Столыпинский вестник

Научная статья

Original article

УДК 004

ИССЛЕДОВАНИЕ СПОСОБА РЕАЛИЗАЦИИ ИМИТАЦИИ ИНТЕРФЕЙСА ОПЕРАЦИОННОЙ СИСТЕМЫ IOS НА ЯЗЫКЕ JAVASCRIPT

INVESTIGATION OF THE IMPLEMENTATION METHOD SIMULATIONS OF THE OPERATING SYSTEM INTERFACE IOS SYSTEMS IN JAVASCRIPT

Берьянов Максим Сергеевич, Магистрант, 2 курс, Факультет ПИиКТ, Университет ИТМОРоссия, г. Санкт-Петербург

Прокофьева Марина Витальевна, Магистрант, 2 курс, Факультет ПИиКТ, Университет ИТМО, Россия, г. Санкт-Петербург

Царев Иван Павлович, Магистрант, 2 курс, Факультет ПИиКТ, Университет ИТМО, Россия, г. Санкт-Петербург

Buryanov Maxim Sergeevich, Master's student, 2nd year, Faculty of Pmik, ITMO University Russia, St. Petersburg

Prokofieva Marina Vitalievna, Master's student, 2nd year, Faculty of Pmik, ITMO University, Russia, St. Petersburg

Tsarev Ivan Pavlovich, Master's student, 2nd year, Faculty of Pmik, ITMO University, Russia, St. Petersburg

Аннотация. Операционная система iOS, как и многие другие операционные системы, необязательно из мира мобильных устройств, является безоговорочно узнаваемой и распространенной среди большинства слоев населения, в частности, среди молодежи. Предлагается рассмотреть возможность реализации графической составляющей интерфейса iOS, в процессе создания которой полезно рассмотреть основные возможности таких инструментов, как CSS, JavaScript и HTML. Благодаря подробному последовательному описанию действий с конечным результатом в виде работающего продукта, текст данной статьи может быть полезен для желающих встроить мобильное решение на свой веб-сайт в достаточно эксклюзивной и креативной форме.

Abstract. The iOS operating system, like many other operating systems, not necessarily from the world of mobile devices, is unconditionally recognizable and common among most segments of the population, in particular, among young people. It is proposed to consider the possibility of implementing graphical component of iOS interface, in process of creating of which it is useful to consider main features of such tools as CSS, JavaScript and HTML. Due to the detailed step-by-step description of actions with the end result in the form of a working product, the text of this article can be useful for those who want to embed a mobile solution on their website in a rather exclusive and creative way.

Ключевые слова: JavaScript, iOS, React.JS, Образование.

Keywords: JavaScript, iOS, React.JS, Education.

При разработке решения фигурирует библиотека React.JS [1]. Первым делом стоит отметить, что будут использованы функциональные компоненты вида *export const Component: React.FC = () => {}.* Данный тренд разработки на React.JS с каждым годом превалирует все больше и больше, в отличие от

классовых компонентов, которые являются в достаточной степени неудобными и громоздкими.

Пусть будет определен компонент PhoneZone. Он будет возвращать контейнер (div), одноименный CSS-класс которого своими свойствами покрывает всю рабочую область браузера (width: 100vw, height: 100vh) и имеет flex-разметку с выравниванием по центру по всем направлениям (display: flex, align-items: center, justify-content: center). Стоит сразу заметить, что родительский тег тела сайта (body) имеет фон (background: url("…"), background-size: cover), а значит, что свойство описанного выше контейнера (backdrop-filter: blur(5px)) будет размывать его (принцип действие заключается в размытии всех компонентов, стоящих под ним). Нужно запомнить данный способ размытия, потому что в дальнейшем он будет часто использоваться (ввиду особенностей дизайна iOS).

