АРХИТЕКТУРА СОВРЕМЕННЫХ JS-ПРИЛОЖЕНИЙ: ТРИ ФРЕЙМВОРКА, ТРИ ПОДХОДА

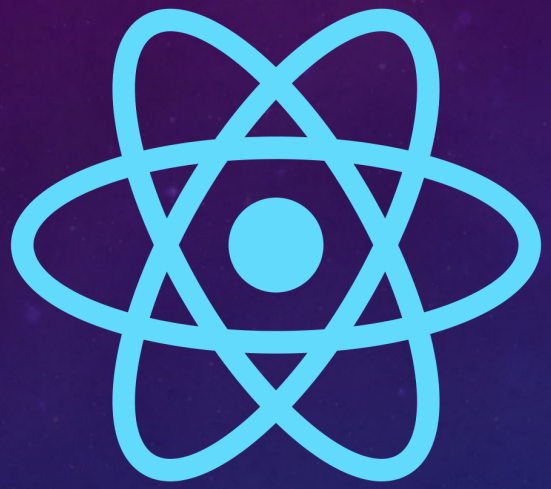
Никита Исаев, HolyJS Санкт-Петербург, 2017

nikis05@mail.ru

О ЧЕМ БУДЕМ ГОВОРИТЬ

- Что общего у трех топовых современных фреймворков и почему?
- В чем подходы различаются?
- В чем преимущества и недостатки того или иного подхода?
- Какой из подходов следует использовать в определенном случае?
- Что можно получить от доклада: повод попробовать что-то новое + системный взгляд

ФРЕЙМВОРКИ



АРХИТЕКТУРА СОВРЕМЕННЫХ JS-ПРИЛОЖЕНИЙ

Приложение = компоненты + State Management

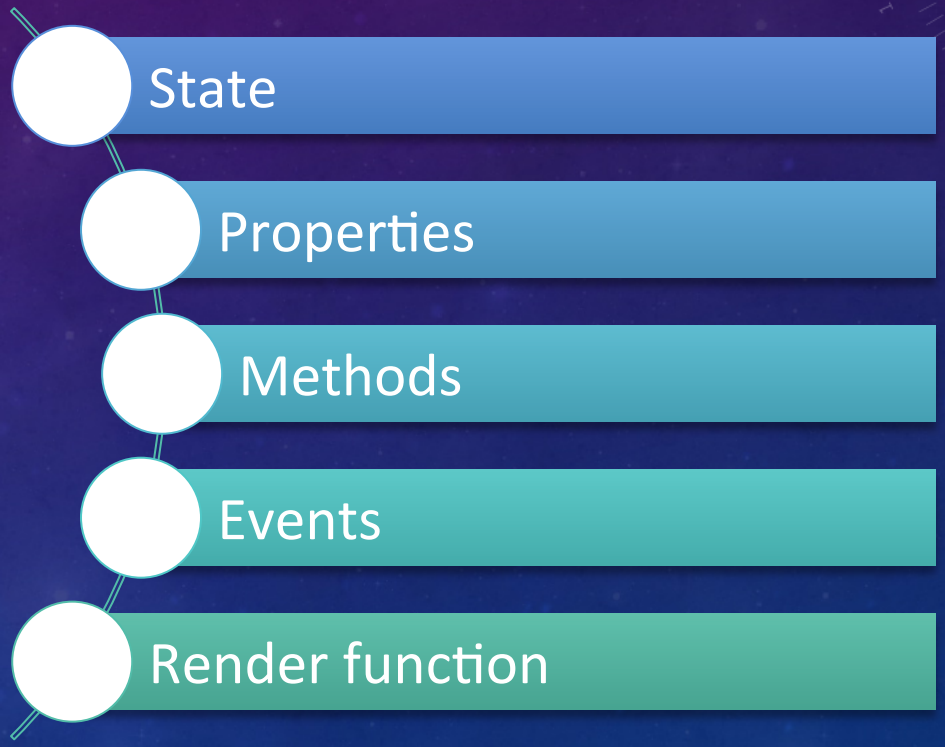
АРХИТЕКТУРА СОВРЕМЕННЫХ JS-ПРИЛОЖЕНИЙ

Приложение = компоненты + State Management

АНАТОМИЯ КОМПОНЕНТА

```
MyFramework.Component('current-count', {
  data: () => {
    return {
      count: 0
    };
  },
  props: ['increaseBy'],
  methods: {
    increase() {
      this.count += this.props.increaseBy;
      this.$emit('increased');
    }
  },
  render: () => (
    <span>
      Current count: <b>{{this.count}}</b><br />
      <button @click="this.increase">
        Click to increase by {{this.props.increaseBy}}!
      </button>
    </span>
  )
});
```

