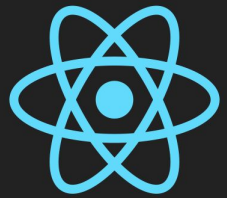
ReactJS

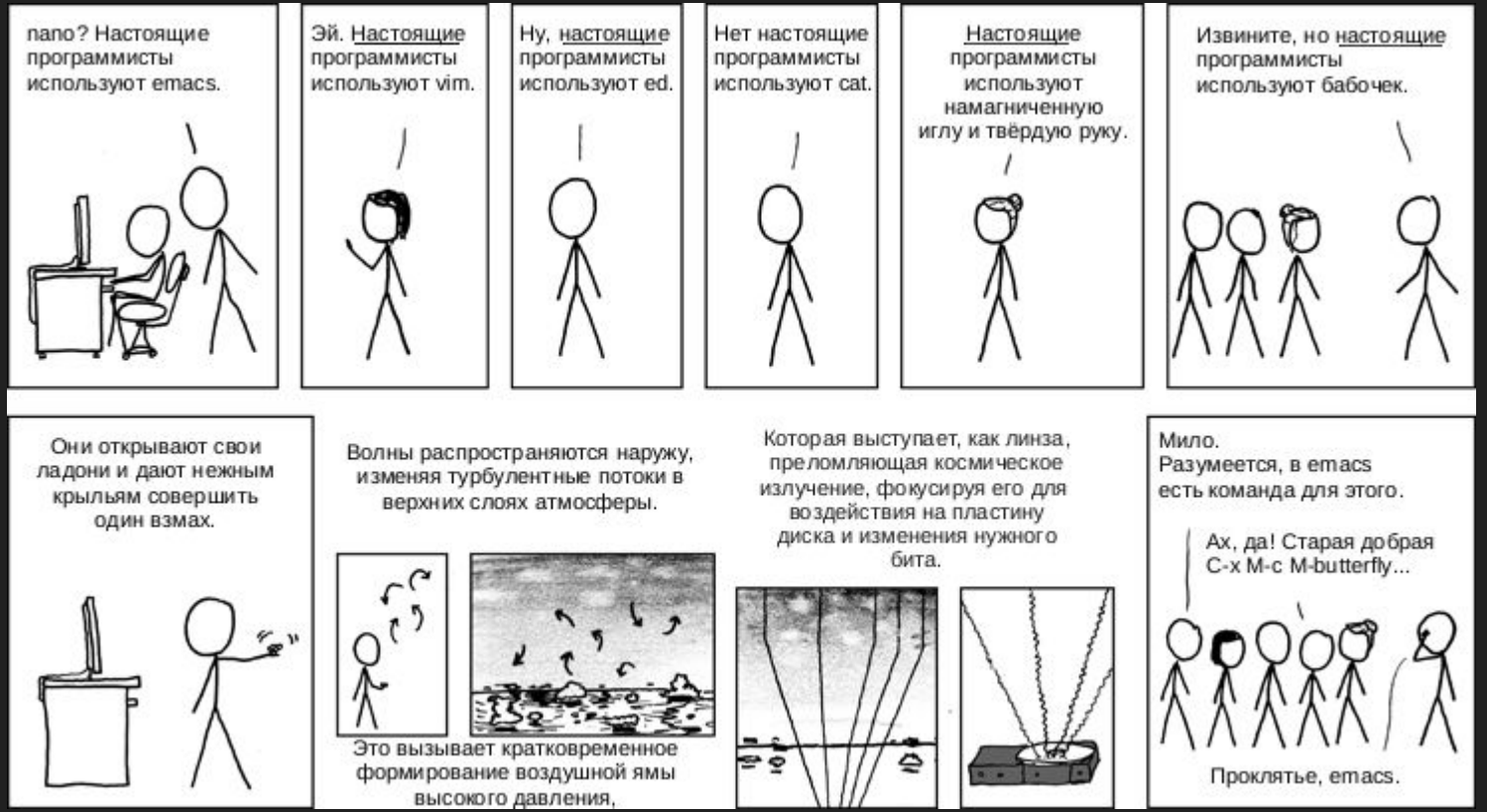
bleizard@cs.ifmo.ru

React - js библиотека для построения UI

- open-source (https://github.com/facebook/react)
- by Facebook
- Текущая версия ~~v15.5.4~~ ~~v15.6.1~~ ~~v16.0.0~~ ~~v16.1.0~~ ~~v16.1.1~~ v16.2.0
- Используют: facebook, instagram, periscope, imdb, twitch, uber, bbc и др.



Зачем?


nano? Настоящие программисты используют emacs.
Эй. Настоящие программисты используют vim.
Ну, настоящие программисты используют ed.
Нет настоящие программисты используют cat.
Настоящие программисты используют намагниченную иглу и твёрдую руку.
Извините, но настоящие программисты используют бабочек.
Они открывают свои ладони и дают нежным крыльям совершить один взмах.
Волны распространяются наружу, изменяя турбулентные потоки в верхних слоях атмосферы.
Это вызывает кратковременное формирование воздушной ямы высокого давления,
Которая выступает, как линза, преломляющая космическое излучение, фокусируя его для воздействия на пластину диска и изменения нужного бита.
Мило.
Разумеется, в emacs есть команда для этого.
Ах, да! Старая добрая C-x M-c M-butterfly...
Проклятье, emacs.

Зачем?

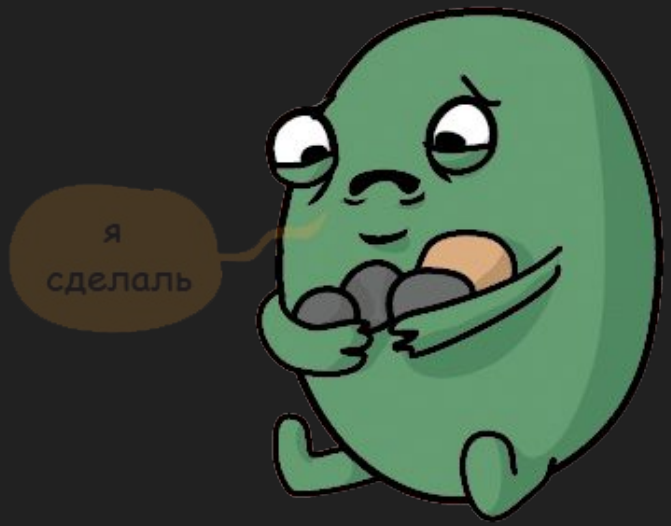
- Декомпозиция
- Переиспользование
- Желание мыслить в ООП стиле
- Делегирование обязанностей по управлению DOM
- Единый интерфейс взаимодействия
- Единый жизненный цикл компонента

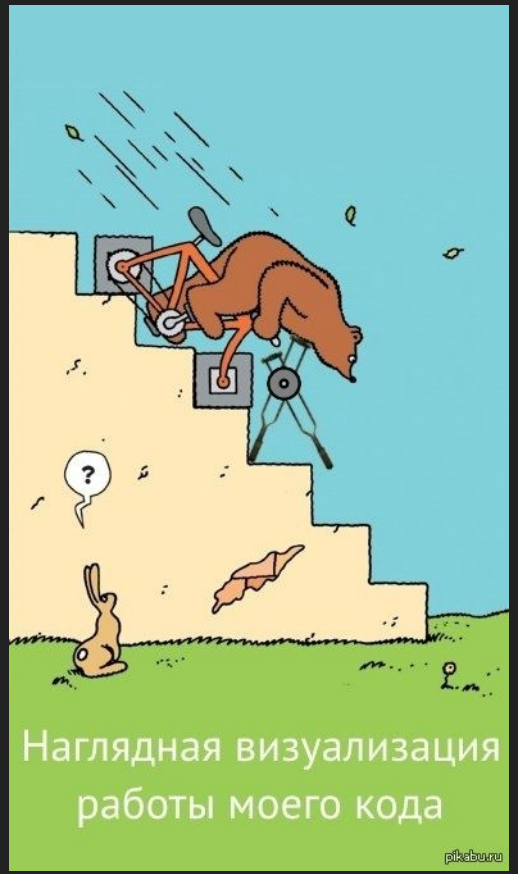
Зачем?

Фреймворки для слабаков, напиши всё сам!

Только vanilla js, только хардкор!

Зачем?