Добавим возможность использовать курсор мыши в виде нажатия. Создадим пользовательский хук [2] useTouch. Данный хук будет возвращать контейнер (класса touch), а также 3 метода, которые изменяют свойства этого контейнера – touchMouseEnter, touchMouseDown, touchMouseUp при движении, нажатии и поднятии курсора мыши соответственно. Внутри хука useTouch создается ссылка на элемент с помощью встроенного React-хука useRef – const touchRef = useRef<HTMLDivElement>({} as HTMLDivElement). Далее, эта ссылка отправляется в качестве атрибута в контейнер (класса touch) ref={touchRef}, а затем обращение к нужному элементу нажатия в перечисленных функциях происходит по полю touchRef.current. В частности, у элемента нажатия вызывается конструкция style.setProperty, которая изменяет значение нужного стиля. В первом случае, происходит вычисление координат курсора через event.clientX, event.clientY, и отцентровка элемента нажатия через вычитание половины его размеров – touchRef.current.offsetWidth, touchRef.current.offsetHeight. Контейнер класса touch имеет абсолютное

позиционирование [3], поэтому его перемещение будет происходить через изменение свойств top и left вида setProperty("top", `${(top - height / 2).toString()}px`). При нажатии кнопкой изменяется ширина и высота контейнера, а также непрозрачность (с помощью изменения свойств width, height, background).

Далее, необходимо нарисовать корпус iPhone. Создадим компонент Phone. Он будет возвращать три контейнера одного уровня вложенности, поэтому они обрамляются с помощью React фрагмента < > < / >. Данные контейнеры будут отвечать за корпус (body-background), за интерактивный экран (body), и за челку (notch). Контейнеры классов body-background и body имеют фиксированное позиционирование и размер. Свойство контейнера класса body-background в виде box-shadow:inset 0px 0px 0px 10px #000000 позволяет создать внутреннюю рамку, а свойство border-radius: 45px скругляет углы. Псевдоэлемент :before добавляет рамку вокруг корпуса с помощью свойства border: 2px solid #0e69a9. Контейнер класса notch (челка), имеющий выверенные размеры и скругленные углы, с помощью свойства margin-bottom: 500px сдвигается на нужную позицию, так как это flex-элемент [4] и изначально находится в центре страницы. Таким образом, корпус телефона создан (рисунок 1).

Рисунок 1 – Корпус iPhone

Далее, элементы взаимодействия будут находиться в компоненте Screen, который создает контейнер класса screen-background. Данный контейнер является flex-контейнером. Он имеет свойство overflow: hidden для того, чтобы скрывать любое содержимое, выходящее за его границы, а также фон.

Дальнейшие разметки экрана блокировки и домашнего экрана будут располагаться именно в данном контейнере.

Приступаем к экрану блокировки – компоненту LockScreen. Стоит сразу заметить, что он оборачивается Provider'ом контекста [5] для того, чтобы не перебрасывать нужные для реализации разблокировки экрана свойства сквозь все дерево. Сам контекст состоит из 4 свойств – unlocked, onUnlock, dispatch, activateOnUnlock, которые отвечают за флаг того, что iPhone разблокирован, находится ли он в состоянии разблокировки в данный момент времени, функцию-вызов действий reducer'а, а также создание действия-разблокировки с некоторым payload. Данные сущности первоначально появляются в пользовательском хуке useUnlock. Внутри данного хука используется встроенный React хук useReducer, так как он более удобен в данном случае, чем хук useState в нескольких, что называется, экземплярах. Мы будем оперировать 3-мя случаями-действиями при разблокировке – случай, когда пользователь тянет с нажатием за жест разблокировки, случай, когда он поднимает курсор и случай, непосредственно, выявляющий, произошла ли разблокировка устройства. Интуитивно это можно описать так – допустим, данное устройство не имеет пароль, поэтому разблокируется оно только тогда, когда путь смахивания доходит примерно до середины экрана по вертикали. Также, в данном хуке реализуются обработчики событий, которые накладываются на уже упомянутый контейнер (класса screen-background), который возвращает компонент Screen. Во-первых, onMouseMoveUnlock использует в себе ссылку на контейнер (класса lockscreen из компонента текущего повествования LockScreen), которая берется из другого контекста, который, в свою очередь, ранее был обернут вокруг компонента Screen и еще не описан в данной статье, и который содержит в себе в том числе свойство lockscreenRef – данную ссылку. Далее, при нажатии на подсказку жеста запоминается позиция курсора (свойство onUnlockTop) и вычисляется

