

Цели занятия

- научиться создавать компоненты на React;
- узнать как работают свойства компонента;
- научиться разделять код на компоненты;

Полезные материалы

- [Компоненты и их свойства](#)
- [Подробнее про деструктуризацию](#)
- [Углубленное изучение JSX](#)

Компоненты и пропсы

Компоненты позволяют разбить интерфейс на независимые части, про которые легко думать в отдельности. Их можно складывать вместе и использовать несколько раз.

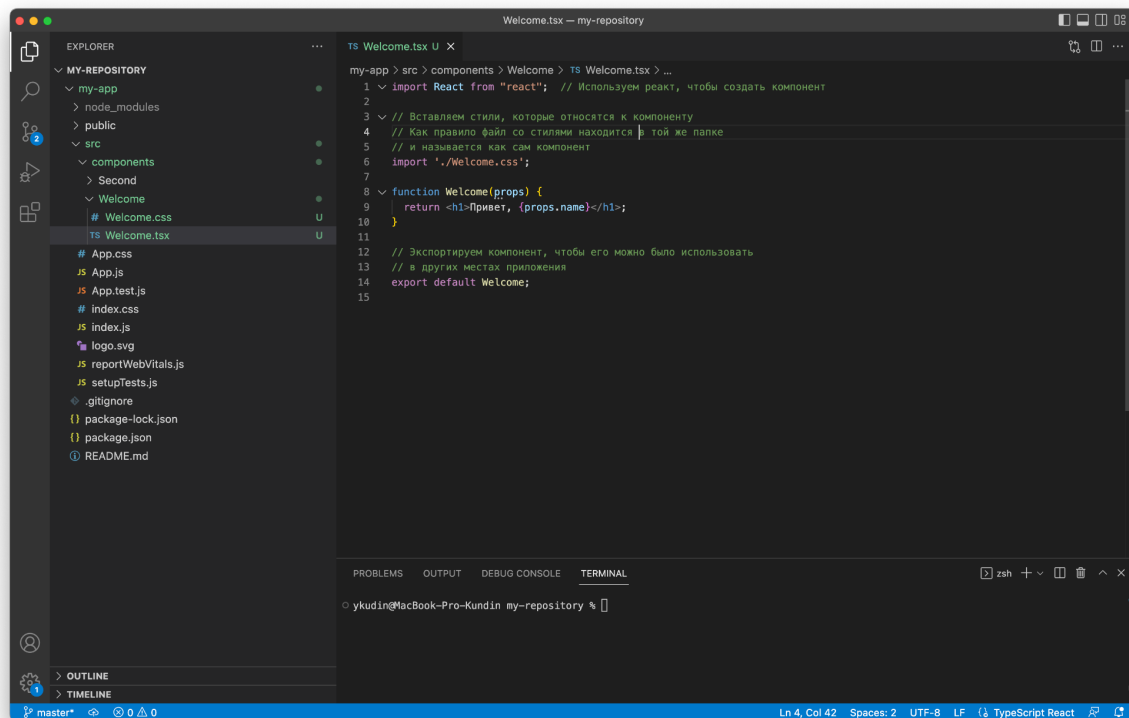
Во многом компоненты ведут себя как обычные функции JavaScript. Они принимают произвольные входные данные (так называемые «пропсы») и возвращают React-элементы, описывающие, что мы хотим увидеть на экране.

Проще всего объявить React-компонент как функцию:

```
function Welcome(props) {  
  return <h1>Привет, {props.name}</h1>;  
}
```

Эта функция — компонент, потому что она получает данные в одном объекте («пропсы») в качестве параметра и возвращает React-элемент. Мы будем называть такие компоненты «функциональными», так как они буквально являются функциями.