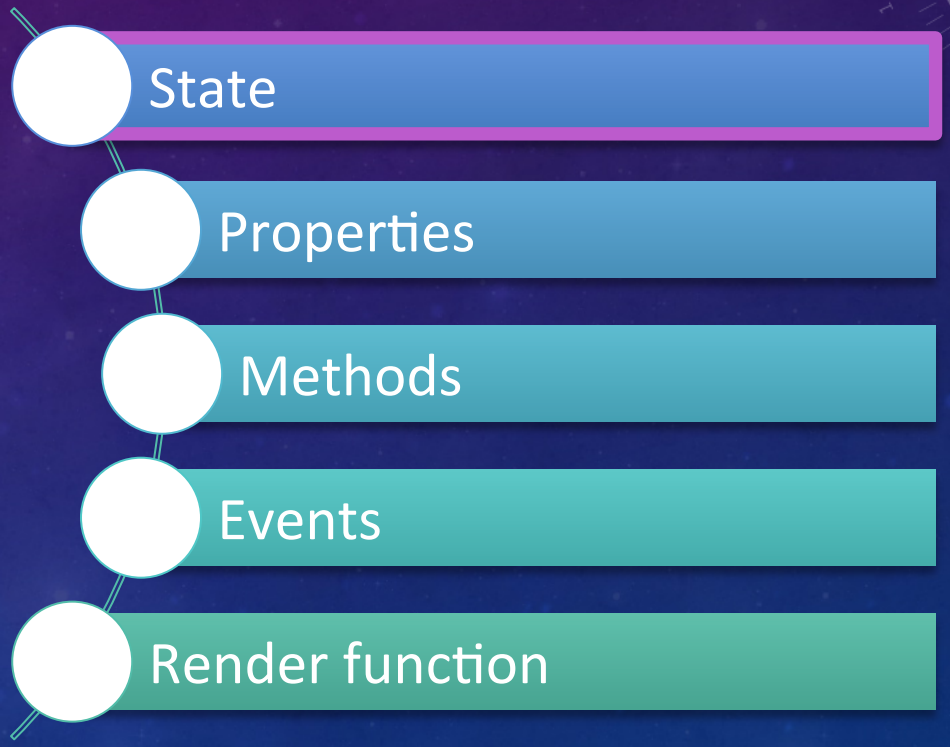
```
<current-count :increase-by="1" />
```



АНАТОМИЯ КОМПОНЕНТА

```
MyFramework.Component('current-count', {
  data: () => {
    return {
      count: 0
    };
  },
  props: ['increaseBy'],
  methods: {
    increase() {
      this.count += this.props.increaseBy;
      this.$emit('increased');
    }
  },
  render: () => (
    <span>
      Current count: <b>{{this.count}}</b><br />
      <button @click="this.increase">
        Click to increase by {{this.props.increaseBy}}!
      </button>
    </span>
  )
});
```

```
<current-count :increase-by="1" />
```



СОСТОЯНИЕ: IMMUTABLE VS OBSERVABLE

- Более явное изменение (+)

```
1  // Mutable
2
3  this.a = 1;
4
5  // Immutable
6
7  this.setState({a: 1});
```

СОСТОЯНИЕ: IMMUTABLE VS OBSERVABLE

- Более явное изменение (+)
- Больше писать (-)

```
1  // Mutable
2
3  this.a = 1;
4
5  // Immutable
6
7  this.setState({a: 1});
```

СОСТОЯНИЕ: IMMUTABLE VS OBSERVABLE

- Более явное изменение (+)
- Больше писать (-)
- Нет четкой схемы (-)

```
Vue.component('my-component', {
  data: () => {
    return {
      username: ''
    }
  },
  methods: {
    setUsername(username) {
      this.username = username;
    }
  }
});
```

СОСТОЯНИЕ: IMMUTABLE VS OBSERVABLE

- Более явное изменение (+)
- Больше писать (-)
- Нет четкой схемы (-)
- Нет ошибок, связанных с Observable (+)

```
Vue.component('my-component', {
  data: () => {
    return {
      users: ['nikis05', 'holyjs']
    }
  },
  methods: {
    addThirdUser() {
      this.users[3] = 'newuser';
    }
  }
});
```

СОСТОЯНИЕ: IMMUTABLE VS OBSERVABLE

- Более явное изменение (+)
- Больше писать (-)
- Нет четкой схемы (-)
- Нет ошибок, связанных с Observable (+)
- setState асинхронен (-)

```
class Counter extends React.Component {

  state = {
    count: 0
  }

  increase() {
    this.setState({
      count: this.state.count + 1
    });
    console.log(this.state.count); // 0
  }

  // ...
}
```

СОСТОЯНИЕ: IMMUTABLE VS OBSERVABLE

- Более явное изменение (+)
- Больше писать (-)
- Нет четкой схемы (-)
- Нет ошибок, связанных с Observable (+)
- setState асинхронен (-)
- setState может вызывать лишние рендеры (-)

СОСТОЯНИЕ: IMMUTABLE VS OBSERVABLE

- Более явное изменение (+)
- Больше писать (-)
- Нет четкой схемы (-)
- Нет ошибок, связанных с Observable (+)
- setState асинхронен (-)
- setState может вызывать лишние рендеры (-)
- Нет Computed Properties (-)

СОСТОЯНИЕ: COMPUTED PROPERTIES

```
<div id="example">
  {{ message.split('').reverse().join('') }}
</div>
```