смещение по вертикали каждый раз, когда происходит срабатывание событие, вычитанием event.clientY. Далее, выключается свойство времени анимации [6] (а именно transform и backdrop-filter, которые анимированы в классе lockscreen с длительностью анимации в 0.4 секунды), а затем в зависимости от смещения изменяется степень размытия фона (с помощью изменения свойства backdrop-filter на смещениях в 0, 25, 50, 75 и т.д. пикселей. Более того, с помощью свойства transform происходит относительное смещение вверх контейнера lockscreen (через отрицательное число пикселей в translateY) – таким образом, что все элементы внутри данного контейнера также смещаются вверх, и скрываются из-за описанного выше свойства overflow:hidden родительского контейнера. Обработчик onMouseUpUnlock, в свою очередь, срабатывает уже после поднятия курсора – происходит аналогичное вычисление смещений, и если оно больше определенного (в 200 пикселей), то можно сказать, что iPhone разблокирован, следовательно изменить степень размытия до достаточно большого, а также скрыть экран блокировки полностью, при этом не отключая длительность анимации императивным образом – анимация полной разблокировки произойдет в таком случае автоматически. Либо, по такому же принципу, если блокировка не произошла, степень размытия перейдет в нулевое состояние, как и смещение – как и в оригинале операционной системы, экран блокировки плавно опустится вниз.

Взаимодействие с экраном блокировки успешно описано, осталось лишь косметически определить элементы на экране блокировки (можно заранее увидеть пример такого взаимодействия на рисунке 2 вместе с компонентами экрана блокировки) в компоненте LockscreenContent (заметим, что flex-разметка будет использоваться по всему проекту, в данном случае размещение элементов идет по вертикали с эмпирически выверенными вертикальными размерами данных элементов):

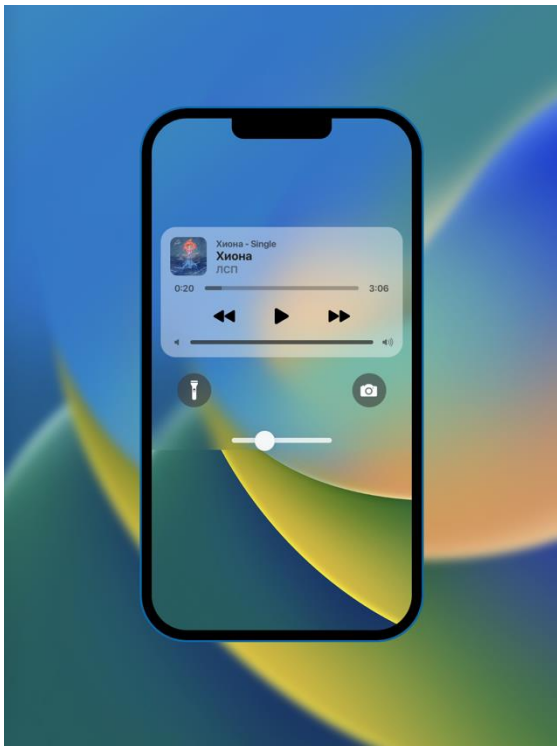

Рисунок 2 – Процесс разблокировки

– ControlPanel – иконка центра управления, содержится в flex-контейнере с выравниванием справа и по центру по вертикали (width: 100% display: flex, height: 10px, justify-content: flex-end, align-items: center), а сама иконка представляет из себя прямоугольник со скругленными углами и некоторой прозрачностью (width: 40px, height: 3px, margin-right: 27px, background: rgba(255, 255, 255, 0.3), border-radius: 3px),
– Clock – часы и дата, контейнеры которых вмещают в себе результат выполнения методов из библиотеки moment [7] по извлечению даты и времени (с помощью функции format) и размещают их по центру в горизонтальной плоскости (width: 100%, height: 50px, display: flex, justify-content: center, align-items: flex-end, color: rgba(255, 255, 255, 0.75), font-size: 14px),
– MusicPlayer – нетривиальный компонент, кратко описать который можно следующим образом. Создается flex-контейнер и разбивается

