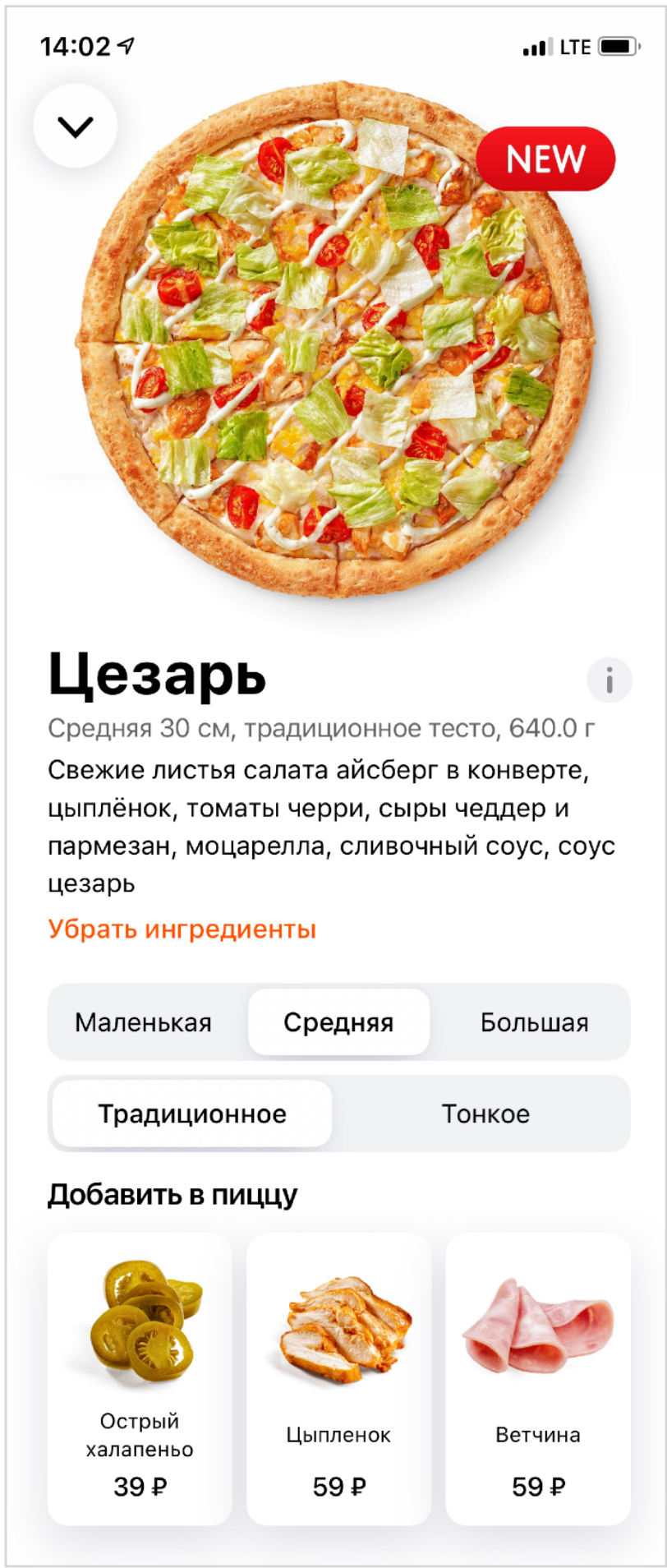
Восстановить доступность легко, достаточно дать кнопкам название через `accessibilityLabel`.

```
closeButton.accessibilityLabel = "Закрыть"
cartButton.accessibilityLabel = "В корзину"
infoButton.accessibilityLabel = "Пищевая ценность"
```

Таббар в приложении Фейсбука закастомизирован, но подписан

Элементы в кастомном `UITabBarItem` подписываются точно так же.

Изображения



С картинками двоякая ситуация. С одной стороны, для незрячих особой ценности они не несут, iOS их даже скрывает от VoiceOver по умолчанию.

С другой стороны, может быть важно знать, что на экране есть изображение: его можно показать знакомым, сделать скриншот и много чего еще. iOS всеми силами старается рассказать о содержимом картинки, даже может распознавать изображение, рассказать что на нем, узнавать ваших знакомых. Например, у этой картинки описание «пицца на белом фоне, new». Распознавание можно включить через ротор.

Все маленькие иконки и декоративные элементы точно нужно скрывать, а вот с большими картинками посложнее. Если вы хотите явно сообщить о наличии изображения, то поставьте ему трейт `.image`, VoiceOver добавит к описанию «изображение».

Если на картинке есть ценная информация, которая больше нигде не дублируется, то это нужно где-то описать. На примере есть бейджик NEW, это важная информация про новинку, VoiceOver должен про это рассказать. Не обязательно сообщать об этом на картинке, можно добавить текст «новинка» к названию пиццы.

Подписи

Как называть элементы?

У хорошего названия элемента несколько свойств:

- Он кратко описывает элемент.
  - Одно слово:

    Добавить, Играть, Удалить, Искать.
  - Короткая фраза:

    Играть музыку, Указать имя, Добавить к событию.
- Не содержит тип, его помечаем через трейт.
- Начинается с заглавной буквы, не заканчивается точкой. Название от значения VoiceOver отделит сам короткой паузой.
- На языке пользователя: переведенное и простое по смыслу.
- Если надпись составная, то части нужно разделить запятой.

  Пепперони, 25 см, тонкое тесто

Больше про синтаксис можно прочитать в документации.

Практика

- Попользуйся приложением, найди элементы, которые не подписаны.
- На какие элементы можно нажать, но они не обозначены как кнопки? Чаще всего непонятно, что ячейки в списках кликабельны.
- В каких случаях текст читается неправильно из-за сокращений? На странные ударения пока внимания не обращай.
- Попадались ли неподписанные картинки, которые стоит скрыть? На каких наоборот, есть важная информация, о которой нужно сообщить вслух?

Упростить

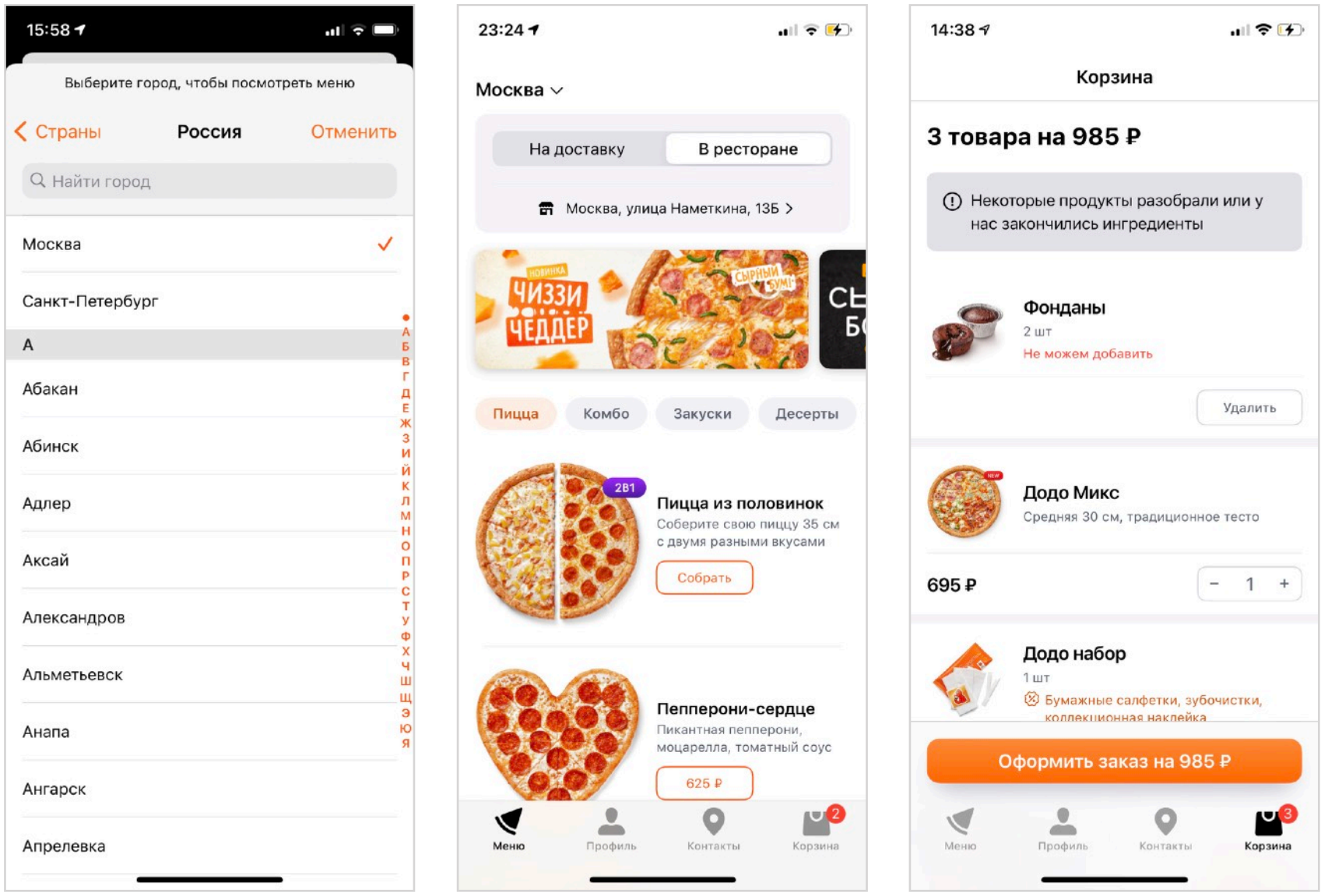
Подписав все элементы и указав их тип, мы исправили самый простой, но массовый слой проблем. Для большинства приложений этого будет достаточно.

Теперь надо разобраться со следующей проблемой: на экране очень много элементов, связи между ними не всегда понятны, нужно постоянно между ними переключаться. Количество элементов можно уменьшить, связи добавить, а удобство использования повысить.

Большинство экранов мобильных приложений — это списки. Мы воспринимаем ячейку в списке как одно целое, поэтому и VoiceOver должен воспринимать ее как единый элемент доступности. Посмотрим на разные ячейки и будем постепенно усложнять их, наблюдая за тем, как меняется доступность этих ячеек.

Упростить

Списки и ячейки



Большая часть интерфейсов приложений — это списки, а для доступности это:

— много ячеек,

— много контролов внутри каждой ячейки.

В идеальном списке каждая ячейка — это один доступный элемент, а описание ячейки рассказывает о всем содержимом, что находится внутри нее.

Интересно, что поведение таблиц и коллекций отличается: в таблицах есть системные стили, они влияют и на правильное описание для VoiceOver. Коллекции примитивней, для них всю доступность нужно поддерживать самостоятельно. Рассмотрим на примерах, постепенно усложняя.

Начнем с ячейки из одной подписи и воссоздадим ее доступность с нуля.

Ячейка является контейнером для `UILabel`, поэтому VoiceOver ее пропустит и поставит фокус именно на надпись. Фокус получится маленький, его размер будет отличаться для каждой ячейки. Самое критичное, что на ячейку можно нажать, но VoiceOver об этом не сообщает, ведь он смотрит на простой текст.

Ячейка, в которой элементом доступности является подпись.

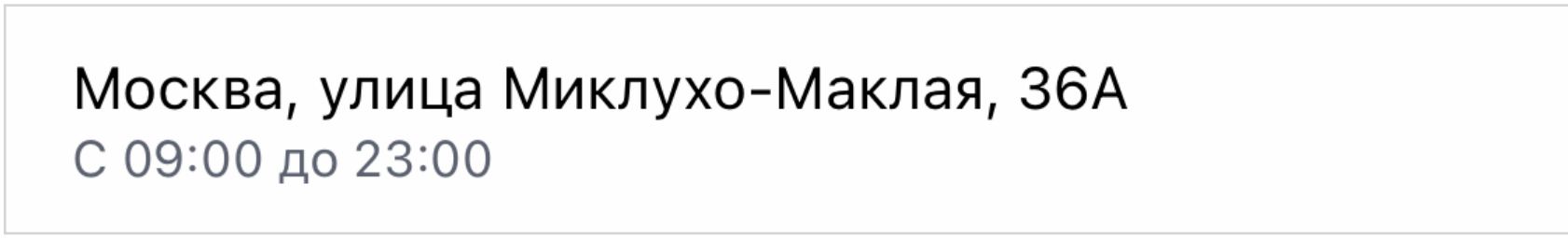
Адаптированная ячейка: «Москва, кнопка». Фрейм занимает весь размер ячейки

Для адаптации ячейки нужно:

— сделать всю ячейку доступным элементом,
— дать ей `label`, который опишет содержимое,
— если на ячейку можно нажать, то указать трейт `.button`.

Доступность самой подписи можно не отключать, VoiceOver уже не увидит ее внутри ячейки. Код для адаптации простой: сообщаем ячейке, что она стала доступным элементом, ставим ей трейт `.button`, а текст заголовка ставим не только в подпись, но и в `.accessibilityLabel`.

```swift
class LocaleCell: UITableViewCell {
    @IBOutlet private weak var nameLabel: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()

        isAccessibilityElement = true
        accessibilityTraits = .button
    }
    var title: String? {
        didSet {
            nameLabel.text = title
            accessibilityLabel = title
        }
    }
}
```

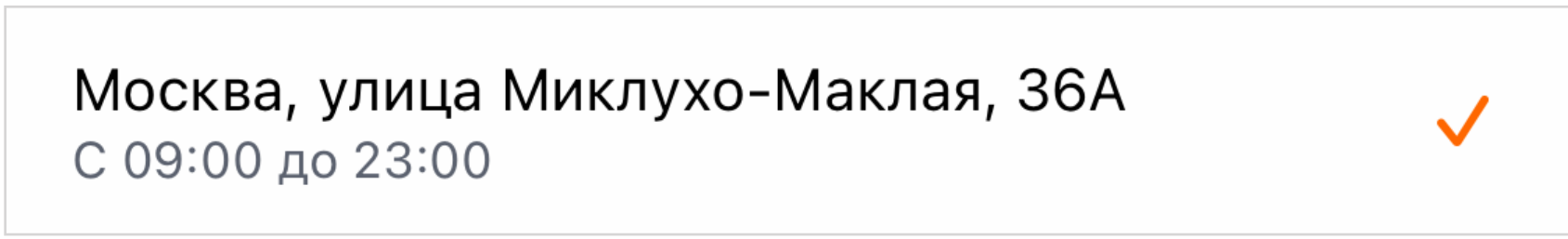


Усложним ячейку. Теперь у нее есть вторая строчка со временем работы. В ячейке две смысловые части: адрес и время работы, причем время нам интересно, только если это нужный нам адрес.

Мы можем объединить два текста в один через запятую и записать в `accessibilityLabel`. Можно интересней: время работы записать в `accessibilityValue`, тогда VoiceOver добавит небольшую паузу и прочитает время с другой интонацией, так появится живость речи и будет проще воспринимать на слух.

label: Москва, улица Миклухо-Маклая, 36А
value: С 9 до 23
traits: кнопка

```
class PizzeriasTableViewCell: UITableViewCell {
    @IBOutlet weak var titleLabel: UILabel!
    @IBOutlet weak var scheduleLabel: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()

        isAccessibilityElement = true
        accessibilityTraits = .button
    }

    var title: String? {
        didSet {
            titleLabel.text = title
            accessibilityLabel = title
        }
    }

    var schedule: String? {
        didSet {
            scheduleLabel.text = schedule
            accessibilityValue = schedule
        }
    }
}
```



label: **Выбрано**, Москва, улица Миклухо-Маклая, 36А
value: с 9 до 23
traits: кнопка

Продолжим усложнять ячейку.

У ячейки может стоять галочка, так мы отмечаем текущую пиццерию. У VoiceOver есть стандартный подход к обозначению выбранного элемента — трейт `.selected`.

Элемент можно отметить одновременно и типом (кнопка), и состоянием (выбрано), поэтому трейты надо указывать через функции работы с `OptionSet`.

```
override public var isSelected: Bool {
    didSet {
        if isSelected {
            accessibilityTraits.insert(.selected)
        } else {
            accessibilityTraits.subtract(.selected)
        }
    }
}
```

Интересно, что даже в таком формате VoiceOver читает время как «с девять до двадцать три». Слова не склоняет, но и лишние ноли не читает.

Москва, улица Миклухо-Маклая, 36А

С 09:00 до 23:00, сейчас закрыто

label: Москва, улица Миклухо-Маклая, 36А
value: С 9 до 23, сейчас закрыто
traits: **недоступно,** кнопка

Если вам нужно показать, что ячейку сейчас выбрать нельзя, то используйте стандартный трейт .notEnabled, он добавит к описанию «недоступно».

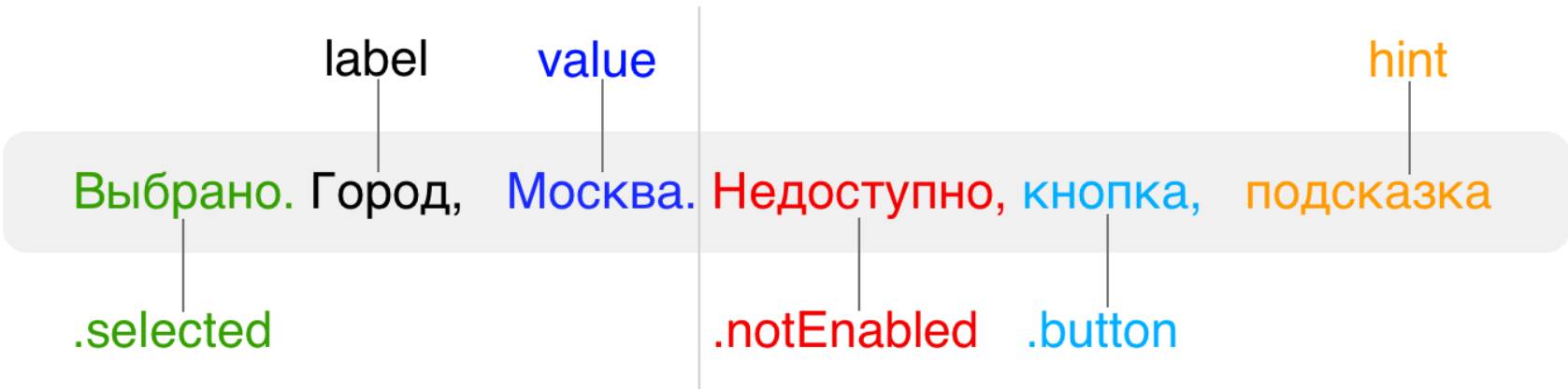
```
accessibilityTraits.insert(.notEnabled)
```

Использовать трейт `.notEnabled` нужно только для тех элементов, с которыми нельзя взаимодействовать. Если я могу выбрать адрес, то трейт `.notEnabled` указывать не нужно и о закрытой пиццерии надо рассказать через value.

За последние 3 страницы мы столкнулись с несколькими трейтами:

- `.selected`,
- `.notEnabled`,
- `.button`.

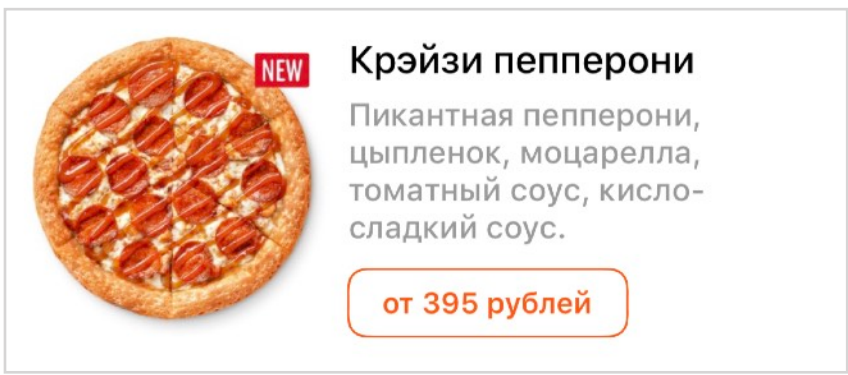
Каждый из них описывает одно из стандартных поведений, при этом подписи появляются в разных местах. С другой стороны, у нас есть `accessibilityLabel`, `accessibilityValue` и `accessibilityHint`. Пора разобраться в порядке чтения.



Если вы листаете большой список из элементов, то вам важнее всего узнать, какой элемент **выбран**. Затем, что это за элемент, его **`label`** и **`value`**. Когда вы поняли, что за элемент в фокусе, вы можете решить, что нужно с ним делать, поэтому **доступность** элемента, его **тип** и **подсказка**, как с ним взаимодействовать, находятся в самом конце описания. Перед `value` и `hint` есть небольшие паузы.

Упростить

Сложная ячейка



Перейдем к сложным ячейкам, когда внутри несколько контролов. Без адаптации фокус будет читать каждую надпись отдельно и надо будет свайпнуть минимум три раза, чтобы перейти к следующему продукту.

Для комфортной работы ячейка должна быть одним элементом, тогда текст будет читаться один за другим. При этом, его можно прервать, выполнив одно из действий: нажать на ячейку, перейти к следующей, или просто остановить чтение.

Составим модель ячейки

Картинка незрячему не нужна, знать, что она там есть тоже бесполезно. Остается 3 надписи. Как их считывает зрячий человек? Название пиццы первое и выделено жирным. Скорее всего, следующим он прочитает цену, ведь на ней цветовой акцент, а состав в последнюю очередь, потому что он длинный и блеклый. Итак, порядок восприятия такой:

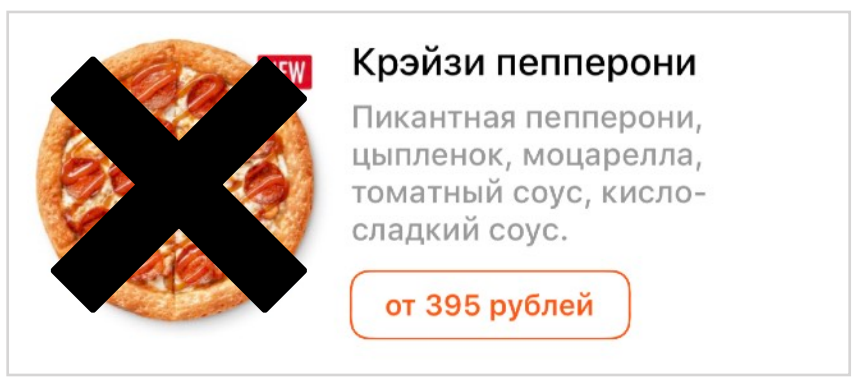
— название,

— цена,

— состав.

Название и цену можно объединить через запятую и поставить в `label`. Состав отделить интонацией и сохранить в `value`.

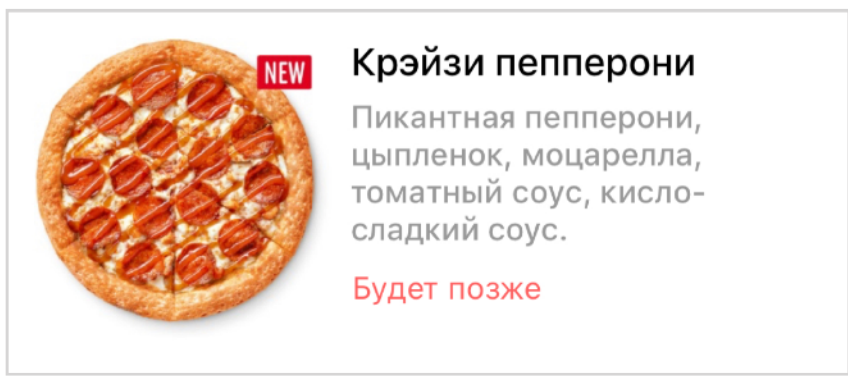
label: Крэйзи пепперони, от 395 рублей
value: Пикантная пепперони, цыпленок, моцарелла, томатный соус, кисло-сладкий соус.
trait: кнопка



Интересно, что в графическом дизайне порядок восприятия не всегда правильный, опрятность верстки иногда важнее смысла.

Код для доступности достаточно прост: делаем ячейку доступной, даем ей описание и поведение кнопки, а затем лишь собираем две строчки с описанием.

```
override func awakeFromNib() {
    super.awakeFromNib()

    isAccessibilityElement = true
    accessibilityTraits = .button
}

func updateAccessibility(title: String,
                         price: String,
                         ingredients: String?) {
    accessibilityLabel = [title, price].joined(separator: ", “)
    accessibilityValue = ingredients
}
```



Если продукт недоступен, то кнопка с ценой заменится на надпись «будет позже». Для VoiceOver графическая смена контролов неважна, поэтому просто меняем текст цены:

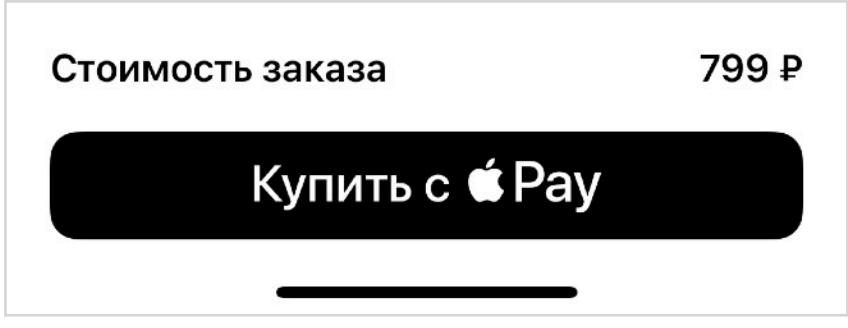
label: Крэйзи пепперони, будет позже
value: Пикантная пепперони, цыпленок, моцарелла, томатный соус, кисло-сладкий соус.
trait: кнопка

С учетом этого, код становится лишь чуть сложнее:

```
func updateAccessibility(title: String,
                         price: String,
                         ingredients: String?,
                         isProductAvailable: Bool) {
    let price = isProductAvailable ? price : "Будет позже”

    accessibilityLabel = [title, price].joined(separator: ", “)
    accessibilityValue = ingredients
}
```

Упростить

Название+значение



В интерфейсах часто встречается паттерн: одна надпись рассказывает о названии, другая о значении. Мы воспринимаем строчки как одно целое, но для VoiceOver это отдельные элементы. На примере хорошо видно, что расположение стоимости и цены совпадает с порядком чтения: название, а затем величина. Так человек получает информацию в правильной последовательности даже без адаптации.

Мы можем улучшить, сделав из всей строки один доступный элемент. Для этого описание двух лейблов нужно склеить в одну строку, а рамку фокуса растянуть на всю строку. Вариантов адаптации несколько. Можно взять текст из первой надписи, через запятую добавить из второй (VoiceOver учитывает пунктуацию в речи) и записать это все в `label`.

```
titleLabel.accessibilityLabel = "Стоимость заказа, 799 рублей"
```

Текст можно забирать прямо из контролов:

```
titleLabel.accessibilityLabel = titleLabel.text!
                              + ", "
                              + valueLabel.text!
```

Можно сделать это чуть изящней: название оставить в label, а величину записать в value. В таком случае VoiceOver немного изменит интонацию для значения, получится красиво.

```
titleLabel.accessibilityLabel = "Стоимость заказа"
titleLabel.accessibilityValue = "799 рублей"
```

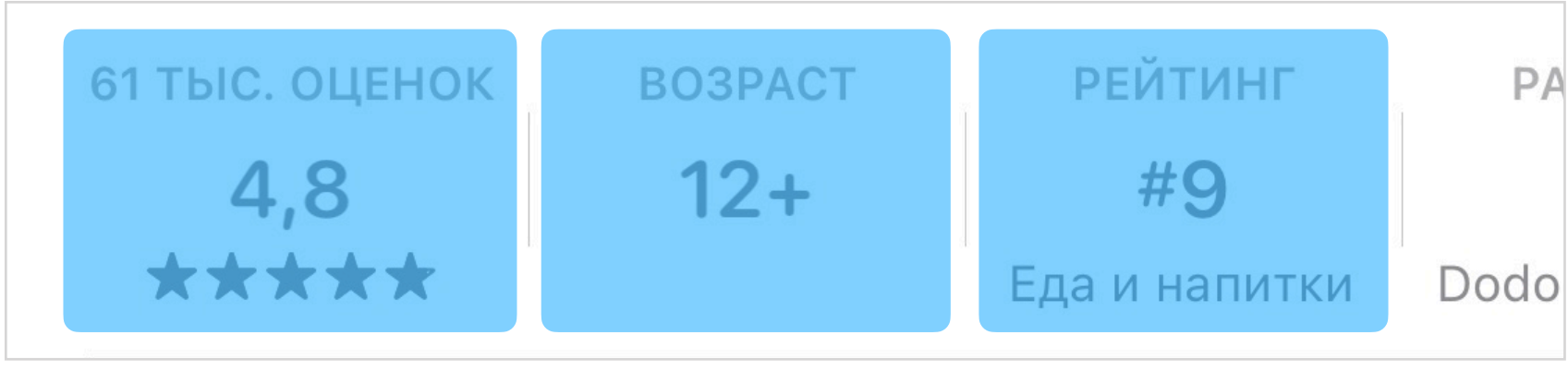
Динамически получится так:

```
titleLabel.accessibilityLabel = titleLabel.text
titleLabel.accessibilityValue = valueLabel.text
```

Теперь нам нужно скрыть label с цифрой от VoiceOver, его текст уже читается у label с названием, а дублирование нам не нужно.

```
valueLabel.isAccessibilityElement = false
```

Можно адаптировать иначе: скрыть от VoiceOver обе надписи, вместо них самому сделать «невидимый» доступный элемент и поставить ему все свойства. Мне это кажется слишком сложным, но пример с этим я покажу в главе про контейнеры.



Если дизайнер разместил значения под подписями, то порядок чтения без адаптации собьется: сначала прочитает все названия, потом все значения, понять что к чему относится будет сложно. Исправить проблему можно таким же объединением в один элемент.

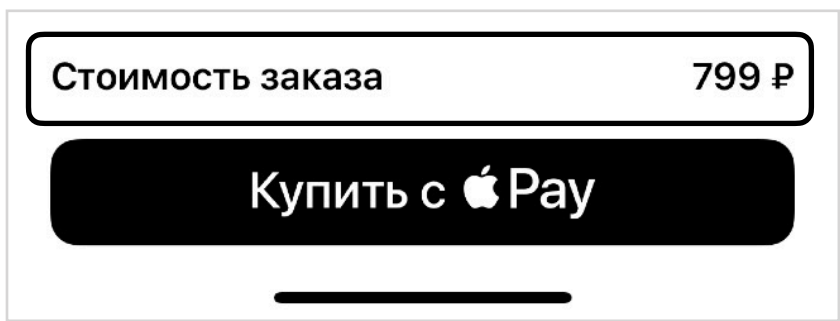
Пример из апстора показывает, что иногда надо чуть поменять текст специально для VoiceOver:

Оценка *4.8 на основе 61 тысячи оценок*

Возраст *12 плюс*

Рейтинг *9 место в «Еда и напитки»*

Рамка фокуса



При объединении двух элементов нужно увеличить рамку, чтобы она обхватывала обе подписи. Это поможет правильно определять элемент под пальцем при изучении касанием. С другой стороны, это поможет правильно ставить фокус другим технологиям доступности, например, Switch Control.

За рамку фокуса отвечает свойство `accessibilityFrame`. Рамка должна быть рассчитана в координатах экрана. Самый простой способ — это использовать уже рассчитанные фреймы и просто объединить их. Важно, чтобы ваш расчет был после того,

```
override func layoutSubviews() {
    super.layoutSubviews()

    titleLabel.accessibilityFrame =
        titleLabel.accessibilityFrame
           .union(valueLabel.accessibilityFrame)
}
```

как посчитает iOS, поэтому лучшее место — внутри `layoutSubviews()`.

Для конвертации фрейма в координаты экрана есть специальная функция `UIAccessibility.convertToScreenCoordinates`. Вызывать ее нужно так же

```
override func layoutSubviews() {
    super.layoutSubviews()

    let commonFrame = titleLabel.frame.union(valueLabel.frame)
    titleLabel.accessibilityFrame = UIAccessibility
        .convertToScreenCoordinates(commonFrame, in: self)
}
```

Упростить

UISwitch в ячейке

Свитчер — это кнопка с состоянием, VoiceOver его называет «кнопка-переключатель» и сообщает текущее состояние: включено или выключено. Переключить можно двойным тапом, VoiceOver прочитает новое состояние. В общем, по умолчанию все адаптировано.

Часто рядом со свитчером есть подпись. Семантически подпись и свитчер — это один контрол, поэтому и VoiceOver должен работать с ним как одним целым: читать подпись в качестве названия и тут же давать возможность нажать.

Должно получиться так:

label	Сообщать о бонусах, акциях и новых продуктах
value	Включено
trait	кнопка-переключатель, вкл.

Интересно, что трейт «кнопка-переключатель» незадокументирован, но доступен под номером 53:

```
let switchButtonTrait: UIAccessibilityTraits = 1 << 53
```

Как можно обработать:

— сделать всю ячейку доступным элементом,

— перенести текст надписи в лейбл `ячейки`,

— перенести трейт со свитчера,

— value брать динамически из свитчера.

Делаем всю ячейку доступной, указываем трейт как у свитчера.

```
override func awakeFromNib() {
    super.awakeFromNib()
    isAccessibilityElement = true
    accessibilityTraits = switcher.accessibilityTraits
}
```

Даем правильный label из двух строк.

```
func configure(title: String, subtitle: String) {
    titleLabel?.text = title
    subtitleLabel?.text = subtitle
    accessibilityLabel = [title, subtitle]
                         .joined(separator: ", ")
}
```

Значение должно считываться прямо из свитчера. Оно уже локализовано, но вы можете написать свой текст.

```
override var accessibilityValue: String? {
    get {
        switcher.accessibilityValue
    }
    set {}
}
```

Нажатие на ячейку должно активировать свитчер. Можно вызвать свитчер внутри функции `accessibilityActivate()`.

```
override func accessibilityActivate() -> Bool {
    switcher.accessibilityActivate()
}
```

Но есть и другой способ: переместить точку активации на свитчер. Про него в следующей главе.

```
override func layoutSubviews() {
    super.layoutSubviews()
    accessibilityActivationPoint = switcher
                                   .accessibilityActivationPoint
}
```

Упростить

Activation Point

Во всех примерах мы старались сделать всю ячейку активным элементом. Обычно, нажатие на ячейку выполняет главное действие, но иногда нужно нажать на кнопку внутри ячейки, даже если в фокусе вся ячейка.

Мы могли бы симулировать нажатие кнопки в функции `accessibilityActivate()`, но есть решение проще — передвинуть точку активации.

За координату отвечает свойство accessibilityActivationPoint, по умолчанию нажимает в центр экрана. Новое значение надо задавать в координатах экрана, поэтому:

— координату нужно задавать в методе `layoutSubviews()`, когда координаты уже известны,

— проще всего брать `activationPoint` другого элемента, чтобы не конвертировать координаты самому.

```
override func layoutSubviews() {
    super.layoutSubviews()
    accessibilityActivationPoint =
                    selectButton.accessibilityActivationPoint
}
```

Если нужно конвертировать, то можно воспользоваться функцией для фрейма, а уже у него взять центр.

```
UIAccessibility.convertToScreenCoordinates(
      selectButton.bounds,
      in: selectButton).center
```

Упростить

Практика

1. Посмотрите в своем приложении как можно адаптировать списки. Проставьте ячейкам трейт `.button`, чтобы было понятно, что на них можно нажать.
2. В каких ячейках очень много контролов и непонятно как адаптировать? Присылайте примеры в канал Dodo Mobile, помогу разобрать
3. Найдите примеры, где надписей много и их можно сгруппировать в пары из названия и значения. Объедините их в один элемент.

Вертикальные свайпы

В предыдущих главах мы подписали элементы и уменьшили их количество, так мы сделали использование приложения возможным и даже удобным. По экрану теперь можно перемещаться горизонтальными свайпами и все элементы будут читаться.

В этой главе обсудим вертикальные свайпы. С их помощью можно выполнять действия над элементами в фокусе, например:

- поменять значение внутри элемента (как в UISlider или UIStepper),
- сменить действие с активации на другое (как действия в ячейках, которые скрываются за свайпом),
- рассказать подробнее о текущем элементе.

Разберем все случаи подробно, ведь они могут сильно повысить удобство использования приложения.

Вертикальные свайпы

Трейт .adjustable

До этого момента мы использовали только один трейт поведения кнопки — **.button**. Он добавлял к описанию текст «кнопка» и по двойному тапу вызывал метод `accessibilityActivate()`.

Какие элементы сложнее чем кнопка? Например, `UIStepper`, UISegmentedControl и UISlider. На первый взгляд элементы очень разные, но работу всех трех элементов можно свести до одного правила: контрол позволяет выбрать значение. Разница только в том, что они дают разный контроль над точностью изменений:

— `UIStepper` изменяет значение на ближайшее;

— `UISlider` позволяет выбрать в широком диапазоне, но с небольшой точностью;

— `UISegmentedControl` показывает все значения, можно переключиться к любому.

Если мы еще сильнее упростим модель, то значение внутри контрола можно уменьшить или увеличить. Это довольно частый паттерн, поэтому у VoiceOver есть для этого специальный модификатор поведения — **трейт `.adjustable`.**

Для переключения между элементами мы используем горизонтальные свайпы: к следующему и предыдущему. При переключении читается лейбл.

А что, если вертикальные свайпы будут влиять на внутреннее состояние контрола? Например, мы бы могли с их помощью менять значение, записанное в `.accessibilityValue`? Тогда свайп «вверх» мог бы увеличивать количество, а «свайп» вниз уменьшать его.

Ровно так и работает трейт .adjustable: если мы добавим его к трейтам элемента, то VoiceOver добавит к описанию «элемент регулировки», в подсказке расскажет про вертикальные свайпы, а после свайпа будет вызывать методы accessibilityIncrement() и accessibilityDecrement(). Также после свайпа VoiceOver прочитает новый value в элементе. Лейбл повторно читаться не будет, потому что название элемента не меняется.

Вертикальные свайпы

UIStepper

Степпер может быть разных форм, но суть остается одна: у нас есть какое-то значение и мы влияем на него с небольшим шагом. Иногда надпись рядом, иногда внутри. В любом случае, сделаем одним элементом: назовем, отделим изменяемое значение и расскажем, как взаимодействовать с ним.

Назвать. Мы меняем количество элементов, можем так и написать: «количество».

Значение. Хватит и цифры, но можно превратить в полноценный текст: «3 события».

Трейт. Можно задать трейт .adjustable, тогда сможем регулировать количество вертикальными свайпами, а после наших изменения VoiceOver прочитает новый value.

label: Количество,
value: 3 события,
trait: элемент регулировки.

Превратим это описание в код. Нужно сделать весь степпер доступным элементом, дать ему название и указать тип `.adjustable`.

```
override public func awakeFromNib() {
      super.awakeFromNib()

      isAccessibilityElement = true
      accessibilityLabel = "Количество"
      accessibilityTraits = .adjustable
}
```

Значение можно сделать вычисляемым полем:

```
public override var accessibilityValue: String? {
      get {
            return "\(count) события"
      } set {}
}
```

Чтобы поменять значение, мы реализуем функции, которые вызываются после свайпов.

```
override public func accessibilityIncrement() {
    (count += 1).limit(by: 0…10)
}
override public func accessibilityDecrement() {
    (count -= 1).limit(by: 0…10)
}
```

Если после свайпа текст value не поменялся, то VoiceOver издаст короткий гудок — сигнал, что пользователь долистал до конца диапазона значений. Так пользователь поймет, что свайпать в этом направлении больше нет смысла.

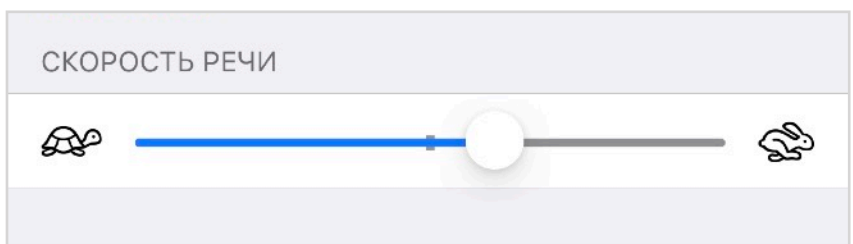
Возвращаясь к тексту рядом с контролом: `UILabel` надо скрыть от VoiceOver, а фрейм увеличить, чтобы он оборачивал и подпись и степпер.

Вместе с количеством товара часто меняется и общая стоимость. Расскажите о новой цене вместе с количеством через `.accessibilityValue`.

```
public override var accessibilityValue: String? {
    get {
        let totalPrice = count * price
        return "\(count), \(totalPrice) рублей"
    } set {}
}
```

Странно, что `UIStepper` по умолчанию не является элементом регулировки, он очень подходит для этого сценария.

UISlider



`UISlider` работает как и `UIStepper`, только изменение значения происходит с небольшим шагом, процентов в 10.

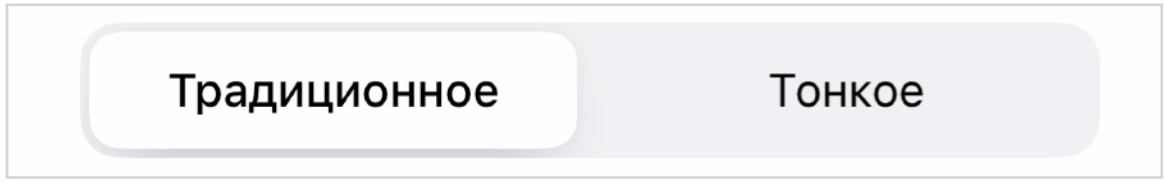
Если нужна большая точность, то можно сделать хитрое действие: тапнуть дважды и не отпускать палец при втором тапе. Таким образом, VoiceOver даст полный контроль над слайдером, будто VoiceOver выключен и получится регулировать плавно.

`UISlider` ставит себе трейт `.adjustable` по умолчанию, дополнительный код писать не нужно, вам нужно лишь дать ему название. Если рядом уже есть текст с названием, скройте его от VoiceOver, чтобы он два раза не читал одно и то же вслух.

label Скорость речи,
value 60%,
trait элемент регулировки.

Вертикальные свайпы

UISegmentedControl

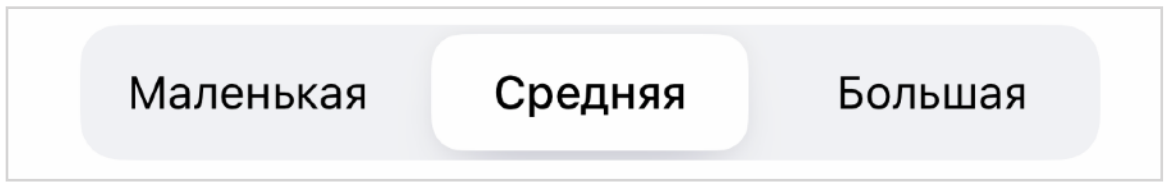


VoiceOver понимает, что `UISegmentedControl` — это группа кнопок, поэтому сообщает об их количестве:

label Традиционное,
value 1 из 2,
trait Кнопка.

Но VoiceOver ничего не знает о контексте, поэтому для всей группы стоит уточнить название через `accessibilityLabel`:

label Тесто,
value Традиционное, 1 из 2,
trait Кнопка.

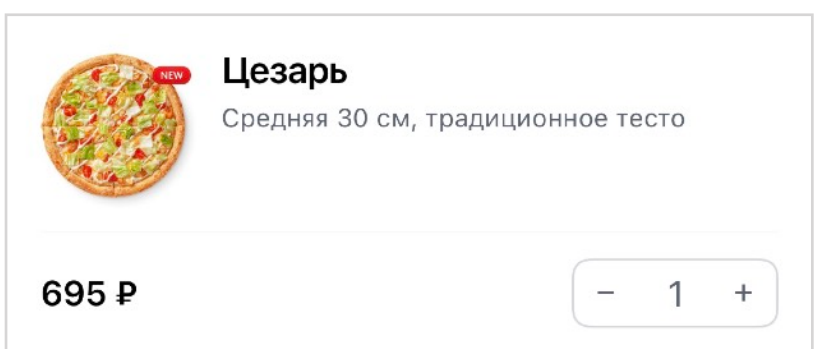


Если `UISegmentedControl` это шкала одного значения, то можно укрупнить его до `.adjustable` элемента: уменьшится количество кнопок, будет удобней перемещаться по экрану. Например, такое подходит для размера пиццы:

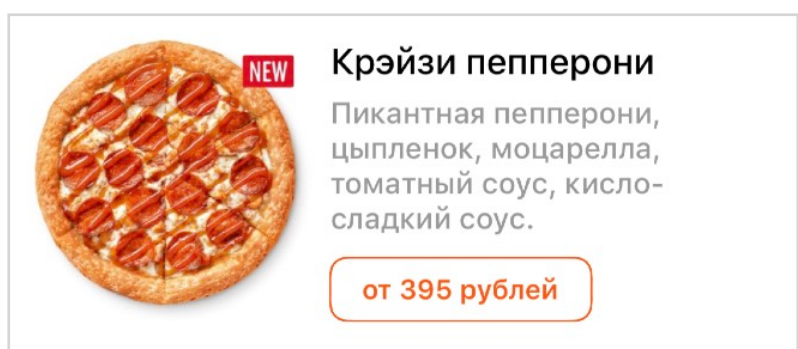
label Размер,
value Средняя, 2 из 3
trait Элемент регулировки.

Получается, что для визуально одинаковых элементов может быть разное поведение VoiceOver, потому что у них разная ментальная модель. Вау!

Контекстные действия



Ячейка в корзине



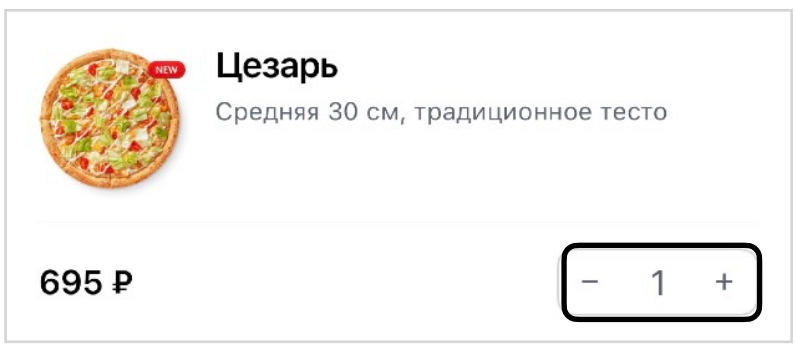
Ячейка в меню

Адаптируем ячейку из корзины. Внешне она очень похожа на ячейку меню: тоже есть название, цена и описание. Для ячейки меню мы объединили название с ценой, а описание перенесли в `value`. С корзиной можно было бы поступить так же, но у нее есть значимое отличие — контрол с количеством.

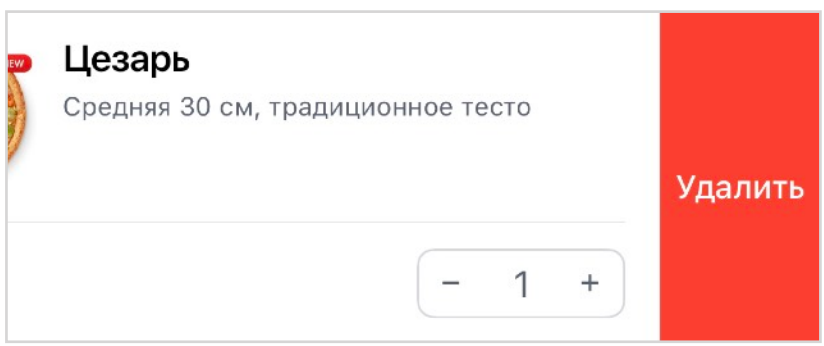
Изменяя количество, мы меняем и цену. После изменения значения VoiceOver прочитает именно .value, количество и цена должны оказаться там. Все остальное не изменяется и поэтому будет в `label`. Описание такое:

label	Цезарь, средняя 30 сантиметров, традиционное тесто,
value	1 штука, 695 рублей.

До этого мы всегда укрупняли элементы в ячейки до одного контрола, но нам важно не потерять переключатель размера, им ведь нужно как-то управлять. Есть еще один нюанс: для зрячих пользователей доступно действие «удалить», которое скрывается за свайпом. Сделать такой свайп из VoiceOver неудобно, действие оказывается недоступным.



Фокус на переключателе количества



Действие удалить за свайпом

У ячеек такие действия за свайпом вполне стандартны для iOS, поэтому для них есть специальная адаптация: когда фокус попадает на такую ячейку, VoiceOver добавляет в конце описания:

> Доступны действия: активировать (выбрано), удалить.
> Чтобы переключить действия, смахните вверх.

Это интересно: вертикальные свайпы подходят не только для управления элементом регулировки, но и могут переключать действие с «активации» (accessibilityActivate) на что-то другое. Выбрать другое действие можно вертикальным свайпом, а активировать двойным тапом.

Вернемся к выбору количества. Это два действия: уменьшить и увеличить, но выбрать можно одно из четырех:

— нажать на ячейку (может ничего не происходить, а можно, например, отредактировать товар),
— увеличить количество,
— уменьшить количество,
— удалить совсем.

Полный текст ячейки будет такой:

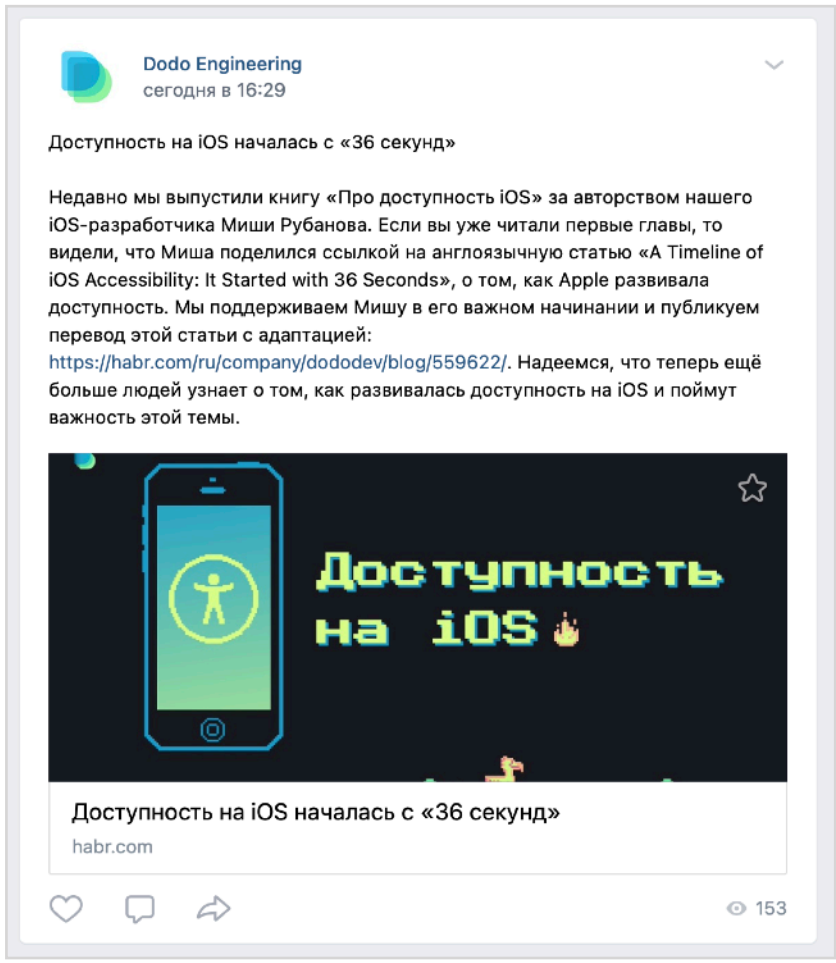
> label Цезарь, средняя 30 сантиметров, традиционное тесто,
> value 1 штука, 695 рублей.
> Доступны действия: активировать (выбрано), добавить, убавить, удалить.

Теперь надо разобраться, как эти действия добавлять. Код прямолинеен: создаем действие, даем ему имя и функцию, что обработает ее, а затем добавляем этот объект в `customActions`:

```
let increase = UIAccessibilityCustomAction(
    name: "Добавить",
    actionHandler: { [unowned self] (action) -> Bool in
        // … Increase action
        return true
    })
accessibilityCustomActions = [increase, decrease]
```

Стандартный жест «удалить» объединяется с добавленными нами и в итоге будет 3 действия.

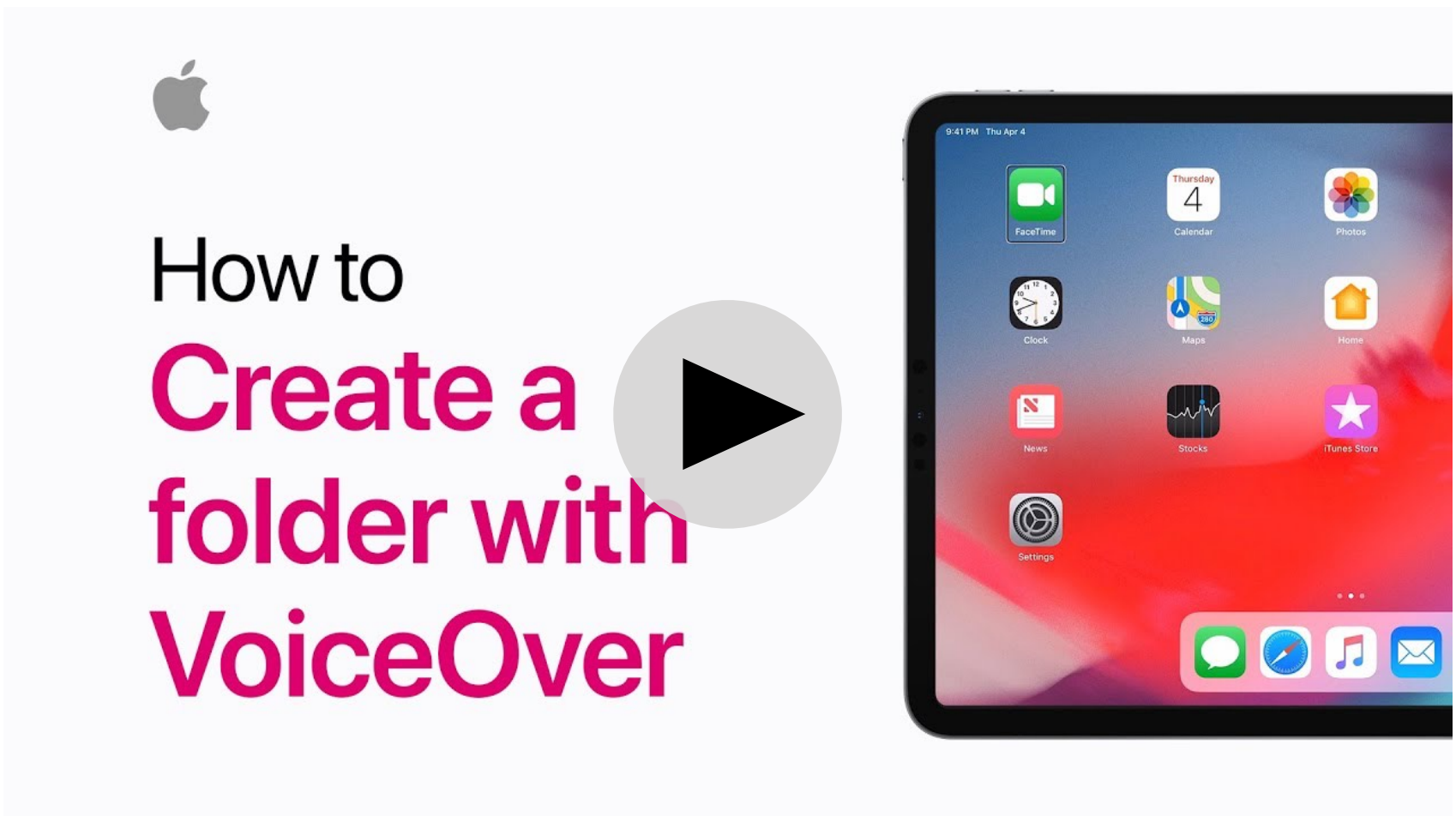
Контекстные действия — это очень сильный инструмент. По сути, все кнопки, которые оказались внутри ячейки, могут быть перенесены в `customActions`. Пост в социальной сети, сообщение в мессенджере — все их действия должны быть перенесены в `customActions`.



label Dodo Engineering,
value Доступность на iOS началась с «36 секунд»...

Доступны действия:
— открыть ссылку (выбрано),
— лайкнуть,
— прокомментировать,
— поделиться,
— перейти к «Dodo Engineering»,
— информация (по двойному тапу прочитать количество просмотров и время записи).

Действия могут быть очень сложными и даже вложенными. Пример можно посмотреть в ролике Apple про создание папок на рабочем столе:

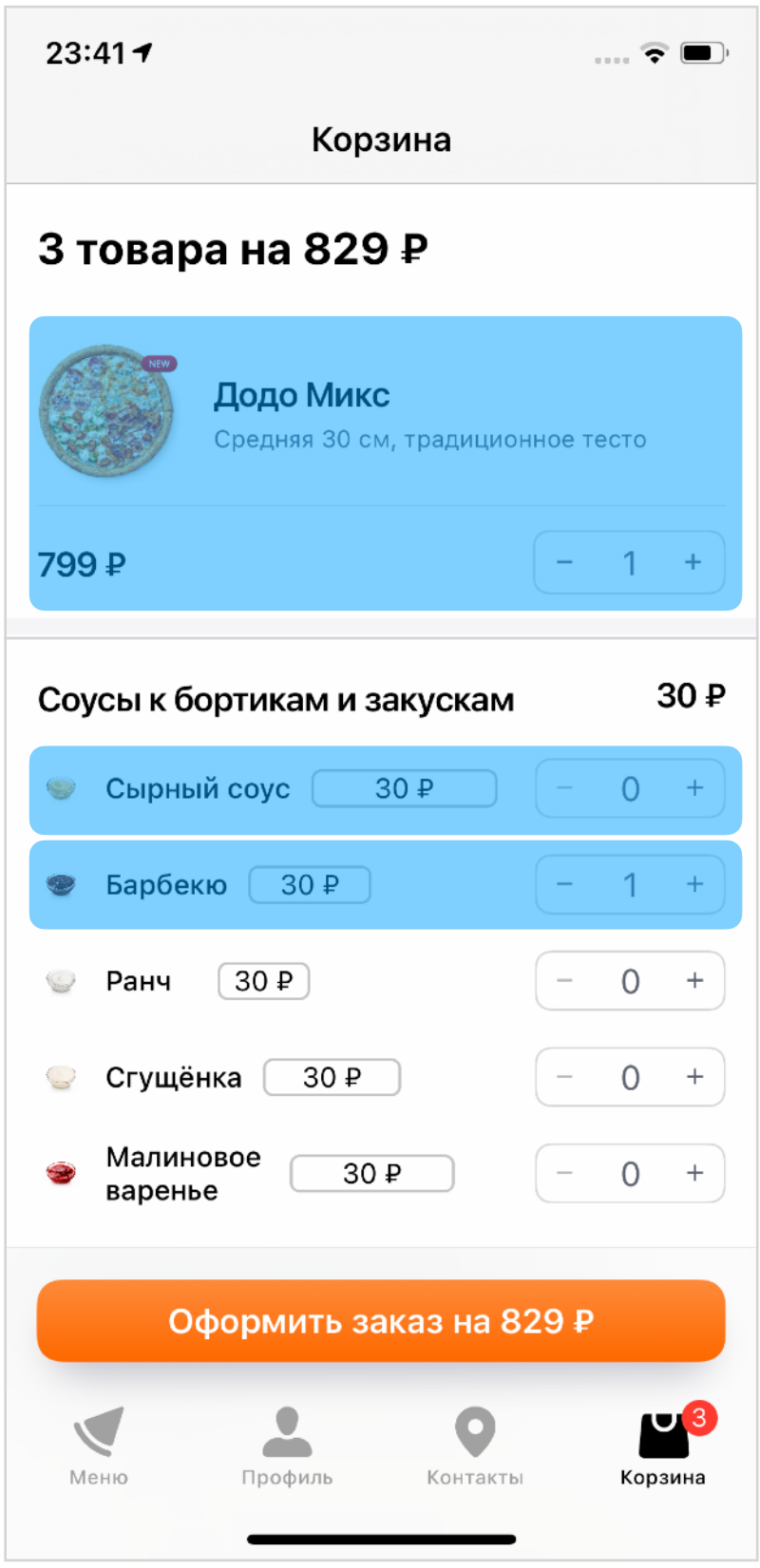


Разные действия

Если посмотреть на весь экран корзины, то каждый ее элемент можно свайпнуть вертикально:

— количество продукта меняется через контекстное действие,

— количество соусов можно адаптировать как `.adjustable` элемент.



При этом мы сильно сократим количество доступных элементов на экране: с 31 до 6, т.е. в 5 раз.

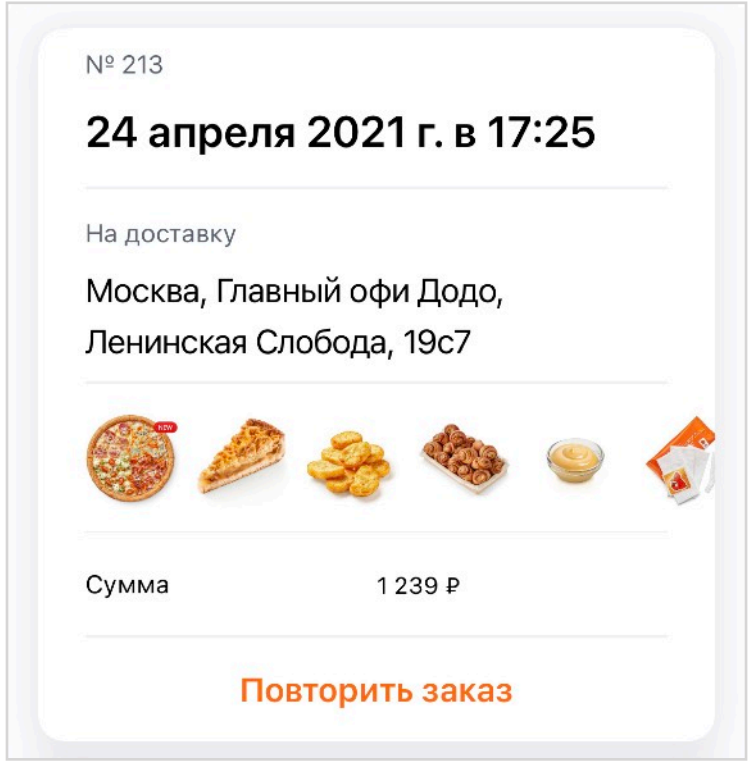
Додо Микс: Средняя 30 сантиметров,
традиционное тесто,
1 пицца, 695 рублей.

Доступны действия: добавить, убавить, удалить.

Сырный соус,
не добавлен, элемент регулировки

Барбекю,
1 штука, элемент регулировки

Большие описания



Не все ячейки можно упростить до одного элемента доступности, иногда контролов внутри ячейки слишком много. Например, в истории заказов мы выводим номер, дату, тип заказа, адрес, список из 5-6 продуктов, сумму и еще пару действий: повторить заказ и открыть детальное описание заказа.

В таком случае есть два варианта:

1. Немного укрупнить элементы, объединив в пары: номер и дату, тип и адрес, текст «сумма» и число. Проходить по ним придется по-отдельности, но это терпимо.

2. Использовать `AXCustomContent`, тогда при фокусе на ячейку VoiceOver будет читать только одно из значений, а остальные можно прослушать, свайпая вертикально. Такое описание можно посмотреть у фотографий в стандартном приложении «Фото»: iOS выводит дату, время, ориентацию фото, кто или что изображено и т.д.

Объект `AXCustomContent` состоит только из названия и значения, при этом читает он их в обратном порядке: сначала значение, а уже потом рассказывает, что оно значило.

```
AXCustomContent(label: "Сумма",
                value: "1239 рублей")
```

Может быть неудобно первый раз, но сильно ускоряет навигацию в последующие разы.

1239 рублей, Сумма

Чтобы описание заработало, нужно реализовать протокол `AXCustomContentProvider` и завести свойство `accessibilityCustomContent`.

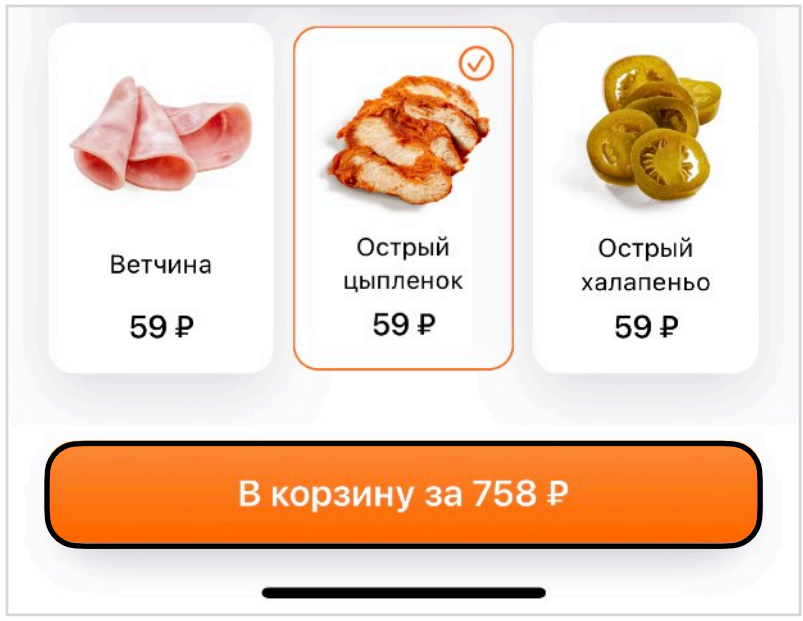
Интересно, что протокол находится во фреймворке `Accessibility`, хотя до этого весь код доступности был в `UIKit`. Не забудьте импортировать фреймворк.

```
import Accessibility

@available(iOS 14.0, *)
extension OrderHistoryItemCell: AXCustomContentProvider {

    var accessibilityCustomContent: [AXCustomContent] {
        get {
            [AXCustomContent(
                label: "Сумма",
                value: "1239 рублей"),
             AXCustomContent(
                label: "213",
                value: "Номер заказа"),
             AXCustomContent(
                label: "Дата",
                value: "24 апреля 2021 г. в 17:25"),
             AXCustomContent(
                label: "На доставку",
                value: "Москва, Главный офис Додо, Ленинская
Слобода, 19с7"),
             productsContent(products: ["Пицца Додо Микс",
"Яблочный пирог"])]
        }

        set {}
    }
```

CustomContent опирается не на порядок элементов, а на их важность. Продукты кажутся самым важным, надо читать их в первую очередь, поэтому я задал им высокий приоритет. Название продуктов я склеил в одну строчку через запятую, этого достаточно.

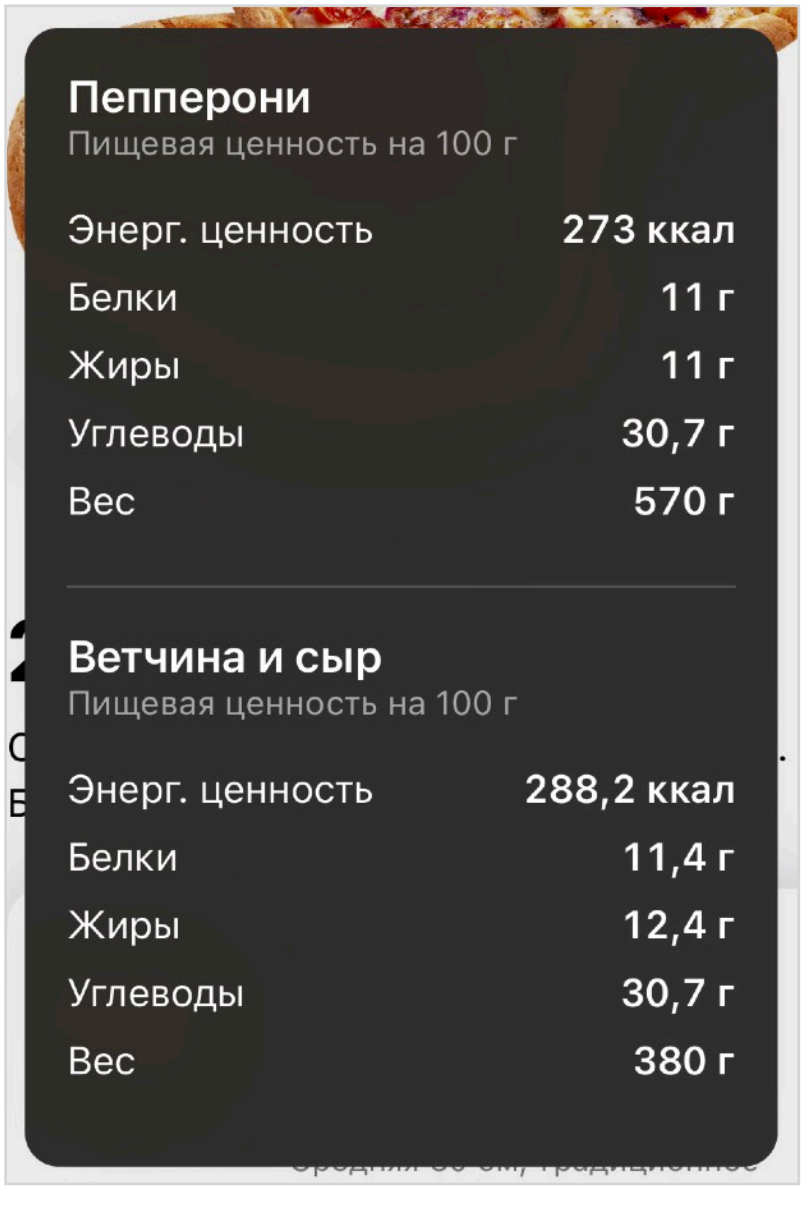
```
    private func productsContent(_ products: [String])
    -> AXCustomContent {
        let content = AXCustomContent(
            label: "Товары",
            value: products.joined(separator: ", “))

        content.importance = .high

        return content
    }
}
```

Такое описание хорошо подойдет и для подытоживающих кнопок. Например, вы настроили пиццу: выбрали размер, убрали лук, добавили побольше мяса. Перед добавлением в корзину хочется все проверить, но не проходить снова по всему экрану. Описание на кнопке добавления сильно упростит работу.



label В корзину за 754 рубля,
value Пицца Песто, средняя, тонкое тесто,
traits Кнопка.

customContent:
- Лук, убрали,
- Острый цыпленок, Добавили.



Еще один пример использования, когда данных очень много. На скриншоте энергетическая ценность для комбо-набора из нескольких продуктов. Горизонтальные свайпы будут переключать продукт, а вертикальные свайпы читать одно из описаний. С помощью свойства `.importance` можно управлять порядком продуктов, например, сначала читать вес.

Увы, `customContent` не работает вместе с `customActions`, придется оставить что-то одно.

Практика

Вертикальные свайпы — это мощный инструмент для незрячего пользователя.

1. Найдите примеры, когда можно применить трейт `.adjustable`. Так вы сможете сильно сократить количество элементов.
2. Если в ваших ячейках есть действия, то попробуйте вынести их в `customActions`.
3. Для ячеек с большими количеством контролов примените `customContent`. Не забудьте импортировать фреймворк `Accessibility`.

Навигация

Мы разобрались с основными элементами и их поведением, теперь нужно перемещаться между ними. Обычно, для этого хватает горизонтальных свайпов, но есть еще несколько удобных жестов для упрощения навигации.

