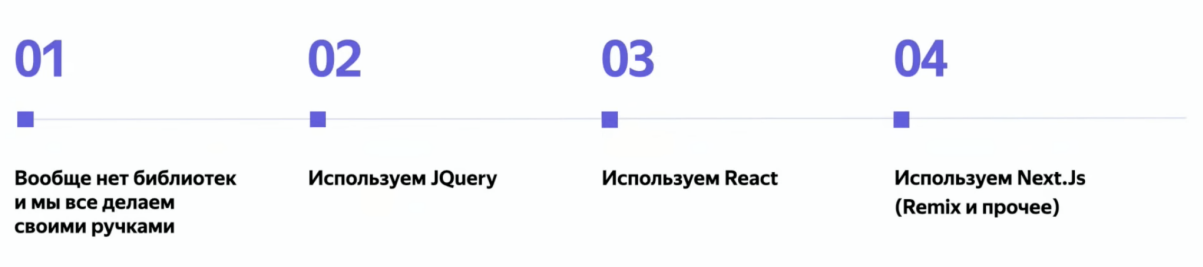
React база

Status	In progress
URL	https://www.youtube.com/watch?v=5LH5M5wvz34&list=PL3RW2ymbp1lgxp2aKGVxT12QZE43DSuu-&index=10

Что такое React? Почему он?

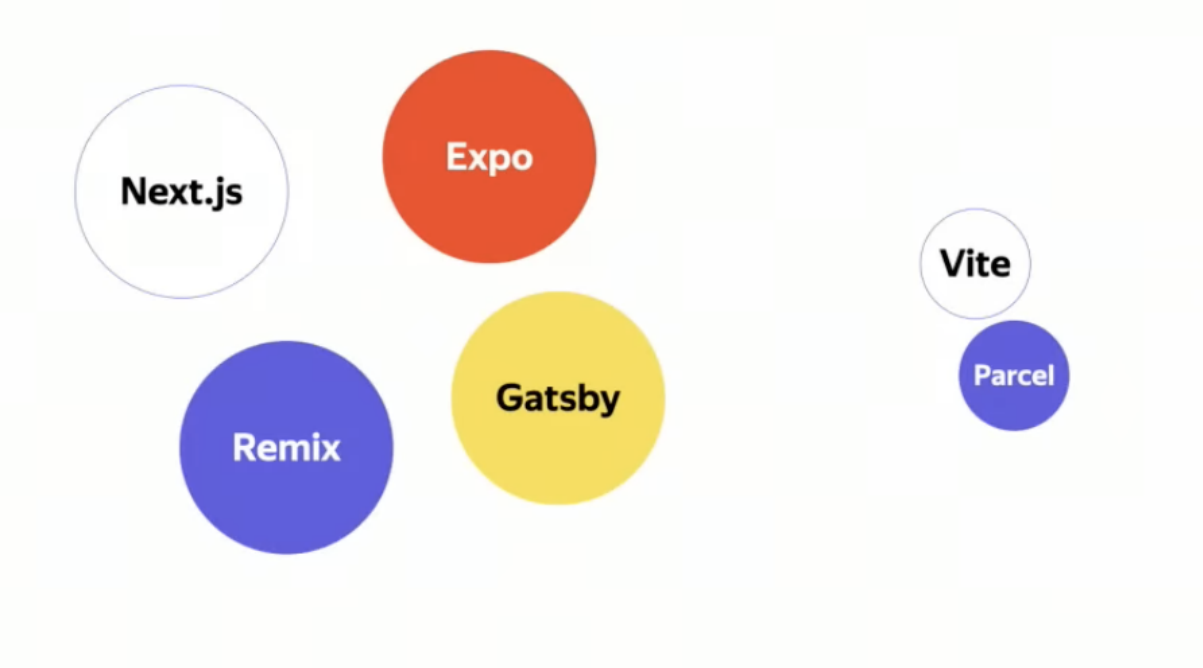


React - js библиотека для создания пользовательских (не только WEB) интерфейсов

Плюсы

- Очень просто что-то сделать
- Декларативность
- Компонентный подход
- Популярность и стабильность

Что такое Next? Почему он?



- Большое - фреймворки

- Маленькое - ближе к чистому реакту
- create-react-app больше нет

Next - это фреймворк для создания веб-приложений

Плюсы

- Фреймворк
- Много проектов уже в проде
- Тесный контакт с разработчиками React

Основы React

Компонентный подход

Строительные объекты

React элементы

- DOM - js объекты, на основе которых появляются объекты DOM в браузере (минимальные детали)
- Component - более сложные детали

Создание React элементов DOM

```
export default function Home(){
  return (
    React.createElement(
    'span',
      { children: 'Hello Students' }
    )
  )
}
```