в соответствующим процентном соотношении под изображение обложки музыки и ее метаданные (последние располагаются в вертикальном порядке). Далее, создается еще один flex-контейнер, который разбивается на 3 части, одна из которых наиболее большая – таким образом реализуется строка проигрывателя (все элементы в ней, как и во всем плеере предлагается имитировать техникой hard-coding). Элементы управления музыкой строятся по тому же сценарию – аналогично и для зоны громкости музыки. Описанное выше справедливо для контейнеров классов music-meta-zone, music-control-zone, music-play-zone, music-sound-zone. Все используемые иконки берутся из библиотеки react-icons [8], которые содержат в себе в том числе иконки из операционной системы iOS. Размытие элементов происходит через свойство backdrop-filter, окрашивание элементов происходит через свойство color, необходимые смещения – через margin или padding в зависимости от ситуации,

– ControlButtons – компонент, имитационно реализующий элементы управления камерой и фонариком – здесь также flex-компонент разбивается на три части, серединная из которых пуста, а в боковых располагаются скругленные компонентами с соответствующими иконками внутри (width: 34px, height: 34px, border-radius: 17px, background: rgba(58, 58, 58, 0.6), backdrop-filter: blur(5px)),
– Swipe – жест-подсказка прямоугольник, выровненная по центру со скругленными углами и некоторой прозрачностью. В зависимости от того, разблокируется ли iPhone в данный момент, надпись, гласящая о смахивании вверх, пропадает или отображается через анимированное свойство opacity. При нажатии на жест-подсказку запоминается позиция курсора с использованием активации reducer’a (dispatch(activateOnUnlock(event.clientY)).

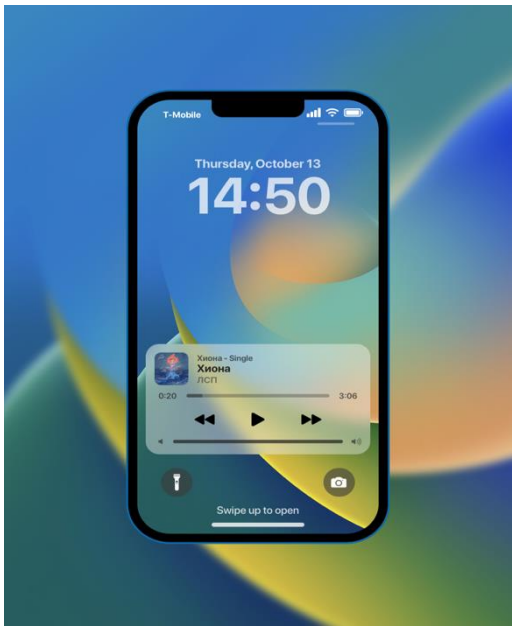

Рисунок 3 – Экран блокировки

Таким образом, экран блокировки полноценно реализован (рисунок 3). Теперь, предстоит описать реализацию домашнего экрана – рабочего стола с анимированным появлением иконок и док-бара.

Домашний экран представлен компонентом HomeScreen. Данный компонент отображается по условию – булева переменная unlocked (изменяющаяся через действие reducer'а типа unlock) определяет, следует ли отображать данный компонент. Данный компонент позиционируется изначально ровно на том же месте, что и экран блокировки, но появляется он лишь после того, как экран блокировки был перемещен вверх и телефон был разблокирован с помощью конструкции {unlocked && <HomeScreen/>}. Здесь стоит сразу заметить, что сохраняется ссылка на контейнер home-screen-pages (homeScreenPagesRef) в соответствующий контекст ScreenContext. Данная ссылка позднее будет использоваться при пролистывании домашнего экрана.

