Лекция 9
Построение сложного интерфейса пользователя

Дмитрий Зайцев

образование

План на сегодня

- Flux
- Redux
- React-Redux
- Домашнее задание

Минутка бюрократии

- Внимание
- Отметки о посещении занятий
- Обратная связь о лекциях


FLUX

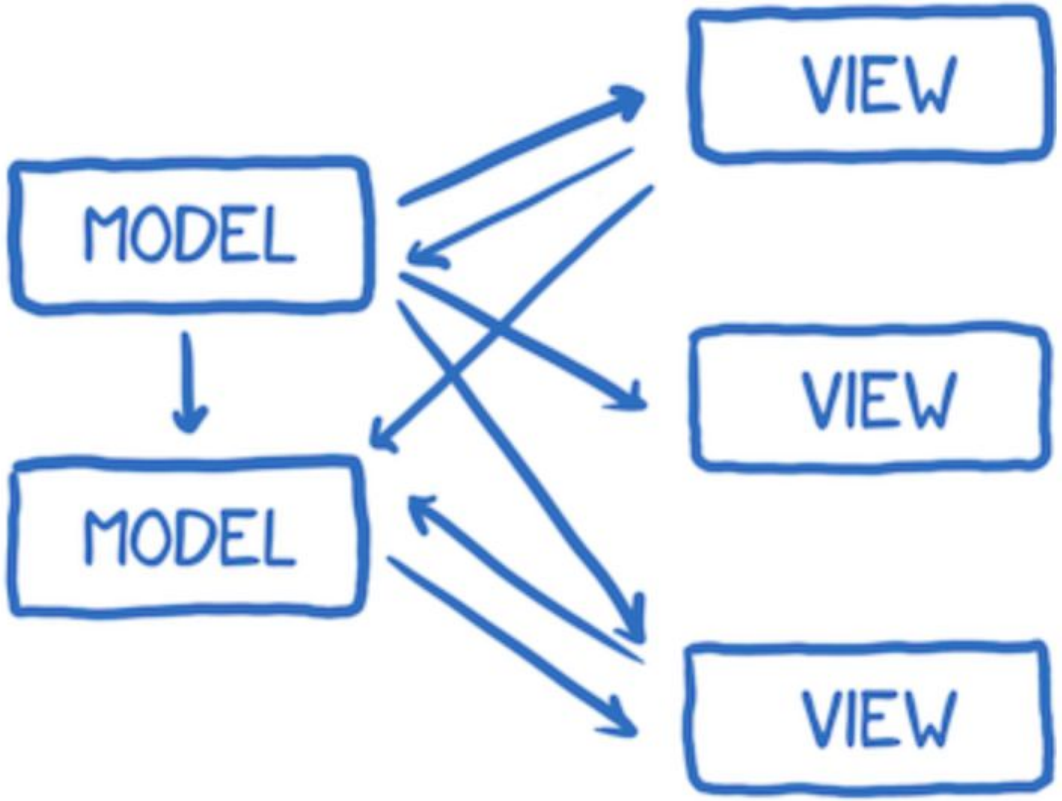
FLUX in a nutshell

1. Представление* создает (dispatch) событие (action).
2. Хранилище данных (store) принимает событие от представления, изменяет данные.
3. Хранилище данных создает событие об изменении данных.
4. Представление принимает событие от хранилища данных и перерисовывает контент с новыми данными.

* Представление – обработчик/хэндлер/вьюха/view

Задача FLUX

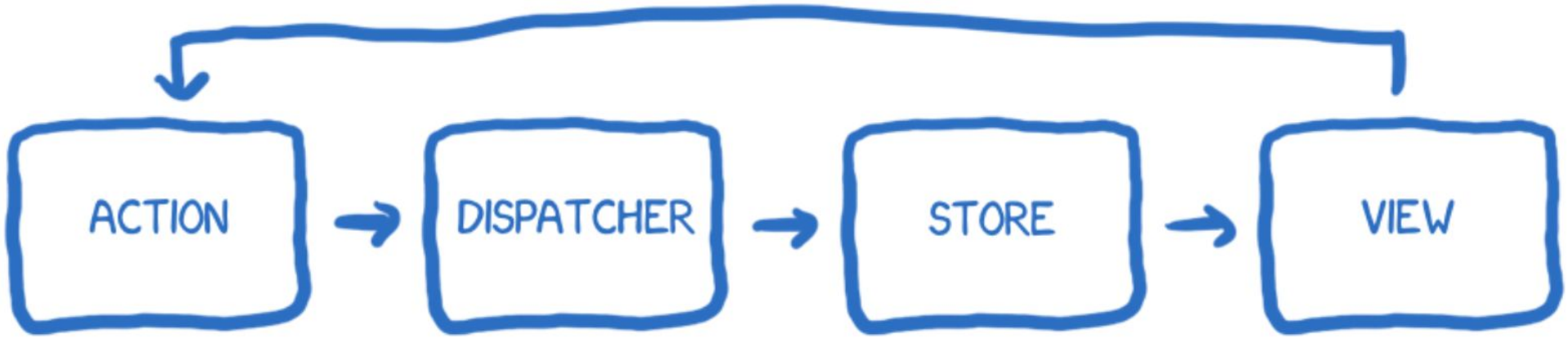
Сделать изменение данных предсказуемым.