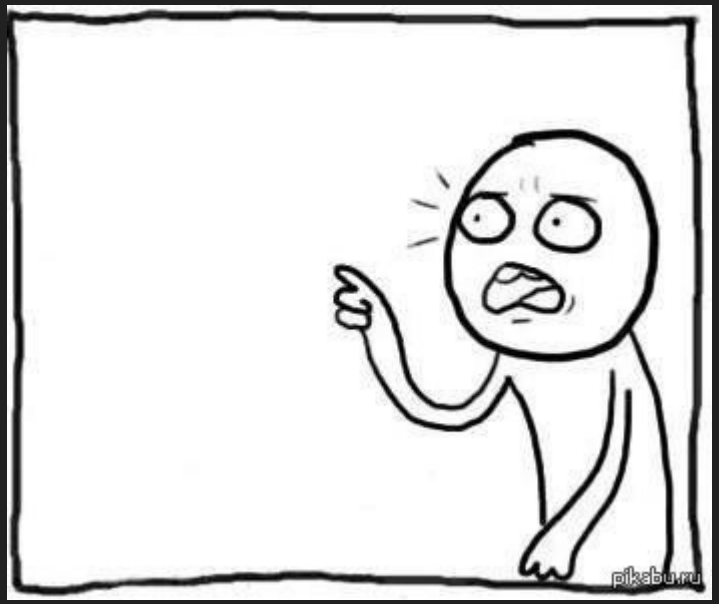

я
сделаль


?
Наглядная визуализация
работы моего кода
pikabu.ru

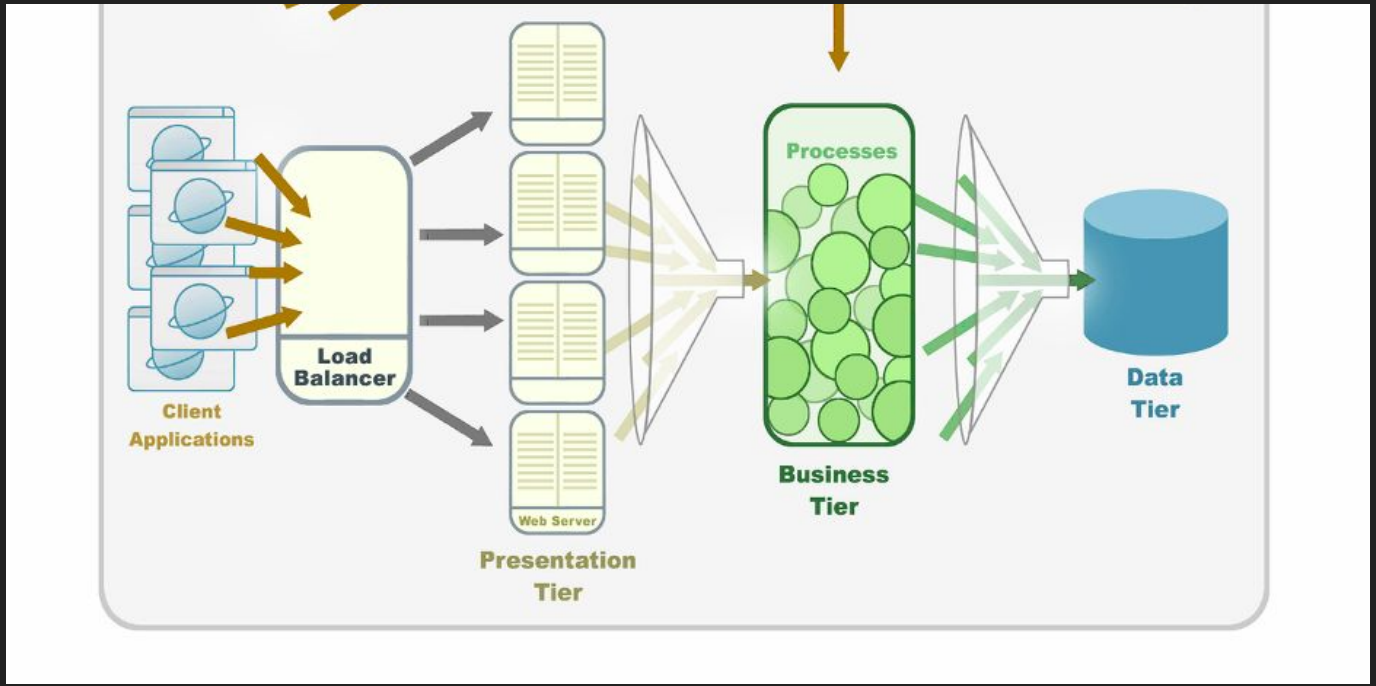
Но есть же JQuery

JQuery делает проще API для:

- работы с DOM
- использования AJAX
- обработки событий
- анимаций



Трехуровневая архитектура



SPA - Single Page Application

Основные идеи:

- Оффлайн режим
- Перенос части работы с сервера на клиента
- Единый html документ
- Routing через JS (html5 history api)
- ~~Page~~ State

SPA - Single Page Application

Зачем?

- Программирование лаба по сетям - бек упал, фронт не завис (браузер не показал 404)
- Передача данных между ~~страницами~~ представлениями
- Снизить нагрузку с сервера
- ‘Уменьшить’ нагрузку на сеть

Browser history API

Зачем?

Управлять историей браузера через JS

А что он умеет?

- history pushState() и replaceState()
- location forward() и back()
- location.href

Он Откуда?

BOM: window.location и window.history

https://habrahabr.ru/post/123106/

Web storage API

Зачем?

чтобы хранить чего-нибудь на клиенте и не пользоваться cookie

А что он умеет?

- две хеш-мапы: localStorage и sessionStorage
- localStorage - у каждого домена свой, пока js не почистим не удалится
- sessionStorage - выключил браузер, всё потерял

Он Откуда?

BOM: window.localStorage и window.sessionStorage

https://www.w3schools.com/html/html5_webstorage.asp

SPA

- Какой такой оффлайн режим?
- У меня долг по проге, я не делал эту лабу ещё
- Оффлайн? У нас же ве~~йп~~б, он же онлайн

SPA оффлайн режим

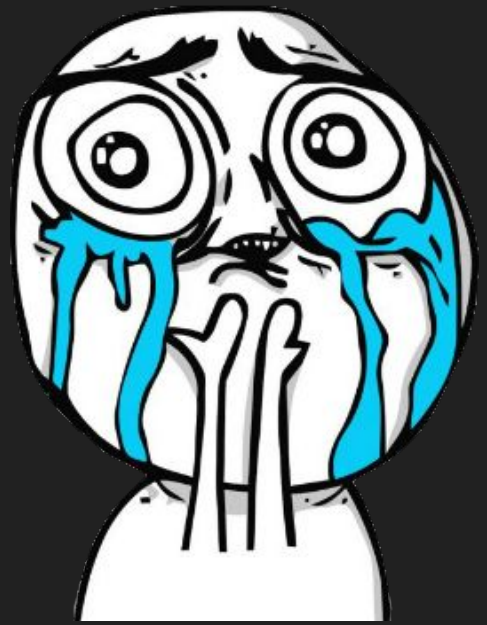
Зачем?

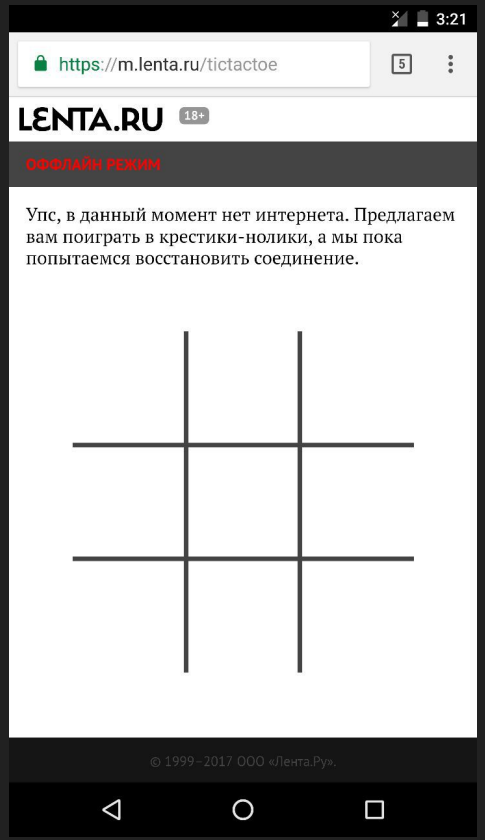
Дать пользователю не потерять данные, когда в метро пропал LTE и дать заняться чем-нибудь кроме игры про динозаврика

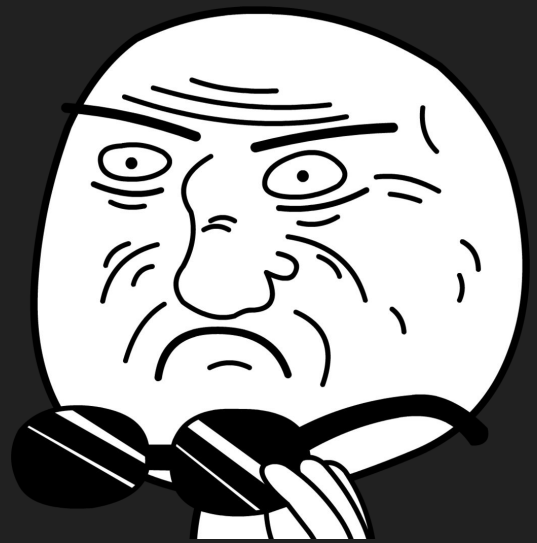
А что он умеет?

