REDUX

Алексей Опалев


НА ПОРТАЛЕ
ОТМЕТИТЬСЯ ВАЖНО
imgflip.com

ПЛАН ЛЕКЦИИ

1. TLDR [что нужно сделать дома]
2. Flux
3. Redux
4. React-Redux
5. Домашнее задание

ЧТО ДЕЛАТЬ ДОМА (ДЗ)

ПОДКЛЮЧАЕМ REDUX В ПРОЕКТ

Необходимо:

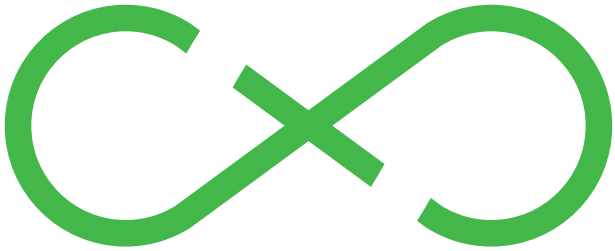
- Подключить redux с помощью библиотеки react-redux
- Определить изменяемые данные, которые нужны в нескольких компонентах
- Объявить эти данные в store
- Подключить к store компоненты, которые используют эти данные

ЧТО ДЛЯ ЭТОГО СДЕЛАТЬ

1. Установить новые зависимости (redux redux-thunk react-redux)
2. Создать constants/ActionTypes.js
3. Создать actions.js
4. Создать reducers/messages.js // и любые другие редьюсеры по желанию
5. Создать reducers/index.js
6. Создать store.js
7. Обернуть приложение в Provider
8. В компонентах функции обернуть в connect

Пример: **https://github.com/track-mail-ru/cra-thunk**

FLUX



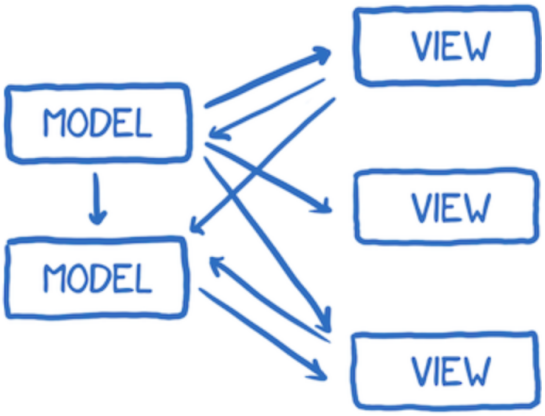
FLUX IN A NUTSHELL

1. Прдеставление* 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

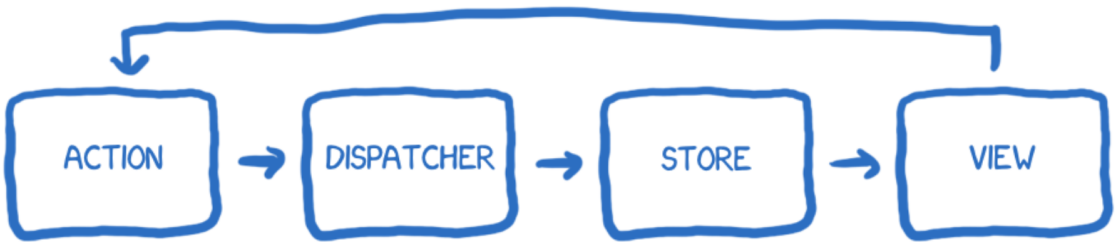
Сделать изменение данных предсказуемым.



