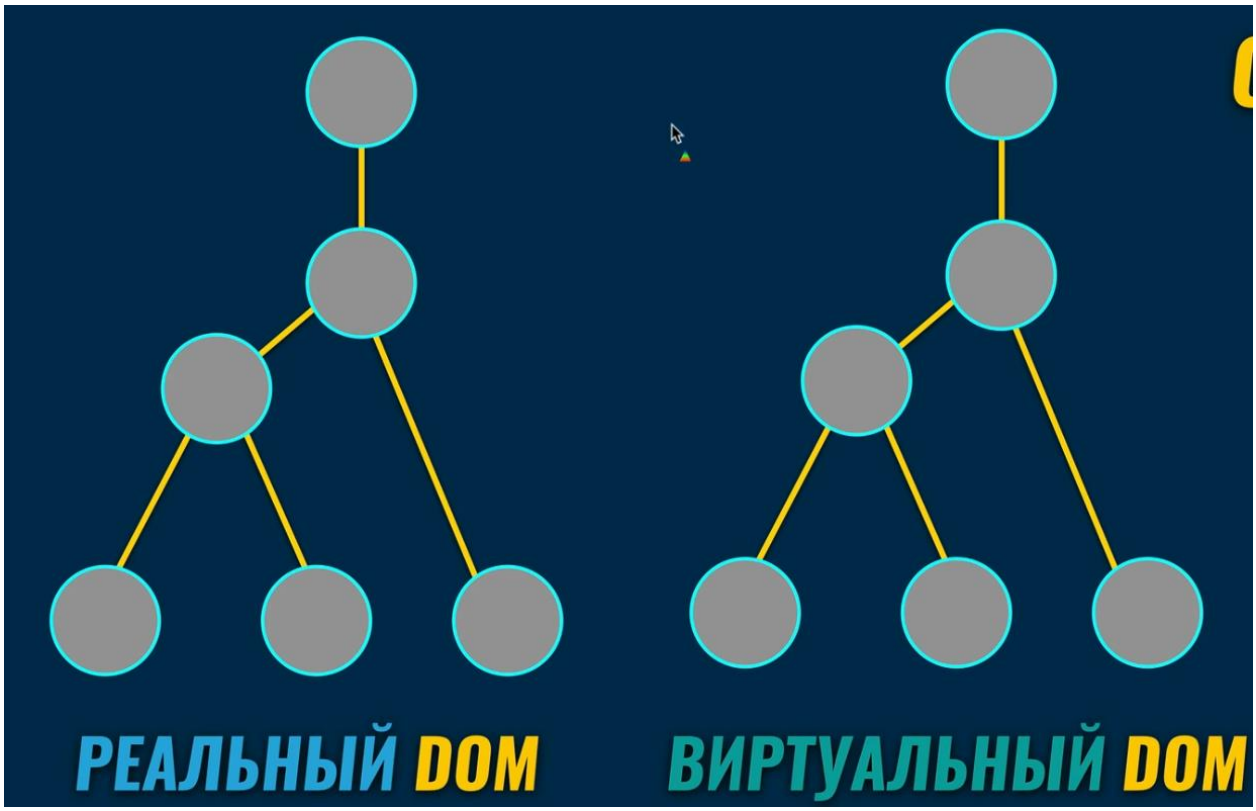
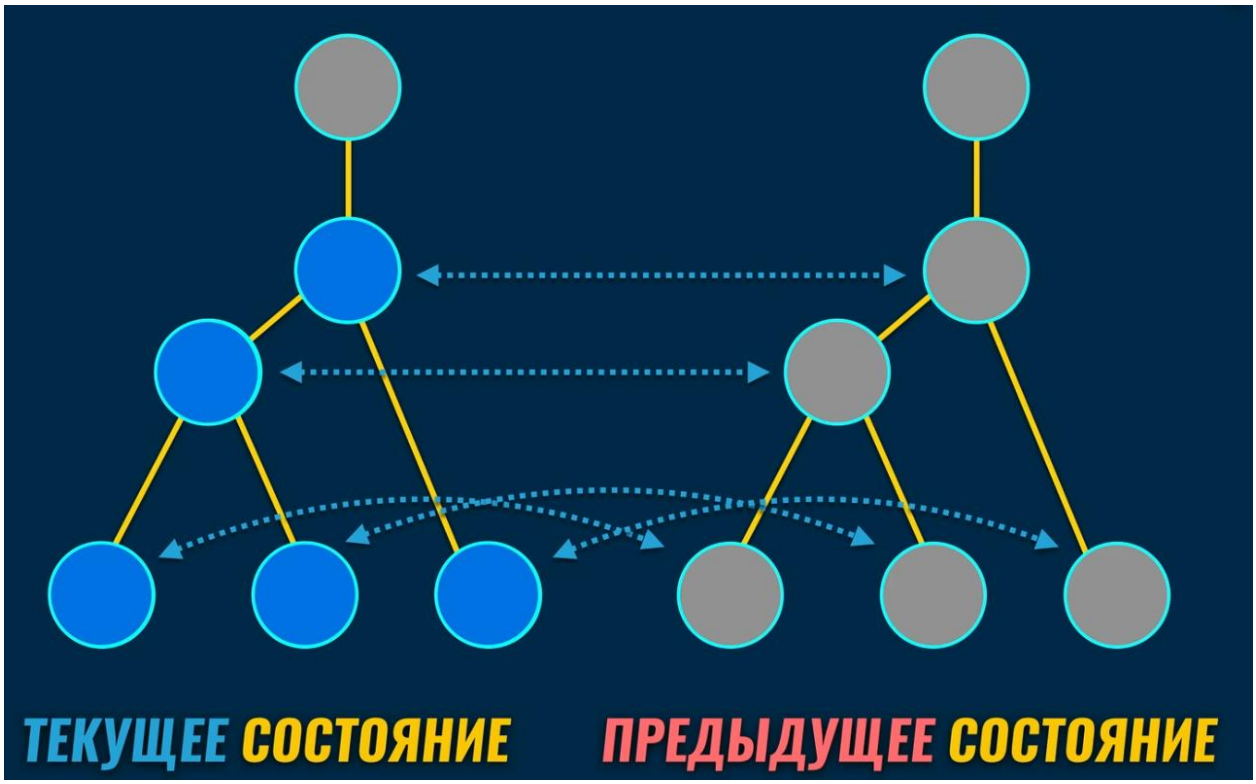
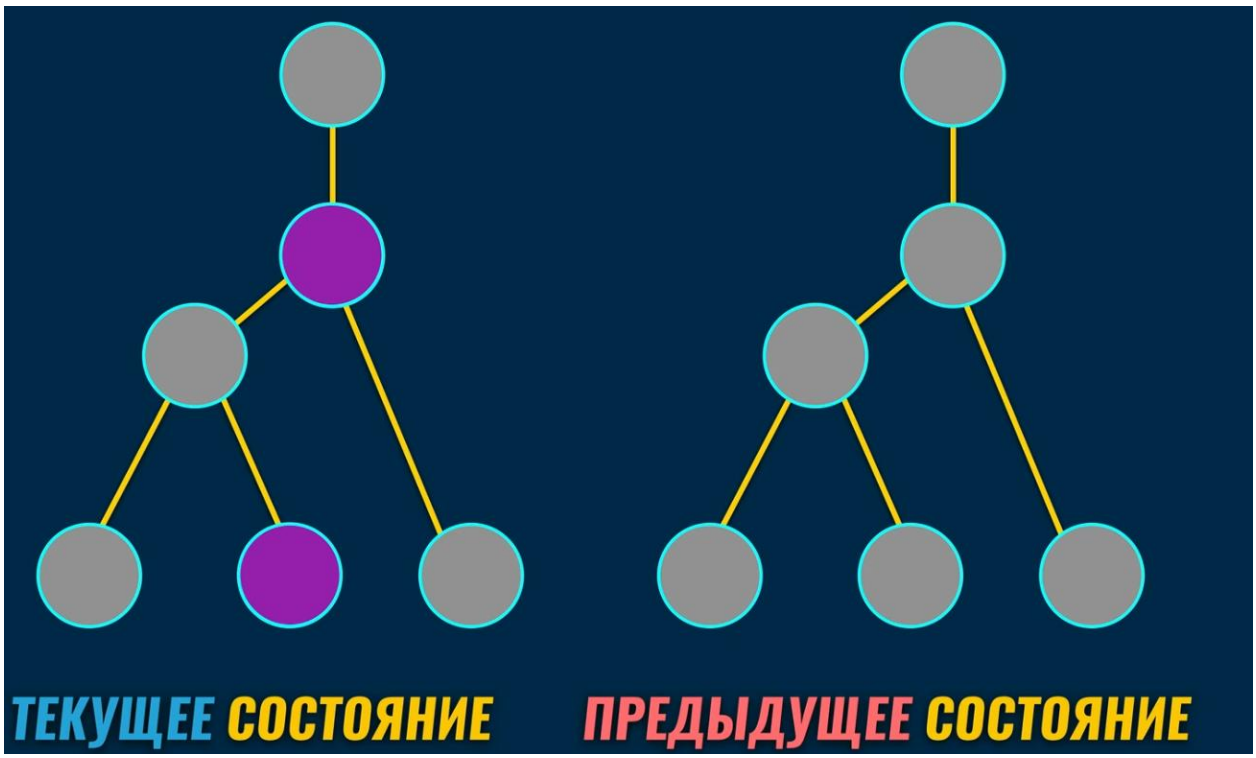