Подход FLUX

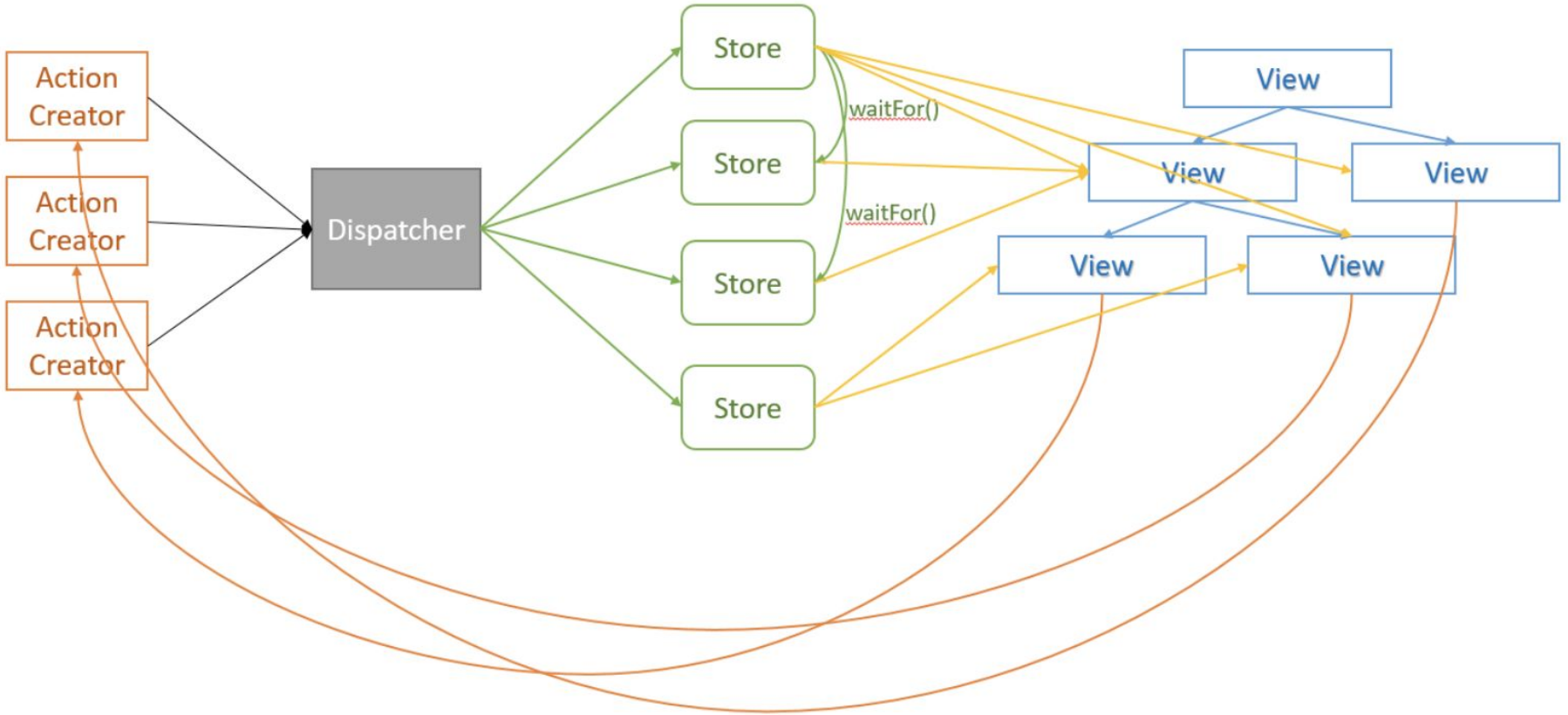
Однонаправленный поток данных.

