**Анимации в React**

Пользовательские приложения с каждым днём становятся сложнее и интерактивнее. Чтобы улучшить пользовательский опыт, нужно «оживить» интерфейс — добавить анимации на типовые сценарии работы: получение нового сообщения, создание записи в ленте новостей, удаление товара из корзины, переход между экранами.

В этом уроке рассмотрим два инструмента, с помощью которых можно добавить анимацию в приложение: CSS-анимации и ReactTransitionGroup.

**CSS-анимации**

Это не только простой и доступный метод, но и самый нативный — CSS-анимации используются при работе с другими фреймворками и библиотеками. Рассмотрим на примере, как реализовать простую анимацию:

Скопировать кодJSX

import React, { useRef, useState } from "react";

import "./style.css";

const ExampleCss = () => {

const wrapperRef = useRef(null);

const [isOpen, setOpen] = useState(false);

const handleClick = () => {

const wrapper = wrapperRef.current;

setOpen(!isOpen);

wrapper.classList.toggle("sidebarIsOpen");

};

return (

<div ref={wrapperRef} className="wrapper">

<div className="sidebar">

<button className="sidebarToggle" onClick={handleClick}>

{isOpen ? "<" : ">"}

</button>

<div className="sidebarContent">

Aliquam convallis eros vitae dolor hendrerit malesuada. Duis

vitae imperdiet justo, non finibus mauris. Integer viverra

facilisis porta. Sed eu arcu eu eros mollis auctor faucibus

a augue. Quisque non condimentum dui, ut facilisis odio.

Quisque varius posuere dignissim. Cras felis dolor, pharetra

nec mi at, lacinia lobortis odio.

</div>

</div>

</div>

);

};

export default ExampleCss;

Добавим стили:

Скопировать кодCSS

*/\* style.css \*/*

.wrapper {

display: flex;

width: 100%;

height: 100%;

transition: margin 0.5s;

margin: 0 0 0 -250px;

}

.wrapper.sidebarIsOpen {

margin-left: 0;

}

.sidebar {

position: relative;

width: 250px;

height: 100%;

padding: 20px;

border-right: 1px solid #ccc;

}

.sidebarToggle {

position: absolute;

top: 0;

right: -60px;

padding: 20px;

font-size: 20px;

cursor: pointer;

transition: color 0.3s;

}

.sidebarToggle:hover {

color: #5eb2ff;

}

Вот так выглядит финальный результат:

CSS-анимации позволяют создавать простые анимации, но у этого инструмента есть ряд ограничений. К примеру, особенности рендеринга, из-за которых HTML-код компонента практически моментально монтируется в DOM и также быстро пропадает при размонтировании. О том, как с этим работать — поговорим дальше.

**ReactTransitionGroup**

ReactTransitionGroup — библиотека, разработанная сообществом React. Установить её можно так:

Скопировать кодBASH

*# npm*

npm install react-transition-group --save

*# yarn*

yarn add react-transition-group

Эта библиотека помогает решить проблему отрисовки компонентов, которая возникает при работе с CSS-анимацией. Для этого у неё есть специальные компоненты, которые при размонтировании своих children добавляют задержку, что позволяет отработать анимации. Только после этого срабатывают методы жизненного цикла компонента, которые отвечают за монтирование и размонтирование.

Библиотека ReactTransitionGroup изменяет классы дочерних элементов на разных этапах жизни компонента, и название этих классов подчиняется определённым правилам. Рассмотрим, как реализовать анимацию элементов списка с помощью ReactTransitionGroup:

Скопировать кодJSX

import React, { useState } from "react";

import { CSSTransitionGroup } from "react-transition-group";

import Button from "./button";

import Item from "./item";

import "./style.css";

const ReactTransitionGroupExample = () => {

const [items, setItems] = useState(["Элемент 1"]);

const handleAdd = () => {

let newItems = [...items];

newItems.push("Элемент " + (items.length + 1));

setItems(newItems);

};

const handleRemove = (i) => {

let newItems = [...items];

newItems.splice(i, 1);

setItems(newItems);

};

const itemsToRender = items.map((item, index) => (

<Item

item={item}

key={index}

index={index}

handleRemove={handleRemove}

/>

));

return (

<div className="react-transition-group">

<Button onClick={handleAdd} />

<div className="container">

<CSSTransitionGroup transitionName="example">

{itemsToRender}

</CSSTransitionGroup>

</div>

</div>

);