А что умеешь ты?

SPA оффлайн режим



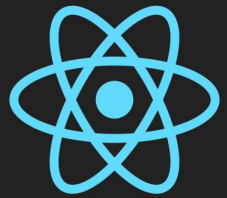




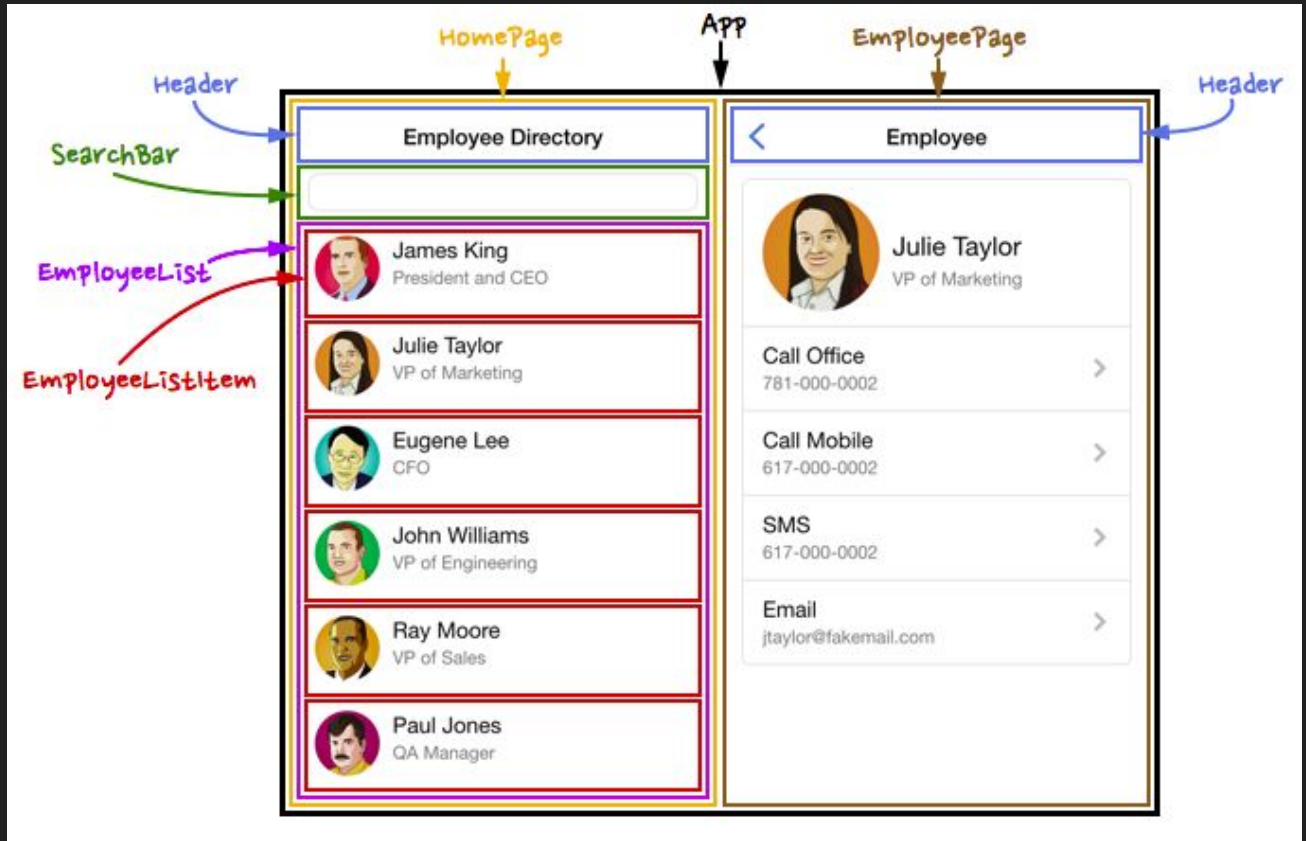
React - js библиотека для построения UI

Основные особенности:

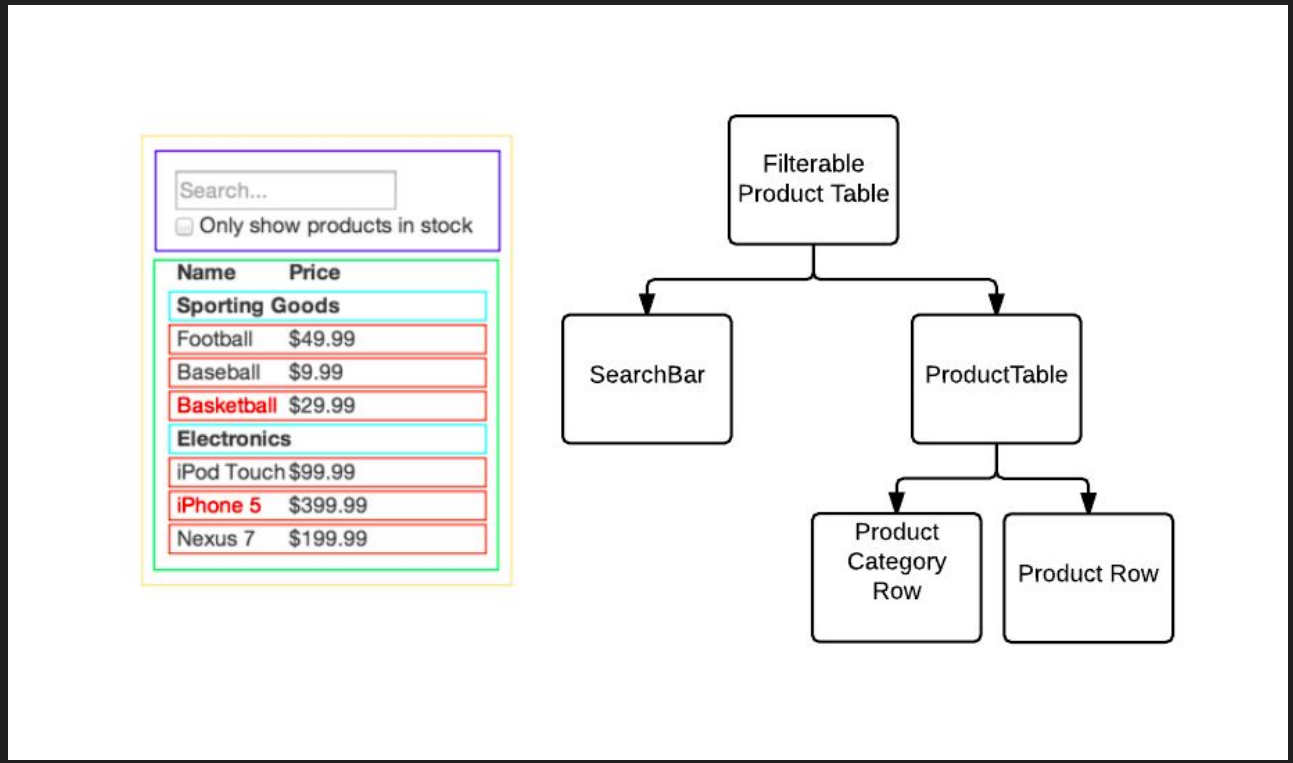
- основан на компонентах
- декларативный
- one-way dataflow
- virtual-dom



React (компоненты)



React (компоненты)



React (Hello world)

```
class Hello extends React.Component {
	render() {
		return React.createElement('div', null,
			`Hello ${this.props.name}`);
	}
}

ReactDOM.render(
	React.createElement(Hello, {name: 'Вася'}, null),
		document.getElementById('root')
);
```

JSX

```
const element = <h1>Hello, world!</h1>;
```

JSX:

- Расширение языка JavaScript
- Сахар для React.createElement(component, props, ...children)
- Компилируется Babel’ом в JS

React (Hello world) c JSX

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


ReactDOM.render(
  <Hello name=”Вася”/>,
  document.getElementById('root')
);
```

JSX

```
class NameForm extends React.Component {
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
          <label>
                  Name {this.name}:
                  <input type="text" value={this.state.value}
                          onChange={this.handleChange} />
          </label>
          <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

JSX

```
class ComponentThatUseAnotherYourComponent extends React.Component {
render() {
        return (
                <div>
                        We use component from previous slide
                        <NameForm/>
                </div>
        );
}
}
```

Компонент *NameForm* объявлен на прошлом слайде.
Здесь происходит его подключение.