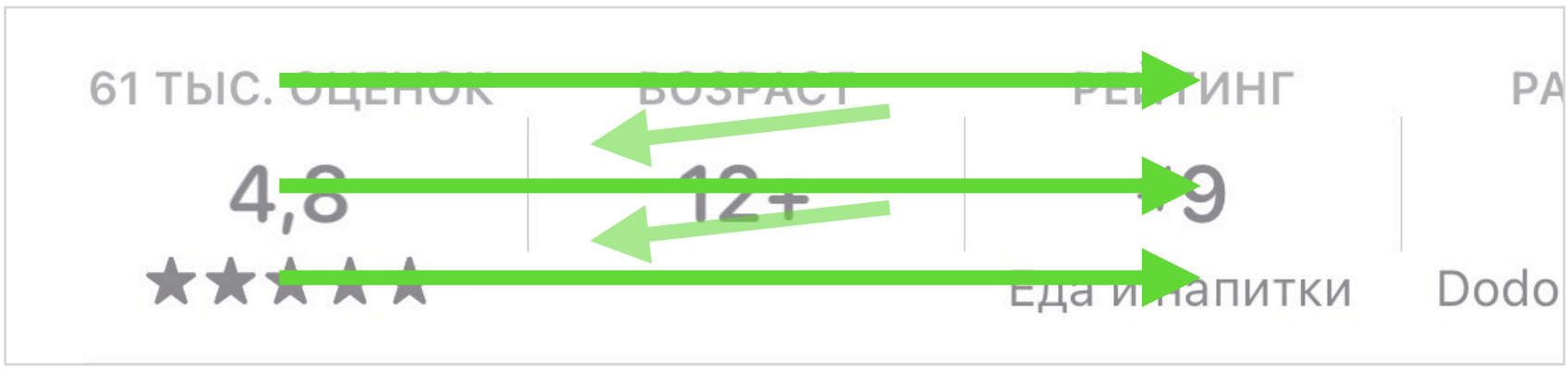
Типичные проблемы навигации: элементы перепутали порядок, фокус застрял в горизонтальной карусели, всплывающее окно нельзя закрыть.

Навигацию можно не только исправить, но и улучшить: правильно разметить заголовки, дать название важным областям интерфейса. У VoiceOver есть очень необычный «ротор», с его помощью можно управлять способом навигации по интерфейсу.

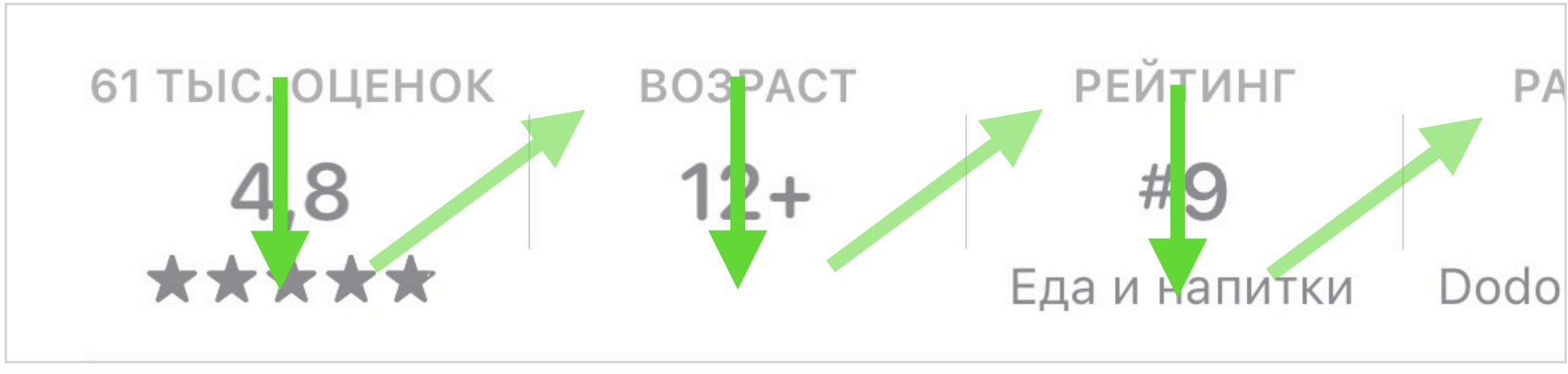
Группировка

VoiceOver читает так же как и мы: слева направо, а затем вниз к следующему элементу. При этом он не обращает внимания на иерархию вьюшек: все «вью-контейнеры», которые не являются доступными элементами, он не учитывает. Получается, что для VoiceOver вложенности у вьюшек нет.

Чаще всего такой порядок чтения будет правильным, но бывают случаи, когда элементы сгруппированы так, что при чтении перемещаться надо к ближайшему. Например, так сгруппирована панель рейтингов в Апсторе. Если не адаптировать эти элементы, то VoiceOver сначала прочитает все заголовки, затем все значения и в конце все подписи. Пользователю будет сложно понять связь между заголовком и значением.



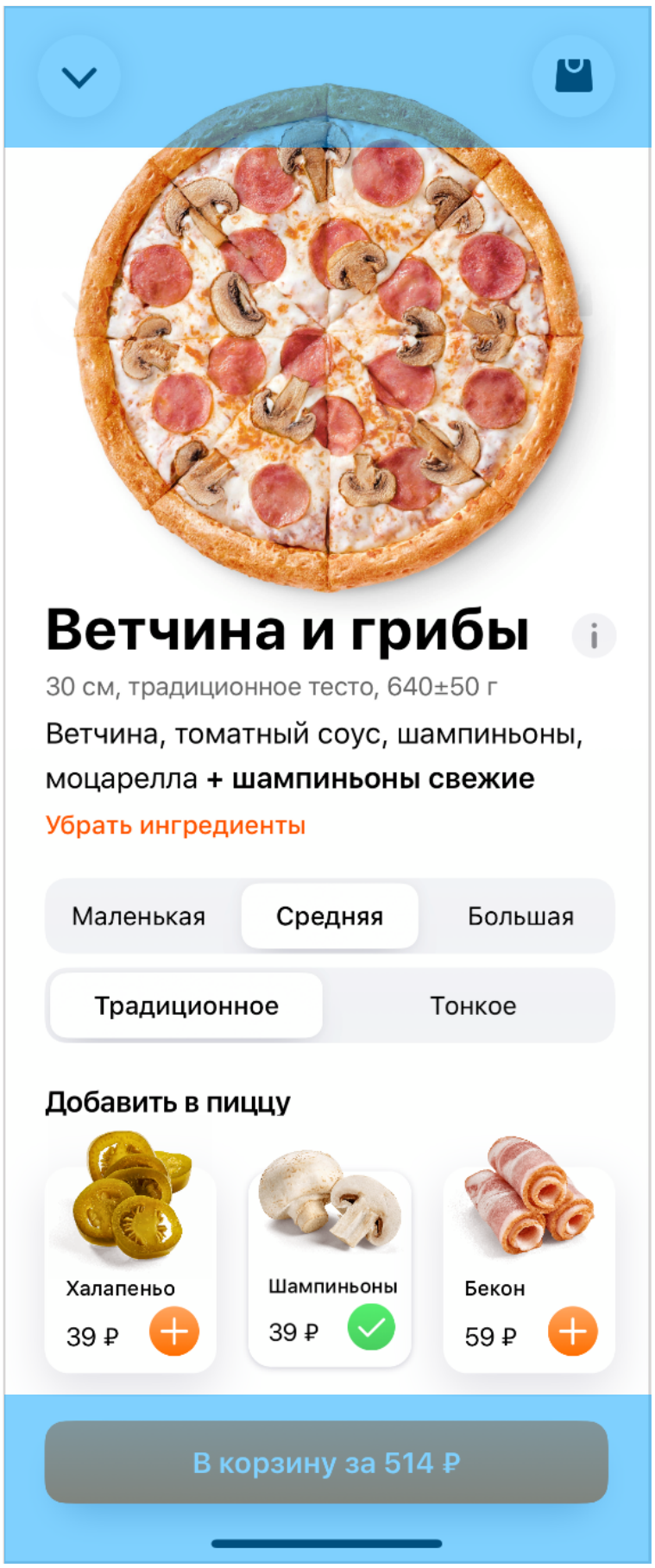
Нужно иначе: сначала прочитать все элементы внутри группы, а когда элементы закончатся, перейти к следующей группе.



Если элементы находятся в одной вью, то можно сгруппировать их, но лучше укрупнить до одного элемента, как мы делали в главе «Название+значение».

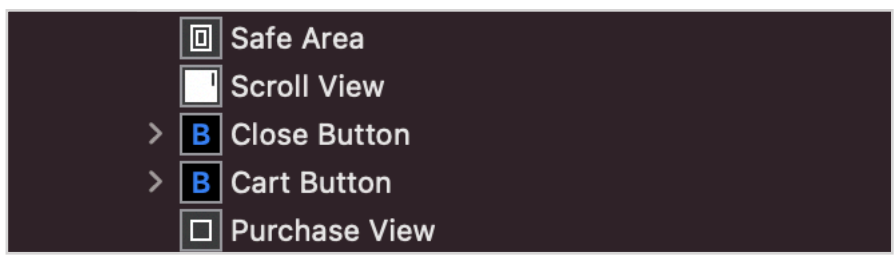
```
reviewView.shouldGroupAccessibilityChildren = true
ageView.shouldGroupAccessibilityChildren = true
ratingView.shouldGroupAccessibilityChildren = true
```

Скрол и порядок элементов



Иногда порядок элементов сбивается очень сильно. Например, в карточке товара нашего приложения кнопки «закрыть» и «в корзину» оказывались последними, хотя должны быть первыми.

Чаще всего такое бывает, когда мы делаем зафиксированные элементы поверх скрола. VoiceOver читает с первого элемента, который найдет среди subviews, им окажется нижний `UIScrollView`. После всех элементов внутри скрола он перейдет к кнопкам, которые графически могут оказаться «выше», чем элементы, что были раньше.



Увы, в UIKit нет простого способа задать порядковый номер для элементов. Единственный способ — явно описать, какие доступные элементы есть у UIView и в каком они порядке. Каждая `UIView` реализует протокол `UIAccessibilityContainer` и рассказывает о своих доступных элементах через 3 функции, чтобы ответить на вопросы: сколько всего элементов и какой под каким номером.

```
func accessibilityElementCount() -> Int
func accessibilityElement(at index: Int) -> Any?
func index(ofAccessibilityElement element: Any) -> Int
```

Вместо трех функций доступа можно использовать свойство с массивом. Работать с ним намного проще, отдельные функции понадобятся, только если вам нужно динамически описать элементы (например, в `UICollectionView`).

```
override var accessibilityElements: [Any]? {
    get {
        return [closeButton,
                cartButton,
                scrollView,
                settingsView]
    }
    set { }
}
```

Обратите внимание, я передаю весь scrollView, хотя он лишь контейнер для вложенных контролов и не является доступным элементом сам по себе. Важно передавать его, потому что он знает количество элементов внутри себя:

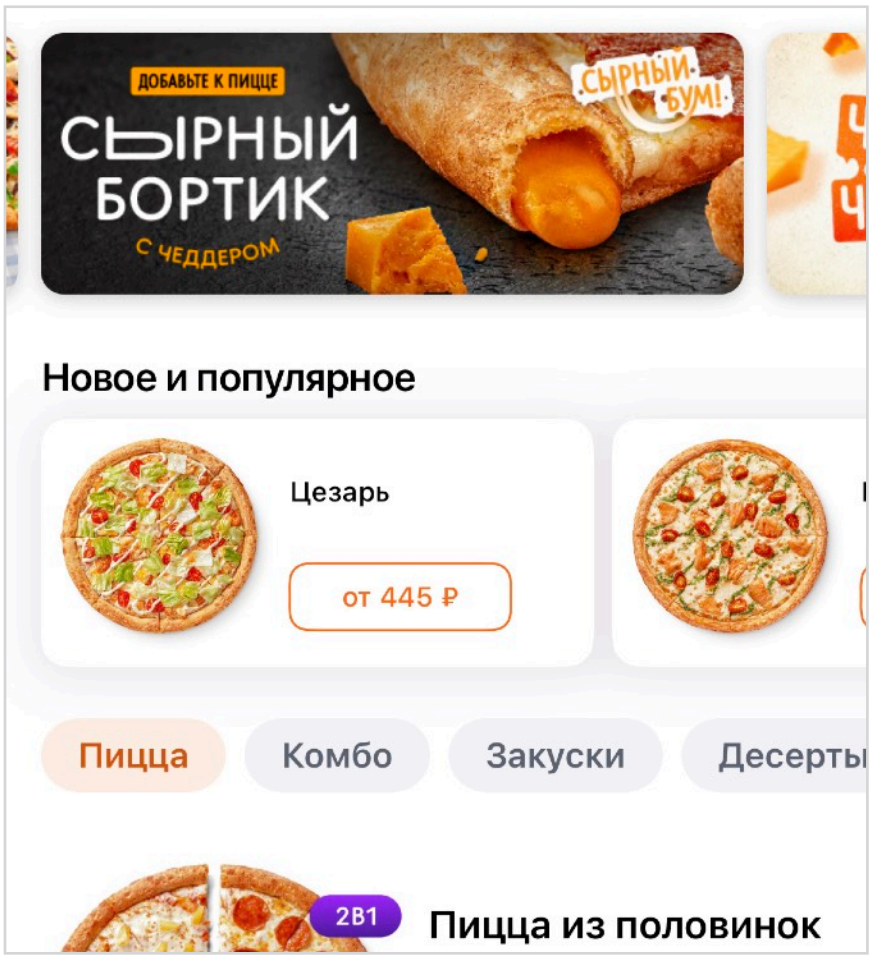
- правильно реагирует на скрол тремя пальцами (описывает количество страниц и положение фокуса после скрола);
- подскроливает список при переключении фокуса между контролами.

Нас не должно беспокоить, что внутри контейнеров есть еще другие вью, а в них есть свои элементы, VoiceOver разберется с этим самостоятельно. В каком-то смысле через `accessibilityElements` мы описываем не конечные элементы, а лишь ветви для accessibility tree.

Можно сделать еще лучше с использованием контейнеров, об этом в главе «контейнеры».

Навигация

Карусель



Карусель — это довольно частый элемент интерфейса. Как правило, элементы внутри карусели имеют один вид и на них можно нажать. Чаще всего люди перемещаются к следующему контролу свайпом. Для VoiceOver карусели становятся проблемой, так как ему сначала приходится пройти по всем элементам внутри нее и только потом получается перейти к следующей группе. Вполне возможно, что следующие контролы тоже будут каруселью и все по новой. Например, после акций идут новинки, а после них виды продуктов. Нужно сделать несколько десятков свайпов, чтобы добраться до первой пиццы.

Было бы намного удобней, если бы вся карусель была одним элементом, а ячейки внутри нее можно было бы переключать вертикальным свайпом. Активировать можно двойным тапом: открыть новый экран, выбрать элемент и т.п.

Для подобного хорошо подходит элемент регулировки и трейт `.adjustable`.

Apple предлагает интересный ход: создать доступный элемент, который будет управлять каруселью, положить его поверх коллекции, а графическую карусель спрятать от VoiceOver. Попробуем.

Создадим новый UIAccessibilityElement. Можно типизировать контейнер и принимать только коллекции, ведь именно из них мы будем брать все нужные параметры.

```
init(accessibilityContainer: UICollectionView, title: String) {
    self.collectionView = accessibilityContainer

    super.init(accessibilityContainer: accessibilityContainer)

    accessibilityTraits = .adjustable
    accessibilityLabel = title
    accessibilityFrameInContainerSpace = collectionView.frame
}
```

Конструктор предполагает, что фрейм у коллекции уже посчитан, поэтому код можно вызывать только после лейаута, например, во viewDidAppear(:). Вызываю его внутри UICollectionViewController, чтобы все что есть в accessibilityElements заменилось на один новый контрол.

```
override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)
    view.accessibilityElements = [AccessibilityCarousel(
            accessibilityContainer: collectionView,
            title: “Акции")]
}
```

В первую очередь карусель — это горизонтальный UICollectionView и скрол по нему должен переключать на следующий элемент. Скролить его можно тремя пальцами: пусть показывает следующий элемент, ставит фокус на него и читает описание.

```
override func accessibilityScroll(_ direction:
UIAccessibilityScrollDirection) -> Bool {
    if direction == .left {
        return collectionView.accessibilityScrollForward()
    } else if direction == .right {
        return collectionView.accessibilityScrollBackward()
    }
    return false
}
```

После скрола VoiceOver будет читать value, для него будем брать центральную ячейку, склеивать её label и value и выводить как описание текущего элемента. В идеале надо добавить и описание позиции: «3 из 12».

```
override var accessibilityValue: String? {
    get {
        guard let cell = centerCell()
        else { return nil }

        return [cell.accessibilityLabel,
                cell.accessibilityValue]
            .compactMap { $0 }
            .joined(separator: ", ")
    }
    set {}
}
```

Коллекция имеет тип `.adjustable`, поэтому нужно реализовать методы, которые вызовутся после свайпов. По вертикальному свайпу пусть тоже скролит до элемента.

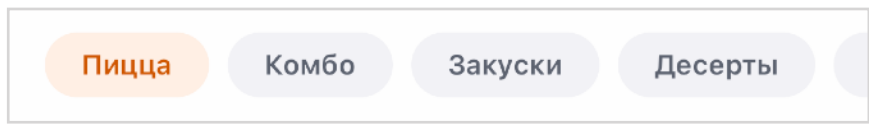
```
override func accessibilityIncrement() {
    collectionView.accessibilityScrollForward()
}

override func accessibilityDecrement() {
    collectionView.accessibilityScrollBackward()
}
```

Функция скрола простая: берем центральный элемент, считаем какой будет следующим и скролим до него. Скрол назад выглядит так же.

```
@discardableResult
func accessibilityScrollForward() -> Bool {
    guard let cell = centerCell(),
          let path = indexPath(for: cell),
          let nextPath = nextPath(for: path)
    else { return false }

    scrollAndFocus(path: nextPath)
    return true
}
```

После свайпа мы должны проскролить до следующей ячейки и центрировать ее. Дополнительно мы опираемся на свойство `selectionFollowFocus` — это стандартное свойство позволяет выбирать элемент, как только он попадает в фокус. Такое поведение пригодится, чтобы моментально выбирать тип продукта, а вот для акций это не нужно.

```
func scrollAndFocus(path: IndexPath) {
    scrollToItem(at: path,
                 at: .centeredHorizontally,
                 animated: true)

    if selectionFollowsFocus {
        selectAsUser(path: path)
    }
}
```



Сразу выбирать после скрола можно с помощью свойства selectionFollowsFocus

Если `selectionFollowFocus` не установлен, то выбрать элемент можно двойным тапом через функцию `accessibilityActivate()`.

```
override func accessibilityActivate() -> Bool {
    guard let path = collectionView.centerPath()
    else { return false }

    collectionView.selectAsUser(path: path)
    return true
}
```

О сложном поведении контрола расскажет стандартная подсказка для `.adjustable`: «смахните вверх или вниз одним пальцем, чтобы изменить значение».

На самом деле, .adjustable-карусель будет мешать UI-тестам, но об этом будет в разделе про Voice Control.

Полный код со вспомогательными функциями можно посмотреть на гитхабе.

На этом критичные проблемы навигации внутри страницы заканчиваются и можно посмотреть, как незрячие еще могут перемещаться по странице.

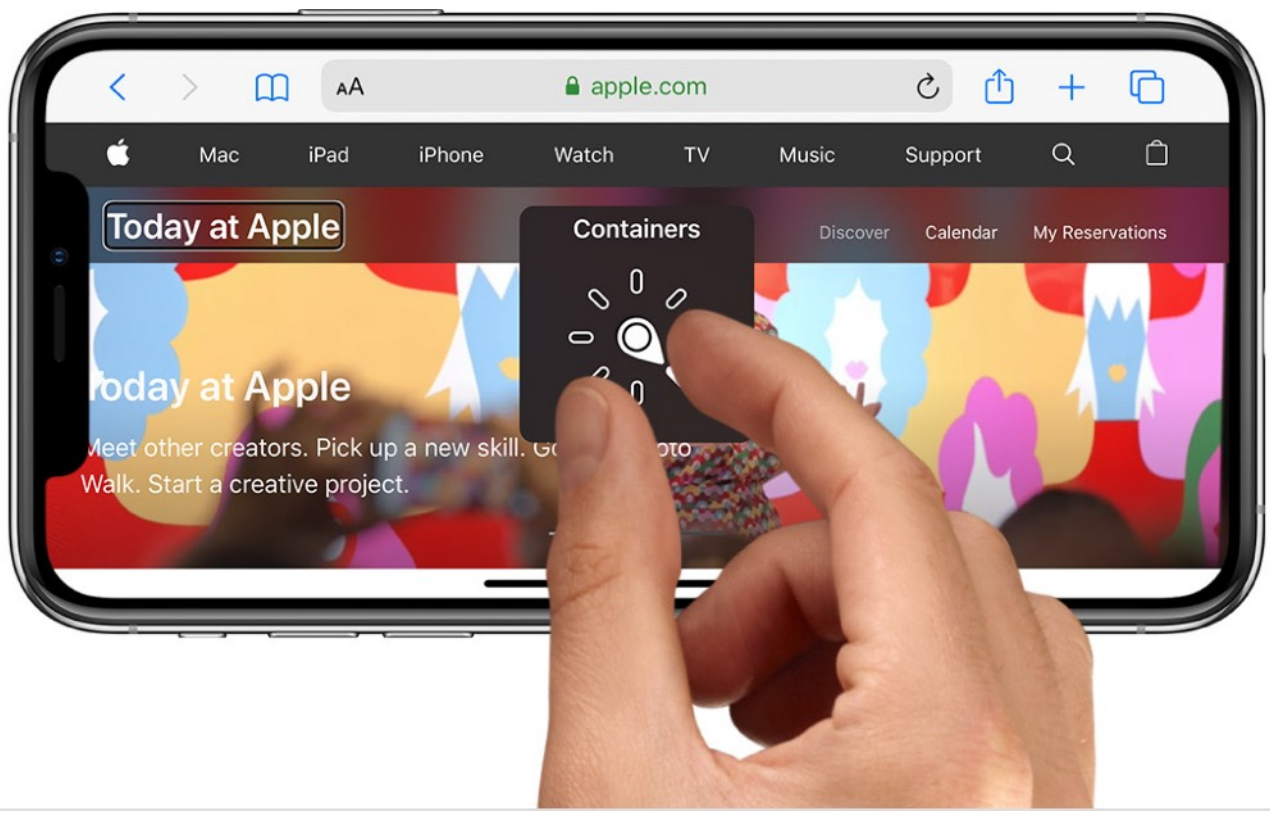
Ротор

До этого мы разбирали перемещение между соседними контролами с помощью горизонтальных свайпов. Переходить по всем контролам подряд может быть утомительно, особенно, если знаешь, что нужный пункт где-то далеко.

Довольно часто нужно перейти к следующей группе элементов, заголовку, кнопке или ссылке. Для ускорения навигации между отдаленными элементами Apple предлагает использовать вертикальные свайпы. Свайпнул вверх — перешел к следующему, вниз — к предыдущему.

Иногда нужен переход по кнопкам, иногда по ссылкам, а иногда по заголовкам. Режим такой навигации нужно выбирать, сделать это можно через «ротор».

Ротор — специальный контрол, отвечающий за режим навигации. Вызвать его можно, повернув двумя пальцами по экрану, будто поворачиваете фото на телефоне или меняете громкость у музыкального пульта.



Продолжая крутить, вы будете менять режим навигации. Режимы будут меняться в зависимости от контекста и настроек пользователя, вариантов очень много. Остановившись на нужном режиме (например, заголовки), посвайпайте вертикально, фокус будет переходить между удаленными друг от друга элементами.

Ротор универсален и точно так же работает и в Apple Watch.

Через ротор можно не только выбирать режим навигации, но и менять некоторые настройки, например, скорость чтения или язык.

Поддержать ротор несложно, достаточно отметить заголовки и дать название областям интерфейса. Для особых случаев можно создавать и свой ротор.

Может показаться, что вертикальные свайпы перегружены: они регулируют `.adjustable` элементы, контекстные действия, дополнительное описание и текущую настройку ротора, но на практике с таким конфликтом встречаешься не так часто.

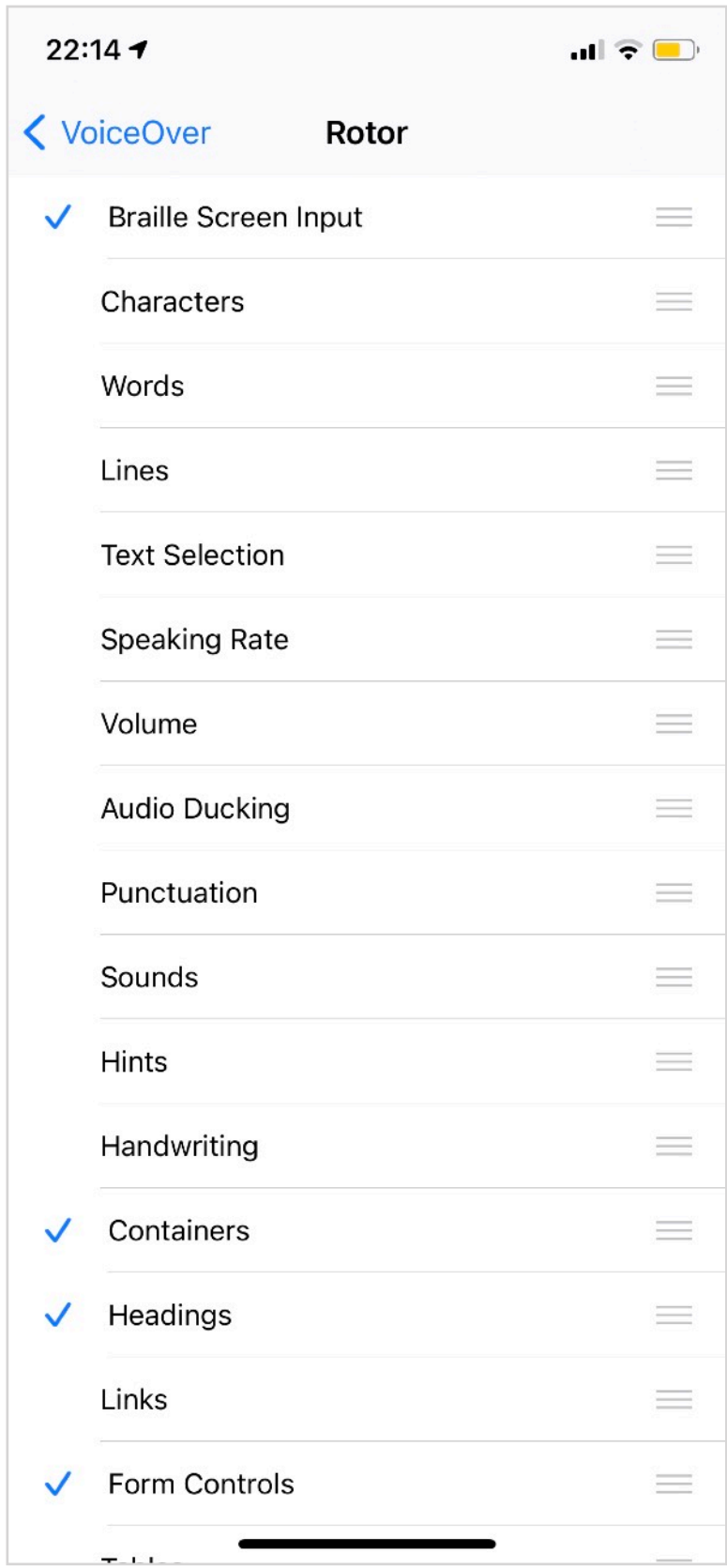
Режимы навигации в роторе

По экрану:
- заголовки,
- контейнеры,
- теги html (landmarks),
- подобный текущему,
- вертикальная навигация,
- статичный текст.

По тексту:
- буквы,
- слова,
- строки.

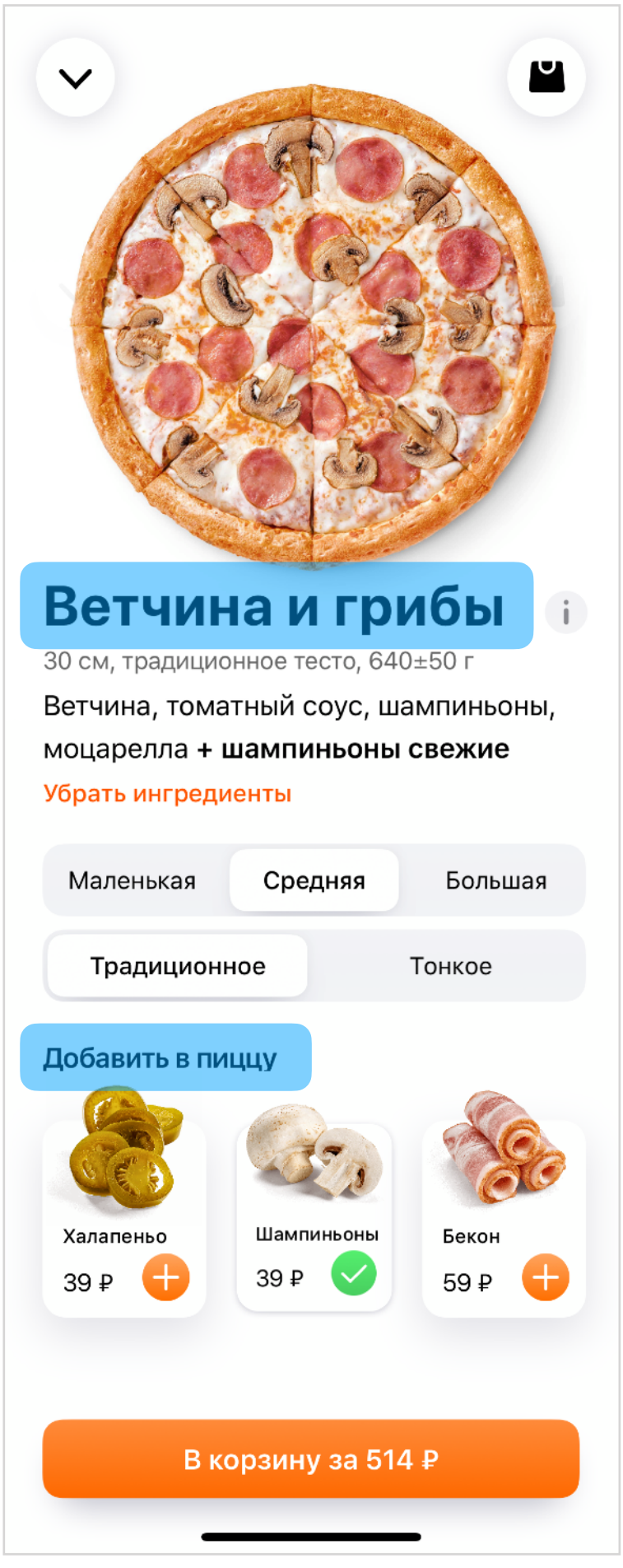
HTML:
- таблицы,
- списки,
- кнопки,
- формы (кнопки и меню),
- текстовые поля,
- поля поиска,
- картинки.

По ссылкам:
- все,
- посещенные,
- непосещенные,
- якоря на странице.



Режимов ротора много.
В iOS 14 я насчитал 40 опций
https://support.apple.com/ru-ru/HT204783

Заголовки



Для удобной навигации по странице надо отметить заголовки, они задают структуру страницы и разделяют ее на понятные блоки.

Заголовки отмечаются через трейт и обычно ставятся только лейблам:

```
titleLabel
    .accessibilityTraits = .header
```

После этого к описанию элемента добавится текст «заголовок». Например:

Ветчина и грибы, заголовок.

Встретив заголовок, мы поймем, что все следующие элементы относятся к нему. Более того, можно открыть ротор, повернув двумя пальцами, выбрать режим «Заголовки» и переключаться между ними вертикальными свайпами.

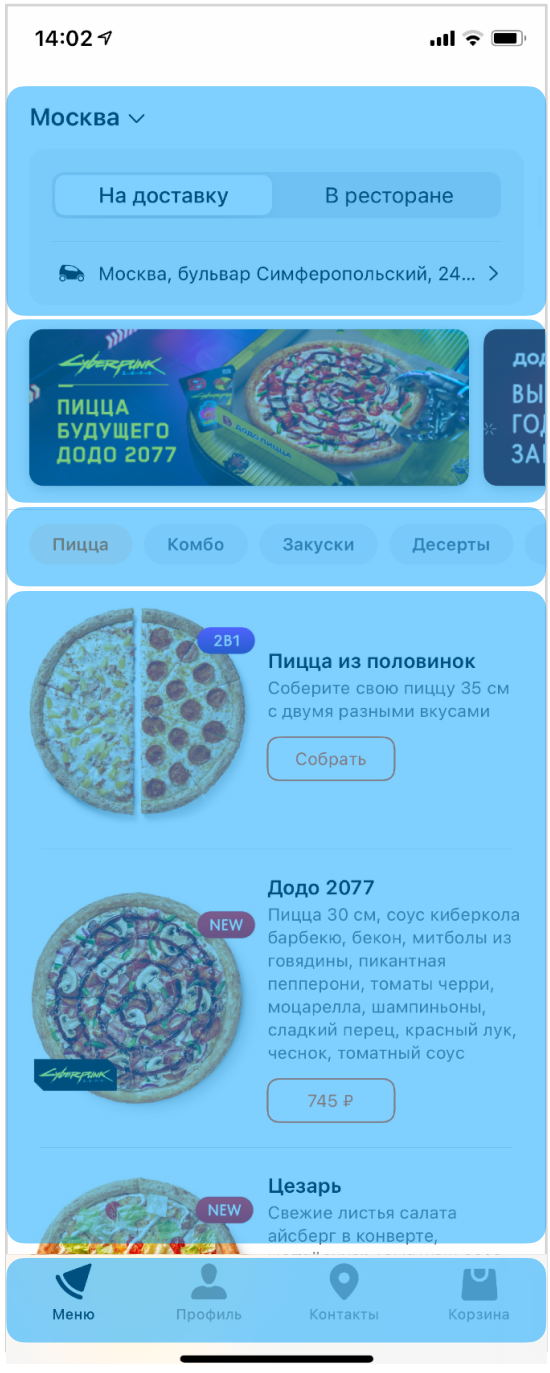
Интересно, что UITableView умеет переключать между заголовками секций, хоть они и находятся за пределами экрана.

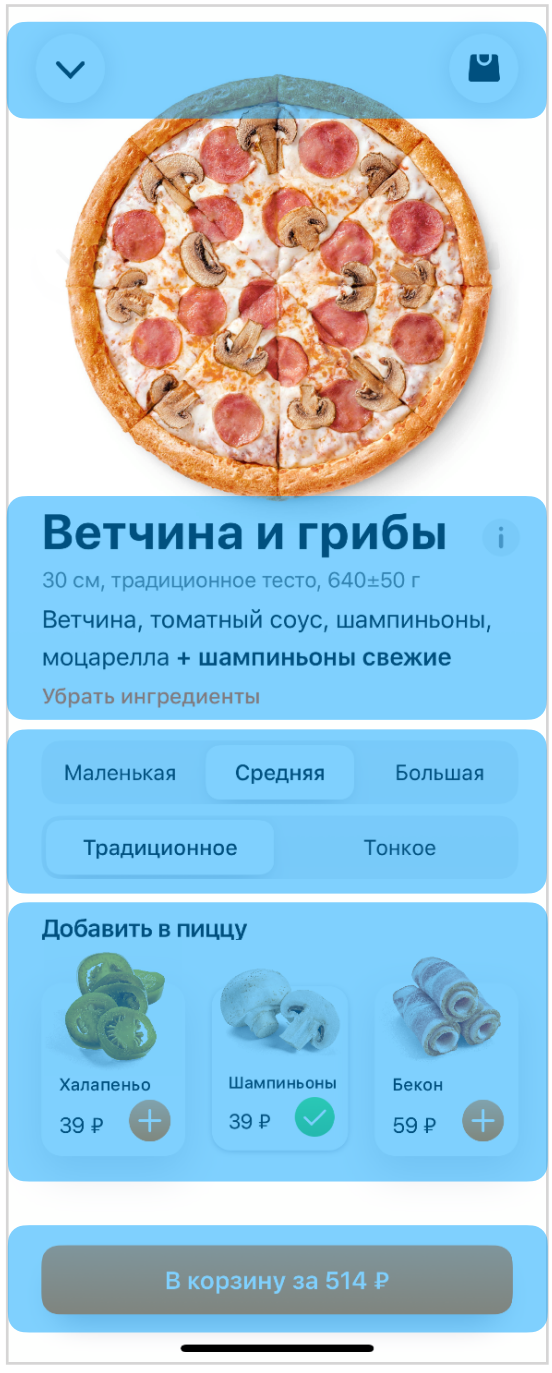
В отличие от html, в интерфейсе iOS нет деления заголовка по уровням: экраны меньше, контент короче, структура проще.

Контейнеры

Помимо заголовков можно дать единое название для области из контролов. Обычно на экране есть несколько явных зон.

Если у `UIView` есть название, то VoiceOver прочитает его, как только вы попадете на первый элемент области. В отличие от заголовка, контейнер расскажет о себе, даже если вы поставите фокус на последний элемент.





В меню и карточке продукта есть явные контейнеры:

— тип заказа, акции, вид продуктов, меню, панель навигации;
— навигация, описание, тесто, добавки, итого.

Правильно обработать контейнер легко:

— он не должен быть доступным элементом, а это поведение UIView по умолчанию;
— надо дать название через label;
— указать, что вью должна быть не только визуальным, но и семантическим контейнером для контролов внутри.

```
// isAccessibilityElement = false
accessibilityLabel = “Тип заказа”
accessibilityContainerType = .semanticGroup
```

Помимо .semanticGroup еще может быть .list и .table. **List** заметного изменения в поведение не добавил, но название контейнера произносит.

Для **.table** адаптация сложнее. Сначала ваш UIView должен реализовать UIAccessibilityContainerDataTable, чтобы рассказать о количестве строк, столбцов и какой UIAccessibilityContainerDataTableCell элемент находится по номеру строки и столбца.

Скорее всего, реализовывать UIAccessibilityContainerDataTableCell будут ячейки или их модели: ячейка должна знать, в каком она столбце и колонке. Если ячейка объединяет несколько строк/столбцов, то это тоже можно указать.

После этого навигация по таблице станет проще: в роторе можно будет выбрать режим «ряды», тогда вертикальные свайпы будут менять ряд ячеек, а горизонтальные будут переключать по столбцам.

Контейнер без вью

Иногда нам нужно объединить контролы, даже если у них нет общей вью. Чтобы не усложнять иерархию для VoiceOver можно добавить невидимый доступный элемент, сделать его контейнером, поместить контролы в него, а его в иерархию доступных элементов. Такой контейнер я сделал, чтобы объединить кнопки вверху карточки продукта: «закрыть» и «в корзину».

```
let navigationContainer = UIAccessibilityElement
                                  (accessibilityContainer: view!)
navigationContainer.isAccessibilityElement = false
navigationContainer.accessibilityLabel = "Навигация"
navigationContainer.accessibilityContainerType = .semanticGroup
navigationContainer.accessibilityElements
              = [closeButton, cartButton]
```

Фрейм нужно считать после лейаута, например, в `layoutSubviews`.

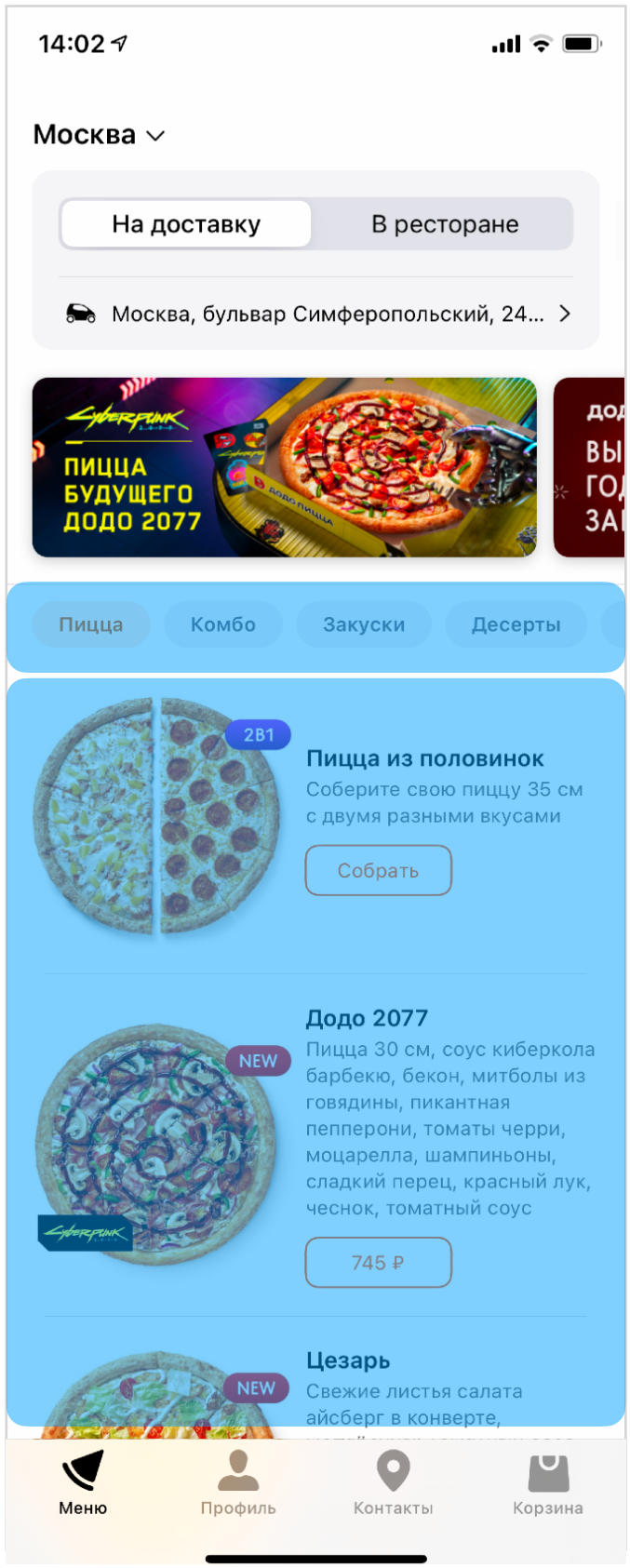
```
override func layoutSubviews() {
      super.layoutSubviews()

      navigationContainer.accessibilityFrameInContainerSpace =
            closeButton.accessibilityFrame
                  .union(cartButton.accessibilityFrame)
}
```

Свойство `accessibilityFrameInContainerSpace` есть только у `UIAccessibilityElement`. Оно позволяет задавать фрейм не в координатах экрана, а относительно родительского элемента, что удобно, потому что мы не всегда знаем, где контрол будет находиться в будущем.

Удобно задавать порядок контролов на уровне контейнеров, решая сразу обе проблемы: и контейнеры названия получат, и порядок исправится.

```
override var accessibilityElements: [Any]? {
    get { return [navigationContainer,
                  scrollView,
                  settingsView] }
```

CustomRotor



Заголовки и контейнеры — это базовая часть быстрой навигации. Если у вас случай посложнее, то можно адаптировать его и добавить в ротор.

Например, у нас в меню длинный список продуктов, он сгруппирован по виду: пицца, комбо, закуски и т.д. Я полистал несколько пицц, выбрал нужную и хочу перейти к напиткам.

В базовой адаптации я вынужден перейти к переключателю категорий, при этом я могу перейти только касанием, ведь свайп назад будет листать список до начала. Было бы замечательно, если бы свайпом вверх в любом месте списка можно было бы сменить категорию.

Добавить такое поведение можно с помощью `UIAccessibilityCustomRotor`. Работа с ним простая:

- создать ротор, присвоить его свойству `customRotors`,
- дать название для ротора и функцию, что обработает результат,
- написать эту функцию. Она определит, какой следующий элемент должен попасть в фокус.

Теперь, когда фокус попадет на ячейку меню, VoiceOver скажет: «Используйте ротор для доступа к объекту: Тип продуктов». Так пользователь узнает о нем и сможет настроить ротор для навигации.

На вход функции передается объект, который хранит текущий элемент в фокусе и направление, в котором его надо поменять. По данным можно понять, на каком следующем элементе надо сфокусироваться. Доскролим до него без анимации.

```
func handleRotorResult(
    predicate: UIAccessibilityCustomRotorSearchPredicate)
            -> UIAccessibilityCustomRotorItemResult? {
    guard let newPath = nextSection(
                        direction: predicate.searchDirection)
    else { return nil }

    scrollToRow(at: newPath, at: .middle, animated: false)
}
```

После скрола нужно взять первую ячейку, которая оказалась видна на экране. Вернем её как результат работы функции, тогда фокус сам переместится на нее.

```
guard let firstCell = cellForRow(at: newPath)
else { return nil }

return UIAccessibilityCustomRotorItemResult(
        targetElement: firstCell, targetRange: nil)
```

Увы, фокус не всегда срабатывает, поэтому я сам сообщаю, на какой элемент надо поставить фокус с помощью оповещение перед `return`. Про оповещения подробно расскажу через пару страниц.

```
UIAccessibility.post(notification: .layoutChanged,
                     argument: firstCell)
```

Остается только добавить ротор. Важно, что добавлять ротор надо не в переключатель типов продукта, а в саму таблицу, тогда ротор будет доступен, пока фокус находится на ячейке таблицы.

```
override func viewDidLoad() {
    super.viewDidLoad()

    tableView.accessibilityCustomRotors
              = [tableView.sectionRotor(title: "Тип продуктов")]
}
```

Полный код для ротора получился таким:

```
// extension UITableView

func sectionRotor(title: String) -> UIAccessibilityCustomRotor {
    UIAccessibilityCustomRotor(
        name: title,
        itemSearch: handleRotorResult)
}

func handleRotorResult(
    predicate: UIAccessibilityCustomRotorSearchPredicate)
-> UIAccessibilityCustomRotorItemResult?
{
    guard let newPath = nextSection
                        (direction: predicate.searchDirection)
    else { return nil }

    scrollToRow(at: newPath, at: .middle, animated: false)

    guard let firstCell = cellForRow(at: newPath)
    else { return nil }

    UIAccessibility.post(notification: .layoutChanged,
                         argument: firstCell)

    return UIAccessibilityCustomRotorItemResult(
        targetElement: firstCell, targetRange: nil)
}
```

Код ротора

Код вспомогательных функций можно найти на гитхабе.

Оповещения

До этого момента мы лишь описывали интерфейс и подсказывали VoiceOver как с ним работать. На определенном уровне адаптации у вас появится желание влиять на работу VoiceOver, например, командовать им.

Для таких команд есть механизм оповещений `UIAccessibility.Notification`. Оповещение можно отправить VoiceOver, чтобы повлиять на его работу: прочитать текст, обновить `accessibility tree` или поставить фокус на другой элемент. Всего существует 6 видов оповещений, но для работы нужно разобраться с тремя базовыми:

— просто прочитать текст,
— оповестить об обновлении текущего экрана,
— рассказать о замене экрана на новый.

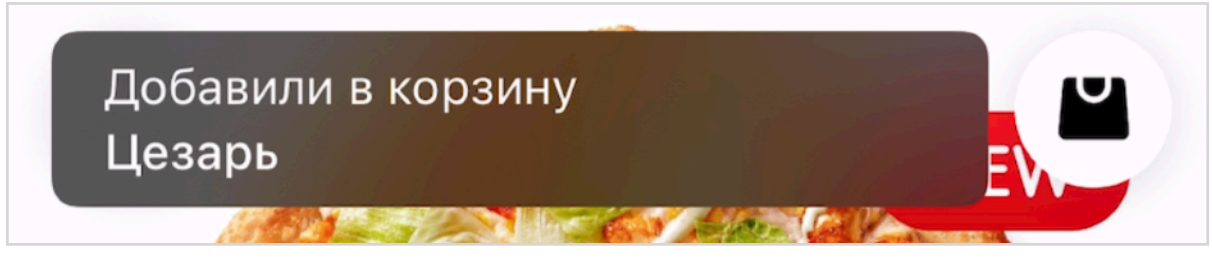
Обычно оповещение вызывается в ответ на какое-то действие пользователя.

`.announcement`

Просто прочитает текст, его нужно передать в качестве параметра:

```
UIAccessibility.post(notification: .announcement,
                     argument: "Заказ оплачен")
```

Чаще всего это оповещение нужно, чтобы сообщить о результате какого-то процесса: заказ готов, загрузка завершена, товар добавлен в корзину и т.п.

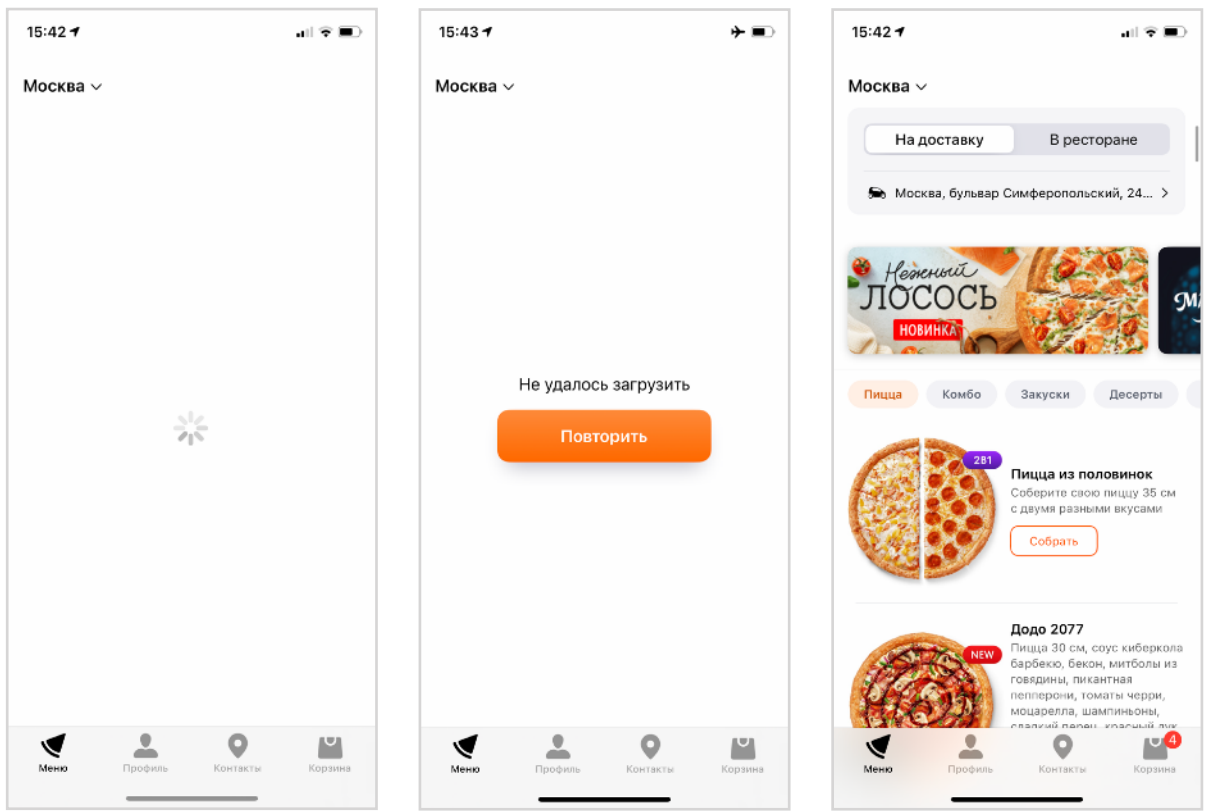


`.screenChanged`

Оповещение `.screenChanged` уведомляет VoiceOver о смене экрана. В качестве параметра можно передать не только текст, но и `UIView`, тогда VoiceOver поставит фокус на него. При использовании стандартных методов `present` или `push` iOS сама отправляет это оповещение при открытии нового экрана. Так VoiceOver узнает, что нужно перестроить `accessibility tree` и поставить фокус на первый элемент нового экрана. Если вы сделали свой способ показа, то вам нужно самостоятельно вызвать это оповещение и передать в него первый элемент экрана.

```
UIAccessibility.post(notification: .screenChanged,
                     argument: view().content)
```

Это же оповещение пригодится и при смене состояния экрана, чтобы сообщить VoiceOver, что все элементы на экране уже другие.



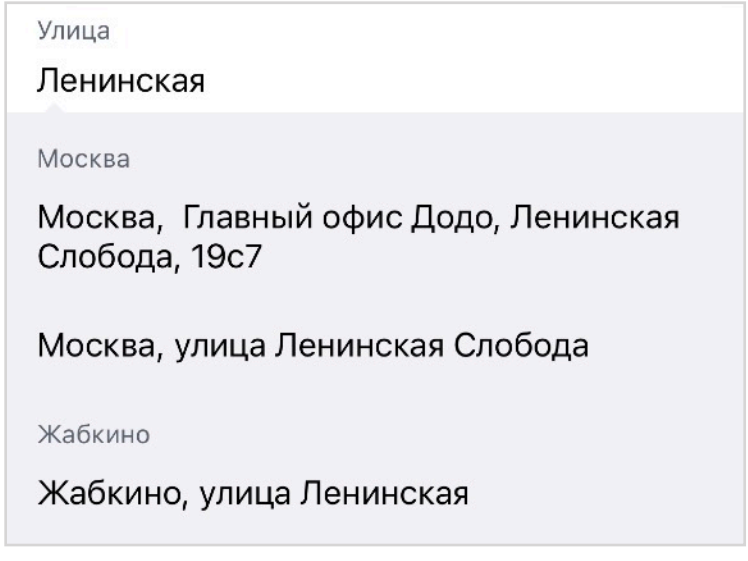
Три состояния экрана: загрузка, повтор и данные

`.layoutChanged`

Если экран сменился не целиком, а лишь обновилась его часть,(например, когда появляются или скрываются контролы, меняется порядок элементов), VoiceOver должен перестроить свое дерево, чтобы фокус правильно перемещался между элементами. Можно поставить фокус на новый элемент и ограничиться этим, а можно передать текст, который нужно прочитать.

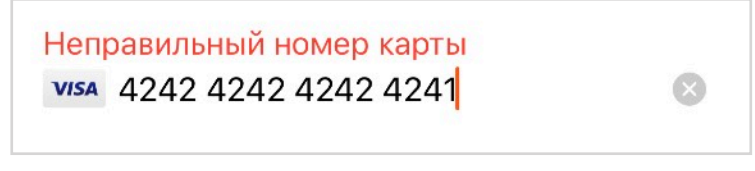
```
UIAccessibility.post(notification: .layoutChanged,
                     argument: view().content)
```

Это оповещение пригодится для ошибок, выпадающих списков, переключения состояния кнопок и т.п.



Во время ввода текста появились подсказки

Проверили номер карты, сообщаем что можно перейти к другому полю…



… или читаем ошибку

`.pageScrolled`

iOS вызывает такое оповещение после скрола, чтобы рассказать о появившихся элементах. Для описания таблицы используют текст из метода `accessibilityScrollStatus(for:)`. Пример в следующей главе «Списки в таблице».

`.pause` и `.resumeAssistiveTechnology`

Эти оповещения можно вызвать для отключения и включения VoiceOver, например, если вы показываете короткий ролик.

Параметром нужно передать идентификатор технологии, которую вы останавливаете: `UIAccessibilityNotification`**`SwitchControlIdentifier`** или `UIAccessibilityNotification`**`VoiceOverIdentifier`**. После `pause` обязательно надо вызвать `resume`, иначе VoiceOver или Switch Control не включатся.

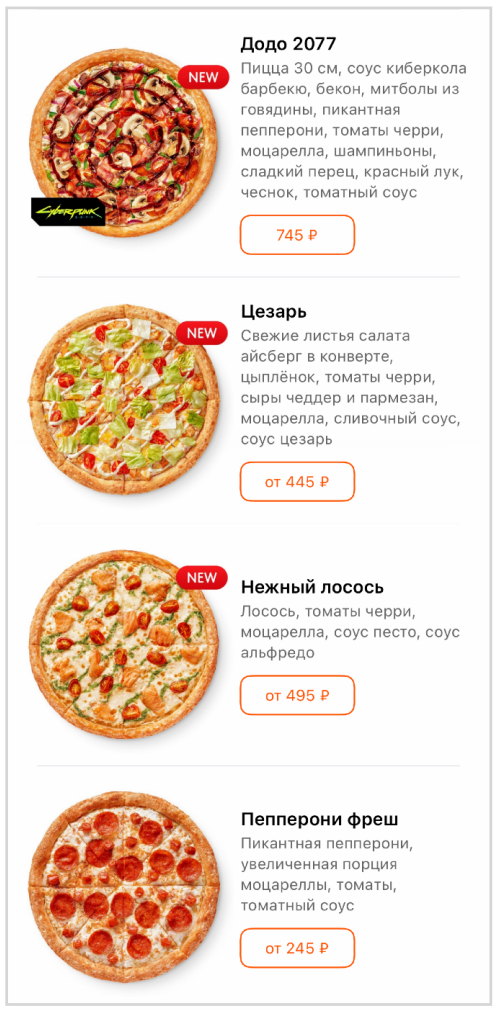
Использовать стоит только в особых случаях и на короткое время, слишком легко дезориентировать пользователя, ведь мы отключаем его главный инструмент взаимодействия с телефоном.

Списки в таблице

Скролить списки можно свайпом тремя пальцами. После свайпа вызывается метод `accessibilityScroll(:)`, на входе у него будет направление, а на выходе статус: обработался жест или нет.

```
func accessibilityScroll(_ direction:
                         UIAccessibilityScrollDirection) -> Bool
```

После скрола нужно понять, какой контент на экране появился. По умолчанию VoiceOver читает «Страница 3 из 12». Если `UIView` реализует протокол `UIScrollViewAccessibilityDelegate`, то сможет кратко рассказать о контенте, который появился на экране.



Додо 2077

Цезарь

Нежный лосось

Внутри этого метода можно пройтись по всем видимым моделям, взять от них название продуктов и прочитать их через запятую.

Тогда после скрола VoiceOver прочитает:

> Додо 2077, Цезарь, Нежный лосось, Пепперони фреш.

Код для такого описания:

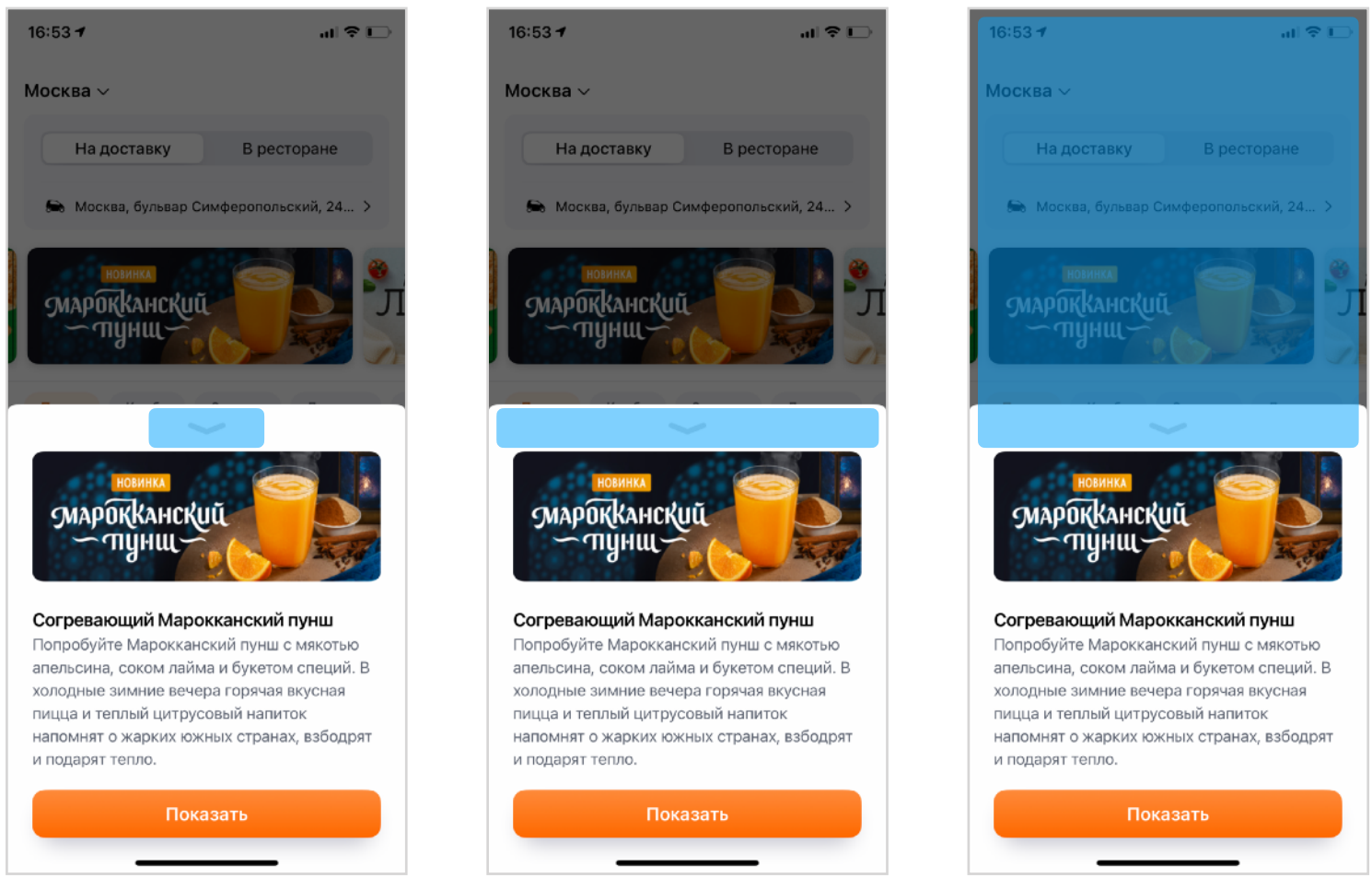
```
extension MenuTableViewController:
UIScrollViewAccessibilityDelegate {

    func accessibilityScrollStatus(for scrollView: UIScrollView)
    -> String? {
        guard let visiblePaths = tableView.indexPathsForVisibleRows
        else { return nil }

        let titles = visiblePaths.map { itemList[$0.row].name }
        let summary = titles.joined(separator: ", ")

        return summary
    }
}
```

Можно сделать еще лучше: добавить описание количества элементов выше и ниже экрана, какие сейчас видны, какой тип продуктов на экране:

> Пиццы: Додо 2077, Цезарь, Нежный лосось, Пепперони фреш.
> С 12 по 15 из 120.

Назад

Кнопка закрытия очень важна: если пользователь оказался не на том экране, то он должен легко вернуться на шаг назад. В `UINavigationController` кнопка оказывается самой первой на экране и фокус попадает именно на нее.



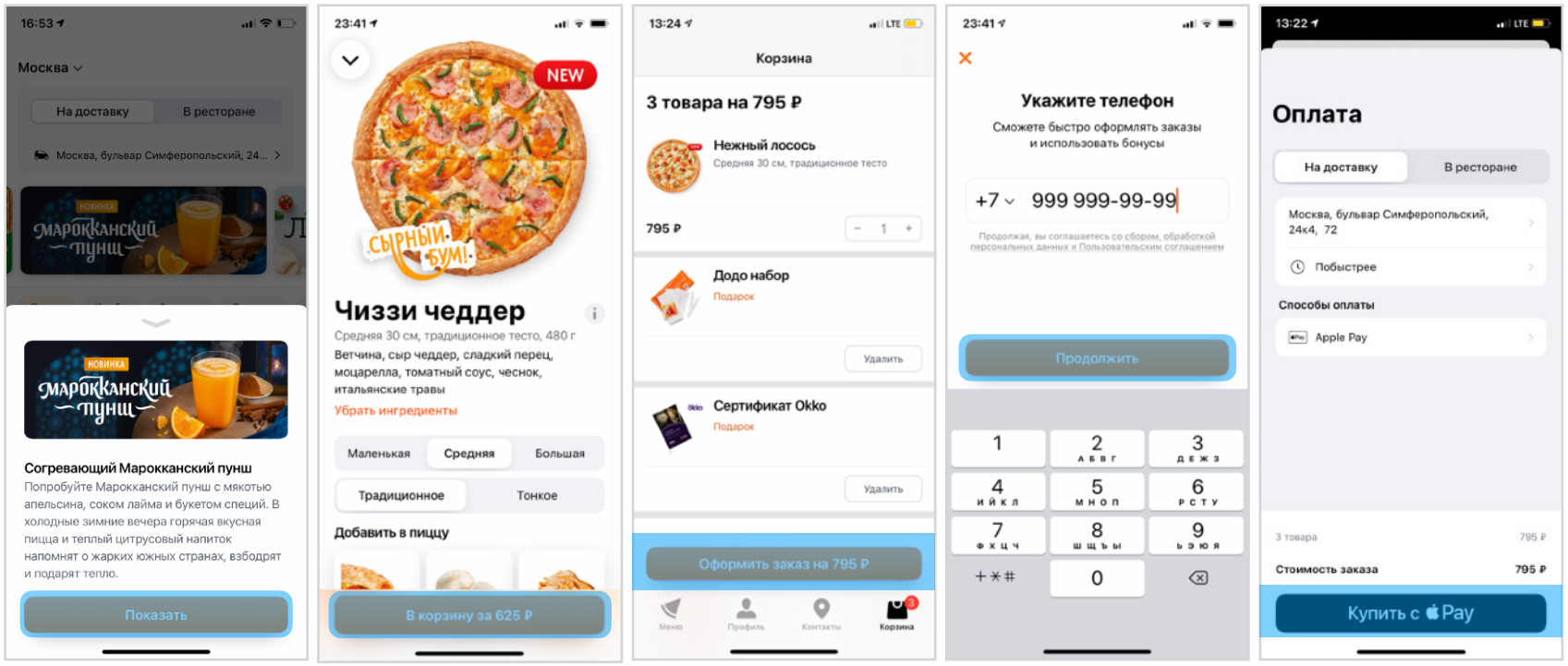
Кнопку закрытия нужно делать очень большой

Закрывающую кнопку стоит делать как можно больше: идеально, если во всю ширину экрана. Закрытие настолько важно, что для него даже есть специальный жест — **Scrub.** Выполняется он двумя пальцами: нужно как бы «почесать экран»,нарисовав на экране букву Z. После скраба вызывается метод `accessibilityPeformEscape()`. Жест работает в нативных модальных окнах, `UINavigationController` и `UIPopoverController`, но если рисуете свое модальное окно, то не забудьте поддержать жест:

```
override func accessibilityPerformEscape() -> Bool {
    dismiss(animated: false)
    return true
}
```

Вперед

Навигация «Вперед» это большая редкость, даже в браузере этой кнопкой редко пользуются. Но при адаптации приложения эта кнопка может оказаться очень полезной. Например, я точно знаю, что на текущей странице я закончил и хочу отправить данные. Чтобы нажать заветную кнопку, мне нужно пролистать все контролы, это неудобно.



Другой сценарий, когда мне нужно быстро выполнить главное действие: ответить на звонок, поставить музыку на паузу, продолжить ее слушать, отправить сообщение и т.п.

В обоих случаях хочется не свайпать, чтобы попасть фокусом на нужный элемент, а сразу выполнить действие. В VoiceOver такое действие называется **Magic Tap,** выполняется двойным тапом двумя пальцами.

В коде все просто: есть отдельная функция, нужно в ней вызвать ваше действие, а затем вернуть `true`, чтобы VoiceOver понял, что вы обработали этот жест.

```
override func accessibilityPerformMagicTap() -> Bool {
    apply()
    return true
}
```

С Magic Tap есть большая проблема: о нем сложно узнать. Одни незрячие могут вообще не знать о его существовании, а другие не догадываться о том, что программа его поддерживает и каким образом.

О том, что действие кнопки вызывается через Magic Tap можно написать в подсказке кнопки, которую он вызывает. Было бы утомительно слышать подсказку каждый раз, когда фокус попадает на кнопку, поэтому все подсказки можно отключить в настройках и незрячие часто это делают. Но в итоге, пользователь может не узнать о жесте в вашей программе, потому что подсказки он выключил!

Жаль, что в VoiceOver нет двух режимов подсказок: системные, которые пользователи и так знают, и особенные подсказки от приложения, потому что тут-то и бывают уникальные случаи. Пока приходится либо слушать все, либо не слушать ничего.

На самом деле, все не так страшно. Обычно незрячие включают подсказки через Ротор, чтобы познакомиться с новой программой. Когда и если подсказки надоедают, они их выключают.

```
accessibilityHint = "Доступно через Magic Tap. Нажмите дважды двумя пальцами чтобы активировать."
```

Подсказки

У подсказок есть несколько своих правил:

– Подсказка кратко описывает результат.

– Лучше начинать подсказку с глагола, пропустить субъект. Все в третьем лице.

– Подсказка начинается с заглавной буквы, заканчивается точкой.

Включит песню.

– Название действия опускается, остается только результат.

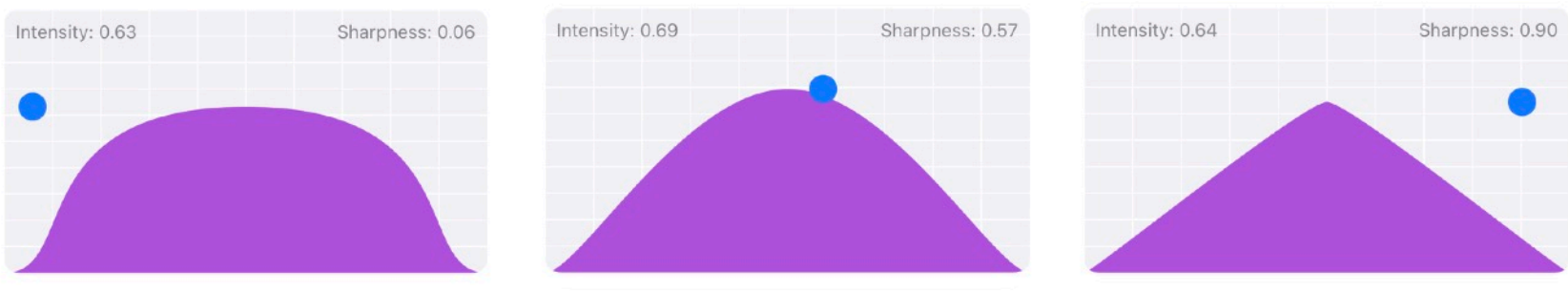
~~Нажми чтобы включить песню.~~ Включит песню.

– Подсказки могут отключить, поэтому не размещайте в них важную информацию,.

```
nextButton.accessibilityHint = "Нажимая «Далее», вы
соглашаетесь со сбором и обработкой персональных данных"
```

Подсказки хорошо подойдут для тех случаев, когда вы нетривиально используете возможности VoiceOver, как только что мы разобрали с Magic Tap. Если есть полезное действие по 3D-тач, то тоже расскажите о нем.

Обычно подсказки нужны при сложном поведении. Например, я работал с контролом, который управлял параметрами тактильной отдачей. В контроле была точка, ее высота регулировала силу отдачи, а горизонтальное положение ее остроту. Крутость контрола в том, что можно двигать точку и раз в секунду ощущать, как чувствуются текущие величины.



По горизонтали изменяется острота тапа, а по вертикали его интенсивность

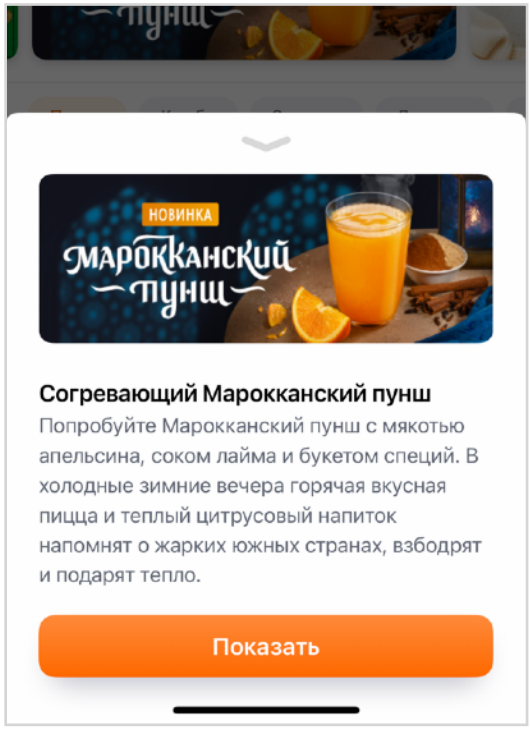
Для VoiceOver нужно тоже дать возможность перемещать эту точку, чтобы интерактивность и постоянная обратная связь не потерялись. Активировать ее перемещение можно так же, как и плавную регулировку у слайдера: тапнуть дважды, на втором тапе оставить палец на экране, так VoiceOver перейдет в режим точного жеста и можно будет двигать точку.

О таком сложном жесте нужно рассказать, лучше всего для этого подходит `hint`. О способе управления можно рассказать так:

> Тапните дважды и зажмите на втором тапе для плавной регулировки. По вертикали изменяет силу отдачи, по горизонтали ее остроту.