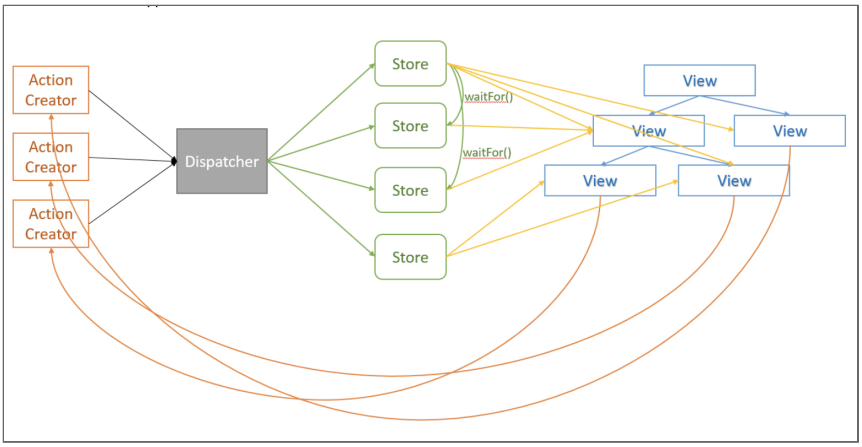
ПОДХОД FLUX

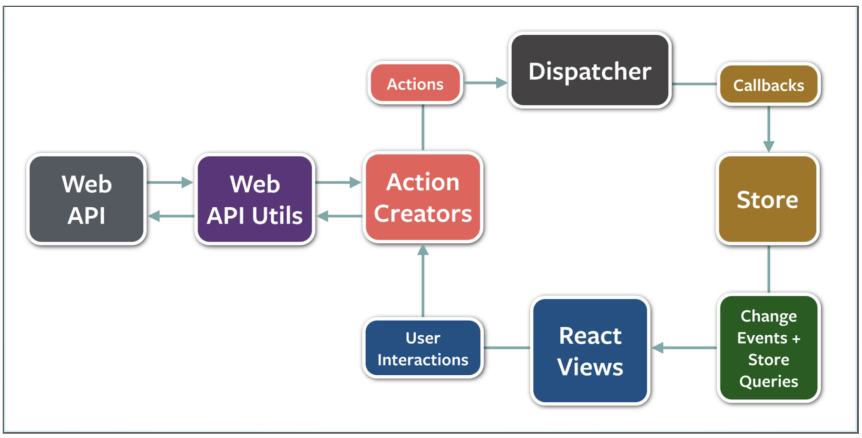
Однонаправленный поток данных.

[Красивое представление]