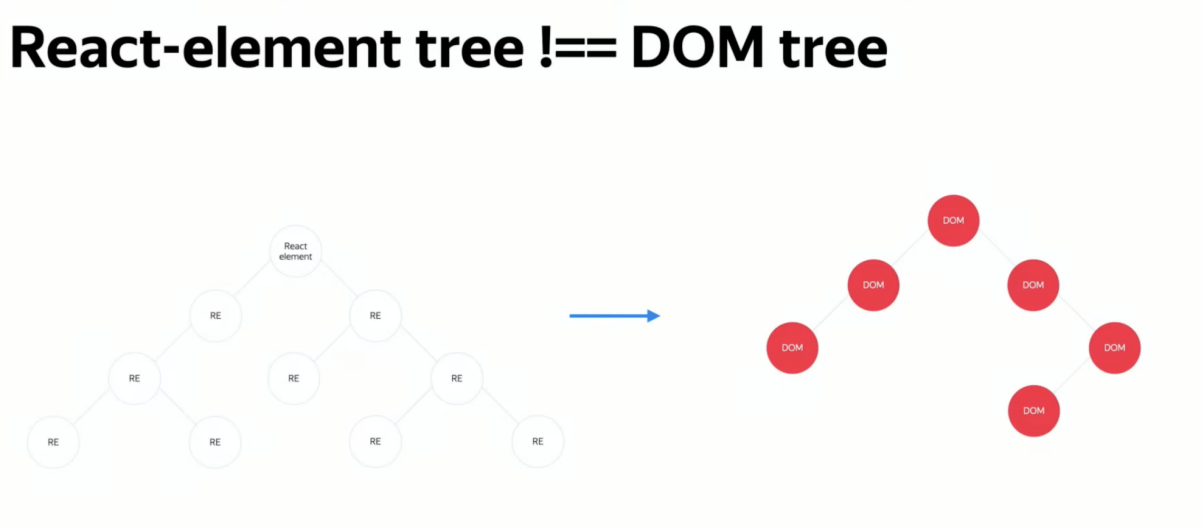
Два вида компонентов

- Классовые
- Функциональные

Функциональные компоненты

- Называть существительными
- С большой буквы
- return ReactElement || null

React-element tree



Этапы жизни

- Mount
- Update
- Unmount

Хуки

- Функции с особыми силами
- Названия начинаются с use

- Хуки можно вызывать только в функциональных компонентах или других хуках
- Хуки нельзя вызывать условно, в циклах, в вложенных функциях
- Количество рендеров = количество вызовов

useState

- На первом рендере (маунте) создается состояние
- На остальных - переиспользуется
- При дестрое - уничтожается
- Для каждого экземпляра компонента - свое состояние
- Аргумент - дефолтное значение или функция, которая его вычисляет (если функция, то она вызовется только один раз)

useEffect

- Хук для запуска сайд-эффектов
- Принимает функцию и массив зависимостей (следит за изменениями в нем, примитивы сравниваются по значению, остальное - по ссылке)
- Исполняется после рендера
- На маунте вызывается всегда
- Если нет массива зависимостей, то вызывается на каждом рендере
- В return можно вернуть функцию - clean up. Она вызывается или перед следующим вызовом useEffect, либо на дестрое

Кастомные хуки

```
export function useCount(initialValue: number = 0){

    //можем использовать, так как это хук внутри хука
    let [count, setCount] = useState(initialValue);

    const decrement = () => {setCount(count - 1)};
    const increment = () => {setCount(count+1)};

    //если хук абстрактный, то возвращать будем, скорее всего, массив
    //если нужно вернуть более двух элементов, то лучше объект
```

```
    return { count, decrement, increment }
}
```

- Если мы будем использовать decrement, increment в useEffect зависимостях, то срабатывать будет каждый раз (так как ссылки меняются)
- Стабильные ссылки - одинаковые на всех рендерах
- setCount - стабильная, increment/decrement - нестабильные
- Надо стараться возвращать из кастомных хуков как можно более стабильные ссылки

useCallback

- Хук, который позволяет создавать стабильные ссылки
- Создает ссылку на функцию
- Второй аргумент - массив зависимостей

```
useCallback(()=>{setCount(count - 1)}, []);
```

useMemo

- сохраняет значение, если параметры пришли те же, не пересчитывает
- Использовать лучше только
  - на сложных операциях (которых на фронте не должно быть)
  - для создания объектов и массивов со стабильными ссылками

```
const filmRating = useMemo(()=>{
  return Math.floor(
    filmDetails.reviews.reduce((sum, review)=> {
      return sum + review.rating
    },0)/filmDetails.reviews.length
  )
}, [filmDetails.reviews]);
```