СОСТОЯНИЕ: COMPUTED PROPERTIES

```
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>
```

```
var vm = new Vue({
  el: '#example',
  data: {
    message: 'Hello'
  },
  computed: {
    // a computed getter
    reversedMessage: function () {
      // `this` points to the vm instance
      return this.message.split('').reverse().join('')
    }
  }
})
```

СОСТОЯНИЕ: COMPUTED PROPERTIES

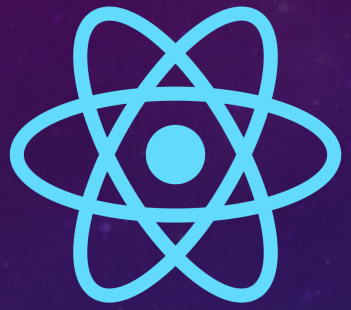
```
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>
```

```
class Example extends SomeFramework.Component {

  state = {
    message: 'Hello!',
    reversedMessage: '!olleH'
  }

  get reversedMessage() {
    this.state.message
      .split('').reverse().join('')
  }

  detectStateChangeSomehow() {
    this.setState({
      reversedMessage: this.reversedMessage
    });
  }

})
```

СОСТОЯНИЕ: OBSERVABLE В REACT С MOBX

```
@observer

class MyComponent extends React.Component {

  // observable properties
  @observable firstCounter = 0;
  @observable secondCounter = 0;

  // computed property
  // reevaluates automatically when relevant observables mutate

  @computed get totalCount() {
    return this.firstCounter + this.secondCounter;
  }

}
```

СОСТОЯНИЕ



- Иммутабельное и чистое
- setState имеет ряд тонкостей
- Нет Computed Properties
- Можно сделать observable, но нужен MobX