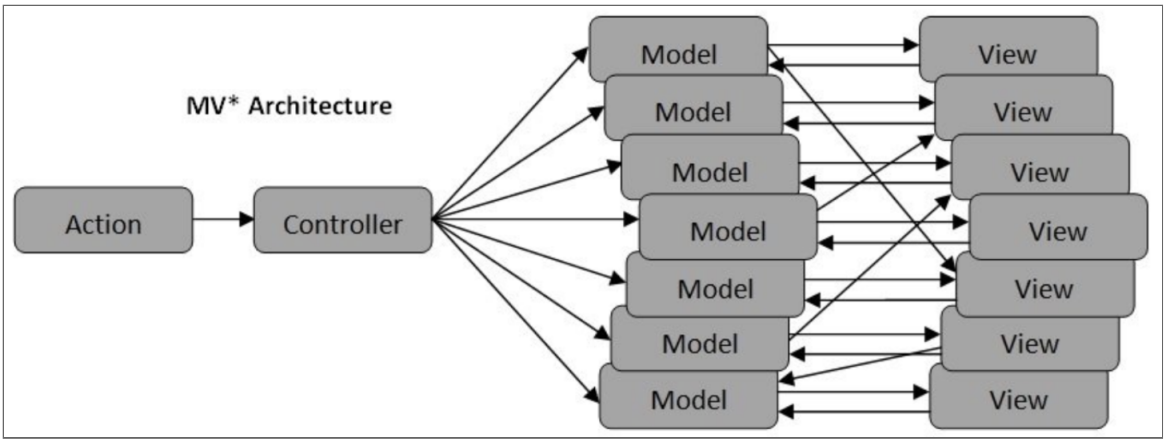
Близкое к правде представление



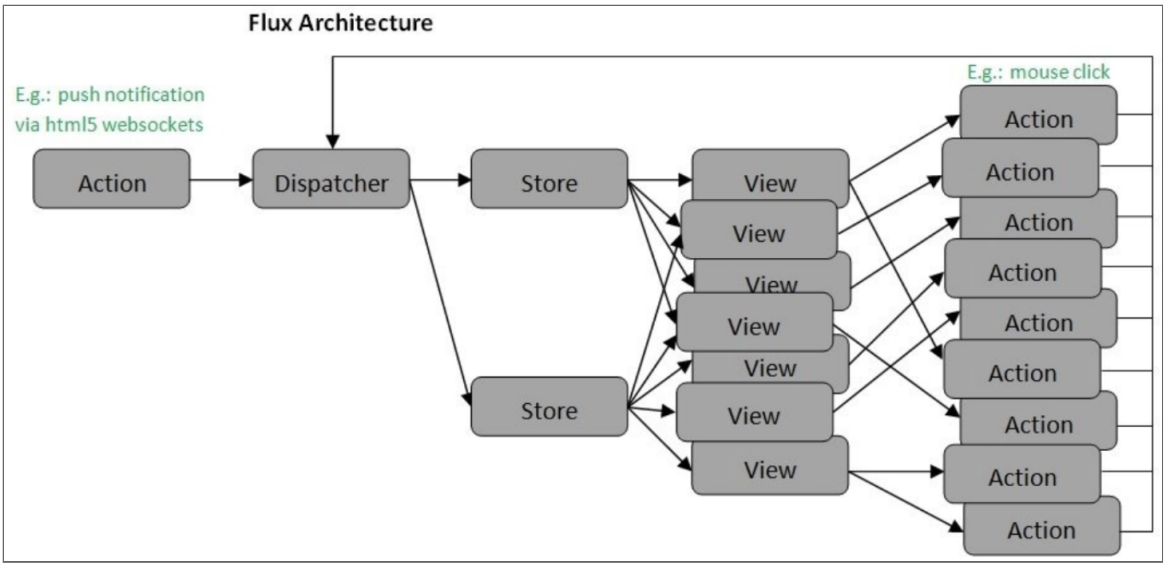

Actions
Dispatcher
Callbacks
Web API
Web API Utils
Action Creators
Store
User Interactions
React Views
Change Events + Store Queries

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
E.g.: mouse click
Action
Dispatcher
Store
Store
View
View
View
View
View
View
View
Action
Action
Action
Action
Action
Action
Action
Action
Action

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

The case for flux by Dan Abramov:

https://medium.com/swlh/the-case-for-flux-379b7d1982c6

Flux for stupid people:

https://blog.andrewray.me/flux-for-stupid-people/

Перевод: **https://habr.com/ru/post/249279/**

Официальный репозиторий Flux:

https://github.com/facebook/flux

REDUX

ОСНОВНЫЕ ПРИНЦИПЫ REDUX

1. Состояние приложеня 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 принмают 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 мож
}
```

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
- логгирование на удаленный сервер при изменении цвета кнопки
- установка задержки в 5 сек при вычеркивании элемента из списка

Async flow:
https://redux.js.org/advanced/async-flow

How to dispatch a Redux action with a timeout?

https://stackoverflow.com/questions/35411423/how-to-dispatch-a-redux-action-with-a-timeout/35415559#35415559

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: pay
})
```

REDUCERS

Редьюсер – чистая функцая с двумя параметрами:

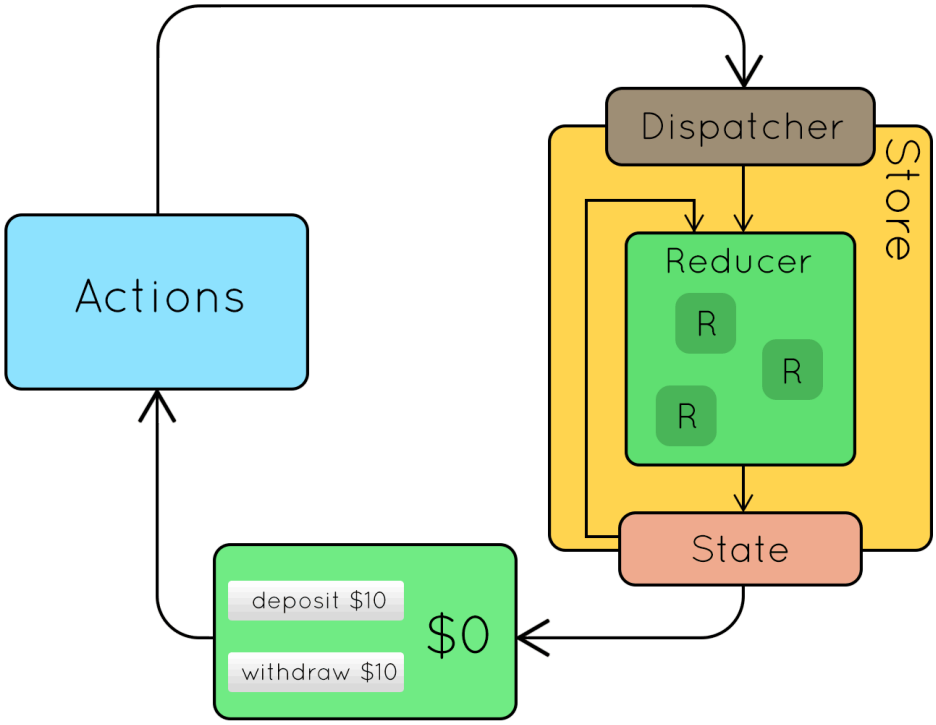
- Текущее состояние)
- событие, описывающее изменеие состояния

Редьюсер вызывается каждый раз, когда вызывается событие (action)

```
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;
  }
}
```

