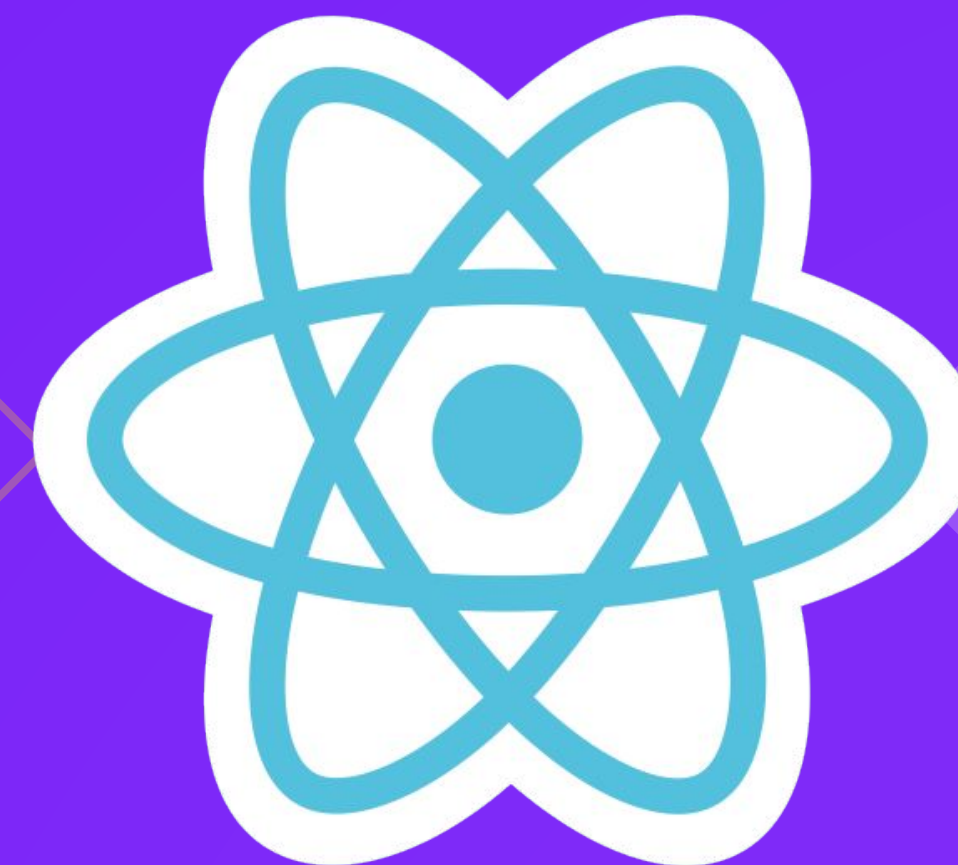




УРОК №9-10

# Использование Redux



# Redux

---

Redux позиционирует себя как предсказуемый контейнер состояния (state) для JavaScript приложений.

Redux - это дополнительная библиотека, с помощью которой будут строиться приложения, реализация модели для представления.

Представлением может быть что угодно, но часто Redux используется в паре с React

# Ключевые моменты Redux

---

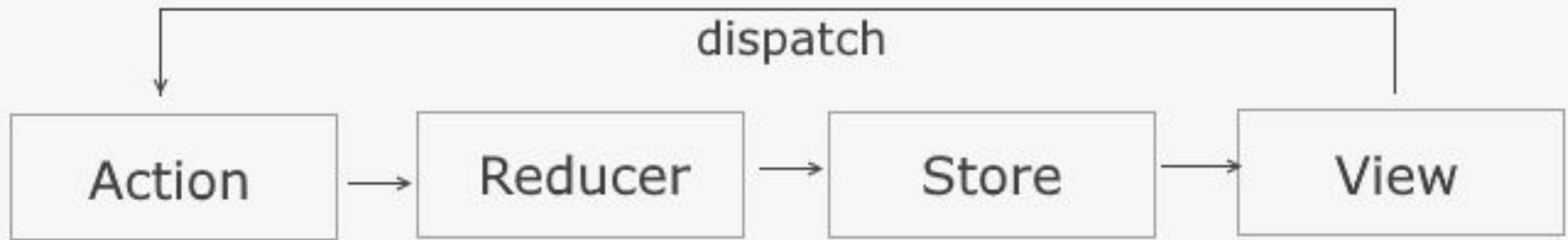
Состояние (state) - состояние приложения

Действия (actions) - объект, описывающий что должно произойти

Редуктор (reducer) - функция, которая получает действие и изменяет состояние

# Принцип работы

---



# Установка

---

npm i redux

npm i react-redux

# Простой пример

---

Создадим приложение “Электронный журнал”, используя React и Redux.  
Для начала реализуем просто список учеников и выставление им оценок.  
Далее будем увеличивать функционал.

# 0. Структура хранилища

---

Для начала прикинем, как будет выглядеть наш объект хранилища.

# 1. Действия

---

Действия - это набор информации, переходящий от приложения к хранилищу.