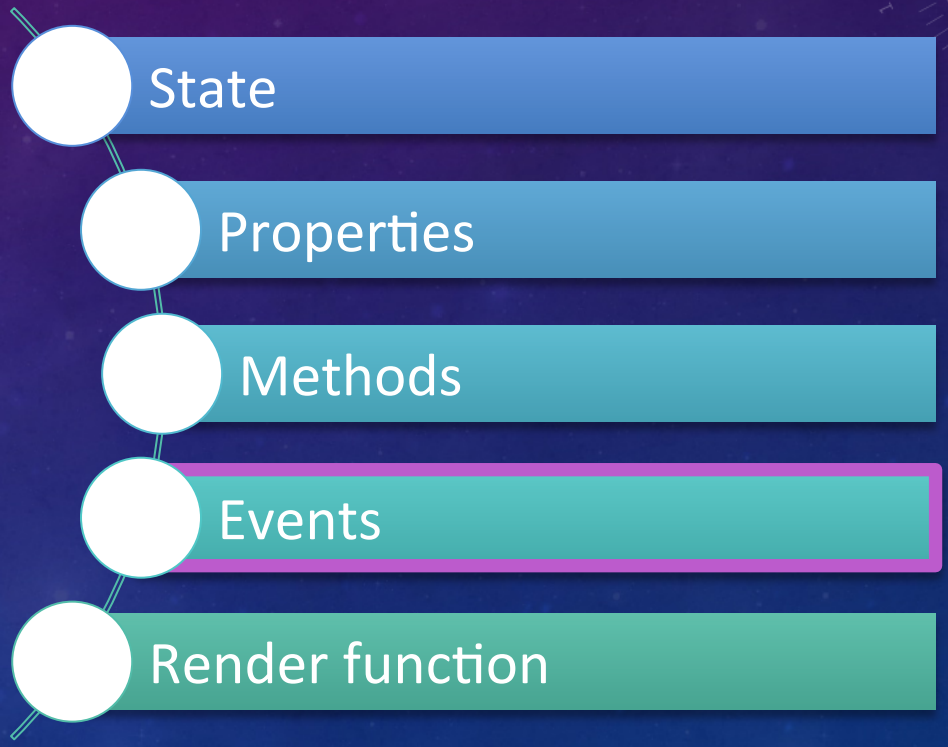
- Observable
- Нет Computed Properties

- Observable
- Нативные Computed Properties

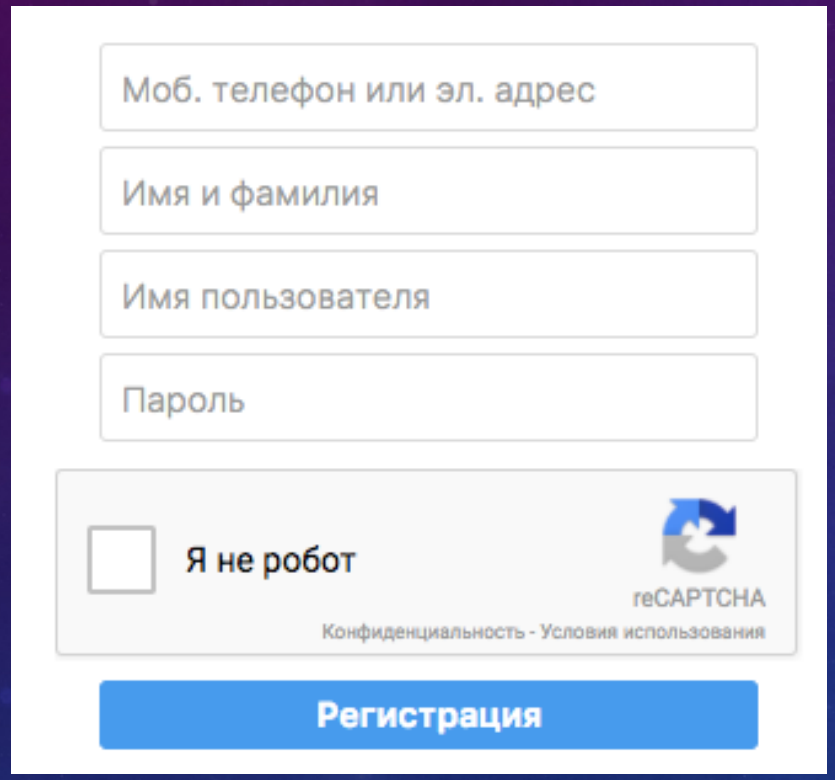
АНАТОМИЯ КОМПОНЕНТА

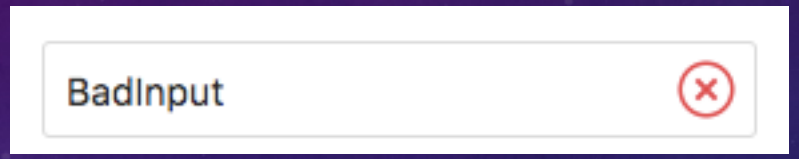
```
MyFramework.Component('current-count', {
  data: () => {
    return {
      count: 0
    };
  },
  props: ['increaseBy'],
  methods: {
    increase() {
      this.count += this.props.increaseBy;
      this.$emit('increased');
    }
  },
  render: () => (
    <span>
      Current count: <b>{{this.count}}</b><br />
      <button @click="this.increase">
        Click to increase by {{this.props.increaseBy}}!
      </button>
    </span>
  )
});
```

```
<current-count :increase-by="1" />
```



ЭВЕНТЫ: CONTROLLED COMPONENTS





ЭВЕНТЫ: CONTROLLED COMPONENTS

```
class MyComponent extends React.Component {

  state = {
    message: ''
  };

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

  render() {
    return (
      <input
        type="text"
        value={message}
        onChange={
          (evt) => this.handleChange(evt)
        }
      />
    );
  }
}
```

```
let vm = new Vue ({
  data: {
    message: ''
  },
  template: `
    <input type="text" v-model="message">
  `
});
```

ЭВЕНТЫ: CONTROLLED COMPONENTS

```
var createReactClass = require('create-react-class');

var NoLink = createReactClass({
  getInitialState: function() {
    return {message: 'Hello!'};
  },
  handleChange: function(event) {
    this.setState({message: event.target.value});
  },
  render: function() {
    var message = this.state.message;
    return <input type="text" value={message} onChange={this.handleChange} />;
  }
});
```

ЭВЕНТЫ: CONTROLLED COMPONENTS

```
var createReactClass = require('create-react-class');

var WithLink = createReactClass({
  mixins: [LinkedStateMixin],
  getInitialState: function() {
    return {message: 'Hello!'};
  },
  render: function() {
    return <input type="text" valueLink={this.linkState('message')} />;
  }
});
```

ЭВЕНТЫ: CONTROLLED COMPONENTS

```
var createReactClass = require('create-react-class');

var WithLink = createReactClass({
  mixins: [LinkedStateMixin],
  getInitialState: function() {
    return {message: 'Hello!'};
  },
  render: function() {
    return <input type="text" valueLink={this.linkState('message')} />;
  }
});
```

Note: `LinkedStateMixin` **is deprecated as of React v15. The recommendation is to explicitly set the value and change handler, instead of using** `LinkedStateMixin`**.**

ЭВЕНТЫ VS КОЛЛБЭКИ

- Эвенты более явные (+)
- Более четкое разделение ответственности (+)
- Не потеряется контекст (+)

Эвенты

```
<my-component @customEvent="doSomething()">
```

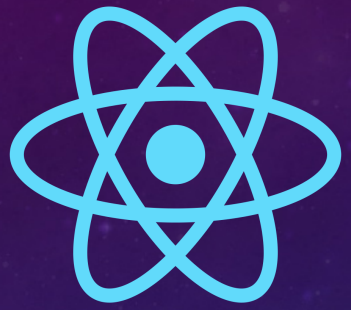
```
this.$emit('customEvent');
```

Коллбеки

```
<my-component
    onCustomEvent={() => this.doSomething()}
>
```

```
this.props.onCustomEvent();
```

ЭВЕНТЫ



- Можно потерять контекст
- Много бойлерплейта при биндинге
- Коллбэки вместо кастомных эвентов

