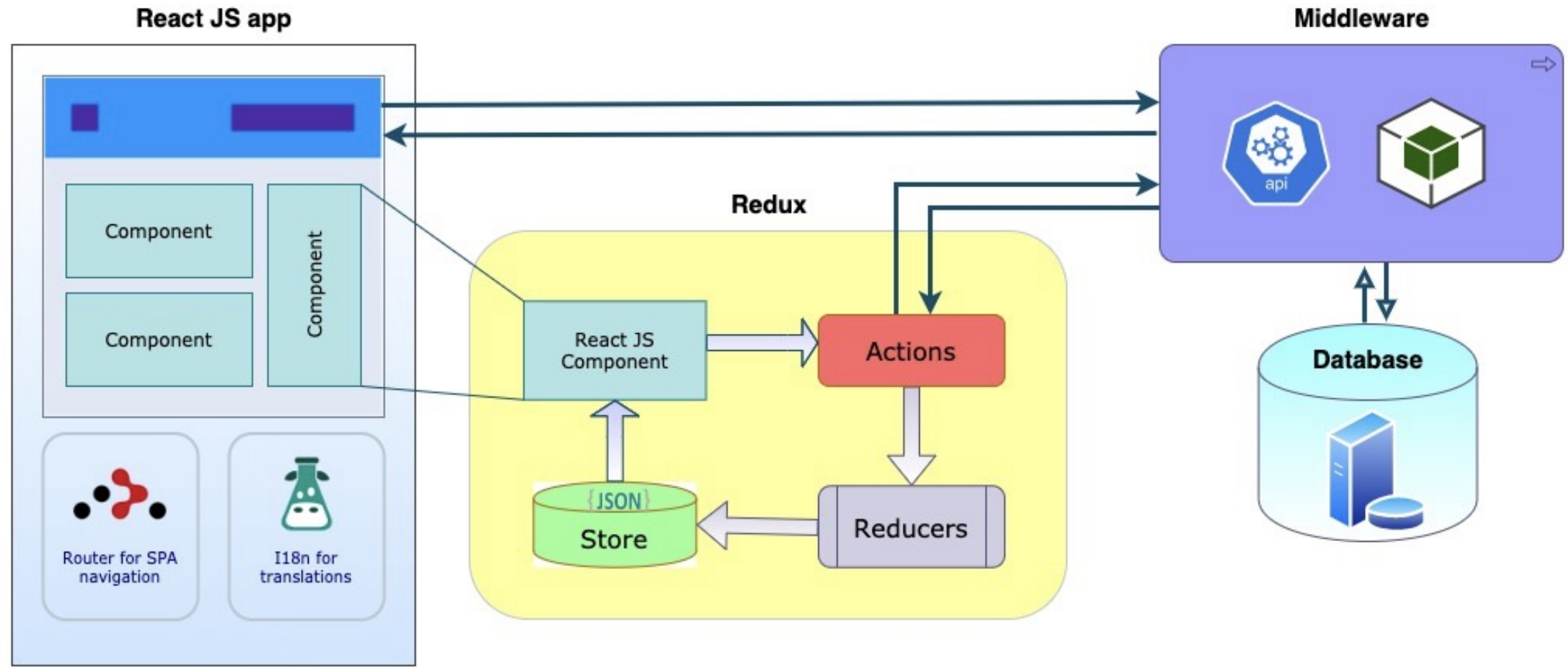
Лекция 9
Redux

Разработка интернет приложений

Канев Антон Игоревич

React


React JS app
Component
Component
Component
Router for SPA navigation
I18n for translations
Redux
React JS Component
Actions
{JSON}
Store
Reducers
Middleware
api
Database

Хуки

```jsx
import React, { useEffect, useState } from 'react'
import Axios from 'axios'

export default function Hello() {

    const [Name, setName] = useState("")

    useEffect(() => {
        Axios.get('/api/user/name')
        .then(response => {
            setName(response.data,name)
        })
    }, [])

    return (
        <div>
            My name is {Name}
        </div>
    )
}
```

```jsx
import React, { Component } from 'react'
import Axios from 'axios'

export default class Hello extends Component {

    constructor(props) {
        super(props);
        this.state = { name: ""};
    }

    componentDidMount() {
        Axios.get('/api/user/name')
        .then(response => {
            this.setState({ name: response.data.name })
        })
    }

    render() {
        return (
            <div>
                My name is {this.state.name}
            </div>
        )
    }
}
```