Схематичное представление



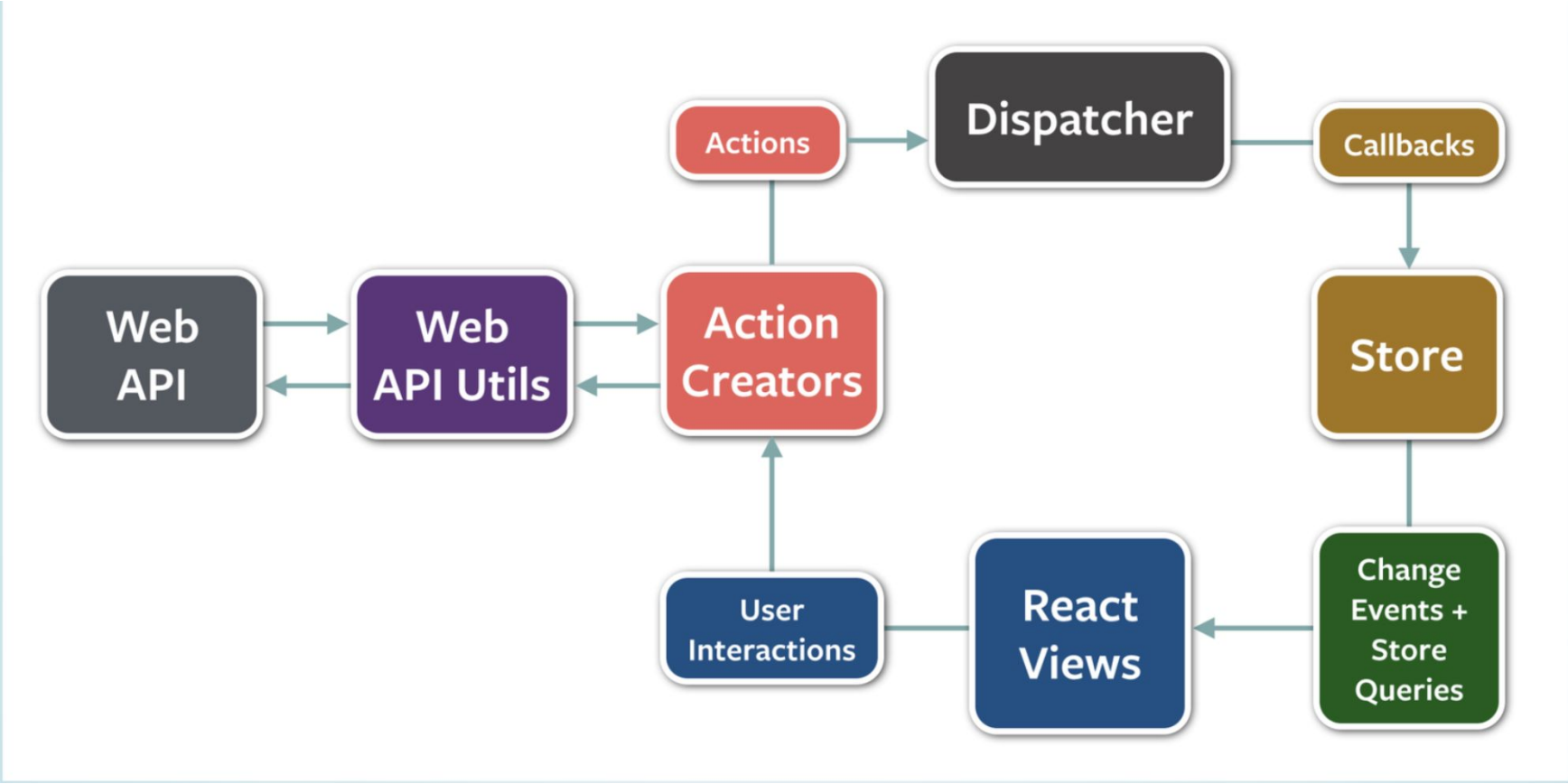
Подход FLUX

Близкое к правде представление



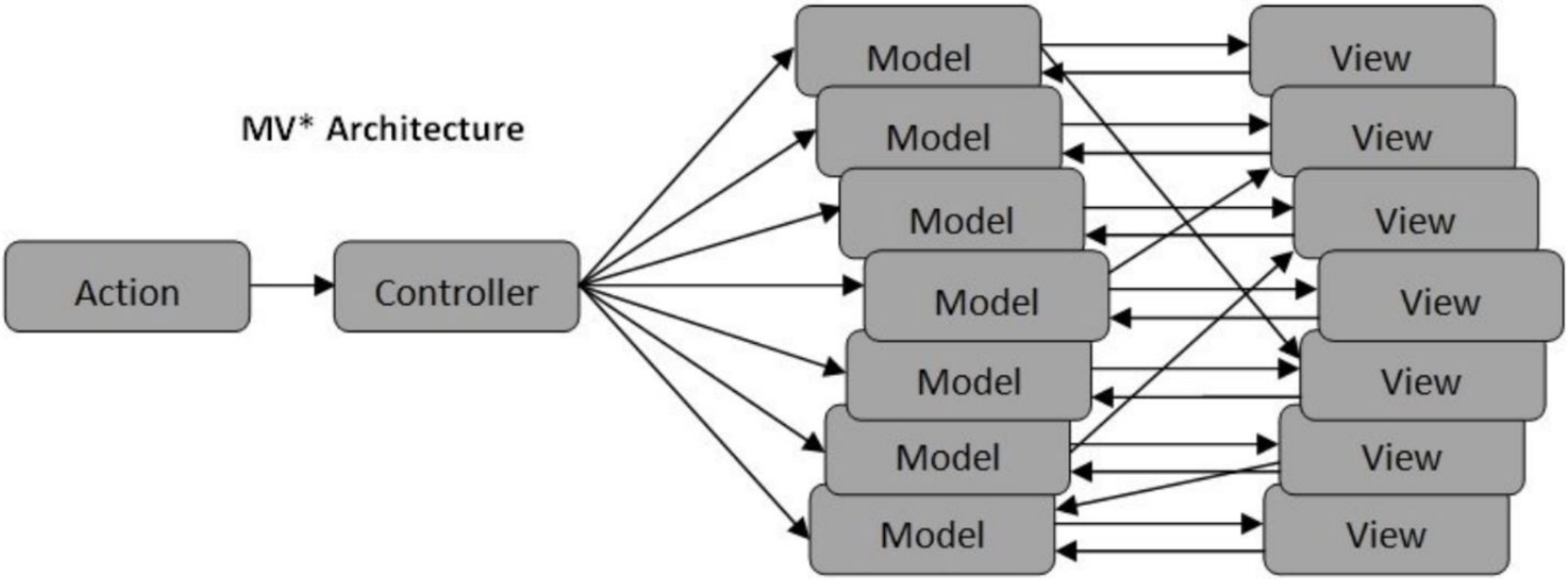
Подход FLUX

Близкое к правде представление