useReducer

- Хук, который позволяет работать со сложными состояниями
- Принимает state и action

```
const reducer = (state: {name: string, text: string, rating: number}, action: {type: s
tring, payload?: any}) => {
        switch (action?.type){
            case FORM_ACTIONS.setName:
                return {name: action.payload.name, text: "", rating: 0};
            case FORM_ACTIONS.setText:
                return {...state, text: action.payload.text};
            case FORM_ACTIONS.setRating:
                return {...state, rating: action.payload.rating};
            default:
                return state;
        }
    }
    const initialState = {
        name: '',
        text: '',
        rating: 0
    }
    const [state, dispatch] = useReducer(reducer, initialState);

    const onNameChange = (event: ChangeEvent<HTMLInputElement>) => dispatch(
        {type: FORM_ACTIONS.setName, payload: {name: event.target.value}}
    )
```

useRef

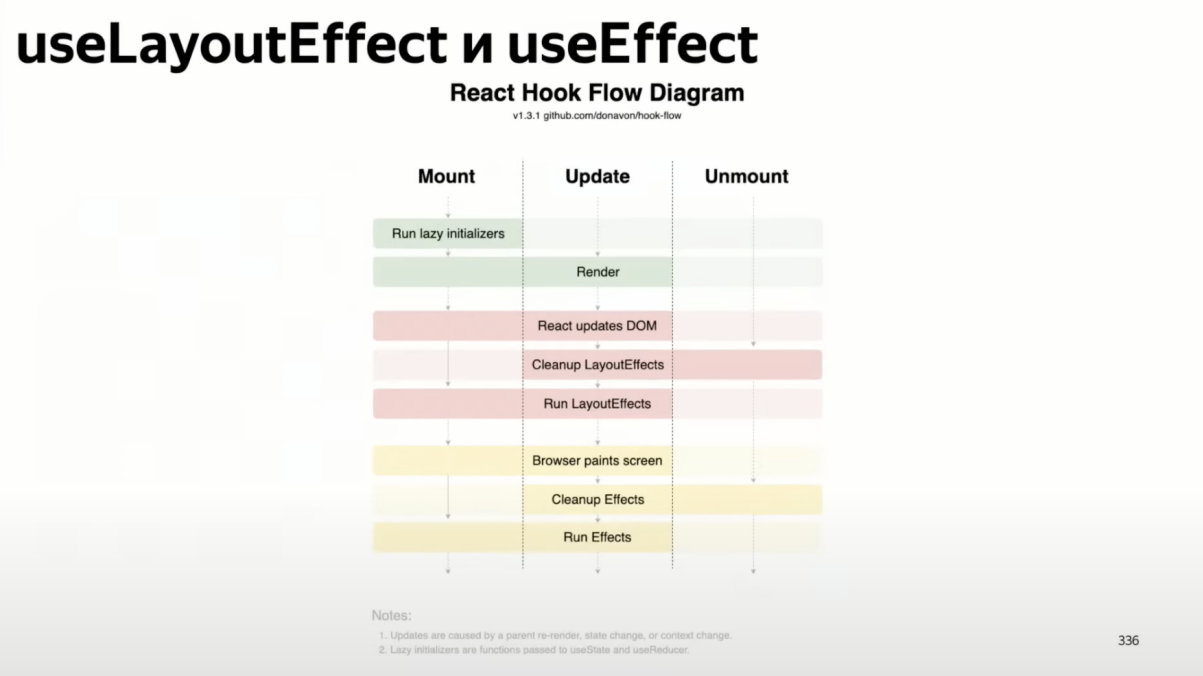
- Возвращает объект, в котором есть поле current (туда можно положить что угодно)
- Используется в двух случаях
  - Как хранилище, когда не хотим вызывать перерендеры
  - Логика, связанная с работой с DOM

```
const ref = useRef();

    useEffect(()=>{
        if (ref.current){
            ref.current.focus();
        }
    }, [])
```

- Если в ref элемента передать функцию, она вызывется при появлении элемента в DOM

useLayoutEffect



- Если эффект должен сработать до того, как произойдет отрисовка - useLayoutEffect (важно отображение)
- Иначе - useEffect

Fragment

- Обертка для компонентов, которой не будет в DOM

Стили

- Один из вариантов: css модули
- Гарантируют уникальность названий классов (добавляет хэш)
- В названии должно быть module
- Лучше использовать библиотеку classnames для сложения стилей

```
.title{
    color: red;
}
```

```
'use client';
import React, {FunctionComponent} from "react";
```

```
import classnames from "classnames";
import styles from './style.module.css';
interface Props {
    title: string;
    genre: string;
    seasonCount: number;
}

export const FilmInfo: FunctionComponent<Props> = ({
       title,
       genre,
       seasonCount,
    })=>{
    return (
        <>
            <p className={classnames(styles.title, styles.text)}>{title || "Неизвестны
й"}</p>
            <p>{Boolean(genre) && <p>{genre}</p>}</p>
            <p>{seasonCount > 0 ? `Кол-во ${seasonCount}`: 'Нет'}</p>
        </>
    )
}
```

- Внешние стили - положение компонента относительно других
- Внутренние стили - всегда с компонентом