React

Особенности React

— "VC" in "MVC". Это только *библиотека рендеринга*

— Компонентный подход

— JSX

— Virtual DOM

— "One-way" Data Flow

JSX

— Синтаксис, позволяющий добавить разметку прямо в js-код

— JSX - выражение; в JSX можно добавить другие js-выражения

— Позволяет разделять код не по *технологиям*, а по *задачам*

— Синтаксический сахар над `React.createElement()`

```
const element = <h1>Hello, {formatName(user)}!</h1>;

function getGreeting(user) {
  return user
    ? <h1>Hello, {formatName(user)}!</h1>
    : <h1>Hello, Stranger.</h1>;
}
```

```
class Hello extends React.Component {
 render() {
  return <div>Hello {this.props.who}</div>;
 }
}

ReactDOM.render(
  <Hello who="World" />, document.body
);
```

```
class Hello extends React.Component {
 render() {
  return React.createElement(
   "div", null, "Hello ", this.props.who);
 }
}
ReactDOM.render(
 React.createElement(Hello, {who: "World"}),
 document.body);
```

Элементы

— Элемент - самый мелкий строительный блок React

— Элементы неизменяемы: отдельный элемент UI в отдельный элемент времени

```
function tick() {
  const element = (
    <div>
      <h4>Hello, world!</h4>
      <h5>It is
        {new Date().toLocaleTimeString()}.
      </h5>
    </div>
  );
  ReactDOM.render(
    element,
    document.getElementById("root")
  );
}

setInterval(tick, 1000);
```

Hello, world!

It is 3:35:39 PM.

Virtual DOM

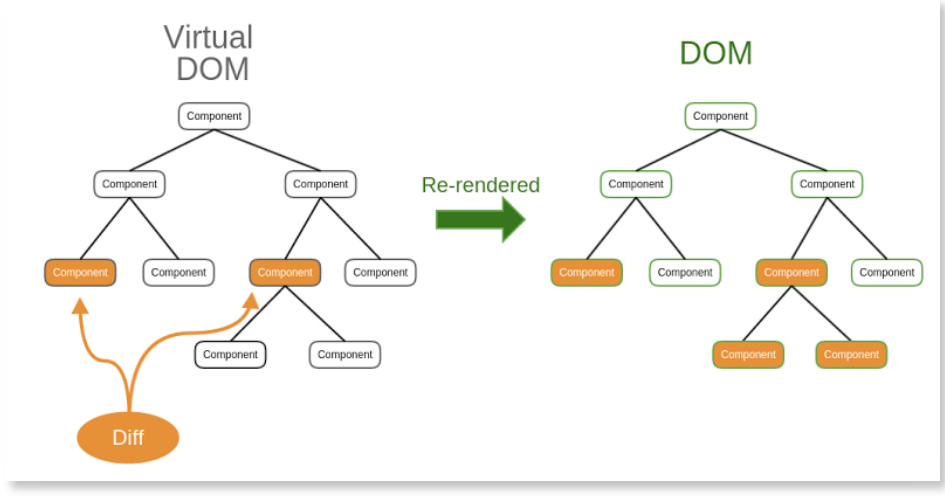
— *Virtual DOM (VDOM)* - структура данных, уровень абстракции над UI

— Операции с VDOM очень быстрые (в отличие от DOM)

— При каждом изменении React проверяет какие части VDOM изменились (reconciliation)

— Перерисовываются (re-render) только измененные части DOM

— *Позволяет писать UI (HTML) как функцию от состояния*

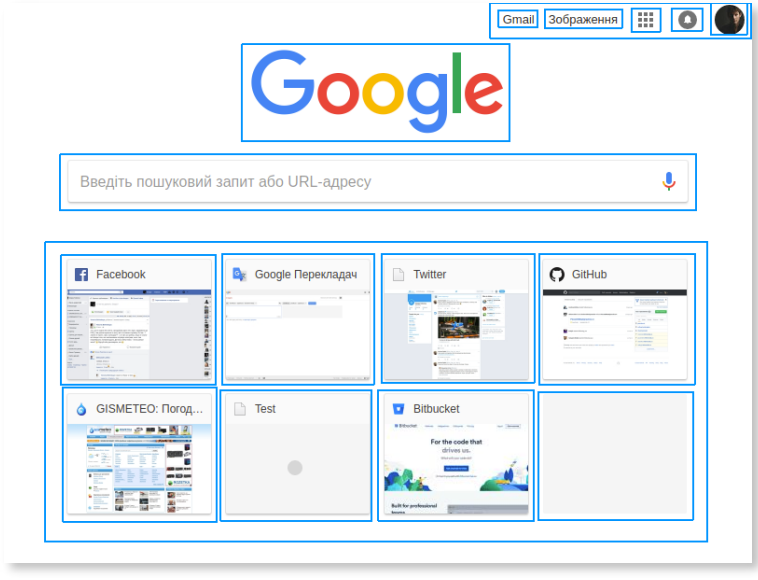


Компоненты

— Компонент - отдельный, независимый, переиспользуемый элемент UI

— Может содержать в себе другие компоненты (композиция)