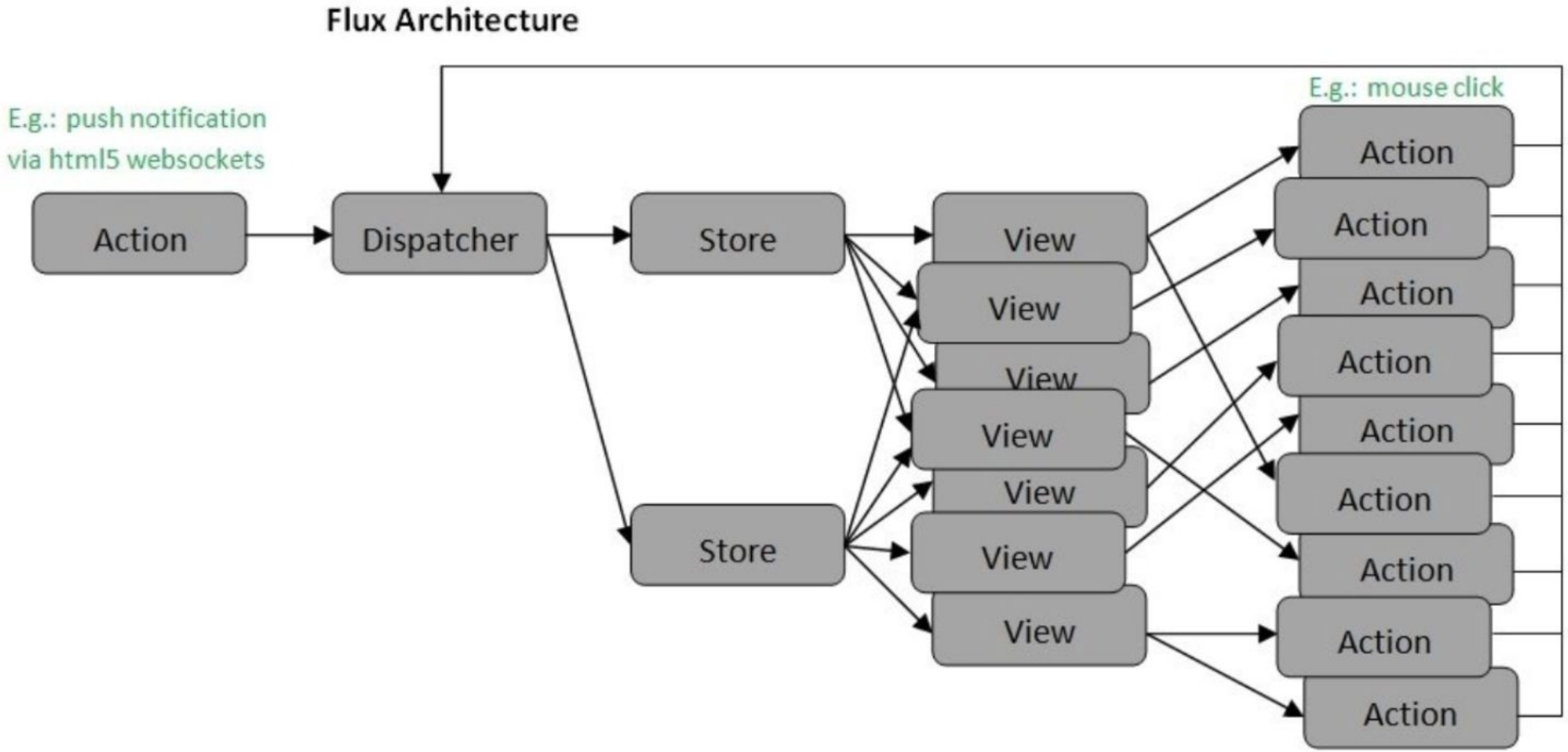


MVC vs FLUX

MVC


MV* Architecture
Action
Controller
Model
Model
Model
Model
Model
Model
Model
View
View
View
View
View
View
View