Ещё компоненты можно определять как классы, но этот считается устаревшим и в рамках курса рассматриваться не будет. Как правило, все компоненты хранят в одной директории `components` и для каждого компонента создают свою директорию, которая называется так же как и компонент. Пример расположения файлов:



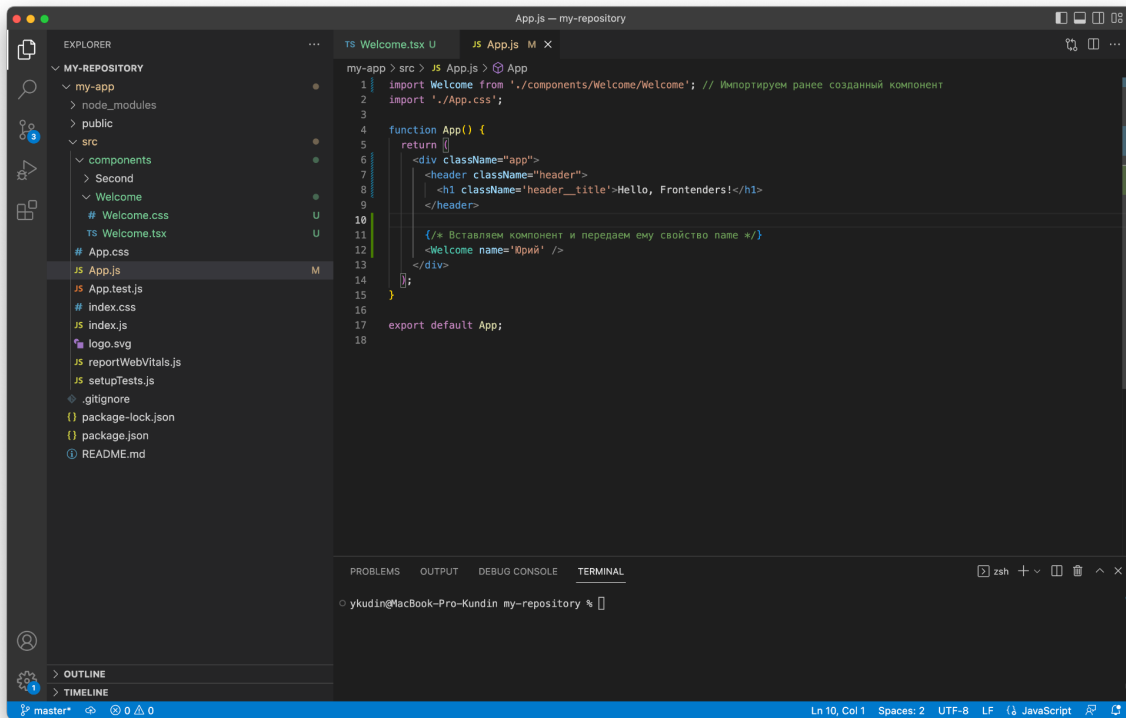
На данном примере показан пример шаблонного компонента, который также подключает необходимые стили. Данную структуру следует запомнить и использовать для создания всех остальных компонентов.

Обратите внимание, что все стили будут добавлены на страницы автоматически в тех местах, где вы используете данный компонент, а CSS или JavaScript не имеют никаких ограничений или отличий от того, что мы уже знаем.

Единственное, что стоит запомнить: для добавления CSS-класса необходимо указывать свойство `className`, а не `class` как в HTML.

Как использовать компонент?

Для этого его необходимо вставить внутрь React-приложения (обычно называется App) и передать ему все необходимые свойства:



Важно: Всегда называйте компоненты с заглавной буквы.

Если компонент начинается с маленькой буквы, React принимает его за DOM-тег. Например, `<div />` это div-тег из HTML, а `<Welcome />` это уже наш компонент Welcome, который должен быть в области видимости.

Чтобы узнать больше про это соглашение, прочитайте [Углублённое изучение JSX](#).

Композиция компонентов

Компоненты могут ссылаться на другие компоненты в возвращённом ими дереве. Это позволяет нам использовать одну и ту же абстракцию — компоненты — на любом уровне нашего приложения. Неважно, пишем ли мы кнопку, форму или целый экран: все они, как правило, представляют собой компоненты в React-приложениях.

Например, компонент App может отрендерить компонент Welcome несколько раз:

```
function Welcome(props) {  
  return <h1>Привет, {props.name}</h1>;  
}  
  
function App() {  
  return (  
    <div>  
      <Welcome name="Алиса" />  
      <Welcome name="Базилио" />  
      <Welcome name="Буратино" />  
    </div>  
  );  
}
```

В приложениях, написанных на React с нуля, как правило, есть один компонент App, который находится на самом верху.