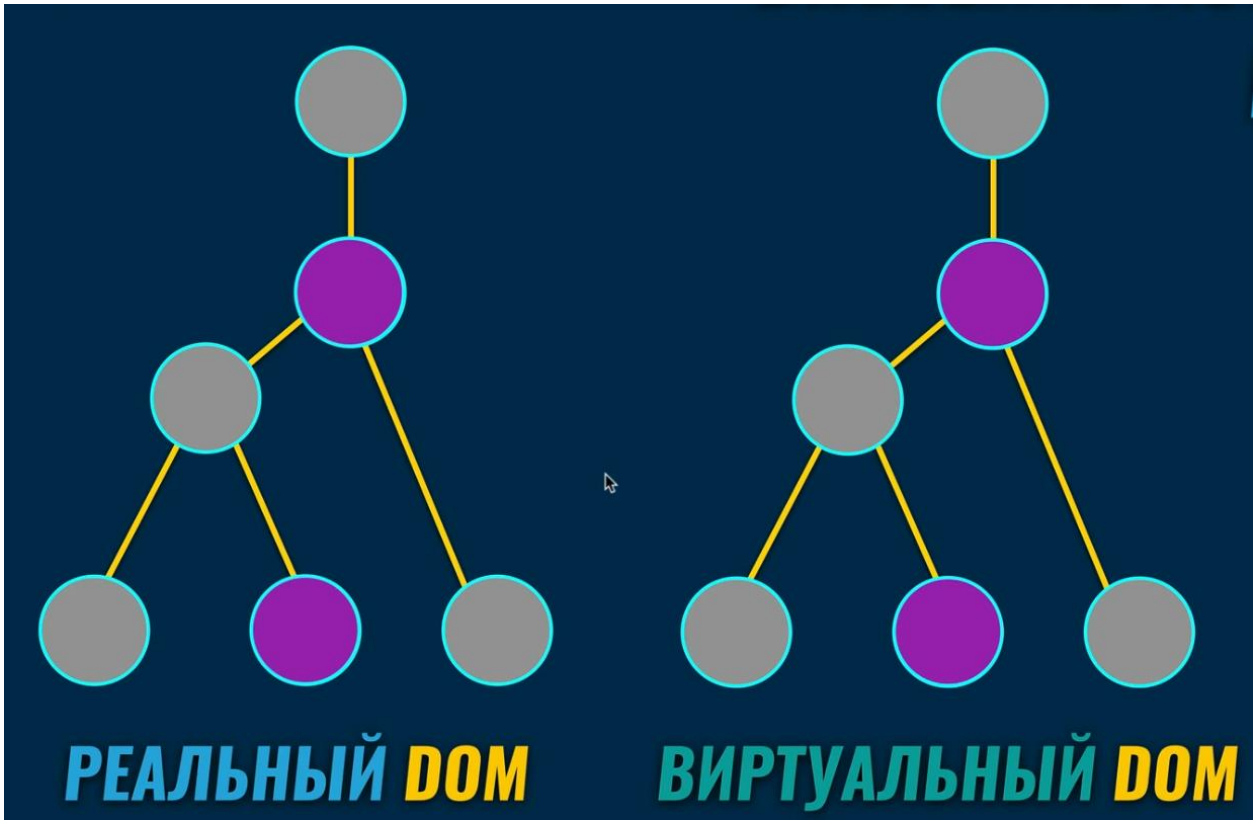
Основы React и взаимодействие с DOM