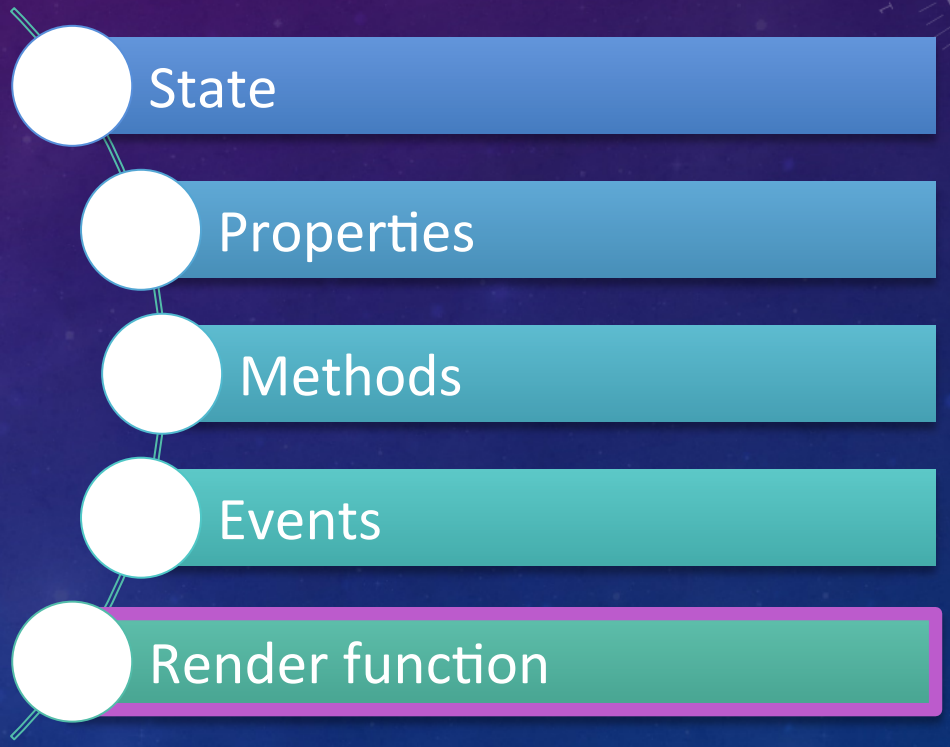
- Простой биндинг с помощью ngModel
- Можно создать кастомный элемент ввода с ControlValueAccessor

- Простой биндинг с помощью v-model
- Можно создать кастомный элемент ввода, добавив компоненту 1 prop и 1 метод

АНАТОМИЯ КОМПОНЕНТА

```
MyFramework.Component('current-count', {
  data: () => {
    return {
      count: 0
    };
  },
  props: ['increaseBy'],
  methods: {
    increase() {
      this.count += this.props.increaseBy;
      this.$emit('increased');
    }
  },
  render: () => (
    <span>
      Current count: <b>{{this.count}}</b><br />
      <button @click="this.increase">
        Click to increase by {{this.props.increaseBy}}!
      </button>
    </span>
  )
});
```

```
<current-count :increase-by="1" />
```



JSX VS TEMPLATES

- JSX – почти настоящий Javascript

JSX VS TEMPLATES

- JSX – почти настоящий Javascript
- Мы не ограничены набором директив

```
<div v-if="true"></div>

<ul>
  <li v-for="item in items">
    {{item}}
  </li>
</ul>

<div v-show="true"></div>

<div :class="{error: isError, important:
isImportant}">
```

JSX VS TEMPLATES

- JSX – почти настоящий Javascript
- Мы не ограничены набором директив
- Директив хватает для большинства вещей, и можно писать свои

```
<div v-if="true"></div>

<ul>
  <li v-for="item in items">
    {{item}}
  </li>
</ul>

<div v-show="true"></div>

<div :class="{error: isError, important:
isImportant}">
```

JSX VS TEMPLATES

- JSX – почти настоящий Javascript
- Мы не ограничены набором директив
- Директив хватает для большинства вещей, и можно писать свои
- Директивы иногда сильно упрощают работу

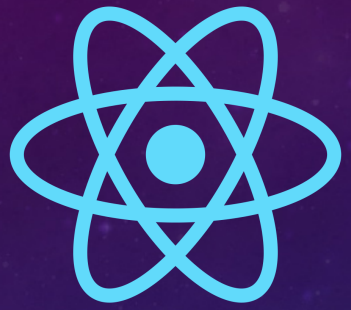
```
Vue.component('my-textform', {
  data: () => {
    return {
      value: ''
    };
  },
  template: `
    <input v-model.trim="value" type="text">
  `
});
```

JSX VS TEMPLATES

- JSX – почти настоящий Javascript
- Мы не ограничены набором директив
- Директив хватает для большинства вещей, и можно писать свои
- Директивы иногда сильно упрощают работу
- JSX нарушает separation of concerns

```
Vue.component('my-textform', {
  data: () => {
    return {
      value: ''
    };
  },
  template: `
    <input v-model.trim="value" type="text">
  `
});
```