Опишем основную особенность при активации домашнего экрана – анимация иконок. Как мы видим, она безоговорочно нетривиальна. Тем не менее, ввиду того что сетка иконок в iOS закреплена, можно реализовать анимацию без каких-либо динамических вычислений. Все иконки будут располагаться в контейнере класса homescreen-icons-zone, который реализует технологию Grid CSS [9] (display: grid, grid-template-columns: 1fr 1fr 1fr 1fr, grid-template-rows: 1fr 1fr 1fr 1fr 1fr 1fr, gap: 5px 0px, justify-items: center, align-items: center, margin-left: 15px, margin-right: 15px, margin-top: 20px). Далее, при размещении иконок в контейнере нужно указать их общий класс homescreen-icon-zone, который видоизменяется с помощью CSS в соответствии с порядком

следования. Происходит это через псевдокласс nth-of-type, где для каждого порядкового номера, например, homescreen-icon-zone:nth-of-type(1), реализуется своя анимация и ее длительность (animation-name, animation-duration). Реализация собственной анимации происходит в виде @keyframes homescreen-icon-zone-appearance-1 {0% {opacity: 0; transform: translateY(50px) translateX(-30px) scale(1.3);} 100% { opacity: 1; transform: none;}}. То есть видно, что в зависимости от конечного положения иконки, происходит ее изначальное смещение через свойство transform (как translateX, так и translateY), а также изменяется масштаб через свойство scale. После того, как анимация воспроизведена, иконка принимает свое первоначальное состояние. Эмпирически было отмечено, что наименее движимые иконки – из первых двух рядов, особенно расположенные по центру, а наиболее движимые иконки – по нижним границам сетки иконок. Таким образом, после визуализации контейнера рабочего стола каждая из иконок рабочего стола будет появляться со своим эффектом анимации (попытку наглядного представления данного поведения можно увидеть на рисунке 4). Более того, после окончания анимации, нужно, чтобы иконки так и оставались непрозрачными – делается это с помощью атрибута onAnimationEnd контейнера и setProperty("opacity", "1") на event.target. Что касается самих иконок, предлагается в качестве фона использовать заранее подобранные .png картинки.

Рисунок 4 – Анимация разблокировки

Для того, чтобы пролистывать страницы (рисунок 5), реализуется пользовательский хук useSwitchPages, куда

Рисунок 5 – Перелистывание страниц

встроен reducer с состоянием onTurn, pageNumber, onTurnPageInitial, свойства которого отвечают соответственно за маркировку случая, когда пользователь хочет перелистнуть страницу, за номер текущей страницы, и номер страницы, откуда было начато перелистывание. Далее, внутри хука также есть такие обработчики событий мыши как следующие: homeScreenPagesMouseDown, homeScreenPagesMouseMove и, конечно же, homeScreenPagesMouseUp. Далее, следует описать их действие. Отмеченная выше функция homeScreenPagesMouseDown по аналогичной технологии вычисления относительного смещения, как и ранее, но на этот раз вправо или влево, определяет нужные для этого значения, перед этим отключая анимацию свойства transform, а homeScreenPagesMouseMove уже производит это необходимое смещение благодаря изменению translateX CSS-свойства transform. Функция homeScreenPagesMouseUp с помощью getBoundingClientRect у контейнеров homescreen и homescreen-pages определяет смещение, в том числе учитывая номер страницы, и если данное смещение достаточно для того, чтобы перелистнуть страницы, то такое поведение происходит. Иначе, смахивание страницы сбрасывается и плавно появляется та страница, с которой было начато перелистывание. Смысл в том, что страницы находятся слева-направо друг от друга в едином общем контейнере homescreen-pages, но показывается лишь определенная часть массива рабочих столов – единственного актуального homescreen-icons-zone. При перелистывании происходит смещение общего контейнера страниц

динамически через setProperty("transform", `translateX(-${(pageNumber) * 270}px)`), то есть на некоторое количество ширин страницы экрана. Общий вид сетки рабочего стола представлен на рисунке 6. Более того, помимо страниц иконок, есть еще индикатор рабочего стола. Он реализован в контейнере класса homescreen-dots-zone. В зависимости от числа страниц добавляется некоторое число контейнеров класса homescreen-dot-active или homescreen-dot-inactive, которые и составляют круглые индикаторы страниц. Важно также отметить зону док-бара, иконки на ней фактически остаются теми же контейнерами, что и иконки выше, с соответствующими для их положения анимациями. Тем не менее, они помещаются в контейнер класса homescreen-dock, который благодаря свойствам background: rgba(255, 255, 255, 0.4), backdrop-filter: blur(5px), border-radius: 20px имитирует подложку iOS док-бара для иконок.

