ОНЛАЙН-ОБРАЗОВАНИЕ

```
let lesson = {
  id:       '10'
  themes:   ['Flux', 'Redux'],
  date:     '03.04.2018',
  teacher: {
    name:       'Юрий Дворжецкий',
    position: 'Lead Developer’
  }
};
```

# Как меня слышно && видно?

Если нет – напишите, если слышите – смайлик в чат.

OTUS

# Скажите пару слов о React

1. Вопросы?
2. Проблемы?
3. Пожелания?

# О вебинаре:

Что сможем делать после вебинара?

- Ориентироваться во Flux
- Разрабатывать приложения на React любого масштаба (и огроооомные тоже)
- С хранением состояния в Redux
- И не только React-приложения

# План

- Немного о хранении состояния приложения
- Немного о Flux
- Много о Redux
- И best-practices, разумеется.

OTUS

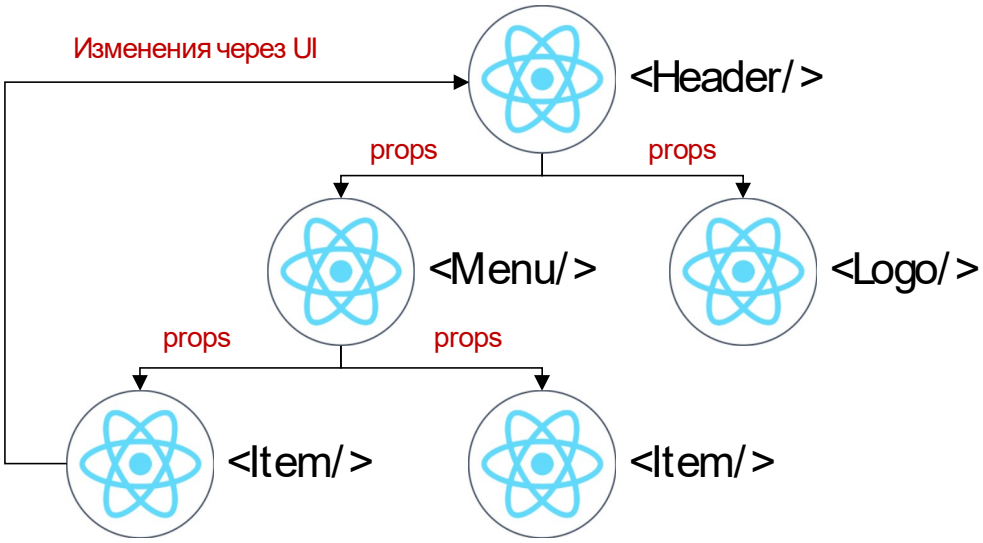
# Состояние приложения

# Упраженение

Дано: App, Page, Header. В Header выводится имя пользователя.

- Где мы будем хранить данные о пользователе?
- Где мы будем их получать?
- Какие + и -

# One-way Data-flow




# One-way Data-flow

```
class Page extends Component {
    constructor() {
        super();
        this.state = {showDialog: false}
    }
    render() {
        return <div>
            <Dialog visible={this.state.showDialog}
                    onClose={
                        () => this.setState({showDialog: false})
                    }/>
        </div>
    }
}

const Dialog = ({visible, onClose}) => {
    return props.visible && <div>
        ...
        <button onClick={onclose}>Закрыть</button>
    </div>
};
```

# Упраженение

Дано: App, Page, Header. В Header выводится имя пользователя.