Свойства компонента

Абсолютно каждый компонент может принимать любое количество свойств, которые необходимы разработчику. Свойства приходят в

виде первого аргумента в компонент и обычно называются props (пропсы). Свойством может быть что угодно: число, строка, объект, массив и так далее:

```
function Comment(props) {
  const author = props.author; // Компонент должен иметь свойство
  author в виде объекта
  const text = props.text; // А ещё свойство text, который будет
  просто строкой

  return (
    <div className='comment'>
      {/* Переменные можно вставлять сразу внутрь JSX-разметки */}
      <div className='author'>
        {`Автор: ${author.name}`}
      </div>

      <div className='text'>{text}</div>
    </div>
  );
}
```

После чего такой компонент можно будет использовать следующим образом:

```
function App() {
  return (
    <div>
      <Comment
        author={{ name: 'Юрий' }}
        text='Текст комментария'
      />
    </div>
  )
}
```

Применяем деструктуризацию

Когда компонент принимает несколько свойств, то их запись можно немного упростить с помощью деструктуризации, например, вот так:

```
function Comment(props) {  
  // Взяли оба свойства и присвоили их в переменные  
  const { author, text } = props;  
  
  return ...;  
}
```

Это не является функционалом React, а одна из возможностей JavaScript, которая упрощает наш код и позволяет сократить объем кода. Её можно использовать не только для свойств компонентов, но и для многих других вещей в разработке, подробнее об этом читайте в [документации](#).

Свойства по умолчанию

Периодически бывает такое, что какое-то из свойств может быть необязательным и нам необходимо использовать какое-то значение по умолчанию. Сделать это можно двумя способами, большой разницы между ними нет, поэтому выбирайте на свой вкус:

Вариант №1:

```
function Welcome(props) {  
  // Если не указать свойство name, то по умолчанию будет Гость  
  const { name = 'Гость' } = props;  
  
  return <h1>Привет, {name}</h1>;  
}
```

Вариант №2

```
function Welcome(props) {  
  const { name } = props;  
  
  return <h1>Привет, {name}</h1>;  
}  
  
// То же самое  
Welcome.defaultProps = {  
  name: 'Гость'  
}
```

Специальное свойство children

Очень часто возникает потребность вкладывать компоненты друг в друга, как и простые теги в HTML. Для таких случаев есть необязательное свойство children и давайте сразу рассмотрим его на примере:

```
function Form(props) {  
  const { title, children } = props;  
  
  return (  
    <form className='form'>  
      <div className='form__title'>{title}</div>  
  
      <div className='form__content'>  
        { /* Здесь будет вставлено все содержимое формы */ }  
        {children}  
      </div>  
    </form>  
  )  
}
```

```
    </form>
  )
}

function App() {
  return (
    <Form title='Моя форма'>
      {/* Всё что внутри компонента попадет в children */}
      <input name='phone' placeholder='+7 900 000-00-00' />
      <button type='submit'>Отправить</button>
    </Form>
  )
}
```

Именно эта возможность позволяет нам вкладывать компоненты друг в друга и получать приложения или веб-сайты любой сложности и гибкости.

Пропсы можно только читать

Компонент никогда не должен что-то записывать в свои пропсы — вне зависимости от того, функциональный он или классовый.

Возьмём для примера функцию `sum`:

```
function sum(a, b) {
  return a + b;
}
```

Такие функции называют [«ЧИСТЫМИ»](#), потому что они не меняют свои входные данные и предсказуемо возвращают один и тот же результат для одинаковых аргументов.

А вот пример нечистой функции — она записывает данные в свои же аргументы:

```
function withdraw(account, amount) {  
  account.total -= amount;  
}
```

React достаточно гибкий, но есть одно правило, которое нельзя нарушать:

React-компоненты обязаны вести себя как чистые функции по отношению к своим пропсам.

Конечно, интерфейсы приложений обычно изменяются с течением времени. Поэтому на следующих занятиях мы с вами научимся изменять так как называемое “состояние” компонента и делать компоненты динамичными.