JSX

```
render() {
    return (
        <div>
            <h3>TODO</h3>
            <TodoList items={this.state.items} />
            <form onSubmit={this.handleSubmit}>
                <input onChange={this.handleChange} value={this.state.text} />
                <button>{'Add #' + (this.state.items.length + 1)}</button>
            </form>
        </div>
    );
}
```

TodoList - пользовательский компонент

div, *h3*, *form*, *input*, *button* - встроенные компоненты

Важно: Пользовательские компоненты должны быть написаны с большой буквы

JSX

```
render() {
        let name = ‘Vasya’;
        return (
                <div>
                        <h3>Name {this.name}</h3>
                        <TodoList items={this.state.items} />
                </div>
        );
}
```

В фигурных скобках ‘{}’ JS выражения. Т.к. это выражения никаких if, for, объявлений и прочего.

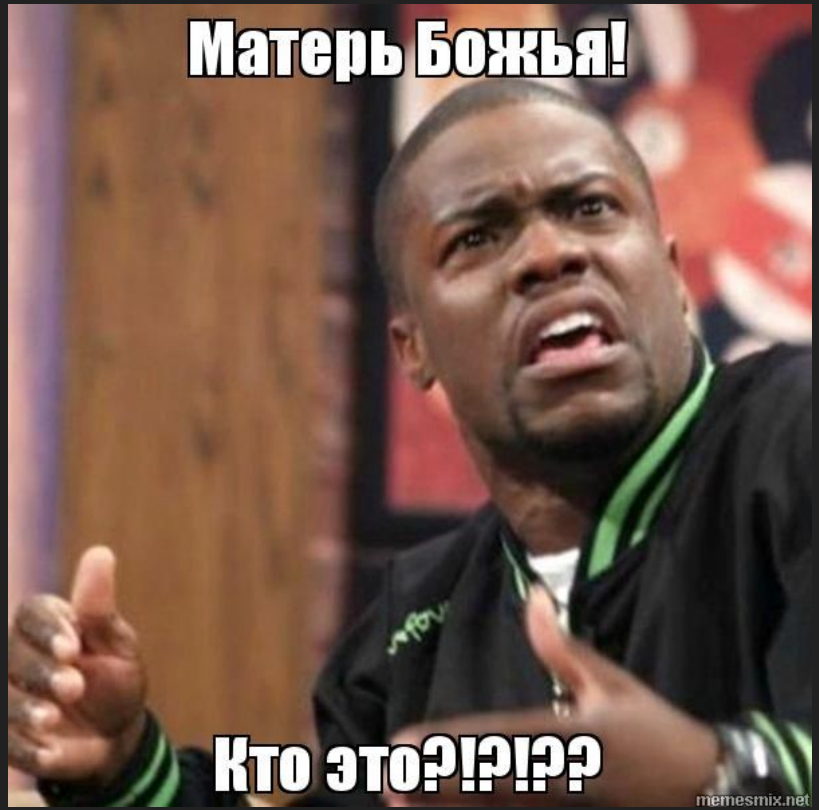
https://habrahabr.ru/post/319270/

JSX

```
import React from 'react';
import { PhotoStory, VideoStory } from './stories';

const components = {
  photo: PhotoStory,
  video: VideoStory
};

function Story(props) {
  // Wrong! JSX type can't be an expression.
  return <components[props.storyType]
story={props.story} />;
}
```

```
import React from 'react';
import { PhotoStory, VideoStory } from './stories';

const components = {
  photo: PhotoStory,
  video: VideoStory
};

function Story(props) {
  // Correct! JSX type can be a capitalized
variable.
  const SpecificStory =
components[props.storyType];
  return <SpecificStory story={props.story} />;
}
```

Babel? JSX? React?



Библиотеки и технологии

С React используют следующее:

- ES6
- npm
- babel
- webpack
- react-router
- redux

ES6

ECMAScript (ES) - язык программирования
ECMA-262 - спецификация языка ECMAScript

ES 2016 года, 7ое издание => ES2016 = ES7, а ES2015 = ES6

JavaScript - реализация ES

http://www.ecma-international.org/ecma-262/6.0/
https://github.com/tc39/ecma262#ecmascript

ES2015/ES6

- Классы
- Наследование через extends
- Модули
- Объявление переменных через let и const
- Деструктуризация
- Функции-стрелки
- Promise
- многое другое

let и const

Переменные let:

- Видны только после объявления и только в текущем блоке.
- Нельзя переобъявлять (в том же блоке).
- При объявлении переменной в цикле for(let …) – она видна только в этом цикле. Причём каждой итерации соответствует своя переменная let.

Переменная const – это константа, в остальном – как let.

https://learn.javascript.ru/let-const

Классы

ES5

```
function User(name) {
  this.name = name;
}

User.prototype.sayHi =
function() {
  alert(this.name);
};
```

=>

ES6/ES2015

```
class User {

  constructor(name) {
    this.name = name;
  }

  sayHi() {
    alert(this.name);
  }

}

let user = new User("Вася");
user.sayHi(); // Вася
```

Наследование

```
var animal = {
  eats: true
};
var rabbit = {
  jumps: true
};

rabbit.__proto__ = animal;
```

=>

```
class Animal {
  constructor(name) {
    this.name = name;
  }
  walk() {
    alert("I walk: " + this.name);
  }
}
class Rabbit extends Animal {
  walk() {
    super.walk();
    alert("...and jump!");
  }
}
new Rabbit("Вася").walk();
// I walk: Вася
// and jump!
```

Promise

ECMA-262
25. Control Abstraction Objects
    25.4. Promise Objects

Promise - это объект, цель которого облегчить написание отложенного/асинхронного кода.
У promise есть три состояния: “ожидание”, “успешно выполнено”, “выполнено с ошибкой”.

Подробнее тут:

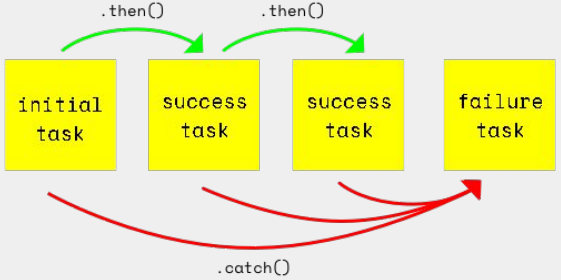
https://habrahabr.ru/post/242767/
https://learn.javascript.ru/promise
http://www.ecma-international.org/ecma-262/6.0/#sec-promise-objects

до Promise

```
callMe1(function(resp, error){
    if (!error && resp) {
        callMe2(function(resp, error){
            if (!error && resp) {
                callMe3(function(){
                    callMeFinal();
                });
            }
        });
    }
});
```

с Promise

```
httpGet('/article/promise/user.json')
  .then(JSON.parse)
  .then(user => httpGet(`https://api.github.com/users/${user.name}`))
.catch(error => {console.log(error); /}
```



Подробно и с картинками тут (на русском), либо гуглите Promise burger party

Модули

1. Новые способы ограничения области видимости
2. Избавляет от конфликта имен
3. Библиотека может вернуть в мир только API
4. До модулей пункт 3 был также осуществим, но выглядело мерзко

```
//------ lib.js ------
export const sqrt = Math.sqrt;
export function square(x) {
    return x * x;
}
export function diag(x, y) {
    return sqrt(square(x) + square(y));
}

//------ main.js ------
import { square, diag } from 'lib';
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```

NPM

NPM - пакетный менеджер для NodeJS

Раньше подключение JS библиотеки выглядело так:

1. Найти и скачать нужную библиотеку в интернете (jquery например)
2. Положить в проект
3. Подключить тегом script в html файле

Есть вариант с CDN, тогда скачивать ничего не нужно, но искать всё равно надо

NPM

- Центральный репозиторий https://www.npmjs.com/
- package.json для описания зависимостей проекта
- node_modules - директория, куда npm кладёт скачанные для проекта зависимости

Документация по ссылке (https://docs.npmjs.com/)

NPM

- npm install <packagename>
- npm i --save <pname> (автоматически добавит в package.json
- npm i -g <pname> (установить глобально)
- npm run start (запустить таргет start)
- ~/.npmrc или npm config - конфиг (например для proxy)

https://github.com/npm/npm

https://habrahabr.ru/post/133363/

https://habrahabr.ru/post/243335/