— Концептуально это функция, возвращающая элемент UI



Компоненты

— Stateless (functional) component - обычная функция

— Атрибуты JSX попадают в отдельный объект - `props`

— Имя компонента должно начинаться с заглавной буквы

— Properties *не должны изменяться* (read-only)

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

ReactDOM.render(
  <Welcome name="Sara" />,
  document.getElementById("root")
);
```

Композиция компонентов

```jsx
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <img className="Avatar"
          src={props.author.avatarUrl}
          alt={props.author.name}
        />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="text">
        {props.text}
      </div>
    </div>
  );
}
```

```jsx
const Avatar = (props) => (
  <img className="Avatar"
    src={props.user.avatarUrl}
    alt={props.user.name}
  />
);
```

```jsx
const UserInfo = (props) => (
  <div className="UserInfo">
    <Avatar user={props.user} />
    <div className="UserInfo-name">
      {props.user.name}
    </div>
  </div>
);
```

```jsx
const Comment = (props) => (
  <div className="Comment">
    <UserInfo user={props.author} />
    <div className="Comment-text">
      {props.text}
    </div>
  </div>
);
```

Stateful компоненты

— Класс, наследующийся от `React.Component` и реализующий метод `render()`

```
const Clock = (props) => (
  <div>
    <h1>Hello, world!</h1>
    <h2>
      It is {props.date.toTimeString()}.
    </h2>
  </div>
);

function tick() {
  ReactDOM.render(
    <Clock date={new Date()} />,
    document.getElementById("root")
  );
}

setInterval(tick, 1000);
```

```
class Clock extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is
          {this.props.date.toTimeString()}.
        </h2>
      </div>
    );
  }
}

function tick() {
  ReactDOM.render(
    <Clock date={new Date()} />,
    document.getElementById("root")
  );
}
setInterval(tick, 1000);
```

Stateful компоненты

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = { date: new Date() };
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is
          {this.state.date.toTimeString()}.
        </h2>
      </div>
    );
  }
}

ReactDOM.render(
  <Clock />,
  document.getElementById("root")
);
```

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = { date: new Date() };
  }

  componentDidMount() {
    this.timerID = setInterval(
      () => this.tick(), 1000
    );
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  tick() {
    this.setState({ date: new Date() });
  }

  render() { ... }
}
```

`.state` и `setState()`

— В `state` находятся данные, участвующие в `render()`

— `state` обновляется *только* через `setState`

— `setState()` работает асинхронно

— `setState()` производит слияние изменений в `state`

```
// Неправильно
this.setState({
  counter: this.state.counter + this.props.increment
});

// Правильно
this.setState((prevState, props) => ({
  counter: prevState.counter + props.increment
}));
```

`state`, `props` и прочие ~~неприятности~~ данные класса

— `props` - read-only "параметры" компонента, поступают извне

— `state` - состояние компонента, изменяется через `setState`

— При изменении `props` и `state` React вызывает `render()` компонента

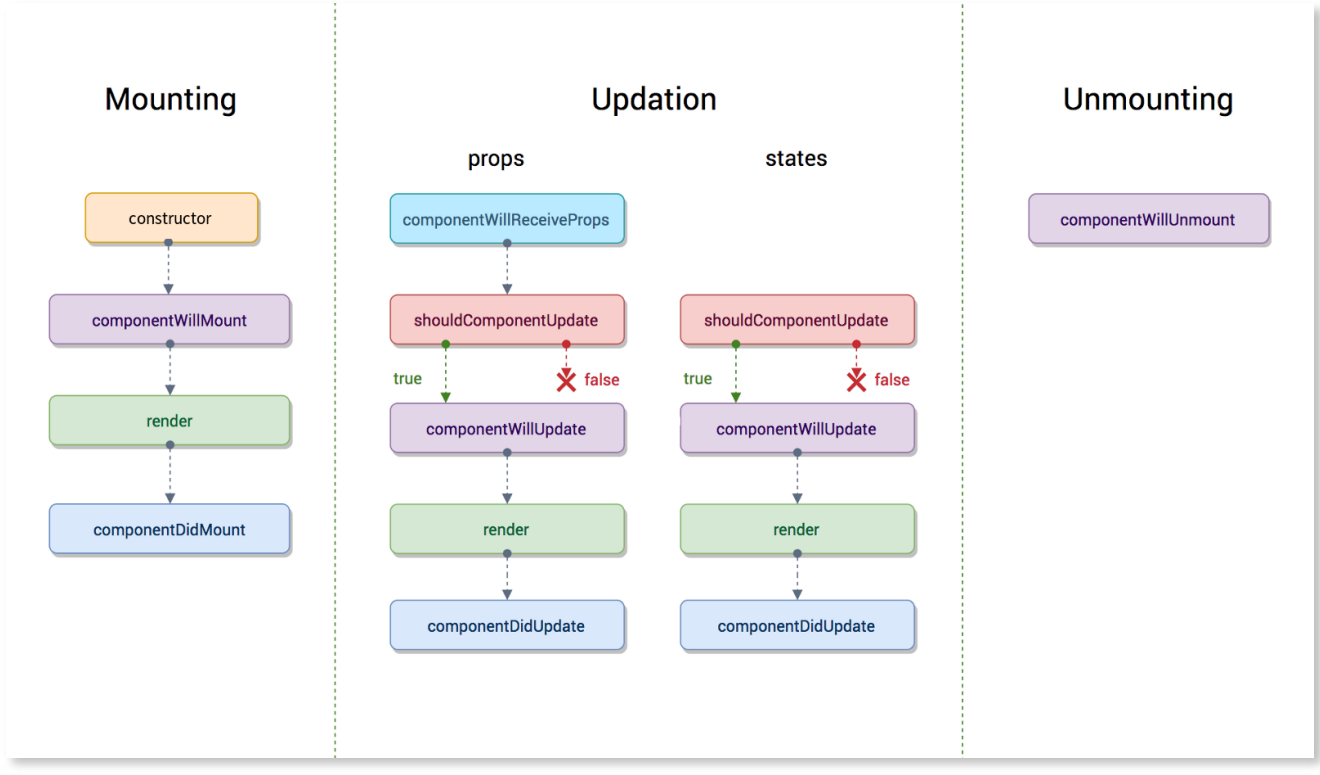
— Остальные данные - обычные свойства компонента