FLUX


Flux Architecture
E.g.: push notification
via html5 websockets
Action
Dispatcher
Store
Store
View
View
View
View
View
View
View
E.g.: mouse click
Action
Action
Action
Action
Action
Action
Action
Action
Action

FLUX

1. Каждый компонент (обработчик) может создавать событие (action)
2. Все события проходят через один диспетчер
3. Каждое событие может быть обработано несколькими хранилищами (store)
4. Одно хранилище данных может зависеть от другого хранилища
5. Каждое хранилище данных может может влиять на перерендеринг нескольких представлений
6. Каждое представление может зависеть от нескольких хранилищ

Полезные ссылки

1. The case for flux by Dan Abramov: https://medium.com/swlh/the-case-for-flux-379b7d1982c6
2. Flux for stupid people: https://blog.andrewray.me/flux-for-stupid-people/
3. Перевод: https://habr.com/ru/post/249279/
4. Официальный репозиторий Flux: https://github.com/facebook/flux

REDUX

Основные принципы REDUX

1. Состояние приложения хранится в единственном дереве состояний (store)
2. Состояние может изменяться только через вызов событий (actions)
3. Редьюсеры (reducer) – чистые функции, которые возвращают новое состояние (state) в ответ на событие (action)

* Редьюсер - концепция в js, где состояние приложения управляется специальными функциями (редьюсерами). Такие функции принимают на вход текущее состояние и событие, возвращая новый объект состояния.

Ключевые абстракции REDUX

- Store
- Actions, action creator
- reducers

Store

Хранилище состояния приложения; хранилище данных.

Actions

JS объект, описывающий изменение состояния приложения

```
{
  type: FETCH_DATA_SUCCESS,  // у каждого объекта есть тип
  payload: data  // payload - необязательный ключ. Значение может быть любым
}
```

Actions синхронные и асинхронные

По умолчанию Redux поддерживает только синхронные события.

Для доступа к асинхронным событиям, необходимо использовать дополнительные промежуточные слои (middleware):

- redux-thunk
- redux-saga
- redux-observable

Синхронные операции

- Добавление элемента в список по нажатию кнопки "добавить"
- Изменение цвета кнопки
- Вычеркивание элемента из "списка дел"

Асинхронные операции

- Запрос в бд при добавлении элемента в список
- Логирование на удаленный сервер при изменении цвета кнопки
- Установка задержки в 5 сек при вычеркивании элемента из списка

Дополнительно:

- Async flow: https://redux.js.org/advanced/async-flow
- How to dispatch a Redux action with a timeout?

Action creators

Функции, возвращающие action

```
const fetchDataSuccess = (data) => ({
 type: FETCH_DATA_SUCCESS,
 payload: data,
})

const fetchDataFailure = (data) => ({
 type: FETCH_DATA_FAILURE,
 payload: data,  // можно вернуть сразу сообщение с ошибкой: payload: data.message
})
```

Reducers

Редьюсер – чистая функция с двумя параметрами:

- Текущее состояние
- событие, описывающее изменение состояния

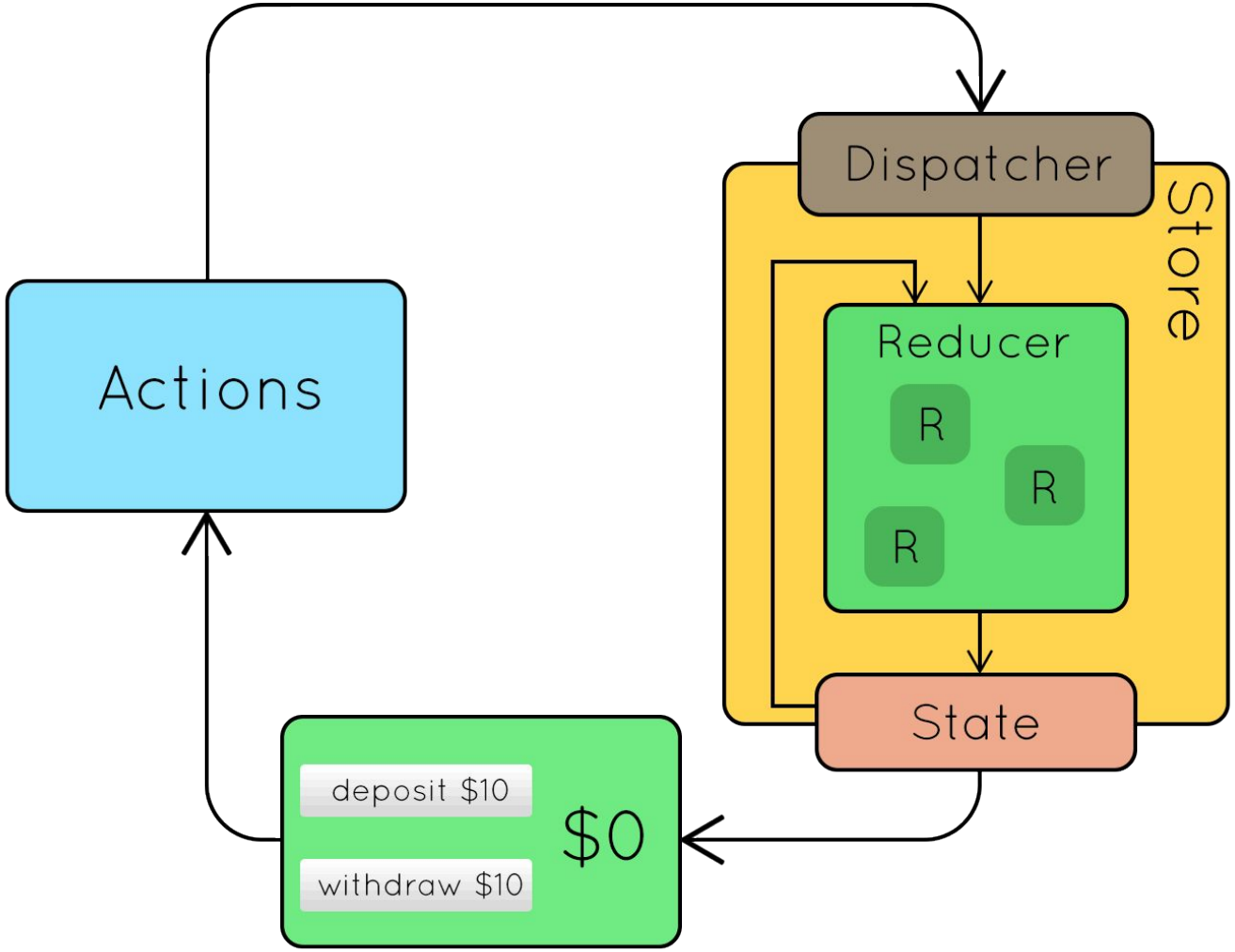
Редьюсер вызывается каждый раз, когда вызывается событие (action)

Reducers

```
const initialState = {
 contacts: [],
 pages: -1
}

export const contacts = (state = initialState, action) => {
 switch (action.type) {
   case FETCH_DATA_SUCCESS:
     return {
       contacts: action.payload.results,
       pages: action.payload.pages,
     }
   case FETCH_DATA_FAILURE:
     toast.error(action.payload.message);
     return {
       message: action.payload.message
     }
   default:
     return state;
 }
}
```

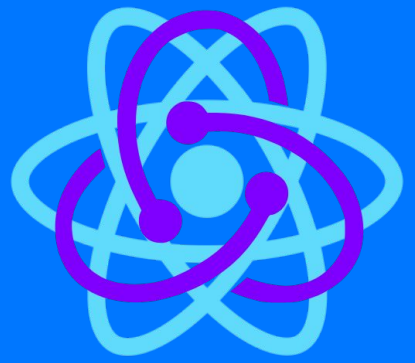
REDUX in a nutshell



Полезные ссылки

1. https://redux.js.org/introduction/getting-started
2. https://redux.js.org/advanced/usage-with-react-router
3. https://github.com/storeon/storeon