РЕНДЕРИНГ



- Только JSX

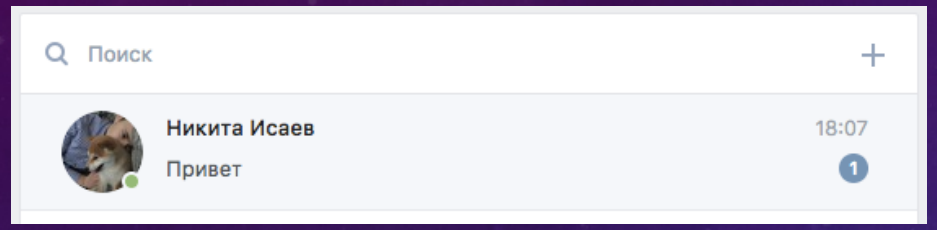
- Только шаблоны

- Шаблоны или JSX

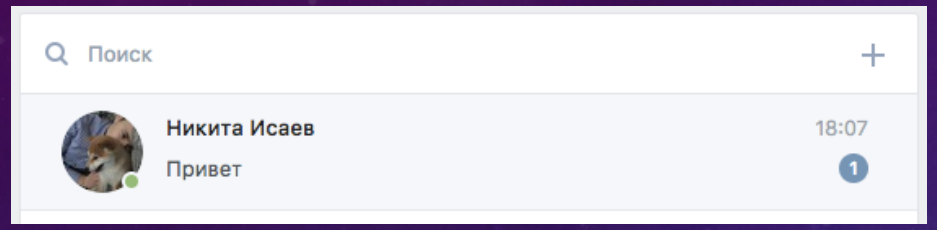
АРХИТЕКТУРА СОВРЕМЕННЫХ JS-ПРИЛОЖЕНИЙ

Приложение = компоненты + State Management

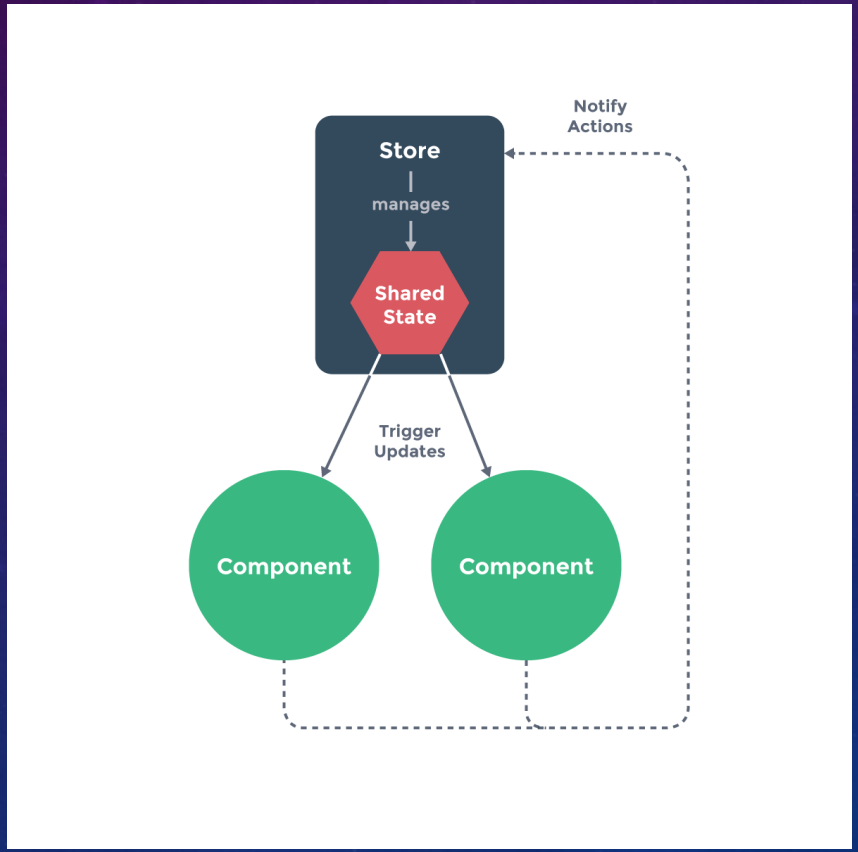
STATE MANAGEMENT



STATE MANAGEMENT

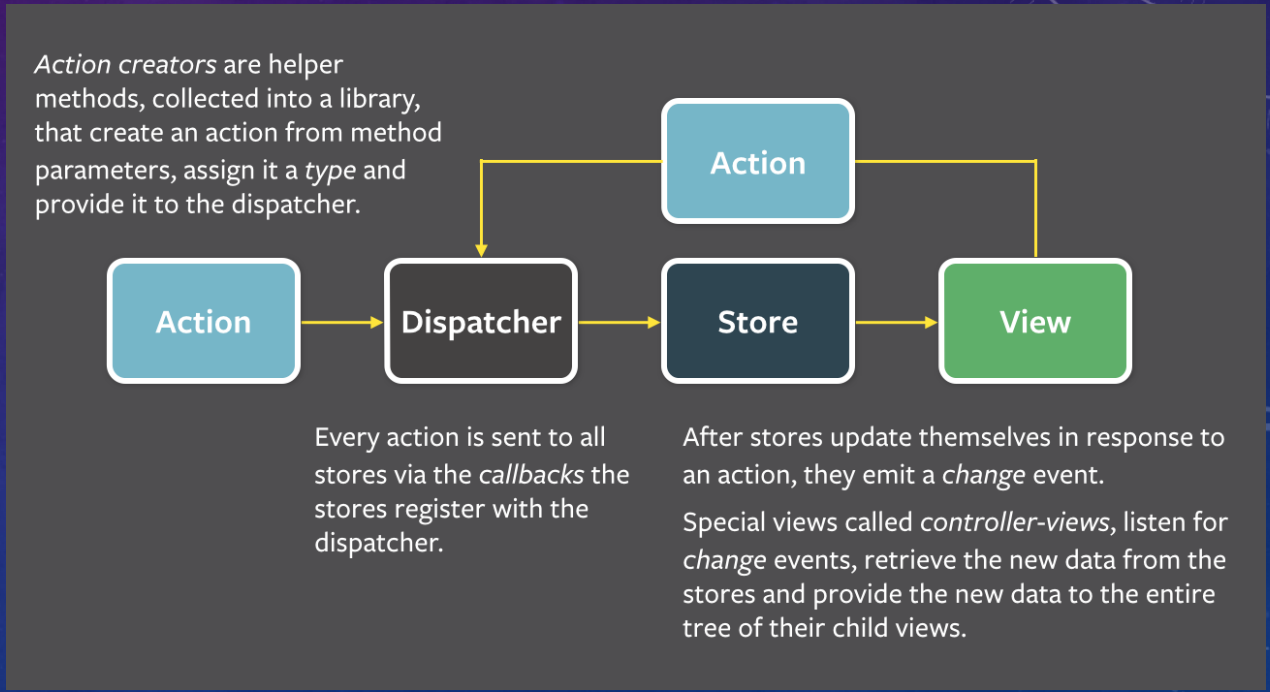


STATE MANAGEMENT: LOCAL + SHARED STATE



STATE MANAGEMENT: FLUX-АРХИТЕКТУРА

- Все состояние в одном месте (+)
- Компоненты отвечают только за отображение (dumb components) (+)
- Логика отображения отделена от данных (-)
- Любое состояние глобально (-)



ФУНКЦИОНАЛЬНЫЕ КОМПОНЕНТЫ

- Меньше писать (+)
- «Чище», проще тестировать, меньше шансов на ошибку (+)

```
class HelloComponent {
  render() {
    return (
      <span>Hello, {this.props.name}</span>