useEffect

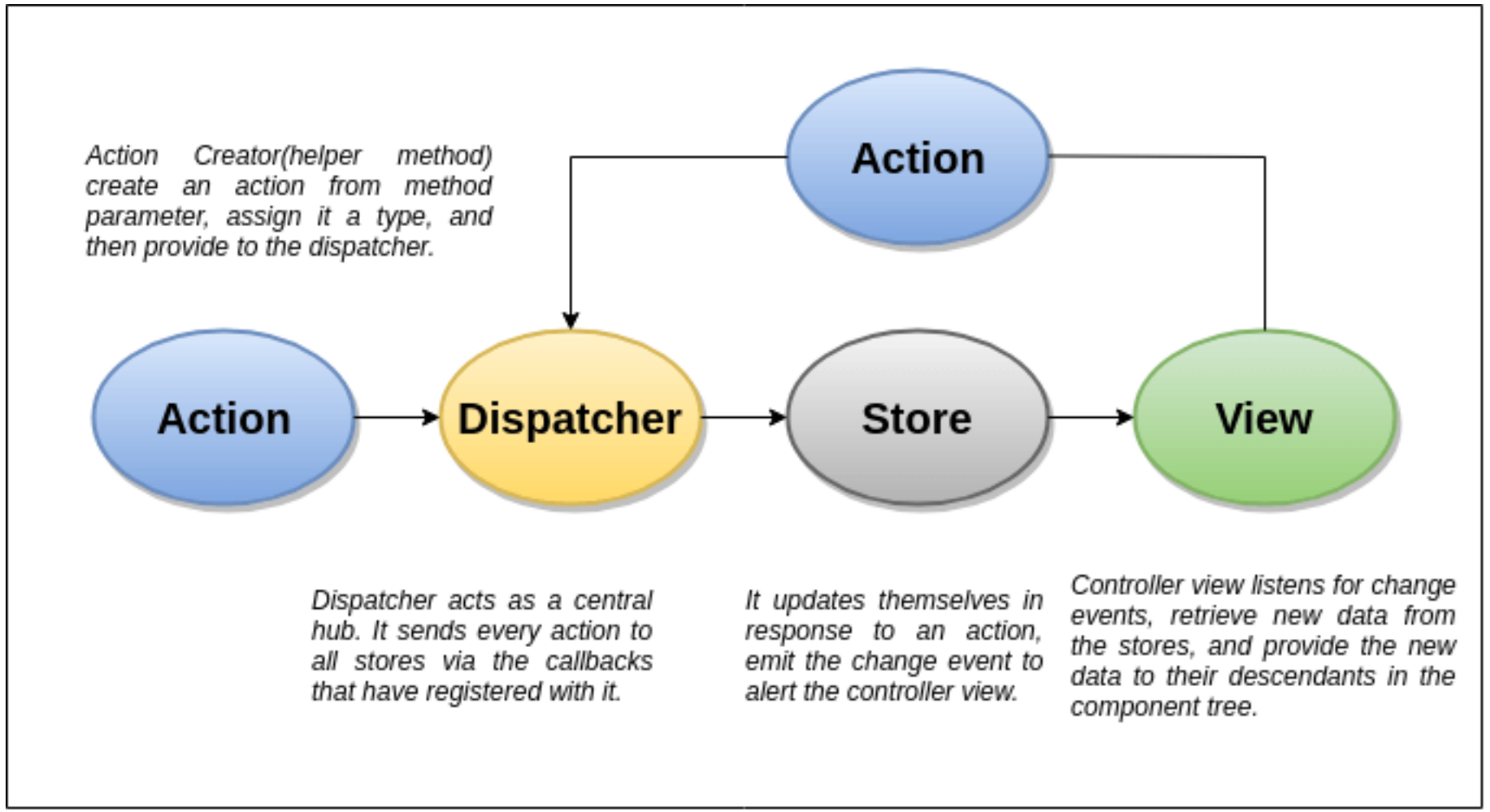
- componentDidMount()
- componentDidUpdate()
- componentWillUnmount()

```
useEffect( effect: ()=>{
    console.log('Этот код выполняется только на первом рендере компонента')
    // В данном примере можно наблюдать Spread syntax (Троеточие перед массивом)
    setNames( value: names=>[...names, 'Бедный студент'])

    return () => {
        console.log('Этот код выполняется, когда компонент будет размонтирован')
    }
}, deps: [])

useEffect( effect: ()=>{
    console.log('Этот код выполняется каждый раз, когда изменится состояние showNames ')
    setRandomName(names[Math.floor( x: Math.random()*names.length)])
}, deps: [showNames])
```