REACT + REDUX



REACT без REDUX

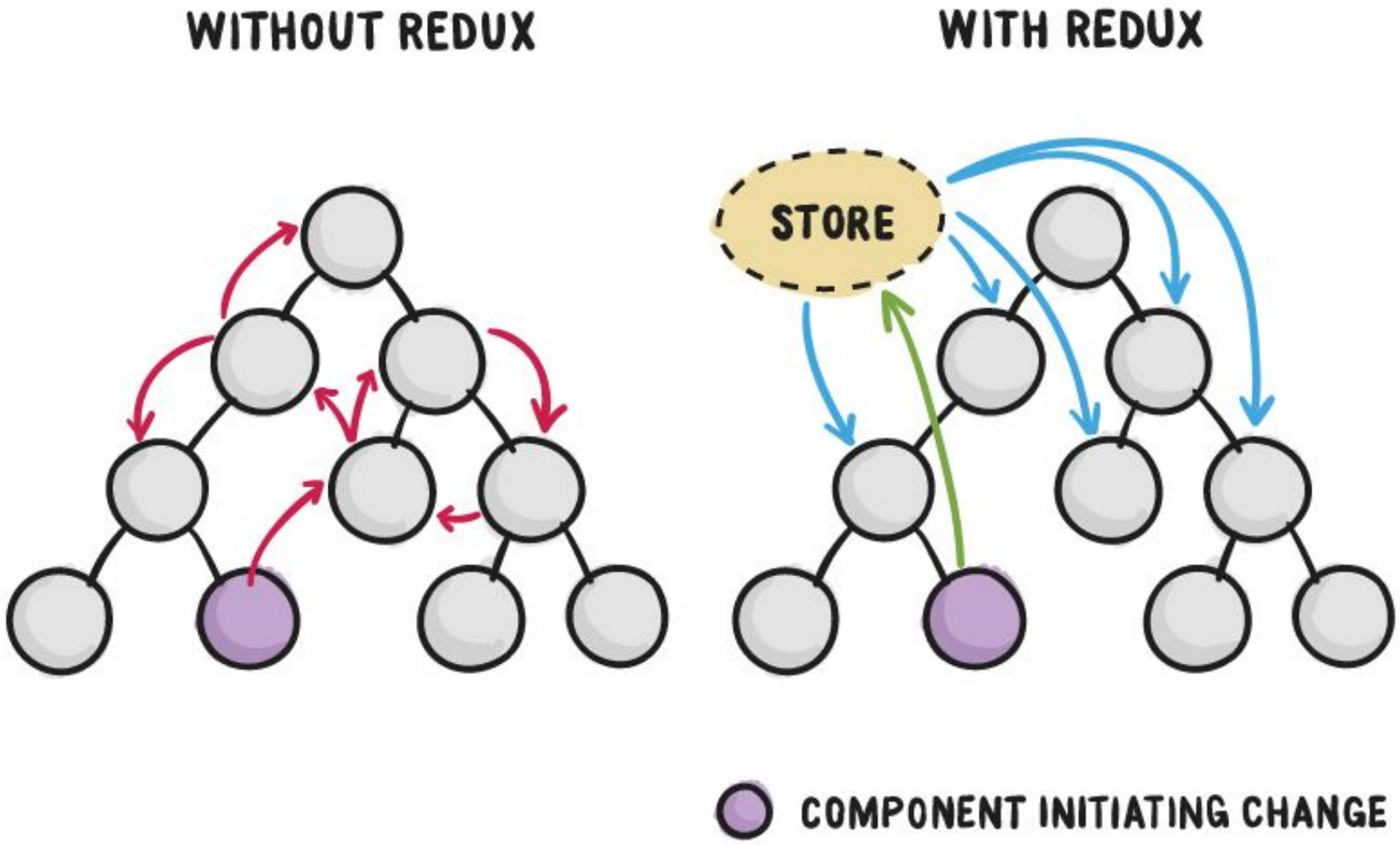
Два способа раздать данные по дочерним элементам:

1. Tunneling, props drilling – пробрасывание props вглубь дерева с целью распределения изменяемых данных между компонентами.
2. Context API

Оба варианта отлично справляются со своими задачами*

* *в небольших приложениях*

REACT + REDUX


WITHOUT REDUX
WITH REDUX
STORE
COMPONENT INITIATING CHANGE

REACT + REDUX

Использование Redux не означает, что ВСЁ состояние приложения должно управляться redux.

Хороший тон:

- React отвечает за UI состояние (пользовательский ввод, открытие модальных окон)
- Redux отвечает за бизнес логику (тема приложения, количество товаров в корзине)

Теория?

Вопросы?

Перерыв! (10 минут)

Препод (с)

Практика

Установка зависимостей

```
$ npm install redux
$ npm install redux-thunk
$ npm install react-redux
```

Создание и импорт action constants

constants/ActionTypes.js или actions/types.js (любое название)

```
// contacts
export const GET_CONTACTS_REQUEST = '@@contacts/GET_CONTACTS_REQUEST';   // можно
export const GET_CONTACTS_SUCCESS = '@@contacts/GET_CONTACTS_SUCCESS';   // задавать
export const GET_CONTACTS_FAILURE = '@@contacts/GET_CONTACTS_FAILURE';   // любое значение
```

```
import {
 GET_CONTACTS_REQUEST,
 GET_CONTACTS_SUCCESS,
 GET_CONTACTS_FAILURE
} from '../constants/ActionTypes'
```

Создание actions

actions/index.js

```
const getContactsSuccess = (contacts) => ({
 type: GET_CONTACTS_SUCCESS,
 payload: contacts
})

const getContactsStarted = () => ({
 type: GET_CONTACTS_REQUEST
})

const getContactsFailure = (error) => ({
 type: GET_CONTACTS_FAILURE,
 payload: { error } // error: error
})
```

```
export const getContacts = () => {
 return (dispatch, getState) => {
   console.log('state: ', getState())
   dispatch(getContactsStarted())

   axios
     .get(`https://tt-front.now.sh/prepods`)
     .then(res => {
       dispatch(getContactsSuccess(res.data))
     })
     .catch(err => {
       dispatch(getContactsFailure(err.message))
     })
 }
}
```