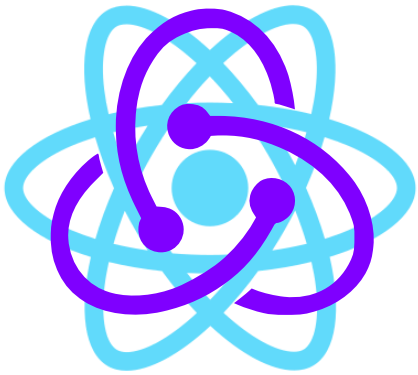
REDUX IN A NUTSHELL



Читать больше

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

REACT + REDUX



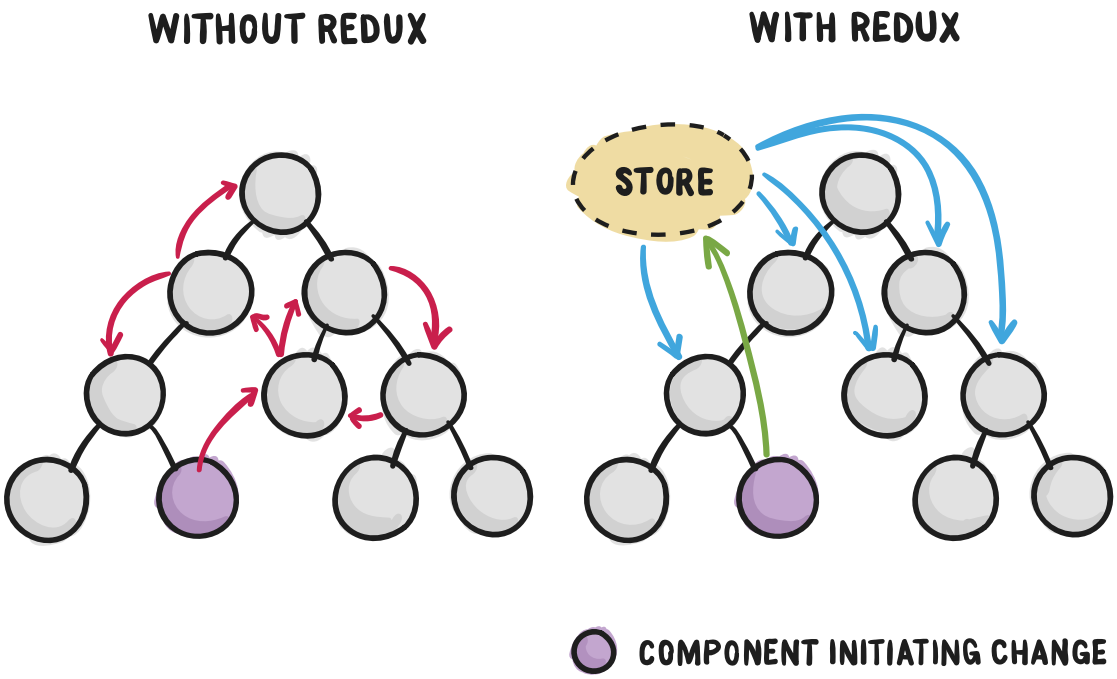
REACT БЕЗ REDUX

Два способа раздать данные по дочерним элементам:

1. Tunneling AKA props drilling – пробрасывание props в глубь дерева с целью расшарить изменяемые данные между компонентами.
2. Context API

Оба варианта отлично справляются со своими задачами*

* в небольших приложениях


WITHOUT REDUX
WITH REDUX
STORE
COMPONENT INITIATING CHANGE

Использование Redux не означает, что ВСЕ 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)

REDUX В КОДЕ

+ Redux-thunk для асинхронных событий

УСТАНОВКА ЗАВИСИМОСТЕЙ

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

СОЗДАНИЕ CONSTANTS

```
// contacts
export const GET_CONTACTS_REQUEST = '@@contacts/GET_CONTACTS
export const GET_CONTACTS_SUCCESS = '@@contacts/GET_CONTACTS
export const GET_CONTACTS_FAILURE = '@@contacts/GET_CONTACTS
```

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

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

СОЗДАНИЕ ACTIONS

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

import axios from 'axios'  // или fetch, или любой другой http

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 = () => {
```

actions/index.js

СОЗДАНИЕ REDUCERS

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

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]
      };
```

reducers/contacts.js

ROOTREDUCER

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

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

reducers/index.js

СОЗДАНИЕ STORE

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

export default createStore(rootReducer, composeWithDevTools(
```

store.js

СВЯЗЫВАНИЕ С ПРОЕКТОМ

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

improt strore 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
    {props.contacts && props.contacts.length
      ? props.contacts.map(el => (<p key={el.id}>{el.name}</
      : <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
import * as Api from actions

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

/*
  Starts fetchUser on each dispatched `USER_FETCH_REQUESTED`
  Allows concurrent fetches of user.
*/
function* mySaga() {
  yield takeEvery("USER_FETCH_REQUESTED", fetchUser);
}

/*
  Alternatively you may use takeLatest.

  Does not allow concurrent fetches of user. If "USER_FETCH_
  dispatched while a fetch is already pending, that pending
```

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

- **https://redux-saga.js.org/docs/introduction/BeginnerTut**
- **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: pr
const fetchUserFulfilled = payload => ({ type: FETCH_USER_FU

const fetchUserEpic = action$ => action$.pipe(
  ofType(FETCH_USER),
  mergeMap(action =>
    ajax.getJSON(`https://tt-front.now.sh/prepods/${action.p
      map(response => fetchUserFulfilled(response))
    )
  )
)

const initState = {}

const users = (state = initState, action) => {
  switch (action.type) {
    case FETCH_USER_FULFILLED:
      return {
        ...state,
```

Больше про redux-observable:

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

ФОРМЫ

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

СПАСИБО ЗА ВНИМАНИЕ