Такое описание противоречит правилам из предыдущего раздела, но здравый смысл и польза для человека бывают важнее простых правил.

Модальное окно



Акции в виде модального окна

Чаще всего появление нового экрана целиком закрывает предыдущий экран. Но если вы сделали всплывающую панель и под ней остался предыдущий интерфейс, то для хорошей работы VoiceOver нужен дополнительный код.

Интересно, что если вы под всплывающую панель подложили `UIView` во весь экран, то iOS поймет, что окно над ним модальное, поставит на него фокус, не даст ему выйти за пределы и вам останется только добавить жест закрытия. Рассчитывать на такое поведение не стоит: оно может отличаться в разных версиях iOS.

Что нужно сделать с модальным окном:

1. После появления поставить фокус на первый элемент окна. Хорошо, если это будет кнопка закрытия, как кнопка «назад» у `UINavigationController`.

```
UIAccessibility.post(notification: .screenChanged,
                     argument: view().closeButton)
```

2. Запретить фокусу уходить из окна. Для этого есть специальное свойство `accessibilityViewIsModal`.

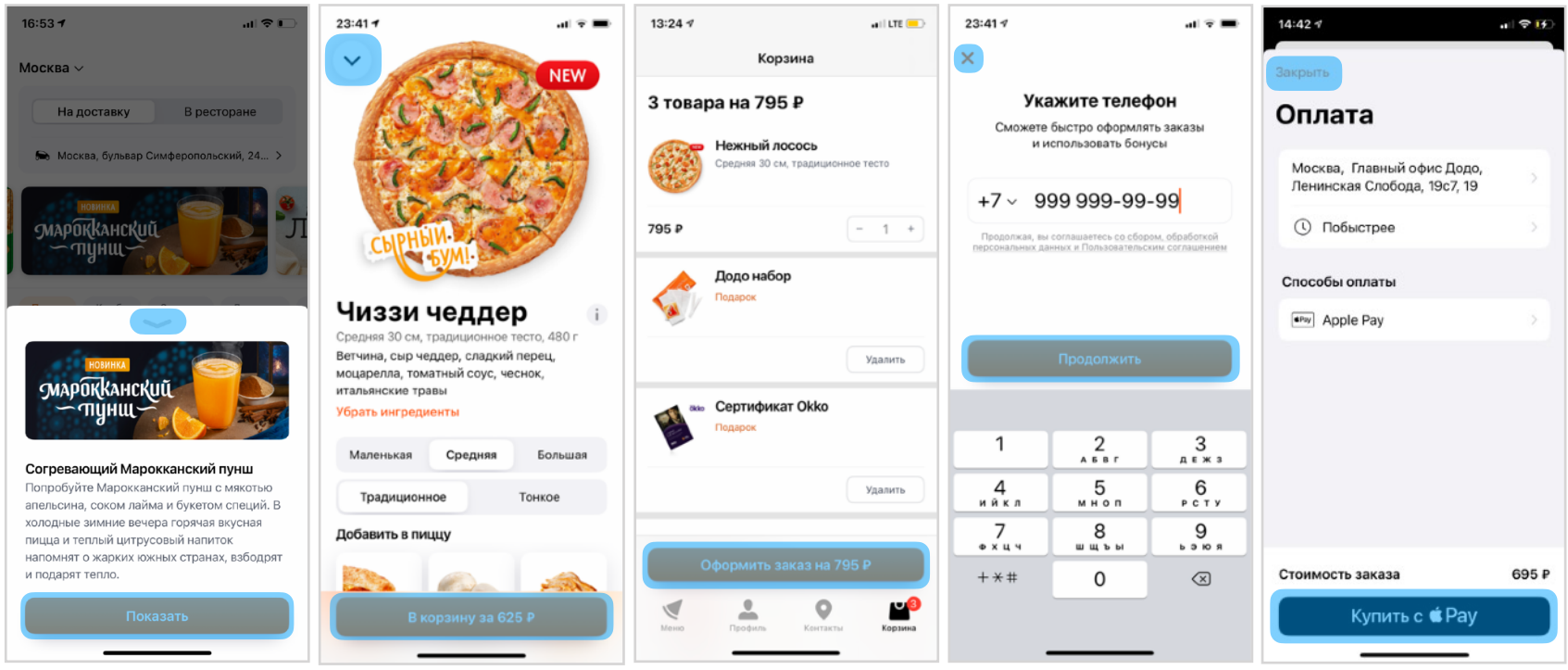
```
accessibilityViewIsModal = true
```

3. Закрывать окно жестом скраб.

```
override func accessibilityPerformEscape() -> Bool {
    dismiss(animated: false)
    return true
}
```

Навигация

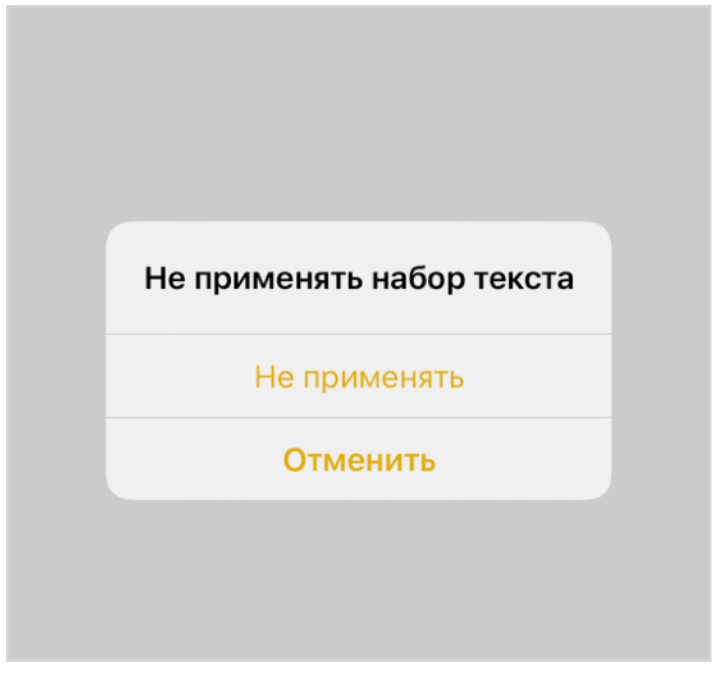
Крайние элементы

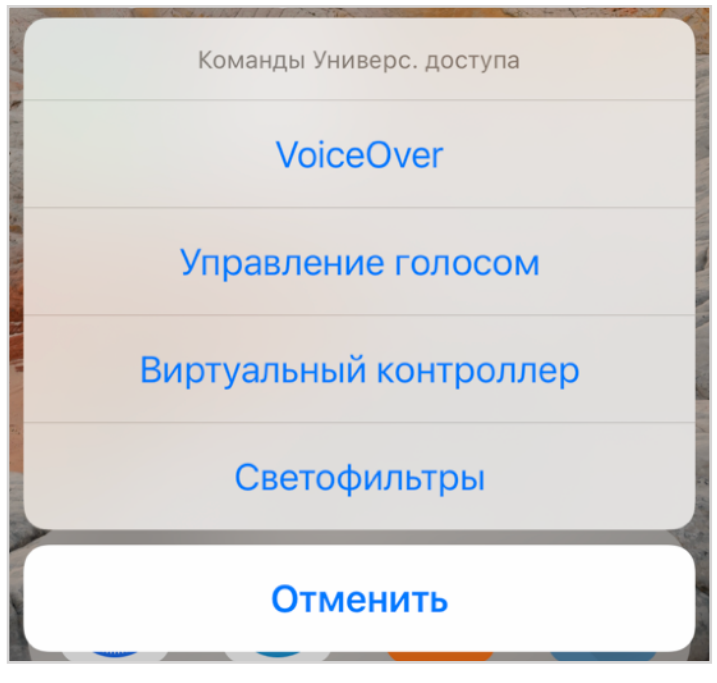


Первый и последний элементы могут играть особую роль во взаимодействии незрячих людей с телефоном, потому что к ним можно быстро переключаться, тапнув тремя пальцами в верхней или нижней части экрана.

В `UINavigationController` кнопка «назад» первая на экране, а в системных алертах кнопка «отменить» последняя.

Учитывайте особую роль этих кнопок и всегда размещайте важный навигационный элемент в крайней точке: кнопки закрытия слева сверху, а подтверждение в самом низу справа.





Скрол

Это сложная глава для самых опытных, чтобы лучше понять как всё работает изнутри.

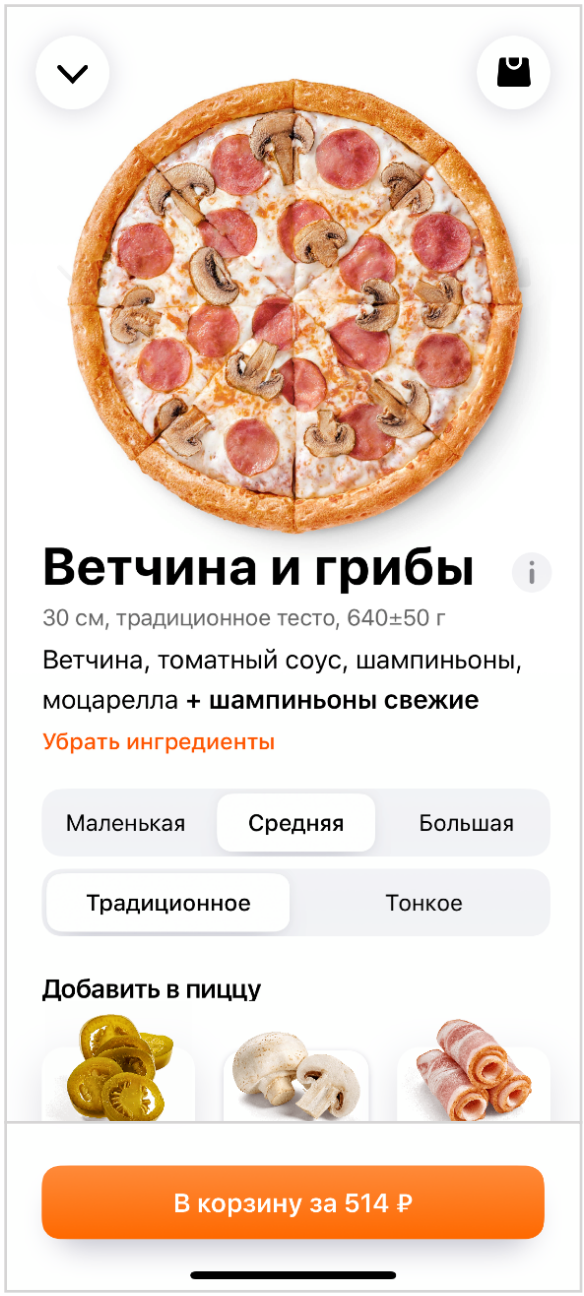
UIScrollView — это интересный пример контейнера для доступных элементов:

- он реагирует на перемещение фокуса и подскроливает, чтобы элемент был виден;
- его можно проскролить тремя пальцами и он обновит положение фокуса;
- после скрола он прочитает описание контента: на какой вы странице (2 из 3), где стоит фокус (в центре экрана).

Обычно для адаптации ничего дополнительного делать не нужно, UIScrollView сам по себе работает хорошо. Тем не менее посмотрим, какие методы нужно реализовать, если вам нужно повторить его поведение. Сценарий VoiceOver такой:

1. Пользователь свайпает тремя пальцами в одном из четырех направлений.
2. VoiceOver ищет элемент, который может обработать событие `accessibilityScroll(:)` и передает ему направление скрола.
3. Вы можете на каком-то уровне перехватить это событие и проскролить. Если вы обработали событие, то верните `true`, так закончится поиск. По дефолту возвращается `false` и VoiceOver продолжает подниматься по иерархии `UIView`, пока не найдет того, кто может обработать.
4. Внутри функции вы решаете, что делать с направлением скрола. Скрол, таблицы и коллекции сдвигаются на один экран.
5. Допустим, вы решили тоже проскролить на один экран. После скрола нужно отправить оповещение `.pageScrolled` и описать новое состояние экрана. `UITableView` запрашивает описание из `accessibilityScrollStatus(for:)`, а `UIScrollView` читает текущее положение: «Страница 3 из 12». Вместе с оповещением VoiceOver издаст специальную вибрацию.

В итоге вам надо реализовать `accessibilityScroll(:)`, правильно проскролить и вызвать оповещение. Давайте посмотрим на примере. Карточка продукта может быть длинной, дополнительных ингредиентов может быть много, они прячутся под кнопкой добавления в корзину. При этом весь экран находится в `UIPageViewController`, его можно свайпнуть горизонтально, чтобы сменить продукт (жаль, что пока визуально это никак не видно).



Проскролить экран нужно в двух случаях:

— если вы свайпнули тремя пальцами вертикально, чтобы посмотреть, что ниже. При этом нужно не сломать горизонтальный скрол для смены продуктов.

— если фокус попал за пределы экрана, то нужно подскролить, чтобы фокус стал виден.

На уровне контроллера, который содержит `UIScollView`, нужно реализовать метод `accessibilityScroll(:)`, обработать только вертикальные свайпы, а для горизонтальных вернуть `false`, так VoiceOver пойдет по иерархии контроллеров к дочерним, чтобы проверить, могут ли они обработать свайп. А они могут: родительский `UIPageController` может обработать горизонтальный свайп и показать другой продукт.

По направлению свайпа можно понять, в какую сторону скролить. Экран простой, поэтому нужно скролить либо до начала экрана, либо до конца, этого достаточно. После скрола сообщите VoiceOver об обновлении через оповещение и расскажите о новом состоянии экрана. Ничего лучше, чем «верх» и «низ» я не придумал.

```
override func accessibilityScroll(_ direction:
UIAccessibilityScrollDirection) -> Bool {
     switch direction {
     case .up:
          view().contentScrollView.scrollToTop()
          UIAccessibility.post(notification:
                               .pageScrolled, argument: "Верх")
          return true
     case .down:
          view().contentScrollView.scrollToBottom()
          UIAccessibility.post(notification:
                               .pageScrolled, argument: “Низ")
          return true
     default:
          return false
     }
}
```

Теперь нужно скролить экран после переключения фокуса на контрол за границами видимой области. Обычно `UIScrollView` обрабатывает это самостоятельно, реализуя протокол `UIFocusItemScrollableContainer`. Если стандартное поведение не срабатывает, то мы можем написать свое: наследоваться от `UIScrollView` и реализовать все, что нужно.

О смене фокуса `UIKit` оповещает через `.elementFocusedNotification`. Подпишемся на него в тот момент, когда вью становится частью иерархии. Отписываться от него не нужно, если мы пишем для iOS 9+.

```
override func didMoveToSuperview() {
    super.didMoveToSuperview()
    observeFocusUpdate()
}
private func observeFocusUpdate() {
    NotificationCenter.default.addObserver(
        self,
        selector: #selector(focusDidChanged(_:)),
        name: UIAccessibility.elementFocusedNotification,
        object: nil)
}
```

При смене фокуса нужно получить текущий элемент, на котором стоит фокус, проскролить до него, чтобы его стало видно и сообщить о том, что лейаут страницы поменялся. Фокус оставим на самом элементе.

```
@objc private func focusDidChanged(_ notification: Notification)
{
    guard let element = notification.userInfo?
                    [UIAccessibility.focusedElementUserInfoKey]
                    as? UIView else { return }

    scrollRectToVisible(element.bounds, animated: false)

    UIAccessibility.post(notification: .layoutChanged,
                         argument: element)
}
```

Для правильной работы VoiceOver нужно выполнить еще несколько условий:

1. Нужно проверять, что элемент является дочерним для вью, так как мы получаем оповещение от всех контролов на экране, а нужны только те, что внутри скрола.
2. Нужно конвертировать фрейм элемента в координаты `UIScrollView`, потому что он может быть вложенным в другие вью.
3. Оповещение нужно, только если скрол реально произошел, иначе VoiceOver будет неправильно ставить фокус, когда вы выходите из `UIScrollView`.

```
@objc private func focusDidChanged(_ notification: Notification)
{
    guard let element = notification.userInfo?
                    [UIAccessibility.focusedElementUserInfoKey]
                    as? UIView else { return }

    guard element.isChild(of: self) else { return }

    let offsetChanged = scroll(to: element)

    guard offsetChanged
    else { return } // No need to set focus of items
                        that has been focused already

    UIAccessibility.post(notification: .layoutChanged,
                         argument: element)
}
```

Вспомогательный код из предыдущих примеров кода:

4. Функция, что конвертирует фреймы, скролит и проверяет изменился ли оффсет. По смещению мы поймем, что надо оповещать VoiceOver о смене лейаута.

```
extension UIScrollView {
    fileprivate func scroll(to view: UIView) -> Bool {
        let oldOffset = contentOffset

        scrollRectToVisible(convert(view.bounds, from: view),
animated: false)

        let offsetChanged = oldOffset != contentOffset
        return offsetChanged
    }
}
```

5. Рекурсивная функция проверяет, является ли элемент дочерним, чтобы UIScrollView обрабатывал только свои элементы.

```
extension UIView {
    fileprivate func isChild(of parentView: UIView) -> Bool {
        for subview in parentView.subviews {
            if subview === self {
                return true
            }

            if isChild(of: subview) {
                return true
            }
        }

        return false
    }
}
```

Теперь при смене фокуса UIScrollView будет подскроливать так, чтобы элемент в фокусе был виден на экране. Обычно он работает и сам, но если что-то пойдет не так, вы знаете как поправить.

Графики

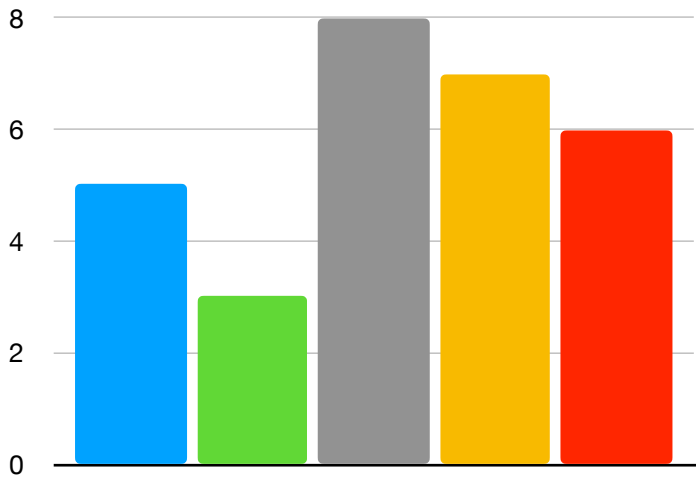
Особую роль в адаптации приложения занимают графики: для iOS это просто изображение, у них нет отдельных элементов `UIKit`, с которых VoiceOver мог бы взять информацию для доступности. Для графика нужно восстановить доступность с нуля: отметить области и подписать их.

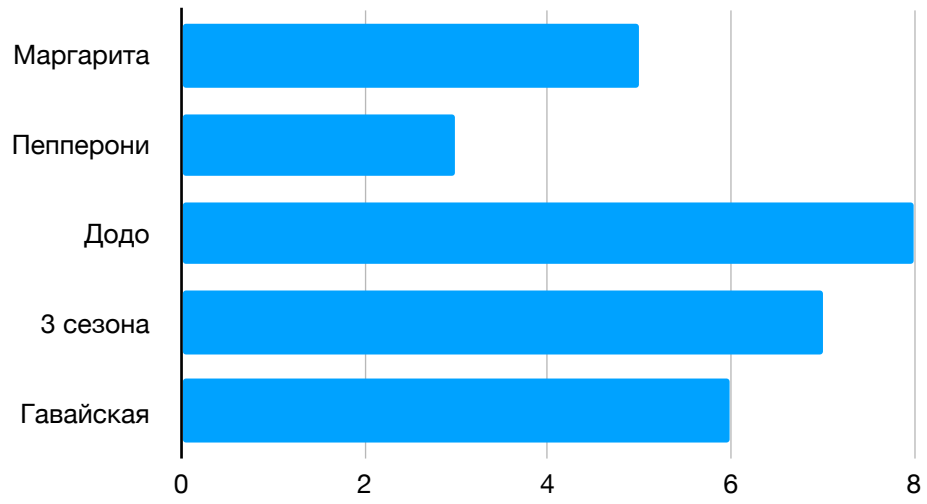
Сложность адаптации зависит от количества данных на графике. Например, если значений меньше десятка, то делаем каждый элемент отдельно, а потом ещё проходимся фокусом и читаем все данные. А вот если ключевых цифр сотни или тысячи на экране, то прочитать все по очереди уже не получится и потребуется сделать имитацию графика звуком или описать характер графика текстом.

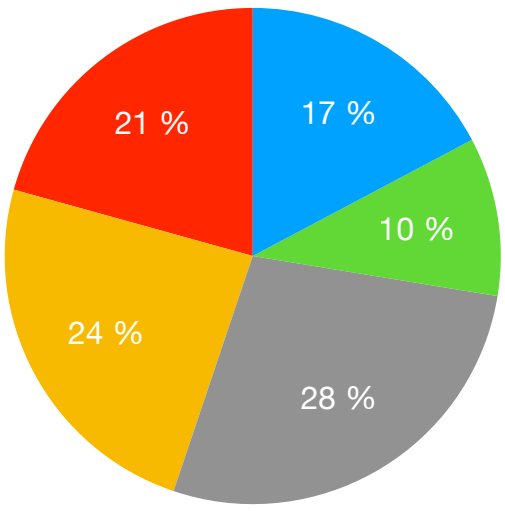
Для начала посмотрим, как добавить доступность к простым графикам, а потом разберем, как Apple сделала доступным сложный график курса акций в стандартном приложении «Акции» с помощью звука. В iOS 15 такое поведение можно повторить с помощью нового API.

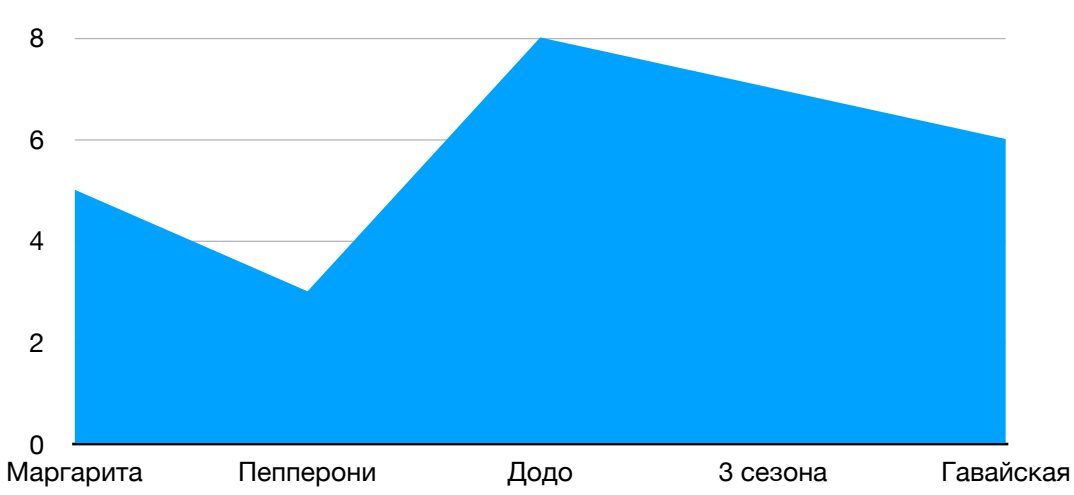
Графики

CustomElement



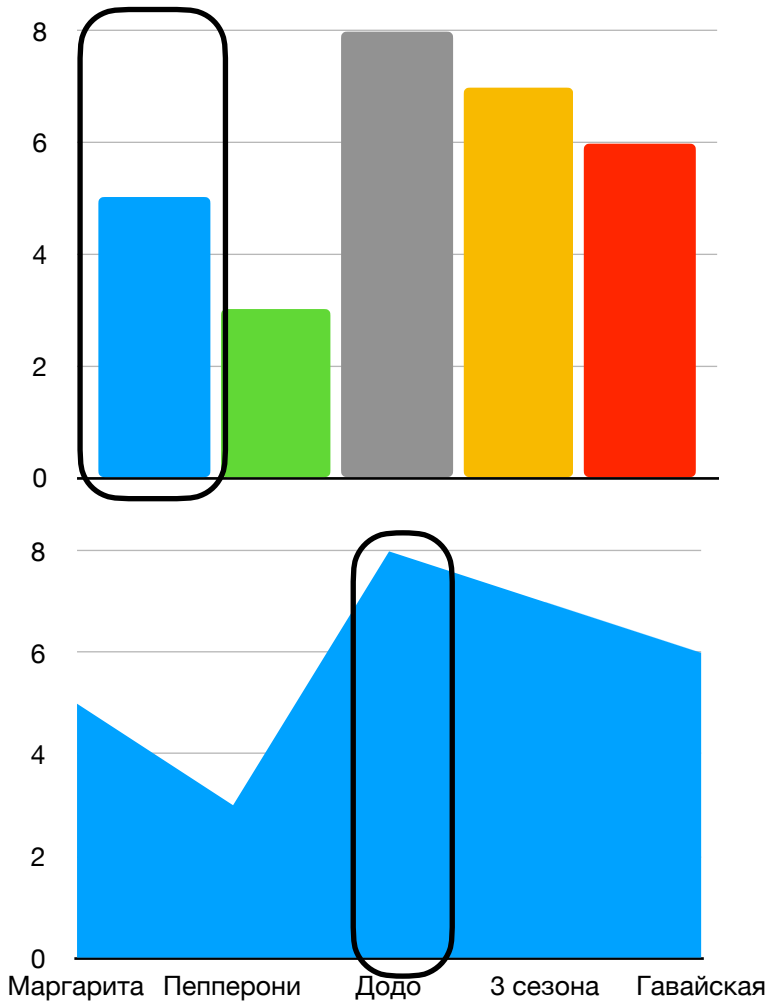






Графики бывают разные: линейные, круговые, точечные. Увы, стоит нам только что-то программно нарисовать, как доступность тут же теряется: VoiceOver больше не может анализировать иерархию контролов, потому что для неё весь график — один элемент.

Из-за этого может показаться, что адаптировать графики сложно: элементов много, а как сравнивать значения — не ясно. Но работа с графиками такая же, как и с остальным интерфейсом: сначала просто всё подписываем, чтобы пользоваться хоть как-то, а затем думаем как упростить с ними работу. Например, дать возможность отсортировать данные от большего к меньшему через `customActions`.



Несмотря на разный вид графиков, суть остается одна: мы даём способ пройтись по всем элементам и узнать их значение. Фокус может вставать на отдельный столбик или величину, поэтому размер столбика делайте как можно больше, чтобы было удобно использовать изучение касанием.

Чтобы сделать график доступным, мы создаём несколько доступных элементов и«кладём их внутрь» графика через свойство `.accessibilityElements`. Так VoiceOver поймет, что `UIView` с графиком — это контейнер для других элементов и начнет переключать по ним.

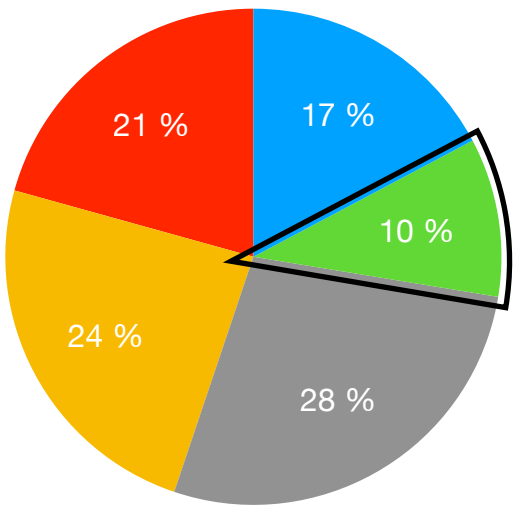
До этого момента VoiceOver всегда опирался на видимые `UIView`, но мы можем создать доступный элемент без привязки к вьюшке, для этого есть `UIAccessibilityCustomElement`. Конструктор у него простой, надо лишь указать, в каком вью этот элемент будет лежать, а дальше работать с ним как с обычным контролом.

Единственное отличие, что теперь обязательно надо задать `.accessibilityFrame`, иначе рамка фокуса не появится на экране и элемент будет недоступен для изучения касанием. При этом есть дополнительное свойство `accessibilityFrameInContainerSpace`, оно нужно на тот случай, если вы не можете рассчитать положение в координатах экрана, например, когда размещаете элементы внутри `UIScrollView`.

```
let element = UIAccessibilityElement(accessibilityContainer: self)
element.accessibilityLabel = graphElement.name
element.accessibilityValue = graphElement.value
element.accessibilityFrameInContainerSpace = graphElement.frame

accessibilityElements = [element]
```

Для создания доступных элементов можно использовать такие же структуры, как и для отрисовки графика. Пример, когда доступный элемент генерируется из описания:

```
struct GraphElement {
    let name: String
    let value: String
    let frame: CGRect

    func accessibilityElement(in parentView: UIView)
    -> UIAccessibilityElement {
        let element = UIAccessibilityElement(
                            accessibilityContainer: parentView)
        element.accessibilityLabel = name
        element.accessibilityValue = value
        element.accessibilityFrameInContainerSpace = frame
        return element
    }
}

accessibilityElements = chart
                        .elements
                        .map { $0.accessibilityElement(in: self)}
```



Возможно, часть графика не получится обвести прямоугольником, например, в пай-чарте, когда нужен фокус в виде сектора. Для этого случая можно задавать `.accessibilityPath` и передавать свою форму для фокуса. Как и у `.accessibilityFrame`, путь конвертируем в координаты экрана с помощью функции `.convertToSceenCoordinates(:in:)`

```
static func convertToScreenCoordinates(_ path: UIBezierPath,
                                       in view: UIView)
                                -> UIBezierPath
```

Графики

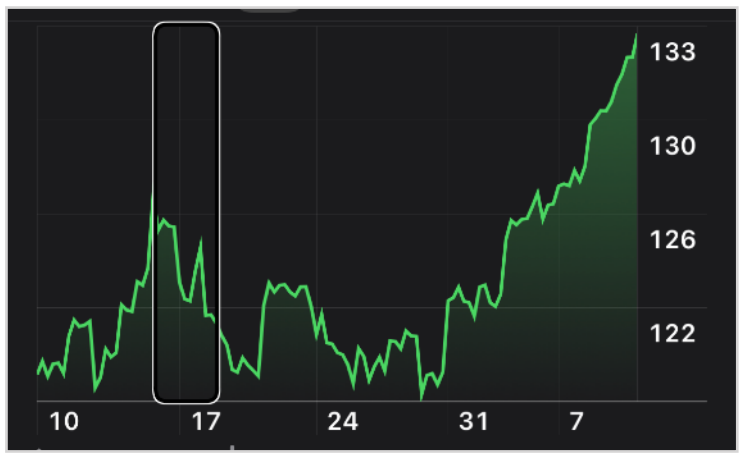
Много данных

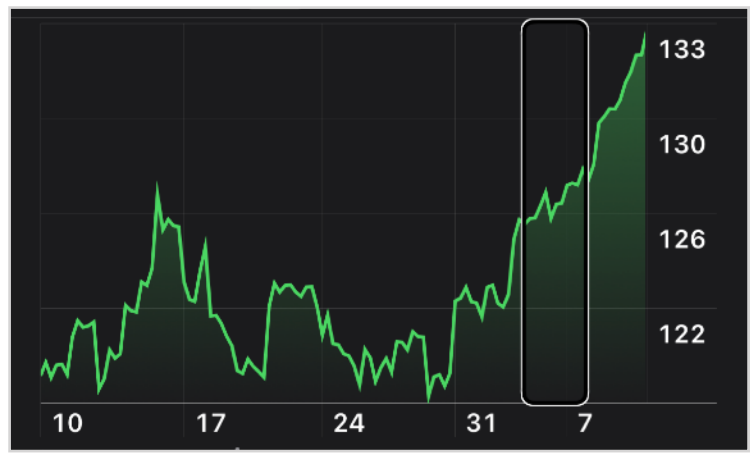
Совсем другая картина, когда данных очень много и визуализация необходима для простоты чтения. Например, рынок акций может сильно колебаться в течение дня, и эти колебания очень важны, поэтому нужно быстро реагировать на изменения.

Посмотрим, что делает с графиком Apple:

— Для начала, диапазоном времени можно управлять и выбрать нужную точность: день, неделя, месяц и т.п. Это элемент регулировки, вертикальным свайпом можно поменять период и увидеть обновленный график.
— Рядом с графиком есть цифры: справа минимальное и максимальное значения, а снизу временной промежуток. Они помогают больше понять о величинах на графике, но это всего лишь контекст.

Самое интересное происходит на графике: он разбит на 10 отрезков, каждый из них это элемент доступности — проходя по ним можно узнать среднее значение внутри отрезка.





Для акций такой точности недостаточно — внутри области могут быть самые разные данные. Чтобы точно узнать значение, тапнем дважды и зажмём: айфон начнет генерировать звук, высота которого будет совпадать с величиной на графике. Двигая пальцем по графику можно услышать изменение величины: звук станет выше или ниже. Если сделать паузу в движении, то VoiceOver прочитает точное значение.

У графика акций есть несколько контекстных действий, каждое помогает лучше понять, что представлено на графике. Увы, перевод на русский очень плохой. Качественный текст очень важен, ведь люди будут не читать такой текст, а воспринимать на слух.

График, описания которого расположены ниже

- **Описать диаграмму.** «Время — Цена, тон представляет собой «Цену» в диапазоне от 46 до 149 (долларов), аудиофрагменты будут воспроизведены из «Времени» от 0 до 115.» **Describe Chart.** «Time versus Price, The pitch of a tone represents Price in the range 46 to 149 and tones will play from Time 0 to 115.»

- **Суммировать цифровые данные.** «Минимум цена из 72.68 с время 19, максимум цена из 77.74 с время 100, означает цена 75.29.» «**Summarize numerical data.** Minimum price of 49.25 with Time 9, maximum Price of 142.06 with Time 95, mean price 92.956.»

- **Описать серии данных.** «Увеличивается, синусоидальная, прочная ассоциация, нет выбросов.» **Describe data series.** «Increasing, linear. very strong association, no outliers.»

- **Воспроизвести график аудио.** Проигрывает аудиодорожку графика, где высота тона описывает высоту графика. Пример можно посмотреть на YouTube.

Все это делает возможным чтение сложного графика акций и работу с ним.

Графики в iOS 15

В iOS 15 Apple дала возможность воспроизводить звук графика и в ваших диаграммах. Работает просто: реализуем протокол AXChart, добавляем проперти accessibilityChartDescriptor и описание для графика. В описании:

— добавляем название и кратко рассказываем про данные;

— описываем оси, по которым меняются значения;

— задаём координаты для всех точек.

После этого озвучиваем график: либо весь за раз, либо нажимаем на него дважды, зажимаем и водим фокусом/пальцем для считывания значений. Значение может меняться по нескольким осям: Apple в примере расположил точки в координатах X и Y, но у них ещё есть размер и цвет. Все это может быть доступным.

В нашем приложении графиков нет, опыта работы с ним тоже, поэтому просто ссылки:

— лекция про работу нового API: https://developer.apple.com/videos/play/wwdc2021/10122/,

— пример кода: https://developer.apple.com/documentation/accessibility/audio_graphs/representing_chart_data_as_an_audio_graph.

Разное

Мы посмотрели, какие базовые элементы бывают, как их подписывать, как правильно перемещаться между ними. Для большинства интерфейсов этого достаточно, но ситуации бывают разные и в VoiceOver есть несколько модификаторов, которые могут оказаться очень кстати. Сталкиваешься с ними не так часто, но знать о них полезно.

Разберем, как трейты меняют поведение VoiceOver, а еще как можно поменять голос, обработать загрузку, ошибки и чем полезна хаптик-отдача.

Трейты

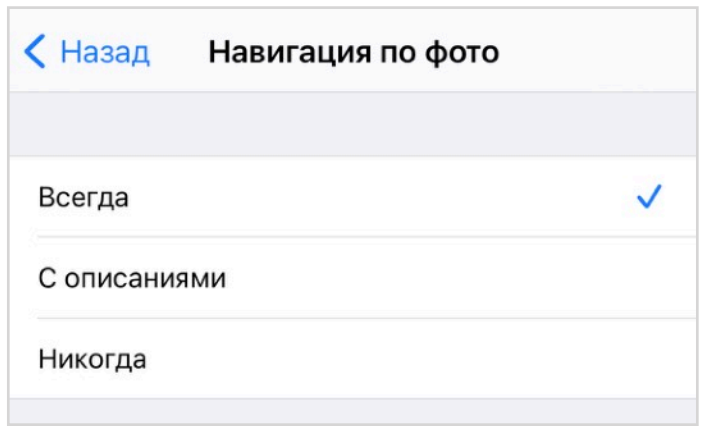
Мы уже использовали несколько трейтов, чтобы повлиять на описание и поведение контролов: `.selected`, `.notEnabled`, `.button`, `.adjustable`. Это основные, но есть еще десяток специфических.

Тип

.button — добавит к произносимому описанию слово «кнопка»: не придётся писать текст «кнопка» самостоятельно и переводить на другие языки.

.adjustable — добавит к описанию «элемент регулировки», вертикальные свайпы будут вызывать методы `increment()` и `decrement()`.

.image — добавит к описанию «изображение». Картинки по умолчанию недоступны для VoiceOver, но если вы хотите сказать о том, что картинка есть, то надо отметить ее через этот трейт. Пользователь сам настроит, нужно ли читать картинки.



.link — добавит к описанию «ссылка». Стоит добавить к кнопкам, которые откроют браузер или другое приложение.

.searchField — добавит к описанию «поле поиска». Если отметить поле ввода, то к нему можно будет быстро перейти через ротор или кнопку Tab на клавиатуре.

.keyboardKey — обработка кнопки будет такой же, как у выбранного способа ввода клавиатуры. Работает как кнопка, но не добавляет слова «кнопка» к названию, что ускоряет работу с клавиатурой. Подробнее про это в главе «Экранная клавиатура».

.tab — используется во вкладках `UITabBarController`: добавляет описание «Вкладка, 2 из 4», при этом сам считает количество элементов.

Состояние

`.selected` — добавит перед описанием «выбрано».

`.notEnabled` — добавляет «недоступно» после описания. Пригодится, чтобы показать `.disabled` состояние кнопки. Важно: если вы хотите просто скрыть элемент от VoiceOver, то поставьте ему `isAccessibleElement = false`, а не используйте этот трейт.

В Interface Builder трейт называется «User Interaction Enabled» и почему-то имеет противоположное значение.

Навигация

`.header` — добавит к описанию «заголовок». Это даст больше информации о структуре страницы, к тому же между заголовками можно быстро перемещаться с помощью ротора.

`.summaryElement` — никак не меняет описание элемента, но зато его описание будет читаться каждый раз при открытии приложения, даже из свернутого состояния. Например, так можно отметить статус заказа и время, через которое привезут пиццу. Тогда, открыв приложение, вы сразу узнаете, сколько осталось ждать, не придётся вручную свайпать до надписи.Такой элемент может быть только один на страницу.

Взаимодействие

`.allowsDirectInteraction` — позволяет взаимодействовать с элементом напрямую. Может пригодиться для игр, программ рисования или поля для подписи. Добавит в подсказку текст:

> Используйте ротор, чтобы включить функцию «Прямое касание» для этого приложения.

`.playsSound` — если кнопка воспроизводит свой звук (или использует оповещение, чтобы прочитать текст), то этот трейт отключит звук VoiceOver после активации элемента.

`.startsMediaSession` — останавливает VoiceOver. Подойдет, если вы начали проигрывать видео или записывать звук и не хотите, чтобы VoiceOver прерывал процесс.

`.causesPageTurn` — после прочтения вызовет `.accessibilityScrollAction()`.

Обновление

`.updatesFrequently` — элемент часто обновляется. Подходит для всего, что показывает секунды. Пока фокус стоит на таком элементе, VoiceOver раз в 5 секунд читает его описание. В приложении-таймере такое хорошо бы сочеталось с `.summaryElement`. Если не поставить трейт, то VoiceOver будет читать новое значение при каждом изменении надписи, это может сильно мешать.

`.staticText` — можно применить к элементу, если хотите спрятать его интерактивность от VoiceOver. Например, технически у вас на экране `UITextField`, но редактирование у него пока отключено и вы не хотите, чтобы VoiceOver добавлял к его описанию текст про «коснитесь дважды, чтобы изменить». Еще по этому трейту VoiceOver поймет, что текст никогда не меняется и его можно не проверять при обновлении accessibility tree.

`.none` — отключает все трейты.

Как задавать трейты

Трейты — это элементы `OptionSet`, их может быть сразу несколько и задаваться они могут в разных частях кода. Например, ячейка в списке — это кнопка, что настраивается при создании ячейки, а вот их выбранность указывается в процессе работы. Поэтому, они задаются не через присваивание, а через функции работы с множествами, чтобы при установке одного свойства не потерять все остальные.

```
if isSelected {
    accessibilityTraits.insert(.selected)
} else {
    accessibilityTraits.subtract(.selected)
}
```

Скрытые трейты

Раз трейты — это OptionSet, и его значения задаются побитово. Значит мы можем перебрать все сдвиги в Int64, применить их к элементу и посмотреть как они поменяют поведение. Рекомендовать применять их не могу, последствия неизвестны. Код можно взять в репозитории.

Результаты в формате «сдвиг бита, **текст на английском** и на русском». Число — это насколько надо сдвинуть бит, поэтому значение трейта можно получить так:

```
let traitRaw = 1 << index
let trait = UIAccessibilityTraits(rawValue: UInt64(traitRaw))
```

19 **Pickeritem.** Инструмент выбора.

20 **Radio button.** Кнопка-переключатель.

23 **Status bar item.** Объект меню статуса.

25 **Inactive.** неактивный.

26 **Footer.** Нижний колонтитул.

28 **Tab.** вкладка.

32 **Visited.** Посещены.

35 **Tap and hold, then move up and down to select index.** Коснитесь дважды и удерживайте объект, затем передвигайтесь вверх и вниз, чтобы выбрать индекс.

38 **Draggable. Double tap and hold, wait for the sound, then drag to re-arrange.** Перетягиваемый объект. Коснитесь дважды и удерживайте, дождитесь звукового сигнала, затем перетяните объект для изменения взаимного расположения объектов.

39 Без описания. Режим обучения, в котором VoiceOver озвучивает ваши действия и говорит, что они делают. Этот режим можно включить в настройках телефона.

40 **Pop-up button.** Всплывающая кнопка. Коснитесь дважды для активации инструмента выбора.

42 **Maths.** Математика.

45 Без описания. Скрывает элемент от фокуса.

50 **Folder.** Папка.

52 **Menu item.** Пункт меню.

53 **Double tap to toggle settings.** Коснитесь дважды, чтобы переключить настройку.

59 **Video playback.** Воспроизведение видео. Коснитесь дважды, чтобы воспроизвести или приостановить.

60 **Icon.** Значок.

Разное

Хаптик

Вибрация и хаптик-отклик могут сильно разнообразить ощущения от использования приложения. При этом вибрация считывается быстрее, чем аудио сигнал и играет важную роль для VoiceOver.

Для вибрации есть несколько стандартных классов.

`UI`**`Selection`**`FeedbackGenerator` расскажет о небольшом изменении в интерфейсе.

```
UISelectionFeedbackGenerator().selectionChanged()
```

`UI`**`Notification`**`FeedbackGenerator` посложнее. Он может сыграть один из трех вариантов:

- подтверждение `.success`;
- предупреждение `.warning`;
- ошибку `.error`.

```
UINotificationFeedbackGenerator().notificationOccurred(.success)
```

Если генератор заранее подготовить, то он точно воспроизведёт вибрацию без задержки. Для этого у него заранее вызовем метод `prepare()`.

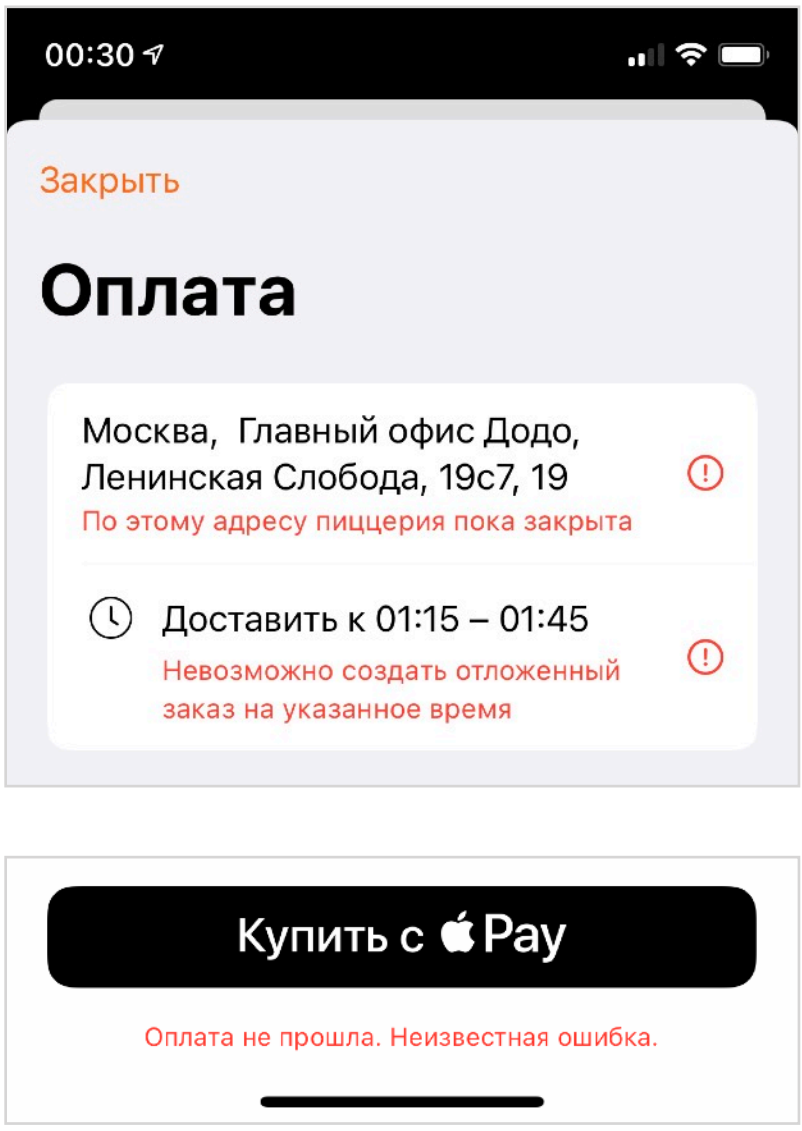
```
let haptic = UINotificationFeedbackGenerator()

override func viewDidLoad() {
    super.viewDidLoad()
    haptic.prepare()
}
func didPay() {
    haptic.notificationOccurred(.success)
}
```

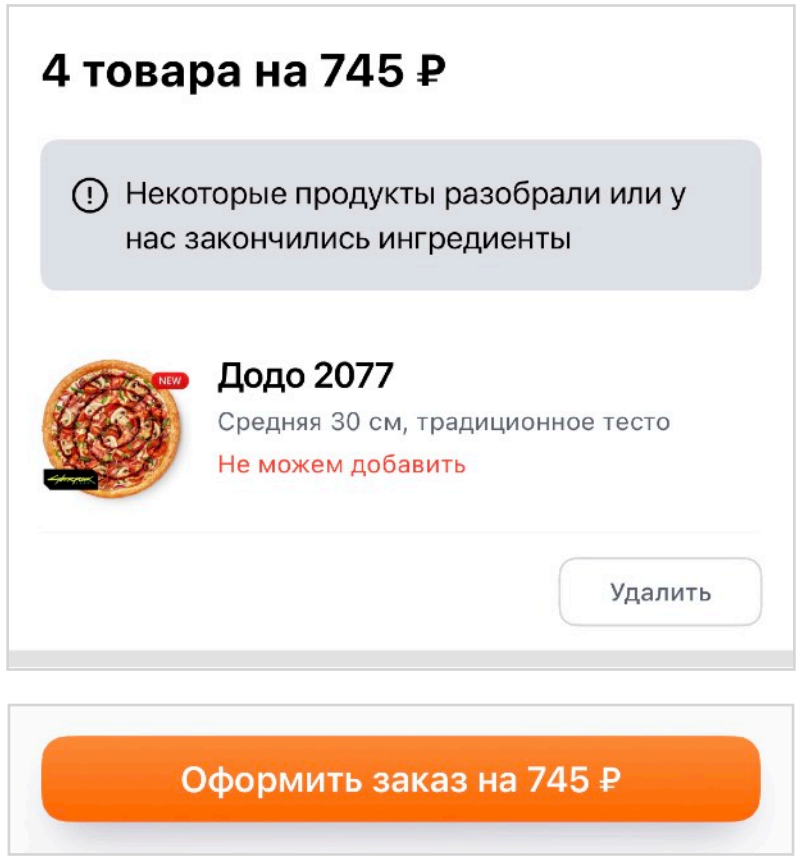
Хаптик воспроизводит бóльшую палитру эмоций и сильно настраивается. Это доступно не на всех моделях телефонов, но поуправлять параметрами вы можете в приложении Haptic Composer. Приложение полностью адаптировано для VoiceOver, в нем вы можете посмотреть интересные примеры адаптации.

Разное

Ошибки

Ошибки стоит показывать рядом с полем ввода, читать после их появления с помощью оповещений и не скрывать по таймеру.





Оформить заказ
1 ошибка, «Додо 2077», Не можем добавить
Доступно действие, 1 из 2: активировать (по умолчанию), перейти к ошибке.

Обработка ошибок станет намного проще, если:

- на кнопке действия рассказывать сколько ошибок, где они и какие;
- предлагать их решение;
- позволить переключаться между ошибками через ротор.

Появление ошибок сообщайте вибрацией с типом `.error`.

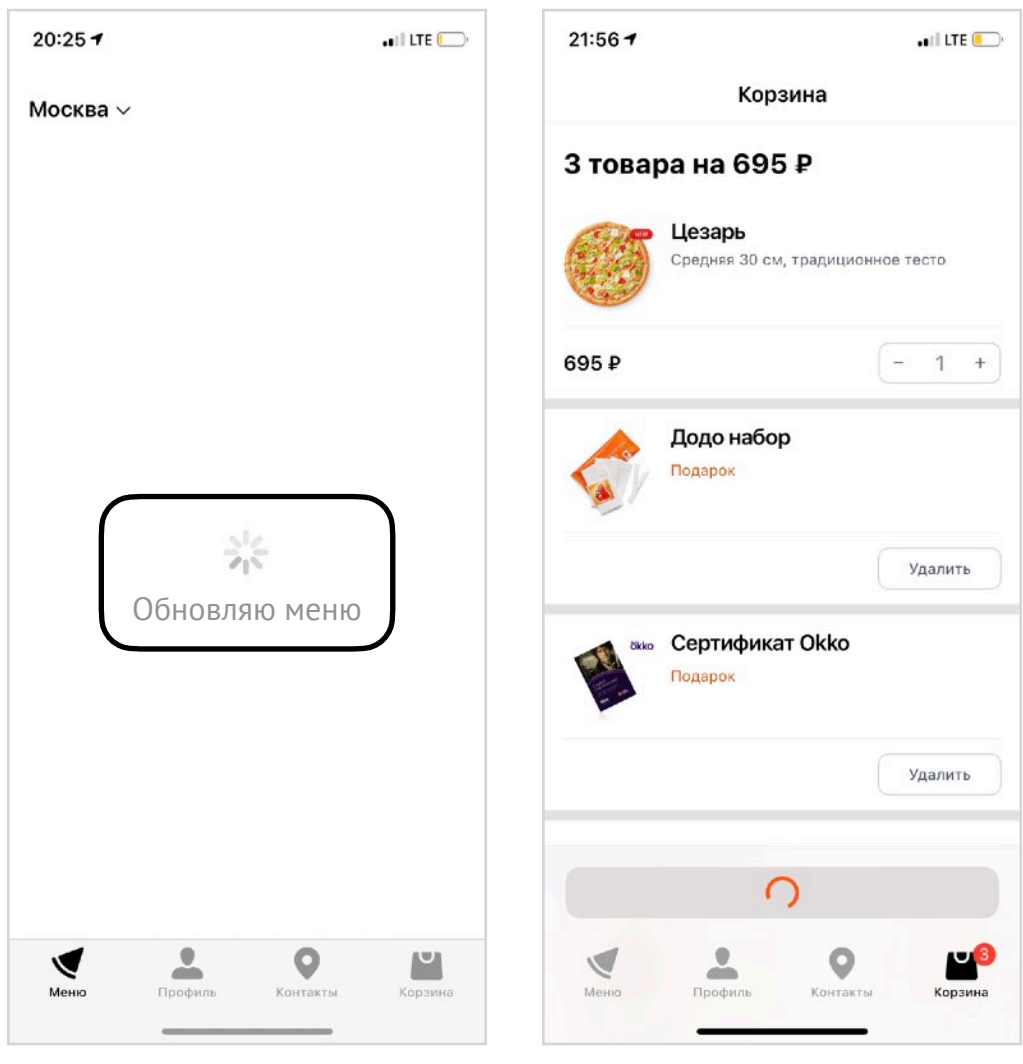
Загрузка

Загрузку можно разделить на три вида:

Короткая, < 0.5 сек. Можно ничего не делать, смена экрана сама расскажет о завершении. Но только если уверены, что загрузка не займет больше временис — сетевой запрос не может быть коротким.

Долгая блокирующая, когда полезных действий нет. Поставьте фокус на индикаторе, чтобы VoiceOver прочитал его.

Долгая неблокирующая. Расскажите о начале загрузки через оповещение.



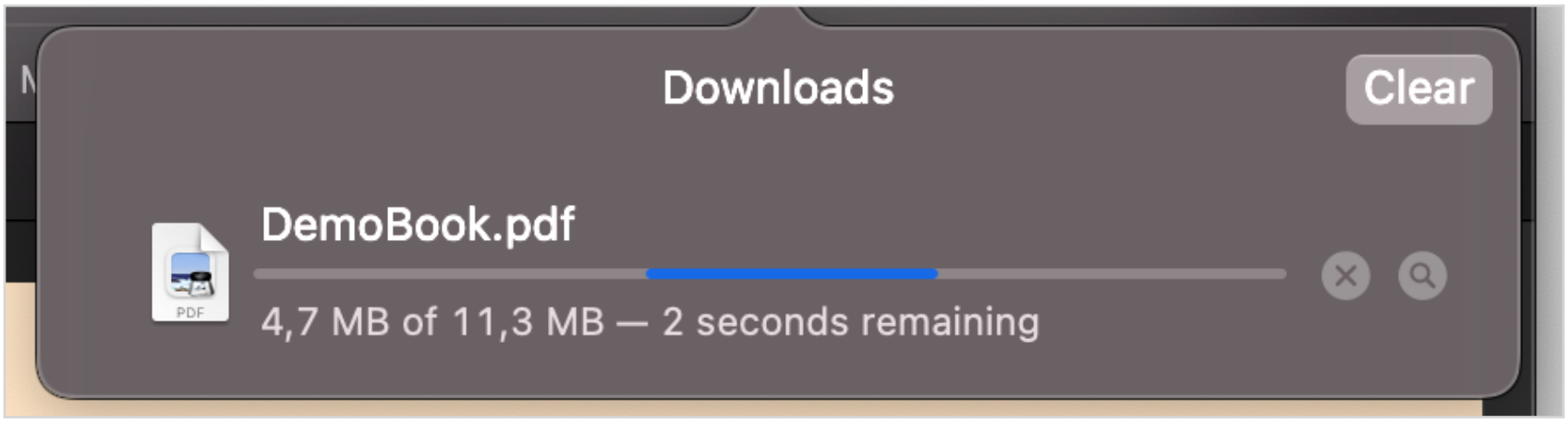
Пример блокирующей и неблокирующей загрузки

Хорошо обработана загрузка в Сафари: издает щелчки и легкую вибрацию каждую секунду ожидания. Я постарался повторить поведение через вибрацию и написал Accessibility Loading.

Прогресс

Если у вас есть долгое действие, которое измеряется в секундах или даже минутах, то обработайте его дополнительно.

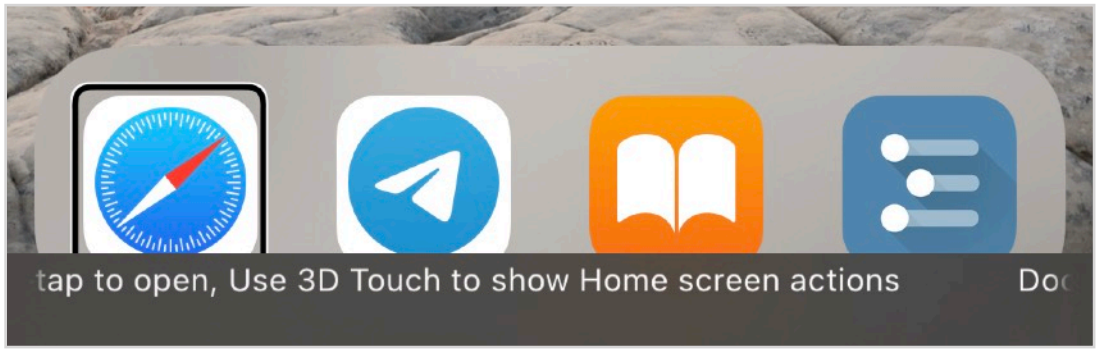
- Расскажите о состоянии загрузки, например, в процентах через value.
- Рассчитайте оставшееся время.
- Если фокус на элементе, то временами повторяйте его состояние. Трейта `.updatesFrequently` достаточно.
- В конце процесса рассказажите о его завершении через оповещение с типом `.announcement`.



3D-тач

3D-тач работает с VoiceOver. Примеров у меня нет, но есть наблюдение.

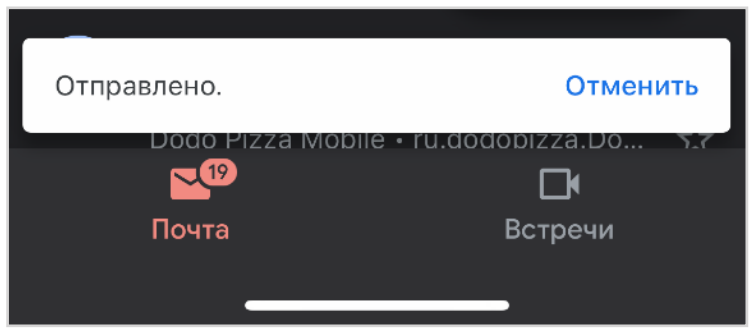
Главная проблема 3D-тача — очень сложно узнать, когда он есть, графически об этом ничего не сообщает. Но у VoiceOver есть подсказки, поэтому вы можете сообщить о поддержке сильного нажатия через них.



Иконки на спрингборде рассказывают о 3Д-таче

Разное

Тост с действием



В интерфейсах иногда появляются всплывающие окна, которые предполагают, что какое-то действие можно отменить. Например, гугл-почта предлагает отменить отправку письма в первые несколько секунд.

Такой контрол — довольно сложный случай для VoiceOver, потому что у него есть несколько проблем:

— появляется на ограниченное время;

— на нем не получится сфокусироваться быстро, а значит нельзя прочитать второй раз или выполнить действие, баннер к этому моменту уже скроется.

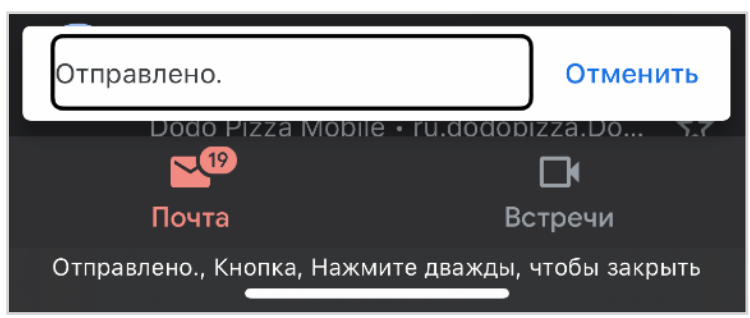
Как исправить эти проблемы.

— Увеличить время отображения в несколько раз для VoiceOver, чтобы на нем можно было успеть сфокусироваться. Лучше вообще не скрывать по времени, а только от действий пользователя, например, когда он ушел на другой экран.

— Поставить плашкой в самый верх или низ экрана, чтобы к ним можно было перейти тапом четырьмя пальцами в верхней или нижней половине экрана. Низ экрана обрабатывать проще, ведь такая плашка находится поверх всего экрана, а значит будет последней по порядку обхода. Если есть действие, то удобно будет навести на плашку внизу экрана, ведь она уже под пальцем, не нужно перехватывать телефон.

— Можно добавить отмену действия по встряхиванию телефона и рассказать о таком поведении при появлении элемента.

— Вибрация с типом `.warning` даст достаточно хорошее понимание, что действие не опасное, но стоит обратить на него внимание.



— Критичные действия могут сами поставить фокус на плашку (например, при отправке письма) и не ставить его для неважных (архивирование письма). Интересно, что Гугл надпись «отправлено» превратил в кнопку и по ней можно закрыть тост. Универсально работает и для VoiceOver и для графического интерфейса.

Голос

У каждого текстового свойства (`label`, `value`, `hint` и т.п.) есть парное свойство, которое может принять `NSAttributedString`, например, `accessibilityAttributedLabel`. В него можно вставить строку с дополнительными параметрами. Все параметры изменяют голос или произношение.

`.speechPunctuation` — прочитает пунктуацию. Принимает `true` или `false`.

`.speechLanguage` — прочитает в нужном языке. Параметр в формате BCP 47 specification например “`ru-RU`”.

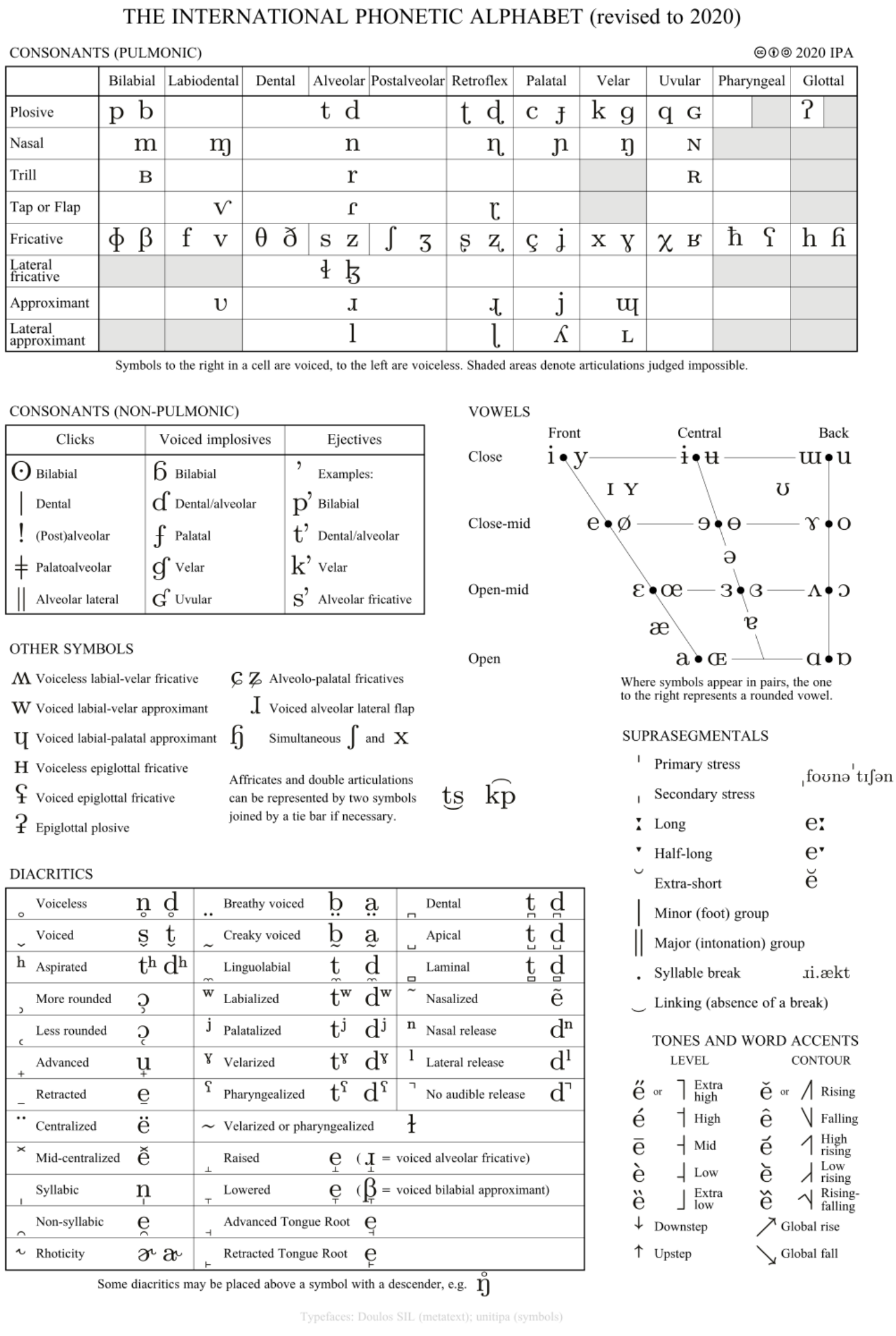
`.speechPitch` — изменит высоту голоса. По умолчанию 1, изменяется от 0 до 2.

`.speechIPANotation` — учтет фонетически правильное прочтение из параметра.

`.speechSpellOut` — прочитает по буквам. Принимает `true/false`.

```
let attributedString = NSMutableAttributedString(
        string: "This is the best app on the App Store!")
let range = attributedString.string.range(of: "best")

attributedString.addAttributes(
     [.accessibilitySpeechPitch: 1.5],
      range: NSRange(range!, in: attributedString.string))

appDescription?.accessibilityAttributedLabel = attributedString
```

Если вы хотите исправить прочтение слова, то укажите его транскрипцию на основе Интернационального фонетического алфавита.



THE INTERNATIONAL PHONETIC ALPHABET (revised to 2020)

CONSONANTS (PULMONIC) — © 2020 IPA

	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Retroflex	Palatal	Velar	Uvular	Pharyngeal	Glottal
Plosive	p b			t d		ʈ ɖ	c ɟ	k g	q ɢ		ʔ
Nasal	m	ɱ		n		ɳ	ɲ	ŋ	ɴ		
Trill	ʙ			r					ʀ		
Tap or Flap		ⱱ		ɾ		ɽ					
Fricative	ɸ β	f v	θ ð	s z	ʃ ʒ	ʂ ʐ	ç ʝ	x ɣ	χ ʁ	ħ ʕ	h ɦ
Lateral fricative				ɬ ɮ							
Approximant		ʋ		ɹ		ɻ	j	ɰ			
Lateral approximant				l		ɭ	ʎ	ʟ			

Symbols to the right in a cell are voiced, to the left are voiceless. Shaded areas denote articulations judged impossible.

CONSONANTS (NON-PULMONIC)

Clicks	Voiced implosives	Ejectives
ʘ Bilabial	ɓ Bilabial	’ Examples:
ǀ Dental	ɗ Dental/alveolar	pʼ Bilabial
ǃ (Post)alveolar	ʄ Palatal	tʼ Dental/alveolar
ǂ Palatoalveolar	ɠ Velar	kʼ Velar
ǁ Alveolar lateral	ʛ Uvular	sʼ Alveolar fricative

OTHER SYMBOLS

ʍ Voiceless labial-velar fricative
w Voiced labial-velar approximant
ɥ Voiced labial-palatal approximant
ʜ Voiceless epiglottal fricative
ʢ Voiced epiglottal fricative
ʡ Epiglottal plosive

ɕ ʑ Alveolo-palatal fricatives
ɺ Voiced alveolar lateral flap
ʩ Simultaneous ʃ and x

Affricates and double articulations can be represented by two symbols joined by a tie bar if necessary. t͡s k͡p

DIACRITICS

◌̥	Voiceless	n̥ d̥	◌̤	Breathy voiced	b̤ a̤
◌̬	Voiced	s̬ t̬	◌̰	Creaky voiced	b̰ a̰
◌ʰ	Aspirated	tʰ dʰ	◌̼	Linguolabial	t̼ d̼
◌̹	More rounded	ɔ̹	◌ʷ	Labialized	tʷ dʷ
◌̜	Less rounded	ɔ̜	◌ʲ	Palatalized	tʲ dʲ
◌̟	Advanced	u̟	◌ˠ	Velarized	tˠ dˠ
◌̠	Retracted	e̠	◌ˤ	Pharyngealized	tˤ dˤ
◌̈	Centralized	ë	◌̴	Velarized or pharyngealized	ɫ
◌̽	Mid-centralized	e̽	◌̝	Raised	e̝ (ɹ̝ = voiced alveolar fricative)
◌̩	Syllabic	n̩	◌̞	Lowered	e̞ (β̞ = voiced bilabial approximant)
◌̯	Non-syllabic	e̯	◌̪	Advanced Tongue Root	e̪
◌˞	Rhoticity	ɚ ɐ˞	◌̺	Retracted Tongue Root	e̺

◌̪	Dental	t̪ d̪
◌̺	Apical	t̺ d̺
◌̻	Laminal	t̻ d̻
◌̃	Nasalized	ẽ
◌ⁿ	Nasal release	dⁿ
◌ˡ	Lateral release	dˡ
◌̚	No audible release	d̚

Some diacritics may be placed above a symbol with a descender, e.g. ŋ̊

VOWELS

Where symbols appear in pairs, the one to the right represents a rounded vowel.

SUPRASEGMENTALS

ˈ Primary stress ˌfoʊnəˈtɪʃən
ˌ Secondary stress
ː Long eː
ˑ Half-long eˑ
◌̆ Extra-short ĕ
| Minor (foot) group
‖ Major (intonation) group
. Syllable break ɹi.ækt
‿ Linking (absence of a break)

TONES AND WORD ACCENTS

LEVEL		CONTOUR	
e̋ or ˥	Extra high	ě or ˧˥	Rising
é ˦	High	ê ˥˧	Falling
ē ˧	Mid	e᪲ ˦˥	High rising
è ˨	Low	e᪳ ˨˧	Low rising
ȅ ˩	Extra low	e᪴ ˩˥	Rising-falling
↓	Downstep	↗	Global rise
↑	Upstep	↘	Global fall

Typefaces: Doulos SIL (metatext); unitipa (symbols)

Internatonal Phonetic Alphabet. Схема из Википедии

Набор текста

Пора разобрать самое сложное — набор текста.

Для набора текста нужно много кнопок, поэтому в iOS встроено несколько механизмов, упрощающих ввод текста: разные режимы ввода для обычной клавиатуры, специальная клавиатура для алфавита Брайля, особые режимы навигации в роторе и поддержка физической клавиатуры.

Сначала разберемся с тем, как текстовые поля работают в VoiceOver, как незрячие вводят текст, а потом посмотрим, что учесть при работе физической клавиатуры.

Поля ввода

Когда фокус VoiceOver попадает на поле ввода, он читает так:

label	Улица
value	(тут будет текст в поле, больше ничего нельзя к value добавлять)
тип	Текстовое поле
hint	Коснитесь дважды, чтобы редактировать.

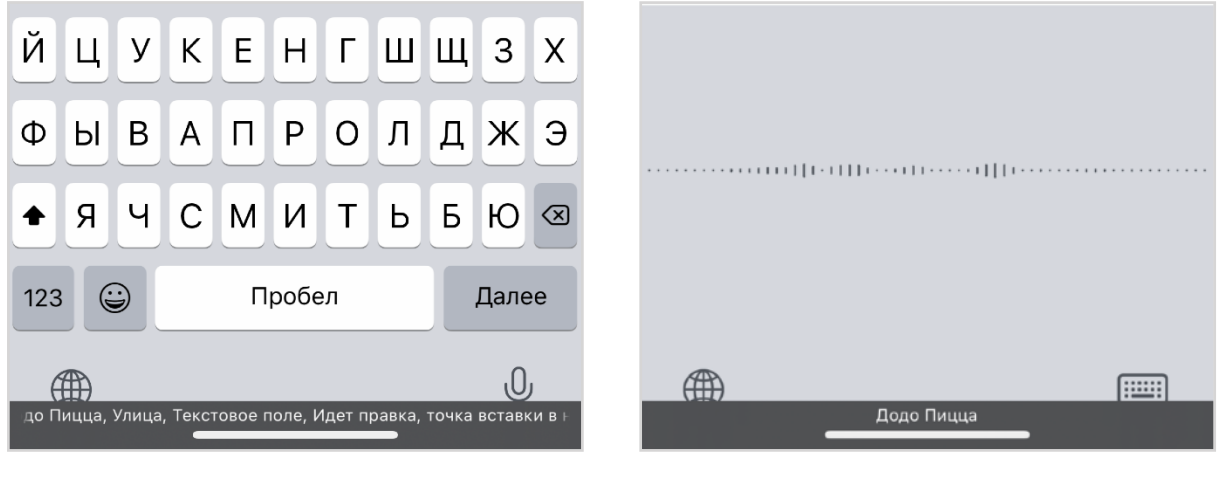
После активации поля и ввода текста:

Улица,
Симферопольская
Текстовое поле,
Идет правка. Точка вставки в начале.