- Где мы будем хранить данные о пользователе?
- Где мы будем их получать?
- Какие + и -

# Выводы:

- Состояние приложения нужно хранить снаружи
- Но не хочется тащить callback-и
- Хочется просто отслеживать изменения
- Ну и поменьше дефектов тоже

# Вопросы?

# Flux

# Flux

- Flux – архитектура приложений, созданная Facebook
- One-way Data Flow в центре этой архитектуры
- Во Flux есть роли: action, store, dispatcher и view

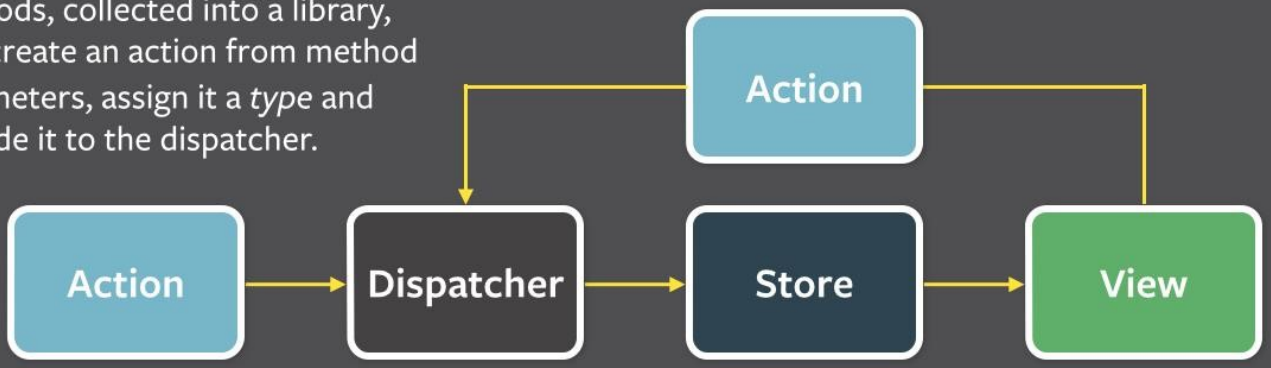
# Flux

- One-Way Data Flow делает Ваши приложения очень простыми, причём, как к понимаю, так и к отладке и дальнейшему расширению.
- Two-Way Data Flow  является причиной каскадных изменений, неэффективности всей клиентской части и источником частых и труднообнаружимых ошибок.

OTUS

# 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.

# Роли Flux: Action

- Action – это просто объект содержащий идентификатор действия (type), и все данные необходимые для совершения действия

```
{
  type: "IncreaseCount",
  local_data: {delta: 1}
}
```

- Разные действия имеют разный атрибут type

OTUS

# Упраженение

Придумайте Ваши объекты-экшены и напишите их в чат

```
{
  type: "IncreaseCount",
  local_data: {delta: 1}
}
```

```
{
  type: "IncreaseCount",
  delta: 1,
  silent: true
}
```

# Роли flux: store

- Store содержит логику приложения и его текущее состояние
- Можно считать, что store отвечает за обработку какой-то области приложения
- Очень похож на модель из MVC, но это не модель. Модель хранит какой-то один объект, а store может хранить  состояние всего приложения.

# Роли flux: store

- Store обновляется callback-ом из диспатчера. Callback получает action как параметр
- Как store будет обновлён, рассылается уведомление слушающим view, которые перерисовывают себя с новыми данными.

# Роли Flux: dispatcher

- Dispatcher – центральный хаб . Он обрабатывает action-ы, и вызывает callback-и со store, в котором он зарегистрирован
- Диспатчер, это не совсем контроллер из MVC, он не содержит никакой логики по обработке данных – только слепое выполнение

# Роли flux: dispatcher

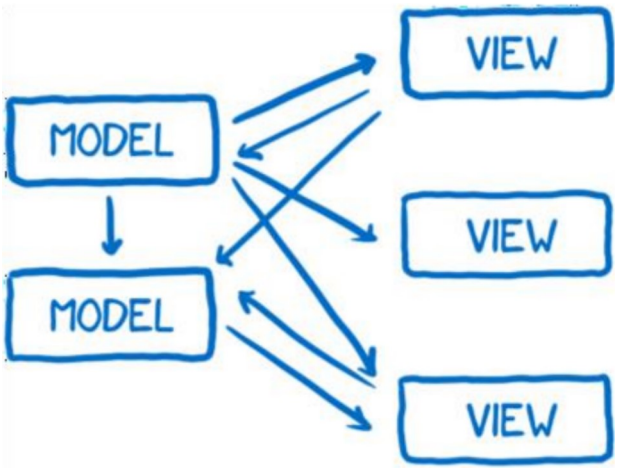
Диспатчер это не Publisher/Subscriber паттерн:

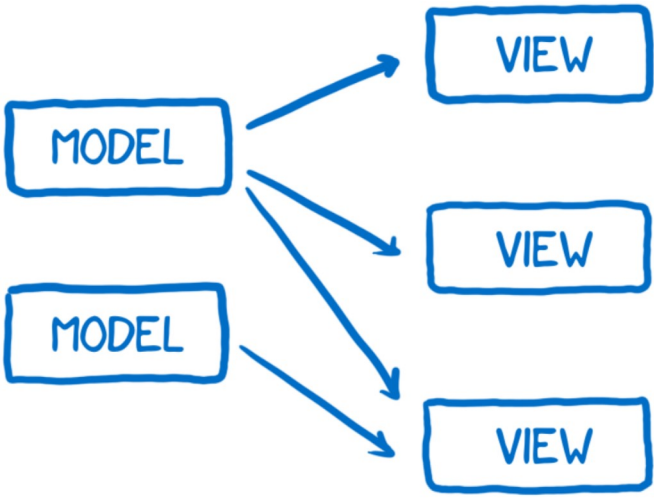
- Callback ни на кого не подписываются
- Callback-и могут простаивать, пока другие выполняются

# Роли Flux: View

- В качестве view обычно выступают react-компоненты
- Наподобие MVC,  View подписываются на обновление данных, и обновляют себя при их изменении
- View могут добавлять новые действия  в диспатчер
- С react-овским one-way data flow работает особенно хорошо и просто

# Без Flux, с Flux

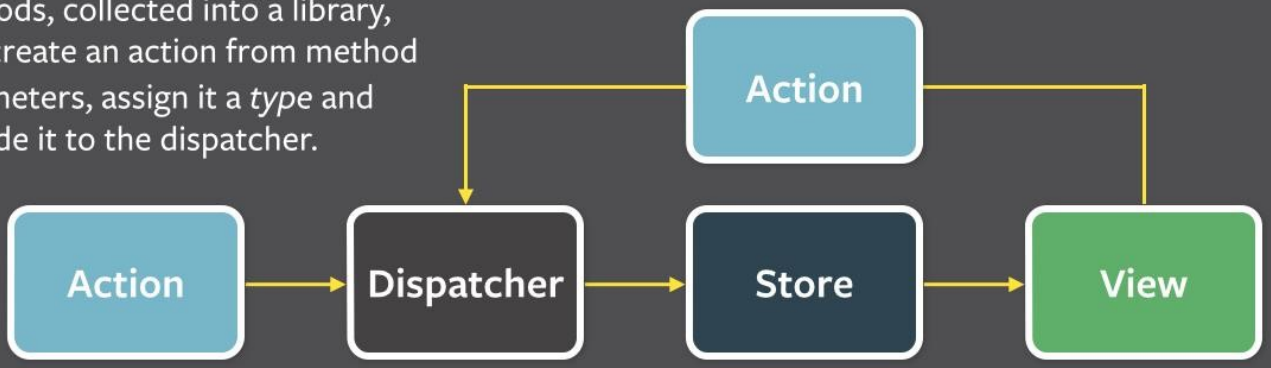


OTUS

# 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.

OTUS

# Упражнение

- Опишите плюсы Flux по картнкам

# Вопросы?

OTUS

# Redux

# Реализации Flux

- Flux Вы можете реализовать самостоятельно, собственно, это только архитектурный паттерн
- А можете воспользоваться одной из готовых реализаций:

**Facebook Flux**, Fluxible by Yahoo, Reflux, Alt, Flummox, Marty.js, McFly, Lux, Material Flux, **Redux**, Redux + Flambeau, Nuclear.js, Fluxette, Fluxxor, Freezer, Fluxury

# Redux

- Flux – потрясающий паттерн. Single direction data flow позволяет точно знать, что происходит с данными.
- Если Вы пишете с Flux, нужно совсем немного времени, чтобы разобраться, что происходит.
- Но если Вы используете чистый Flux, то у Вас появляется масса вспомогательного кода (boilerplate).
- Так и появился Redux

# Immutability

- Самый простой путь сравнить два объекта на равенство – это сравнить их ссылки.
- Вместо глубокого сравнения: _.isEqual(object1, object2)
- Хочется просто написать: object1 === object2
- Но это верно только для неизменяемых объектов (immutable)

# Immutability

- Можно реализовывать такие объекты с Immutable.js или React.addons.update

- А можно просто следовать правилу: “If you change it, replace it.”

Упражнение:
Напишите immutable животное, которое может быть голодным/сытым и может есть

# Immutability

Добиться immutability можно используя хорошие функции (а плохие – не использовать):

Для массивов хорошие:

- […arrays],
- concat, slice, splice
- filter, map

Упражнение:
Назовите плохие

Для Объектов хорошие:

- Object.assign()
- {…obj}

# Pure (чистая) function

```
var values = {a: 1};

function impureFunction(items) {
    var b = 1;
    items.a = items.a * b + 2;
    return items.a;
}

var c = impureFunction(values);
```

```
var values = {a: 1};

function pureFunction(a) {
    var b = 1;
    a = a * b + 2;
    return a;
}

var c = pureFunction(values.a);
```

# Redux

- Redux это предсказуемый контейнер для данных в JS приложениях
- Redux следует идеям Flux, но заметно сокращает сложность
- Redux не касается рендеринга, роутинга, безопасности и type-safety

# Redux

Основные идеи:

- Single source of truth – Состояние всего приложения хранится в единственном store, и только в нём.
- State является read-only – Единственный способ изменить состояние – создать новое.
- Изменения осуществляются чистыми функциями (pure) . Для того, чтобы задать как состояние меняется, вы пишите чистые редьюсеры (reducers).

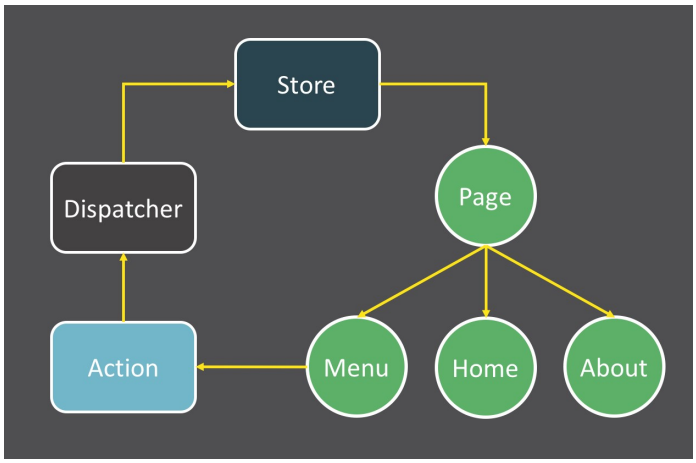
# Redux (пример)