Действия в Redux - это обычный объект, с обязательным полем `type`. Также необходимо создать генераторы действий - функции, возвращающие `action`.

У нас будет два действия - добавить оценку и удалить оценку.



```
//actions.js
```

```
export const ADD_GRADE = "ADD_GRADE";
```

```
export const DELETE_GRADE = "DELETE_GRADE";
```

```
export function addGrade(index, grade) {
```

```
  return {
```

```
    type: ADD_GRADE,
```

```
    index,
```

```
    grade
```

```
  }
```

```
}
```

```
export function deleteGrade(indexOfStudent, indexOfGrade) {
```

```
  return {
```

```
    type: DELETE_GRADE,
```

```
    indexOfStudent,
```

```
    indexOfGrade
```

```
  }
```

```
}
```

## 2. Редюсеры

---

Редюсеры - это чистые функции, которые изменяют состояние (state) приложения.

То есть, действия просто описывают изменения, а редюсеры уже изменяют их.

Редюсеры принимают в качестве параметра состояние и действие.

Редюсеры должны возвращать **НОВЫЙ ОБЪЕКТ!**

```
//reducers.js
```

```
import {ADD_GRADE, DELETE_GRADE} from './actions.js';
```

```
const prevState = { students: [  
  {name: "Ivanov", grades: [11, 10, 10]},  
  {name: "Arystanuly", grades: [9, 12]}]}
```

```
export function grades(state = prevState, action) {  
  let newState = [...state.students];  
  switch (action.type) {  
    case ADD_GRADE:  
      newState[action.index].grades.push(action.grade);  
      return Object.assign({}, state, {students: newState});  
    case DELETE_GRADE:  
      newState[action.indexOfStudent].grades.splice(action.indexOfGrade, 1)  
      return Object.assign({}, state, {students: newState});  
    default: return state;  
  }  
}
```

## 3. Хранилище

---

Хранилище - это то, что объединит все части нашего приложения.

Хранилище предоставляет доступ к состоянию, содержит это состояние, обновляет при помощи редюсеров состояние, регистрирует слушателей.

Есть методы, которые позволяют работать с хранилищем, но мы рассмотрим работу с React

## 4. Компоненты-контейнеры

---

React-Redux базируется на разделении компонентов на представление и контейнеры.

	Компоненты-представления	Компоненты-контейнеры
Назначение	Как выглядит (разметка, стили)	Как работает (загрузка данных, обновление состояния)
Знают о Redux	Нет	Да
Читают данные	Читают данные из props	Подписываются на Redux-состояние
Изменяют данные	Вызывают колбеки из props	Отправляют Redux-действия
Написаны	Руками	Обычно генерируются React Redux

# Создадим компоненты-представления

---

Компонент Students, Grades

# Создадим компонент-контейнер

---

Для начала нам нужно импортировать необходимые компоненты и функции: генераторы действий, функцию connect и компонент для взаимодействия

```
import { addGrade, deleteGrade } from "../model/actions";  
import { connect } from "react-redux";  
import Students from "../Students";
```



# Преобразователи

---

Функции `mapStateToProps` и `mapDispatchToProps` преобразуют в props состояние приложения и связывают действия Redux с props.

# Преобразователи

---

```
const mapStateToProps = (state)=>{
  return state;
}

const mapDispatchToProps = (dispatch)=>{
  return {
    onAddGrade: (index, grade)=> {
      dispatch(addGrade(index, grade))
    },
    onDeleteGrade: (indexOfStudent, indexOfGrade)=>{
      dispatch(deleteGrade(indexOfStudent, indexOfGrade))
    }
  }
}
```

# Функция connect

---

Функция connect создаёт компонент, связанный с другим компонентом, а также принимает в качестве параметра функции, преобразующие данные приложения в props

```
const StudentsTable = connect(mapStateToProps,  
mapDispatchToProps)(Students);
```

```
export default StudentsTable;
```

## 5. Привязка хранилища

---

Для того, чтобы создать хранилище, необходимо в функцию createStore библиотеки redux передать наш редюсер.

Для привязки хранилища используется компонент Provider из react-redux

```
import {Provider} from 'react-redux';  
import {createStore} from 'redux';  
import {grades} from '../model/reducers';  
const store=createStore(grades);  
ReactDOM.render(<Provider store={store}><App /></Provider>,  
document.getElementById('root'));
```

# Данные получены

---

Осталось только связать события с действиями.

Для этого к обработчикам нужно привязать соответствующие пропсы.

# Пример

---

```
<span key={index} style={{padding: "5px"}} onClick={() => {  
this.props.onDeleteGrade(i, index)}}>{grade}</span>
```

# Задание

---

Используя Redux создайте приложение, которое отслеживает работу компьютеров в сети.

Информация о компьютере: имя, IP адрес, включен или отключен.

Действия: изменить имя, назначить новый IP, включить либо выключить.

# Конец

## ПОСЛЕСЛОВИЕ

Давайте подведем итоги урока!  
Чему мы научились? Что мы использовали?  
К чему мы пришли?