    );
  }
}

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

ФУНКЦИОНАЛЬНЫЕ КОМПОНЕНТЫ

- Меньше писать (+)
- «Чище», проще тестировать, меньше шансов на ошибку (+)
- Нет instance (+-)

```
class HelloComponent {
  render() {
    return (
      <span>Hello, {this.props.name}</span>
    );
  }
}

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

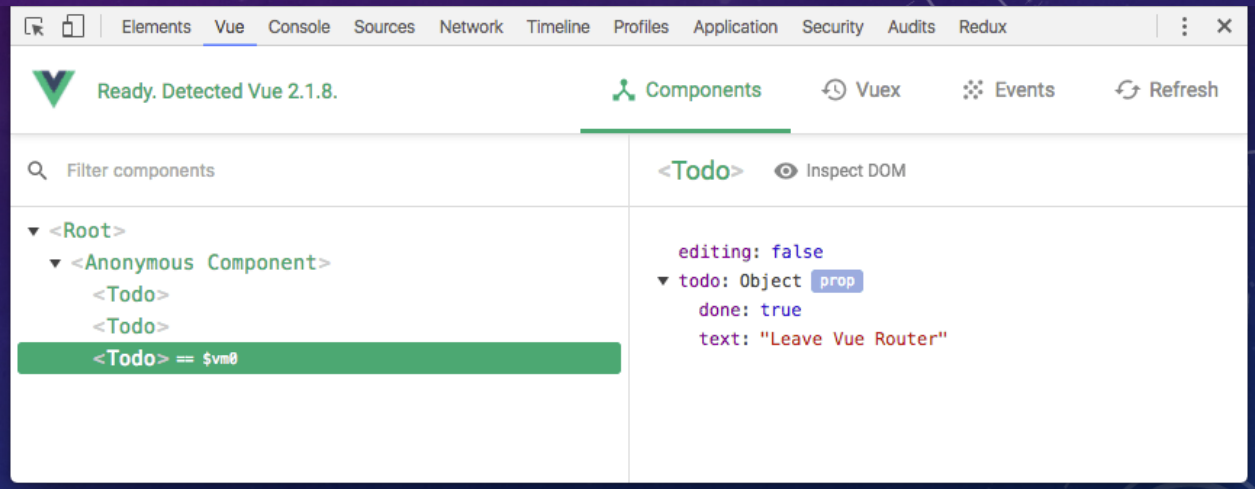
ФУНКЦИОНАЛЬНЫЕ КОМПОНЕНТЫ

- Меньше писать (+)
- «Чище», проще тестировать, меньше шансов на ошибку (+)
- Нет instance (+-)
- Нет LifecycleHooks (-)

```
class HelloComponent {
  render() {
    return (
      <span>Hello, {this.props.name}</span>
    );
  }

  // Impossible in a functional component

  componentDidMount() {
    console.log('mounted!');
  }
}
```