ТЕКУЩЕЕ СОСТОЯНИЕ
ПРЕДЫДУЩЕЕ СОСТОЯНИЕ


РЕАЛЬНЫЙ DOM
ВИРТУАЛЬНЫЙ DOM

КЛЮЧЕВЫЕ ПОНЯТИЯ В REACT

КОМПОНЕНТЫ
components

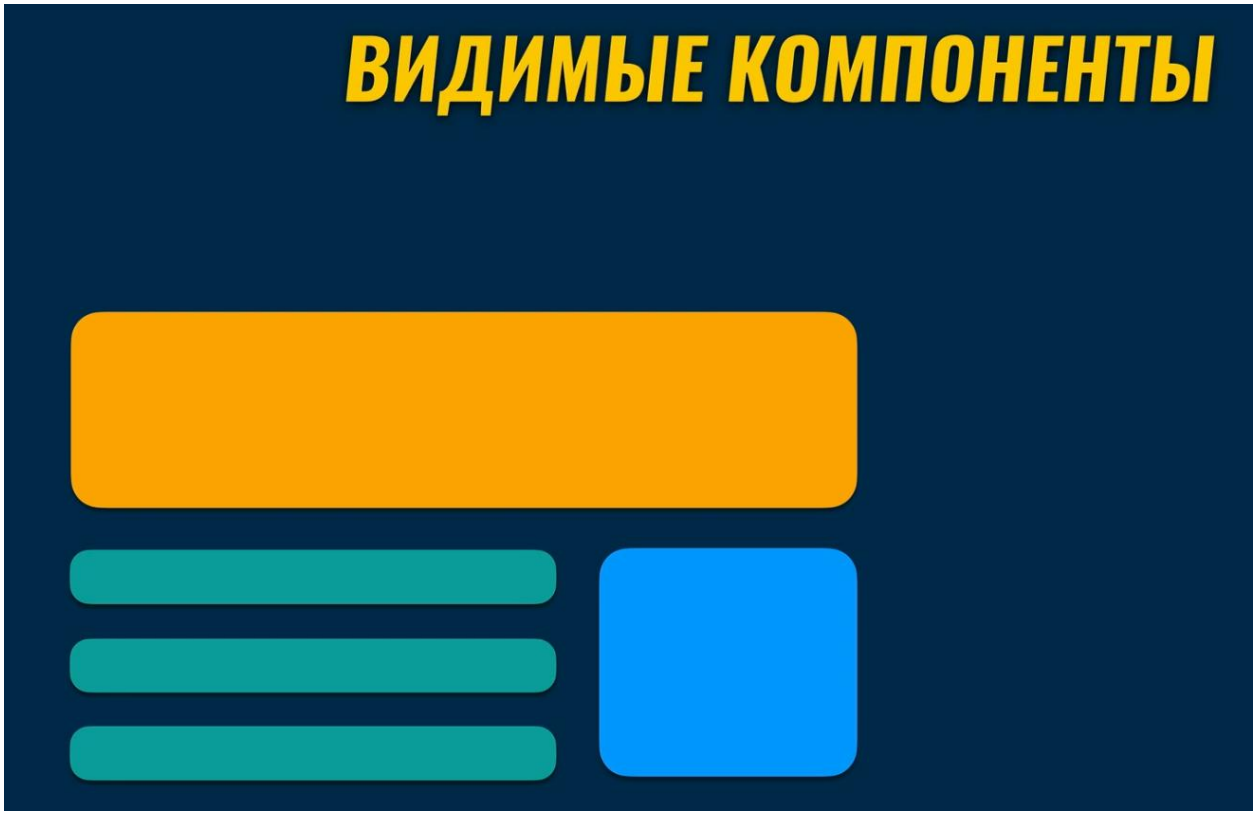
JSX
JavaScript Syntax Extension

СВОЙСТВА
props

СОСТОЯНИЕ
state

ИЕРАРХИЯ КОМПОНЕНТОВ


Корневой
компонент
Компонент,
объединяющий
схожие
компоненты


ВИДИМЫЕ КОМПОНЕНТЫ


ПЕРЕИСПОЛЬЗОВАНИЕ
КОМПОНЕНТОВ


АНАТОМИЯ КОМПОНЕНТА REACT
КОМПОНЕНТ REACT
HTML
CSS
JavaScript

ФУНКЦИОНАЛЬНЫЕ КОМПОНЕНТЫ

```
function HelloWorld() {
  return <h1>Hello World</h1>
}
```

ФУНКЦИОНАЛЬНЫЕ КОМПОНЕНТЫ

```
const HelloWorld = () => {
  return <h1>Hello World</h1>
}
```

КЛАССОВЫЕ КОМПОНЕНТЫ

```
class HelloWorld extends Component {
  render() {
    return <h1>Hello World</h1>
  }
}
```

JAVASCRIPT SYNTAX EXTENSION (JSX)

```
<Card
  style={{
    backgroundColor: `rgb(${initialColor}, ${opacity})`,
  }}
  className="m-2"
>
  <Card.Img
    variant="top"
    style={
      imageLoaded
        ? { opacity: 1, transition: 'opacity 2s ease-in-out' }
        : { opacity: 0 }
    }
    src={image}
    alt={title}
    onLoad={() => setImageLoaded(true)}
  />
</Card>
```

Похоже на CSS

Похоже на HTML

JavaScript

JSX - СИНТАКСИЧЕСКАЯ НАДСТРОЙКА НАД JS

```
React.createElement(Card, {
  style: {
    backgroundColor: `rgb(${initialColor}, ${opacity})`
  },
  className: "m-2"
}, React.createElement(Card.Img, {
  variant: "top",
  style: imageLoaded ? {
    opacity: 1,
    transition: 'opacity 2s ease-in-out'
  } : {
    opacity: 0
  },
  src: image,
  alt: title,
  onLoad: () => setImageLoaded(true)
}));
```