```
package.json
{
	"name": "my_package",
	"version": "1.0.0",
	"engine": {
		"node": ">=6",
		"npm": ">=3.10.1"
	},
	"dependencies": {
		"react": "^15.3.1",
		"react-router": "^2.8.1",
	},
	"devDependencies" : {
		babel-core": "^6.13.2",
		"webpack": "^1.13.1",
	}
	"scripts": {
		"start": "\"npm run build\" && \"npm run watch\"",
		"build": "webpack --config webpack.config.js",
		"watch": "webpack --watch",
	 }
}
```

Babel

Babel - транспайлер (компилятор), который преобразует один язык в другой.

Например, можно писать код на ES6 и преобразовывать его в ES5, чтобы оно работало в IE.

npm install --save-dev babel-loader babel-core

https://babeljs.io/
https://learn.javascript.ru/es-modern-usage#babel-js
https://habrahabr.ru/post/330018/

Babel

Зачем?

Браузеры внедряют новые языковые фичи медленно

А что он умеет?

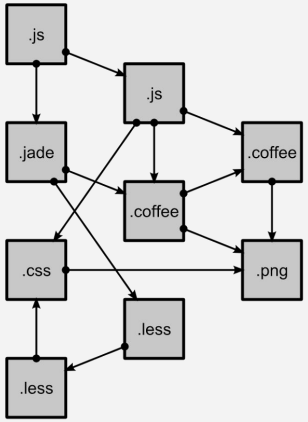
ES2015, JSX, Typescript, Coffeescript, Clojurescript и др.

Webpack

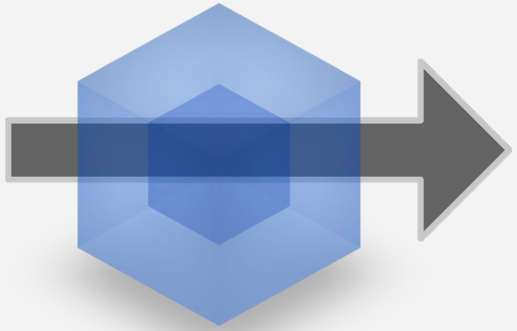
- Сборщик модулей
- Преобразует множество модулей(библиотек) в единый bundle
- Строит граф зависимостей
- npm i -g webpack
- Конфигурационный файл webpack.config.js
- Модули и плагины - точки расширения для управление сборкой и дополнительной обрабткой модулей (минимизация, css-препроцессоры, babel и прочее)
- webpack-dev-server
- hot-reload

https://webpack.js.org/concepts/
https://habrahabr.ru/post/245991/

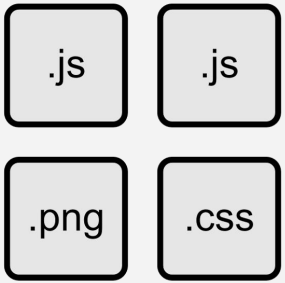
Webpack



modules
with dependencies



webpack
MODULE BUNDLER



static
assets

Webpack

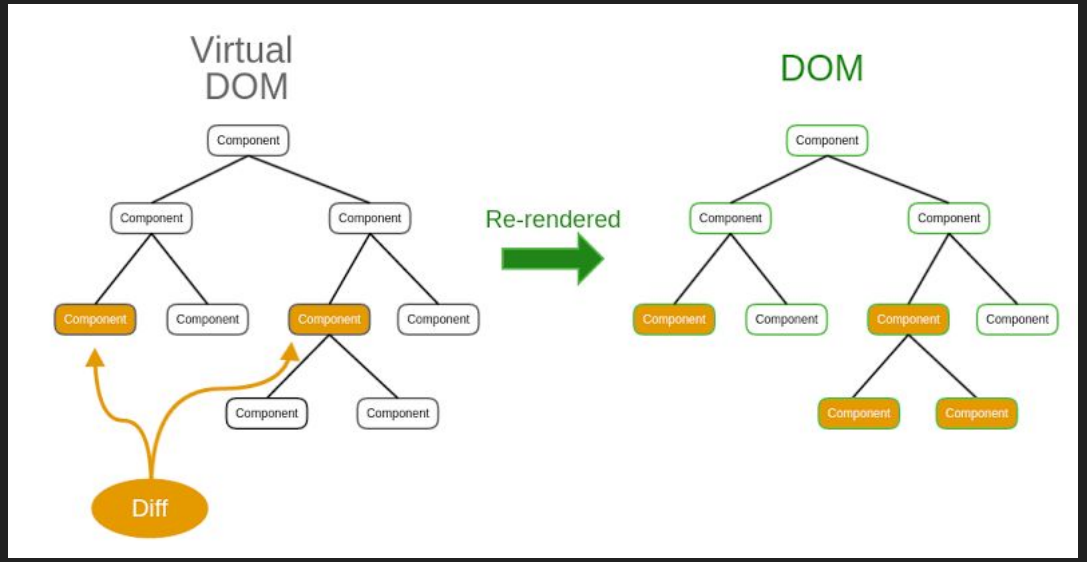
```
const HtmlWebpackPlugin = require('html-webpack-plugin'); //installed via npm
const webpack = require('webpack'); //to access built-in plugins
const path = require('path');
const config = {
  entry: './path/to/my/entry/file.js',
  output: {
    path: path.resolve(__dirname, 'build'),
    filename:. 'bundle.js'
  },
  module: {
    loaders:
          [ /* Следующий слайд */],

  },
  plugins: [
    new webpack.optimize.UglifyJsPlugin(),
    new HtmlWebpackPlugin({template: './src/index.html'})
  ]
};
module.exports = config;
```

Webpack

```
module: {
    loaders:
            [{
               test: /\.(jsx|js)?$/,
                exclude: /node_modules/,
                loader: 'babel-loader',
                query: {
                    presets: ['react', 'es2015', 'stage-0'],
                },
            },
            {
                test: /\.(less|css)$/,
                loader: ExtractTextPlugin .extract('style-loader', 'css-loader!less-loader'),
            }],

}
```

VirtualDOM

Virtual DOM — это техника и набор библиотек / алгоритмов, которые позволяют нам улучшить производительность на клиентской стороне, избегая прямой работы с DOM путем работы с легким JavaScript-объектом, имитирующем DOM-дерево.



https://habrahabr.ru/post/256965/

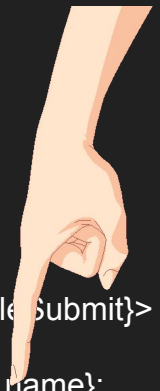
state & props

Каждый компонент обладает состоянием ‘state’ и свойствами ‘props’.

Свойства служат для передачи данных между родительским компонентом и дочерним.

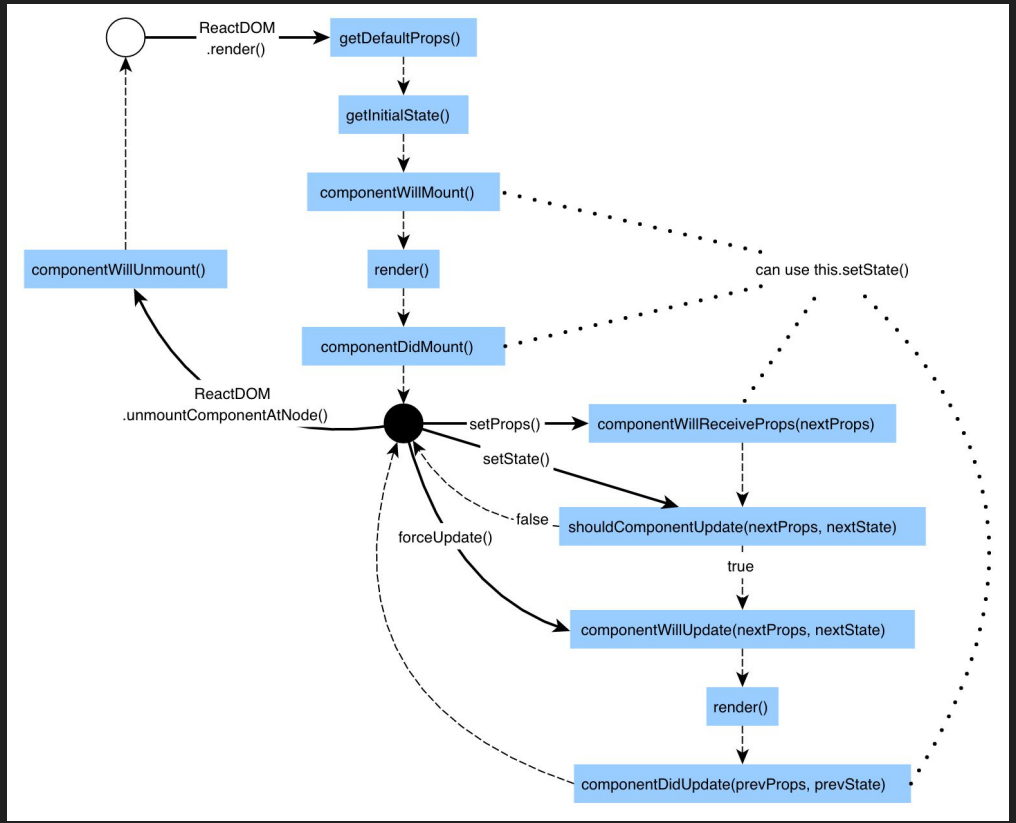
state & props

```
class MyComp extends React.Component {
constructor(props) {
    super(props);
    this.state = {myName: 'Vasya'"};

render() {
        return (
                <div>
                        We use component from previous slide
                        <NameForm name={this.state.myName}/>
                </div>
}
}
```