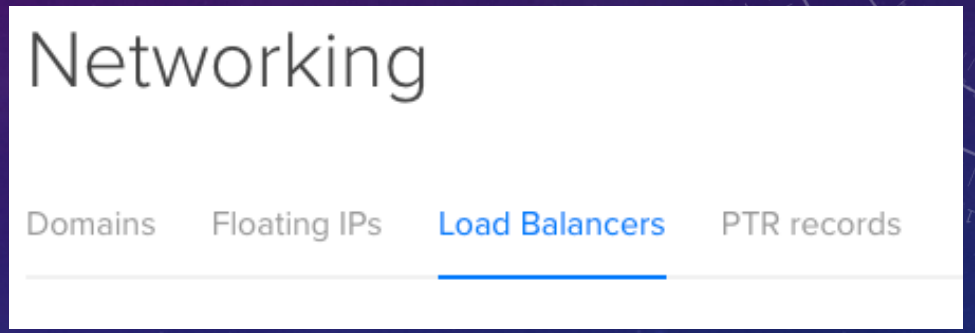
ФУНКЦИОНАЛЬНЫЕ КОМПОНЕНТЫ

- Меньше писать (+)
- «Чище», проще тестировать, меньше шансов на ошибку (+)
- Нет instance (+-)
- Нет LifecycleHooks (-)
- Не работают инструменты отладки в Vue (-)



ФУНКЦИОНАЛЬНЫЕ КОМПОНЕНТЫ

- Меньше писать (+)
- «Чище», проще тестировать, меньше шансов на ошибку (+)
- Нет instance (+-)
- Нет LifecycleHooks (-)
- Не работают инструменты отладки в Vue (-)
- Иногда локальное состояние лучше (-)



ФУНКЦИОНАЛЬНЫЕ КОМПОНЕНТЫ

- Меньше писать (+)
- «Чище», проще тестировать, меньше шансов на ошибку (+)
- Нет instance (+-)
- Нет LifecycleHooks (-)
- Не работают инструменты отладки в Vue (-)
- Иногда локальное состояние лучше (-)

```
MyFramework.component('my-menu', {
  functional: true,
  render: () => {
    return (
      // update store at select event
      <menu-selector @select="store.commit('menuSelectorUpdate', $event.selected)"/>
      // use store as a source of truth
      <menu-content :selected="store.menuSelectorSelected"/>
    );
  }
});
```

ФУНКЦИОНАЛЬНЫЕ КОМПОНЕНТЫ

- Меньше писать (+)
- «Чище», проще тестировать, меньше шансов на ошибку (+)
- Нет instance (+-)
- Нет LifecycleHooks (-)
- Не работают инструменты отладки в Vue (-)
- Иногда локальное состояние лучше (-)

```
MyFramework.component('my-menu', {
  // The data model is not separated from the component
  // Data that only make sense for this component are stored locally
  state: () => {
    return {
      selected: 0
    }
  },
  methods: {
    changeSelected(selected) {
      this.selected = selected;
    }
  }
  render: () => {
    return (
      // update store at select event
      <menu-selector @select="changeSelected($event.selected)"/>
      // use store as a source of truth
      <menu-content :selected="selected"/>
    );
  }
});
```

STATE MANAGEMENT

- Много вариантов
- Нужно связывание со Store
- Идеален для Flux

- Можно что-то придумать
- Нужно связывание со Store

- Есть свое решение
- Другие варианты тоже подойдут
- При использовании Vuex связка автоматическая

ПРАКТИКА

- Изучение
- Сообщество
- Сборка
- Тестирование
- SSR
- Мобильные платформы

ИСПОЛЬЗУЙТЕ REACT, ЕСЛИ:

- Любите ФП
- Любите собирать стек на свой вкус
- Разрабатываете для мобильной платформы
- Хотите использовать GraphQL
- У вас сложная логика клиент-сервер

ИСПОЛЬЗУЙТЕ ANGULAR 2, ЕСЛИ:

- Любите Typescript
- Хотите много возможностей «из коробки»
- Не хотите полагаться на сообщество
- Уже есть сложная система сборки
- Разрабатываете «толстое» SPA

ИСПОЛЬЗУЙТЕ VUE.JS, ЕСЛИ:

- Ищете, с чего начать
- Нужен быстрый старт
- Давит идеология
- Важно разделение подходов
- Разрабатываете небольшой проект

Q/A

- Никита Исаев, HolyJS Санкт-Петербург, 2017
- nikis05@mail.ru