ПОЛЬЗОВАТЕЛЬСКИЕ И ВСТРОЕННЫЕ КОМПОНЕНТЫ В JSX

Встроенный компонент h1

```
return (
  <div>
    <h1>JSX в React</h1>
    <p>JSX похож на HTML</p>
    <Footer text={text} />
  </div>
)
```

Пользовательский компонент Footer

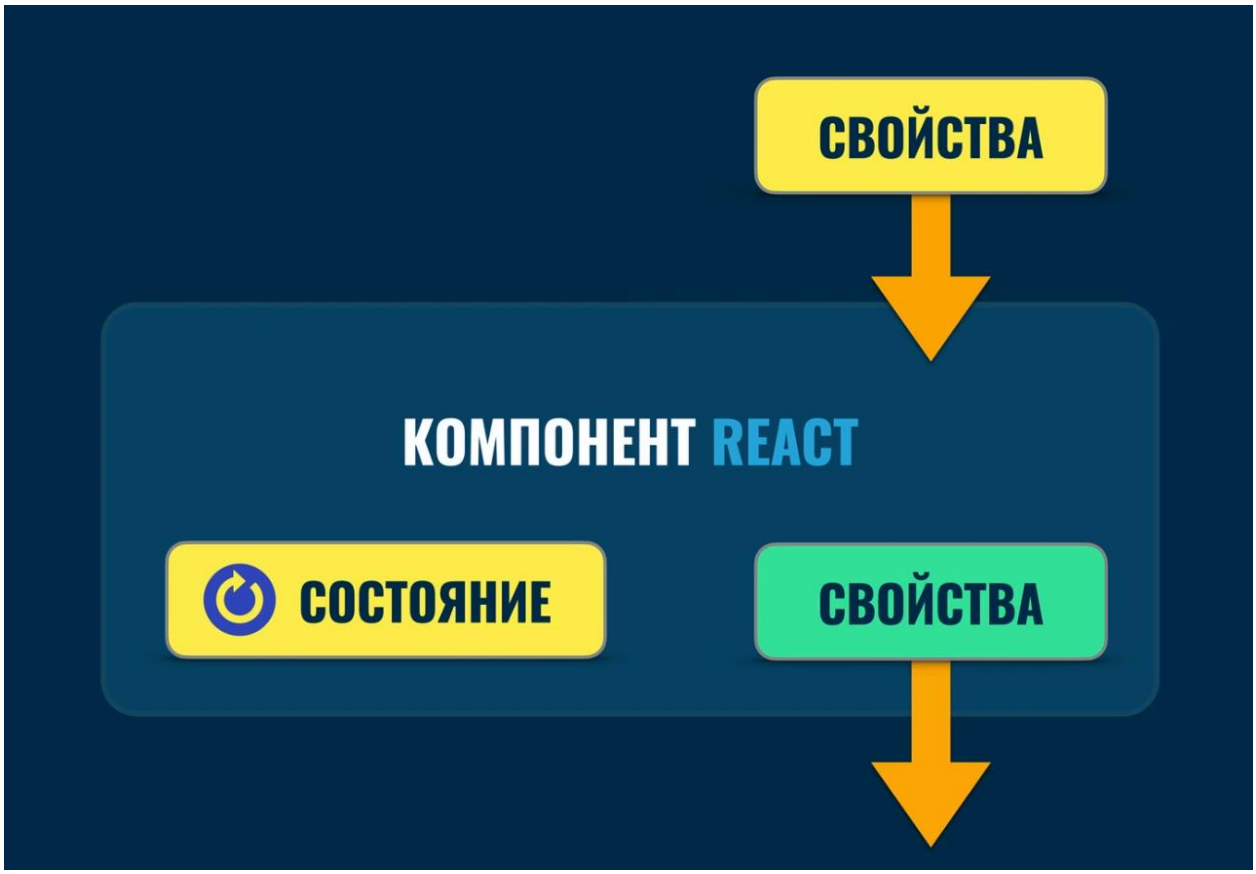
JSX ДОЛЖЕН ИМЕТЬ ОДИН КОРНЕВОЙ ЭЛЕМЕНТ

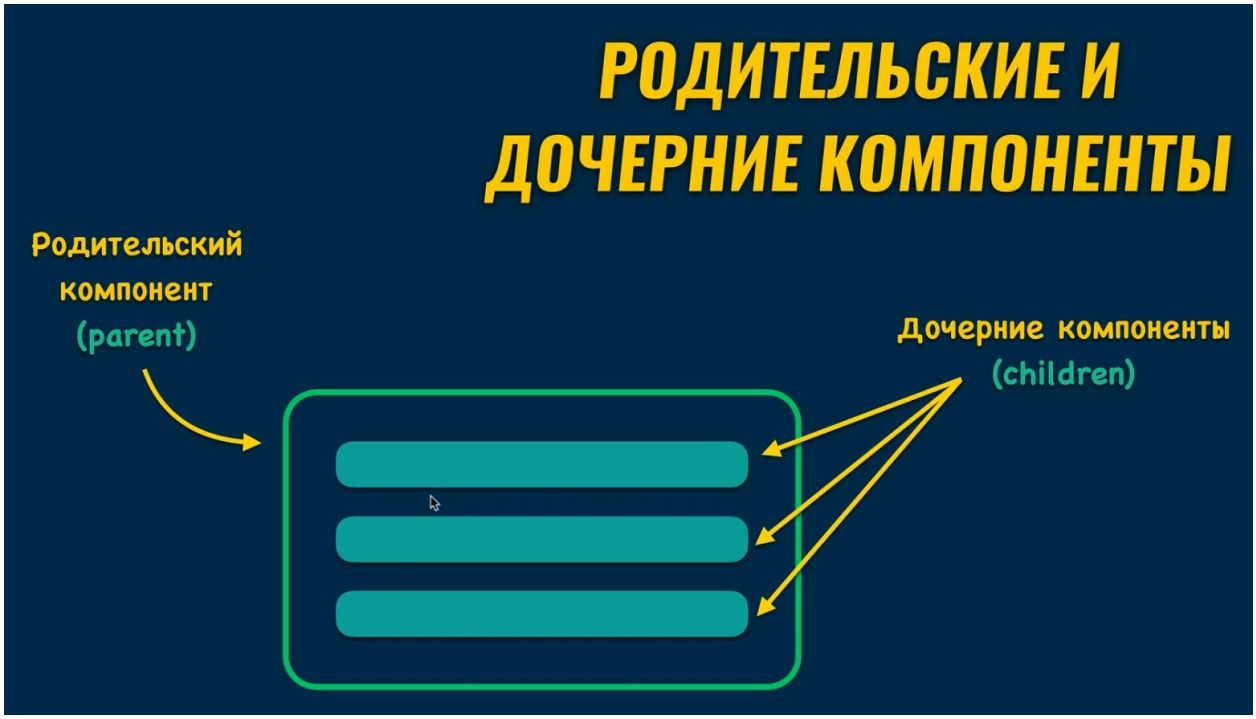
```
return (
  <div>
    <h1>JSX в React</h1>
    <p>JSX похож на HTML</p>
    <Footer text={text} />
  </div>
)
```