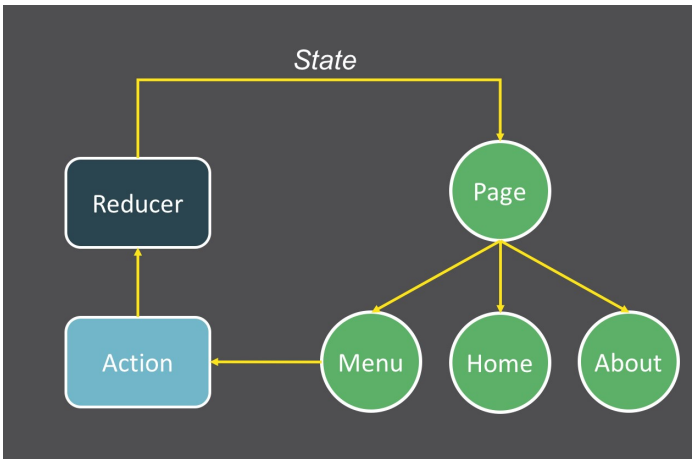
Упражнение:
Угадайте что здесь что?

```
function counter(state = 0, action) {
    switch (action.type) {
        case 'INCREMENT':
            return state + 1
        case 'DECREMENT':
            return state - 1
        default:
            return state
    }
}
```

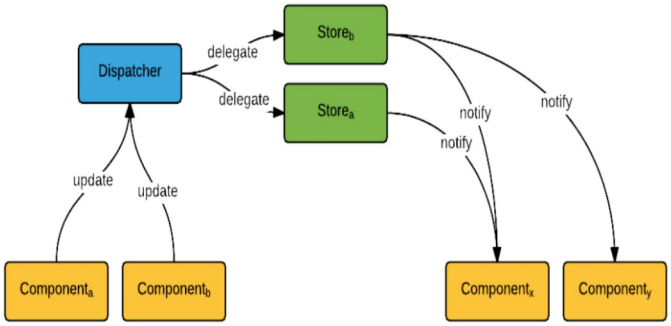
```
let store = createStore(counter)
store.subscribe(
    () => console.log(store.getState())
)
store.dispatch({type: 'INCREMENT'}) //1
store.dispatch({type: 'INCREMENT'}) //2
store.dispatch({type: 'DECREMENT'}) //1
```

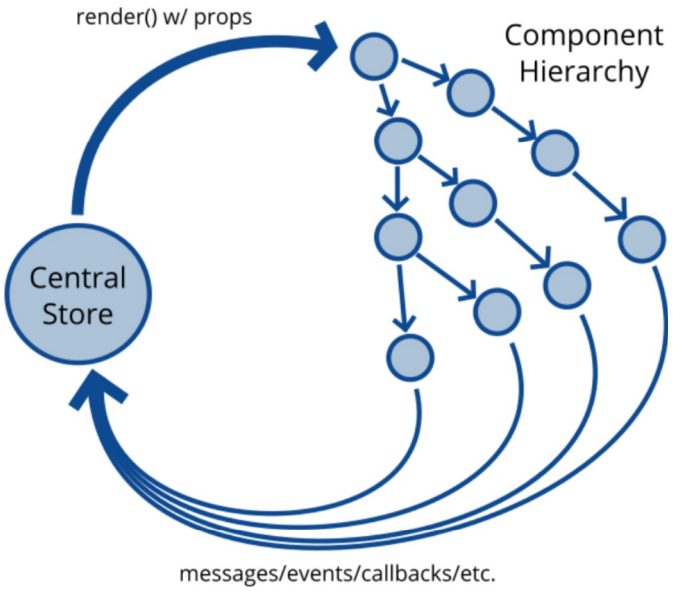
# Flux vs Redux: Flow (нет диспатчера)




# Flux vs Redux: один store




# Как сделать один store (reducer)?

```
const todos = (state = [], action) => {
    switch (action.type) {
        case 'SOME_ACTION':
        //DO SOMETHING HERE
        default:
            return state;
    }
};
const visibilityFilter = (state = 'SHOW_ALL', action) => {
    switch (action.type) {
        case 'SET_VISIBILITY_FILTER':
            return action.filter;
        default:
            return state;
    }
};
```

Упражнение:
Угадайте что здесь что?

```
const todoApp = (state = {}, action) => {
    return {
        todos: todos(state.todos, action),
        visibilityFilter: visibilityFilter(
           state.visibilityFilter, action
        )
    };
};
```