```
class NameForm extends
React.Component {
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
          <label>
                  Name {this.props.name}:
                  <input type="text"
value={this.state.value}
                          onChange={this.han
dleChange} />
          </label>
          <input type="submit"
value="Submit" />
      </form>
    );
  }
}
```

Component lifecycle



Контролируемые компоненты

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};

    this.handleChange =
this.handleChange.bind(this);
    this.handleSubmit =
this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" value={this.state.value}
onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Простейшая форма

```
class Input extends Component {
 constructor(props) {...}
 inputChange(event) {
   event.preventDefault();
   this.setState({value: event.target.value});
 }
 render() {
   return (
     <div className='input-row'>
       <div className='input-label-wrapper'>
         <label htmlFor={this.props.id} className='label'>{this.props.label}</label>
       </div>
       <div className='input-wrapper'>
         <input className='input' name={this.props.id} id={this.props.id}
type={this.props.inputType}
             value={this.state.value} placeholder={this.props.placeholder}
onChange={this.inputChange}/>
       </div>
     </div>
   );
 }
}
```

```
class Form extends Component {
 constructor(props) {...}
 submit(submitEvent) {
   submitEvent.preventDefault();
   //put your submit here
 }
 render() {
   return (
     <div style={this.props.style}>
       <form onSubmit={this.submit}>
         <Input id='sample-1' label='Sample input' inputType='text'
placeholder='Type your text here' />
         <Input id='sample-2' label='Yet another input' inputType='number'
placeholder='13' />
         <button type='submit' onClick={this.submit}>Click me!</button>
       </form>
     </div >
     );
 }
}
```

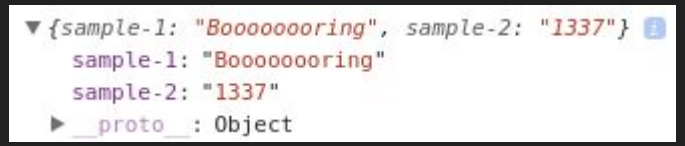
Эй, но она же ничего не делает!

Вверх!

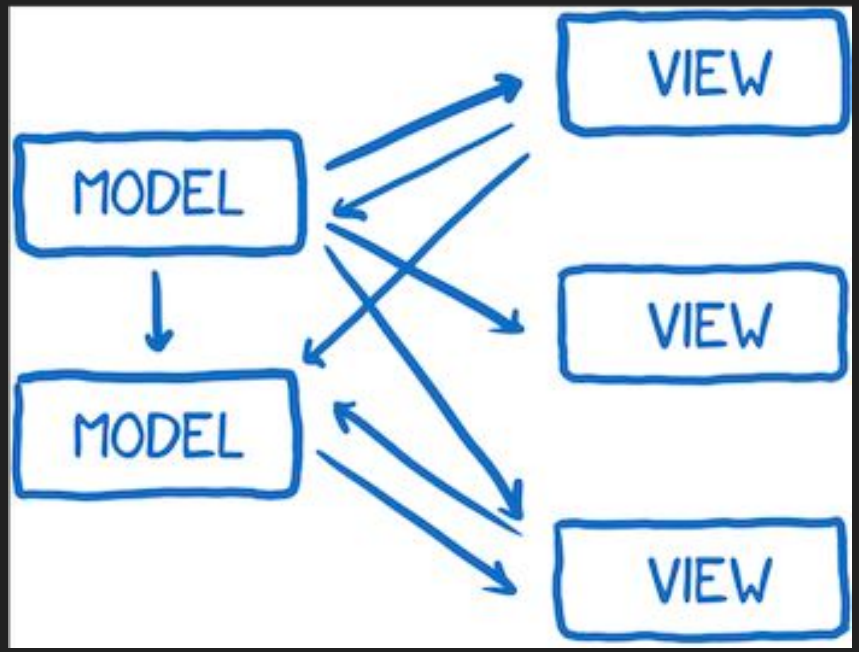
```
const Input = (props) => (
 <div className='input-row'>
   <div className='input-label-wrapper'>
     <label htmlFor={props.id}
className='label'>{props.label}</label>
   </div>
   <div className='input-wrapper'>
     <input className='input' name={props.id} id={props.id}
type={props.inputType}
       value={props.value} placeholder={props.placeholder}
onChange={props.handleChange} />
   </div>
 </div>
);
```

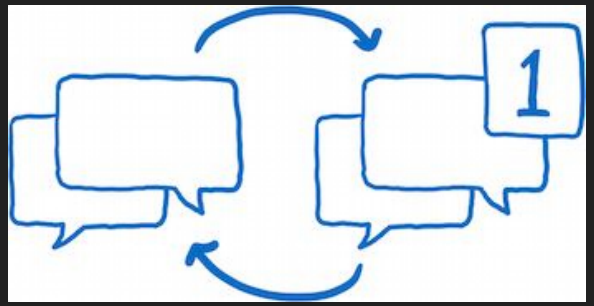
```
class Form extends Component {
 submit(submitEvent) {
   submitEvent.preventDefault();
   console.log(this.state);
   //put your submit here
 }
 handleChange(event) {
   this.setState({
     [event.target.name] : event.target.value
   })
 }
 render() {
   return (
     <div style={this.props.style}>
       <form onSubmit={this.submit}>
         <Input id='sample-1' label='Sample input' inputType='text' placeholder='Type
your text here'
         handleChange={this.handleChange} />
         <Input id='sample-2' label='Yet another input' inputType='number'
placeholder='13' handleChange={this.handleChange} />
         <button type='submit' onClick={this.submit}>Click me!</button>
       </form>
     </div >
     );
 }
}
```

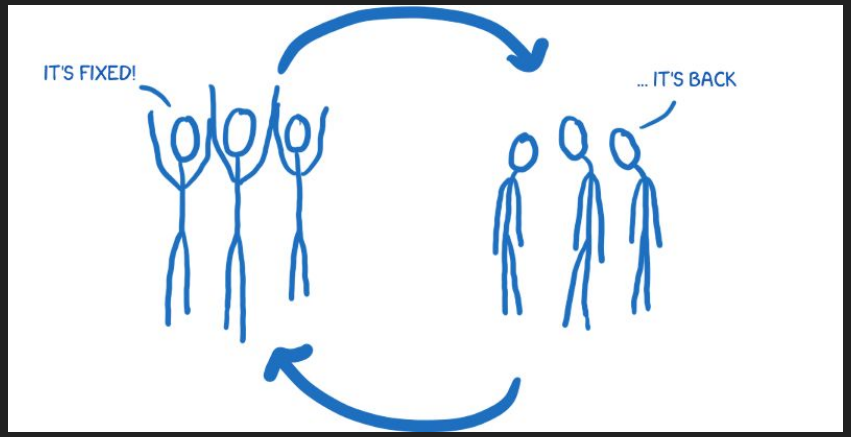

Sample input
Booooooooring
Yet another input
1337
Click me!


{sample-1: "Booooooooring", sample-2: "1337"}
sample-1: "Booooooooring"
sample-2: "1337"
__proto__: Object

Flux/Redux — введение

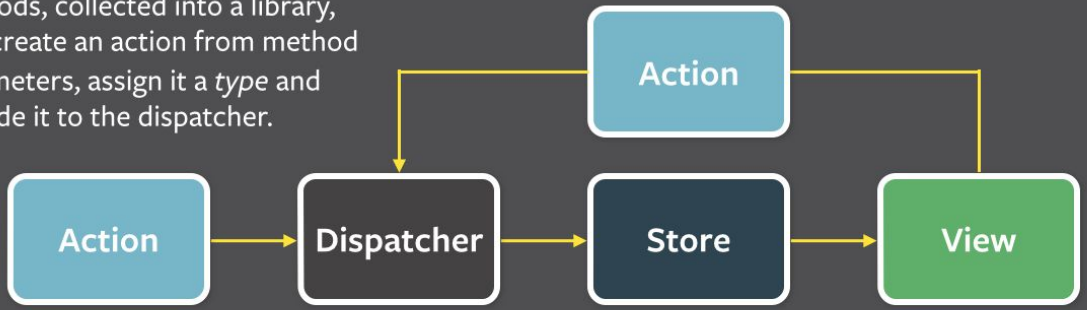






flux

Action creators are helper methods, collected into a library, that create an action from method parameters, assign it a *type* and provide it to the dispatcher.



Every action is sent to all stores via the *callbacks* the stores register with the dispatcher.

After stores update themselves in response to an action, they emit a *change* event.

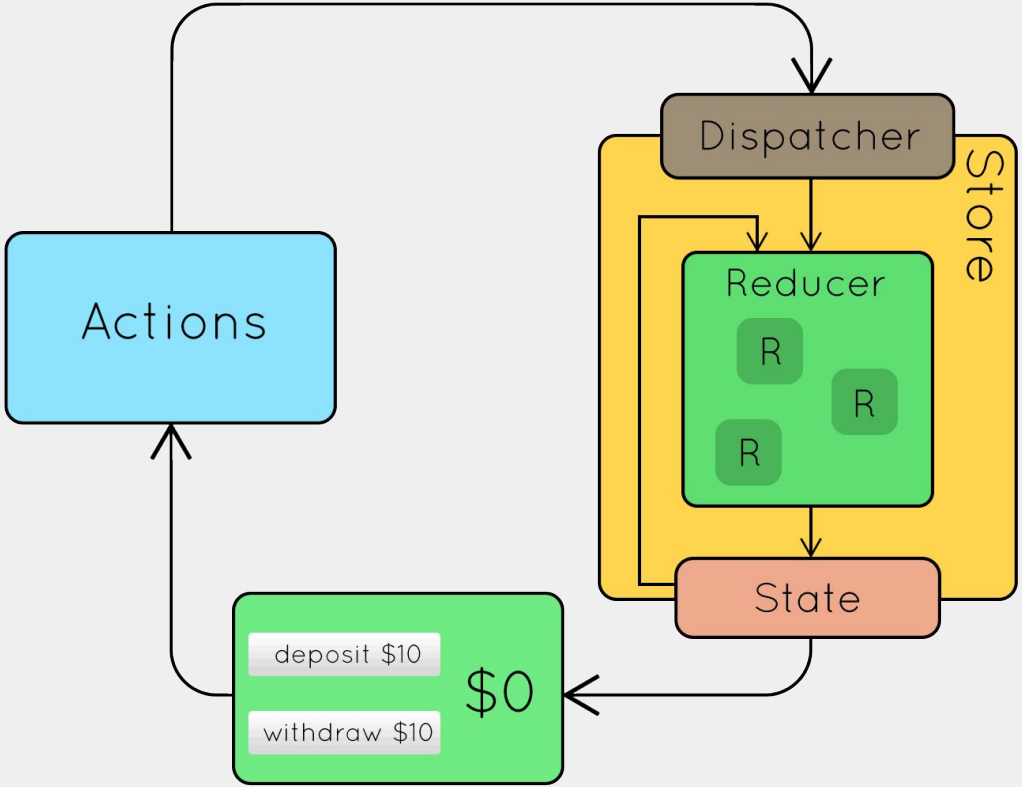
Special views called *controller-views*, listen for *change* events, retrieve the new data from the stores and provide the new data to the entire tree of their child views.

Redux

API из 5 функций:

- createStore
- combineReducers
- bindActionCreators
- applyMiddleware
- compose

Redux


Dispatcher
Store
Reducer
R
R
R
Actions
State
deposit $10
withdraw $10
$0

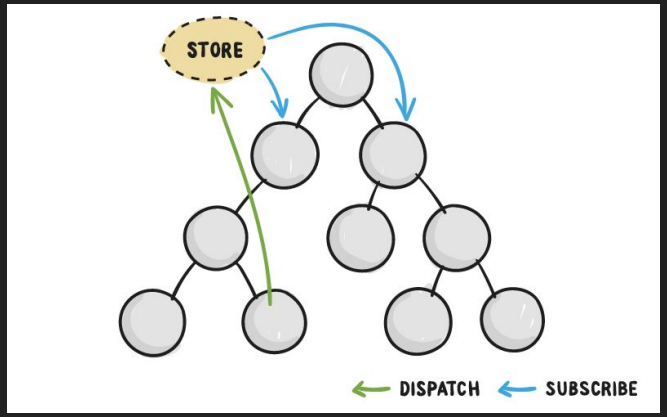
createStore()

function createStore(reducer, preloadedState, enhancer)

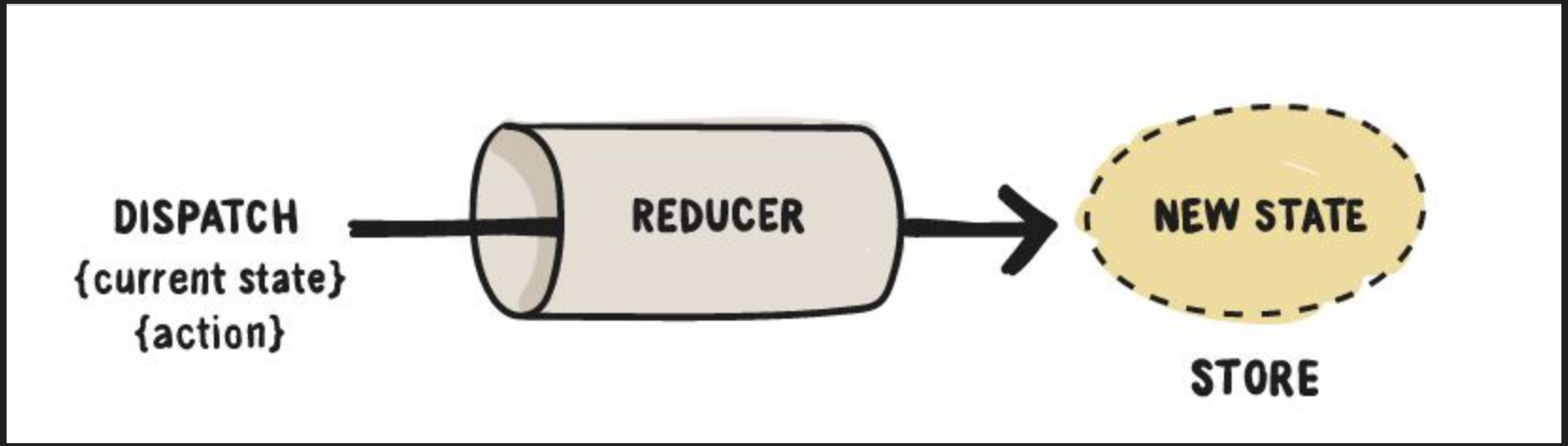
- reducer - на основе action и state генерирует новый state
- preloadedState - начальное состояние приложения
- enhancer - цепочка middleware

Store

Redux предоставляет единое хранилище состояния. Ключевой его особенностью является неизменяемость (привет, функциональщина!) — все изменения в него вносятся посредством создания нового объекта состояния.



Reducer



- Чистая функция, её выходное значение зависит только от входных
- Необходимо возвращать новое состояние, состояние должно быть неизменяемо

Reducer