Валидный JSX

```
return (
  <h1>JSX в React</h1>
  <p>JSX похож на HTML</p>
)
```

Невалидный JSX


СВОЙСТВА
КОМПОНЕНТ REACT
СОСТОЯНИЕ
СВОЙСТВА

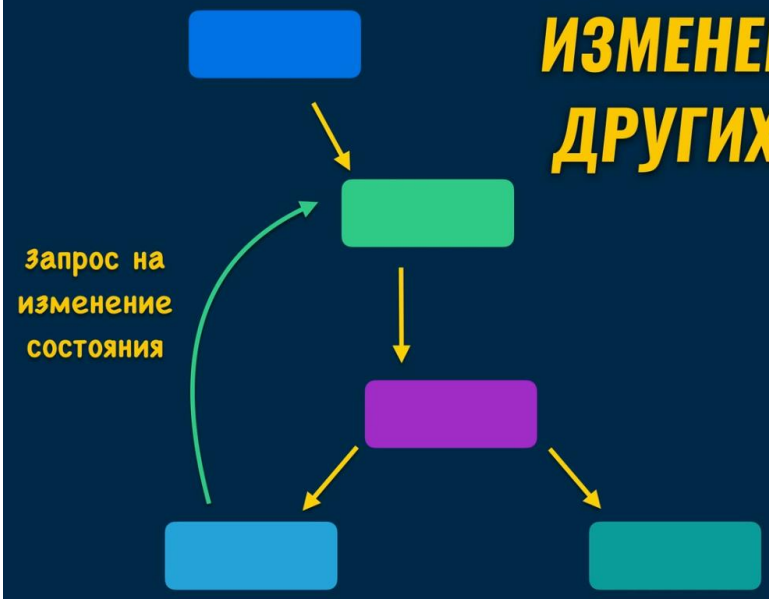

РОДИТЕЛЬСКИЕ И ДОЧЕРНИЕ КОМПОНЕНТЫ
Родительский компонент (parent)
Дочерние компоненты (children)

КОМПОНЕНТ НЕ ДОЛЖЕН ИЗМЕНЯТЬ СОБСТВЕННЫЕ СВОЙСТВА

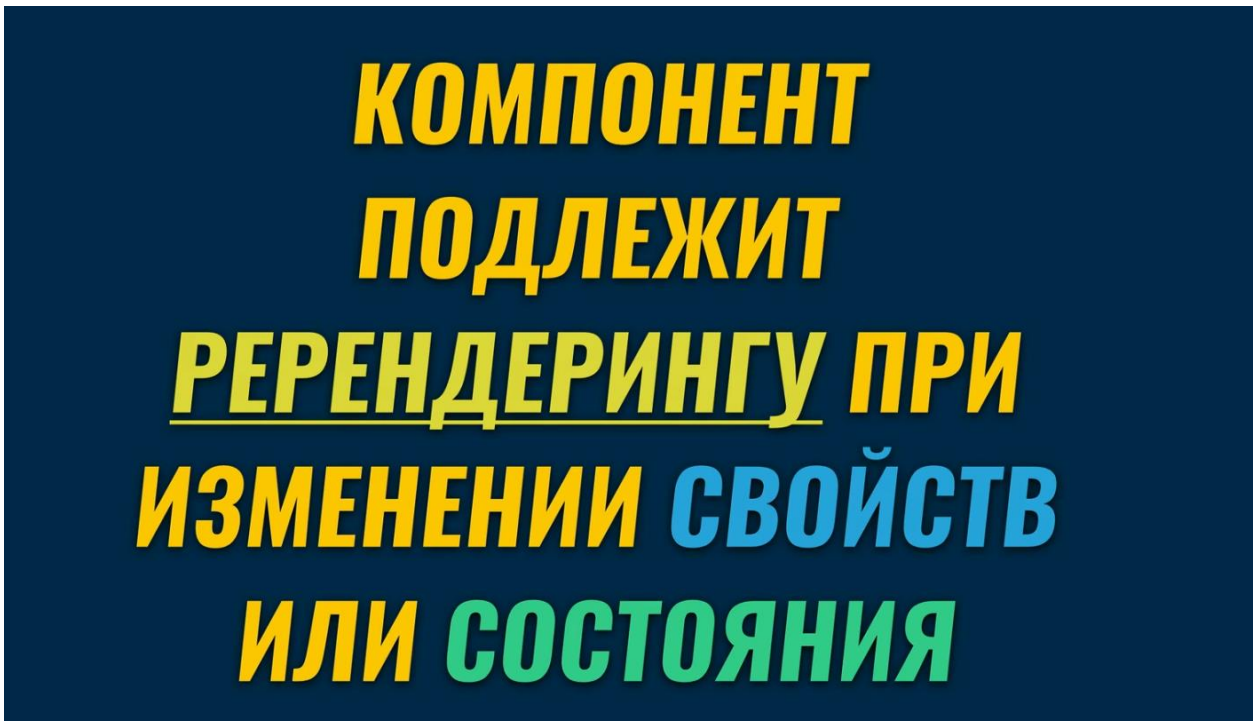
КОМПОНЕНТ МОЖЕТ ИЗМЕНЯТЬ СОБСТВЕННОЕ СОСТОЯНИЕ