# Store

- Это просто объект, хранит состояние всего приложения
- Состояние получается через getState();
  Изменяется только через dispatch(action);
- Листенеры регистрируются через subscribe(listener);
- Store должен быть один на всё приложение

# Store

- Создаётся с помощью createStore
- Для создания требуется редьюсер (один, смерженный из всех остальных)
- Дальше с ним можно работать как с обычным объектом

```
import {createStore} from 'redux'
import todoApp from './reducers'

let store = createStore(todoApp)
```

# Роли Flux: Action

Action – это просто объект содержащий идентификатор действия (type), и все данные необходимые для совершения действия (факт действия)

```
{
  type: "IncreaseCount",
  delta: 1,
  silent: true
}
```

- Они передаются в метод dispatch
- И приходят в параметры reducer-а

# Reducer

- Редьюсеры, это то, что объединяет actions и store
- Action описывает сам факт действия, но не говорит, о том, что нужно сделать, а вот редьюсер как раз и делает.
- Редьюсер – это чистая функция, которая принимает предыдущий state из store и action, а возвращает новый state

# Reducer

- Поэтому и называется редьюсер:
  (previousState, action) => newState

// это сейчас был reduce

- Очень важно, чтобы редьюсер был чистой функцией: не осуществлял изменния аргументов, не осуществлял запросов, не использовал Date.now() или Math.random()

OTUS

# Ещё раз пример редьюсера

- 0 – это initialState
- Принимает текущее состояние (число) и экшн
- Возвращает новое состояние

```
function counter(state = 0, action) {
    switch (action.type) {
        case 'INCREMENT':
            return state + 1
        case 'DECREMENT':
            return state - 1
        default:
            return state
    }
}
```

# Упражнение

- Напишите редьюсер для приложения-калькулятора (арифметические операции, ресет)

- Какой state? Initial?
- Какие типы action-ов?

# Как сделать один store (reducer)?

```
const todoApp = (state = {}, action) => {
    return {
        todos: todos(state.todos, action),
        visibilityFilter: visibilityFilter(
           state.visibilityFilter, action
        )
    };
};
```

Упражнение:
Угадайте что здесь что?

```
import {combineReducers} from 'redux'

export default combineReducers({
    reducer1,
    reducer2
})
```

OTUS

# Middlewares

- Middlewares в Redux - это что в других технологиях называется third-party extensions
- Redux позволяет подключить эти third-party extension в тот момент, после того, как action отправили в dispatch, но до того, как этот метод попал в reducer.
- Разберём пример логгирования экшенов и следующего состояния store