```
class CustomButton extends React.Component {
  constructor(props) {
      super(props);
      this.state = { active: false };
  }
}

CustomButton.defaultProps = {
  color: "blue"
};

<CustomButton />
// CustomButton c this.props.color === "blue"
```

Жизненный цикл компонента



События

— Обработчики прописываются прямо в JSX

— Обычно обработчик - это метод компонента

— В обработчик передается `SyntheticEvent` - обертка над событием браузера

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = { label: "OFF" };
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(prevState => ({
      label: prevState.label === "OFF" ? "ON" : "OFF"
    }));
  }

  render() {
    return <button onClick={this.handleClick}>{this.state.label}</button>;
    // {e => this.handleClick(e)} или {this.handleClick.bind(this)}
  }
}
```

JSX посложнее

— JSX позволяет смешивать JS-код и разметку:

— Вывод того или иного элемента по условию

— Вывод списков с помощью `map`

— Ключи (`keys`) помогают отслеживать изменения элементов массива

— Ключ - уникальный id элемента в *контексте этого списка*

```
function NumberList(props) {
  return (
    <ul>{ props.numbers.map((number) =>
      <li key={number.toString()}>
        {number}
      </li>) }
    </ul>
  );
}

<NumberList numbers={[1, 2, 3, 4, 5]} />
```

Формы

```
class MyForm extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      textValue: "",
      textareaValue: "",
      selectValue: []
    };

    this.handleInputChange =
      this.handleInputChange.bind(this);
  }

  handleInputChange(e) {
    this.setState({
      [e.target.name]: value
    });
  }
```

```
  render() {
    return (
      <form>
        <input name="textValue"
          type="text"
          value={this.state.textValue}
          onChange={this.handleInputChange} />

        <textarea name="textareaValue"
          value={this.state.textareaValue}
          onChange={this.handleInputChange} />

        <select name="selectValue"
          multiple={true}
          value={selectValue}
          onChange={this.handleInputChange}>
          <option value="1">1</option>
          . . .
        </select>
      </form>
    );
  }
}
```

Совместный доступ к данным

— Часто разные компоненты используют одни и те же данные

— React предлагает хранить такие данные в `state` родителя

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = { temp: 0 };
  }
  onChange(temp) {
    this.setState({ temp: Number(temp) });
  }
  render() {
    return (
      <div className="App">
        <TemperatureInput
          temp={this.state.temp}
          onChange=
            {this.onChange.bind(this)} />
        <BoilingVerdict
            temp={this.state.temp} />
      </div>
    );
  }
}
```

```
const TemperatureInput = props => (
  <input value={props.temp}
         onChange={e =>
            props.onChange(e.target.value)}
  />
);

const BoilingVerdict = props =>
  (props.temp >= 100)
    ? <p>The water would boil.</p>
    : <p>The water would not boil.</p>;
```

Композиция компонентов

— Часто компонент служит "оберткой" для других компонентов

— Вся JSX-разметка которая находится внутри тегов компонента, попадает в специальное свойство `props.children`

```
const FancyBorder = props => (
  <div
    className={"FBorder-" + props.color}>
    {props.children}
  </div>
);

const WelcomeDialog = props => (
  <FancyBorder color="blue">
    <h1 className="Dialog-title">
      Welcome
    </h1>
    <p className="Dialog-message">
      Thank you for visiting us!
    </p>
  </FancyBorder>
);
```

```
const SplitPane = props => (
  <div className="SplitPane">
    <div className="SplitPane-left">
      {props.left}
    </div>
    <div className="SplitPane-right">
      {props.right}
    </div>
  </div>
);

<SplitPane
  left={<Contacts />}
  right={<Chat />}
/>
```

`propTypes`

— Простой способ добавить проверку типов `props`, которые поступают в компонент

```
import PropTypes from 'prop-types';

class Greeting extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.name}</h1>
    );
  }
}

Greeting.propTypes = {
  name: PropTypes.string
};
```