КОМПОНЕНТ НЕ МОЖЕТ ИЗМЕНЯТЬ СОСТОЯНИЕ ДРУГИХ КОМПОНЕНТОВ

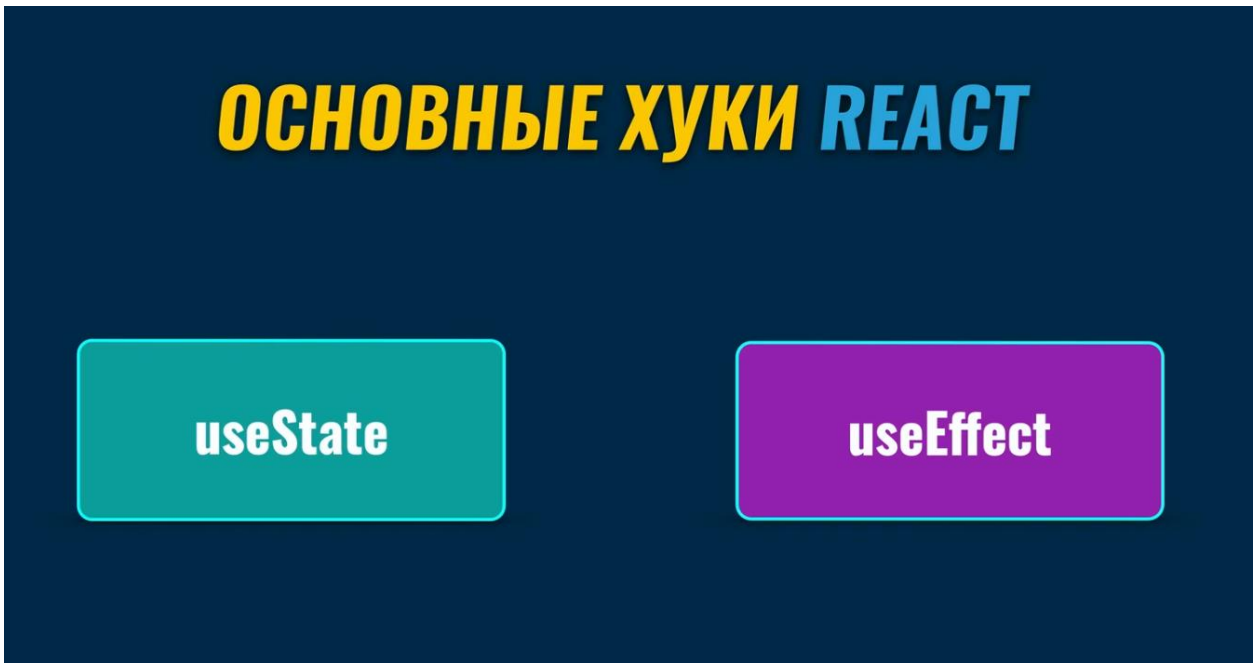
НО МОЖНО ВЛИЯТЬ НА ИЗМЕНЕНИЕ СОСТОЯНИЯ ДРУГИХ КОМПОНЕНТОВ



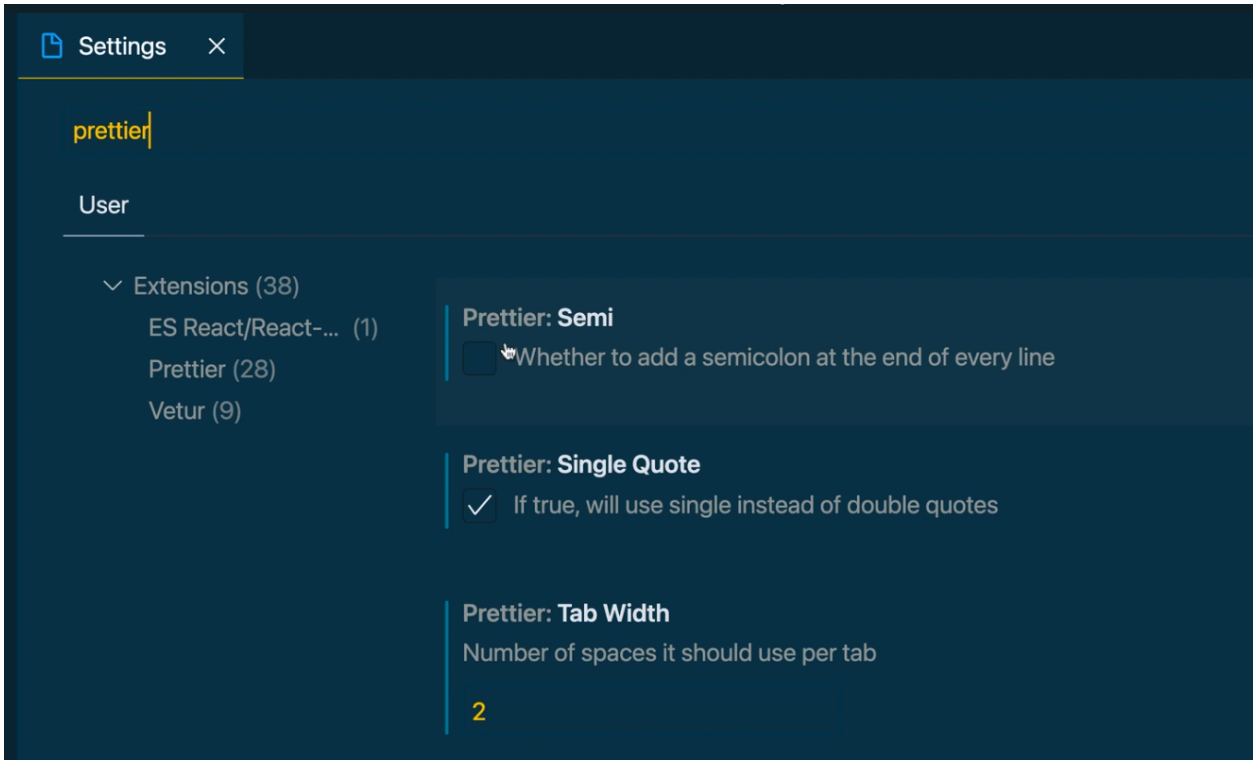
МОЖНО ПЕРЕДАВАТЬ ЧАСТЬ СВОИХ СВОЙСТВ И СОСТОЯНИЯ ДОЧЕРНИМ КОМПОНЕНТАМ В ВИДЕ СВОЙСТВ