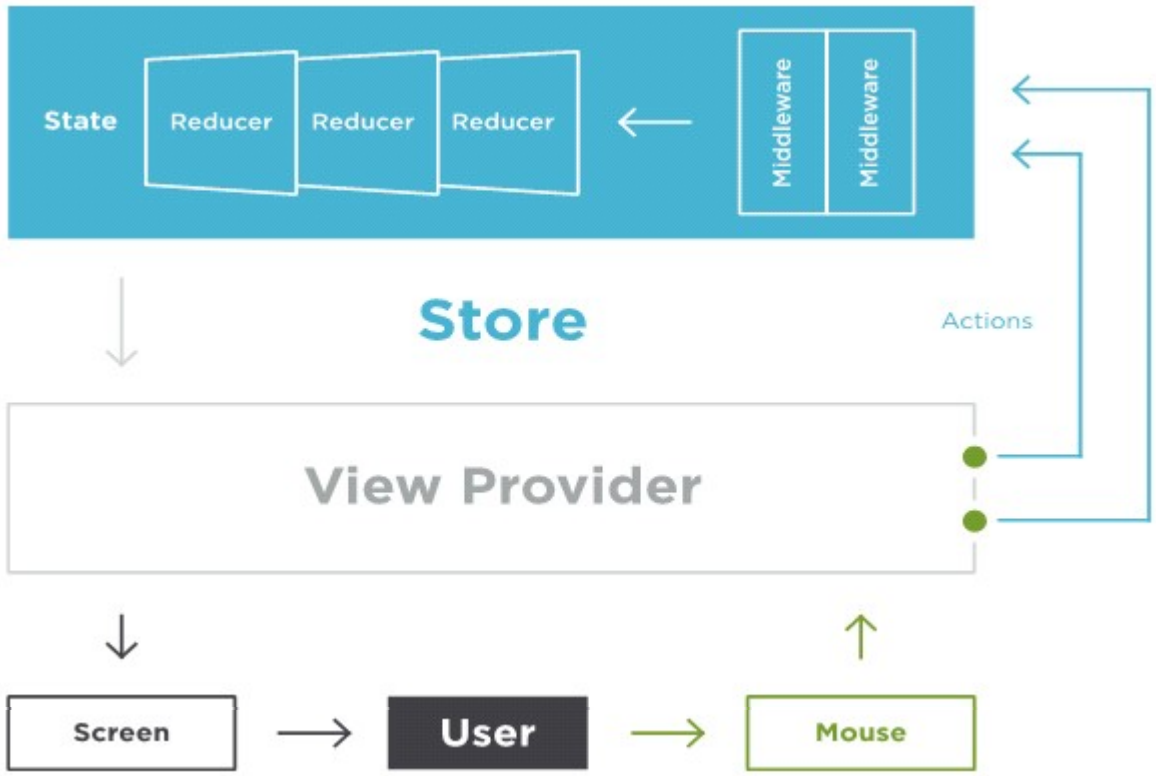
# Thunk Middleware

Упражнение:
Угадайте где actions?

```
function fetchTodos(url) {
    return dispatch => {
        dispatch(startFetchingTodos(url))
        return fetch(url)
            .then(response => response.json())
            .then(json => dispatch(todosReceived(url, json)))
    }
}

store.dispatch(fetchTodos("http://awesomePlaceWithTodos"))
```

# Итоги (промежуточные)




# Пример (App)

```
import {createStore} from 'redux'
import {Provider} from 'react-redux'
import todoApp from './reducers'

let store = createStore(todoApp);


const App = () => (
    <Provider store={store}>
        <MyRootComponent />
    </Provider>
);

ReactDOM.render(<App/>, rootEl);
```

# Пример (App)

```
import {createStore} from 'redux'
import {Provider} from 'react-redux'
import todoApp from './reducers'

let store = createStore(todoApp);


const App = () => (
    <Provider store={store}>
        <MyRootComponent />
    </Provider>
);

ReactDOM.render(<App/>, rootEl);
```

# Пример (connect)

```
import { connect } from 'react-redux'
import { toggleTodo } from '../actions'

const mapStateToProps = state => ({
    todos: state.todos
})

const mapDispatchToProps = dispatch => ({
    toggleTodo: id => dispatch(toggleTodo(id))
})
const TodoList = ({ todos }) => (
    <ul>{todos}</ul>
);

export default connect(
    mapStateToProps, mapDispatchToProps
)(TodoList)
```

# Пример (connect)

```
import { connect } from 'react-redux'
import { toggleTodo } from '../actions'

const TodoList = ({ todos }) => (
    <ul>{todos}</ul>
);

export default connect(
    state => ({
        todos: state.todos
    }),
    dispatch => ({
        toggleTodo: id => dispatch(toggleTodo(id))
    })
)(TodoList)
```