```
export default function user(state = initialState, action) {
    switch (action.type) {
       case USER_INFO_REQUEST: {
           return Object.assign({}, state, {fetchingInfo: true});
        }
        case USER_INFO_COMPLETE: {
          return fetchingUserInfoComplete(state, action);
        }
        case USER_INFO_FAIL: {
            return Object.assign({}, state, {
                fetchingInfo: false,
                fetchInfoError: action.error,
            });
        }
}
```

Reducer

```
function fetchingUserInfoComplete(state, action) {
    const {
        firstname: firstName,
        lastname: lastName,
        authorities,
    } = action.payload.body;
    return Object.assign({}, state, {
        fetchingInfo: false,
        authorities,
        displayName: `${firstName} ${lastName}`,
        role: mapAuthoritiesToRole(authorities),
    });
}
```

combineReducers()

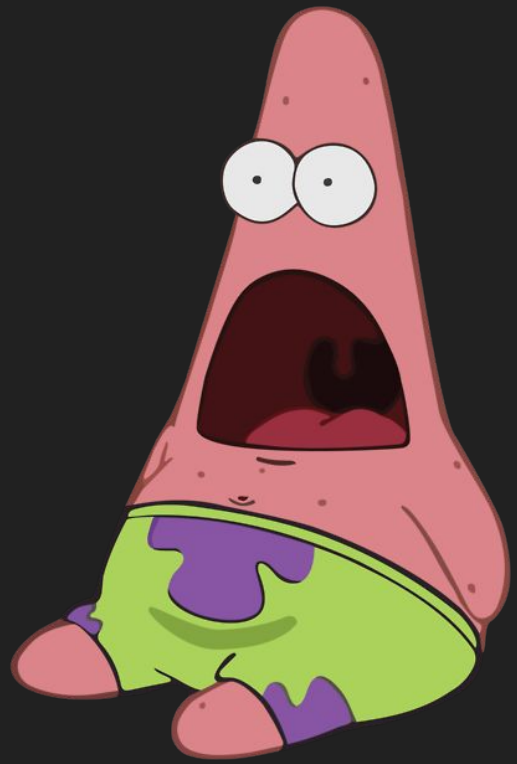
export default function combineReducers(reducers);

Пример комбинирования Reducer’а