};

export default ReactTransitionGroupExample;

Сперва мы импортировали CSSTransitionGroup из react-transition-group. Затем этим компонентом обернули список элементов itemsToRender. Также мы установили пропс transitionName компоненту CSSTransitionGroup. Именно этот пропс управляет набором классов, когда добавляется или удаляется дочерний элемент в CSSTransitionGroup:

Скопировать кодJSX

<CSSTransitionGroup

transitionName="example"

>

{items}

</CSSTransitionGroup>

Поскольку у пропса transitionName установлено значение example, классы стилей должны начинаться с этого префикса:

Скопировать кодCSS

*/\* Этот класс и следующий сработают при монтировании \*/*

.example-enter {

opacity: 0.01;

}

*/\* Такая система классов позволяет элементам появляться плавно \*/*

*/\* А вся анимация при монтированнии срабатывает "между" этими классами \*/*

.example-enter.example-enter-active {

opacity: 1;

transition: opacity 300ms ease-in;

}

*/\* Этот класс и следующий сработают при размонтировании \*/*

.example-leave {

opacity: 1;

}

*/\* Аналогично и с размонтированием — это позволяет элементам плавно исчезать \*/*

.example-leave.example-leave-active {

opacity: 0.01;

transition: opacity 300ms ease-in;

}

А вот и результат:

У библиотеки ReactTransitionGroup есть ещё несколько козырей в рукаве:

* SwitchTransition позволяет управлять цепочками анимаций и плавно переходить от одной анимации к другой.
* TransitionGroup — компонент для более продвинутой анимации в списках. Позволяет плавно удалять и добавлять элементы списка и гарантирует плавный сдвиг остальных элементов.
* Transition — компонент, который помогает управлять этапами анимации программно. Может выручить, если вы используете css-in-js подход. Также он несколько похож на CSSTransitionGroup из этого урока.

Подробнее ознакомиться с возможностями этой библиотеки вы можете [на официальном сайте](http://reactcommunity.org/react-transition-group/). В следующем уроке расскажем о декларативной анимации с использованием библиотеки Framer Motion.

**Анимации в React. Библиотека Framer Motion**

В предыдущем уроке вы познакомились с инструментами, с помощью которых можно создавать анимации в React-приложениях. Их достаточно, чтобы сделать незамысловатую анимацию. Но если нам потребуется реализовать более сложные сценарии анимации и связать её с действиями пользователя, лучше воспользоваться другим инструментом — библиотекой Framer Motion. Она следует декларативному подходу к программированию и позволяет описать простым кодом даже самую сложную анимацию.

**Что умеет библиотека**

Библиотека имеет богатый набор вспомогательных функций и хуков для работы с жестами, drag-событиями, также lazy-loading обёртки для уменьшения объёма итогового кода, который загружается при первом посещении сайта и прочие инструменты. В этом уроке разберём, как применять библиотеку Framer Motion для создания анимаций в пользовательских интерфейсах.

**Использование библиотеки**

Framer Motion — библиотека, которая содержит удобные абстракции для «оживления» пользовательского интерфейса. Установить её можно так:

Скопировать кодBASH

*# npm*

npm install framer-motion --save

*# yarn*

yarn add framer-motion

Библиотека предоставляет собственные обёртки над привычными HTML-элементами. Такие обёртки позволяют передавать дополнительные пропсы, с помощью которых можно настраивать анимации:

1. Гибкий пропс animate, который позволяет управлять перемещением по осям, поворотами, искривлениями, переходами анимации (keyframes) и прочими параметрами:

Скопировать кодJSX

import { motion } from "framer-motion"

export const MyComponent = () => (

<motion.div

animate={{

scale: [1, 2, 2, 1, 1],

rotate: [0, 0, 270, 270, 0],

borderRadius: ["20%", "20%", "50%", "50%", "20%"],

}}

/>

)

1. Не менее гибкий пропс transition, который позволяет точечно влиять на процесс воспроизведения анимации, например на её продолжительность с применением кривой Безье:

Скопировать кодJSX

import { motion } from "framer-motion"

export const MyComponent = () => (

<motion.div

animate={{ scale: 2 }}

transition={{ duration: 0.5, repeat: Infinity }}

/>

)