Сам React этого не делает. Это выполняет другая библиотека: ReactDom – для Веб-приложений и React Native (RN) – для мобильных приложений


ОСНОВНЫЕ ХУКИ REACT
useState
useEffect

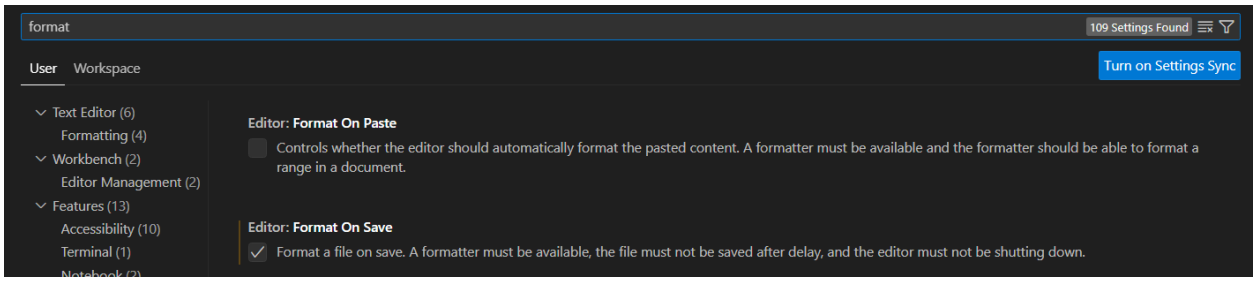
Настройка Prettier



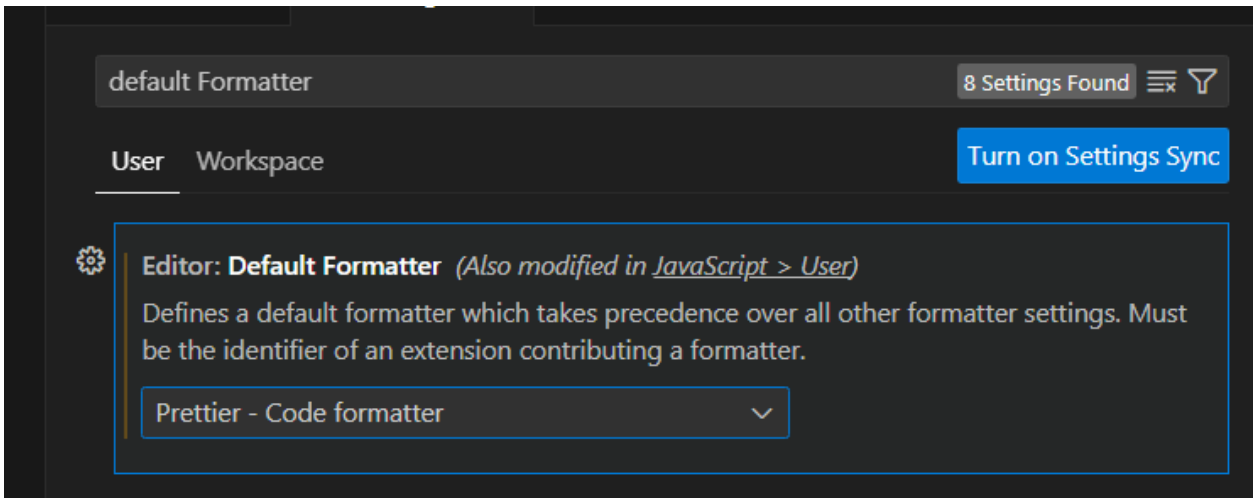
Semi – автоматической выставление «;» в конце строки кода