Создание reducers

reducers/contacts.js

```
const initialState = {
 loading: false,
 contacts: [],
 error: null
}
```

```
export default (state = initialState, action) => {
 switch (action.type) {
   case GET_CONTACTS_REQUEST:
     return {
       ...state,
       loading: true
     };
   case GET_CONTACTS_SUCCESS:
     return {
       loading: false,
       error: null,
       contacts: [...state.contacts, action.payload]
     };
   case GET_CONTACTS_FAILURE:
     return {
       ...state,
       loading: false,
       error: action.payload.error
     };
   default:
     return state;
 }
}
```

ROOTREDUCER

reducers/index.js

```
import { combineReducers } from 'redux'
import contacts from './contacts'

export default combineReducers({
 contacts,
})
```

Создание Store

store.js

```
import { createStore, applyMiddleware } from 'redux'
import { composeWithDevTools  } from 'redux-devtools-extension';
import thunk from 'redux-thunk'
import rootReducer from './reducers'

export default createStore(rootReducer, composeWithDevTools({})(applyMiddleware(thunk)))
```

Интеграция в проект

```
import React from 'react';
import { render } from 'react-dom';
import { MainComponent } from './components/MainComponent';
import { BrowserRouter as Router, Route } from 'react-router-dom';
import { Provider } from 'react-redux';

import store from './store';

render(
 <Provider store={store}>
   <Router>
     <Route path="/" component={MainComponent} />
   </Router>
 </Provider>,
 document.getElementById('root')
)
```

Использование в компонентах

```
import React from 'react';
import { connect } from 'react-redux';
import { getContacts } from './actions';

const Contacts = (props) => (
 <>
   <button onClick={() => props.getContacts()}>Get contacts now!</button>
   {props.contacts && props.contacts.length
     ? props.contacts.map(el => (<p key={el.id}>{el.name}</p>))
     : <p>Nothing</p>
   }
 </>
)

const mapStateToProps = (state) => ({
 contacts: state.contacts.contacts,
})

export default connect(
 mapStateToProps,
 { getContacts },
)(Contacts)
```

Альтернативы REDUX-THUNK

- redux-saga
- redux-observable

REDUX-SAGA

SAGAS.JS

```
import { call, put, takeEvery, takeLatest } from 'redux-saga/effects'
import * as Api from actions

function* fetchUser(action) {
  try {
     const user = yield call(Api.fetchUser, action.payload.userId);
     yield put({type: "USER_FETCH_SUCCEEDED", user: user});
  } catch (e) {
     yield put({type: "USER_FETCH_FAILED", message: e.message});
  }
}

function* mySaga() {
 yield takeEvery("USER_FETCH_REQUESTED", fetchUser);
}


function* mySaga() {
 yield takeLatest("USER_FETCH_REQUESTED", fetchUser);
}


export default mySaga;
```

INDEX.JS

```
import { createStore, applyMiddleware } from 'redux'
import createSagaMiddleware from 'redux-saga'

import reducer from './reducers'
import mySaga from './sagas'

// create the saga middleware
const sagaMiddleware = createSagaMiddleware()
// mount it on the Store
const store = createStore(
 reducer,
 applyMiddleware(sagaMiddleware)
)

// then run the saga
sagaMiddleware.run(mySaga)

// ...
```

Больше про redux-saga

1. https://redux-saga.js.org/docs/introduction/BeginnerTutorial.html
2. https://redux-saga.js.org/docs/recipes/

REDUX-OBSERVABLE

```
import { map, mergeMap } from 'rxjs/operators'
import { ofType } from 'redux-observable'

const FETCH_USER = 'FETCH_USER'
const FETCH_USER_FULFILLED = 'FETCH_USER_FULFILLED'
const fetchUser = prepod => ({ type: FETCH_USER, payload: prepod });
const fetchUserFulfilled = payload => ({ type: FETCH_USER_FULFILLED, payload });
const fetchUserEpic = action$ => action$.pipe(
 ofType(FETCH_USER),
 mergeMap(action =>
   ajax.getJSON(`https://tt-front.now.sh/prepods/${action.payload}`).pipe(
     map(response => fetchUserFulfilled(response))
   )
 )
)

const initState = {}

const users = (state = initState, action) => {
 switch (action.type) {
   case FETCH_USER_FULFILLED:
     return {
       ...state,
       [action.payload.prepod]: action.payload
     };

   default:
     return state;
 }
};
```

Больше про redux-observable

1. https://redux-observable.js.org/
2. https://redux-observable.js.org/docs/basics/Epics.html

ФОРМЫ

https://react-hook-form.com/

Вопросы?

Вопросы?

Домашнее задание №9

1. Подключить redux с помощью библиотеки react-redux
2. Определить изменяемые данные, которые нужны в нескольких компонентах (например, сообщения и др.)
3. Объявить эти данные в store
4. Подключить к store компоненты, которые используют эти данные

Расширенное описание задания, подсказки, а также презентации с лекций всегда есть в репозитории.

Срок сдачи

8 декабря

Спасибо за внимание!

образование

Пока!

Присоединяйтесь к сообществам про образование в VK

- VK Джуниор
- VK Образование