Можно даже фокусом уйти с поля ввода, клавиатура не скроется. Если вернетесь к полю, то он уточнит, где находится курсор, например так: точка вставки между «в» и «р».

Самый простой способ ввести текст — продиктовать его. Чтобы начать ввод, дважды тапните двумя пальцами. По сути, диктофон включается через Magic Tap для текстового поля. VoiceOver распознает знаки препинания, но только если произнести их явно: точка, запятая, двоеточие и т.п.

Название текстового поля находится в label, а содержимым value управляет самая iOS, там будет введенный текст, поэтому никогда не модифицируйте value сами.



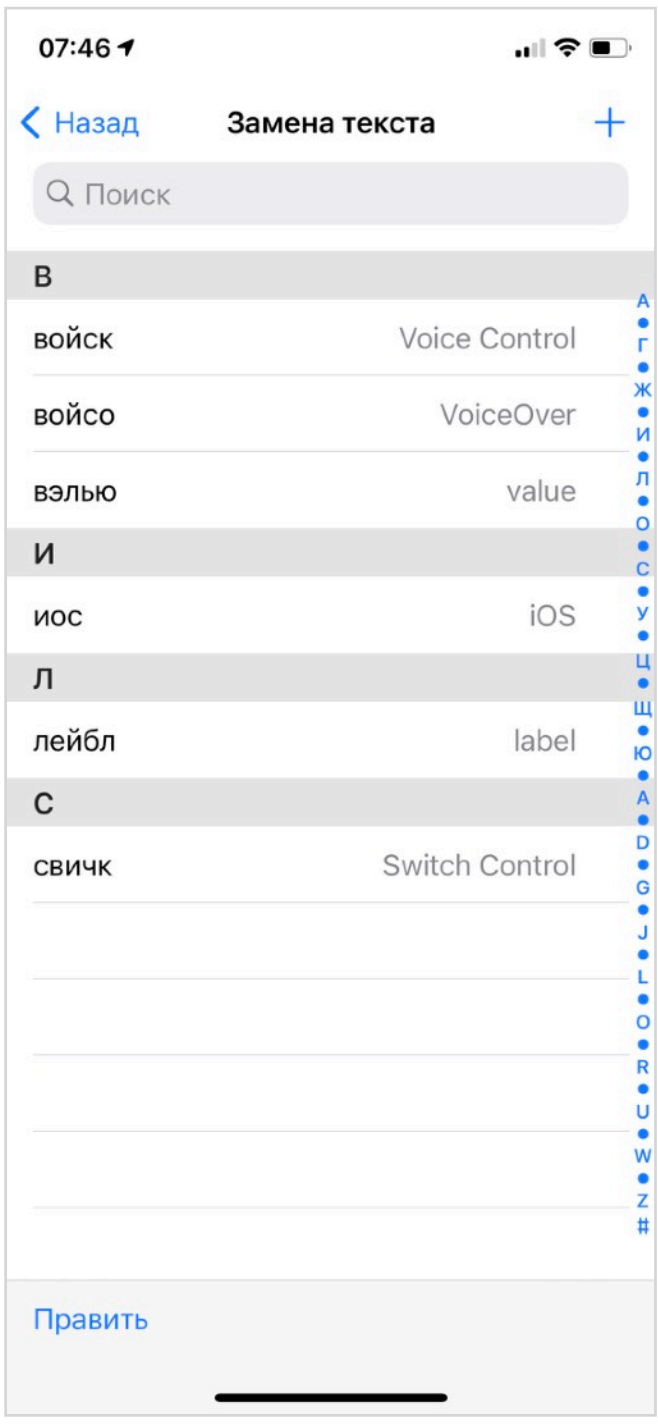
Диктовка текста включается по Magic Tap

Замена текста

С помощью стандартной автозамены VoiceOver может на лету поменять простую фразу на сложный текст. Например, вместо «емейл» написать вашу почту, вместо «адрес» подставить полный адрес. Такие команды можно использовать и как место для хранения особенных строк: почтового индекса, номера паспорта и т.п.

Автозамену можно использовать и для того, чтобы не переключать язык ввода с русского на английский.

Путь до настроек замены: Настройки → Основные → Клавиатуры → Замена текста.



Правила замены текста при работе с книгой

Ротор и клавиатура

Ротор важен при наборе текста, через него настраиваются разные режимы для вертикального свайпа:

— Переключение по словам с ошибкой.

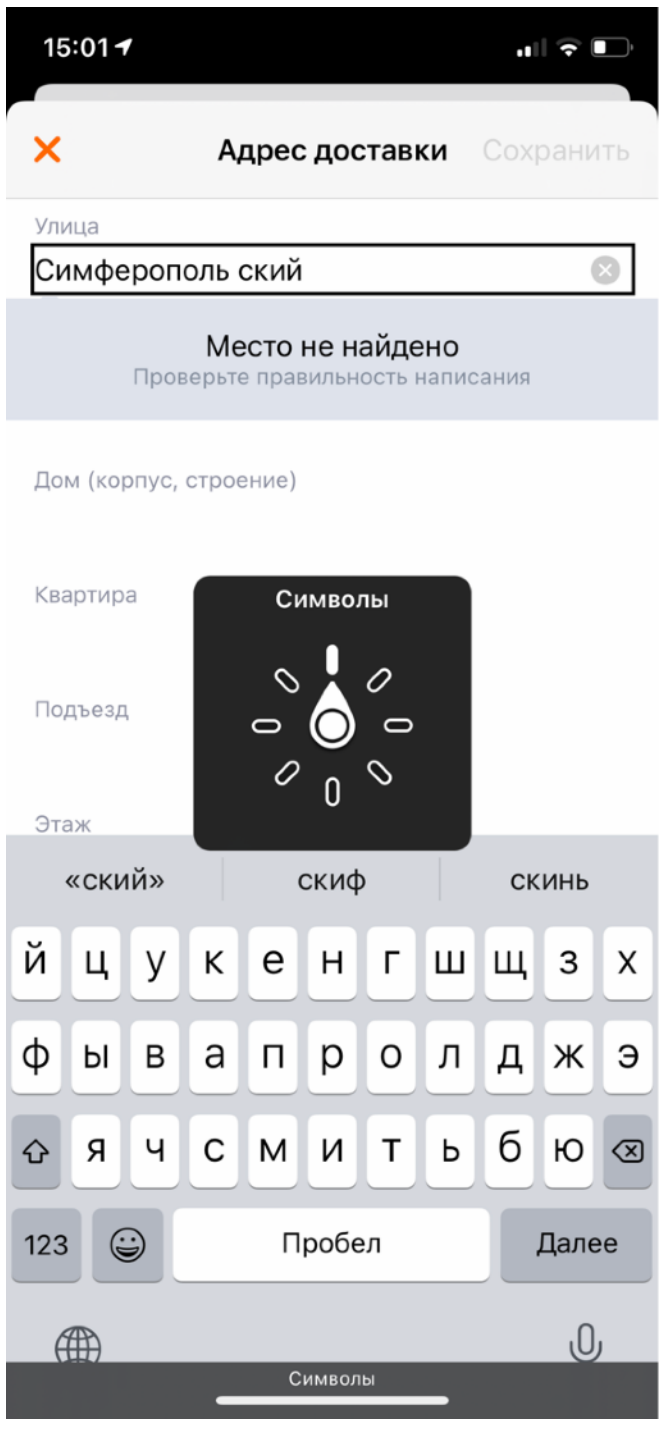
— Навигация по строкам, слову или символу.

— Режим ввода клавиатуры.

— Действия с текстом: выделение, копирование, вставка.

Они похожи на контекстные действия, но вынесены в ротор, так как через ротор много чего может изменяться и вертикальные свайпы будут заняты другими действиями.

Например, если я понимаю, что в тексте может быть ошибка, то выбираю режим «символы» вертикальным свайпом, переключаю курсор ввода между символами и VoiceOver читает их побуквенно.



Экранная клавиатура

Текст можно вводить не голосом, а печатая на экранной клавиатуре. Кнопок на ней много, читать лишь один звук и после него добавлять «кнопка» избыточно, поэтому VoiceOver читает звук и имя, которое с него начинается: «А, Антон». Для букв без имен скажет их название: мягкий знак, знак доллара, минус и т.п.

Нажимать дважды для активации каждой буквы слишком сложно, поэтому VoiceOver имеет 3 альтернативных режима для кнопок клавиатуры.

Обычный набор, Standard Typing

Выберите клавишу навигацией свайпом, нажмите дважды для ввода. Удобно использовать изучение касанием и две руки:

— одним пальцем двигайте по клавиатуре — VoiceOver в это время читает буквы;

— второй рукой касайтесь экрана для подтверждения ввода.

Буква на кнопке читается и при попадании в фокус и после ввода, чтобы избежать ошибок от случайного сдвига пальца. 3D-тач при этом работает: не отпуская палец, можно нажать сильнее, чтобы передвинуть курсор.

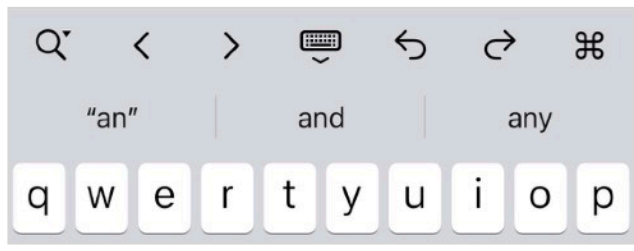
Набор одним касанием, Touch Typing

Удобен для ввода одной рукой: коснитесь клавиатуры, наведите на нужную кнопку, отпустите для ввода. Если изначально коснулись не той кнопки, то переведите палец на нужную. VoiceOver прочитает букву только при фокусе на кнопке.

Набор прямым касанием, Direct Touch Typing

Обычный ввод, будто VoiceOver выключен.

Если вы добавляете кнопки около клавиатуры, то используйте трейт .keyboardKey, чтобы способ активации этих кнопок совпадал с режимом ввода.



Пример панели рядом с клавиатурой, iA Writer

Набор текста

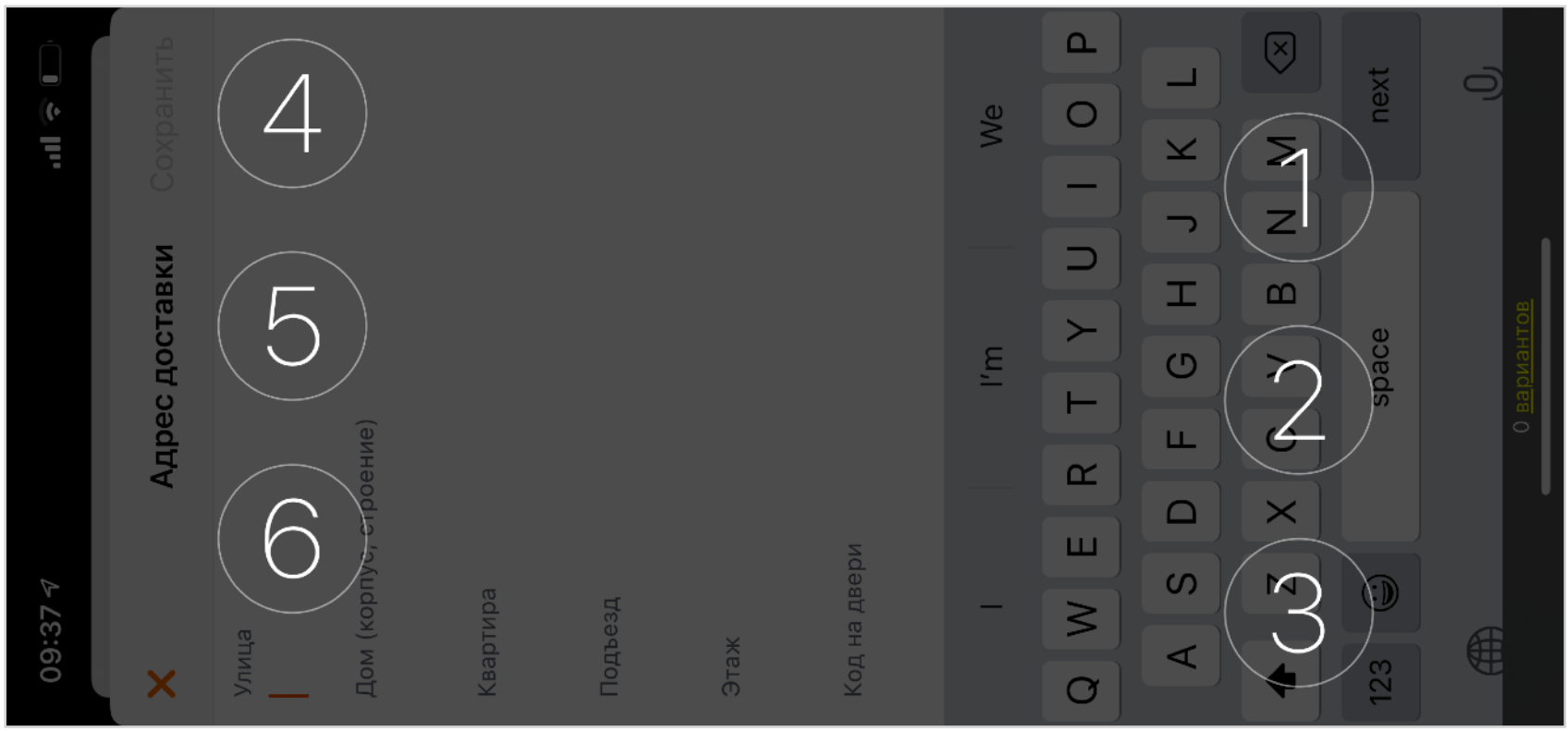
Клавиатура Брайля

Сильно увеличить скорость печати можно с помощью экранной клавиатуры Брайля.

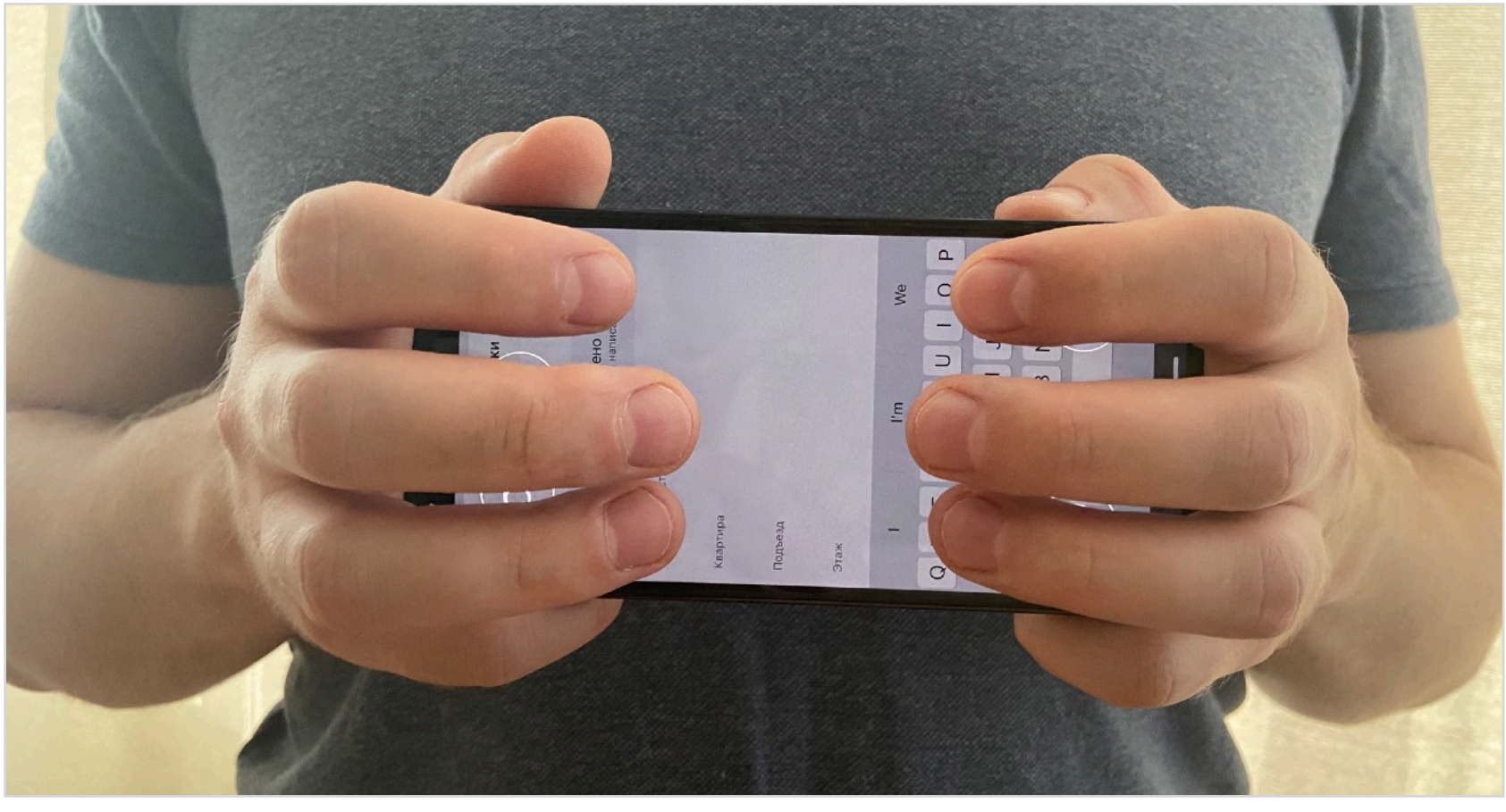
На бумаге буква в шрифте Брайля состоит из 6 точек, некоторые из которых выпуклые. Приложив палец к такому набору точек, можно считать выпуклости и понять, какая буква под пальцем.

Экраны телефонов не умеют передавать выпуклости, но им это и не нужно: VoiceOver предлагает вводить буквы в виде точек из шрифта Брайля. Почитать подробнее про шрифт можно в блоге Анатолия Попко.

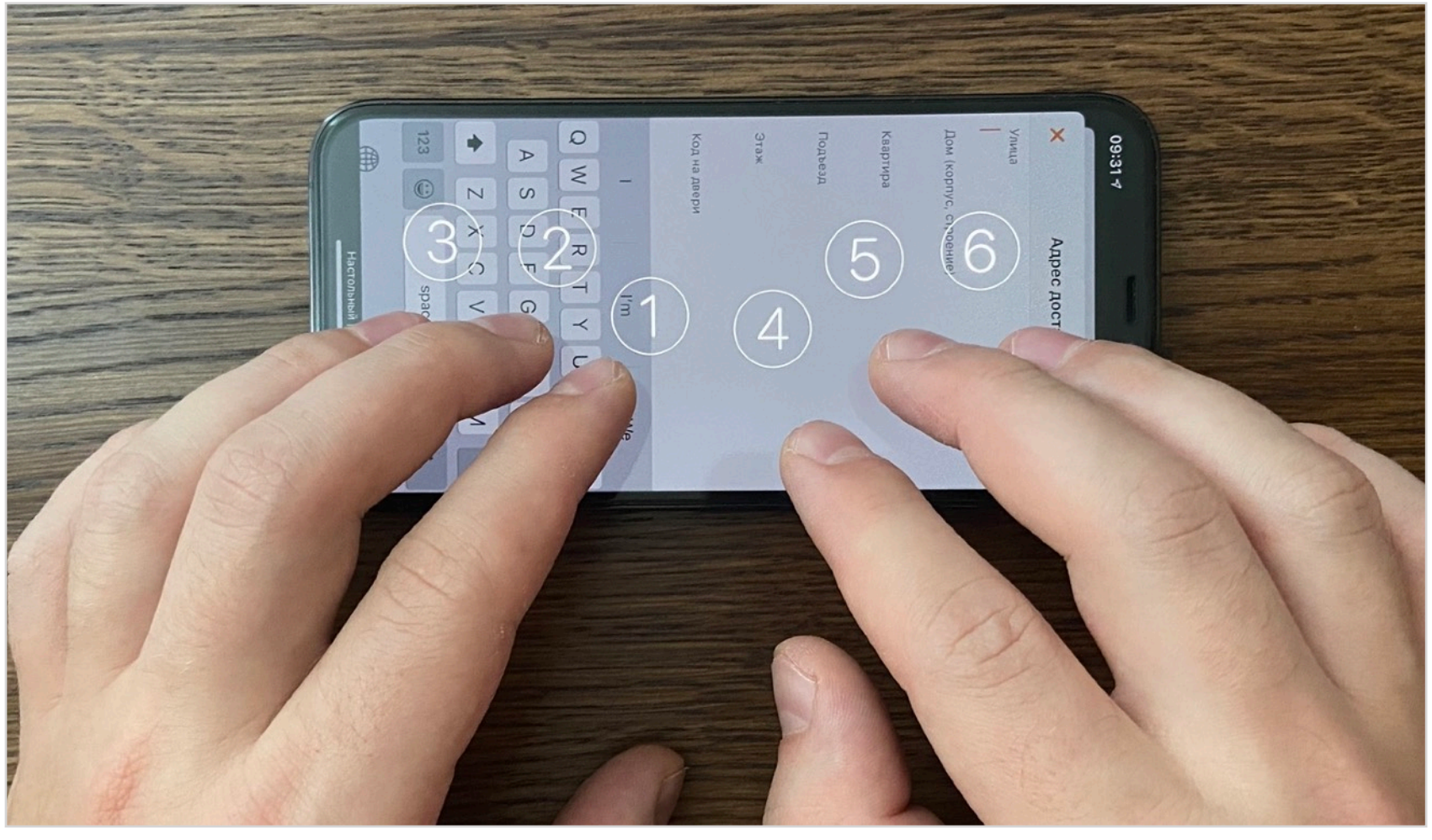
Клавиатура включается через ротор, но сначала включите ее в настройках ротора. При активации клавиатура Брайля покажет 6 точек по краям экрана. Чтобы набрать символ нажмите несколько кнопок одновременно, например, для набора буквы М нажмите 1, 3 и 4. Попадать в цифры на экране не нужно: достаточно попасть пальцами по экрану, а VoiceOver сам поймет набор цифр и букву.



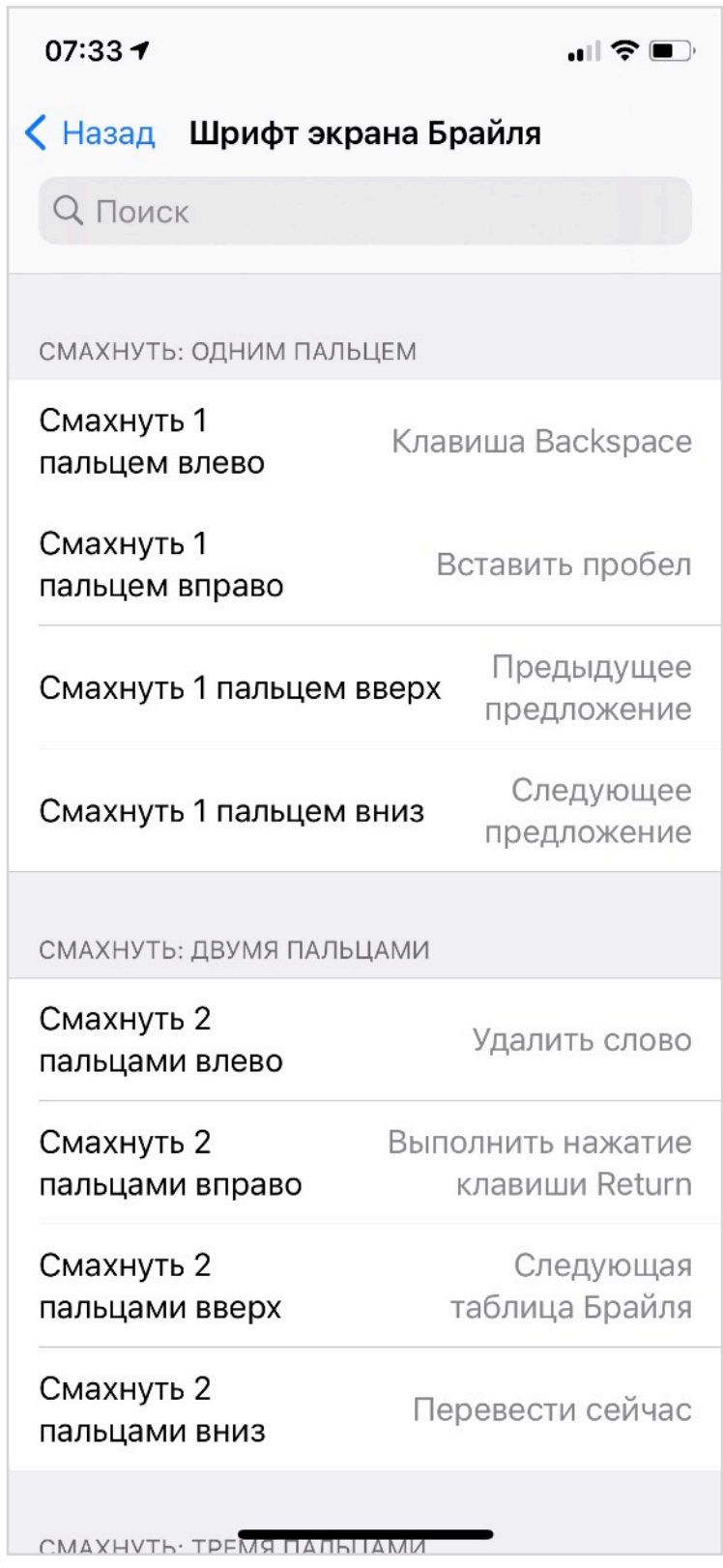
Интересен способ удержания телефона в этот момент: столбики цифр на экране в другом порядке, но их проекция на спинку телефона совпадает с тем, как они читаются, поэтому нужно повернуть телефон экраном от себя.



Есть другое расположение кнопок, оно появляется, когда телефон лежит на столе. Соответствие пальца номеру при этом не изменится, нажимать можно так же приблизительно.



Для ввода частых символов есть специальные жесты: пробел, новая строка, удаление слова и т.п.

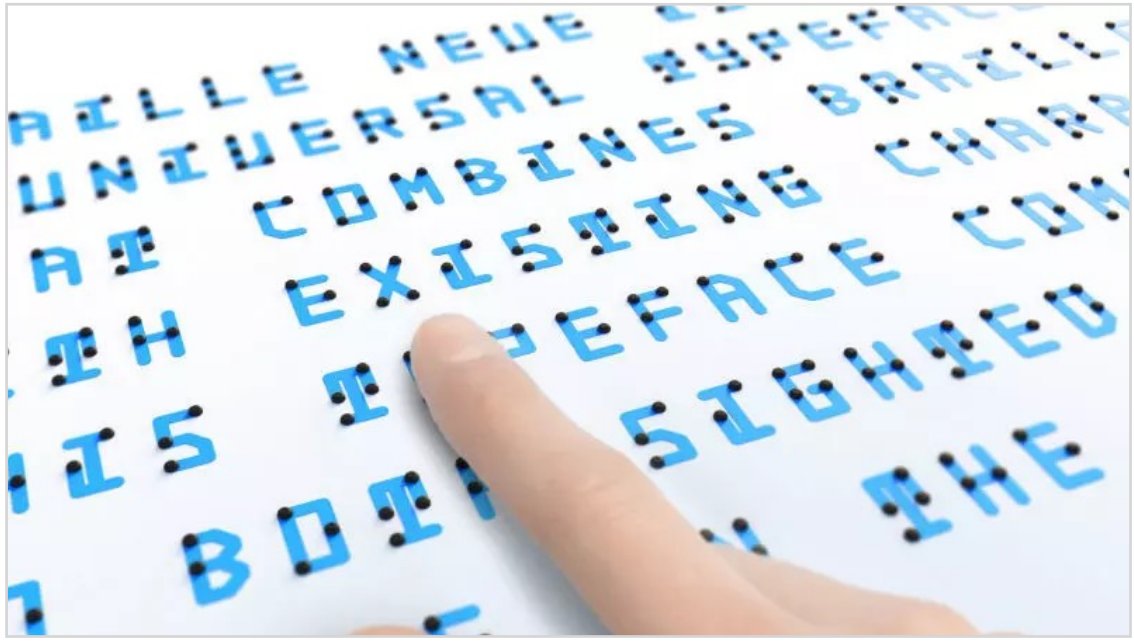


Правильную работу текстовых полей надо проверять с помощью экранной клавиатуру Брайля. Например, если вы программно переставите фокус на другой элемент, а клавиатура Брайля при этом не закроется, то человек окажется в странном состоянии интерфейса.

Пара интересных фактов про алфавит Брайля:

1. Можно уметь писать на шрифте, для этого его надо выучить. При этом можно не уметь на нем читать, ведь для этого надо тренировать чувствительность пальцев.

2. Существует шрифт Braille Neue, который пытается объединить два алфавита: латинский и Брайлевский.



Шрифт Braille Neue объединяет латинский и Брайлевский алфавиты

3. Еще есть Брайлевский дисплей: он управляет точками и может выводить слова, которые можно ощутить тактильно, очень важный инструмент в жизни слепоглухих. VoiceOver может выводить описание интерфейса на такой дисплей. На картинке устройство выводит до 14 символов за раз.

Фото Брайлевского дисплея

Набор текста

Физическая клавиатура

Вряд ли много людей подключают клавиатуру к телефону, а вот работать с клавиатурой и айпадом вполне нормально. Физическая клавиатура дает больше возможностей при работе с VoiceOver: набирать текст легче, а для всех жестов есть хоткей-аналог.

Есть специальная клавиша, чтобы различить ввод текста от команды для VoiceOver, ее нужно зажать перед нажатием кнопки с командой. Обычно это Control+Option, но можно поменять на CapsLock: Настройки → Доступность → VoiceOver → Набор текста → Клавиша модификации → Верхний регистр. Такой хоткей называют VO-клавишей.

При навигации с клавиатуры стрелки работают как и свайпы: горизонтальные переключат фокус на следующий элемент, а вертикальные меняют действие, управляют элементом регулировки или ротором.

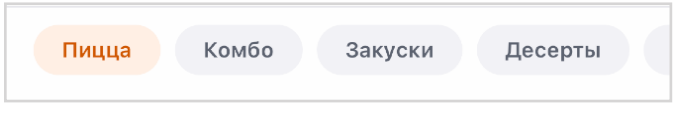
— Для активации элемента зажмите клавишу VO, а затем нажмите пробел. Клавиша Esc работает как жест скраб.
— Скрол экрана проводится с помощью вертикальных стрелок, зажав Alt.
— Tab ставит фокус в поле поиска, видимо, находит по трейту .searchField.
— Конкретные действия ротора вызываются прямо с клавиатуры — горячих клавиш для этого много.

О всех сочетаниях клавиш можно узнать на сайте Apple и переназначить их как вам удобно

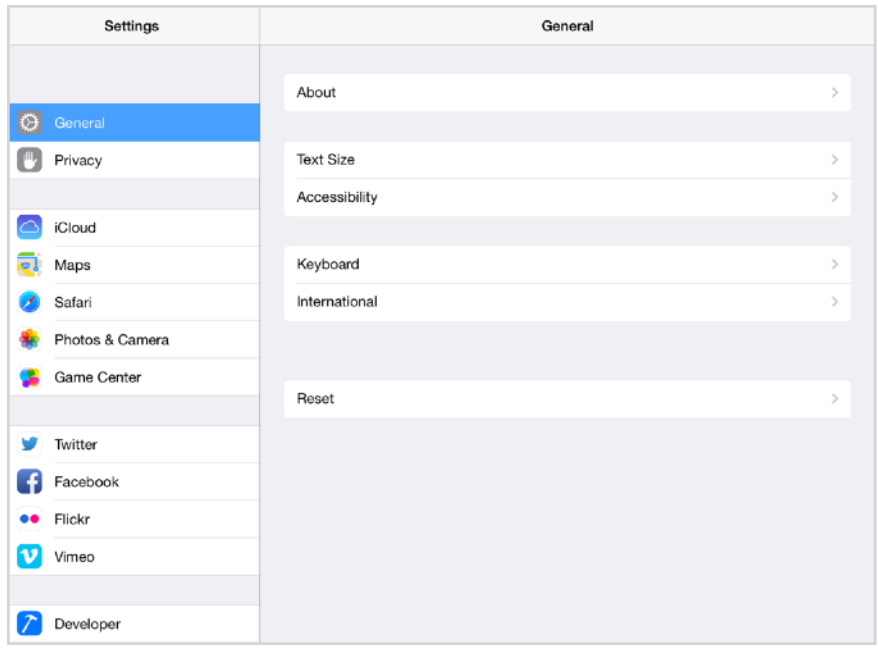
SelectionFollowFocus

Обычно элементы не выбираются в тот момент, когда на них попадает фокус, но иногда это удобно. Если хотите, чтобы элемент сразу выбирался, то воспользуйтесь свойством selectionFollowFocus, которое есть у `UITableView` и `UICollectionView`.

Например, переключатель типа продуктов сразу может пролистывать список под ним, а мастер-таблица в `UISplitViewController` может заменять содержимое детального контроллера.



При смене фокуса на типе продуктов хочется сразу выбирать его

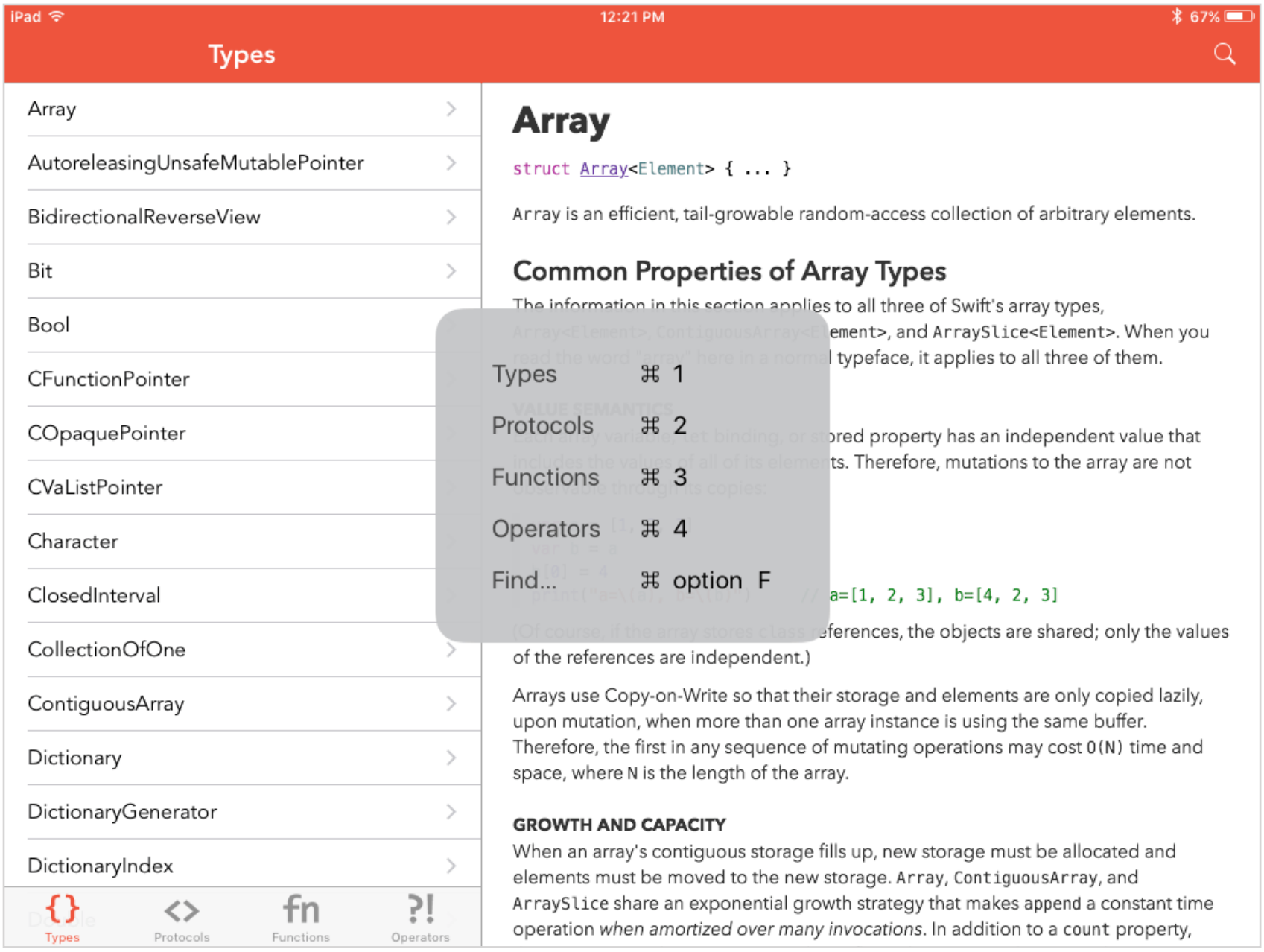


Выбор элементов в левой половине влияет на содержимое правой

Если вы знаете, что вашим приложением пользуются с клавиатуры, то вы можете улучшить опыт, добавив шорткаты для действий.

Зажав Command вы увидите полный список шорткатов, но работает это только на айпаде, если вы указали шорткату `discoverabilityTitle`.

Почитать больше про шорткаты можно у NSHipster.



Скриншот с сайта NSHipster.com

Пример

Мы разобрали каждый компонент VoiceOver, настало время посмотреть, как это работает в связке. Мы пройдемся по экранам Додо Пиццы и разберем как должна выглядеть адаптация приложения, какой она бывает и чем дизайнер может помочь прямо в макете.

Кода будет мало, будем обсуждать на уровне ментальной модели. Эту часть работы может выполнять дизайнер, если он понимает, как работает VoiceOver. Хороший фронтенд-разработчик тоже справится.

Пример

Ментальная модель

Главный инструмент, который поможет задизайнить взаимодействие с VoiceOver — это простой текст. Для начала рядом с макетом достаточно написать текст, который VoiceOver будет озвучивать. Такое простое упражнение поможет вам создать ментальную модель каждого экрана приложения.

Прежде чем всё подписать, задайте себе следующие вопросы:

— С какими элементами можно взаимодействовать на этом экране? Как?
— Как уменьшить количество элементов?
— Как работает навигация на экране?

После ответа на эти вопросы у вас появится понимание, что и как подписать. А это сильно упростит работу программистам.

Текст должен быть лаконичным, ведь он превратится в звук, а скорость восприятия на слух медленнее, чем скорость чтения взглядом. Кроме того, длинный текст может быть назойливым. Не нужно писать особенный текст для VoiceOver, достаточно правильно озвучить существующий видимый.

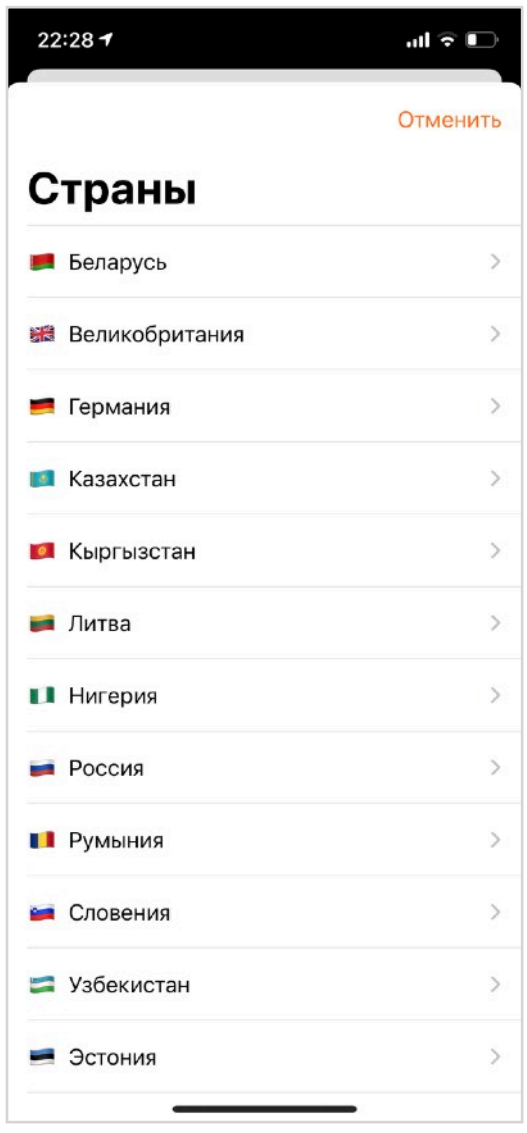
При адаптации каждого экрана набор действий одинаковый:

— подписать все элементы;
— проставить трейт «кнопка» ячейкам в списках;
— уменьшить количество контролов;
— добавить другие трейты при необходимости;
— проверить навигацию по экрану;
— проверить сценарий пользователя и подумать как сделать удобней.

Разберем каждый экран приложения Додо Пиццы, отметим все, что пришлось сделать на экране и как это может быть подписано в макете.

Пример

Страны и города



Страны

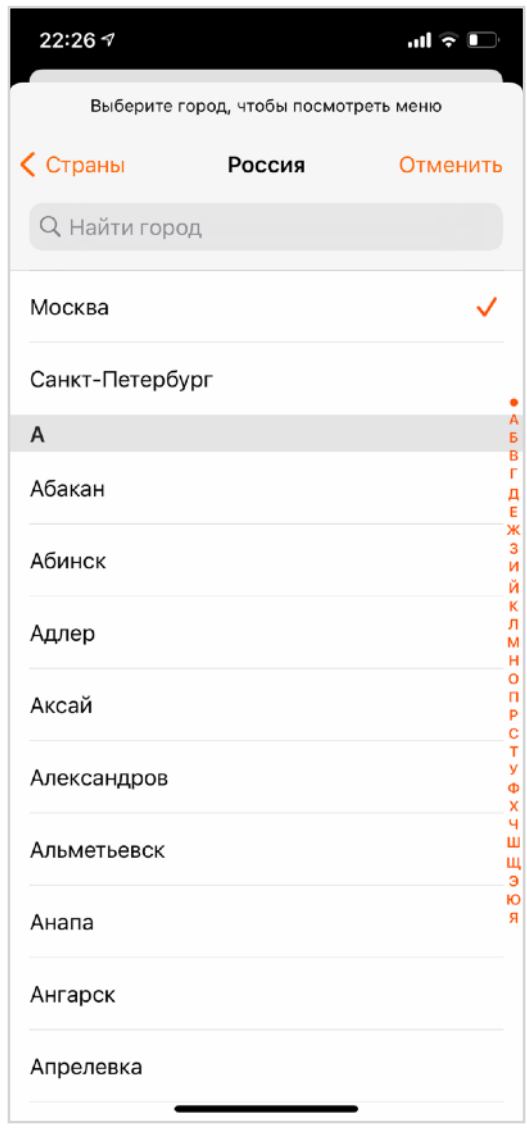
После установки приложения нужно выбрать страну и город. Страну мы определяем по текущей локализации пользователя, поэтому многие этот экран не видели.

На вид это самый простой экран, но даже у него нашлась проблема для VoiceOver: флаги представлены эмоджи-текстом и читаются вместе с названием страны, что создает дублирование.

Ячейки сверстаны стандартным стилем, по наличию стрелки-шеврона `UIKit` понял, что перед нами кнопка и добавил трейт `.button` самостоятельно.

Города

С выбором города все просто, но пришлось добавлять `.button` и трейт `.selected`, когда ячейка выбрана.



Для удобной работы с огромным списком городов мы добавили алфавитный переключатель. Он сразу адаптирован: это элемент регулировки, вертикальные свайпы листают список до нужной буквы.

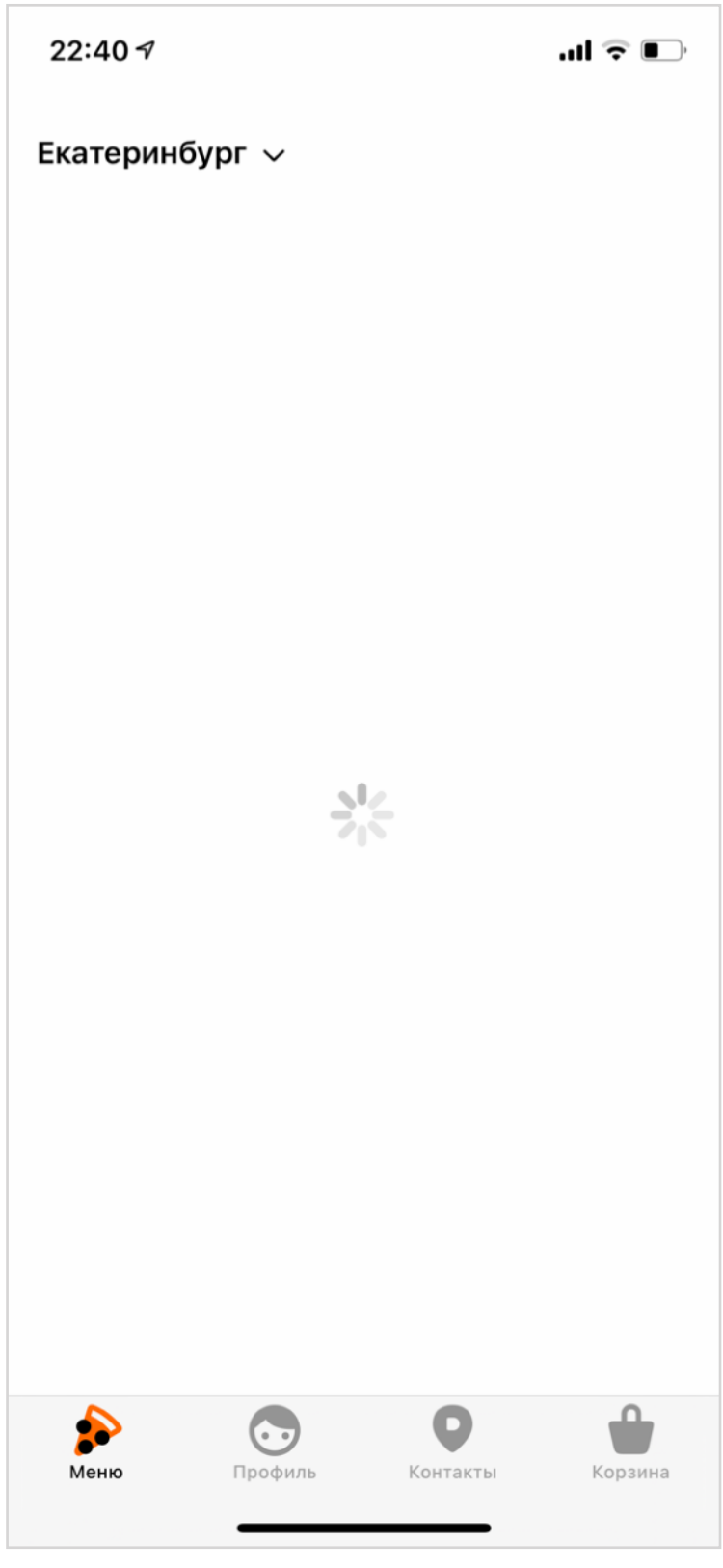
Чуть сложнее дела с поиском: при вводе текста мы фильтруем список городов, поэтому после ввода каждой буквы нужно сообщать о количестве видимых вариантов. Если ничего не нашлось, то об этом тоже надо сказать.

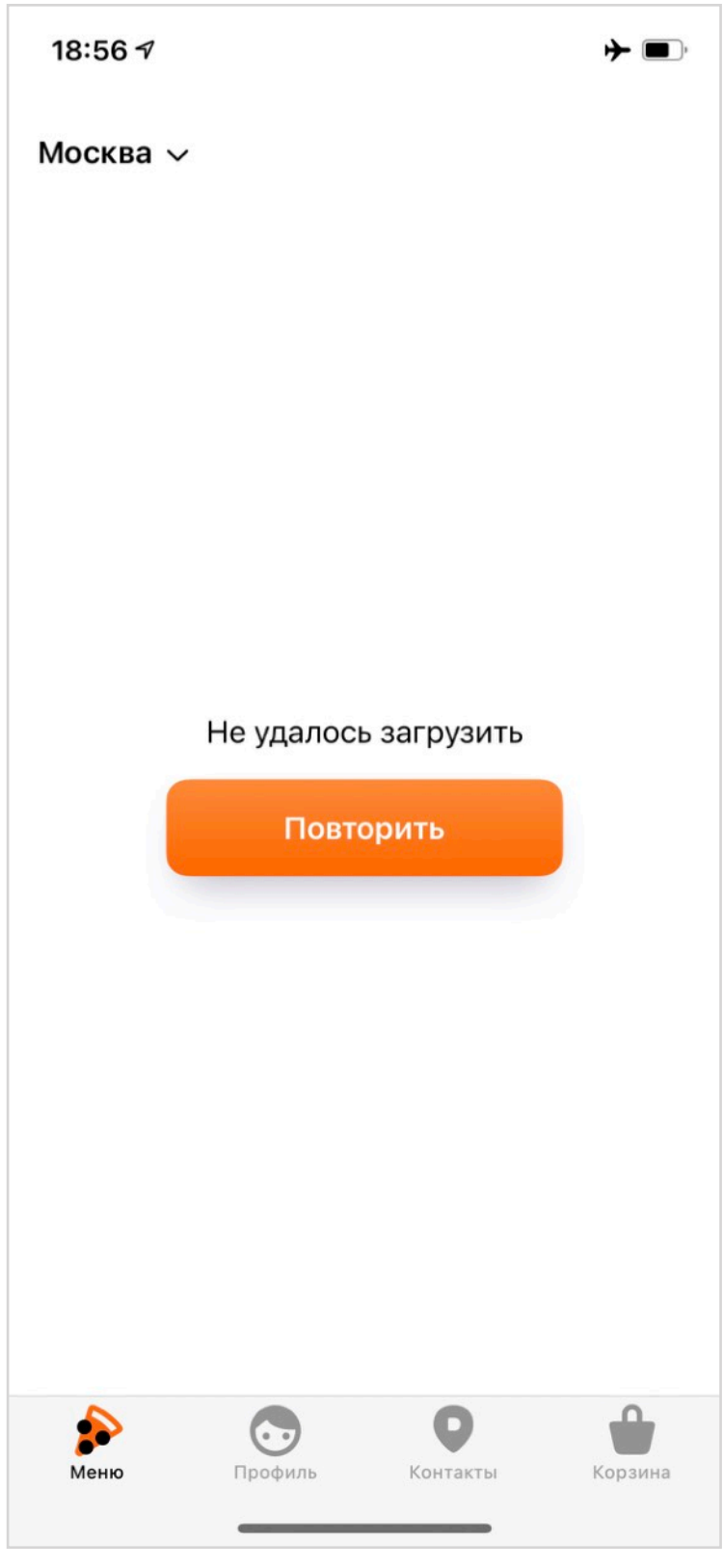
Пример

Экран меню

Экран меню — первый экран с явной загрузкой. При загрузке нужно сообщить, что происходит, поэтому при ее появлении мы сообщаем «Загружаю...». Это стандартный текст в компоненте дизайн-системы, лучше бы его уточнить до «Получаем меню». Можно поставить фокус на индикатор загрузки, тогда он сообщит стандартное «идет выполнение операции».

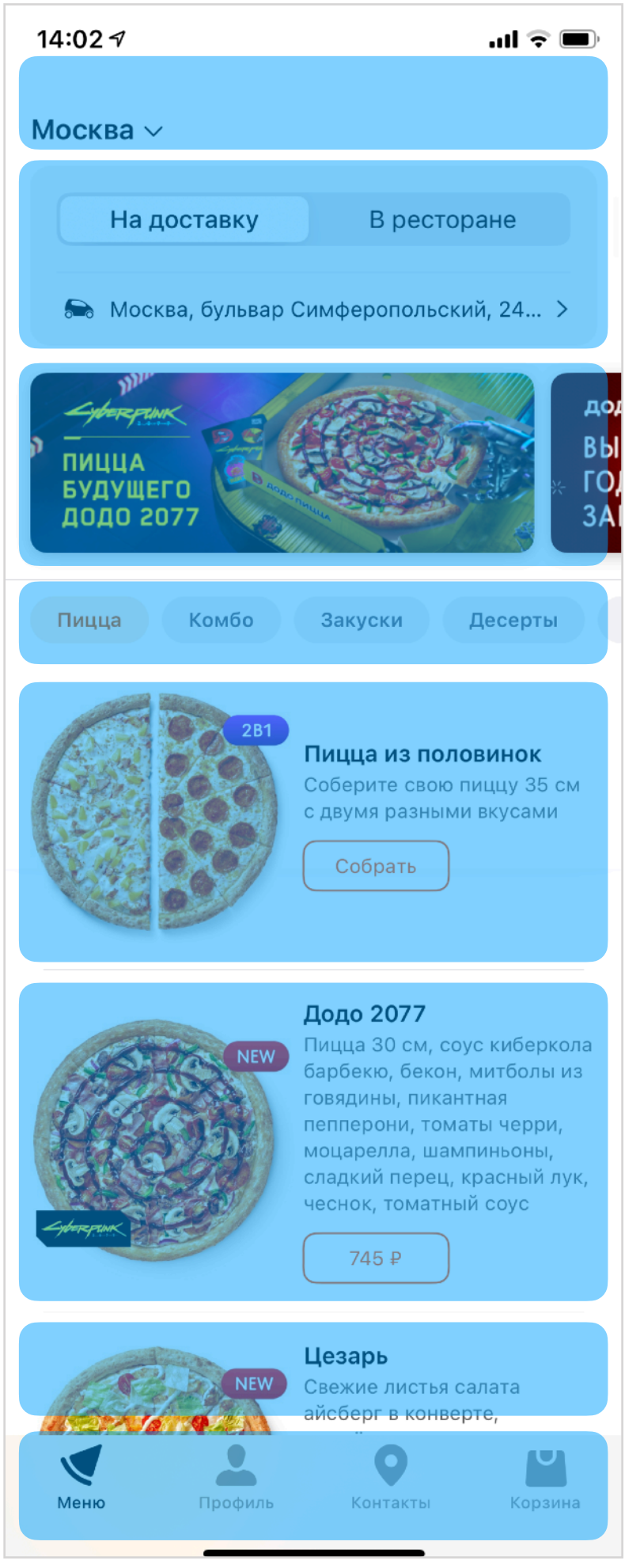
Если загрузить меню не получилось, то ставим фокус на надпись «Не удалось загрузить».





После загрузки появляется экран с типом заказа, акциями и списком продуктов.

На экране много элементов и первая задача адаптации — уменьшить их количество. Идеально упростить все до вертикального списка из элементов.



1.Переключатель «На доставку/В ресторане» можно сделать `.adjustable` элементом. Кнопку адреса стоит оставить отдельной, но скрыть все иконки.

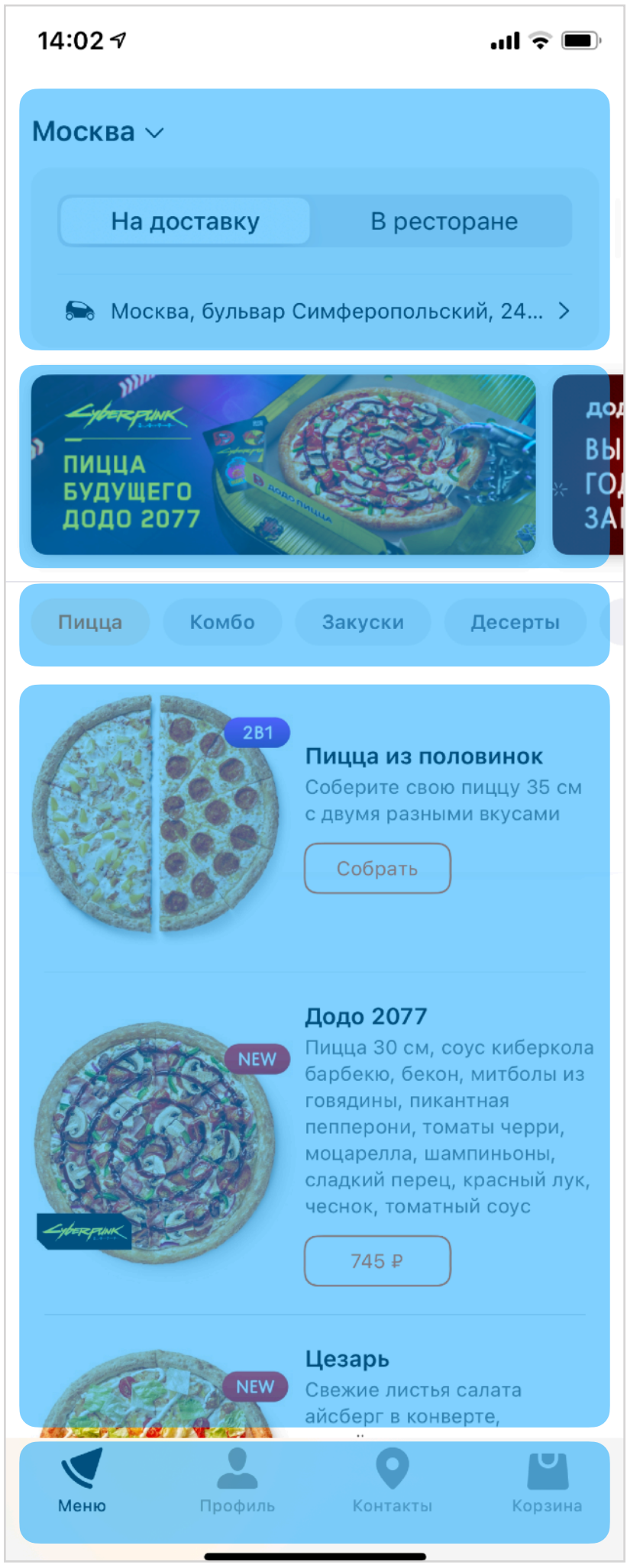
2. Карусель акций и типов продуктов укрупнить до `.adjustable` контрола, как описано в главе «Карусель».

3. Ячейка продуктов должна стать одним элементом доступности, без фокуса на отдельных контролах внутри нее.

4. Таббар мы не меняли, он доступен по умолчанию: VoiceOver называет текущий таб, его номер и сколько их всего.

В конце прокачали навигацию на экране, чтобы пользоваться стало проще.

Разметили на экране явные контейнеры: адрес, акции, список продуктов, вкладки. По ним можно будет переключаться вертикальными свайпами в режиме ротора «контейнеры».



Адрес

Акции

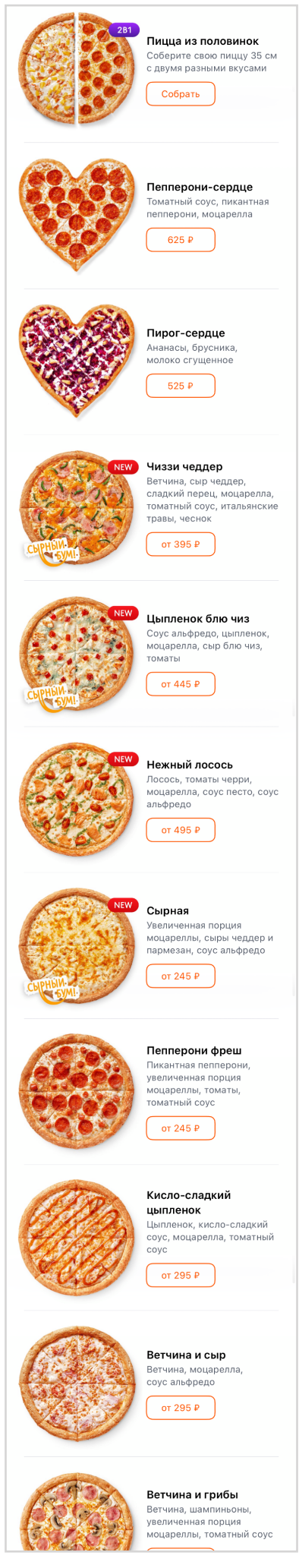
Тип продуктов

Меню

Вкладки

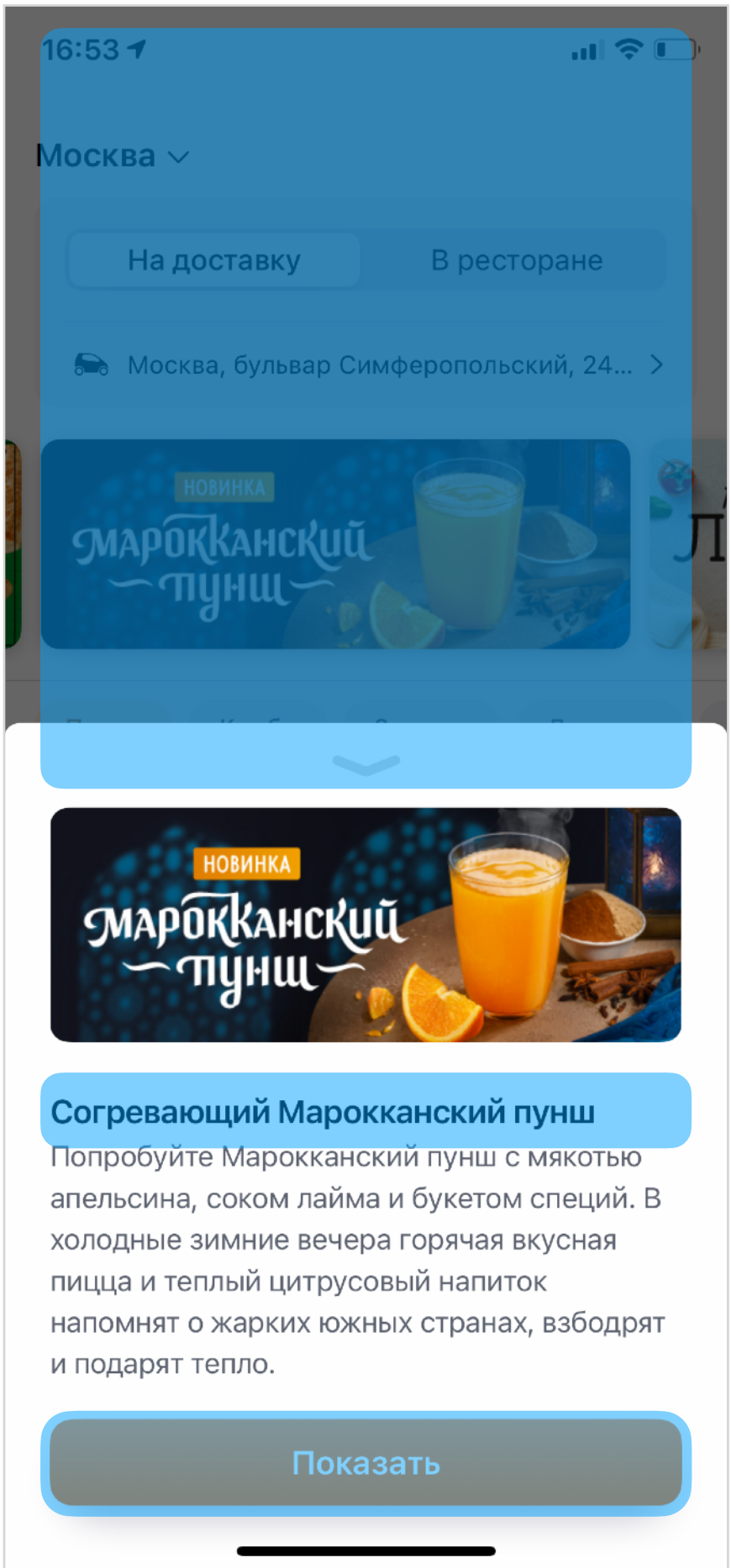
Список продуктов очень длинный. Чтобы удобно по нему перемещаться, выполним две задачи:

— Кратко опишем видимые элементы и расскажем о них после скрола.
— Напишем кастомный ротор, который бы вертикальным свайпом меняет тип продуктов.



Пример

Акции



Окно акций простое: подписать кнопку закрытия и проставить трейт `.header` для заголовка.

Самое главное для модального окна — обработать модальность и работу фокуса, чтобы он переключался на окно при открытии и не выходил за его пределы.

Карточка открывается не во весь экран, а анимацию перехода мы написали сами. Чтобы не фокусироваться на кнопке закрытия добавим жест скраб.

Явное действие привяжем к Magic Tap и расскажем о нем в подсказке к кнопке. Если кнопка не ведет на другой экран, а, например, применяет промокод, то сообщим о результате через оповещение. Чтобы оповещение не перебивалось повторным прочитыванием названия кнопки, поставьте для кнопки трейт `.playsSound`.

Пример

Карточка продукта

Сложные экраны я адаптирую в несколько проходов. Сначала разбираюсь с очевидными проблемами:

— подписываю кнопки с иконками,

— применяю к контролам .adjustable,

— укрупняю ячейки до одного элемента, а если ячейка выбрана, то ставлю ей трейт `.selected`.

Скорее всего, критичные проблемы еще будут в навигации:

— порядок обхода часто ломается, если есть нависающие над скролом элементы;

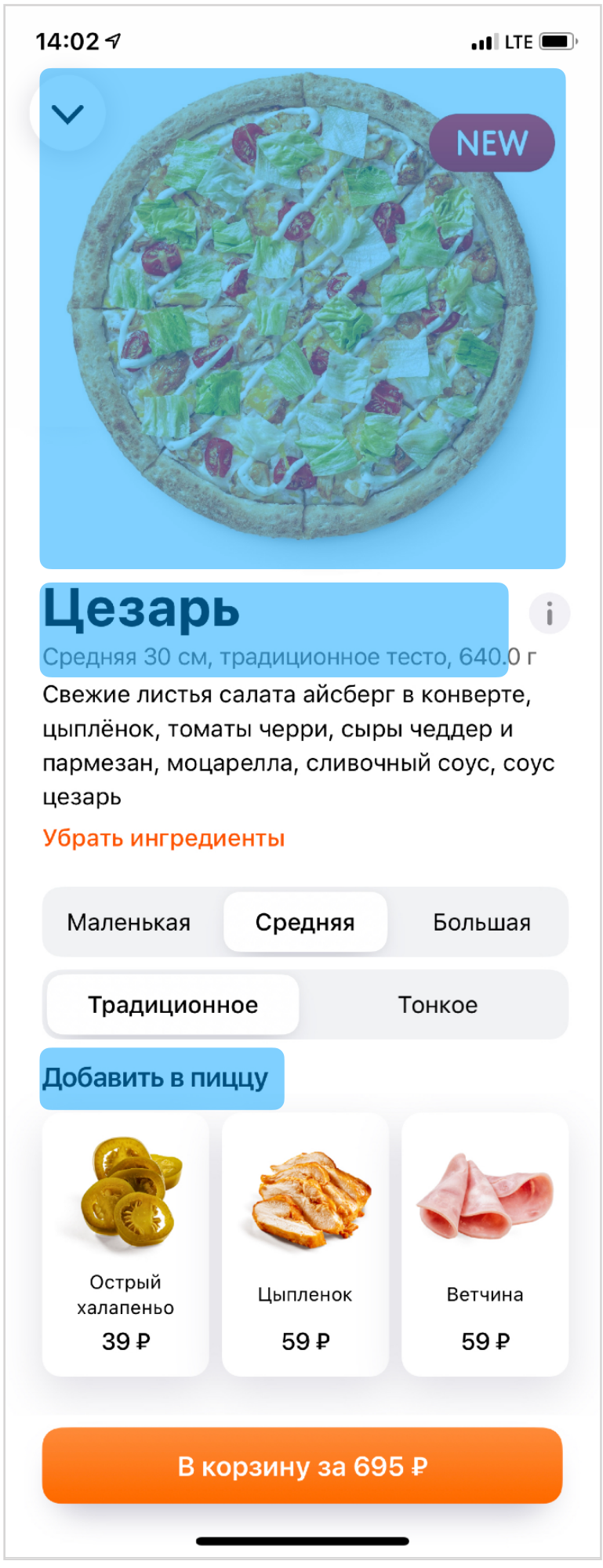
— указать, где заголовки и контейнеры;

— обработать скраб и меджик тап.

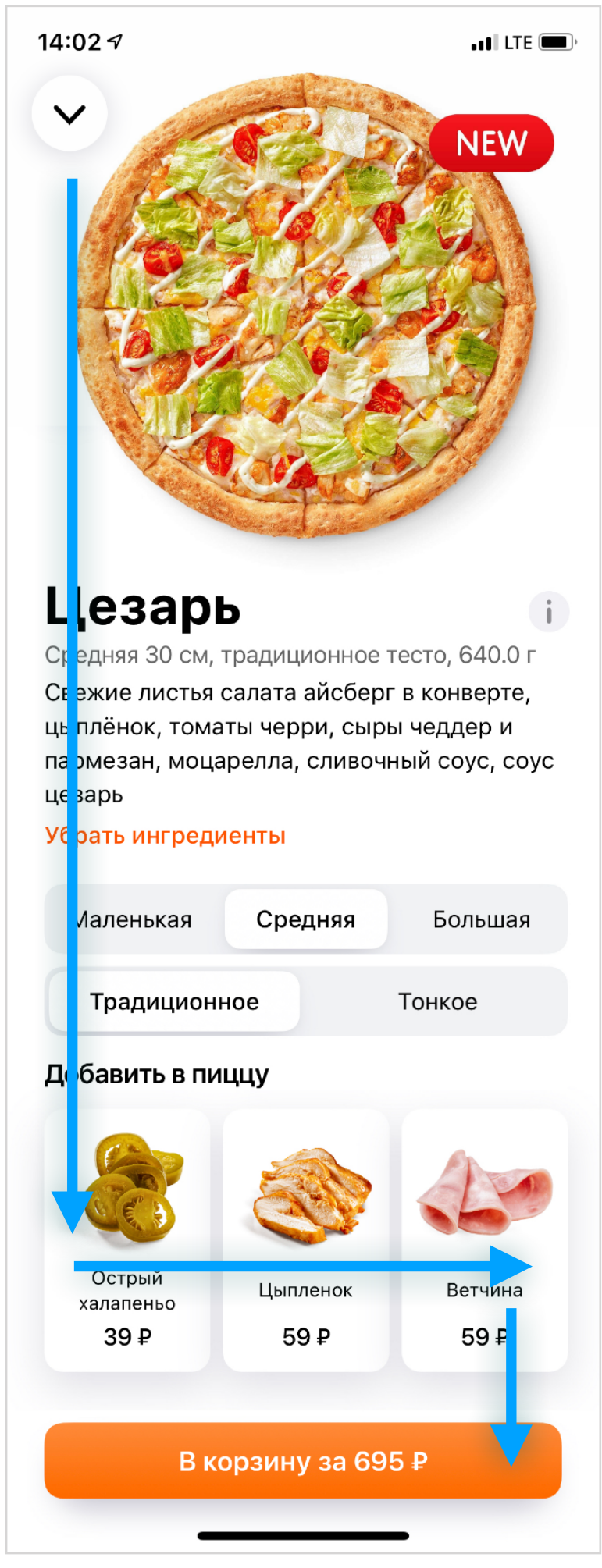
В конце можно заняться логических задач. Первая — кнопка покупки должна сообщать цену, это у нас есть и в графическом дизайне. Рассказывать о всех выбранных настройках надо через `customContent`. Вторая — при изменении размера или добавлении топинга надо называть новую цену.

Выбирая пиццу, незрячий может показать картинку своему зрячему другу, а для этого он должен знать, что картинка есть. Тогда картинку надо включить как доступный элемент. В этот раз в картинке оказался полезный бейджик NEW, о нем точно нужно рассказать в описании картинки.

Для правильной работы ротора укажем заголовки и контейнеры. Для навигационных кнопок пришлось сделать невидимый контейнер.