```
import {combineReducers} from 'redux';
import page from './objects';
import {routerReducer} from 'react-router-redux';
import user from './user';
const rootReducer = combineReducers({
        page,
        user,
        routing: routerReducer,
});

export default rootReducer;
```

Middleware

```
function applyMiddleware(...middlewares)


middleware = (store) => (next) => (action) => {

}
```



Middleware

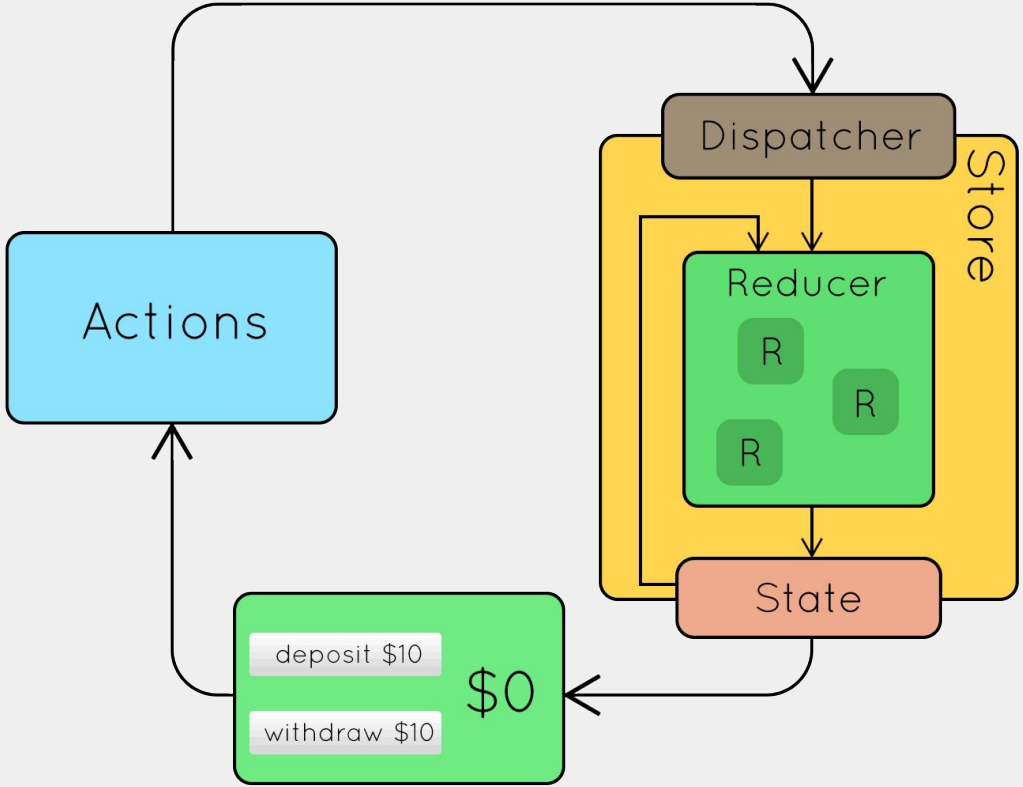
Каррирование или **карринг** (англ. *currying*) в информатике — преобразование функции от многих аргументов в набор функций, каждая из которых с одним аргументом. Возможность такого преобразования впервые отмечена в трудах Готтлоба Фреге, систематически изучена Моисеем Шейнфинкелем в 1920-е годы, а наименование получило по имени Хаскелла Карри — разработчика комбинаторной логики, в которой сведение к функциям одного аргумента носит основополагающий характер.

Middleware

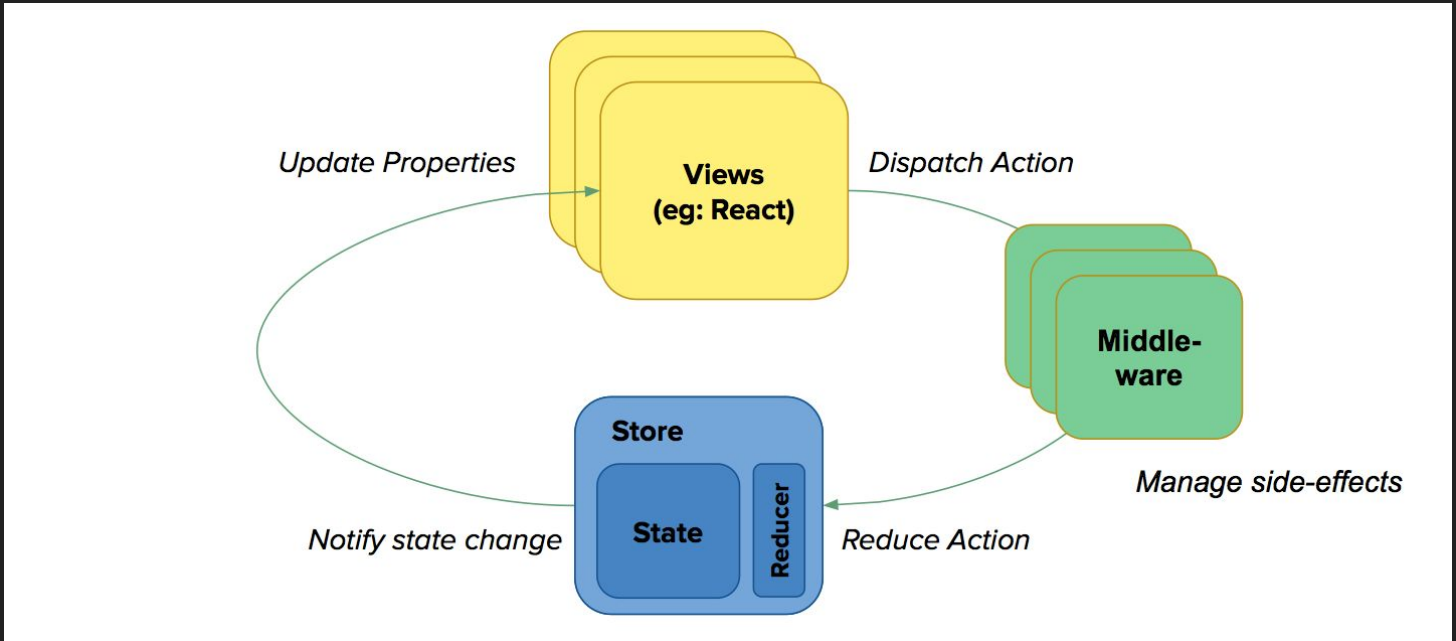
А зачем?

reducer - чистая функция
action - объект

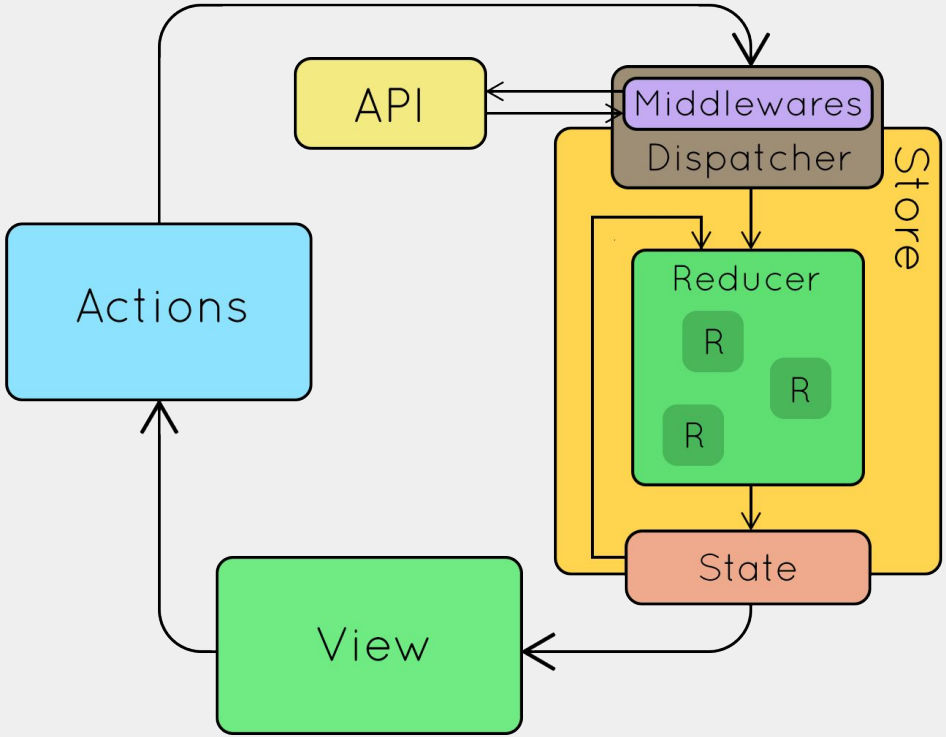
Redux


Dispatcher
Store
Reducer
R
R
R
Actions
State
deposit $10
withdraw $10
$0

middleware


Update Properties
Views
(eg: React)
Dispatch Action
Middle-
ware
Store
State
Reducer
Manage side-effects
Notify state change
Reduce Action

Redux


API
Middlewares
Dispatcher
Store
Actions
Reducer
R
R
R
State
View

Action

```
export const addTodo = (name, date, comment) => ({
        type: types.ADD_TODO,
        payload: {
                name,
                date,
                comment,
        },

});
```

dispatch

```
import {addTodo} from ‘../actions/todo’;
import dispatch from ‘redux’;

func() {

    dispatch(addTodo);

}
```