Рисунок 6 – Рабочий стол

Таким образом, была создана реализация имитации интерфейса iOS системы. В процессе повествования был прокомментирован ряд технологических особенностей созданного решения. Весь существующий код можно найти в репозитории автора данной статьи по ссылке: https://github.com/msberyanov/ios-interface-imitation [10], а представленное подробное описание в тексте данной статьи может помочь при встраивании демо-интерфейса мобильного приложения (iOS) для продукта определенной компании по аналогии на веб-сайт.

Библиографический список

1. React.JS. // [Электронный ресурс] (https://reactjs.org, дата доступа – 12.10.2022).
2. Building your own hooks. // [Электронный ресурс] (https://reactjs.org/docs/hooks-custom.html, дата доступа – 12.10.2022).
3. Position – CSS. // [Электронный ресурс] (https://developer.mozilla.org/en-US/docs/Web/CSS/position, дата доступа – 12.10.2022).
4. Полное руководство по Flexbox CSS. // [Электронный ресурс] (https://habr.com/ru/post/467049, дата доступа – 12.10.2022).
5. Context – React. // [Электронный ресурс] (https://reactjs.org/docs/context.html, дата доступа – 12.10.2022).
6. CSS Animations. // [Электронный ресурс] (https://www.w3schools.com/css/css3_animations.asp, дата доступа – 12.10.2022).
7. Moment.JS. // [Электронный ресурс] (https://momentjs.com, дата доступа – 12.10.2022).
8. React Icons. // [Электронный ресурс] (https://react-icons.github.io/react-icons/, дата доступа – 12.10.2022).
9. Grid Layout – CSS. // [Электронный ресурс] (https://developer.mozilla.org/ru/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout, дата доступа – 12.10.2022).
10. GitHub | msberyanov / ios-interface-imitation. // [Электронный ресурс] (https://github.com/msberyanov/ios-interface-imitation, дата доступа – 12.10.2022).

Bibliographic list

1. React.JS . // [Electronic resource] (https://reactjs.org , access date – 12.10.2022).

2. Building your own hooks. // [Electronic resource] (https://reactjs.org/docs/hooks-custom.html , access date – 12.10.2022).
3. Position – CSS. // [Electronic resource] (https://developer.mozilla.org/en-US/docs/Web/CSS/position , access date – 12.10.2022).
4. Complete guide to Flexbox CSS. // [Electronic resource] (https://habr.com/ru/post/467049 , access date – 12.10.2022).
5. Context – React. // [Electronic resource] (https://reactjs.org/docs/context.html , access date – 12.10.2022).
6. CSS Animations. // [Electronic resource] (https://www.w3schools.com/css/css3_animations.asp , access date – 12.10.2022).
7. Moment.JS . // [Electronic resource] (https://momentjs.com , access date – 12.10.2022).
8. React Icons. // [Electronic resource] (https://react-icons.github.io/react-icons /, access date – 12.10.2022).
9. Grid Layout – CSS. // [Electronic resource] (https://developer.mozilla.org/ru/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout , accessed 12.10.2022).
10. GitHub | msberyanov / ios-interface-imitation. // [Electronic resource] (https://github.com/msberyanov/ios-interface-imitation , access date – 12.10.2022).

Для цитирования: Берьянов М.С., Прокофьева М.В., Царев И.П. ОСОБЕННОСТИ ИНТЕГРАЦИИ «ИССЛЕДОВАНИЕ СПОСОБА РЕАЛИЗАЦИИ ИМИТАЦИИ ИНТЕРФЕЙСА ОПЕРАЦИОННОЙ СИСТЕМЫ IOS НА ЯЗЫКЕ JAVASCRIPT // Научный сетевой журнал «Столыпинский вестник» №8/2022.