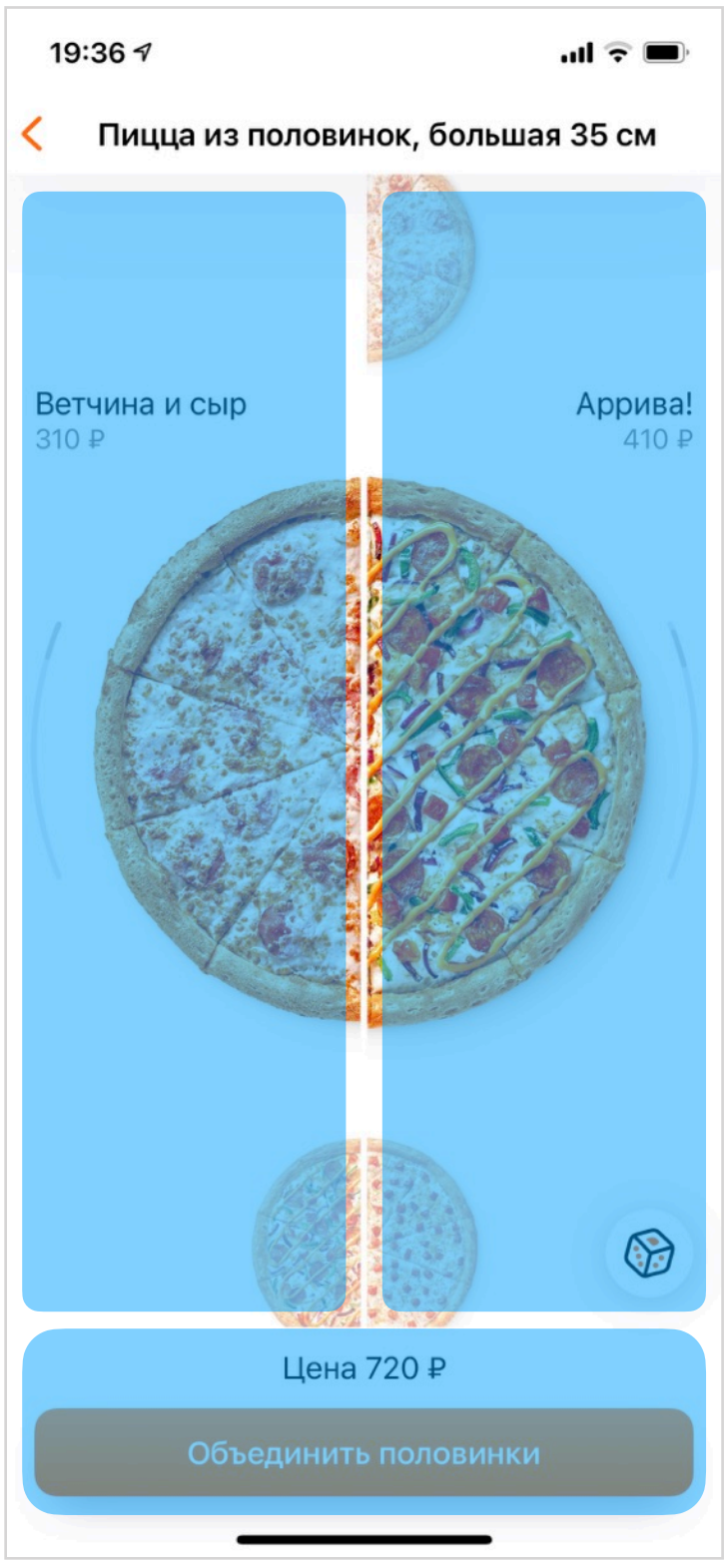


Карточка описания больше, чем размер экрана, поэтому она скролится, но несколько элементов зафиксированы и не перемещаются. Такая иерархия сломает порядок обхода — поправим через accessibilityElements.



Пример

Половинки



У нас есть очень необычный экран — конструктор пиццы из половинок. Это два коллекшена, скролл которых анимирован.

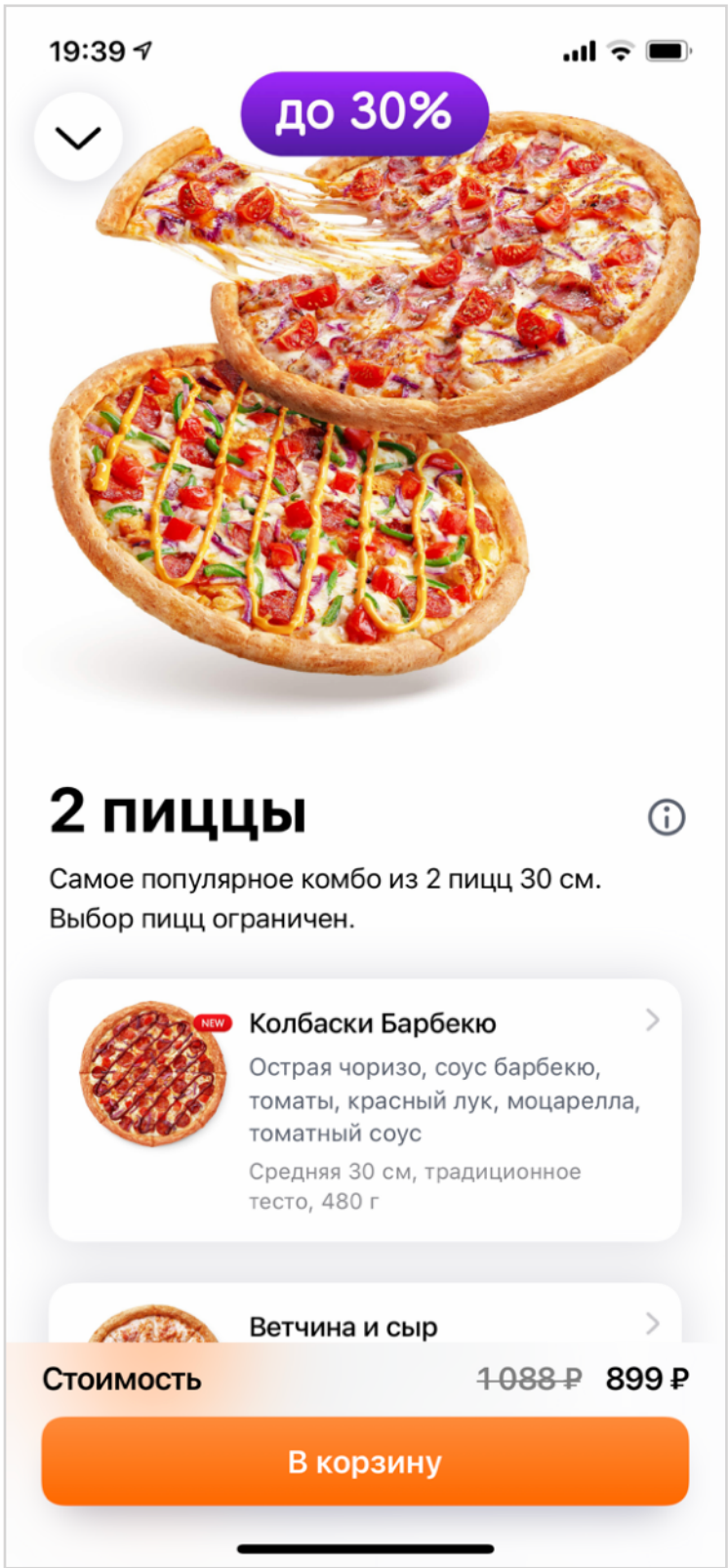
Посмотреть на ютубе.

При визуальной сложности это всего лишь два `.adjustable` элемента. Забавно, что в таком случае вертикальный свайп VoiceOver совпадает с направлением свайпа у зрячего человека.

Цену и кнопку «перемещать» можно унести в `value` и `customAction` для кнопки «Объединить половинки».

Пример

Конструктор комбо



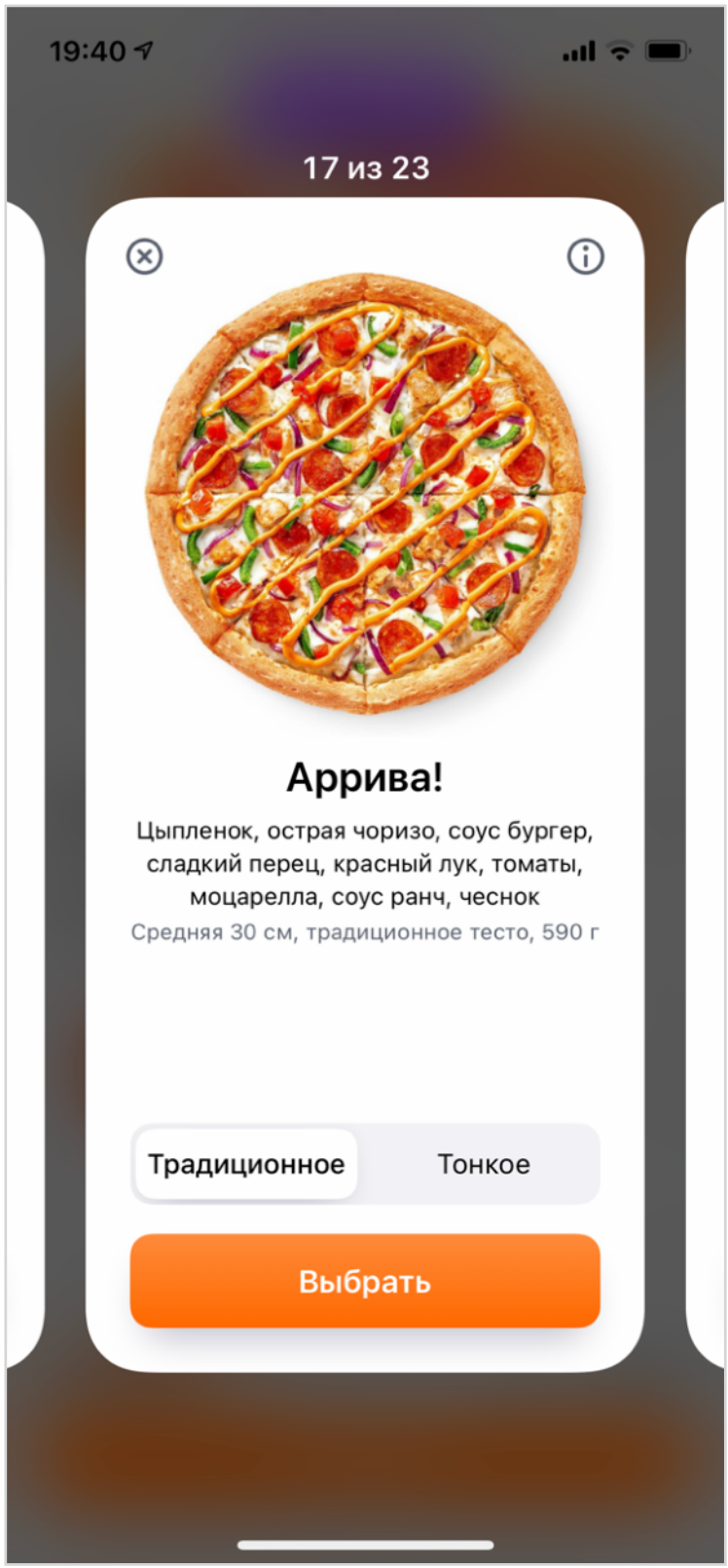
Среди меню есть комбо: набор из нескольких продуктов на выбор.

Посмотреть на ютубе.

Карточка с описанием конструктора простая: вертикальный список, где каждая ячейка должна быть одним элементом доступности.

Закрыть, кнопка

Тесто традиционное, 1 из 2 элемент регулировки



Энергетическая ценность, кнопка

Песто

Цыпленок, острая чоризо, соус бургер, сладкий перец, красный лук, томаты, моцарелла, соус ранч, чеснок

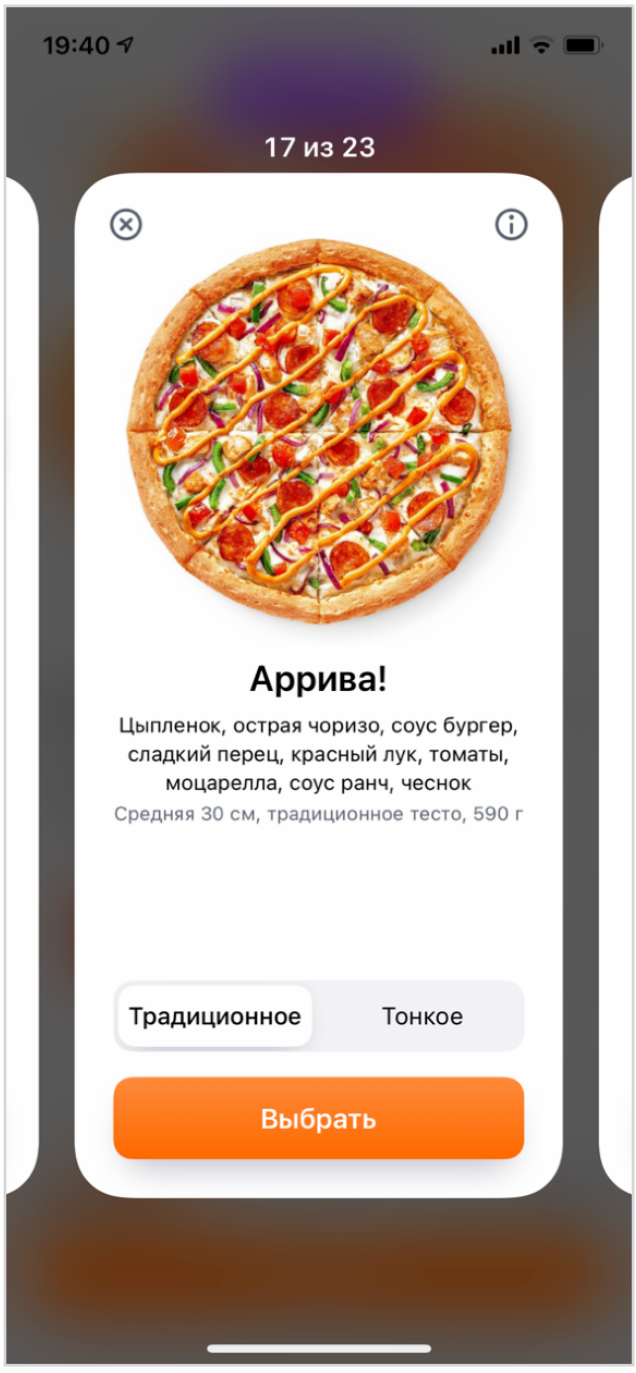
Средняя 30 см, традиционное тесто

Выбрать, кнопка

В конструкторе комбо очень много элементов, хотя визуально это просто горизонтальный список, где внутри каждой ячейки несколько контролов: описание, кнопки, сегмент контрол.

Контролов много, но модель простая: я выбираю один из продуктов и могу настроить у него тесто. Еще могу узнать пищевую ценность, состав и размер, но это не так важно.

Для начала я подписал все данные прямо на макете. Описание — черным, действия — оранжевым. Сейчас это просто перечень контролов, для некоторых дополнительно подписал их тип. Можно было бы оставить на таком уровне, но попробуем из карточки продукта сделать единый доступный контрол, а из всего конструктора простой сценарий.



Песто

Цыпленок, острая чоризо, соус бургер, сладкий перец, красный лук, томаты, моцарелла, соус ранч, чеснок

Средняя 30 см, традиционное тесто

Выбрать, кнопка

Тесто традиционное, 1 из 2
элемент регулировки

Энергетическая ценность, кнопка

Закрыть, кнопка

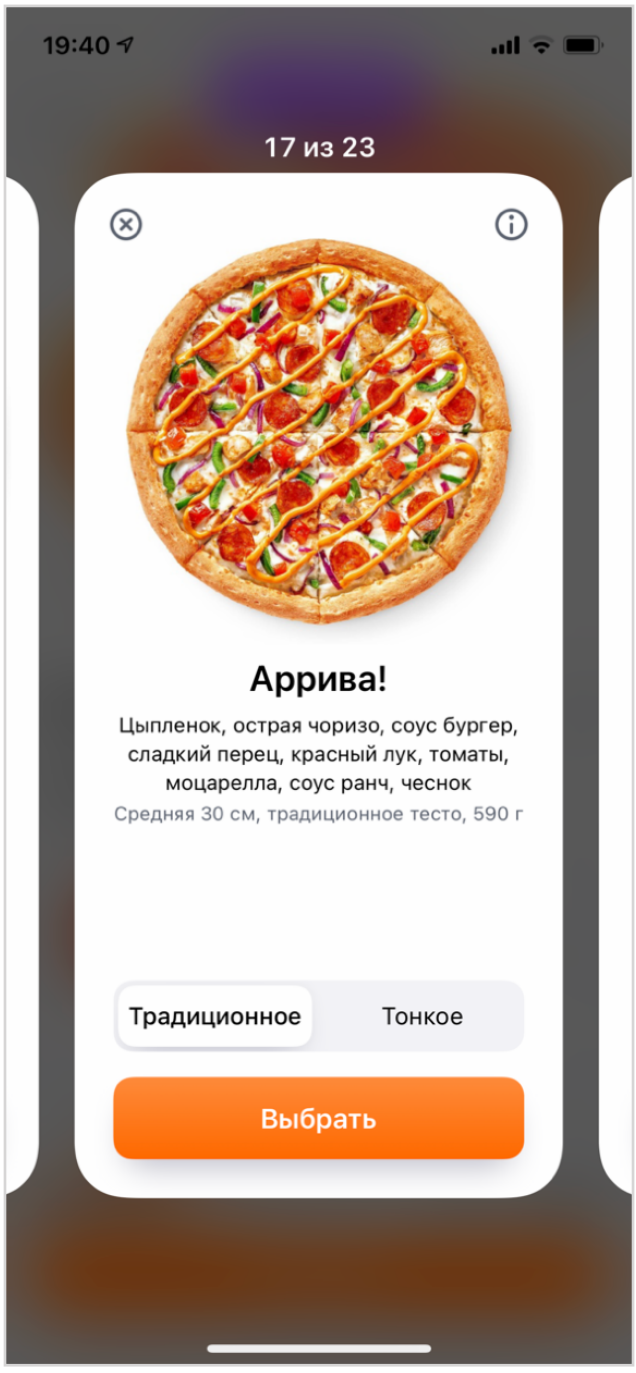
Ментальная модель

Составил список: сначала описание, потом действия. Описание надо будет сгруппировать в `label` и `value`.

Действий в карточке много. Самые главные — выбор и смена типа теста. Для действий внутри карточки подойдут `customAction`, будут доступны по вертикальному свайпу.

Надо решить, как будем переключать карточки. Первое, что приходит на ум — использовать `.adjustable`, тогда продукт можно будет переключать вертикальными свайпами. Но это конфликтует с контекстными действиями, значит от них нужно либо избавиться, либо придумать другой жест.

Карточки — это горизонтальный `UICollectionView`, а значит его можно листать, свайпая тремя пальцами. Это необычное поведение, про него нужно рассказать в подсказке. Увы, она может быть отключена.



Песто. 4 из 23

Цыпленок, острая чоризо, соус бургер, сладкий перец, красный лук, томаты, моцарелла, соус ранч, чеснок

Средняя 30 см, традиционное тесто

Выбрать, кнопка

Тесто традиционное, 1 из 2
элемент регулировки

Энергетическая ценность, кнопка

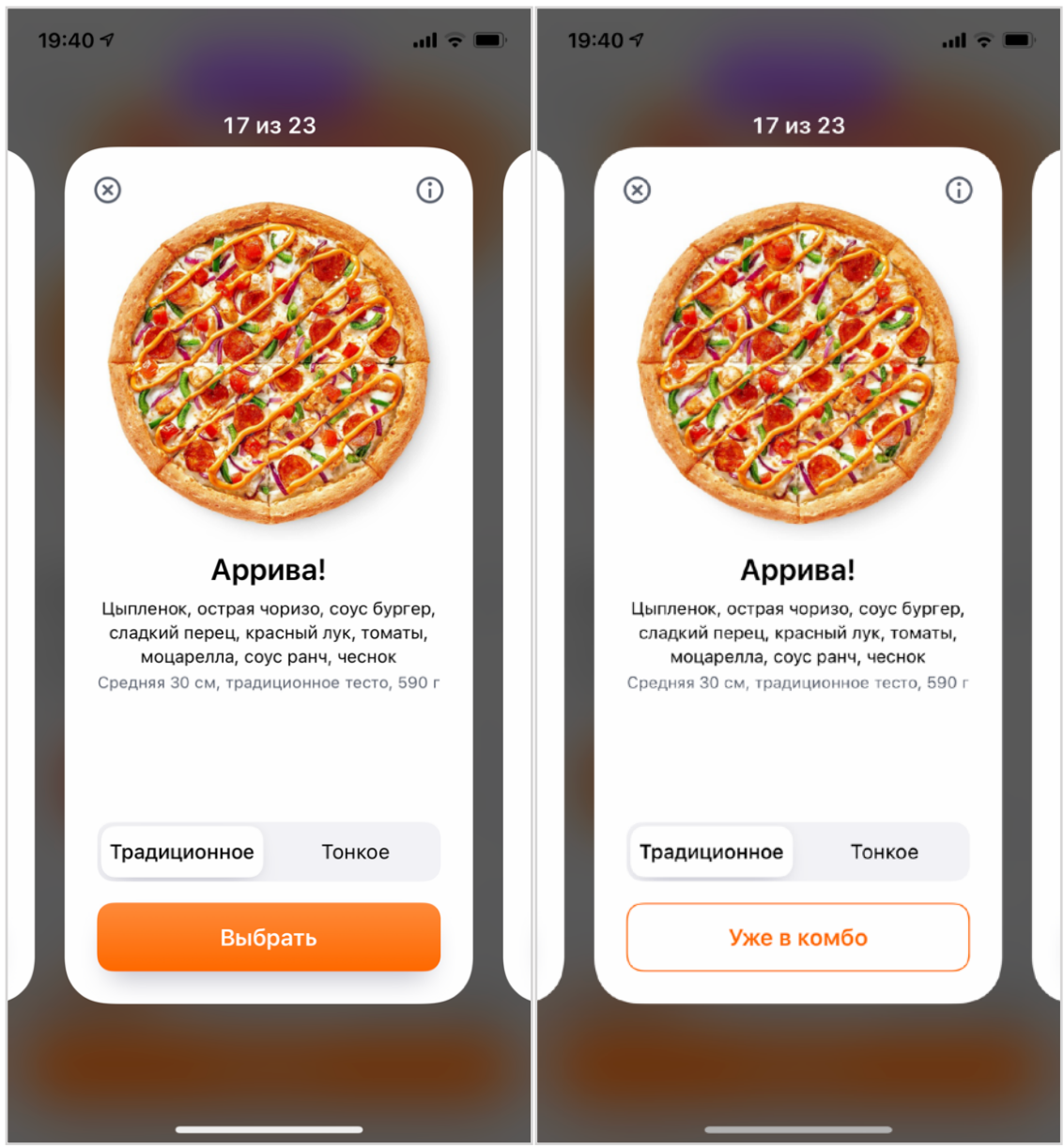
Закрыть, кнопка

Описание

После свайпа надо читать название продукта и рассказывать о его позиции в общем списке. Это будет лейблом. Состав длинный, его стоит унести в value, другая интонация создаст динамику.

Остается размер и тип теста. Тип дублируется в описании и в переключатели, убираю из описания.

Места для размера пока не нахожу, но по дизайну все альтернативы будут одного размера, поэтому нет нужды повторять об этом каждый раз. Наверное, можно будет вынести за скобки, оставлю это на потом.



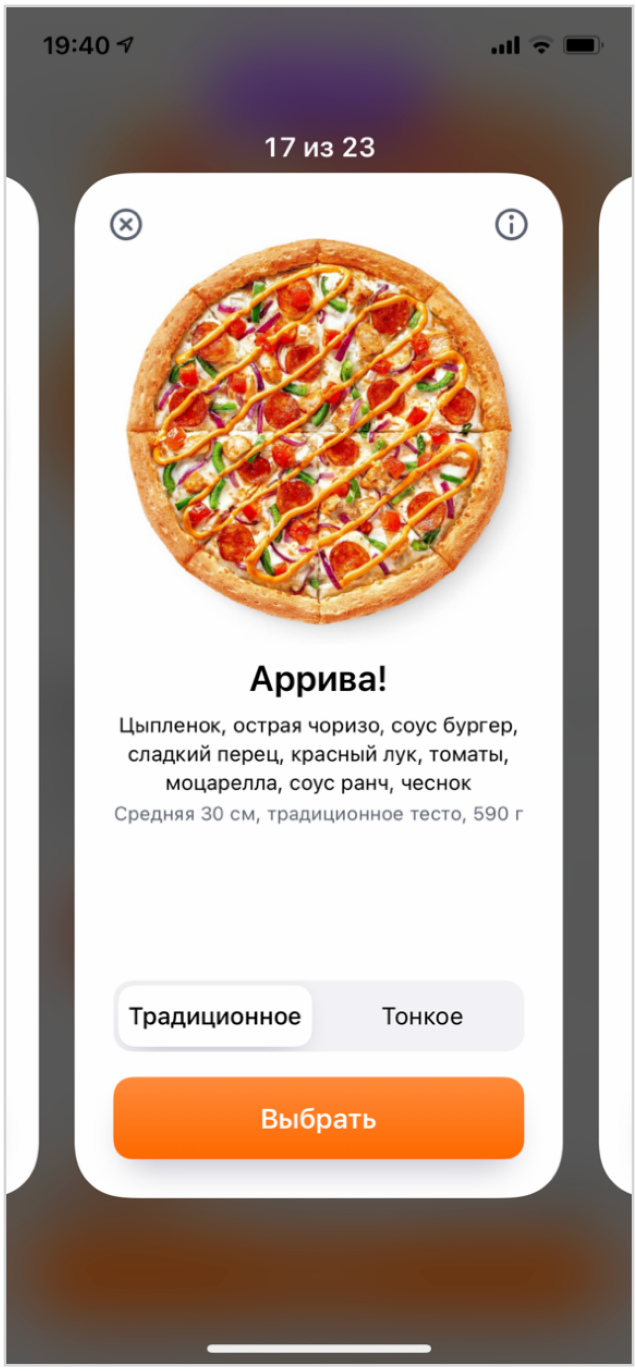
Состояние

Карточка бывает в разных состояниях, это можно указать через трейт. Применить трейт можно прямо к карточке, ведь мы сделали ее единым элементом доступности.

Если бы кнопка была отдельной и сама описывала свое состояние, то только долистав до нее, мы бы узнали, что товар добавить нельзя.

Контейнеры

У нас осталась надпись «30 сантиметров». Прочитать ее достаточно один раз при открытии экрана, потому что у всех продуктов одинаковая размерность. Можно сказать, что это характеристика самого конструктора, поэтому надпись поставим в описание контейнера.



Песто. 4 из 23

Цыпленок, острая чоризо, соус бургер, сладкий перец, красный лук, томаты, моцарелла, соус ранч, чеснок

Средняя 30 см

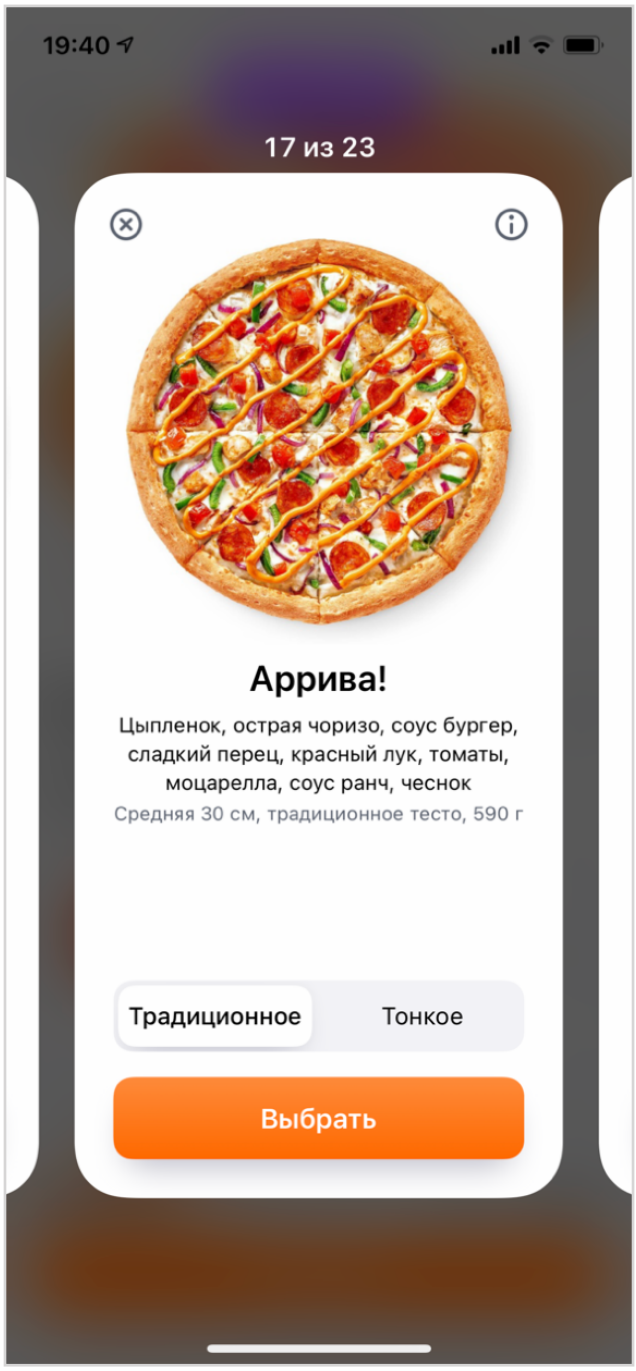
Доступны действия

— Выбрать
— Сменить на тонкое тесто
— Энергетическая ценность
— Закрыть

Контекстные действия

Переключатель теста состоит всего из двух вариантов, поэтому можно заменить его на действие «сменить тип теста», но уточнять на какое. У остальных кнопок просто скроется тип. Теперь действие можно менять вертикальным свайпом, а активировать — двойным тапом.

Закрытие нужно оставить и отдельным жестом, и контекстным действием. Карточка продукта выглядит законченной, доступность элементов внутри карточки можно отключать. Осталось понять, как объяснить переключение между продуктами.



Контейнер: Пицца, средняя, 30 сантиметров

Песто. 4 из 23

Цыпленок, острая чоризо, соус бургер, сладкий перец, красный лук, томаты, моцарелла, соус ранч, чеснок

Средняя 30 см

Доступны действия

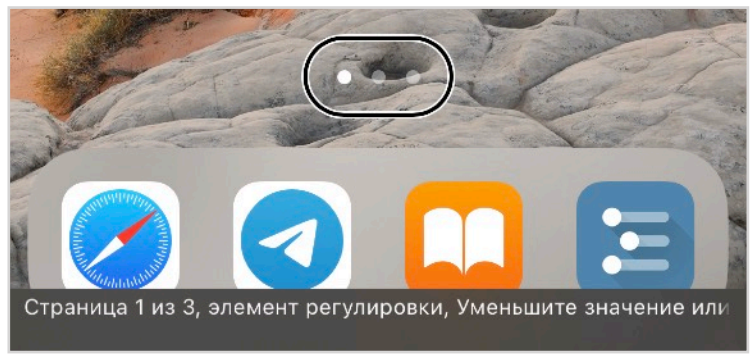
- Выбрать
- Сменить на тонкое тесто
- Энергетическая ценность
- Закрыть

Навигация

Нам удалось запихнуть все элементы в карточку. Кроме текущей карточки на экране ничего нет, поэтому переключиться на следующую можно горизонтальными свайпами.

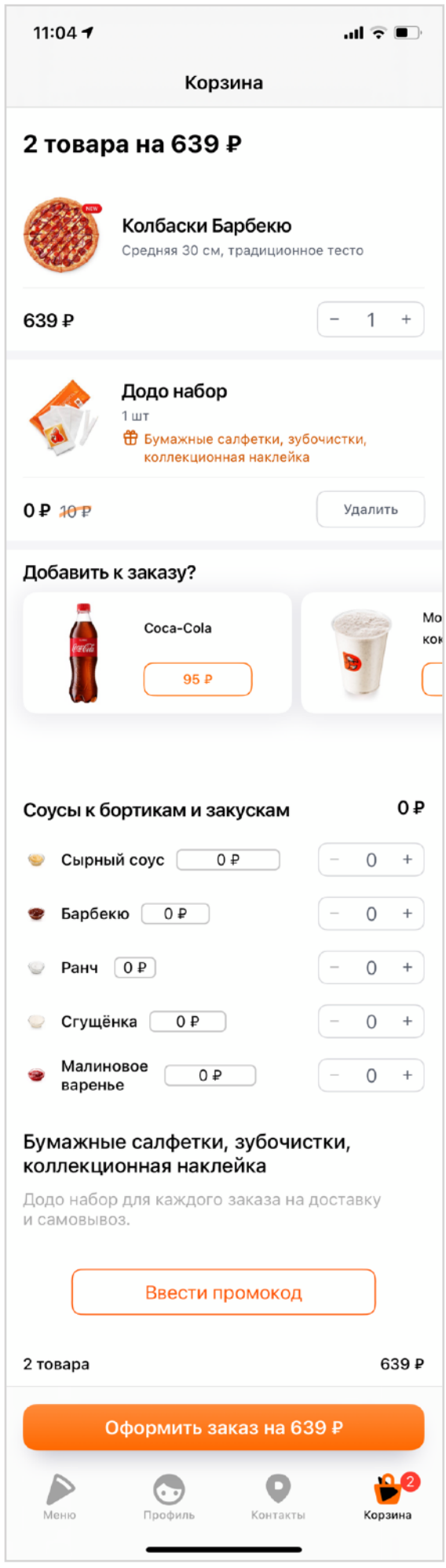
При этом работает и скрол тремя пальцами, он пролистнет один экран и переключит на следующий продукт.

Если бы мы не смогли поместить все контролы в карточку, то надпись «17 из 23» сделали бы элементом регулировки, который листает карточки. На домашнем экране айфона у точек такое же поведение.



Пример

Корзина



Корзина — довольно сложный экран, где важно правильно подытожить промежуточный результат. При этом на нем же можно влиять на позиции: добавлять продукты, выбирать соус, ввести промокод и т.п.

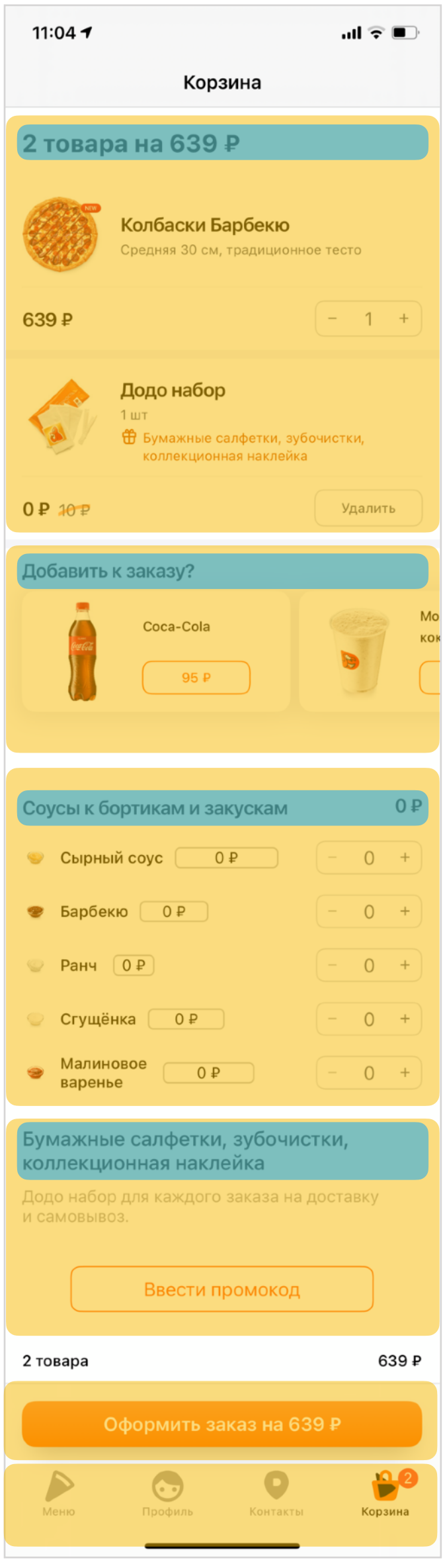
В таком длинном списке важно уменьшить количество элементов, но сохранить у них всю ключевую информацию: подарки, ошибки, акции и т.п.

Для начала надо объединить все контролы внутри ячеек продуктов. Подробно мы разбирали это в главе про контекстные действия.

Затем нужно превратить горизонтальную карусель «добавить к заказу» в один контрол, как мы это сделали в главе «Карусель».

Для редактирования соуса так же хорошо подходят элементы регулировки, 5 элементов одного соуса превращаются в один регулируемый.

Объединять ли заголовок и описание акции вопрос открытый, это сильно зависит от длины описаний. Если оно большое, да и сам заголовок длинный, то лучше разделить. Если описания короткие, то можно унести их в value. А вот два лейбла про итоговую сумму нужно соединить в одну строчку. Или вообще перенести это описание в value кнопки оформления.



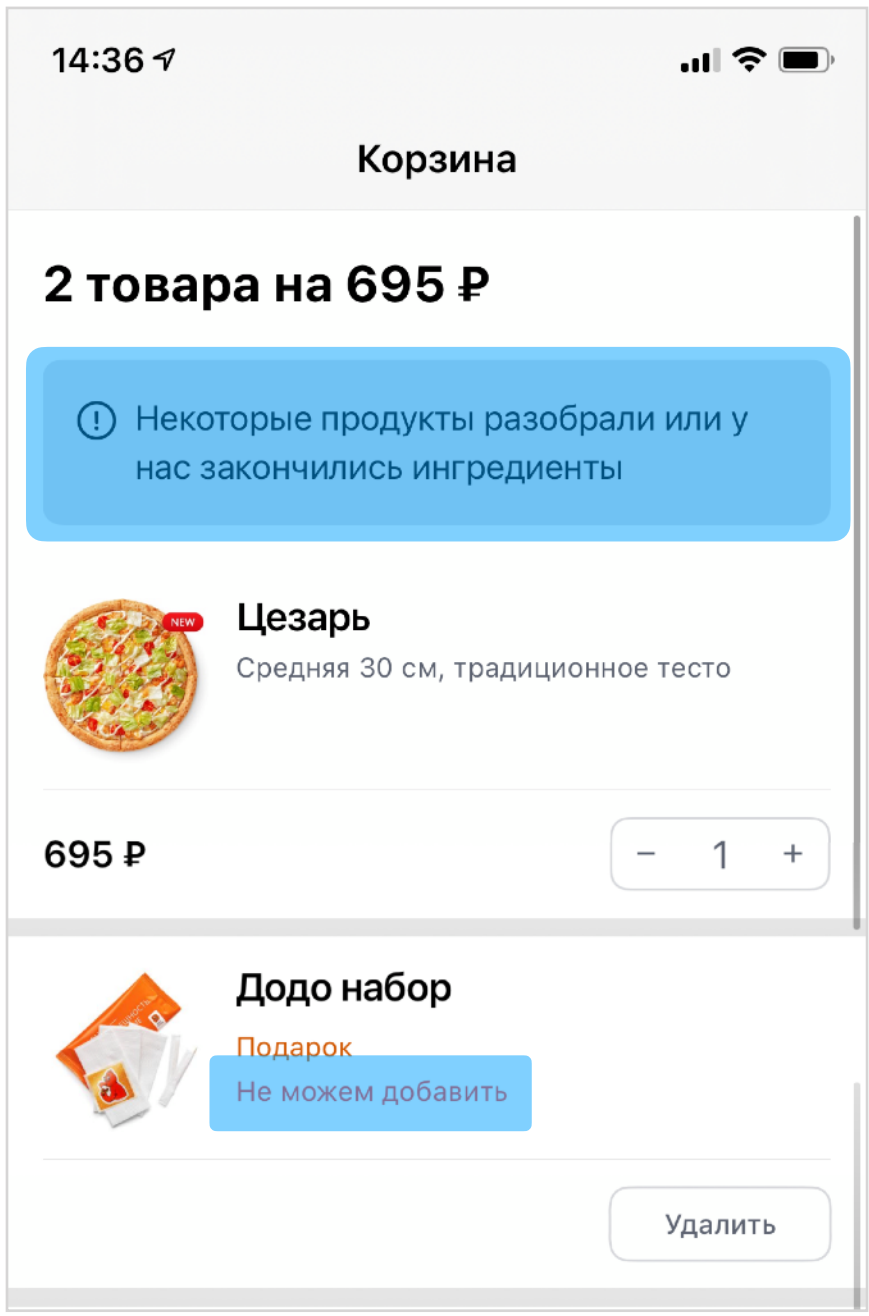
Навигация

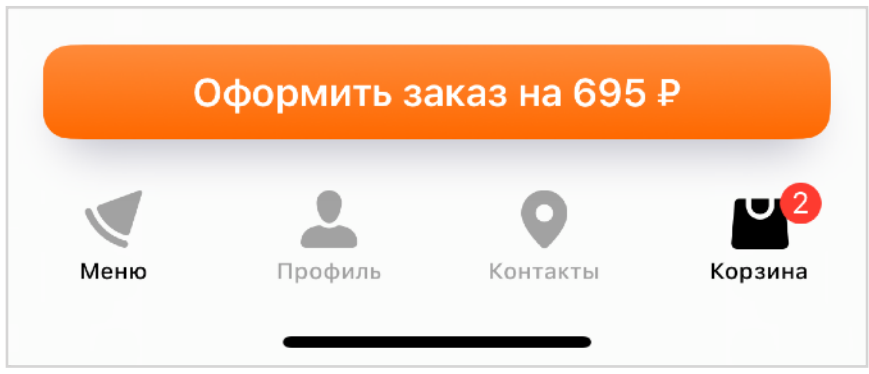
Следующим шагом можно поправить навигацию на экране: поставить заголовки, контейнеры и обработать ошибки.

На экране много заголовков, им нужно добавить трейт `.header`. Это не даст переключаться по ним через ротор, но они помогут понять структуру страницы.

На экране много смысловых контейнеров: продукты, предложения для дозаказа, соусы, акции, итоги, нижняя панель навигации. Если все сверстано отдельными ячейками, то перемещаться по контейнерам в динамичном списке не получится, VoiceOver так не умеет. А вот если экран сверстать в более статичной манере, когда элементы за пределами экрана не исчезают, то навигация по контейнерам заработает.

Для оранжевой кнопки заказа разумно добавить Magic Tap и рассказать о нем в подсказке к этой кнопке.





Ошибки

Мы исправили описания отдельных контролов и добавили базовую навигацию на страницу, теперь можно заняться обработкой случаев посложнее.

В корзине могут возникать ошибки: товар кончился, акция не сработала, промокод не применился и т.д. Скорее всего, ошибка не позволит перейти к следующему экрану.

Статус ошибки должен быть у каждой ячейки, при этом слово «ошибка» должно быть первым в описании, чтобы сразу обращать на себя внимание.

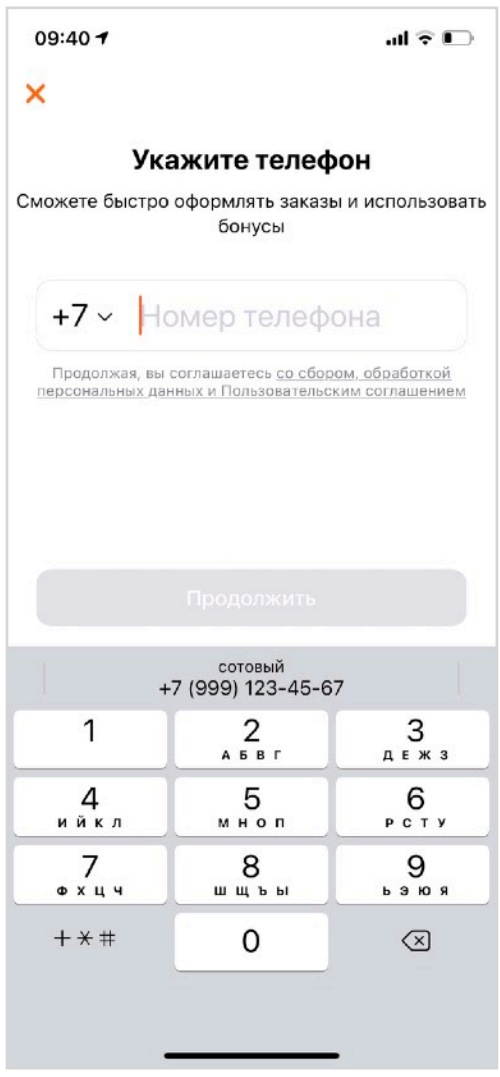
Обо всех ошибках на экране можно рассказывать в value кнопки оформления:

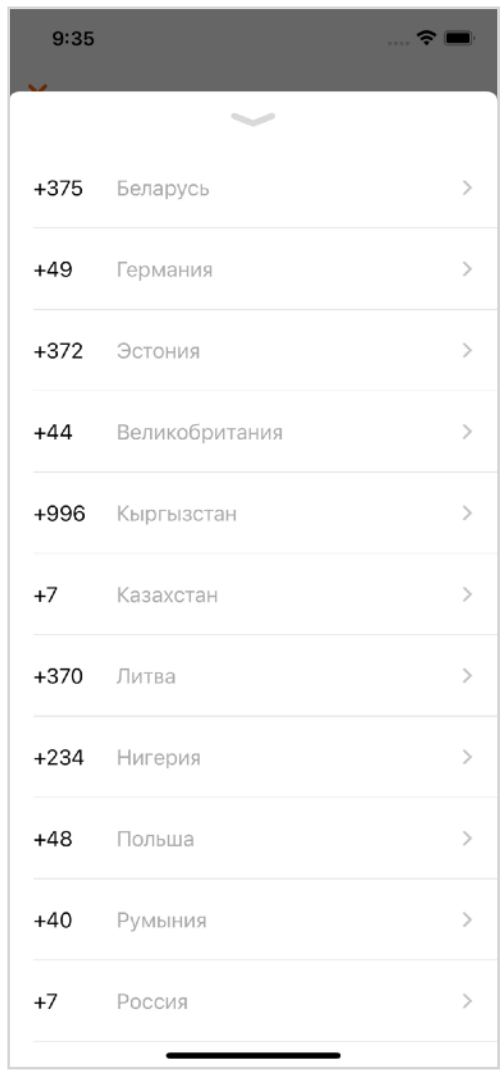
Оформить заказ на 695 рублей
1 ошибка, Не можем добавить Додо набор,
Кнопка

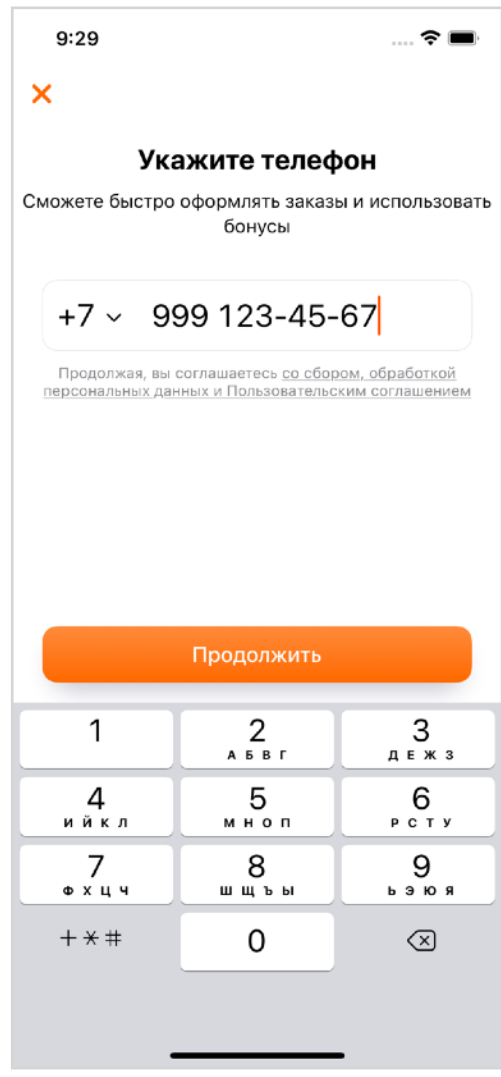
Если добавить ротор, который может переключаться по объектам с ошибками, то пользоваться станет еще проще.

Пример

Регистрация





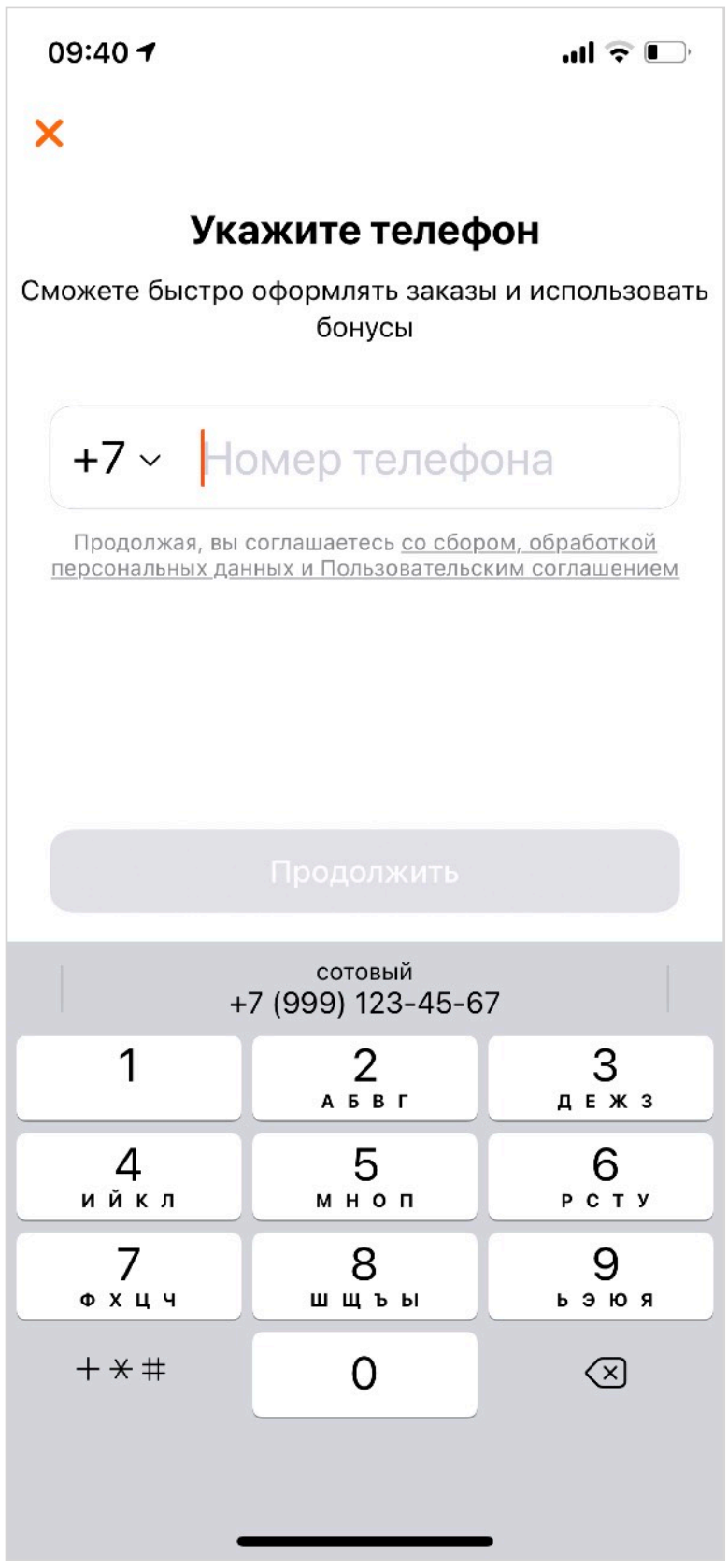


Экран регистрации лишь кажется простым, а на самом деле у него и сложная валидация, и отдельная кнопка по выбору кода страны, и подсказки для ввода от операционной системы.

Для удобства зрячих экраны агрессивно перехватывают поведение и это проблема.

— Когда открывается экран телефона — ставим фокус в поле ввода, появляется клавиатура. Для незрячего фокус тоже смещается, но он не может так легко прочитать контекст всего экрана.
— Когда вводим номер телефона — автоматически перемещаемся на следующий экран. Так можно пропустить кнопку пользовательского соглашения.

Для упрощения ввода есть предиктивный ввод, но iOS никак не сообщает о нем для незрячих, хотя так телефон и смс можно ввести намного быстрее.



Для незрячего удобно, что фокус VoiceOver может перемещаться без прекращения ввода текста, поэтому, несмотря на перехватывание поведения, он может исследовать весь экран.

Для начала надо исправить очевидные проблемы:

— Подписать кнопку закрытия.

— Поставить трейт заголовок.

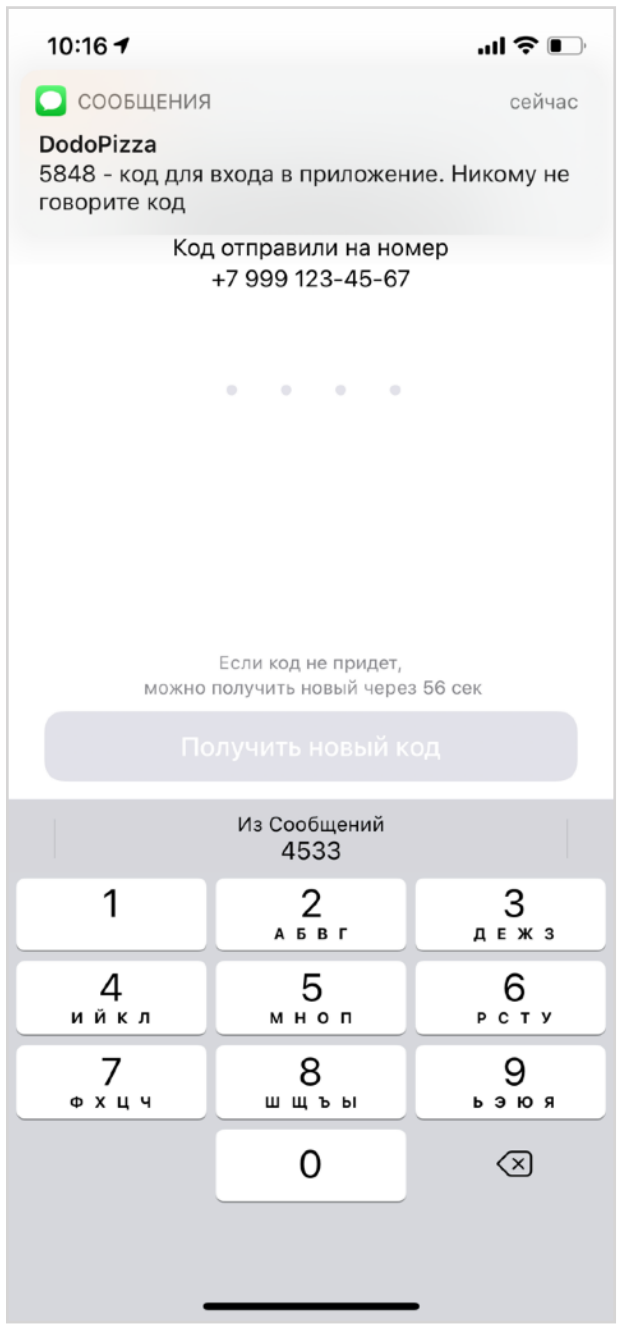
— Добавить трейт `.notEnabled`, когда кнопка недоступна.

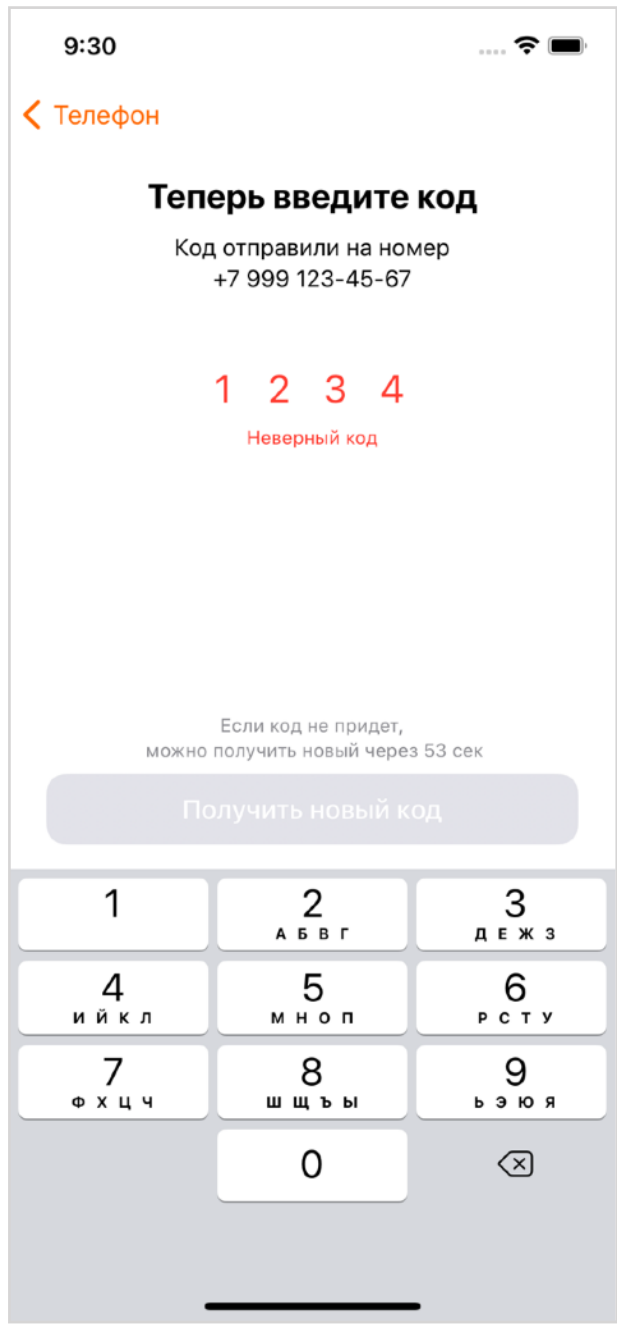
С вводом телефона сложнее: мы разделили код страны и основную часть телефона, об этом нужно рассказать. Я сделал так:

— Телефонный код страны, +7, кнопка
— Номер телефона без кода страны, текстовое поле.

Мы учитываем, что телефон могут вставить из буфера обмена или использовать предложенный телефон вместе с кодом, в таких случаях мы сами удаляем код страны. Такое поведение надо проверить на случай, когда телефон вводят голосом и в тексте сработала автозамена «мой телефон» на «+7 999 123-45-67».

Пользовательское соглашение достаточно сделать ссылкой с помощью трейта `.link`.





Смотрите, что происходит, когда мы открываем экран смс:

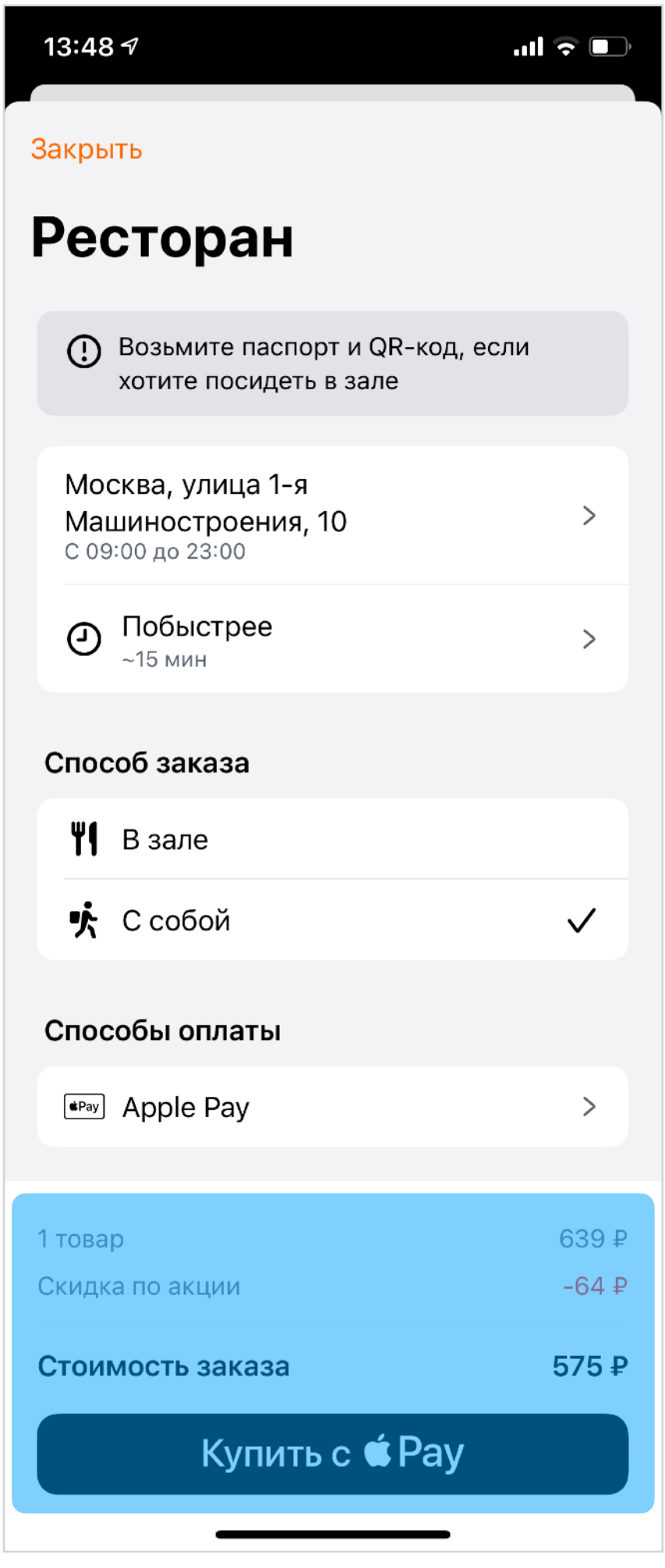
— фокус оказывается на текстовом поле,
— приходит пуш с кодом, VoiceOver читает его текст,
— над клавиатурой появляется кнопка с кодом.

Такое мельтешение может сбить с толку незрячего пользователя, но спустя несколько секунд состояние экрана стабилизируется и дальше человек будет работать с адаптированной версией экрана. Для правильной адаптации нужно:

— поставить заголовок,
— подписать поле ввода, если ввели неправильно, то сообщать о результате валидации через оповещение,
— тексту со временем добавить трейт `.updatesFrequently`. Если минута прошла, кнопка стала доступна и смс можно отправить еще раз, то сообщать об этом через оповещение.

Пример

Оформление заказа



Экран оформления заказа можно считать центром управления: он подытоживает разные настройки, которые сохранились с предыдущего заказа, но дает возможность их поменять.

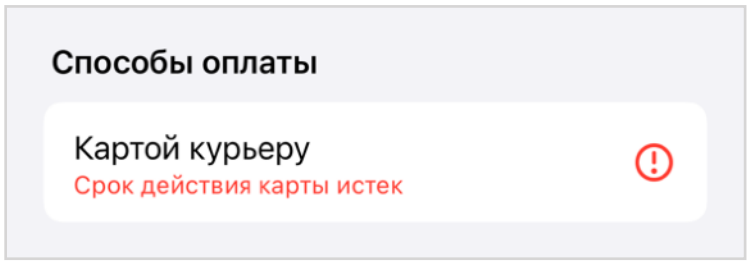
Базовая адаптация стандартная: ячейки укрупнить, в них явно отделить `label` от `value`. Не забыть про трейт `.button` и `.selected` для ячеек с галочками. Если вы используете нативные таблицы, то, скорее всего, эту работу за вас уже сделала iOS.

Для заголовков секций добавить трейт `.header`.

Нижняя панель с итогами не скролится, она должна быть в отдельном контейнере от списка. Подытоживающие строки про стоимость заказа укрупняются в пары название-значение.

Обработка ошибок

Для экрана оплаты важно уметь понятно отображать ошибки и помогать их исправить. Для этого на экране есть 3 вида подсказок на разные случаи.

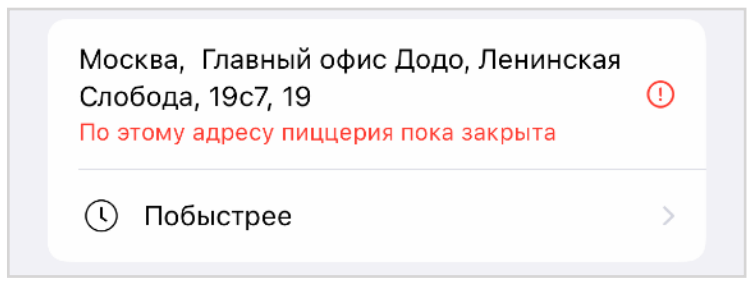


Данные не подходят

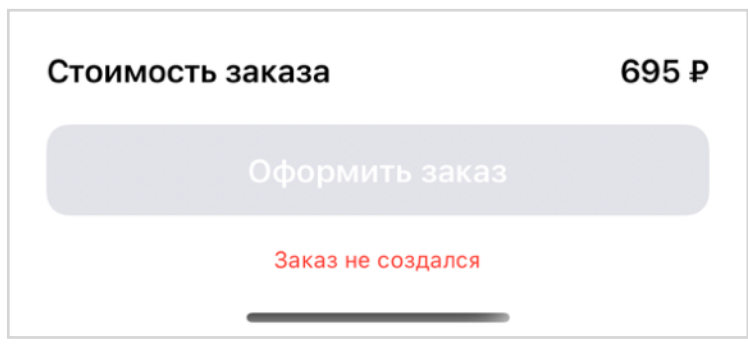
Если текущее значение не подходит, то ошибка появится прямо в поле.

Озвучить надо так:

> Ошибка, срок действия карты истек.
> Оплата картой курьеру.



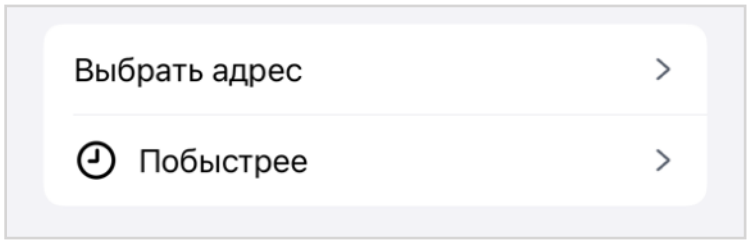
Работа с ошибками сделана системно, поэтому по ним можно перейти ротором и о них можно прочитать в `customContent` кнопки оформления заказа.

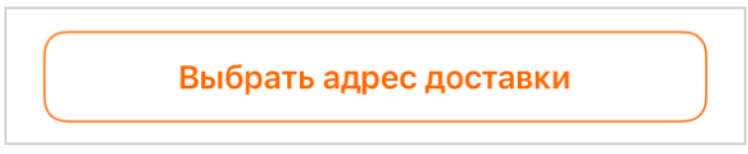


Неизвестная ошибка

Если ошибку не удалось привязать к какому-то полю, то показываем ее под кнопкой оформления заказа. При этом, текст ошибки записан в value кнопки оформления:

> Оформить заказ,
> Заказ не создался,
> недоступно, кнопка.





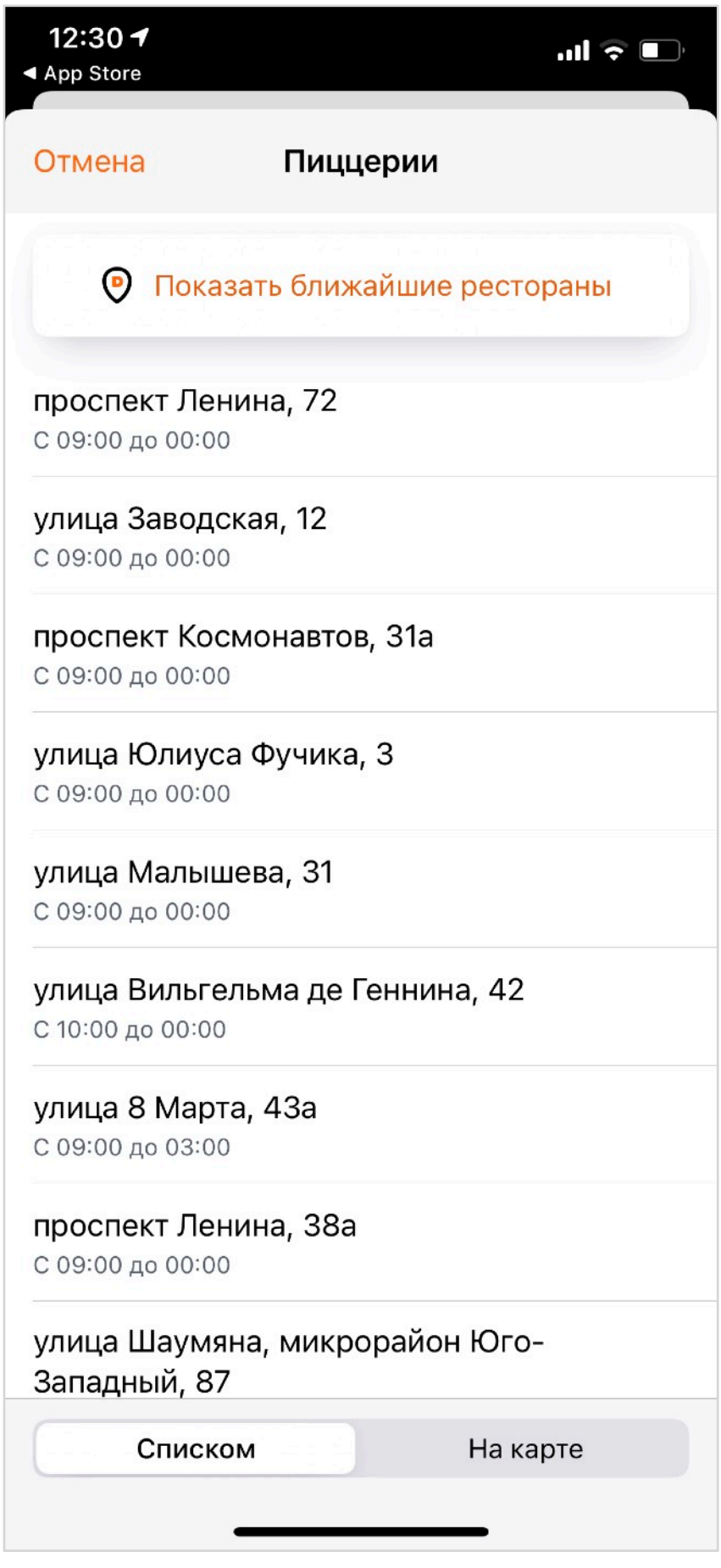
Нет данных

Если какое-то из полей не заполнено, то главная кнопка предложит ввести данные.

В итоге обработка ошибок хорошо работает как для зрячих пользователей, так и для незрячих, VoiceOver лишь правильно озвучивает текст.

Пример

Список пиццерий



Список пиццерий очень похож на список городов, разве что в ячейках явно видно разделение данных на `label` и `value`.

В Москве длинный список пиццерий, поэтому важно после скрола сообщать, с какой по какую начальную букву видны адреса.