1. Пропс initial, который помогает указывать начальное положение элемента. Если передать false, то при монтировании анимация не сработает.
2. Пропс variants, который служит для описания предзаданных визуальных состояний элемента, переход между которыми возможен по условию:

Скопировать кодJSX

import { motion } from "framer-motion"

const variants = {

open: { opacity: 1, x: 0 },

closed: { opacity: 0, x: "-100%" },

}

export const MyComponent = () => {

const [isOpen, setIsOpen] = useState(false)

return (

<motion.nav

animate={isOpen ? "open" : "closed"}

variants={variants}

>

<Toggle onClick={() => setIsOpen(isOpen => !isOpen)} />

<Items />

</motion.nav>

)

}

Ещё этот пропс позволяет использовать динамически вычисляемые значения с помощью вспомогательного пропса custom:

Скопировать кодJSX

const variants = {

visible: i => ({

opacity: 1,

transition: {

delay: i \* 0.3,

},

}),

hidden: { opacity: 0 },

}

return items.map((item, i) => (

<motion.li

custom={i}

animate="visible"

variants={variants}

/>

))

В этом примере каждый элемент списка появляется с задержкой в 0.3 секунды по отношению к предыдущему.

Кроме этого, variants можно использовать для оркестрирования анимациями у вложенных элементов при условии, что у контейнера с элементами настроены transition свойства:

Скопировать кодJSX

import \* as React from "react";

import { motion } from "framer-motion";

const names = ["Василий", "Моника", "Тигран", "Сабиха", "Борис", "Ева"];

const container = {

hidden: { opacity: 0 },

show: {

opacity: 1,

transition: {

*// Управляет задержкой срабатывания анимации у дочерних элементов*

staggerChildren: 0.3

}

}

};

*// Предзаданные состояния элементов списка*

const listItem = {

hidden: { opacity: 0 },

show: { opacity: 1 }

};

export const Example = () => {

return (

<motion.ul variants={container} initial="hidden" animate="show">

{names.map((name, i) => (

<motion.li key={i} variants={listItem}>

{name}

</motion.li>

))}

</motion.ul>

);

};

Обратите внимание, что ключи variants у motion.ul совпадают с ключами «вариантов» у дочерних motion.li, но библиотека не накладывает никаких ограничений на количество вариантов, а так же на их название. Поэтому вместо show и hidden можно использовать что-то по своему вкусу. Важно при этом изменить значения пропсов initial и animate у motion.ul так, чтобы они соответствовали ключам в variants.

Вот результат такой работы конфигурации:

1. Пропс layout, который позволяет автоматически анимировать изменения вёрстки, к примеру:
   * изменение порядка элементов в списке,
   * изменения flex-direction, justify-content и подобных CSS-директив у родительского элемента,
   * изменение postion или width у анимируемого элемента.

Вот как это работает:

Скопировать кодJSX

*// ... Код компонента*

const [isOn, setIsOn] = useState(false);

const toggleSwitch = () => setIsOn(!isOn);

return (

<div className="switch" data-isOn={isOn} onClick={toggleSwitch}>

<motion.div className="handle" layout={true} />

</div>

);

Скопировать кодCSS

*/\* ... Стили компонента \*/*

.switch {

width: 160px;

height: 100px;

background-color: rgba(255, 255, 255, 0.4);

display: flex;

justify-content: flex-start;

border-radius: 50px;

padding: 10px;

cursor: pointer;

}

.switch[data-isOn="true"] {

justify-content: flex-end;

}

.handle {

width: 80px;

height: 80px;

background-color: white;

border-radius: 40px;

}

Свойство justify-content изменяется при нажатии на компонент:

В этом примере мы не указываем параметры анимации, но библиотека Framer Motion отслеживает изменения вёрстки и применяет анимацию автоматически.

**Заключение**

В этом уроке вы познакомились с основами библиотеки Framer Motion. Она позволяет реализовывать сложные цепочки анимаций, трансформировать svg, обрабатывать жесты пользователей, симулировать законы физики при анимировании элементов и многое другое. Если эта библиотека показалась вам интересной и полезной, рекомендуем изучить её более подробно [на официальном сайте](https://www.framer.com/api/motion/).