# Структура папок

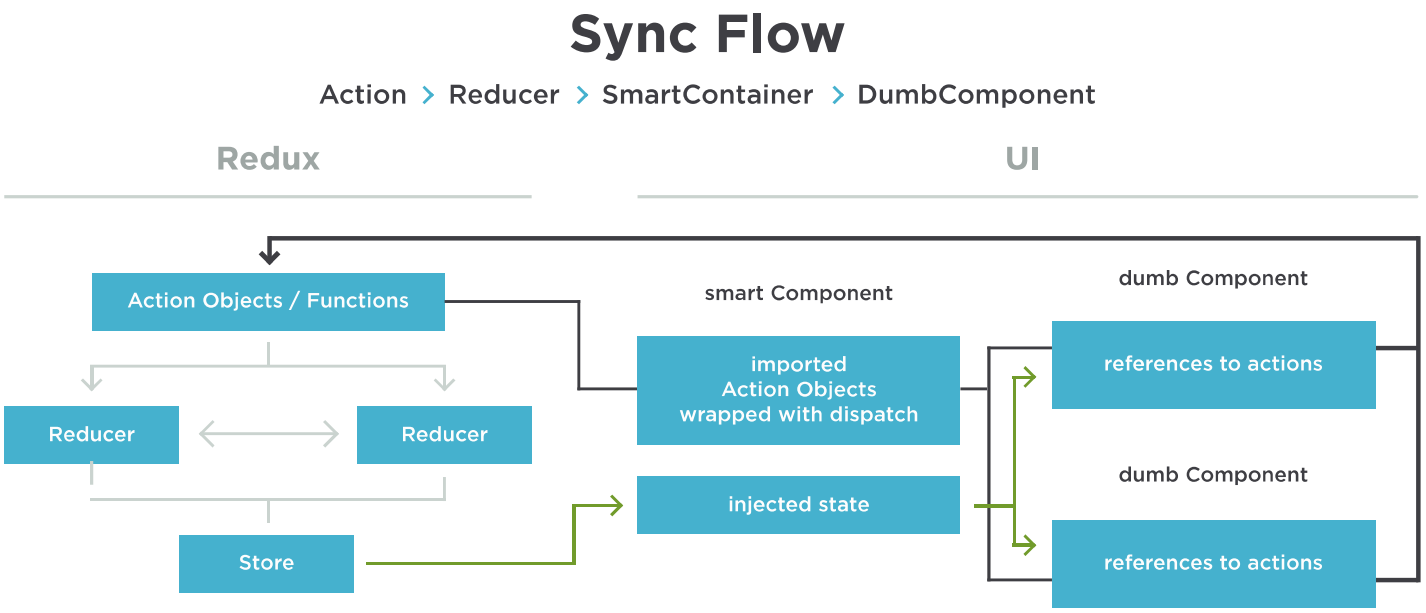
В Redux приложении обычно задаётся следующая структура папок

- actions – сюда располагают action creators, а в этих файлах и константы
- components – собственно, React –компоненты
- reducers – редьюсеры
- store – создание store и middleware

# Структура папок (professional)

- actions – сюда располагают action creators, а в этих файлах и константы
- constants – константы экшенов
- components – простые React –компоненты (не знают про Redux)
- containers – React-контейнеры (знают о Redux, connect)
- reducers – редьюсеры
- store – создание store и middleware

# Redux Flow


Sync Flow
Action > Reducer > SmartContainer > DumbComponent
Redux
UI
Action Objects / Functions
Reducer
Reducer
Store
smart Component
imported
Action Objects
wrapped with dispatch
injected state
dumb Component
references to actions
dumb Component
references to actions

# Вопросы?

OTUS

# Домашнее задание

На весь блок React:
Приложение для самостоятельной работы в блоке React - веб-приложение погоды. На странице приложения должна быть возможность добавлять города в список избранных. По каждому городу показывается информация о температуре, ветре, другие параметры.

ДЗ сегодня: Реализовать компонент фильтра и поиска городов. Данные по городам сохранять в браузерном хранилище. Исходные данные хранятся как статичные json файлы.

Дополнительно: получать данные с сервера.

Спасибо за внимание!

Redux Вам в помощь!