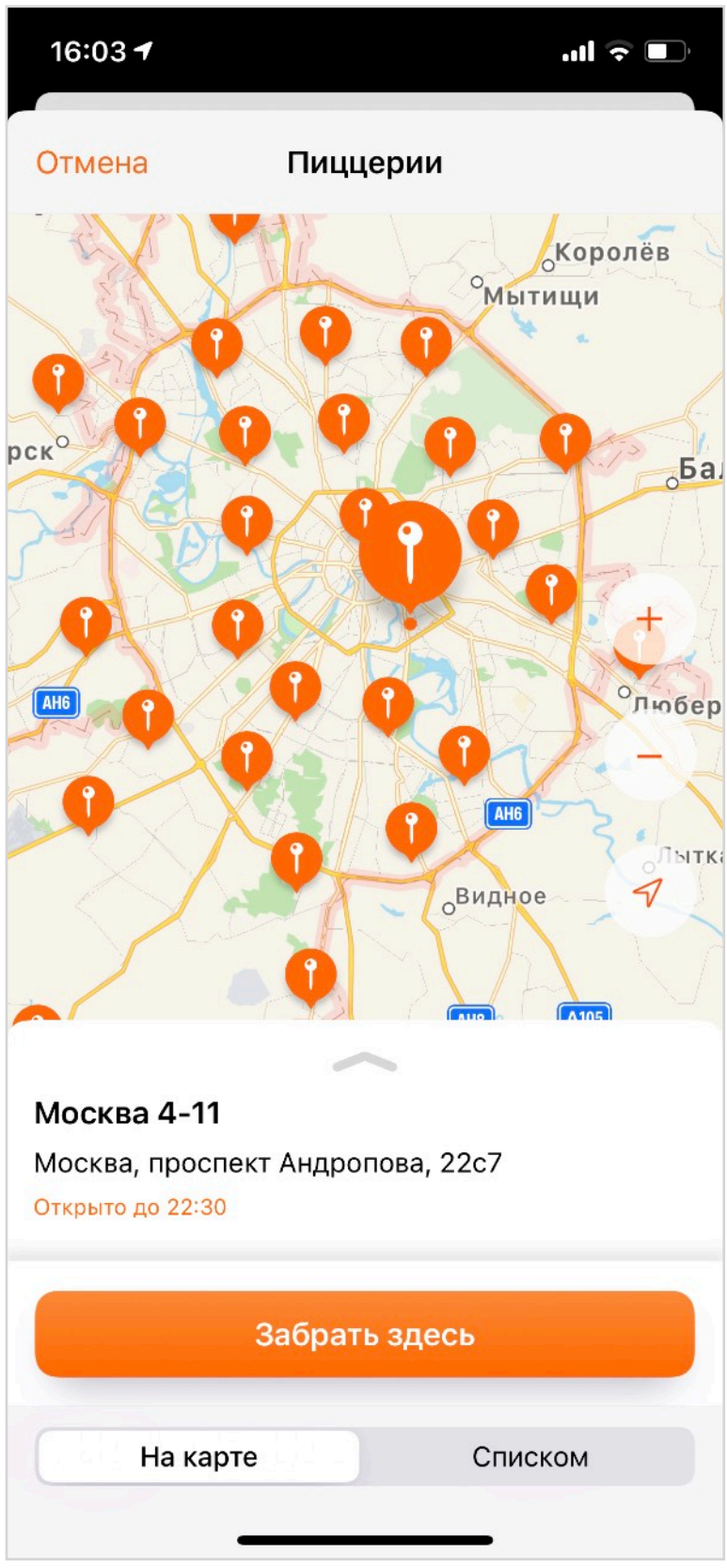
Кнопка «показать ближайшие пиццерии» находится вверху экрана, значит и незрячий наткнется на нее сразу, это хорошо. А вот кнопка переключения типа заказа в самом низу и про нее он может даже не узнать.

Нажав на кнопку, мы отсортируем список, кнопка исчезнет и фокус встанет на первую ячейку, а это и будет ближайшая пиццерия.

После смены типа отображения со списка на карту нужно вызвать оповещение `.screenChanged`.

Пример

Карта



Карта кажется сложной для адаптации, потому что на ней много элементов. Но если подойти к адаптации как к сценарию, то все станет понятней. Сначала мы настраиваем масштаб:

1. Карта должна рассказать о своем масштабе, после зума сообщать о количестве интересных мест в каком-то радиусе.

2. Кнопками + и - можно менять этот радиус, сообщать о количестве пиццерий рядом, есть ли среди них закрытые.

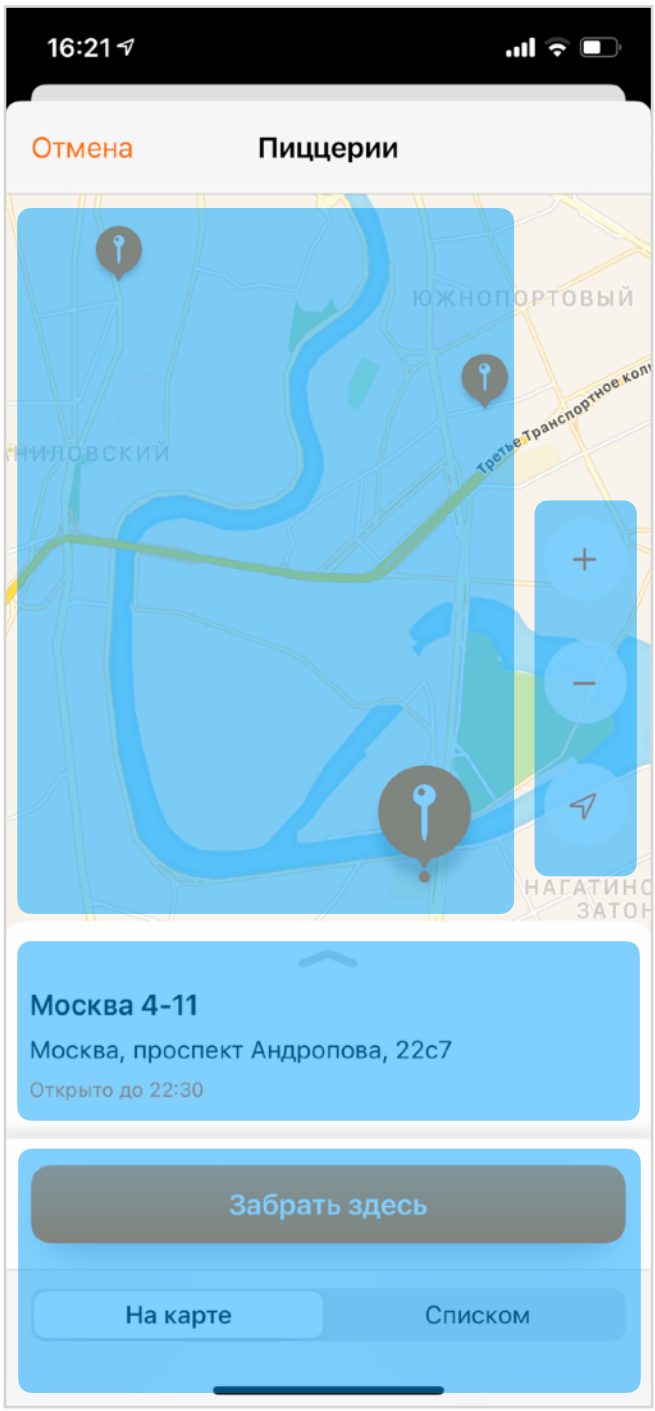
3. Можно не зумить самостоятельно, а определить текущее местоположение.

Если масштаб устраивает, то мы переключимся на конкретные точки, пройдемся фокусом по ним. На этом уровне надо знать адрес и открыта ли пиццерия прямо сейчас.

1. Когда мы выбрали одну из точек, важно узнать адрес, время работы, телефон и ближайшие места-ориентиры. Если пиццерия находится в ТЦ, то важно узнать, как найти вход внутри помещения. Панель с информацией должна нормально обрабатывать модальность.

2. Мне может понадобиться маршрут до пиццерии, но это уже задача экрана, который будет после оформления заказа.

Весь этот сценарий можно выполнить и со списком адресов, но ситуации бывают разные, любой интерфейс должен быть доступным.



Каждый этап нашего сценария это работа с одним из контейнеров:

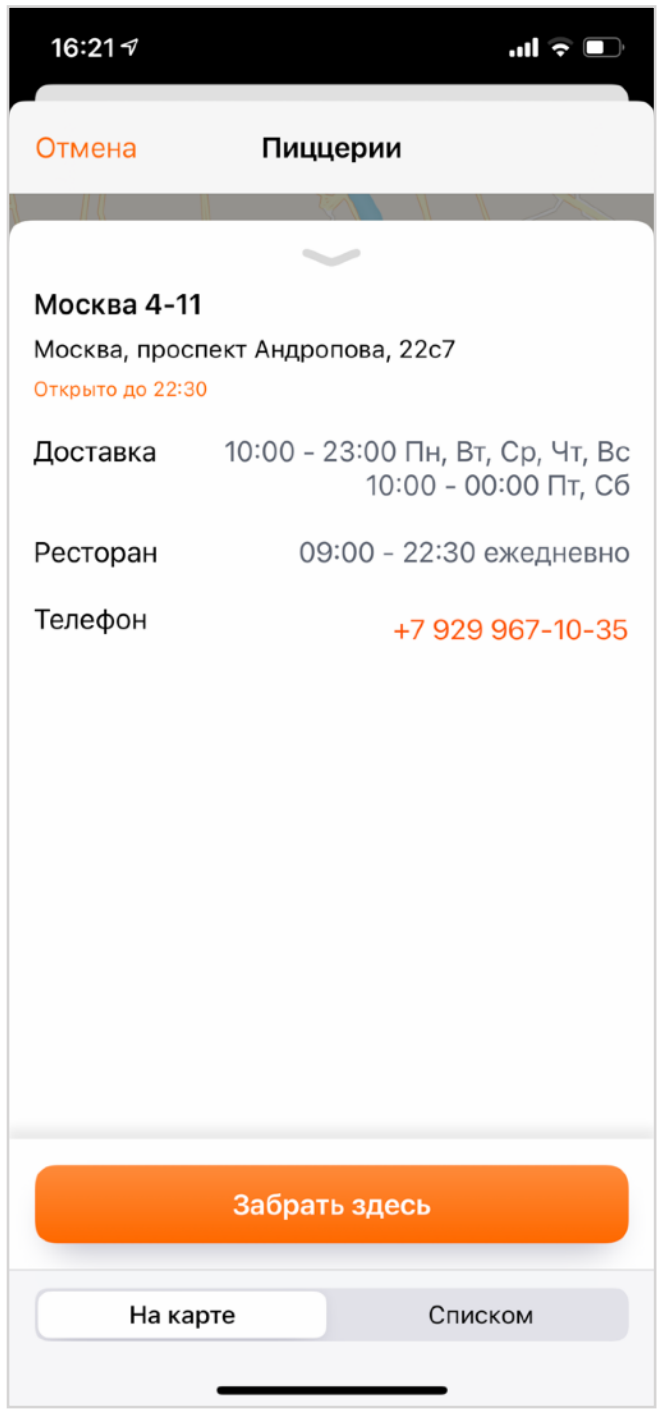
— кнопки масштаба,
— карта с маркерами,
— описание места,
— выбор и переключение в список.

После выбора правильного масштаба маркеров на карте остается мало, уже понятно как работать с ними с помощью фокуса.

Навигацию можно упростить с помощью ротора, например, переключаться вертикальными свайпами только по доступным точкам.

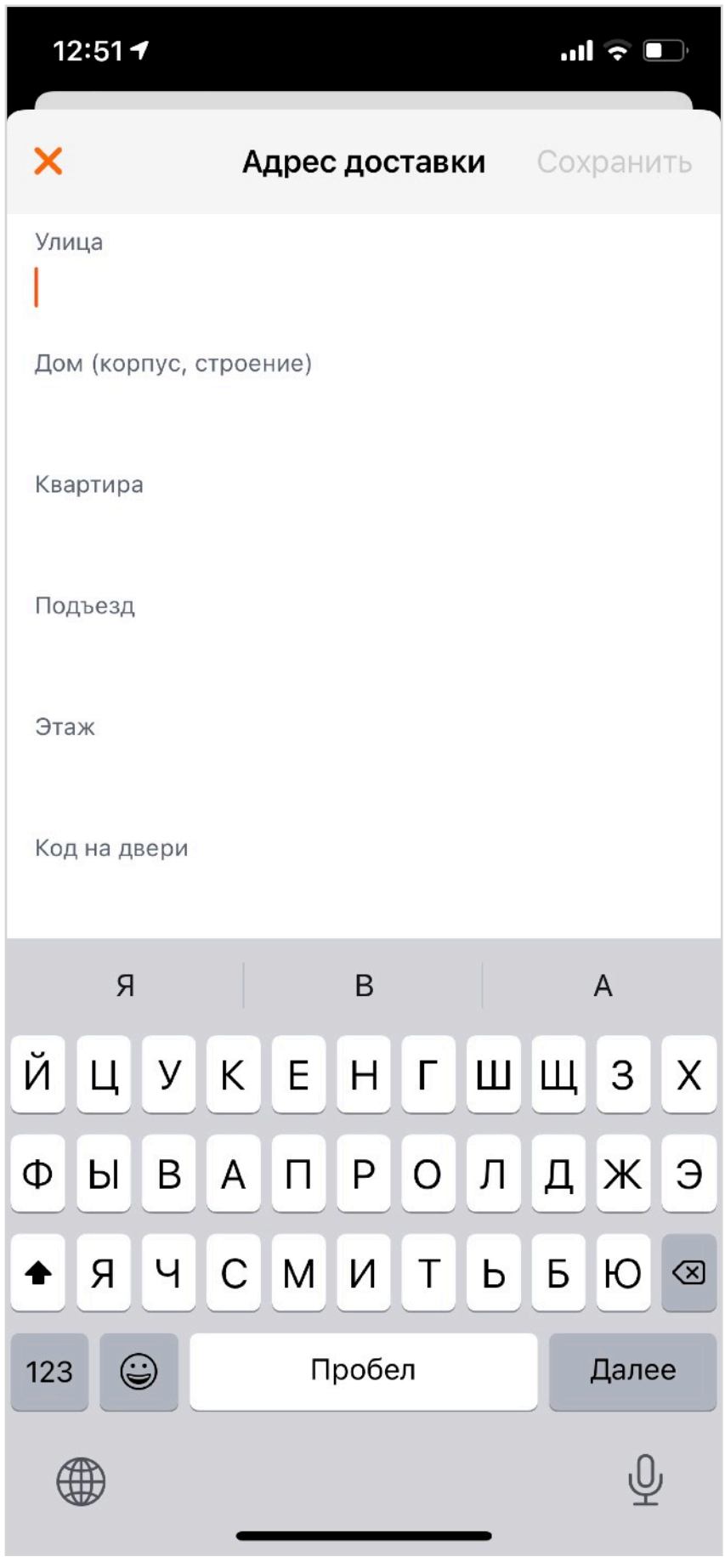
Если мы выбрали точку, то фокус надо переставить на карточку с описанием. Внутри карточки адаптация простая, нужно лишь объединить названия слева со значениями справа.

При закрытии карточки важно возвращать на выбранный пин, обычно он в центре экрана. Скраб должен сначала закрывать карточку описания, а если она закрыта, то окно выбора пиццерий.



Пример

Адрес



Адрес — сложный экран, потому что в нем надо ввести много данных самому, а это может приводить к ошибкам.

Сначала надо уменьшить количество элементов, подписи должны перейти в `label` текстовых полей.

Упростить ввод поможет правильная работа кнопки «далее» на клавиатуре.

Сохранение адреса

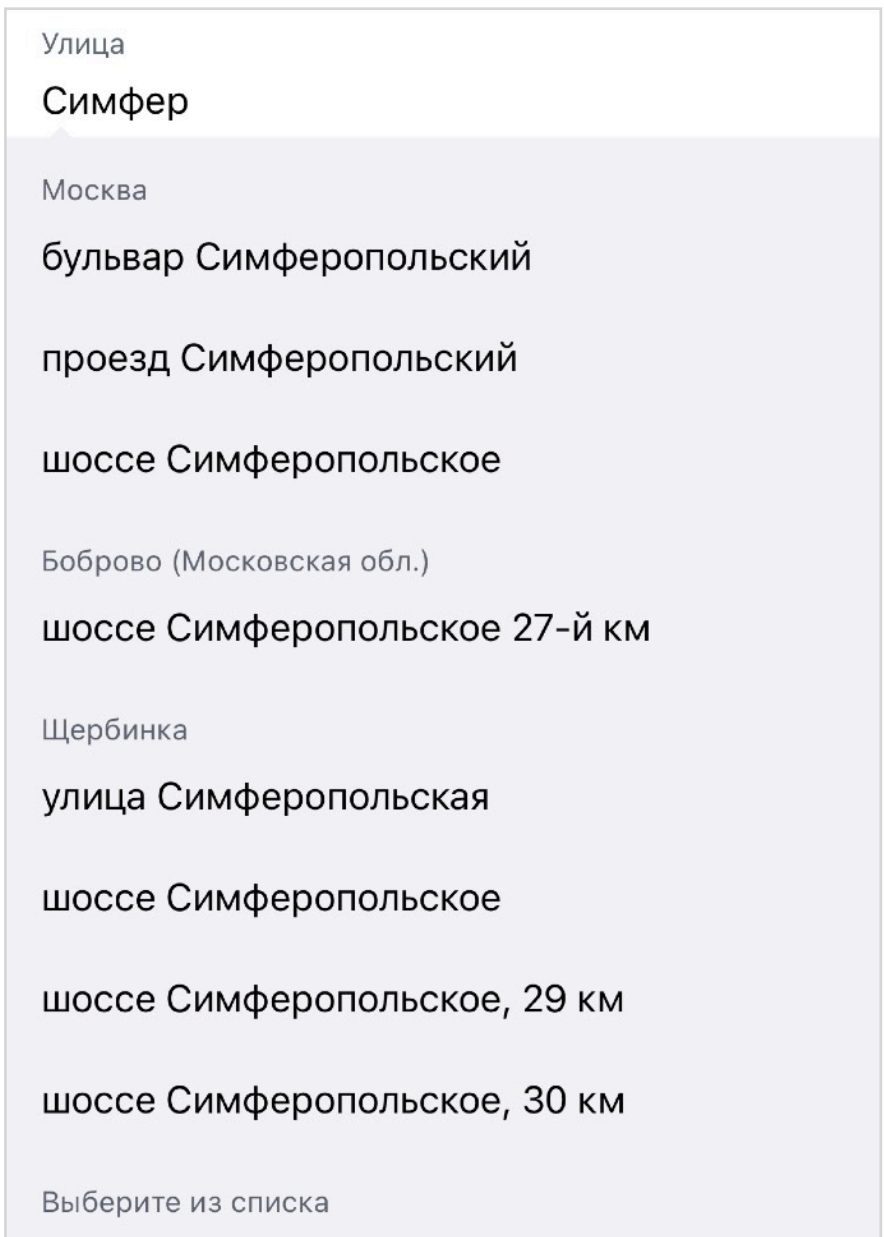
На кнопке «сохранить» нужно прочитать весь введенный адрес или сказать в каких полях еще нужные данные, чтобы сохранение стало доступно.

У нас обязательно лишь поле улицы, после его ввода кнопка «сохранить» станет доступна, об этом нужно сообщить через оповещение `.layoutChanged`. Фокус на кнопку ставить не нужно, сообщите текстом: «теперь адрес можно сохранить».

Меджик тап для кнопки сохранения добавить не получится, потому что фокус всегда на одном из текстовых полей, а они будут перехватывать его и включать голосовой ввод текста.

Пример

Обязательный выбор адреса



После ввода части адреса появляется выбор из нескольких вариантов адресов.

Через `hint` нужно заранее рассказать о том, что надо будет выбирать один из предложенных адресов.

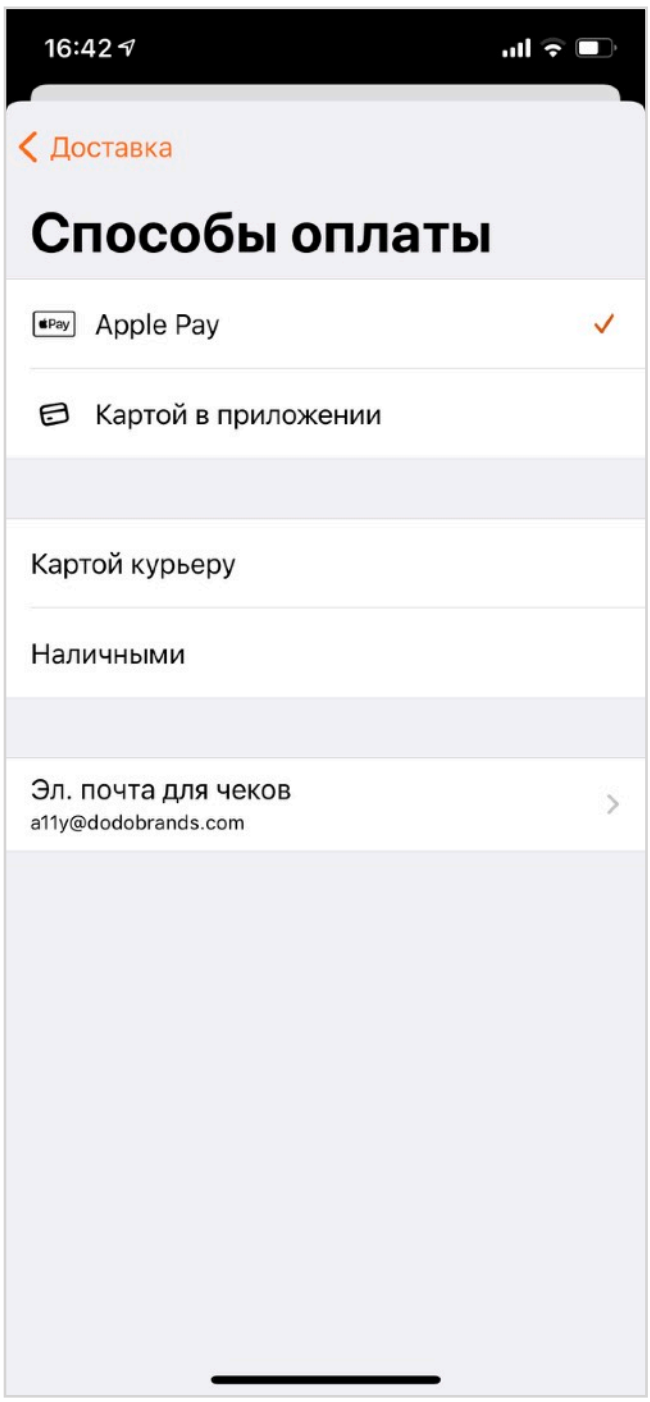
О появлении подсказок надо коротко сообщить: системного звука для этого нет, но достаточно произнести короткое «5 адресов» или «2 адреса».

Названия отдельных городов надо выделить с помощью трейта `.header`: графически заголовок даже меньше надписи, но главная его роль — структурировать, поэтому он и нужен.

В самом низу есть надпись «выберите из списка». Там она не нужна и ее можно скрыть, а вот весь список с вариантами стоит сделать контейнером и дать ему название «варианты адресов».

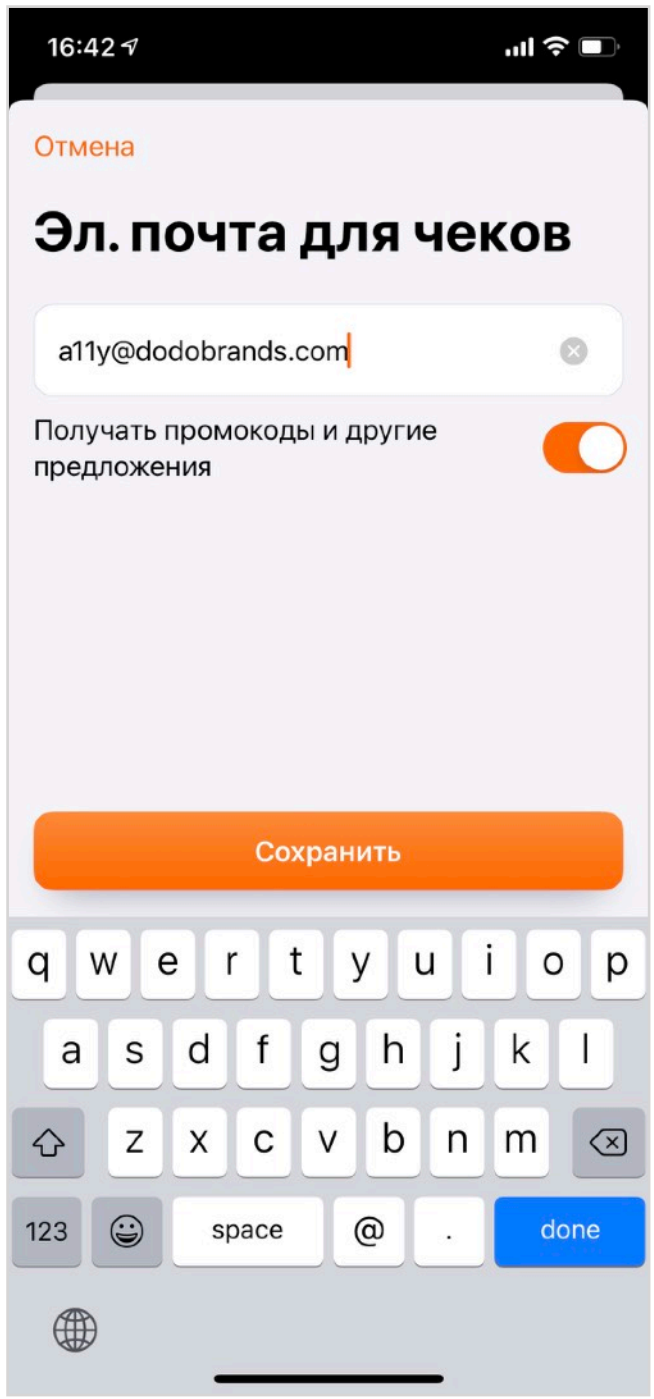
Пример

Способ оплаты



Способ оплаты это простой экран, но надо не забыть про трейт `.selected` и сообщить, что каждая ячейка — это кнопка.

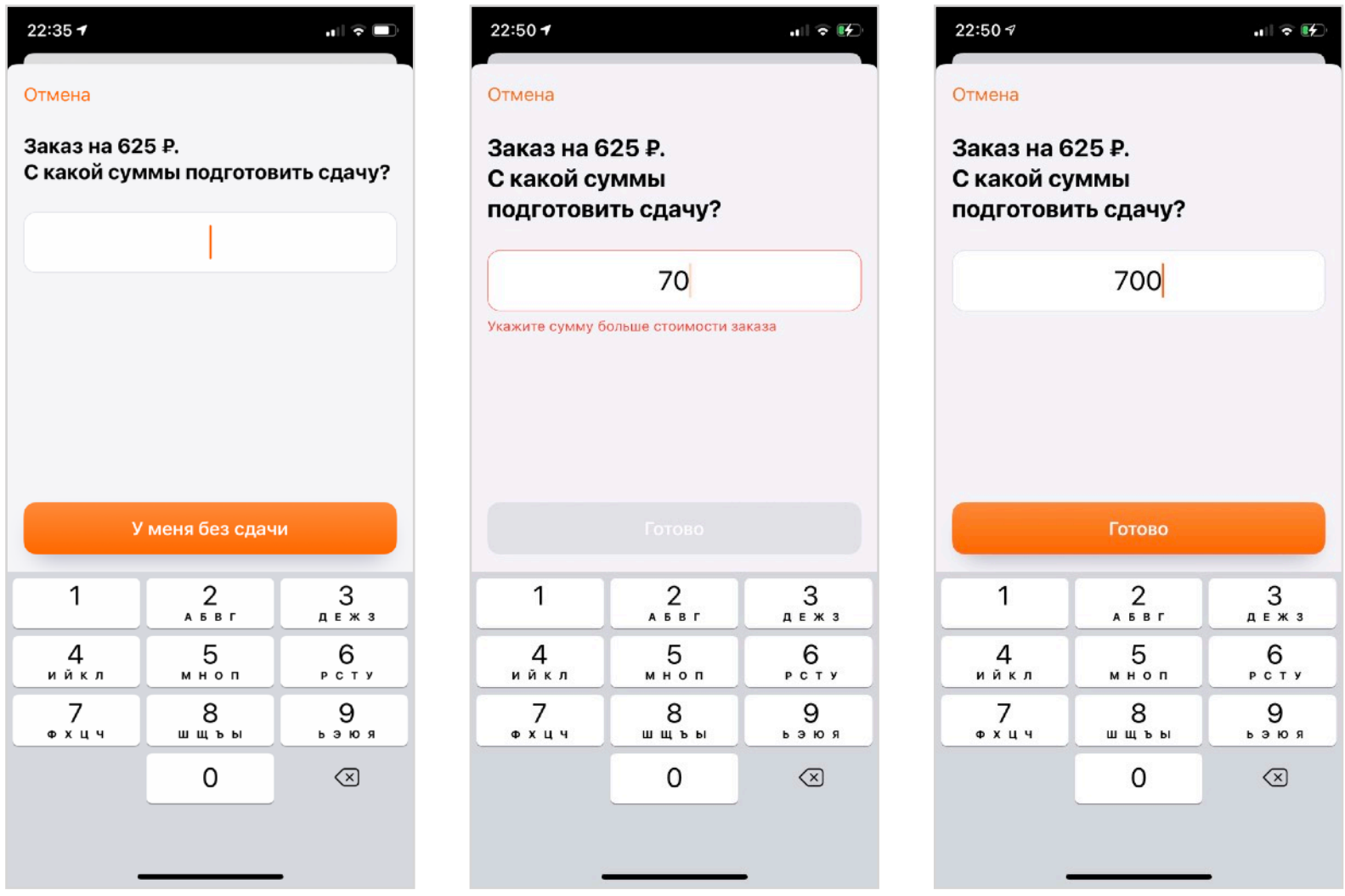
Интересно, что экран сгруппирован, но заголовков у групп нет. Скорее всего, дизайнер не смог подобрать названия, хотя по смыслу группы разделены на оплату «сейчас» и «при получении».



Ввод почты для чеков достаточно простой: переключатель для рекламы надо объединить с подписью, а после ввода валидной почты сообщать о доступности кнопки через оповещение.

Пример

Оплата наличными



У экрана ввода наличных три состояния экрана:

— ничего не ввели, можно оплатить без сдачи;

— ввели что-то, но принять это не можем, есть ошибка;

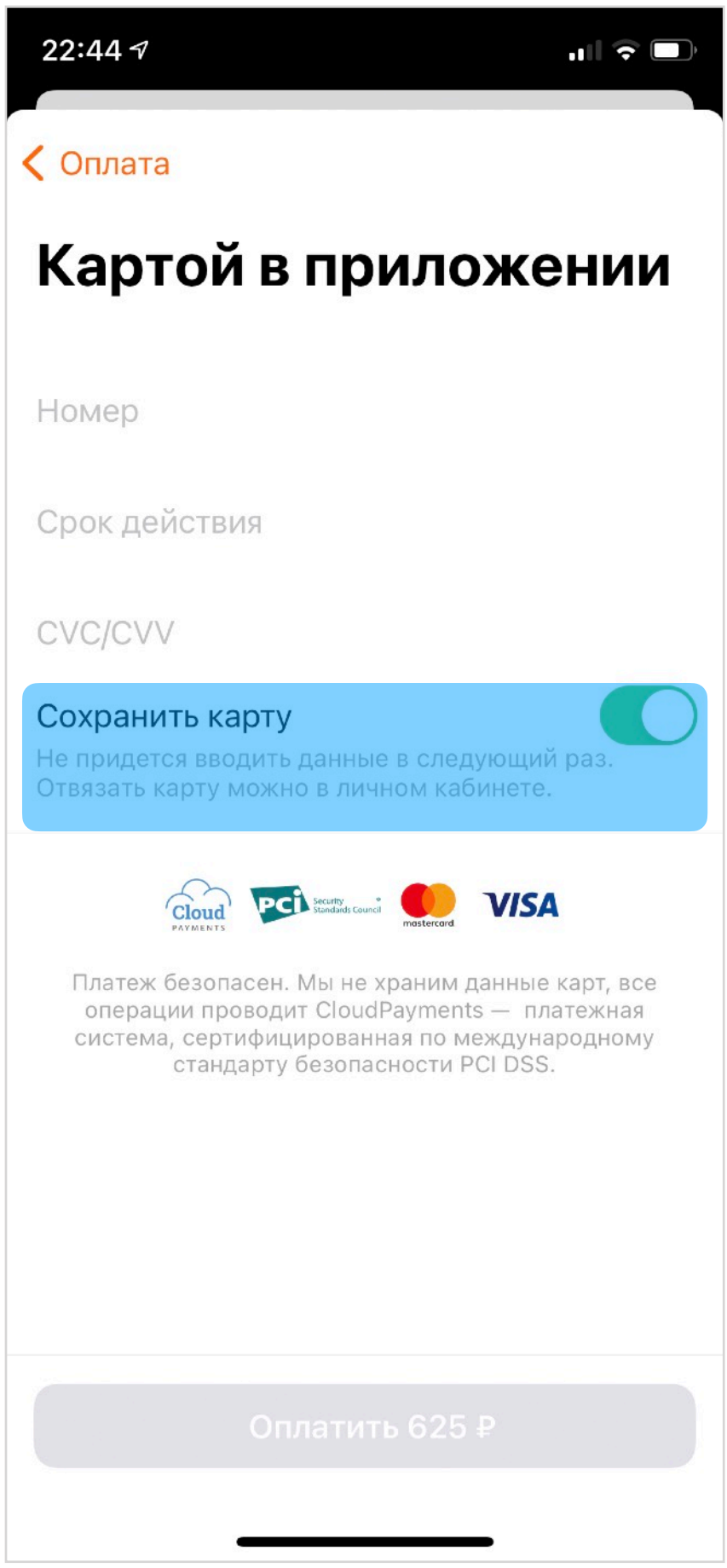
— число подходит, с этой суммы нужна сдача.

Во время ввода читать ошибку не нужно, только если ввод явно закончили и перешли фокусом к следующему элементу, это будет кнопка «готово». В таком состоянии кнопка совсем не готова и хорошо бы в ее `value` рассказать, почему кнопка недоступна. Получается, что текст ошибки лучше скрывать от VoiceOver, но перенести его в `value` кнопки.

После валидации правильной суммы нужно сообщить, что кнопка «Готово» стала доступна.

Пример

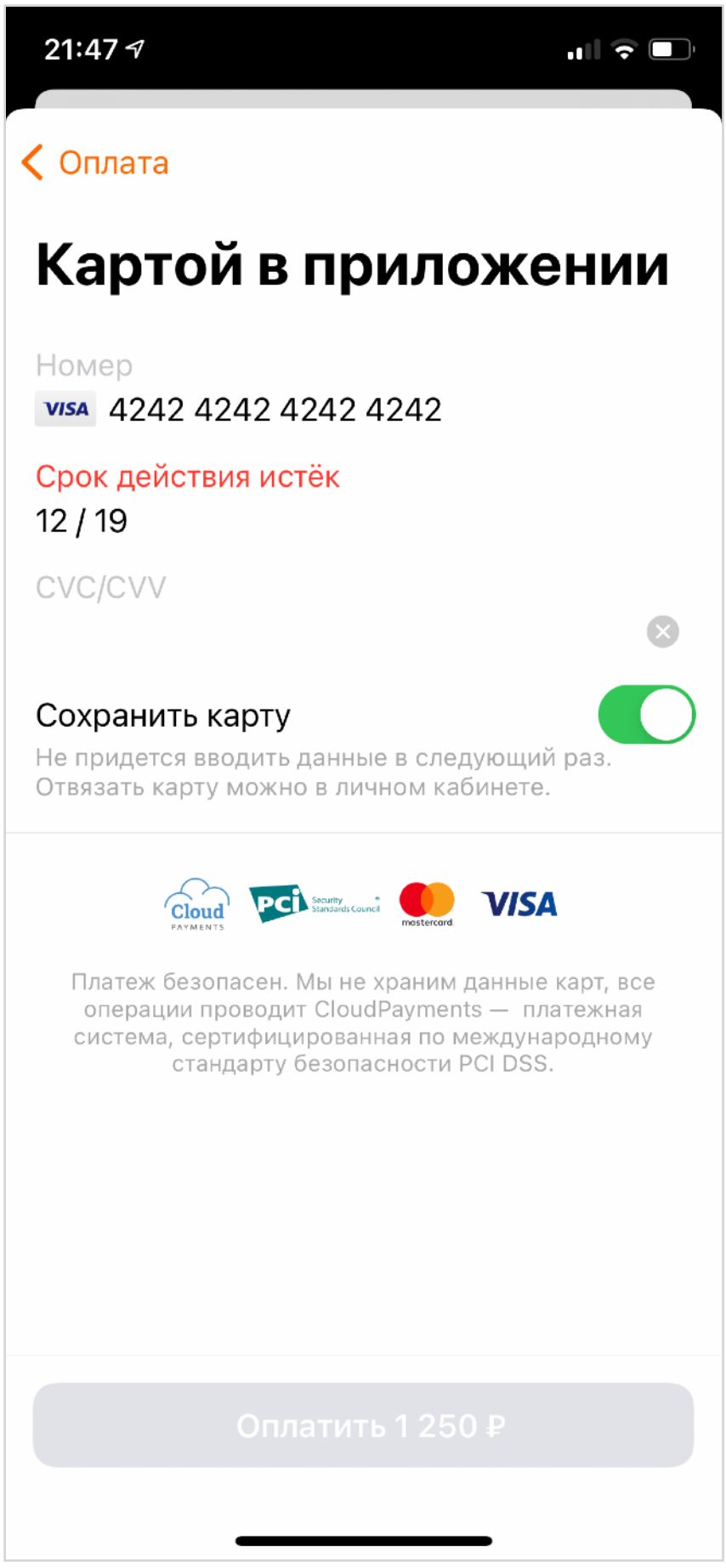
Оплата картой



Ввод данных с карты хорошо валидируется, после ввода фокус сам перемещается на новое поле.

Базовая адаптация обычная: текстовым полям даем название, а логотипы превращаем в надписи.

Группу элементов «сохранить карту» нужно объединить в один элемент, по активации всего элемента должен переключаться свитчер. О включенной опции сохранении карты стоит сообщать в кнопке «оплатить».



Если данные введены некорректно и пользователь сам переключил ввод на следующее поле, то об ошибке нужно сообщать через оповещение.

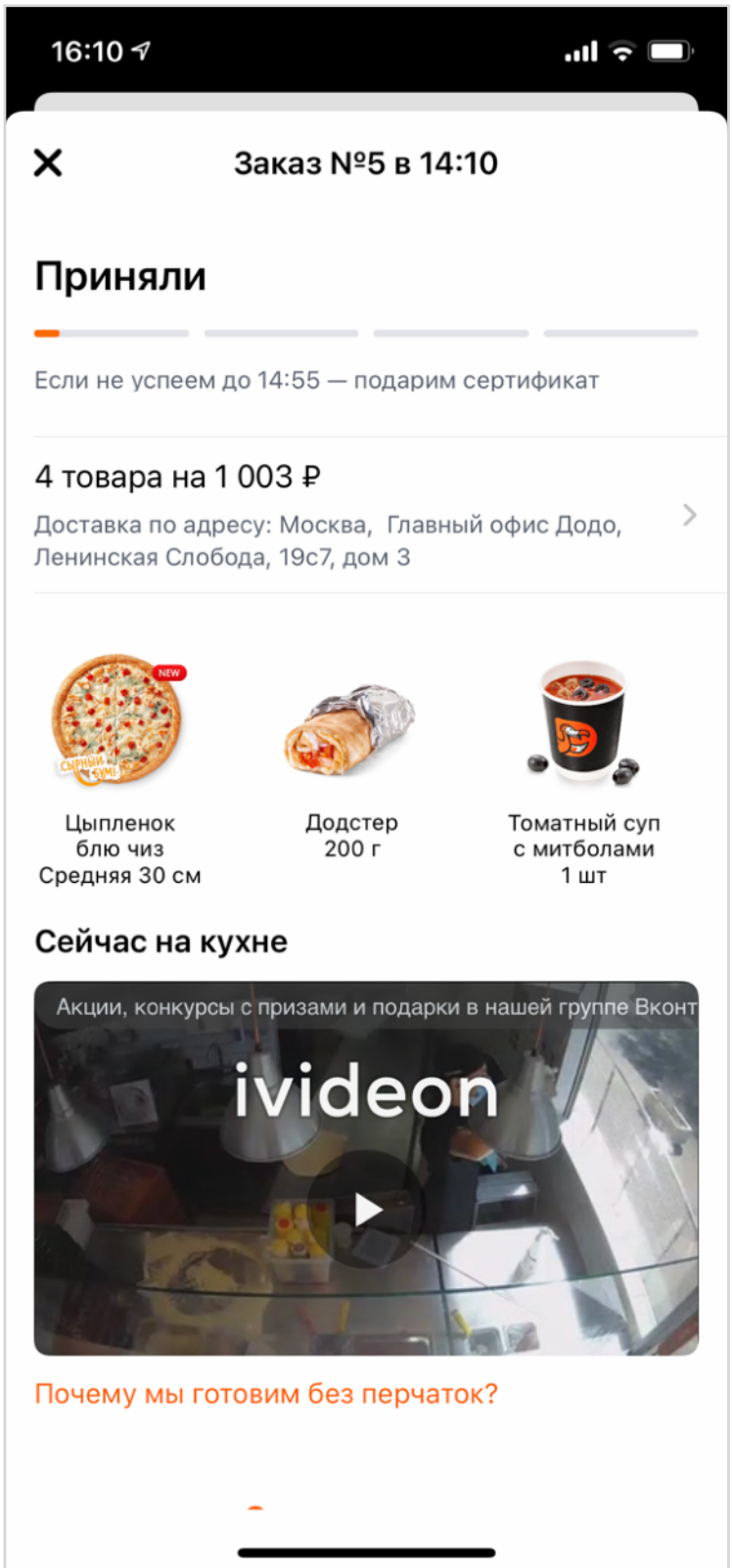
Текст ошибки нужно добавлять к описанию элемента, лучше перед описанием контрола, чтобы проще было находить элемент с ошибкой среди всех. У нас получилось полностью заменить названия текстового поля на описание ошибки, вроде бы это неплохо. В `value` текст ошибки добавлять нельзя, потому что `value` для UITextField — это редактируемый текст.

Получается такое описание:

Ошибка, срок действия истек. Дата. 12/19.

Пример

Ожидание заказа



После заказа пользователь попадает на экран состояния заказа.

В первую очередь, на экране есть время, которое обновляется. Поставив трейт `.updatesFrequently` мы подскажем VoiceOver, что нужно читать value подписи каждые несколько секунд.

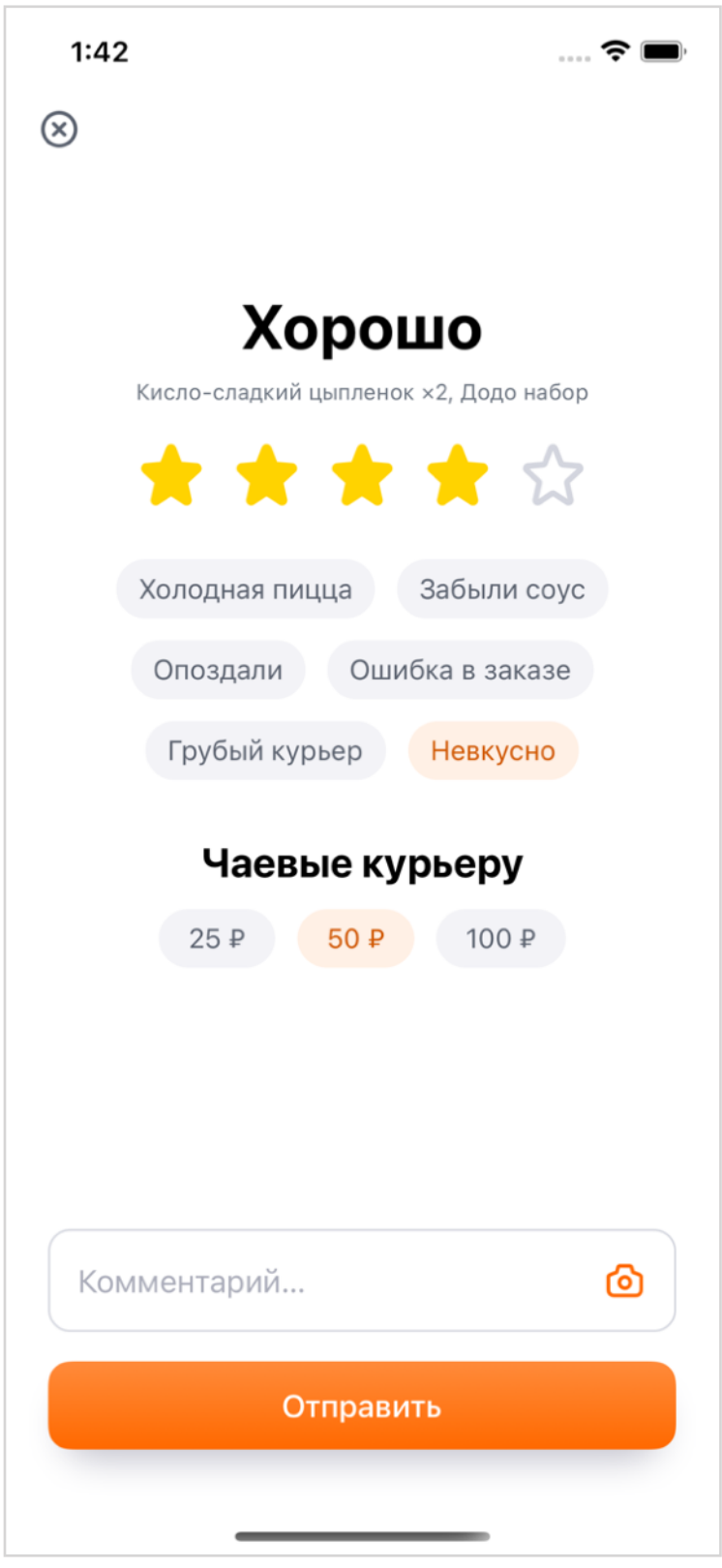
Доставка в среднем около 30 минут, за это время многие открывают экран несколько раз и проверяют оставшееся время. С прошлого запуска фокус может переместиться на любой контрол, при открытии приложения надо вернуть его на таймер. Такое поведение даст трейт `.summaryElement`, на таймере.

«Почему мы готовим без перчаток?» нужно отметить трейтом `.link`, потому что она откроет браузер.

У заказа есть несколько статусов, содержимое экрана отличается от статуса: во время приготовления видно камеру с кухни, а после получения — оценку заказа. Смену статуса поддерживаем через оповещение `.screenChanged`.

Пример

Оценка заказа



Оценка заказа содержит полный набор различных кнопок и состояний.

Что у нас есть:

— описание заказа,

— оценка в виде звезд и надписи «хорошо»,

— стандартные проблемы в заказе, выбрать можно несколько,

— чаевые, выбрать можно только один вариант,

— комментарий и возможность приложить фото,

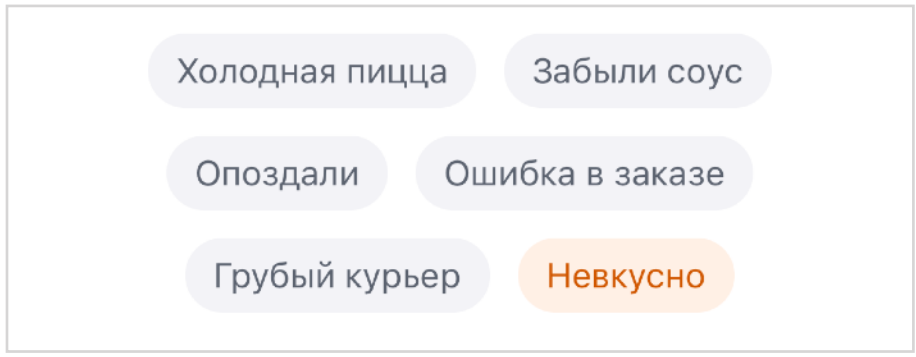
— кнопка отправки.

Оценка в звездах и чаевые могут иметь только одно значение из нескольких, для этого хорошо подходит `.adjustable` трейт: так не придется проходить фокусом по всем звездочкам и чаевым, навигация станет проще. Подпись «хорошо» я бы объединил со звездами: не так важно, что звезды есть, выбирать надо между состояниями «плохо», «хорошо», «супер».

Важно понимать, сколько у нас всего степеней оценки, поэтому итоговое описание контрола с оценкой такое:

Оценка
Хорошо, 4 из 5

Так же работают и чаевые. Название контролов совпадает с заголовками, которые находятся рядом с ними, значит заголовки не нужны и их можно скрыть от VoiceOver.



А вот типичные проблемы с заказом работают иначе: выбрать можно несколько, а значит трейт .adjustable не подойдет и нужно работать с ними как с отдельными кнопками. Для отмеченных проблем нужно ставить трейт `.selected`.

Текст заголовков мы перенесли в названия контролов, но совсем без заголовков навигация становится не такой удобной, поэтому улучшить навигацию можно с помощью контейнеров.

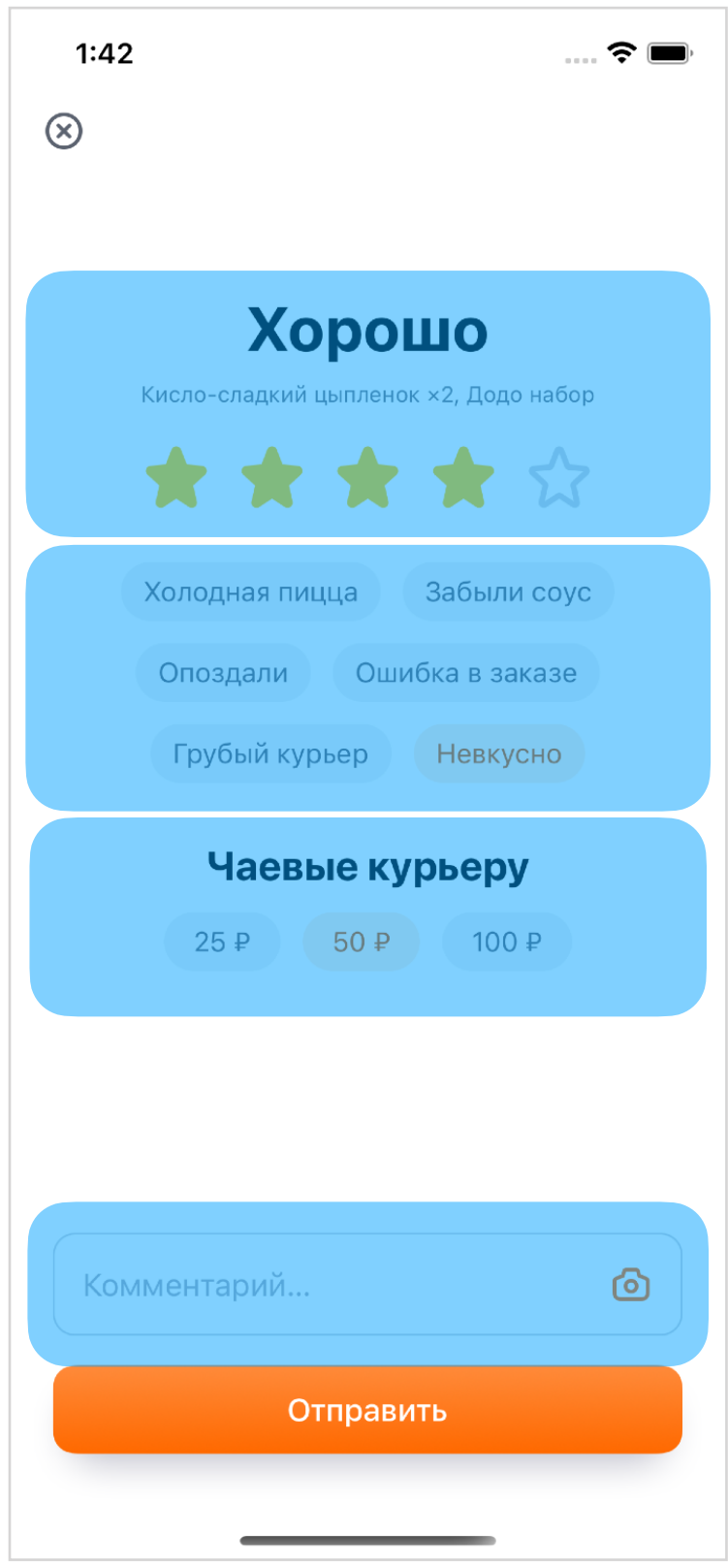
«Комментарий» это обычное текстовое поле, ему нужно лишь дать label.

С кнопкой для фотографий интересная ситуация: скорее всего, незрячий не будет отправлять фотографии, но скрывать этот функционал от него нельзя: вдруг незрячий заказывал пиццу на компанию друзей, они помогут сделать фото, если нужно. Ситуации бывают разные, все не предусмотришь, поэтому наша задача максимально адаптировать существующий, а не решить за пользователя, что можно, а что нельзя.

Кнопка отправить суммирует экран: какую оценку поставили, какие проблемы выбрали и сколько чаевых:

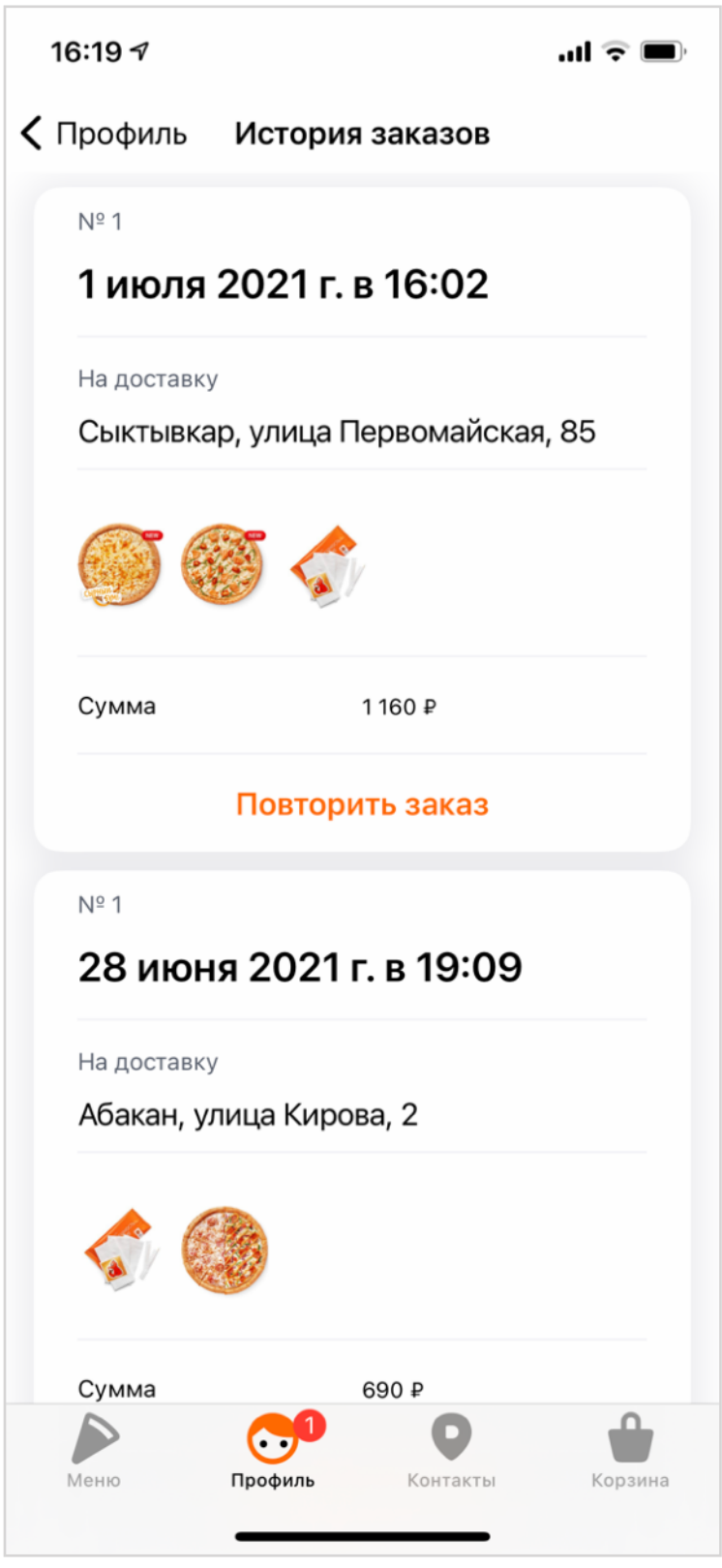
> Хорошо, Невкусно, 50 рублей

Отправить оценку можно через меджик тап.



Пример

История заказов



В ячейках истории заказов слишком много данных, сделать ячейку единым доступным элементом не получится. Описание ячейки можно сделать с помощью customContent, как мы делали в главе «Большие описания».

Можно проще: все надписи объединить в пары, а ячейку сделать контейнером с названием в виде даты.

Экран с описанием продуктов очень похож на меню, адаптация такая же.

Voice

Control

Voice Control

В 2019 году вместе с релизом iOS 13 Apple представила новый механизм доступности — Voice Control. Он позволяет людям с нарушениями моторики управлять приложениями с помощью голоса. При работе накладывает поверх интерфейса подписи, номера или сетку, человек может их называть и так взаимодействовать с интерфейсом.

Voice Control не так требователен к адаптации как VoiceOver, потому что не нужно менять восприятие интерфейса с графического на звуковой. Тем не менее, несколько особенностей у него есть, и использование Voice Control можно сделать сильно удобней.

Телефон учитывает, что он может услышать не только владельца, поэтому он следит за тем, чтобы ваш взгляд был направлен на телефон, когда вы отдаете команду.

Увы, Voice Control пока работает только с английским языком, но всё ещё может поменяться.

Виды навигации

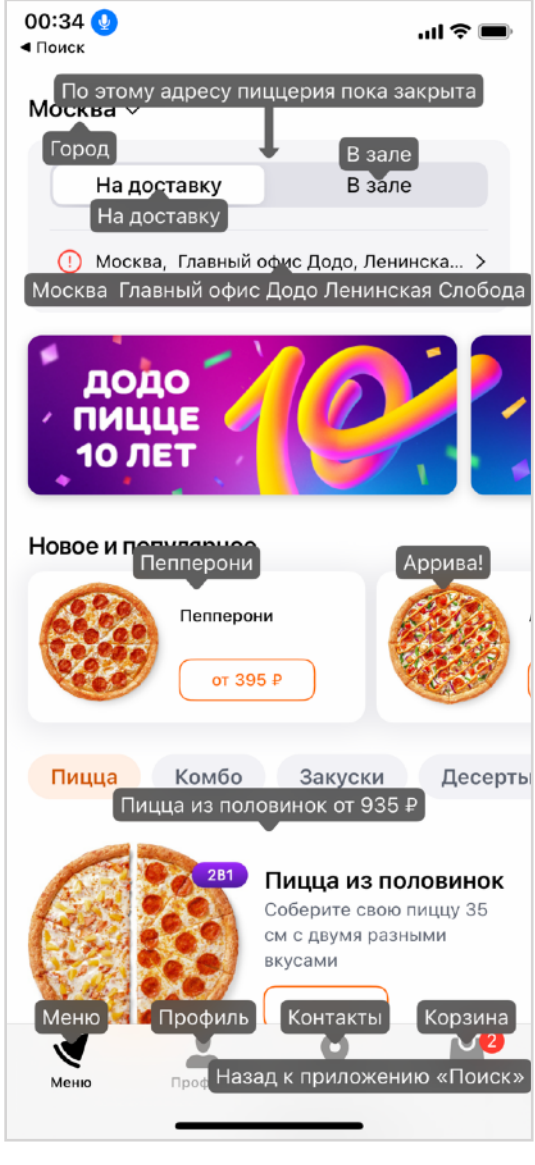
Управление Voice Control происходит с помощью голосовых команд. Команды простые: нажми, проскроль и т.д. Чтобы понять, с каким элементом выполнить действие, VoiceOver предлагает 3 способа именования контролов:

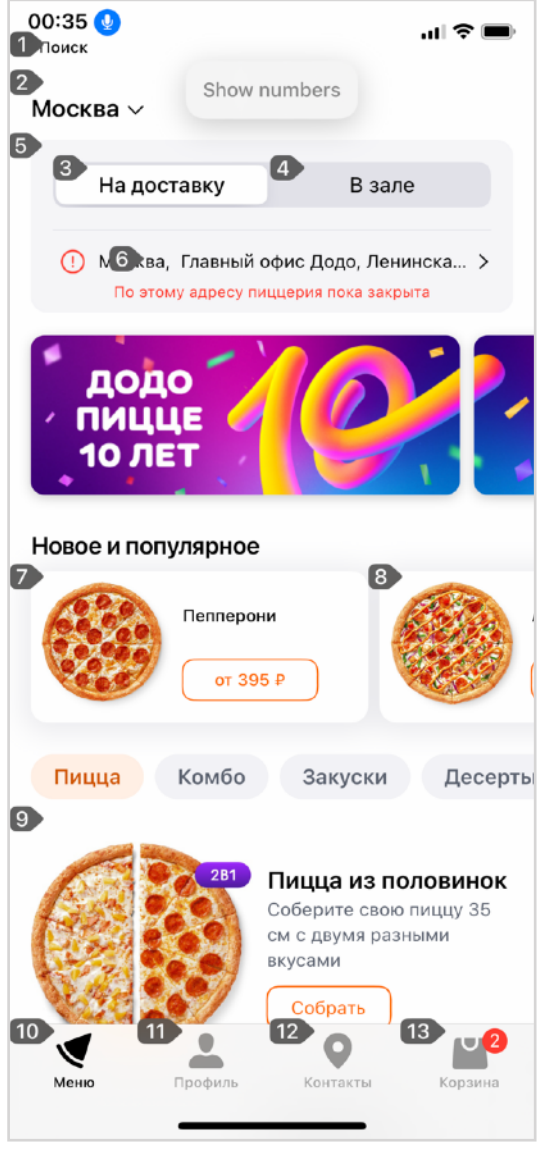
— **Показать надписи.** В адаптированных приложениях это самый удобный вариант, около элементов появляются их названия.

— **Показать цифры.** Voice Control нумерует все элементы, цифры становятся названиями этих контролов. Нажать на кнопку можно командой «нажми 6».

— **Показать сетку.** Вы можете сказать, сколько столбцов вам нужно, чтобы центр ячейки попадал на нужный элемент. Особенно удобен этот режим при работе с картами.

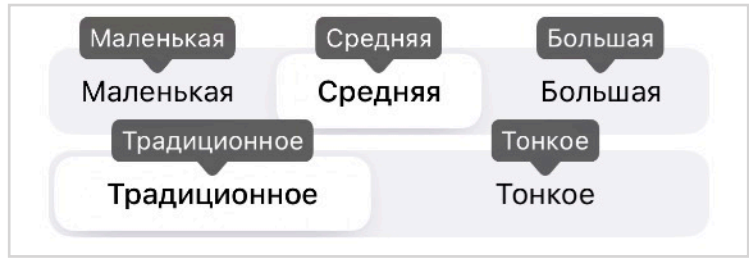
Voice Control позволяет проверять и качество адаптации для VoiceOver, например, покажет, у каких элементов нет названия.





Снова про adjustable

Voice Control иногда расходится с поведением VoiceOver. Например, не стоит делать `UISegmentedControl` как `adjustable`-элемент, для Voice Control удобнее, когда можно нажать на любую кнопку, а не говорить «Размер, увеличить».

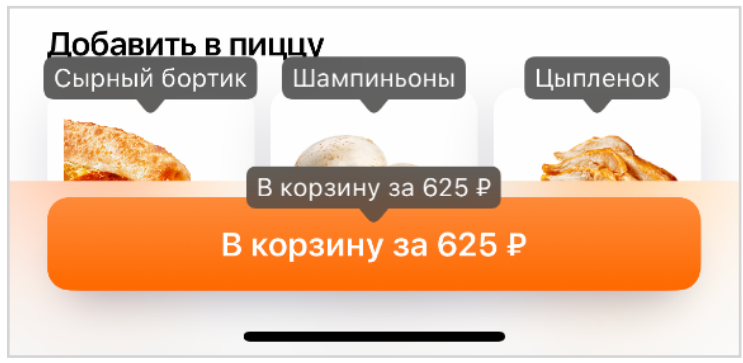


То же самое и с UI-тестами: намного проще нажимать отдельную кнопку, а не свайпать контрол до нужного состояния. Объединять в `adjustable` можно специально для VoiceOver и не включать для других технологий.

```
override public var accessibilityTraits: UIAccessibilityTraits {
    get {
        UIAccessibility.isVoiceOverRunning ? .adjustable: .none
    }
    set {}
}

public override var isAccessibilityElement: Bool {
    get {
        UIAccessibility.isVoiceOverRunning
    }
    set {}
}
```

Альтернативные названия

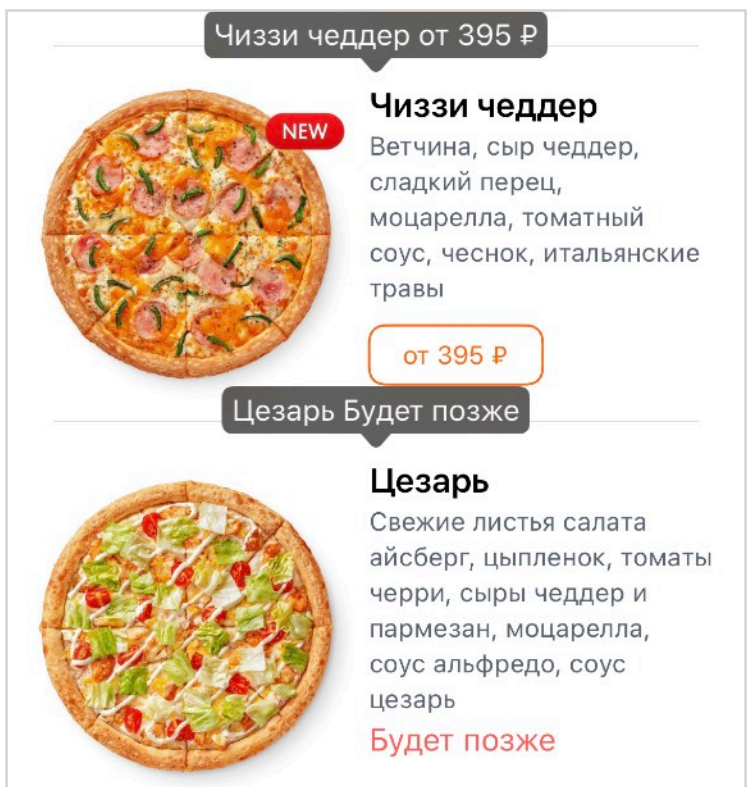
Другая особенность связана с голосовыми командами: мы можем отдавать их по-разному и не всегда это точно совпадает с названием кнопки. Я могу сказать «купить», «в корзину», «добавить» и всё это будет иметь смысл «нажми оранжевую кнопку».



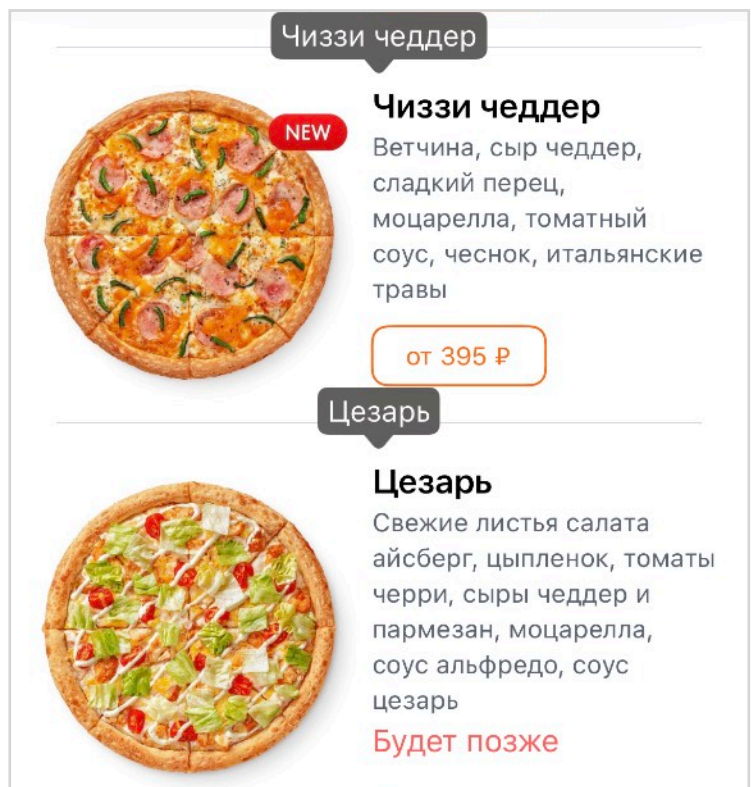
Альтернативы для названия можно уточнить через свойство `accessibilityUserInputLabels`.

```
putToCartButton.accessibilityUserInputLabels
    = ["Купить", "Добавить", "В корзину"]
```

Для ячеек меню мы добавляли цену к названию и записывали текст в `label`, но для Voice Control цена будет лишней. Можно исправить это, указав только название продукта в `userInputLabels`.



Адаптация VoiceOver избыточна для Voice Control

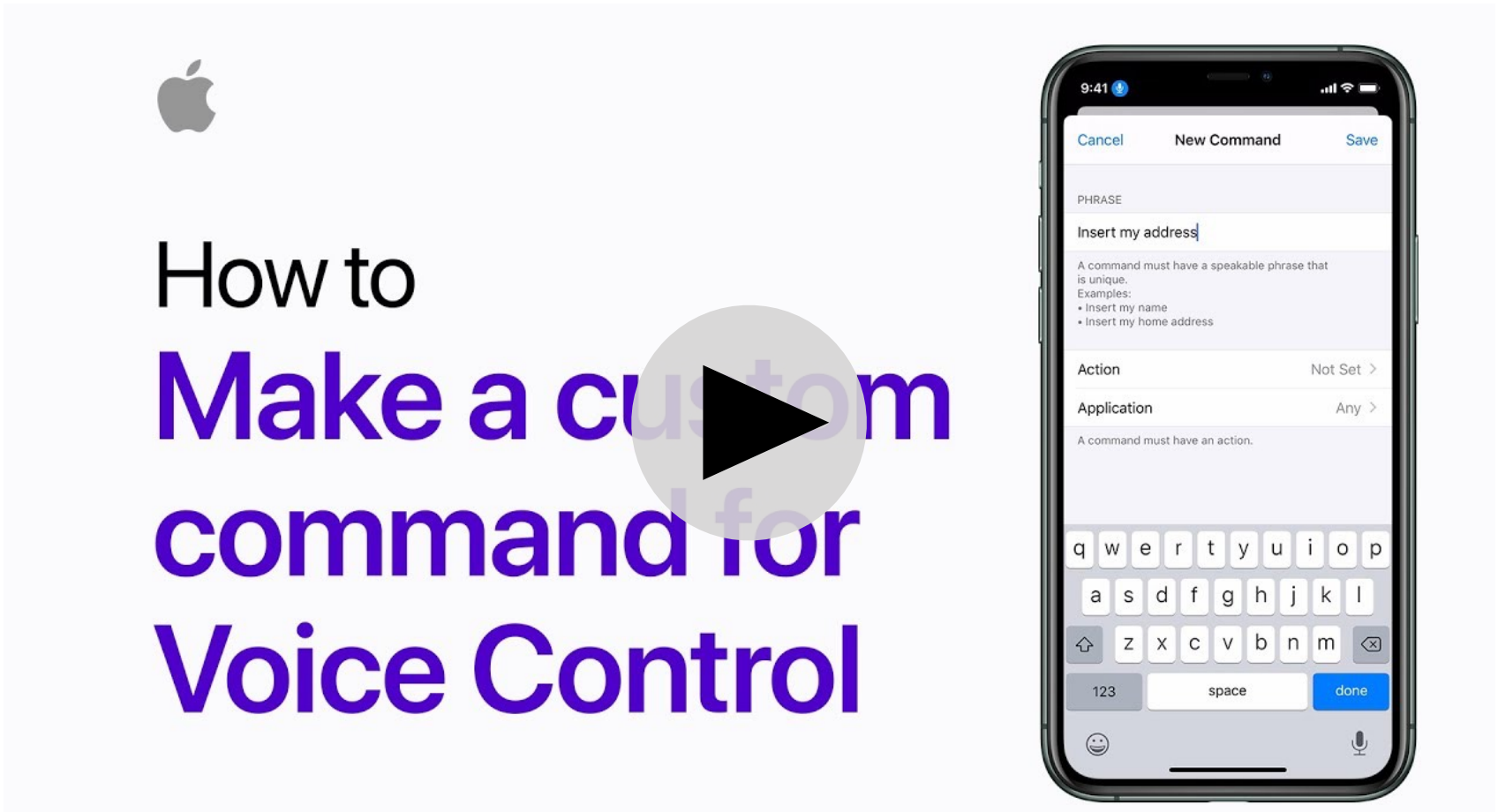


Названия продукта достаточно

Голосовые команды

У Voice Control много команд: для пролистывания, сворачивания приложения, изменения громкости и т.д. С полным списком команд можно познакомиться на сайте imore.com.