Single Quote – использование одинарных кавычек

Tab Width - табуляция



Format on Save – применить автоматическое форматирование кода при сохранении

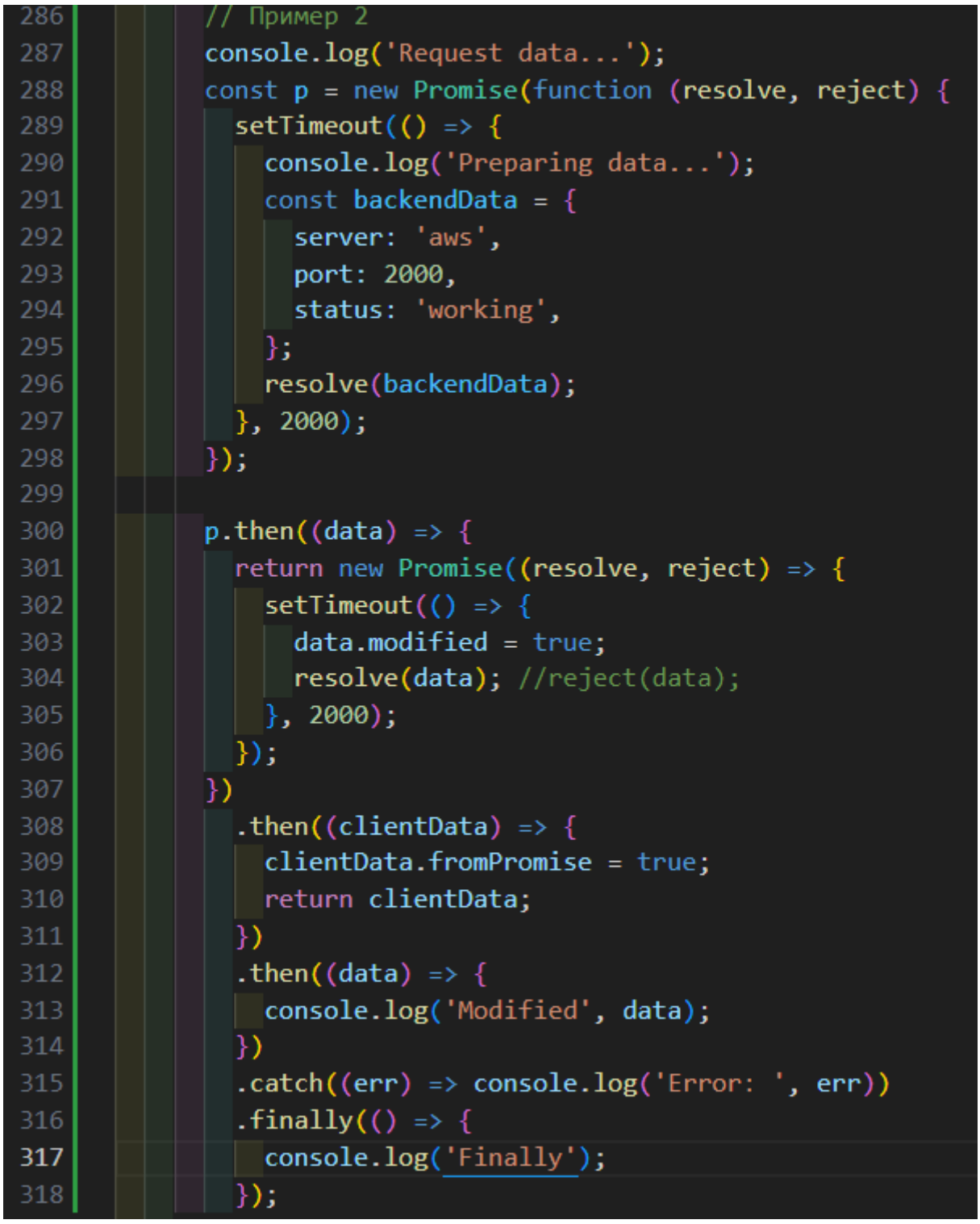



```
// ======================================================
// Promise

// Пример 1
let a = 7;
console.log(a);

let b = new Promise(function (resolve, reject) {
  setTimeout(() => {
    resolve((a = 99));
  }, 2000);
});

b.then(function () {
  console.log(a);
}).catch(console.error);
```



```
// Пример 2
console.log('Request data...');
const p = new Promise(function (resolve, reject) {
  setTimeout(() => {
    console.log('Preparing data...');
    const backendData = {
      server: 'aws',
      port: 2000,
      status: 'working',
    };
    resolve(backendData);
  }, 2000);
});

p.then((data) => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      data.modified = true;
      resolve(data); //reject(data);
    }, 2000);
  });
})
  .then((clientData) => {
    clientData.fromPromise = true;
    return clientData;
  })
  .then((data) => {
    console.log('Modified', data);
  })
  .catch((err) => console.log('Error: ', err))
  .finally(() => {
    console.log('Finally');
  });
```

Модифицируем код, что бы ветка cath заработала: строку resolve(data); во втором промисе заменим на reject(data);

```
// Promise.all и Promise.race

const sleep = (ms) => {
  return new Promise((resolve) => {
    setTimeout(() => resolve(), ms);
  });
};

// sleep(2000).then(() => console.log('After 2 sec'));
// sleep(3000).then(() => console.log('After 3 sec'));

Promise.all([sleep(2000), sleep(5000)]).then(() => {
  console.log('All promeses');
});

Promise.race([sleep(2000), sleep(5000)]).then(() => {
  console.log('Race promeses');
});
```

```
// Пример. Опреление победителя в автогонках

class Car {
  constructor(mark, model, speed) {
    this.mark = mark;
    this.model = model;
    this.speed = speed;
    this.time = Math.floor(Math.random() * 10000) - this.speed;
  }
}

const result = [];

const competitors = (car) => {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log(
        'Car',
        car.mark,
        car.model,
        'finished in',
        car.time,
        'ms'
      );
      result.length == 0 ? (car.isWinner = true) : (car.isWinner = false);
      result.push(car);
      resolve();
    }, car.time);
  });
};

let race = [
  competitors(new Car('ford', 'mustang', 400)),
  competitors(new Car('mazda', 'rx-7', 350)),
  competitors(new Car('BMW', 'x5', 300)),
];

Promise.all(race).then(() => {
  console.table(result);
});
```

```
// async/await

const delay = (ms) => {
  return new Promise((r) => setTimeout(() => r(), ms));
};

// delay(2000).then(() => console.log('2 sec'));

const url = 'https://jsonplaceholder.typicode.com/todos';

function fetchTodos() {
  console.log('Fetch todo started ...');
  return delay(2000)
    .then(() => fetch(url))
    .then((response) => response.json());
}

fetchTodos()
  .then((data) => {
    console.log('Data: ', data);
  })
  .catch((e) => console.error(e));

async function fetchAsyncTodos() {
  console.log('Fetch todo started ...');
  try {
    await delay(2000);
    const response = await fetch(url);
    const data = await response.json();
    console.log('Data ', data);
  } catch (e) {
    console.error(e);
  } finally {
    console.log('Finally');
  }
}

fetchAsyncTodos();

//https://jsonplaceholder.typicode.com/
```

```
421  /********** Замыкание  -  Closure ********/
422
423  function outer() {
424    let x = 10;
425    function inner() {
426      console.log(x);
427    }
428    return inner;
429  }
430
431  let closureFn = outer();
432  closureFn();
433
434  function createCalcFn(n) {
435    return function () {
436      console.log(100 * n);
437    };
438  }
439
440  const calc = createCalcFn(42);
441  calc();
442
443  function createIncrementor(n) {
444    return function (m) {
445      return n + m;
446    };
447  }
448
449  const addOne = createIncrementor(12);
450  console.log(addOne(3));
451
452  const addTen = createIncrementor(10);
453  console.log(addTen(100));
454
455  function urlGeneration(domain) {
456    return function (url) {
457      return `https://${url}.${domain}`;
458    };
459  }
460
461  const comUrl = urlGeneration('com');
462  console.log(comUrl('google'));
463  console.log(comUrl('VK'));
464
```

```
465  // самостоятельно: написать функцию ruUrl, которая замыкает функцию urlGeneration() и выводи в консоль сообщения типа
466  // https://VK.ru
467  // https://netflix.ru
468  // https://google.ru
469  //
```