Можно создавать свои команды, например, для ввода адреса, об этом есть видео Apple.



Советы по настройке команд для ввода

Switch Control

Switch Control

VoiceOver поможет, если человек не видит, но может касаться экрана, свайпать по нему и слышать речь. Если на экран нельзя нажать, но можно говорить, то выручит VoiceControl.

Если нельзя взаимодействовать с экраном, при этом речь нечёткая и Voice Control не понимает команды, то справится Switch Control. Switch Control — самая экстремальная, но вместе с тем, самая гибкая технология доступности.

Идея Switch Control простая: телефону можно отдавать команды с помощью любого инструмента ввода, достаточно даже одной кнопки или датчика. Кнопку можно нажать кулаком, затылком, ногой, а датчик может быть привязан к одному работающему мускулу на щеке или замечать движение брови с помощью камеры. К такому сигналу можно привязать любое действие на телефоне, и так человек сможет с ним взаимодействовать.

Как управлять

Чтобы понять устройство Switch Control, начнём с самого сложного случая, когда телефону можно подать только один сигнал. Прежде всего к телефону нужно подсоединить любую кнопку или датчик, который может подавать сигнал, прямо как у компьютера: 0 или 1. Теперь этот сигнал нужно превратить в команду. Но в какую?

Точно надо как-то выбирать элементы, но на экране их много, надо как-то перемещаться между ними. Если взять фокус из VoiceOver, то одной кнопкой его можно передвигать, а другой — выбирать. Если перемещать фокус автоматически, то достаточно будет вовремя нажать, чтобы выбрать. Долго? Конечно. Помогает ли это людям в сложных ситуациях? Ещё как!

Конечно, в Switch Control есть разные инструменты, которые ускоряют работу в таком режиме. Способов дать команду может быть несколько, разным командам можно назначить разные действия, каждая дополнительная кнопка сильно повышает удобство использования. Уже с двумя кнопками можно отказаться от автоперемещения фокуса: одной кнопкой переключаться к следующему элементу, а другой — выбирать этот элемент. С тремя кнопками можно перемещаться вперёд, назад и т.д.

Кнопки и датчики могут быть разными: моргнуть, сжать зубы, мотнуть головой и нажать кнопку виском или выгнуться и задеть затылком… Необычные примеры можно посмотреть в лекции Convenience for You is Independence for Me.

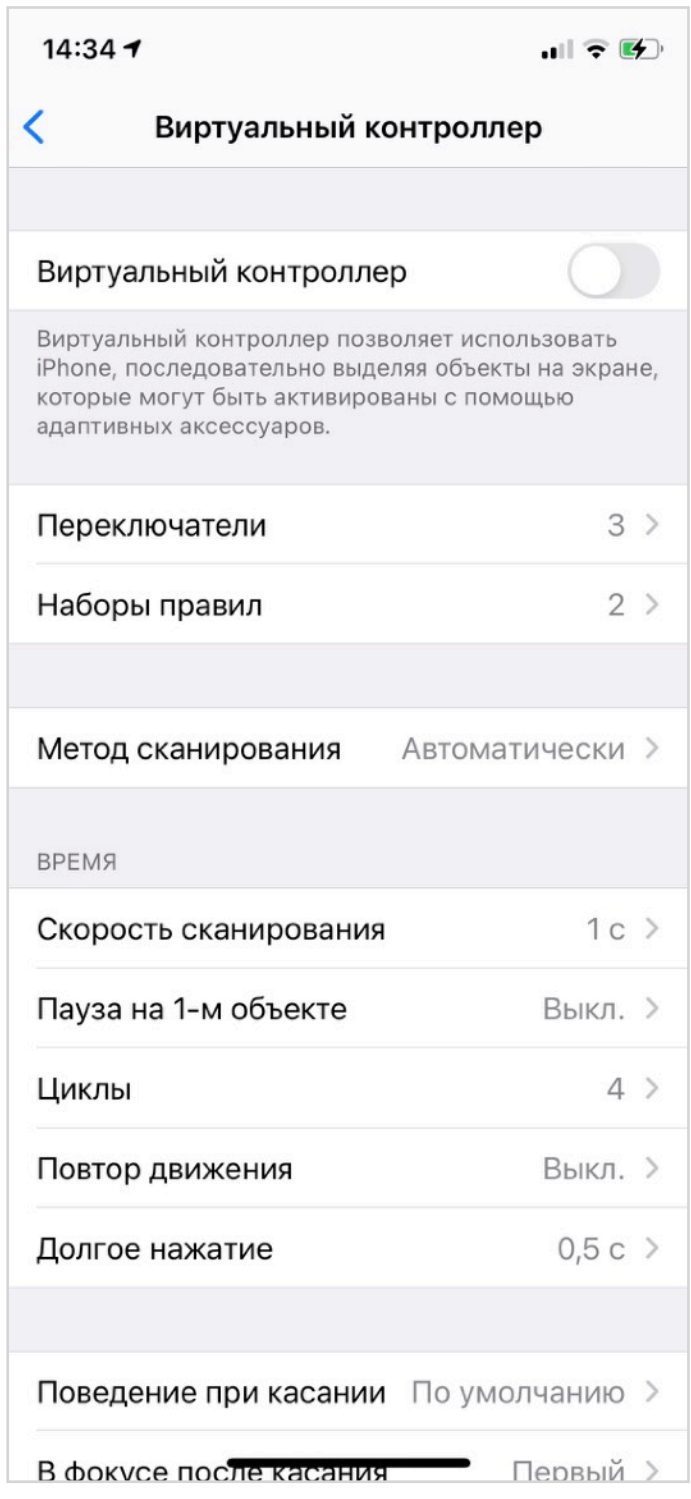
AbleNet Blue2 сделан специально для Switch Control

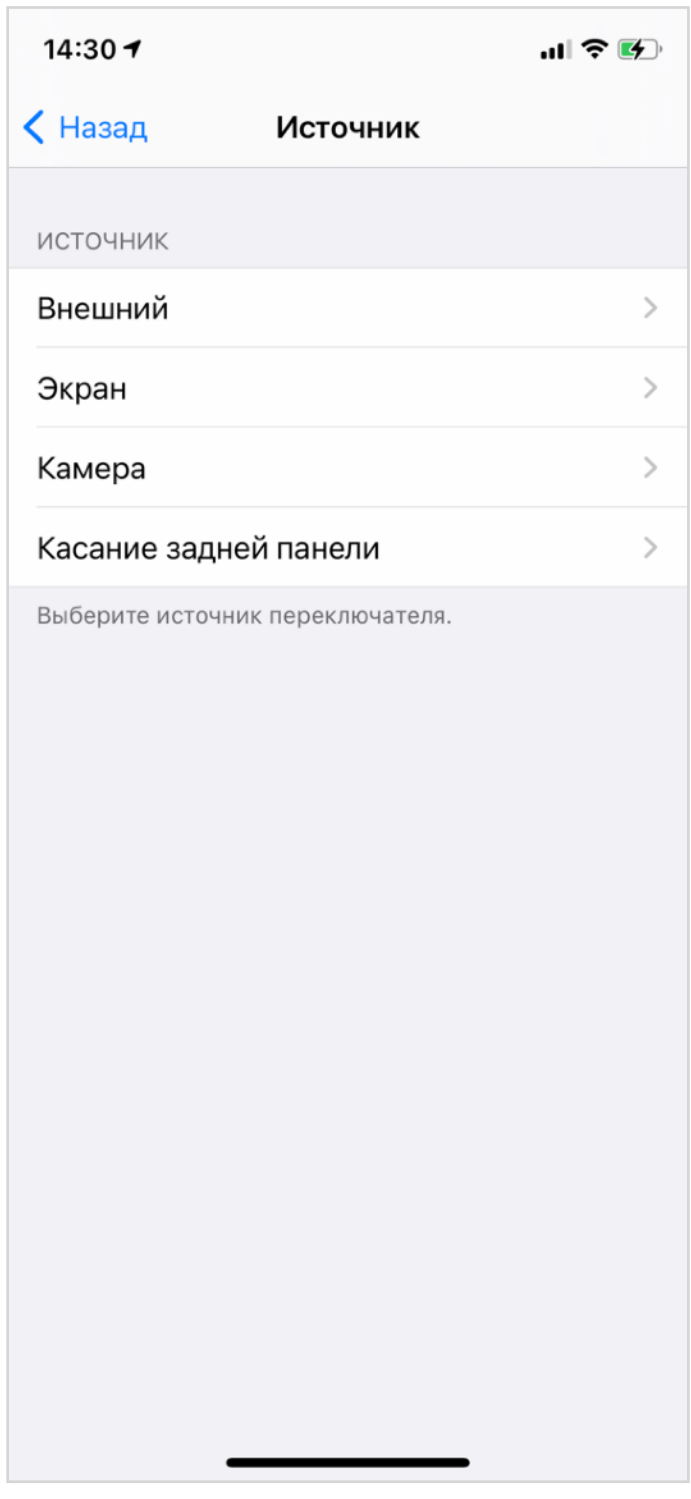
Настройка

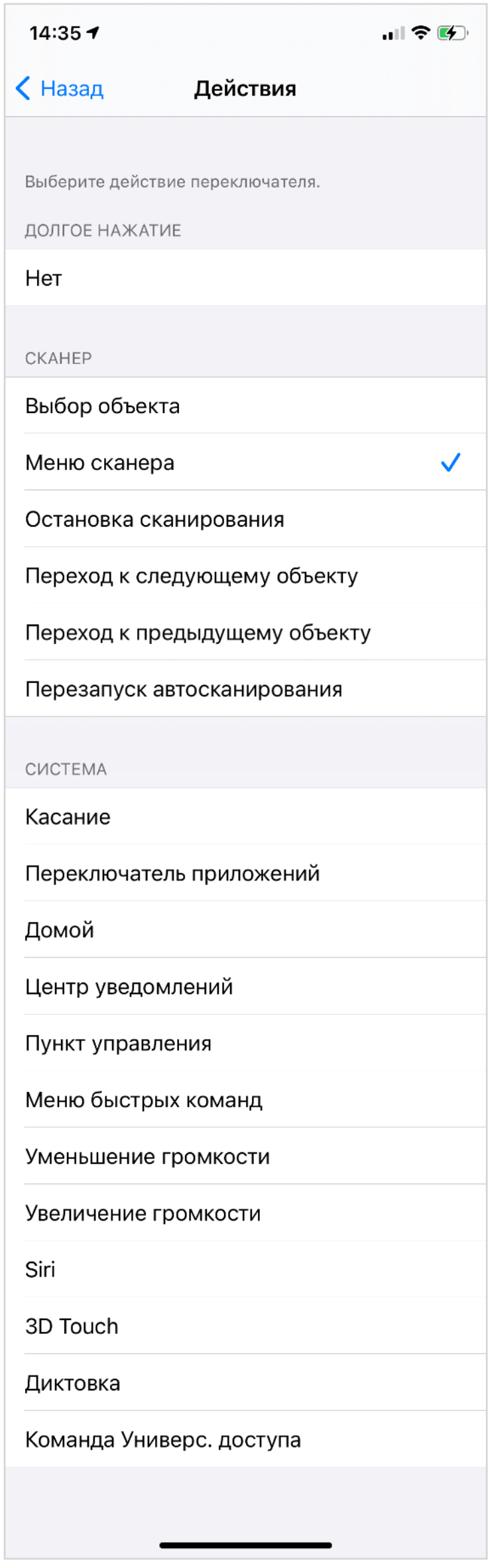
Чтобы начать работать со Switch Control, его нужно настроить: показать, какой сигнал вы отправите телефону и какое действие он выполнит.

Настройки → Доступность → Виртуальный контроллер → Переключатели.

Сначала выбираем источник сигналов: это может быть внешнее устройство (клавиатура или другие кнопки), касание экрана, поворот головы на камеру или 2-3 касания задней панели телефона.







Возможные варианты действий.

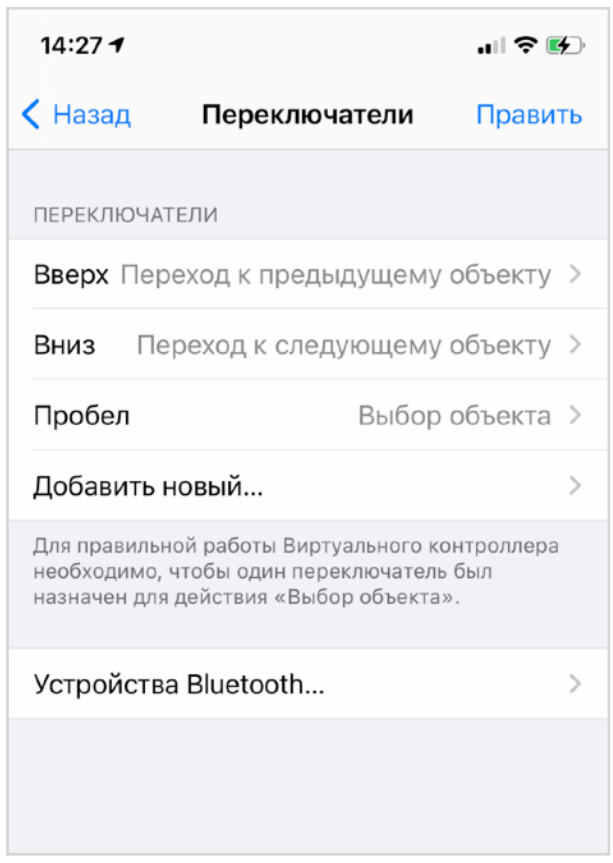
К источнику нужно добавим действие. Для тестирования понадобится несколько функций:

— переход к следующему объекту с помощью стрелок;

— выбор пробелом;

— вызов меню, зажатием пробела. Зажимание клавиши дополнительно включим в группе настроек «Время».

Минимально должно быть хотя бы одно действие «Выбор объекта», остальное на усмотрение пользователя. Случай с одной кнопкой интересный: Switch Control сам проходит по всем элементам, доходит до конца экрана и начинает заново — человеку надо только вовремя выбрать.



Мой вариант конфигурации

Switch Control

Навигация

Режим объектов

Точный выбор

Контекстное меню

Фокус Switch Control работает в двух режимах: «режим объектов» и «точный выбор».

На примере спрингборда: в режиме «объектов» он сначала проходит по рядам, а после выбора ряда начнёт ходить по иконкам внутри. В таком режиме он знает про отдельные контролы и учитывает адаптацию VoiceOver.

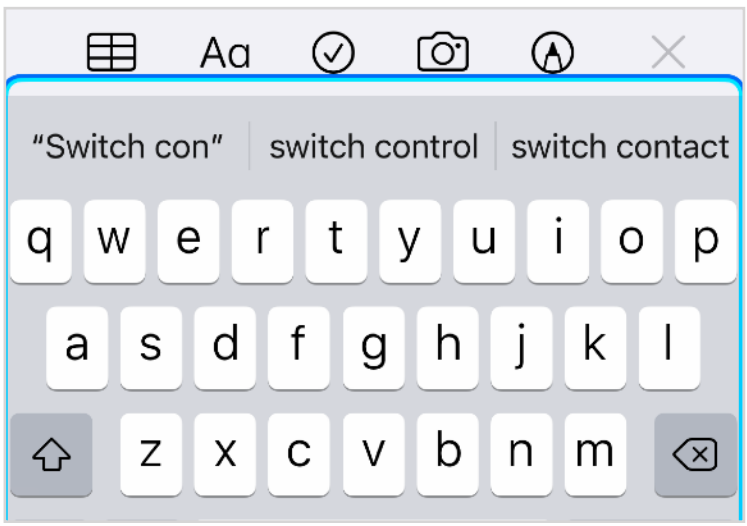
В режиме «точного выбора» можно навести прицел на любую кнопку. Сначала будет перемещаться большая горизонтальная полоса, после выбора начнет бегать по вертикали тонкая линия, после ещё одного выбора линия начнёт перемещаться горизонтально.

После выбора элемента появляется контекстное меню. Его содержимое можно изменять в настройках iOS, но тут же будут появляться и `UIAccessibilityCustomAction`. Для Switch Control можно добавить к действиям иконку, будет удобней.

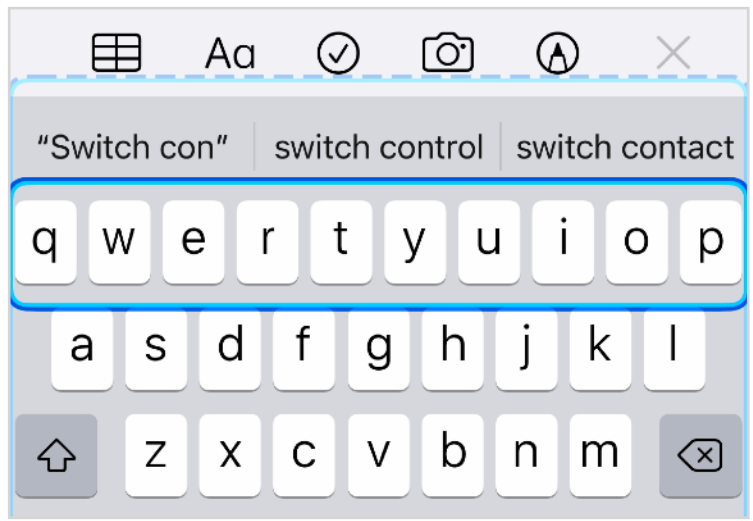
Сложно описать, проще один раз увидеть на YouTube.

Режим объектов

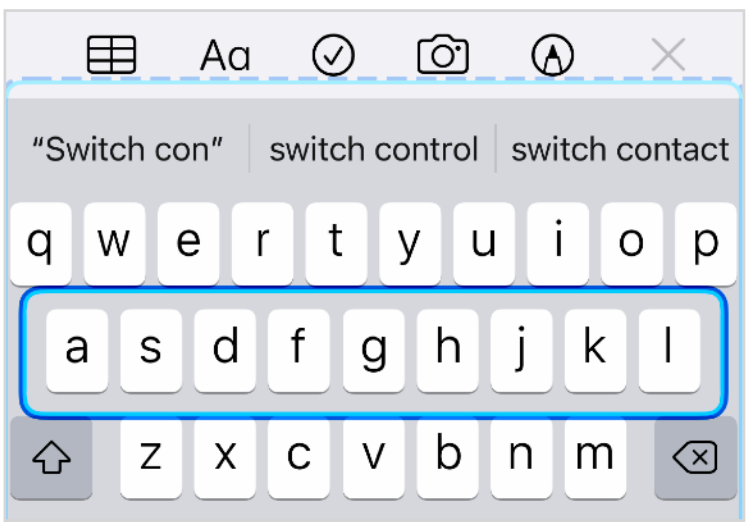
Switch Control старается повысить эффективность навигации с помощью группировки объектов. Разберём на примере клавиатуры. Кнопок — тьма, проходить по всем по очереди было бы слишком долго. Например, чтобы выбрать букву «f» надо пройтись фокусом по всем кнопкам, а это 23 элемента. Но зачем, если можно в 3 раза быстрее?



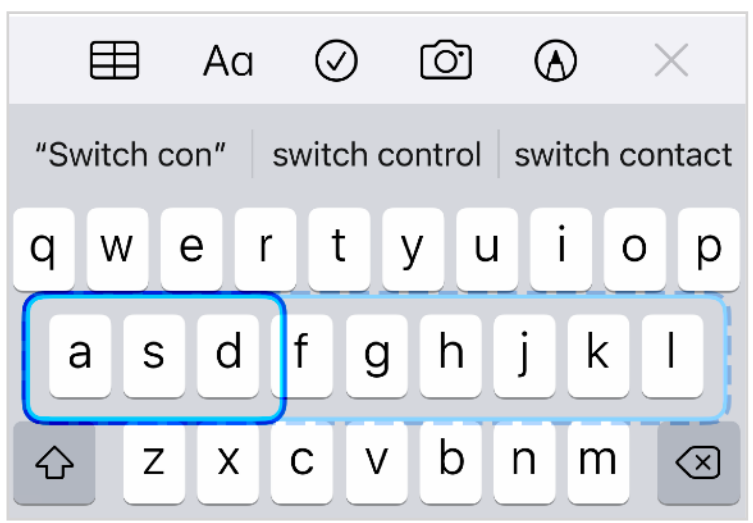
1. Выбираем область с клавиатурой



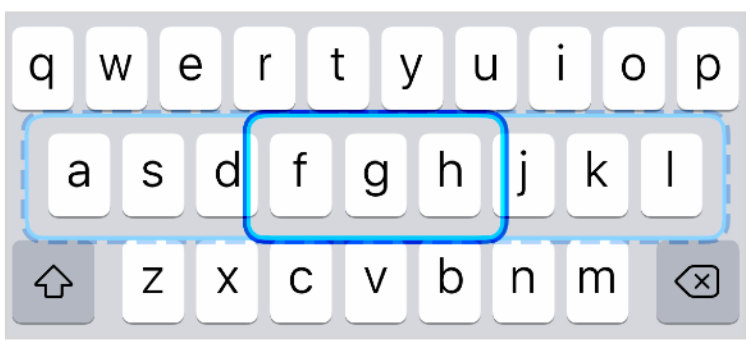
2. Переключаем ряды



3. Выбираем второй ряд

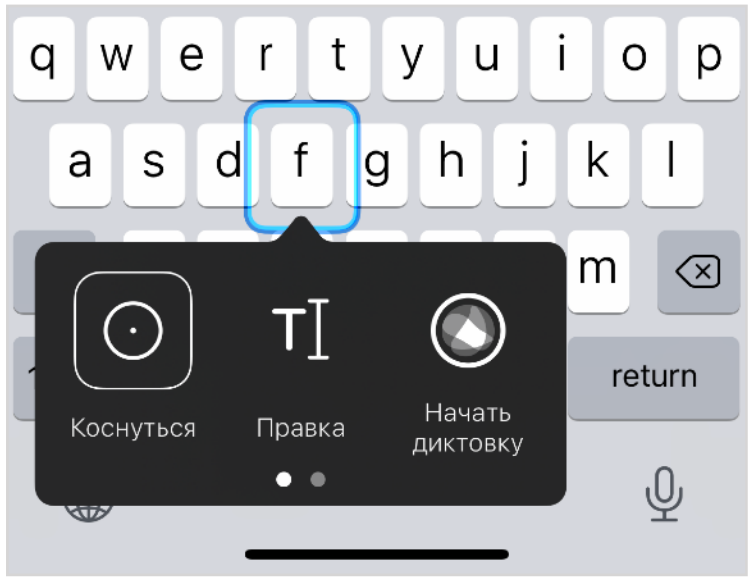


4. Переключаем среди групп по три символа



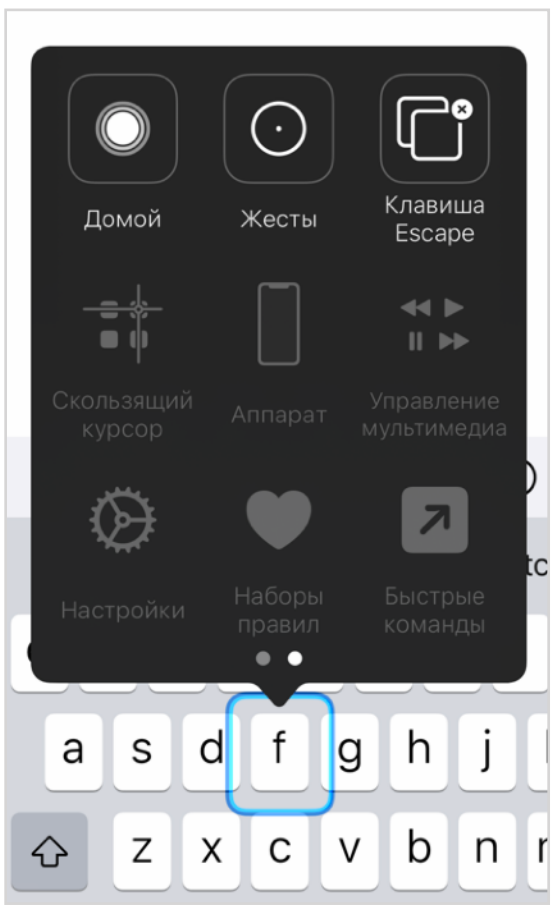
5. Выбираем вторую группу

6. Выбираем символ f

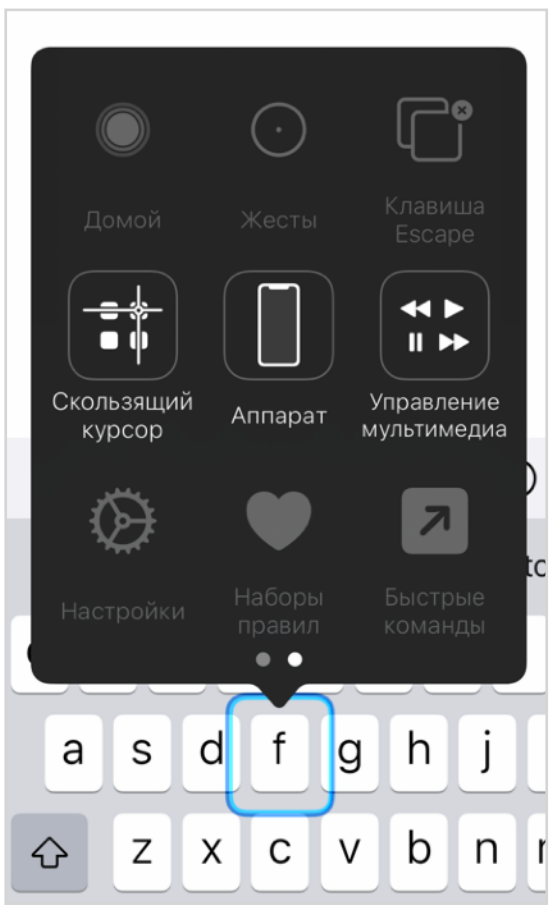


Открывается меню с действиями.
7. Выбираем «коснуться»

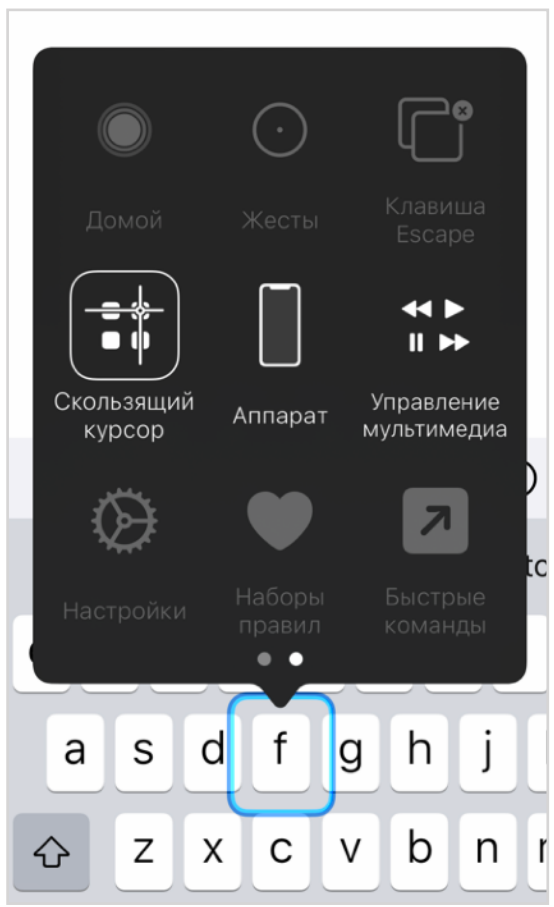
Внутри меню можно выбрать три точки и откроются остальные действия. Навигация внутри них точно такая же, только меняется стиль оформления.



Выбираем ряд



Второй



Выбираем действие

Switch Control проходит фокусом сначала по рядам, затем по группам из трёх клавиш, а в конце по отдельным клавишам. Каждый раз выбор только из трёх-четырёх элементов, цикл прохода повторяется часто, выглядит удобно. Справились за 7 действий.

Ввод целого слова можно посмотреть на YouTube.

За разделение на такие группы отвечает свойство `accessibilityNavigationStyle`. Обычно там стоит `.automatic`, Switch Control опирается на иерархию `UIView` и группирует элементы по ним. Режима навигации три: отдельными элементами, группами и на усмотрение Switch Control — можете уточнить нужный режим для существующих вью.

Если вы хотите уточнить поведение, а общей `UIView` нет, то можете создать свой `UIAccessibilityCustomItem`, добавить в него кнопки и указать нужный стиль навигации.

```
/*
 The following values describe how the receiver's elements
should be navigated by an assistive technology.
 */
@available(iOS 8.0, *)
public enum UIAccessibilityNavigationStyle : Int {


    /*
     The assistive technology will automatically determine how
the receiver's elements should be navigated.
     This is the default value.
     */
    case automatic = 0


    /*
     The receiver's elements should be navigated as separate
elements.
     */
    case separate = 1


    /*
     The receiver’s elements should be combined and navigated as
a single item.
     When the combined item has been selected, the assistive
technology will navigate each element separately.
     */
    case combined = 2
}
```

Switch Control

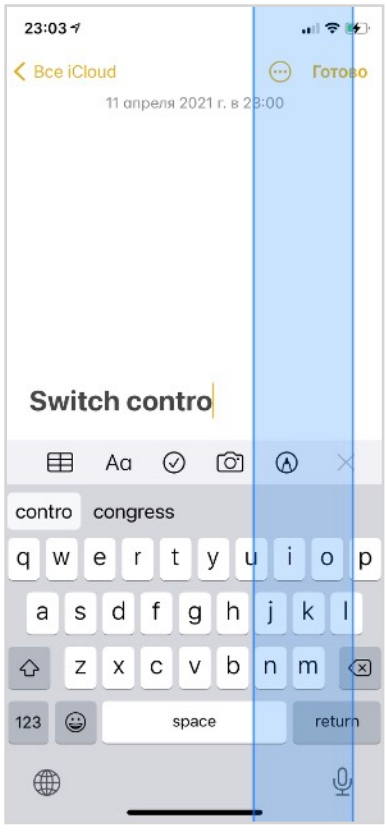
Скользящий курсор

В режиме объектов не всегда получается выбрать нужный элемент, особенно если приложение не адаптировано. Для сложных случаев есть режим точного выбора, или, по-другому, «скользящий курсор». Этот режим напоминает игровой автомат, в котором надо вытащить игрушку робо-рукой.

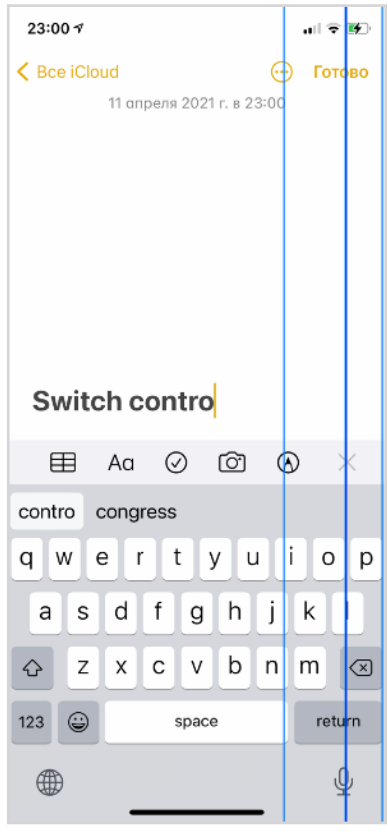
Прицел сам ходит туда-сюда, нужно только вовремя нажать для выбора.

1. Наведите вертикальную область на нужное место по горизонтали.
2. Выберите точное расположение по горизонтали
3. Повторите оба шага по вертикали.
4. Выберите нужное действие во всплывающем окне уже в режиме объектов.

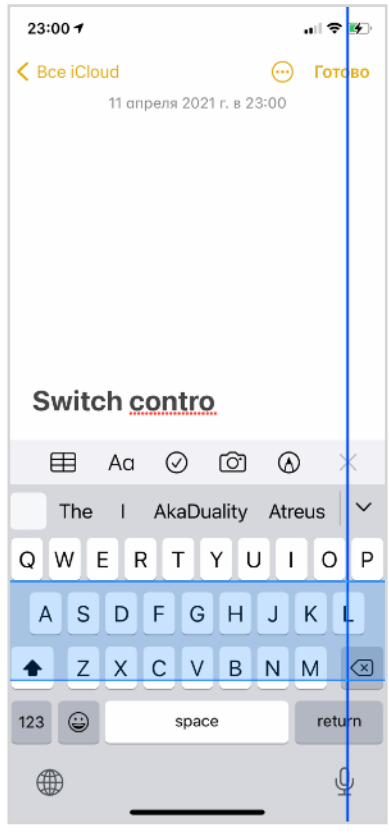
Видео на YouTube.



Наводим область по горизонтали



Внутри области точно наводим на кнопку



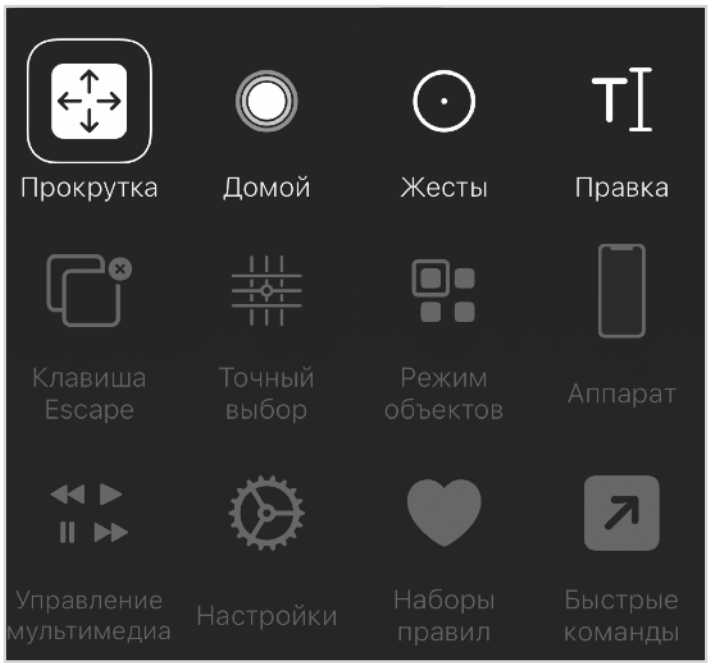
Наводим область по вертикали

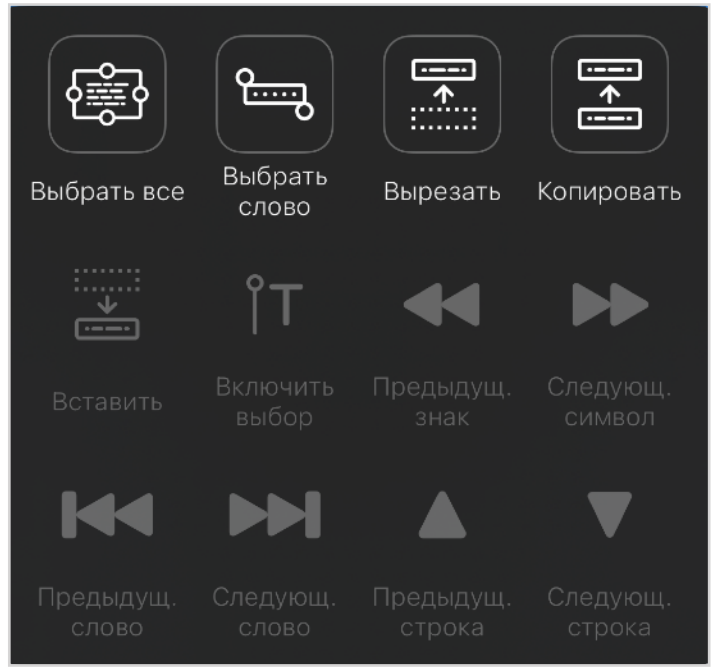
Внутри области точно наводим на кнопку

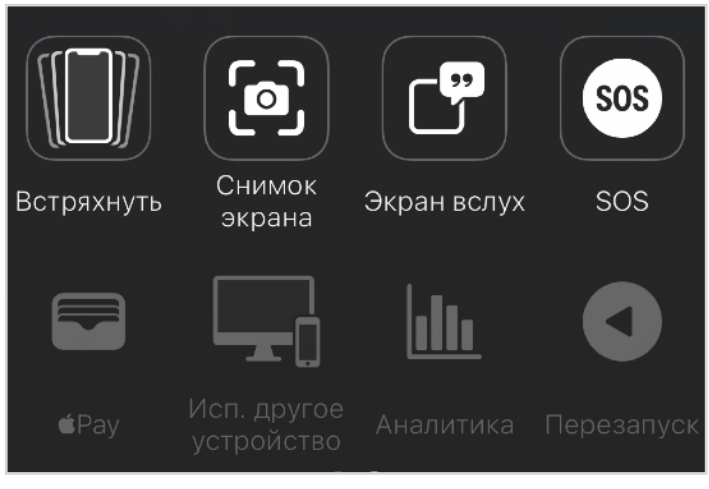
Контекстные действия

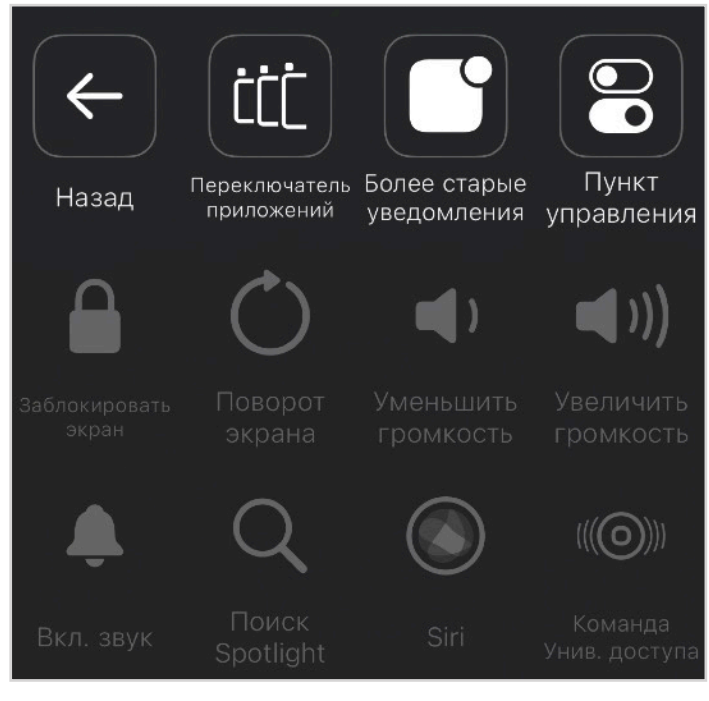
Switch Control понимает все контекстные действия, которые мы делали для VoiceOver, но ему можно дополнительно указать иконку действия. Ничего сверхординарного не нужно, просто визуальный маячок.

Если иконку не указать, то вместо неё будет просто точка.

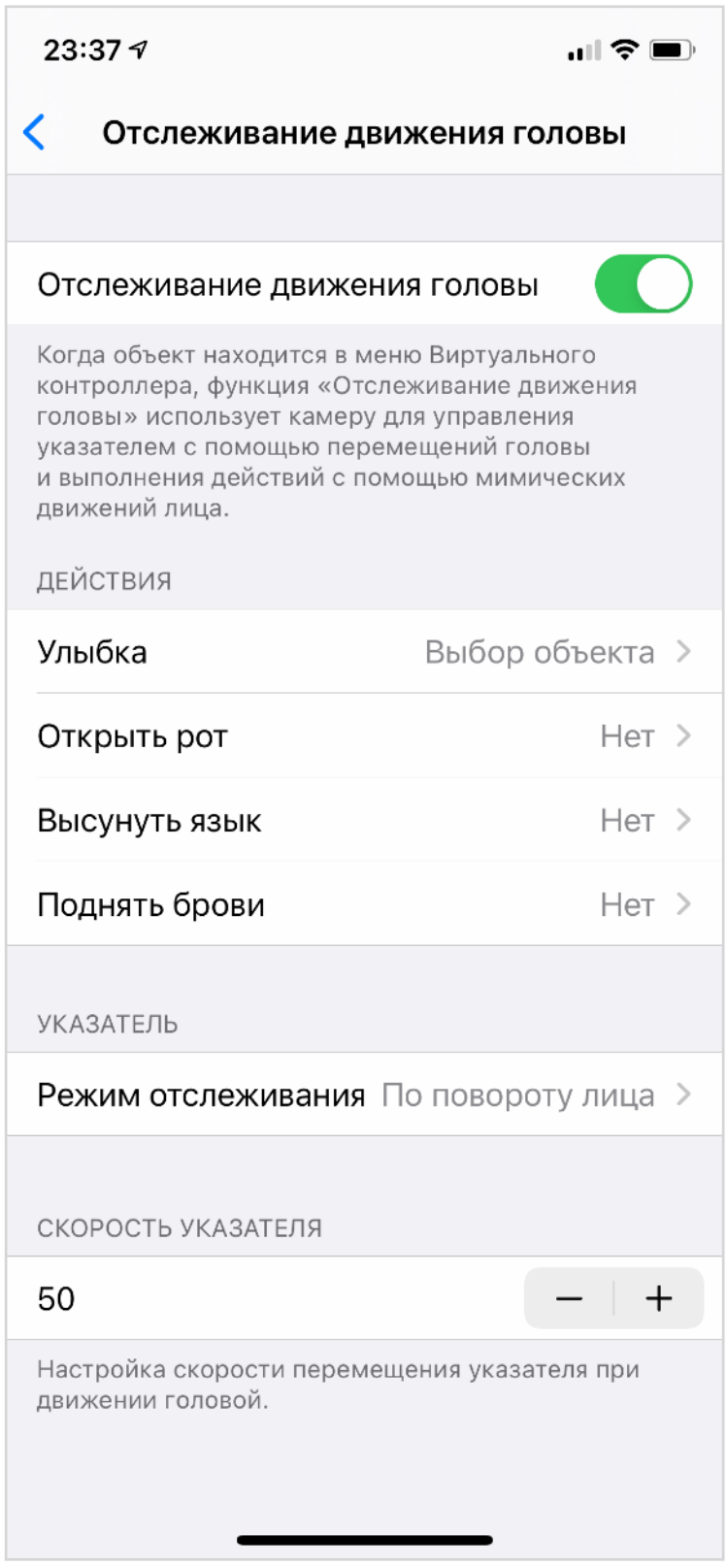








Трекинг головы



Иногда проще не нажимать кнопки, а управлять положением головы и мимикой лица. На экране можно показать курсор в виде круга, поворотом головы двигать его, а мимикой — выбирать.

В качестве действия можно использовать одно из мимических движений лица:

— улыбку;

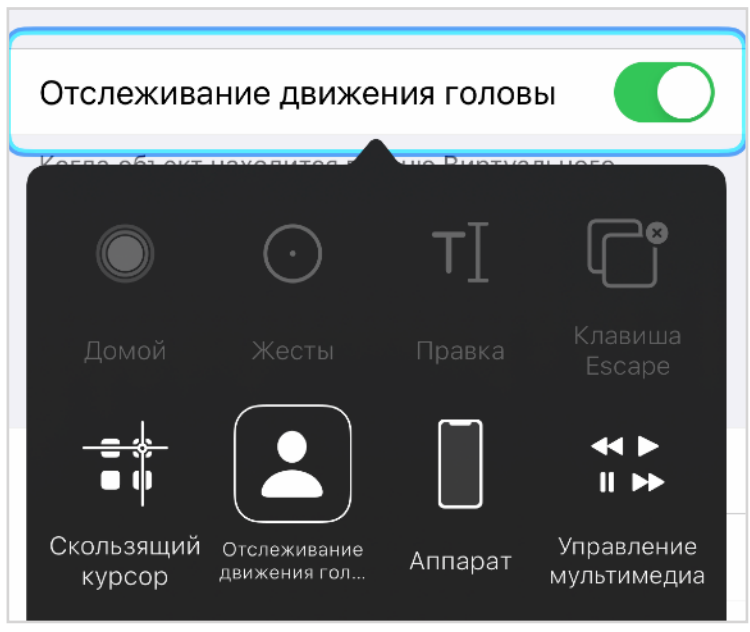
— открытие рта;

— высунутый язык;

— поднятие бровей.

Трекинг головы — это такой же режим, как и режим объектов. Чтобы он заработал, его надо выбрать в меню.

И снова в действии на YouTube.



Трекинг головы нужно выбрать в этой панеле.

Начиная с iOS 15 управлять можно не только головой, но издавая разные звуки. Пример посмотрите на видео из твитера.

Dyna

mic

type

Dynamic type

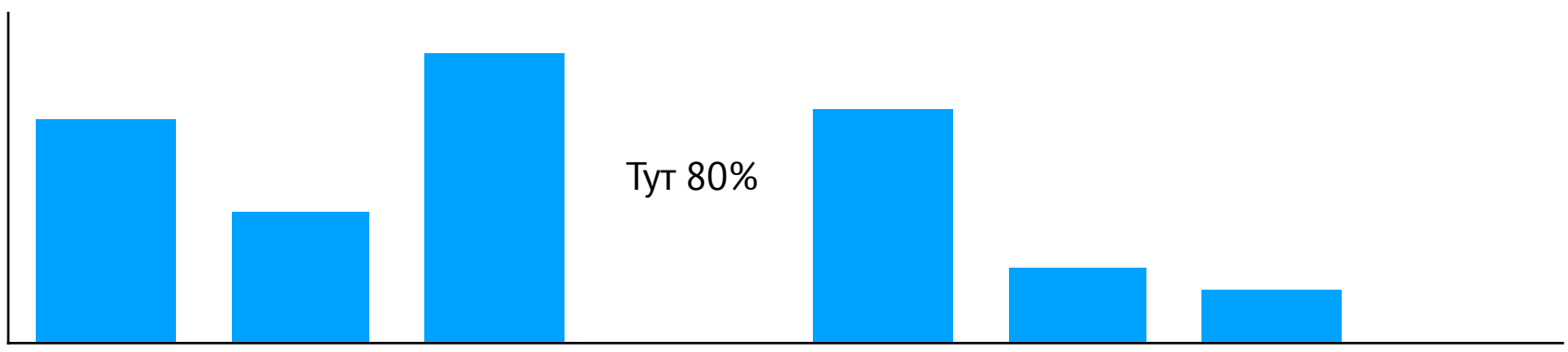
У многих людей есть проблемы со зрением, чтобы прочитать текст в книге, приходится надевать очки или линзы. Телефон и компьютер намного «пластичней»: текст можно увеличить настолько, что можно его прочитать даже без очков. Вместе с текстом нужно увеличить и остальной интерфейс.

iPhone готов помочь нам: есть несколько настроек, с помощью которых можно сделать интерфейс крупнее. Можно увеличить чуть-чуть, можно очень сильно, а если приложение не поддерживает динамичную верстку, то включить экранную лупу.

Вдобавок есть несколько настроек, которые важно учитывать, чтобы облегчить использование приложений для людей, которые не различают цвета, не могут работать с анимированным интерфейсом или слабо различают буквы.

Статистика

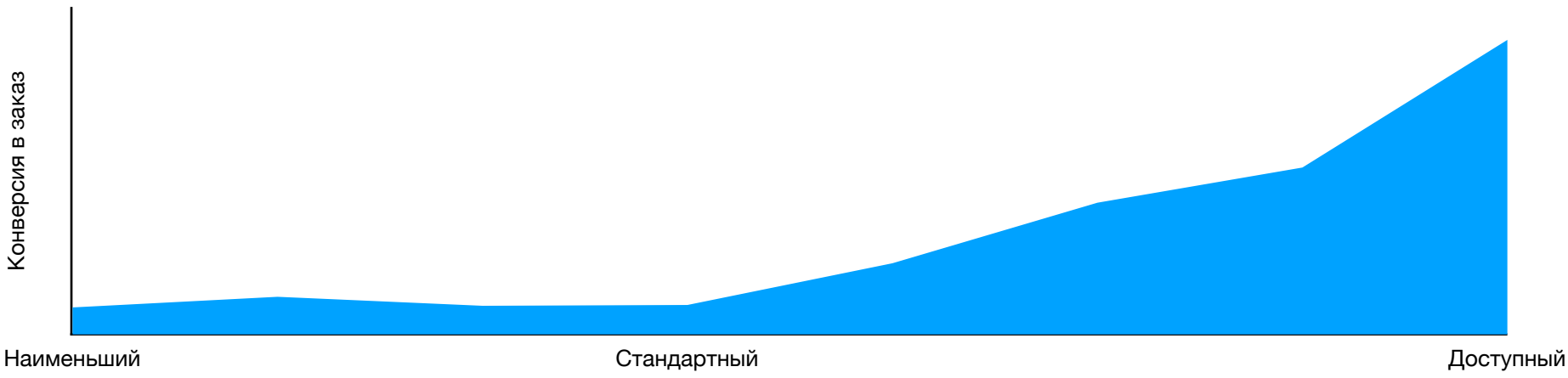
Dynamic Type широко распространён: по нашей аналитике, 20% людей меняют стандартный размер шрифта (статистика собрана по России в приложении Додо Пиццы, наша аудитория моложе 40 лет). У приложения Immoweb цифры выше — до 30%. Кажется, что эти цифры будут только расти: новое «старшее» поколение будет использовать компьютеры и телефоны активнее, чем текущее. Если наши бабушки не знали, что такое интернет, то мы и наши родители пользуемся им каждый день.



Распределение количества пользователей по выбранному размеру шрифта без учёта стандартного размера

— 13% выбирают шрифт поменьше.
— 7% выбирают побольше.
— 0.04% выбирают «доступные» размеры шрифта, когда текст настолько большой, что нужно даже менять верстку. В приложении Додо Пиццы это полторы тысячи человек.

Самым интересным оказалось, что те, кто выбрал больший размер шрифта, заказывают увереннее, конверсия растёт до +5%. Т.е. для 7% очень лояльных людей можно сделать приложение ещё удобней. Мы замеряли на частично адаптированном приложении, интересно будет сравнить цифры после полной адаптации.



С увеличением шрифта конверсия растёт до +5%. График построен относительно базовой конверсии.

Dynamic type

Стили текста

В iOS 7 Apple представила «стили» шрифта: набор начертаний, которые вы можете использовать для вёрстки.

Стандартных стилей 11.

Headline
Subhead
Body
Callout
Footnote
Caption 1
Caption 2

Large Title
Title 1
Title 2
Title 3

Стиль текста можно выбрать в Interface Builder или через код.

```
label.font = UIFont.preferredFont(forTextStyle: .body)
```

Dynamic type

Адаптивный размер

Разными стилями шрифта дело не ограничивается. Если выбрать один из таких стилей, это повлияет на настройку предпочитаемого размера шрифта: он может сильно увеличиваться или немного уменьшаться. Настолько сильное изменение размера текста обязательно надо учитывать в вёрстке, иначе вы можете сломать интерфейс для части людей.

Минимальный, стандартный и самый большой размер. Разница в 4 раза.

Не только текст меняет свой размер, но и контролы вынуждены адаптироваться. Что-то перестаёт помещаться в размеры экрана и нужно добавить скролл. Где-то надо перерисовать или изменить вёрстку. Что-то нельзя увеличить и надо дать другой способ ввода.

В итоге при увеличении шрифта меняется всё. Сначала может показаться, что тут много работы, но на самом деле нужно лишь придерживаться нескольких принципов и использовать несколько вспомогательных контролов, а десяток размеров текста можно свести до двух: обычные размеры со стандартной вёрсткой и большие размеры с адаптацией.

Принципы интерфейса для Dynamyc Type:

— все контролы сами отвечают за свой размер;
— каждый экран может скролиться;
— меняем вёрстку и поведение только для больших размеров;
— лучше всего, когда контрол занимает всю ширину экрана.

Каждый из принципов разберём отдельно.

XS, 80%

Body

S, 85%

Body

M, 90%

Body

L, 100%

Body

XL, 110%

Body

XXL, 120%

Body

XXXL, 135%

Body

AM, 160%

Body

AL, 190%

Body

AXL, 235%

Body

AXXL, 275%

Body

AXXXL, 310%

Body

Все варианты стиля body. Стандартный размер первый во втором ряду

Dynamic type

Как изменить размер

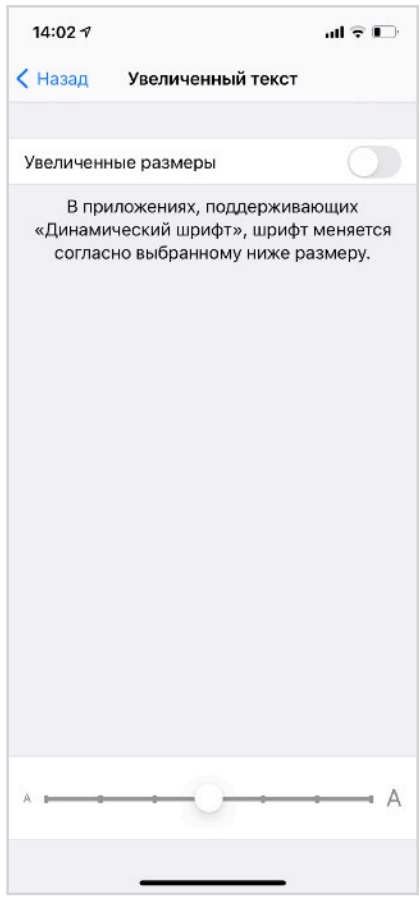
При адаптации вам придётся часто менять размер текста, для разных случаев подходят разные способы. Всего изменить размер можно четырьмя способами.

Единый размер в настройках телефона

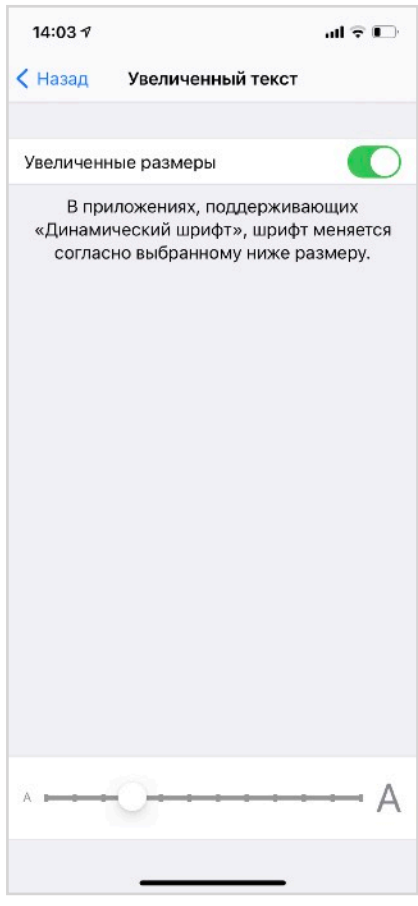
Настройки → Универсальный доступ → Дисплей и размер текста → Увеличенный текст

Скролл позволяет выбрать один из пяти размеров. Текст изменится незначительно и для таких изменений особой адаптации не нужно, достаточно того, что контролы правильно рассчитают свой размер.

На том же экране есть свитчер «увеличенные размеры». При его включении размеров станет больше, для самых крупных уже нужно будет заниматься адаптацией интерфейса.



Стандартный размер находится в середине шкалы

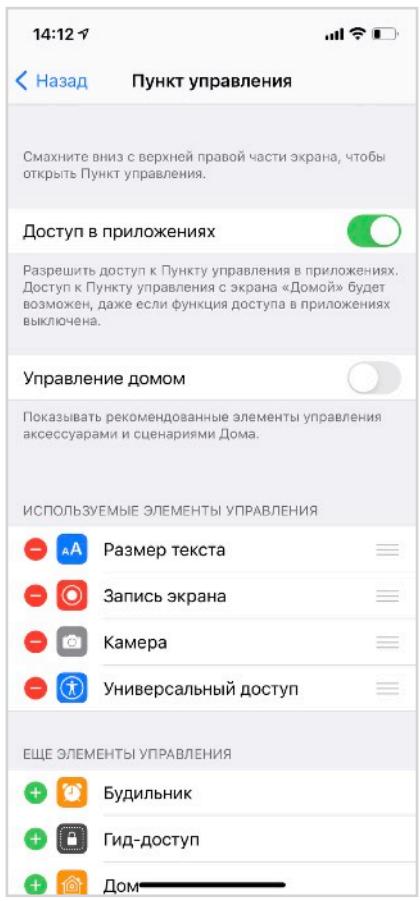


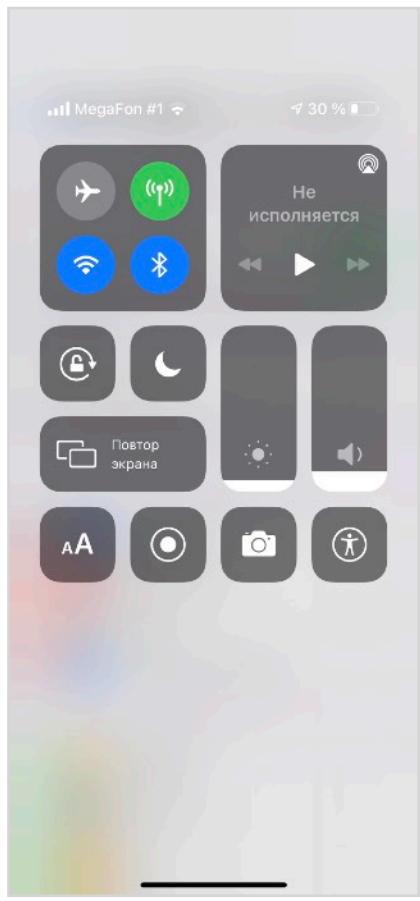
При включении «увеличенных размеров» вариантов становится намного больше

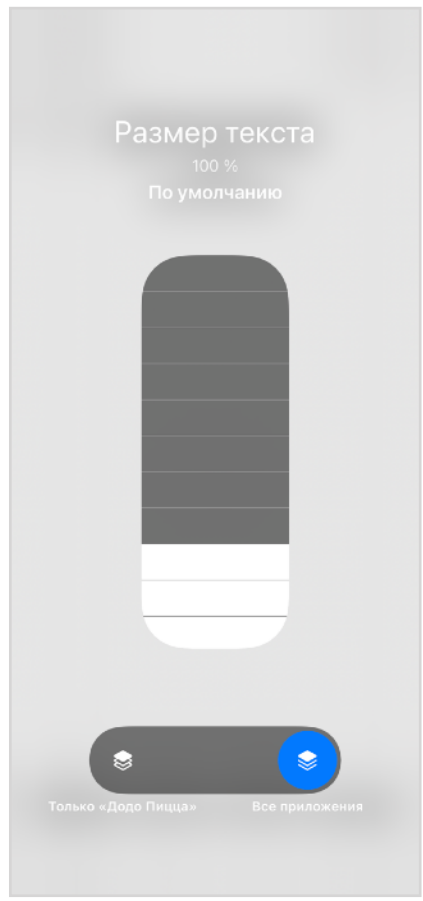
Быстрый доступ через Центр управления

При частой смене размера удобно пользоваться Центром управления. В него можно вынести настройку «Размер текста» – появится кнопка с регулятором размера. Стандартное значение дополнительно подписывается как «По умолчанию».

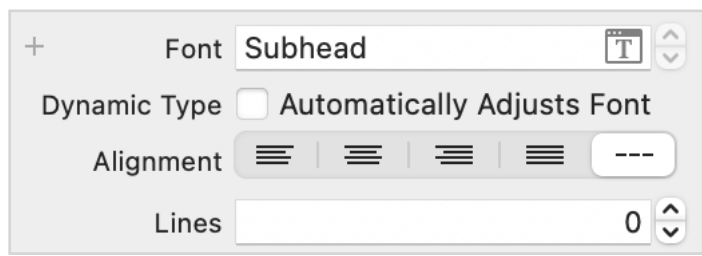
В iOS 15 можно выбрать, на что влияет эта настройка: на все приложения или только на открытое. Удобно отключать dynamic type, если какое-то приложение его неправильно поддерживает.







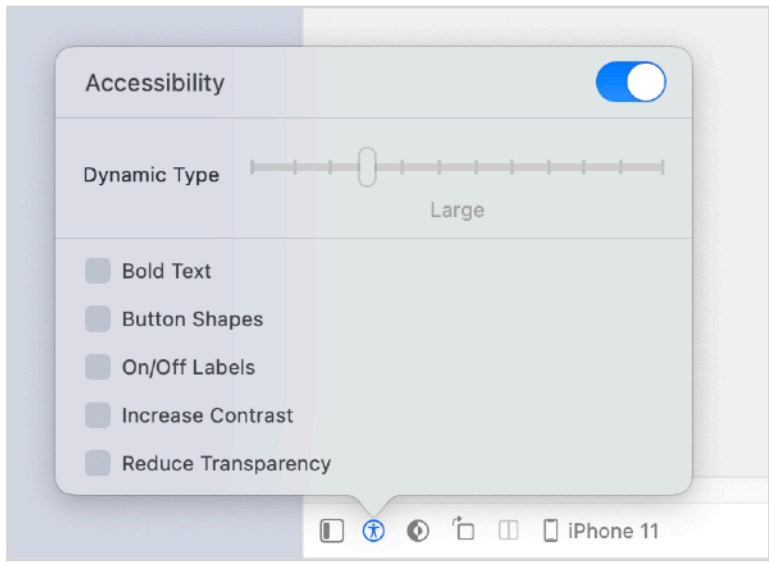
Галочка Automatically adjusts fonts заставит шрифты реагировать на изменение настроек без перезапуска приложения или перерисовки экрана, что удобно для отладки. Задать значение можно и через код, называется чуть-чуть иначе.




```
label.adjustsFontForContentSizeCategory = true
```

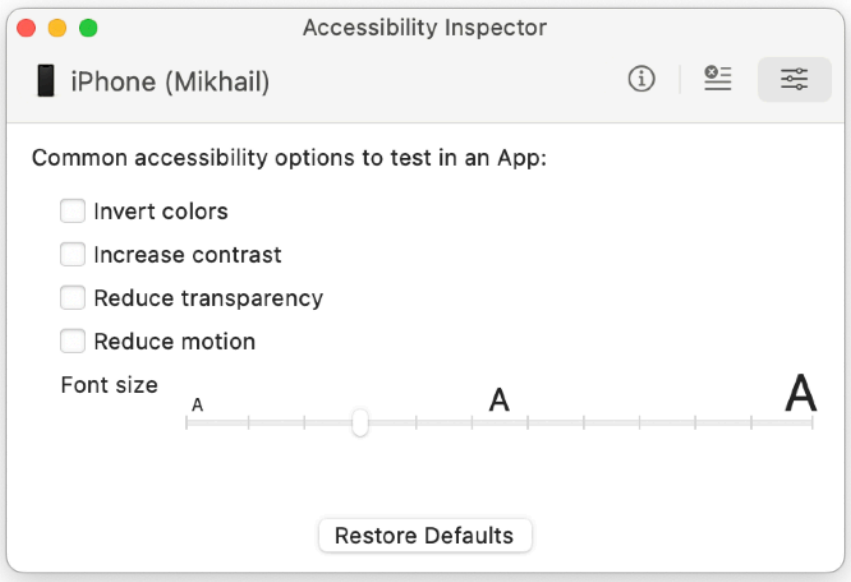
При разработке через Interface Builder

В Xcode 13 появилась возможность менять размер текста в Interface Builder. Это самый быстрый способ посмотреть на поведение интерфейса, даже не запуская симулятор. Там же находятся другие «зрительные» настройки доступности.

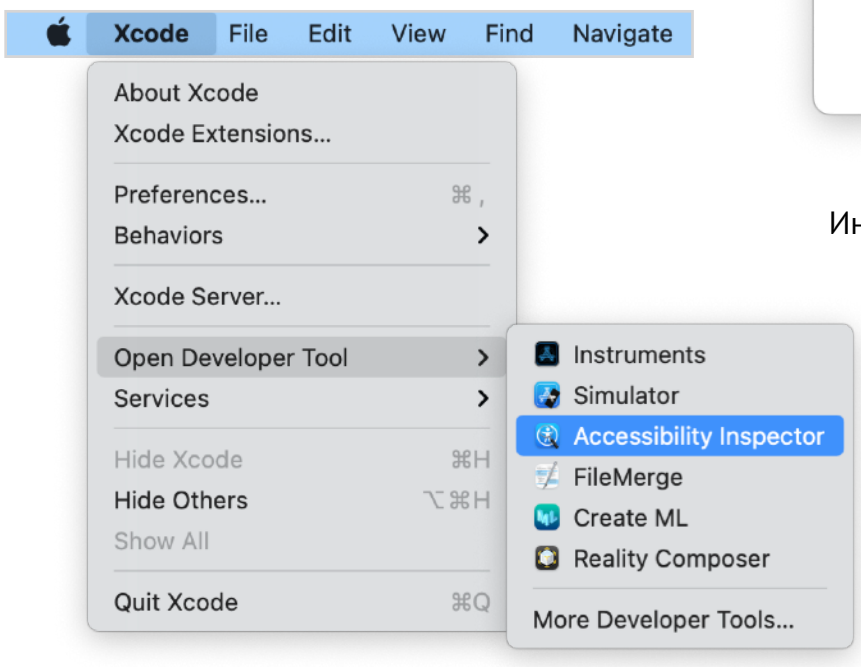


В симуляторе через Accessibility Inspector

В симуляторе размер текста меняется через Accessibility Inspector. Включается он через меню Xcode → Open Developer Tool →Accessibility Inspector. В открывшемся приложении нужно выбрать симулятор или телефон, а на третьей вкладке находится выбора размера.



Интерфейс Accessibility Inspector дает доступ к основным настройкам доступности



Запустить инспектор можно через меню Xcode

Базовая гигиена

Главное, что нужно сделать для поддержки Dynamic Type, – отказаться от фиксированной высоты всех контролов. Основным инструментом расчёта является intrinsicContentSize — размер, который рассчитывает сам элемент. Обычно ширина не указывается, считается только высота.

```
public override var intrinsicContentSize: CGSize {
    CGSize(width: UIView.noIntrinsicMetric,
           height: 48)
}
```

Цифру нужно заменить на вычисление :-)

Надписи

Все надписи должны поддерживать многострочность и расти по высоте. По-умолчанию надписи поддерживают лишь одну строку. Чтобы указать, что строк может быть сколько угодно, установите количество в 0.

```
titleLabel.numberOfLines = 0
```

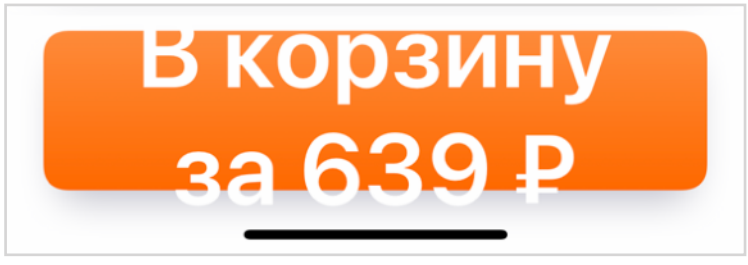
Кнопки

У стандартных кнопок сразу несколько проблем:

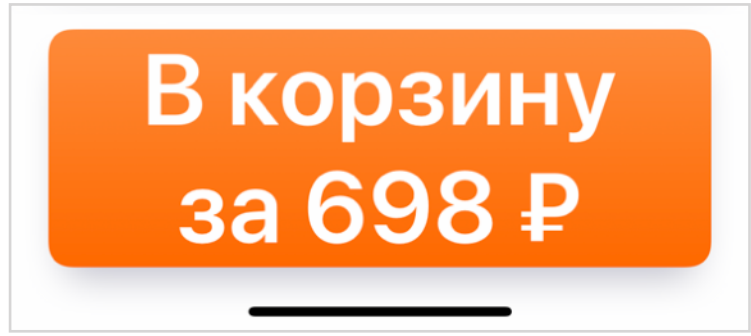
— titleLabel не привязан констрейнтами к размеру кнопки, с увеличением размера шрифт начнёт вылезать за кнопку. Добавить связь можно через констрейнты.

```
extension UIButton {
    func pinTitleToBounds() {
        guard let titleLabel = titleLabel else { return }
        translatesAutoresizingMaskIntoConstraints = false
        NSLayoutConstraint.activate([
            topAnchor.constraint(equalTo:
                                 titleLabel.topAnchor),
            bottomAnchor.constraint(equalTo:
                                    titleLabel.bottomAnchor)
        ])}}
```

После этого кнопка станет «резиниться» нормально. В идеале высота должна быть привязана к `layoutMargins`. Для поддержки RTL-языков лучше задавать `directionalLayoutMargins`. Увы, решение не подходит для кнопок с иконками;



До



После

— titleLabel изначально рассчитан на одну строку и выровнен по левому краю, придётся компенсировать вручную.

```
button.titleLabel?.numberOfLines = 0
button.titleLabel?.textAlignment = .center
```

Ячейки

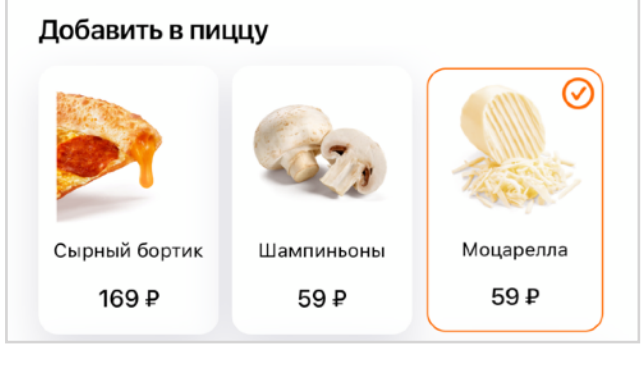
Ячейки сами должны считать свою высоту. Если используете констрейнты, то авторасчёта высоты будет достаточно.

```
tableView.rowHeight = UITableView.automaticDimension
tableView.estimatedRowHeight = 56
```

Относительные значения

Некоторые параметры, например, cornerRadius и borderWidth, стоит пересчитывать в соотвествии с увеличением размера шрифта.

```
var borderWidth: CGFloat {
    UIFontMetrics.default.scaledValue(for: 1)
}
```



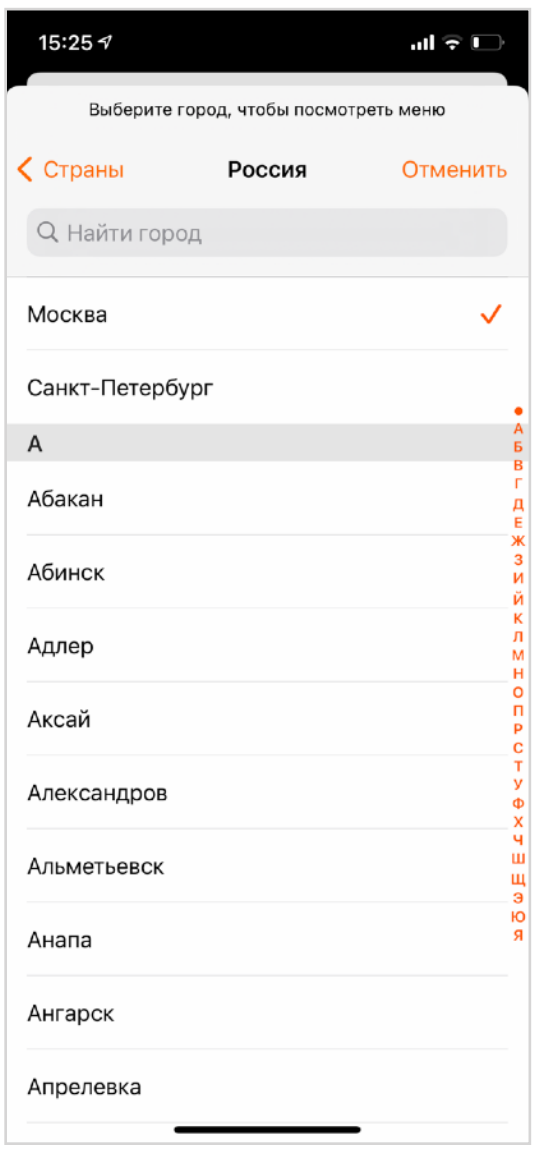
Большому тексту толстая рамка

Простой пример

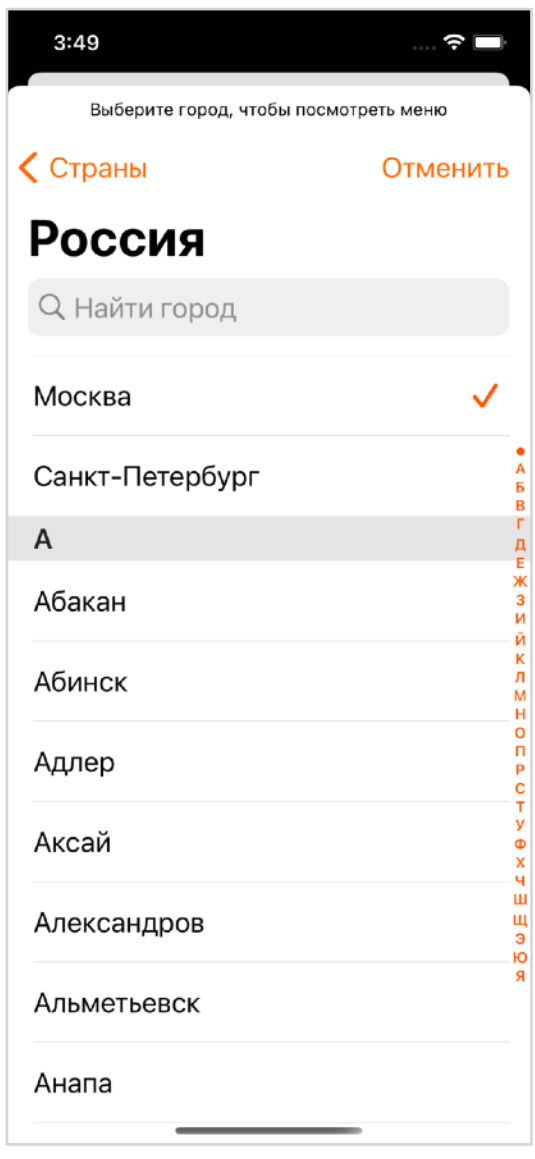
Посмотрим, как меняется интерфейс от размер текста на экранах города и страны.

Экран выбора города целиком собран из стандартных контролов, а они все умеют работать с динамичными размером. Для ячеек я добавил автоматический расчёт высоты: так размер текста будет «давить» на ячейку изнутри, а размер ячейки считаться автоматически. То же самое касается хедеров секций с буквой.

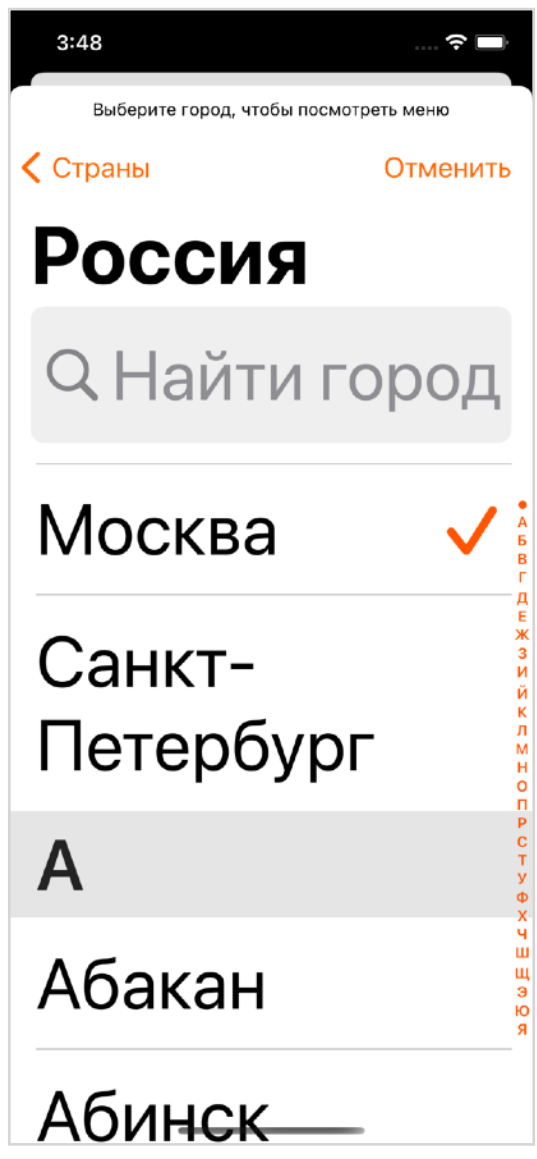
```
tableView.rowHeight = UITableView.automaticDimension
tableView.estimatedRowHeight = 56
```



Стандартный размер

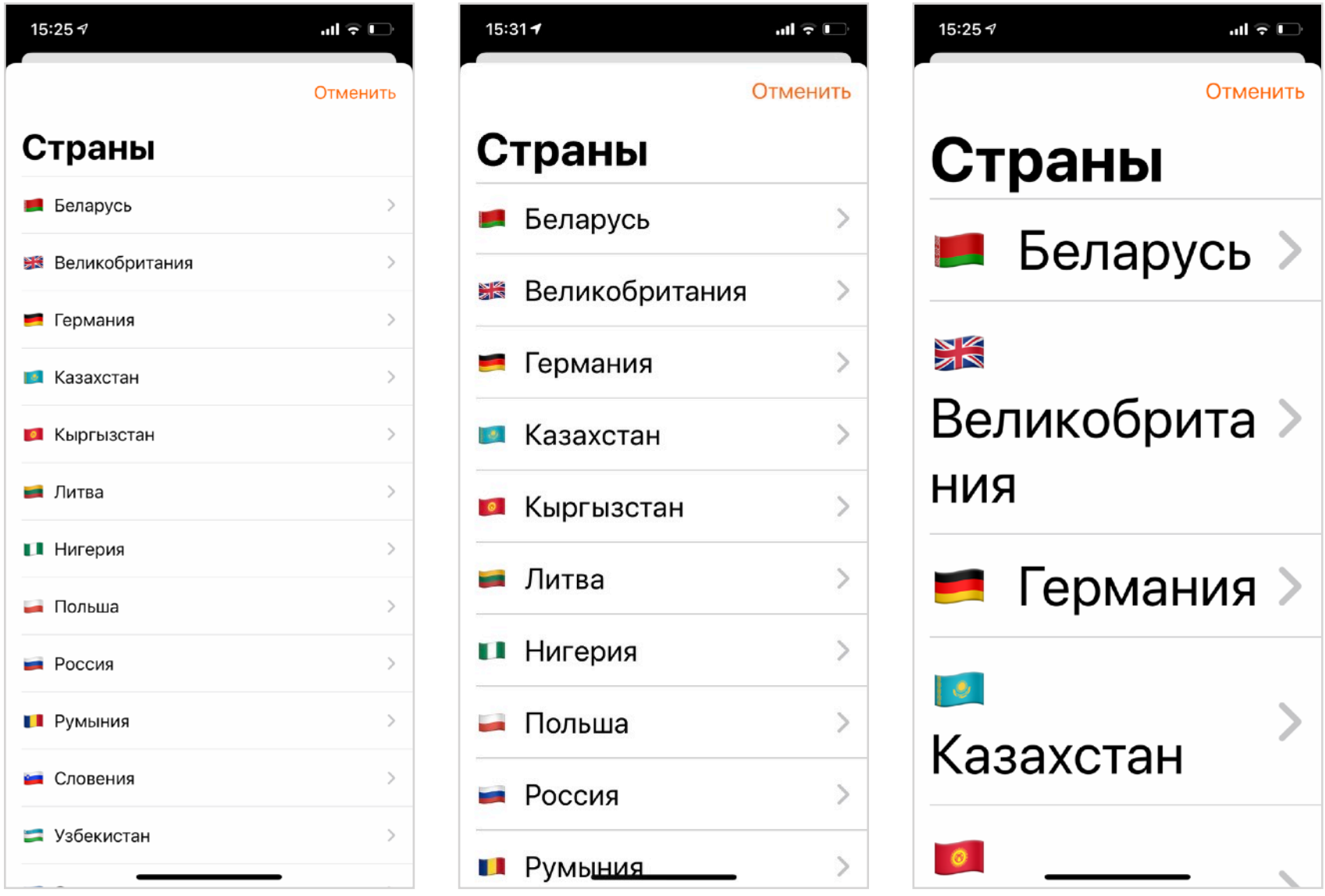


Увеличенный



Максимальный

При этом `UILabel` поддерживает несколько строк, чтобы не обрезать текст.



Такая же история и со списком стран, разве что в самом большом размере флаги отвалились от названия. Есть несколько вариантов решения:

— совсем скрывать флаги в большом размере, но они важны так же, как и в маленьком;
— все флаги поставить на отдельную строку, чтобы была рифма;
— ничего не делать, это не критично.

Ничего не делать тоже вариант: мне не придётся усложнять код и в это время смогу заняться проблемами доступности посерьёзнее.

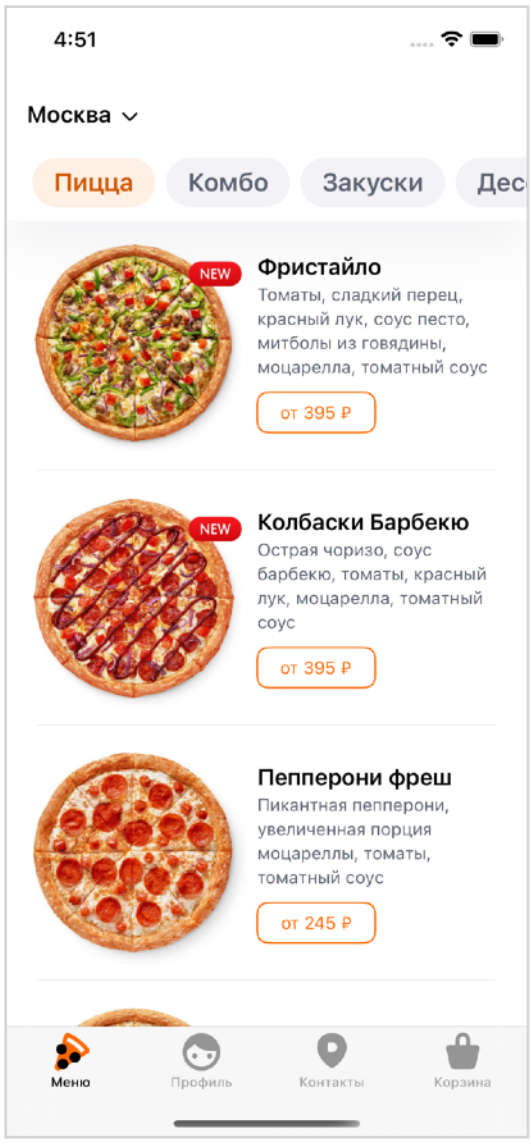
Подытожим ключевые моменты:

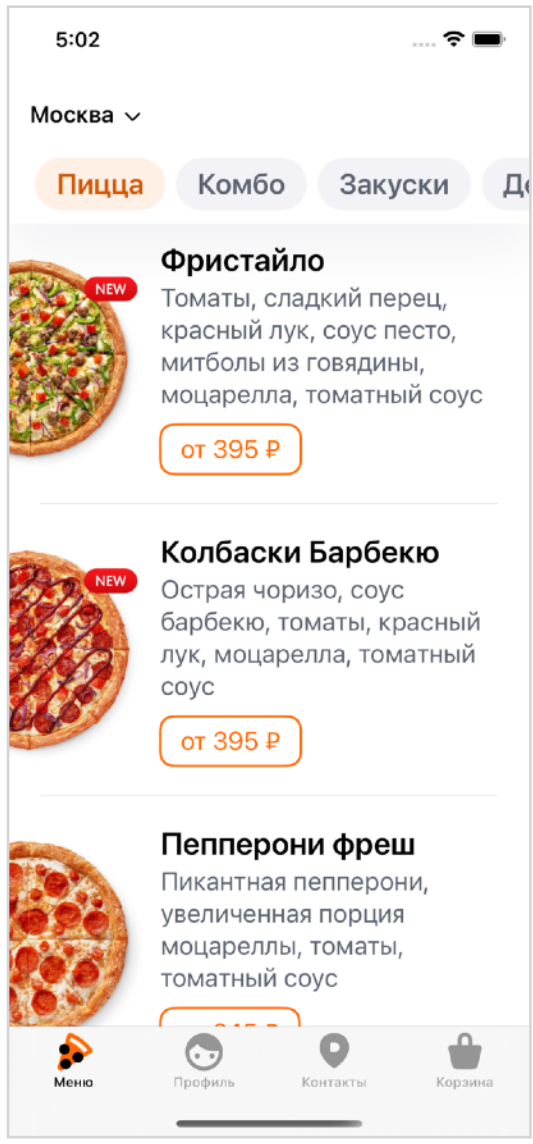
— тексту нужно поставить динамичный шрифт,
— текст может быть в несколько строк,
— все компоненты на экране должны сами расчитывать свой размер.

Dynamic type

Меняем вёрстку

В интерфейсах многие элементы расположены горизонтально, например, картинка слева от текста в меню. Увеличиваясь, текст требует больше места. Можно было бы изображение пицц сдвигать влево, но это не подойдёт для картинок напитков, поэтому вёрстку мы меняем только для «доступных» размеров шрифта.





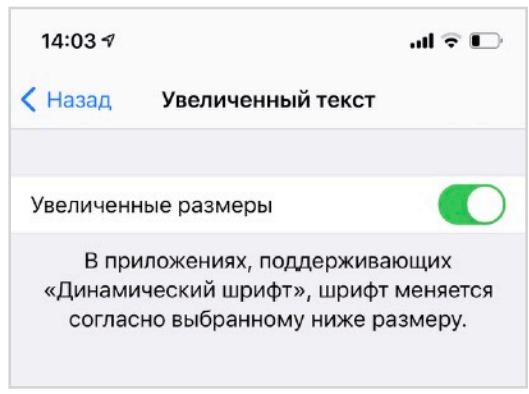
В самом большом размере нужно кардинально менять вёрстку. Совсем отказаться от картинки мы не можем, поэтому ставим её над текстом и сделаем как можно больше. Текст тоже займёт всю ширину. Вместе с ним и кнопку стоит растянуть до размеров экрана.

Сделать такую адаптацию можно двумя путями.

StackView

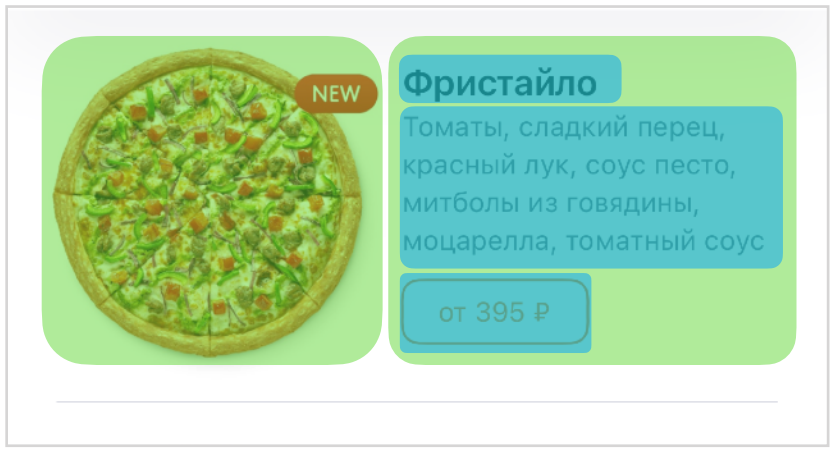
Если сверстать ячейку с помощью `UIStackView`, то сможем легко менять её ориентацию с горизонтально на вертикальную с помощью свойства `axis`.

Ориентацию стоит привязать к свойству `isAccessibilityCategory`, тогда вы сможете поменять весь интерфейс, если пользователь выбрал шрифт большого размера. Завязываться на размер каждой отдельной надписи не стоит, ведь тогда соседние элементы могут иметь разный лейаут.

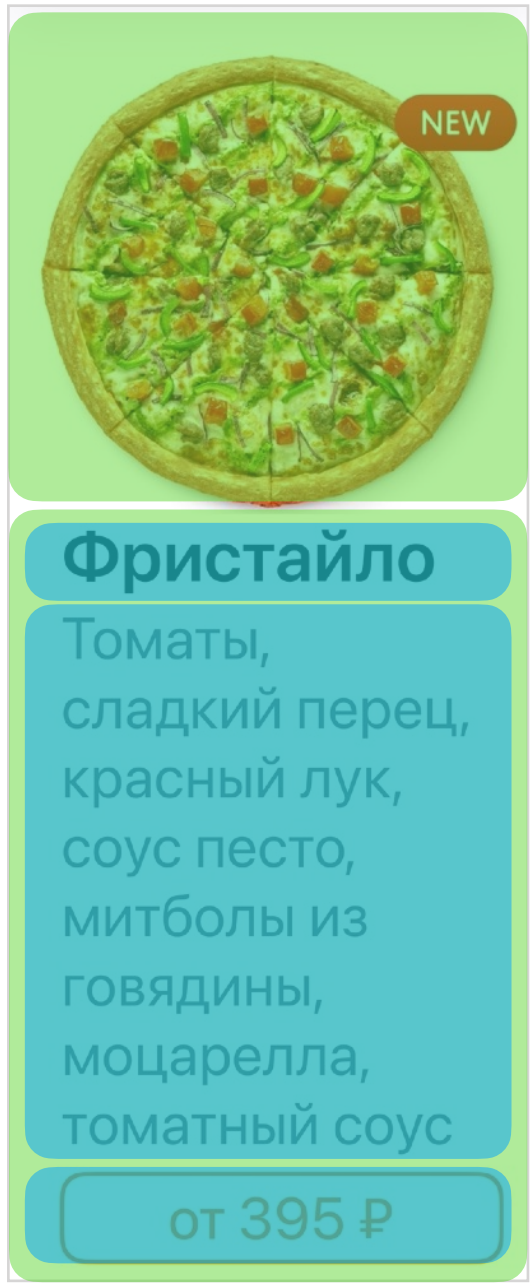


Этот переключатель дает доступ к большим размерам

Связав `axis` и `isAccessibilityCategory`, вы получите сильный инструмент динамичной вёрстки, можно будет менять ориентацию в зависимости от трейта. Так мы можем сделать стек из картинки и вьюшки со всем текстом, а при увеличении трейта менять его ось на вертикальную.



Горизонтальное расположение UIStackView



Ячейка меню свёрстана из двух stackview: один горизонтальный, состоит из картинки, а вот второмвертикально расположен текст. При увеличении размера у первого мы меняем ориентацию, а у второго — выравнивание. Посмотрите, как на примерах кнопка с ценой выросла до ширины экрана, потому что `alignment` сменился с `.leading` на `.fill`.

Можно было бы завести и два разных вида ячеек, но мы этим ни разу не пользовались, использовать динамичные стаквью проще.

Для того чтобы не писать обработку трейтов каждый раз, мы завели класс `AccessibleStackView`. Ему можно задать предпочитаемую ось и выравнивание для разных размеров, а дальше он сам справится.

Смена оси при изменении размера текста будет выглядеть так:

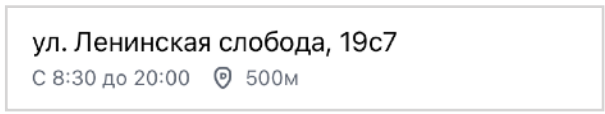
```
override func traitCollectionDidChange(
    _ previousTraitCollection: UITraitCollection?
) {
    super.traitCollectionDidChange(previousTraitCollection)

    if traitCollection
          .preferredContentSizeCategory
          .isAccessibilityCategory
    {
        axis = .horizontal
    } else {
        axis = .vertical
    }
}
```

Dynamic type

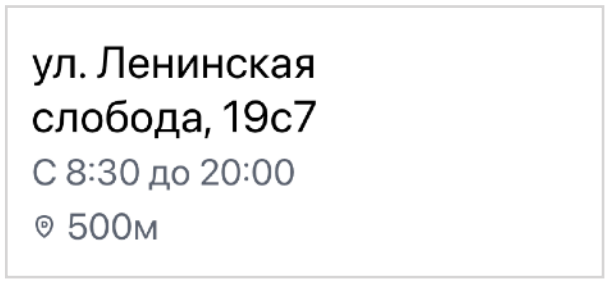
Скриншот-тесты

Работа с динамичным размером контролов несёт свои сложности: сначала нужно проверять, как контролы ведут себя в разных размерах, а потом не ломать их поведение. Решить обе проблемы помогут скриншот-тесты.

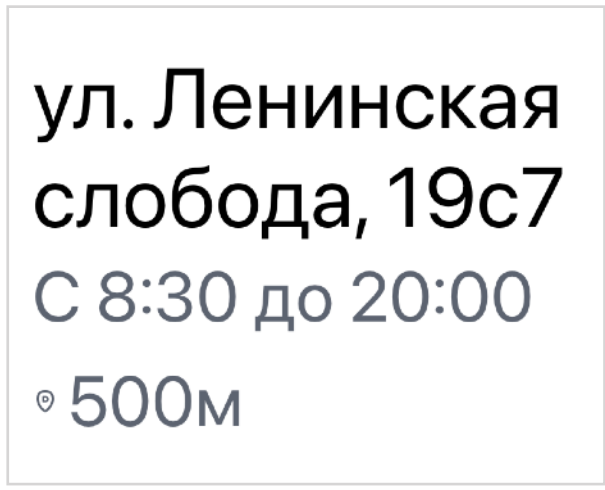
С помощью скриншот-тестов мы можем написать тест, который создаст ячейку, отрисует её в нескольких размерах, сгенерирует картинку и положит её в репозиторий проекта. При следующем прогоне он создаст новую картинку, сравнит её с предыдущей и, если картинки не совпадут, то тест упадёт и вы сможете увидеть, в чём была разница.



Стандартный размер



Увеличенный



Максимальный

Для тестирования я использую библиотеку SnapshotTesting от Point-Free, мне она кажется самой простой в установке и работе.

Скриншот-тест состоит из нескольких этапов:

— создаём ячейку с помощью вспомогательного метода (реализация внизу);
— наполняем ячейку данными, которые хотим отобразить;
— делаем скриншоты с помощью метода `assertSnapshot`, указываем размер и trait.

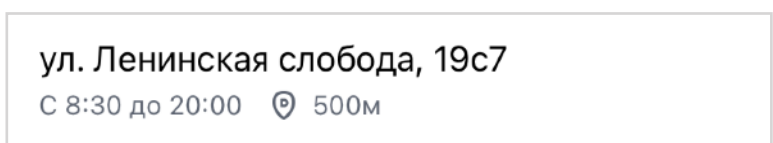
После этого у вас сгенерируется нужное количество файлов и вы сможете посмотреть все состояния ячейки, которые вам интересны.

С помощью скриншот-тестов можно проверять и другие состояния UI: отключённость, выбранность или вёрстку данных разной длины.

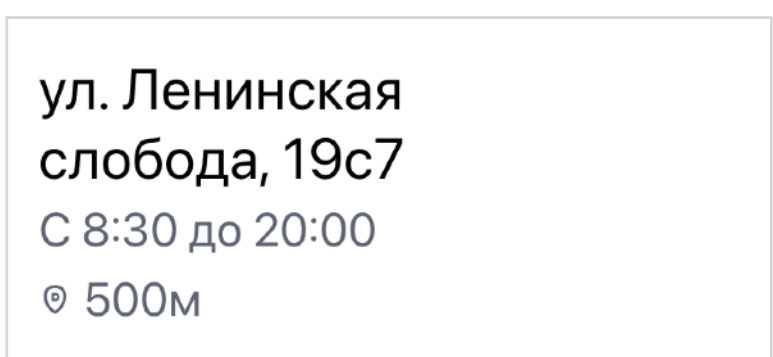
```swift
import SnapshotTesting

class PizzeriaInCityTableViewCellTests: XCTestCase {
    var cell: PizzeriaInCityTableViewCell!

    override func setUp() {
        cell = cellFromNib()

        let viewModel = PizzeriaInCityViewModel(
            address: "ул. Ленинская слобода, 19с7",
            schedule: .init(today: "С 8:30 до 20:00"),
            distance: "500м")

        cell.setup(viewModel: viewModel)
    }
    override func tearDown() {
        cell = nil
    }
    func test_accessibleMediumSize() {
        assertSnapshot(
            matching: cell,
            as: .image(size: .init(width: 375, height: 170),
            traits: .init(preferredContentSizeCategory:
                          .accessibilityMedium)))
    }
    func test_greatestFontSize() {
        assertSnapshot(
            matching: cell,
            as: .image(size: .init(width: 375, height: 300),
            traits: .init(preferredContentSizeCategory:
                          .accessibilityExtraExtraExtraLarge)))
    }
    /// Вспомогательный метод для загрузки ячеек из .xib
    func cellFromNib<View: UIView>() -> View {
        let bundle = Bundle(for: View.self)
        let nib = UINib(nibName: String(describing: View.self),
                        bundle: bundle)
        return nib.instantiate(withOwner: nil,
                               options: nil).first as! View
    }
}
```

Пример двух тестов для тестирования Dynamic Type

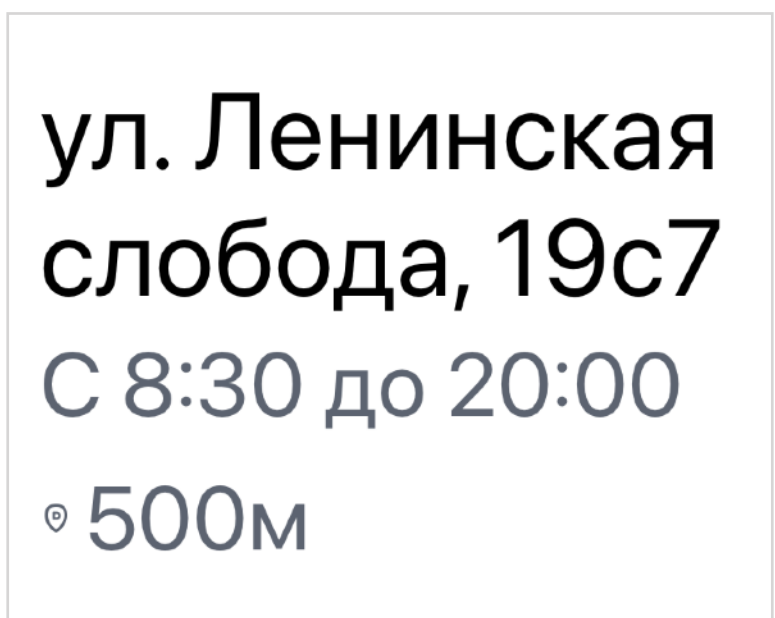
Констрейнты



Стандартный размер



Увеличенный



Максимальный

В предыдущем примере мы увидели, что с увеличением размера иконка не увеличивается. Нужно согласовать иконку и надпись. Один из вариантов — сделать SFSymbol, и тогда он начнёт согласовываться с текстом. Если вы поддерживаете версии старше, чем iOS 13, то нужно другое решение.

Если мы знаем, что текст будет всегда в одну строчку (а это очень сильное допущение для Dynamic Type), то мы можем сказать, что высота иконки должна совпадать с высотой всего лейбла. Идеальное решение выровняет высоту иконки лишь с высотой первой строки, а не всей надписи.

Можно использовать подкласс констрейнты, чтобы считать размер пропорционально стилю текста автоматически по оповещению `UIContentSizeCategory.didChangeNotification`. Тогда нам достаточно указать класс для констрейнта и он станет динамической.

Dynamic type

Стильно, но в одном размере

Если вы хотите применить стандартные стили, но не хотите «резинить» интерфейс, то размер шрифта можно зафиксировать. Причём можно как полностью отключить динамичность, так и лишь ограничить максимальный размер.

Полное отключение размеров.

— создаём трейт стандартного размера `.large`;

— создаём `UIFont` с заданным стилем и трейтом

```
extension UIFont {
    static func preferredFont_fixed(
        for textStyle: UIFont.TextStyle
    ) -> UIFont {
        let traitCollection = UITraitCollection(
                        preferredContentSizeCategory: .default)

        return UIFont.preferredFont(
            forTextStyle: textStyle,
            compatibleWith: traitCollection)
    }
}

extension UIContentSizeCategory {
    static var `default`: UIContentSizeCategory = .large
}
```

Изменение размера «на лету» при этом **нужно отключить**, иначе всё сбросится при первом изменении трейта.

```
fixed.font = .preferredFont_fixed(forTextStyle: .body)
fixed.adjustsFontForContentSizeCategory = false
```

Можно не полностью запрещать изменять размер шрифта, а лишь ограничить максимальный размер. Этот тот же код, но с дополнительной проверкой на доступные размеры шрифта.

```
public static func preferredFont_limited(
    forTextStyle textStyle: UIFont.TextStyle,
    by contentSize: UIContentSizeCategory = .latestNonAccessible
) -> UIFont {
    if UIContentSizeCategory.current > contentSize {
        let traitCollection = UITraitCollection(
             preferredContentSizeCategory: contentSize)

        return UIFont.preferredFont(
                            forTextStyle: textStyle,
                            compatibleWith: traitCollection)
    }
    return UIFont.preferredFont(forTextStyle: textStyle)
}
```

Использование

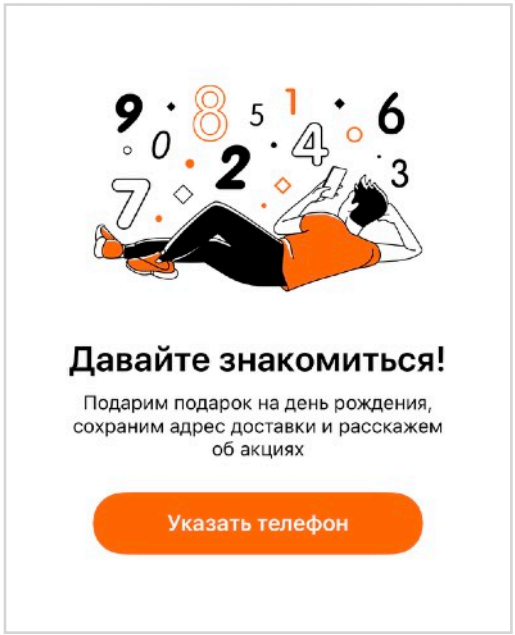
Обе функции отличаются лишь суффиксом. При разработке вы можете решить, насколько готовы заниматься динамичным размером прямо сейчас, и если времени нет, то использовать стандартный стиль, но ограничить его полностью или только запретить самые большие размеры.

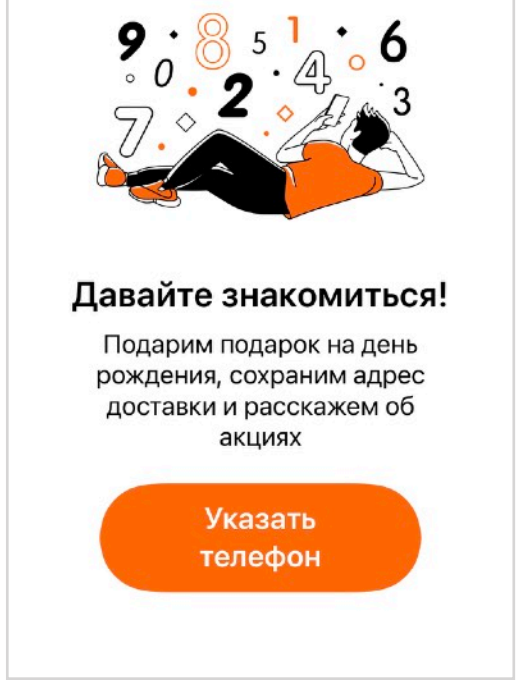
```
dynamic.font = .preferredFont(forTextStyle: .body)
fixed. .font = .preferredFont_fixed(forTextStyle: .body)
limited.font = .preferredFont_limited(forTextStyle: .body)
```

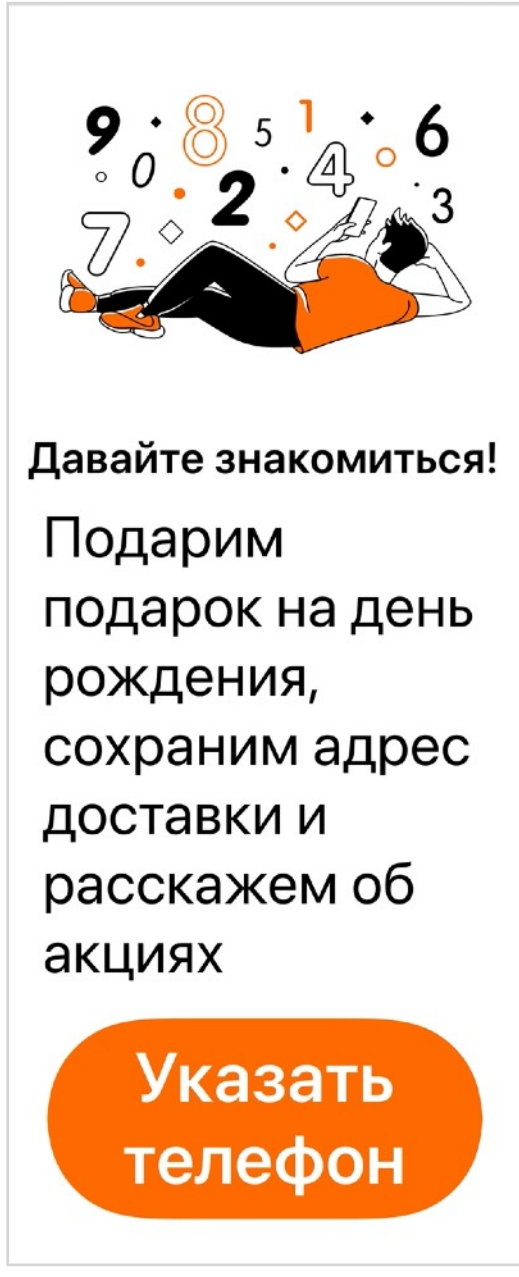
Dynamic type

Скролл всего

Раз размер может увеличиться у любого экрана, то он, скорее всего, перестанет помещаться в размеры экрана. Например, простой экран с предложением ввести номер телефона может увеличиться в два раза.







Добавлять `UIScrollView` на каждый экран неудобно, нужно решение проще. Хочется, чтобы мы могли работать над интерфейсом как обычно, чтобы не было разницы, находится он в скролле или нет.

Хорошим решением будет контроллер-обёртка, который обернёт дочерний контроллер в `UIScrollView`. Например, https://github.com/drewolbrich/ScrollingContentViewController. Для использования достаточно унаследоваться от `ScrollingContentViewController`, но есть и другие варианты интеграции. Подобнее в документации.

```
class MyViewController: ScrollingContentViewController {}
```

Есть только одно ограничение для дочернего контроллера: нужно поставить констрейнты со всех сторон, чтобы размер контента определялся однозначно. Обычно мне достаточно поставить констрейнт снизу, чтобы он был ⩾ 0.

В итоге, если контент меньше экрана, то скролла нет. Но если экран увеличился и содержимое уже не помещается, то появится скролл и можно будет увидеть всё. Такое поведение полезно и для маленьких устройств, и для локализации, ведь текст может кратно вырасти на одном из языков и не поместиться в экран.

`ScrollingContentViewController` берёт на себя и менеджмент клавиатуры, ведь теперь он может просто сужать видимую область при появлении клавиатуры.

Выравнивание текста

На примере выше при увеличении размера я меняю и выравнивание текста. Вся работа происходит в `traitCollectionDidChange`.

```
override func traitCollectionDidChange(
    _ previousTraitCollection: UITraitCollection?) {
    super.traitCollectionDidChange(previousTraitCollection)

    let isAccessibleSize = traitCollection
        .preferredContentSizeCategory
        .isAccessibilityCategory

    if isAccessibleSize {
        descriptionLabel.preferredMaxLayoutWidth = 0
        descriptionLabel.textAlignment = .left
    } else {
        descriptionLabel.preferredMaxLayoutWidth
                                = headerLabel.frame.width
        descriptionLabel.textAlignment = .center
    }
}
```

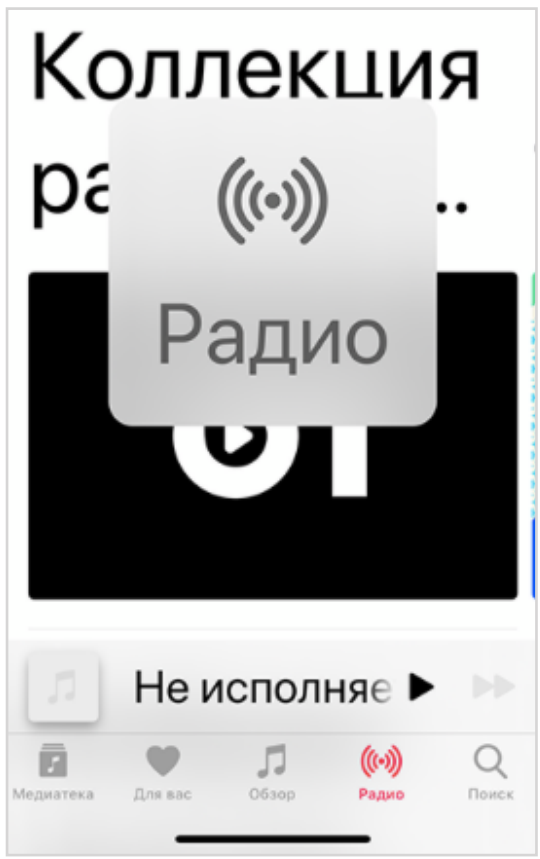
Ширина основного текста интересно задается через `preferredMaxLayoutWidth`: основная часть должна быть согласована по длине с заголовком, поэтому текст хоть и ограничен марджинами экрана, но дополнительно сужается по заголовку.

Dynamic type

Превью

Бывает, что для каких-то элементов совершенно не получается придумать альтернативное поведение и они остаются маленькими. Например, табы в стандартной навигации.

Для таких случаев Apple сделала так, чтобы можно было зажать, а ее иконка и название появились крупно.



Для кнопки «закрыть» и «энергетической ценности» мы добавили превью

Добавляется это в 3 строчки:

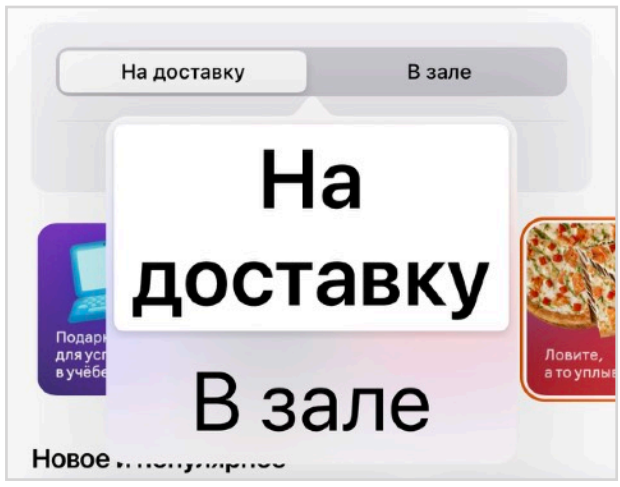
```
showsLargeContentViewer = true
addInteraction(UILargeContentViewerInteraction())
largeContentTitle = nutritionButton.accessibilityLabel
```

Про дополнительные возможности API можно прочитать в статье Алексея Берёзки iOS 13 под лупой.

Другое поведение

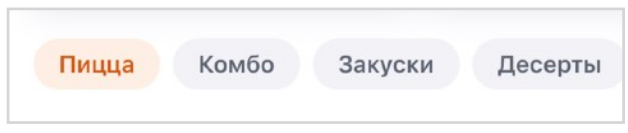
Некоторые контролы сложно как-то разумно увеличить в размере, поэтому они могут менять свое поведение.

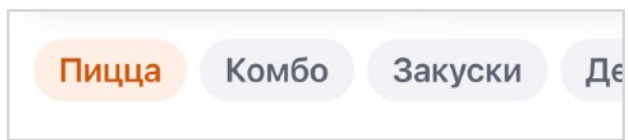
Например, `UISegmentedControl` — это набор горизонтальных надписей. Если развернуть их и поставить в вертикальную колонку, то она займёт много места. Поэтому,при настроенном «доступном» размере шрифта можно не только нажать на контрол, но и удерживать палец на нём, чтобы вызвать крупное всплывающее окно с вариантами.

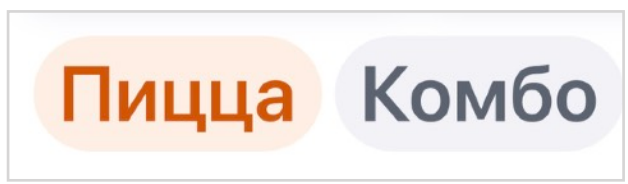


Всплывающее окно для UISegmentedControl

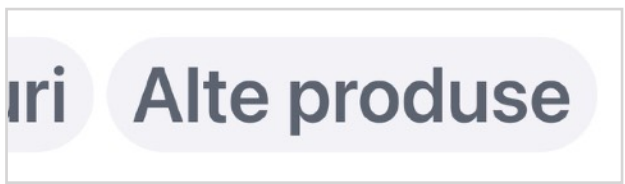
Многие горизонтальные элементы не выдерживают увеличения. Например, шрифт в карусели выбора типа продуктов может увеличиваться, но им неудобно пользоваться: текст не может переместиться на две строки, а значит начнёт вылазить за пределы экрана на некоторых языках.





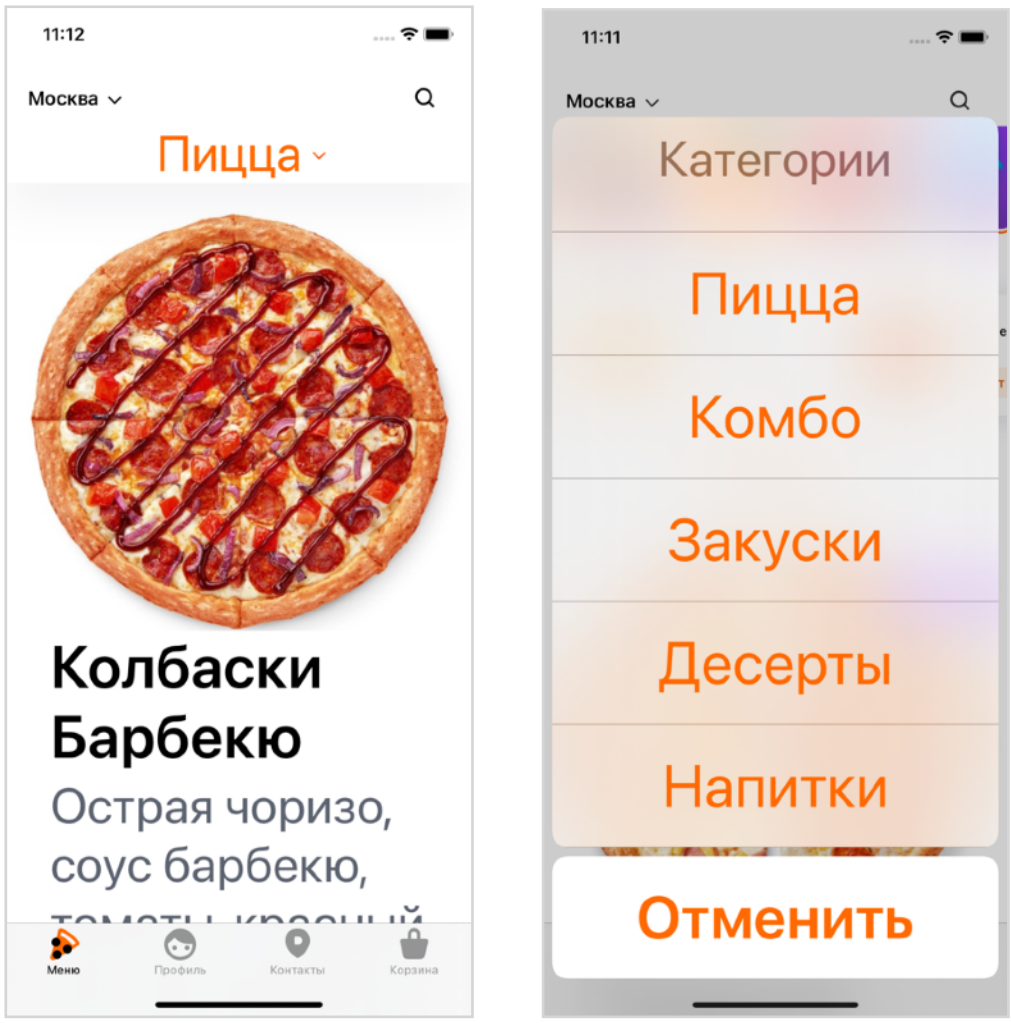


Текст в карусели динамичный



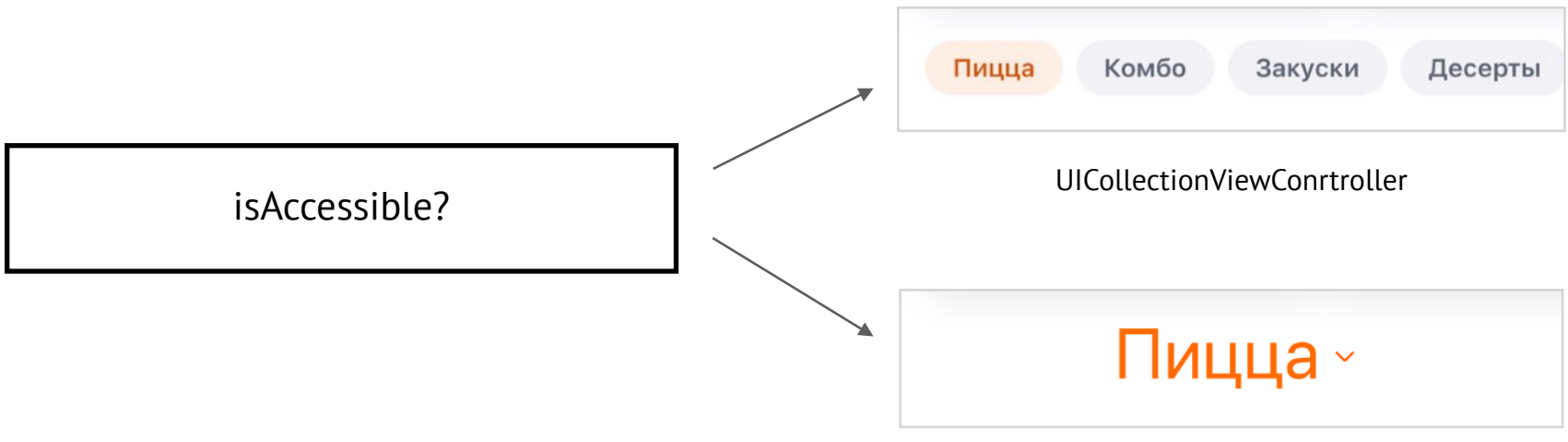
«Другие продукты» на чешском

Что можно сделать в таком случае? Например, вместо горизонтальной карусели нарисовать лишь одну кнопку с текущим типом продуктов. Одиночная кнопка может быть любого размера, даже двустрочной. При нажатии на неё можно показать стандартный `UIAlertController`, он поддерживает Dynamic Type.



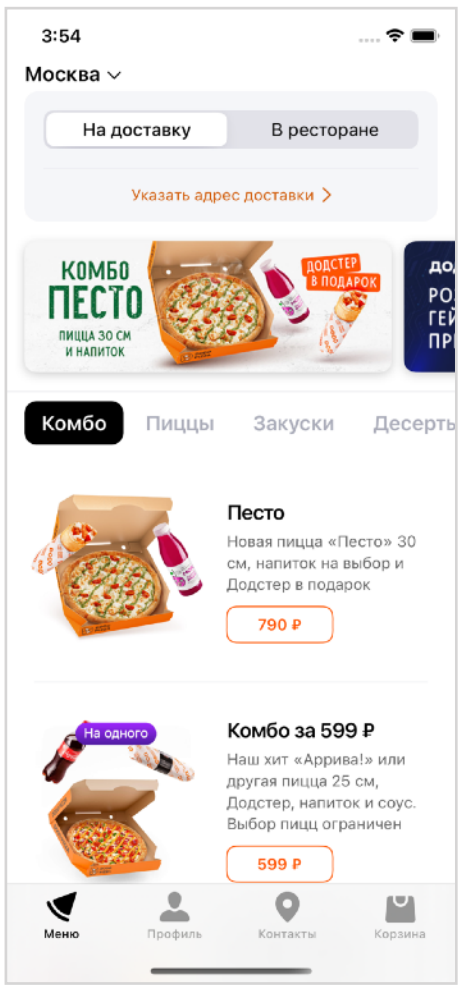
Переключаться между разными типами элементов удобно на уровне `UIViewController`, тогда у каждого контрола будет полная свобода и если он захочет показать поверх себя другой контроллер, то покажет.

В итоге, мы сделали два контроллера: один для горизонтальной коллекции, это `UICollectionViewConrtoller`, а другой с кнопкой. Переключаемся между ними с помощью другого стейт-контроллера, который выбирает, какой из двух контроллеров показать.

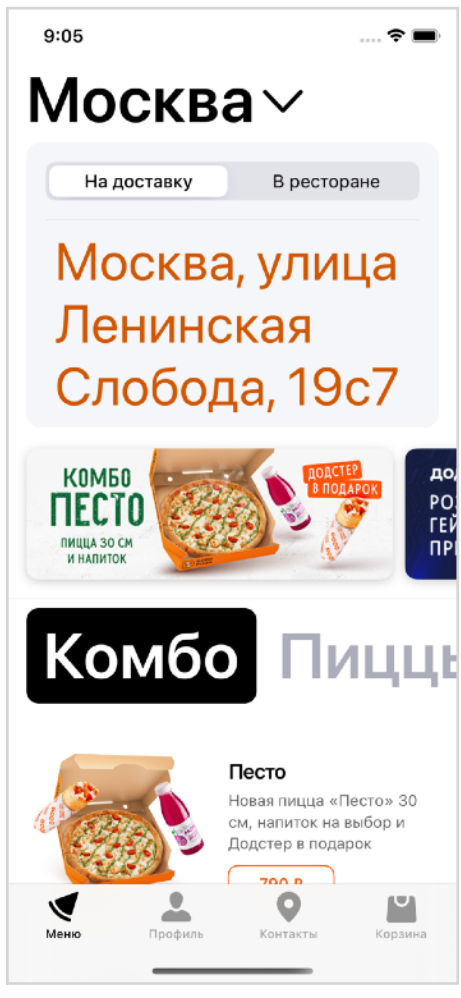


Все ли нужно увеличивать?

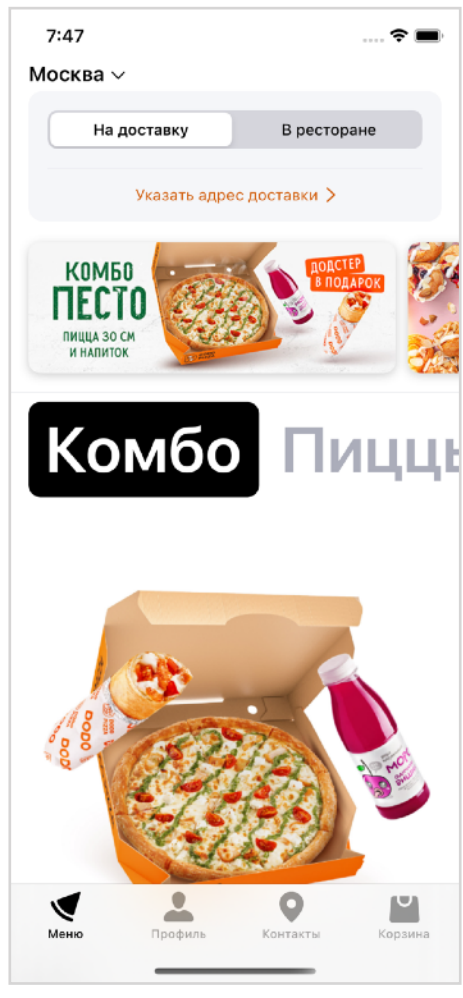
Если бездумно увеличивать всё подряд, то можно получить состояние, когда контент, за которым человек пришёл, уходит с экрана, что может сбивать с толку. На примере ниже это второй скриншот: мы увеличили каждый контрол, в результате пицца почти ушла с экрана и уже может быть неочевидно, что можно проскроллить.



Стандартный размер



Увеличено все что можно



Увеличили только ячейки

Можно увеличивать только ключевой контент, а всё остальное оставить на увеличение через превью. Можно заредизайнить экран, например, унести кнопку выбора города на экран редактирования адреса — так мы освободим место.

Мне кажется, что люди с эктремально большим размером шрифта привыкли к тому, что каждый экран надо проскроллить, и большой проблемы в этом нет — можно увеличивать всё подряд, удобство здесь важнее красоты. Но если получается сделать и красиво — супер!

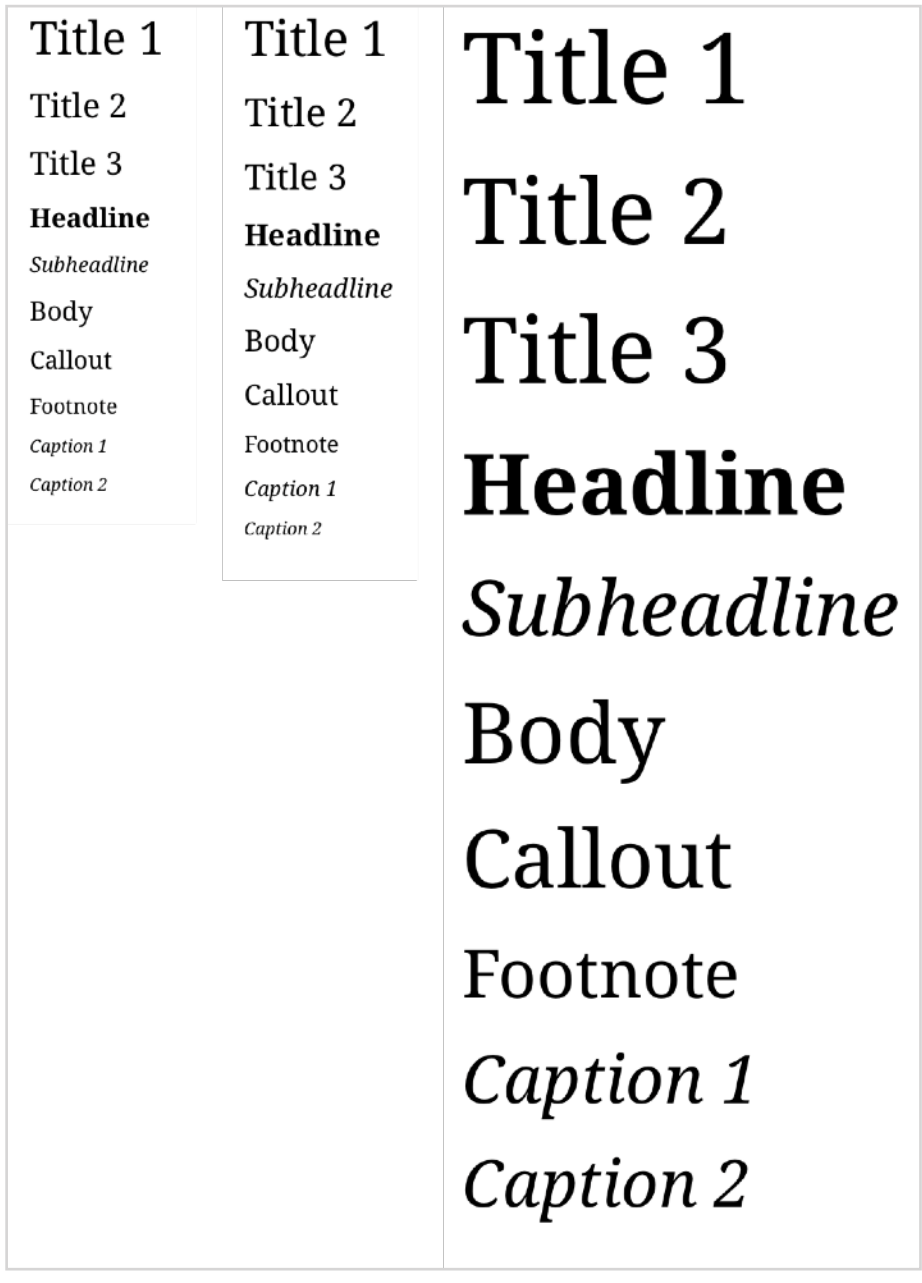
Dynamic type

Стиль для шрифта

Применить стиль можно к любому шрифту, он унаследует параметры стиля и будет меняться соответствующе.

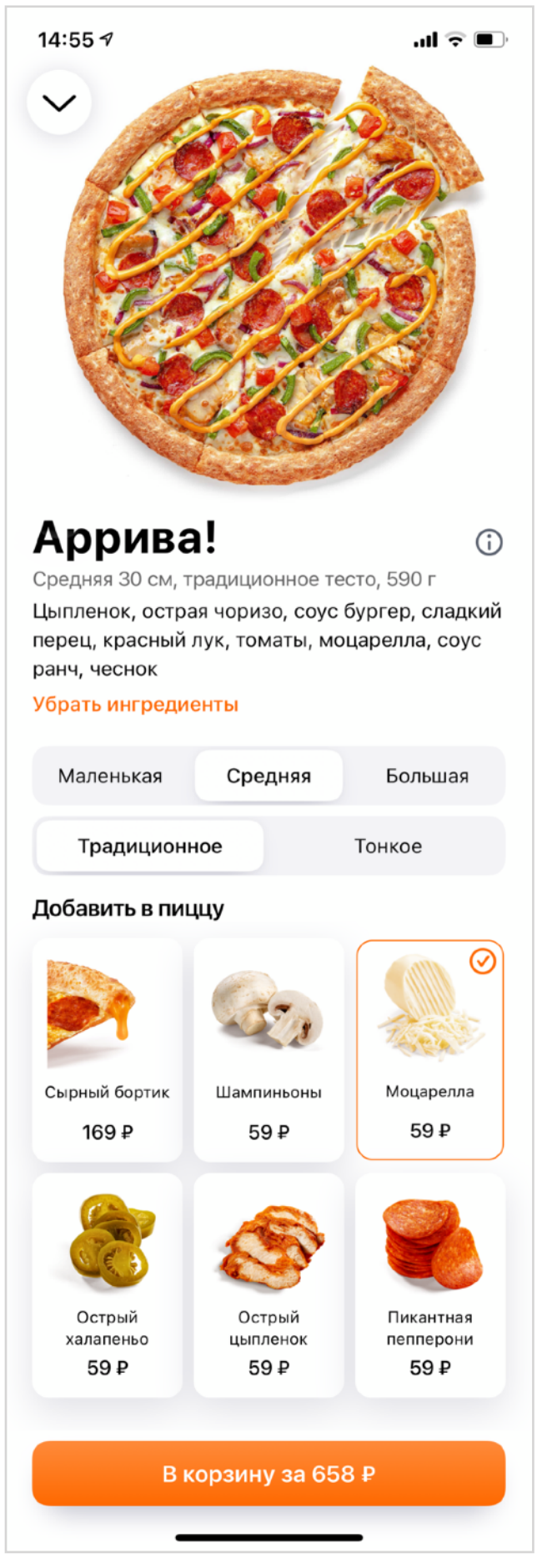
```
let customFont = UIFont(name: "CustomFont-Light",
                        size: UIFont.labelFontSize)!

label.font = UIFontMetrics(forTextStyle: .headline)
              .scaledFont(for: customFont)
```

Подробнее можно прочитать в документации Apple или на useyourloaf.



Пример со шрифтом NonoSans от useyourloaf.

Пример



Попробуем разобрать увеличение интерфейса для целого экрана, для примера возьмём карточку с пиццей. В карточке есть изображение пиццы, её описание, настройка размера и типа теста, добавки и кнопка «Купить». Попробуем увеличить каждый из размеров.

Можно ли увеличить **изображение**, которое и так в полную ширину экрана? Можно, ведь важна не столько форма, сколько начинка пиццы. Или можно временно увеличивать картинку, пока пользователь удерживает палец на ней.

Круглые кнопки увеличили иконки внутри себя, потому что включил свойство adjustsImageSizeForAccessibilityContentSizeCategory.

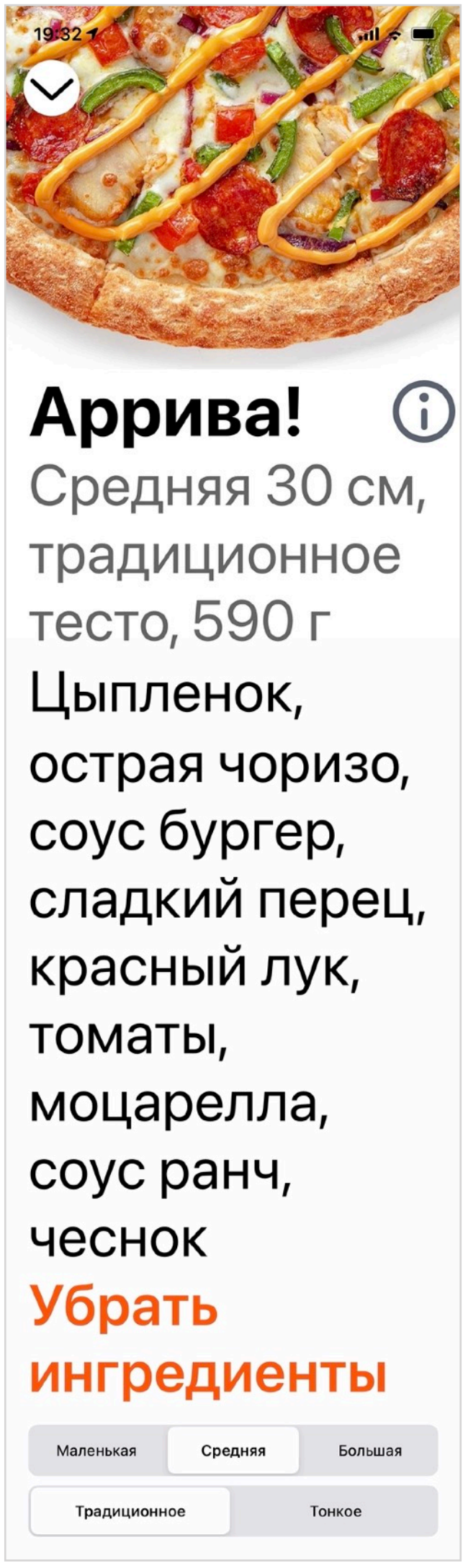
С текстом всё просто: заменяем его на стили, размер начнёт увеличиваться.

Сегмент контролы не меняем, стандартные дают возможность показать всплывающее окно с выбором размера.

Добавки нужно увеличить до одной ячейки в ряду. Вместо галочки можно покрасить фон, но при этом мы ещё обозначаем выбранность и рамкой, которая становится толще с увеличением размера.

Кнопка корзины сама рассчитывает свой размер, потому что мы ей задали contentInsets, а не жёсткие ограничения на высоту и ширину.

В итоге карточка увеличивается в длину почти в 3 раза, но всё содержимое можно прочитать.



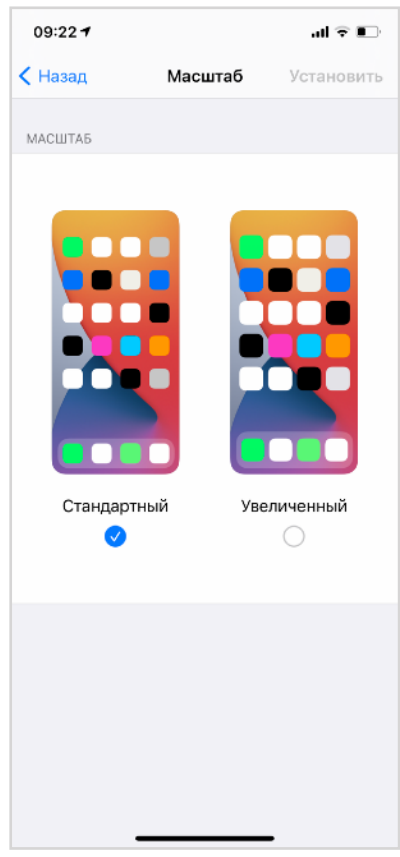
Dynamic type

Масштаб и лупа

В айфонах есть пара возможностей, которые помогают людям с плохим зрением даже тогда, когда разработчики не поддерживают Dynamic Type в приложении.

Масштаб

При первой настройке телефон спрашивает, какой масштаб предпочтительней: стандартный или увеличенный. К сожалению, на этом экране телефон не даёт выбрать размер размер шрифта, поэтому многие, кто хотел бы использовать увеличенный размер, просто не знают о настройках Dynamic Type. Изменить масштаб можно и позже: Настройки → Экран и яркость → Вид. Обратите внимание, что настройка находится не в разделе доступности.



Работает так же, как и настройка масштаба на макбук: вы выбираете scale, с которым картинка будет рендериться, а всё остальное рассчитывается автоматически. Поменяется размер одной точки при расчётах картинки, но мы верстаем интерфейс в пойнтах, поэтому изменение происходит незаметно и вам ничего не нужно делать дополнительно в коде. Настройка влияет на параметр UIScreen.main.scale. Понять, включено ли на телефоне увеличение, можно, если сравнить два вида скейла.

```
var isZoomed: Bool {
    return UIScreen.main.scale != UIScreen.main.nativeScale
}
```

Масштаб работает, только если существует модель телефона с меньшим размером экрана, но с такими же пропорциями. По этой причине масштабирование было недоступно на iPhone X до тех пор, пока не вышел iPhone Mini.

На свойство `isZoomed` не нужно завязывать логику UI, можно только использовать для аналитики, чтобы понять, как у вас много людей, которые хотят интерфейс покрупнее.

Жирный шрифт

Среди наших клиентов 4% включают жирный шрифт (у Immoweb 6.3%). Чтобы учесть предпочтения пользователя, надо где-то в настройках вашей дизайн-системы добавить проверку и сделать шрифт пожирнее.

```
extension UIFont {
    func accessibleBoldEnough(of size: CGFloat) -> UIFont {
        UIFont(descriptor: accessibleFontDescriptor, size: size)
    }

    var accessibleFontDescriptor: UIFontDescriptor {
        if UIAccessibility.isBoldTextEnabled {
            return fontDescriptor.withSymbolicTraits(.traitBold)!
        } else {
            return fontDescriptor
        }
    }
}
```

Другие настройки

UIAccessibility содержит полтора десятка других свойств, которые необходимо учитывать для доступности интерфейсов. Так вы сможете помочь людям, у которых автоматическое включение видео может вызвать эпилептический припадок или их укачивает, кто с трудом воспринимает неконтрастный цвет или не различает цвета из-за дальтонизма и поэтому хочет, чтобы разные состояния отличались не только цветом, но и формой. Для каждого свойства есть аналогичное оповещение, которое вызывается при смене настройки. Многие настройки достаточно учесть всего однажды на уровне дизайн-системы.

Самое важное — помнить про эти настройки и компенсировать ими дизайн. Например, в макете цвет слишком бледный? Спросите у дизайнера цвет поконтрастней и используйте его, если включен `isDarkerSystemColorsEnabled`. Используете слишком тонкий шрифт? Сделайте его жирнее, если включен `isBoldTextEnabled`. Добавили функционал, который срабатывает от встряхивания телефона? Отключите его и дайте иной способ включения, если выключена настройка `isShakeToUndoEnabled`. Сделали интересный переход между экранами? Круто, но замените его на стандартный кросфейд, если включена настройка `prefersCrossFadeTransitions`. Привязали статус заказа к цвету? Не забудьте, что есть `shouldDifferentiateWithoutColor`, лучше добавьте еще и описание текстом (все равно ведь он понадобится для VoiceOver).

```
// Returns whether the system preference for invert colors is
enabled.
public static var isInvertColorsEnabled: Bool { get }
// Returns whether the system preference for bold text is enabled
public static var isBoldTextEnabled: Bool { get }
// Returns whether the system preference for button shapes is
enabled
public static var buttonShapesEnabled: Bool { get }
// Returns whether the system preference for grayscale is enabled
public static var isGrayscaleEnabled: Bool { get }
// Returns whether the system preference for reduce transparency
is enabled
public static var isReduceTransparencyEnabled: Bool { get }
// Returns whether the system preference for reduce motion is
enabled
public static var isReduceMotionEnabled: Bool { get }
// Returns whether the system preference for reduce motion:
prefer cross-fade transitions is enabled
public static var prefersCrossFadeTransitions: Bool { get }
// Returns whether the system preference for auto-play videos is
enabled
public static var isVideoAutoplayEnabled: Bool { get }
// Returns whether the system preference for darker colors is
enabled
public static var isDarkerSystemColorsEnabled: Bool { get }
// Returns whether the system preference for Speak Selection is
enabled
public static var isSpeakSelectionEnabled: Bool { get }
// Returns whether the system preference for Speak Screen is
enabled
public static var isSpeakScreenEnabled: Bool { get }
// Returns whether the system preference for Shake to Undo is
enabled
public static var isShakeToUndoEnabled: Bool { get }
// Returns whether the system preference for Differentiate
without Color is enabled.
public static var shouldDifferentiateWithoutColor: Bool { get }
// Returns whether the system preference for On/Off labels is
enabled.
public static var isOnOffSwitchLabelsEnabled: Bool { get }
```

Рядышком находится и набор настроек, по которым можно понять, какая технология доступности сейчас включена.

```
/*
 Assistive Technology

 Use UIAccessibilityIsVoiceOverRunning() to determine if
VoiceOver is running.
 Listen for UIAccessibilityVoiceOverStatusDidChangeNotification
to know when VoiceOver starts or stops.
 */
public static var isVoiceOverRunning: Bool { get }

// Returns whether system audio is mixed down from stereo to
mono.
public static var isMonoAudioEnabled: Bool { get }

// Returns whether the system preference for closed captioning is
enabled.
public static var isClosedCaptioningEnabled: Bool { get }

// Returns whether the app is running under Guided Access mode.
public static var isGuidedAccessEnabled: Bool { get }

/*
 Use UIAccessibilityIsSwitchControlRunning() to determine if
Switch Control is running.
 Listen for
UIAccessibilitySwitchControlStatusDidChangeNotification to know
when Switch Control starts or stops.
 */
public static var isSwitchControlRunning: Bool { get }

// Returns whether the system preference for AssistiveTouch is
enabled.
// This always returns false if Guided Access is not enabled.
public static var isAssistiveTouchRunning: Bool { get }
```

Здесь будет еще

Продолжение книги еще выходит. Подписывайтесь на канал Dodo Mobile или мой твиттер, я сообщу о выходе.

Можно приходить на курс про доступность. Все расскажем, покажем, научим. Курс для дизайнеров, исследователей, iOS, андроид и web-программистов.

Сейчас у вас версия книги 0.13, выпущена 14 сентября 2021. Актуальную версию книги можно скачать на сайте.

Благодарность

Много людей помогло этой книжке появиться.

Мама **Любовь Рубанова** научила меня учиться.

Денис Петров рассказал в универе, как верстать книжки, показал интерфейс айфона и рассказал, чем он крут. Так я пришёл в мобильную разработку.

Елизавета Швец начала работу с IT-брендом в Dodo. Помню, как выпустили первую статью и я пообещал, что напишу книгу про интерфейсы. Даже подумать не мог, что интерфейсы будут для незрячих.

Анатолий Попко пришёл к **Вилсакому** и рассказал про проблемы незрячих людей. После этого продакт **Сергей Грязев** увидел презентацию о VoiceOver и сказал: «Прикольно, давайте посмотрим, что из этого получится».

Армен Хатаян поработал вместе со мной и рассказал многое из своего опыта. Мы вместе написали несколько статей — так про наш опыт стало известно за пределами компании.

Валерия Курмак убедила провести курс о цифровой доступности. Я готовился к нему 3 месяца — тогда и собрал весь материал. **Алексей Берёзка** присоединился к курсу и рассказал о Dynamic Type. **Все iOS—разработчики Dodo** единогласно поддержали доступность и слушали десяток выступлений на нашей iOS-гильдии.

После курса стало понятно, что надо делать книгу. **Максим Котин** издал уже не одну и помог советами. Редакторы **Олеся Балашова, Гаянэ Довгаль, Константин Рисков** выращивали во мне автора, помогали со статьями, занимались вычиткой и делились советами. **Татьяна Турманидзе** нарисовала обложку для книги и отдельных глав.

Юлия Рубанова поддерживала каждый вечер, который я проводил за книгой. Сын **Лев** по вечерам спал в кроватке, а по утрам не будил.

Если убрать любого из этой цепочки, то ничего бы не получилось. Спасибо всем!