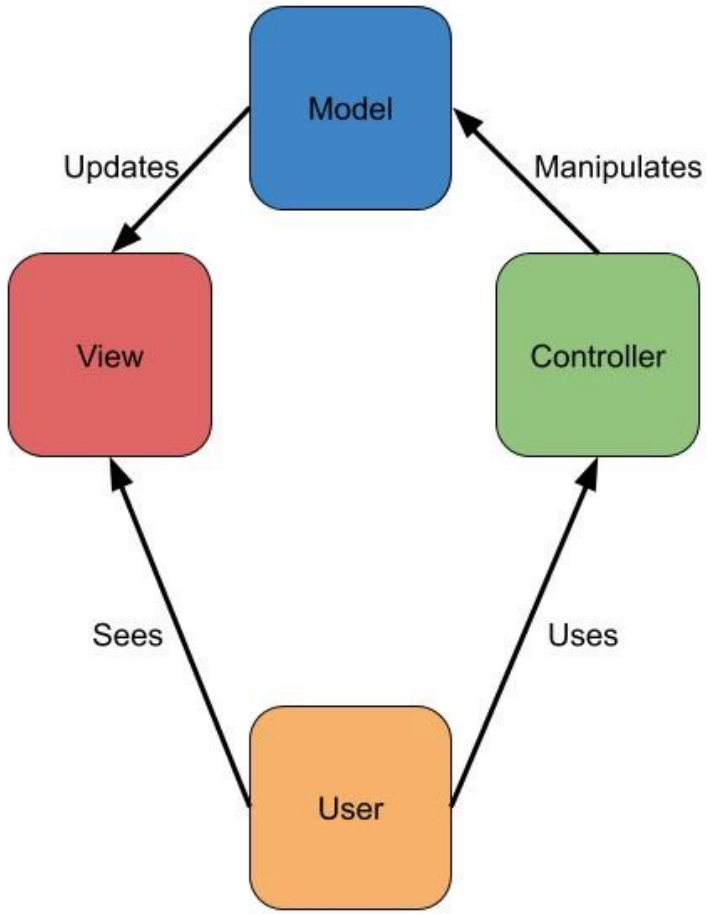
Другие хуки

- useContext: позволяет работать с контекстом — с механизмом, используемым для организации совместного доступа к данным без необходимости передачи свойств.
- useRef: позволяет напрямую обращаться к DOM в функциональных компонентах. Обратите внимание на то, что useRef, в отличие от setState, не вызывает повторный рендеринг компонента.
- useReducer: хранит текущее значение состояния. Его можно сравнить с Redux.
- useMemo: используется для возврата мемоизированного значения. Может применяться в случаях, когда нужно, чтобы функция возвратила бы кешированное значение.
- useCallback: применяется в ситуациях, когда дочерние элементы компонента подвергаются постоянному повторному рендерингу. Он возвращает мемоизированную версию коллбэка, которая меняется лишь тогда, когда меняется одна из зависимостей.

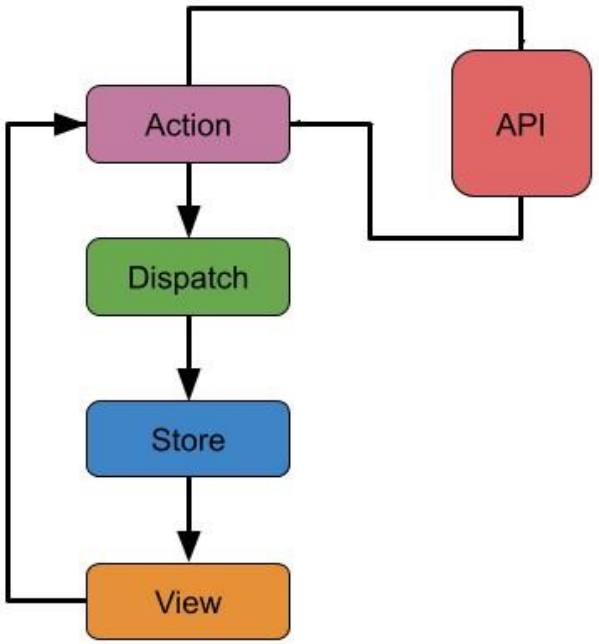
React MVC


Action Creator(helper method) create an action from method parameter, assign it a type, and then provide to the dispatcher.
Action
Action
Dispatcher
Store
View
Dispatcher acts as a central hub. It sends every action to all stores via the callbacks that have registered with it.
It updates themselves in response to an action, emit the change event to alert the controller view.
Controller view listens for change events, retrieve new data from the stores, and provide the new data to their descendants in the component tree.

MVC vs React

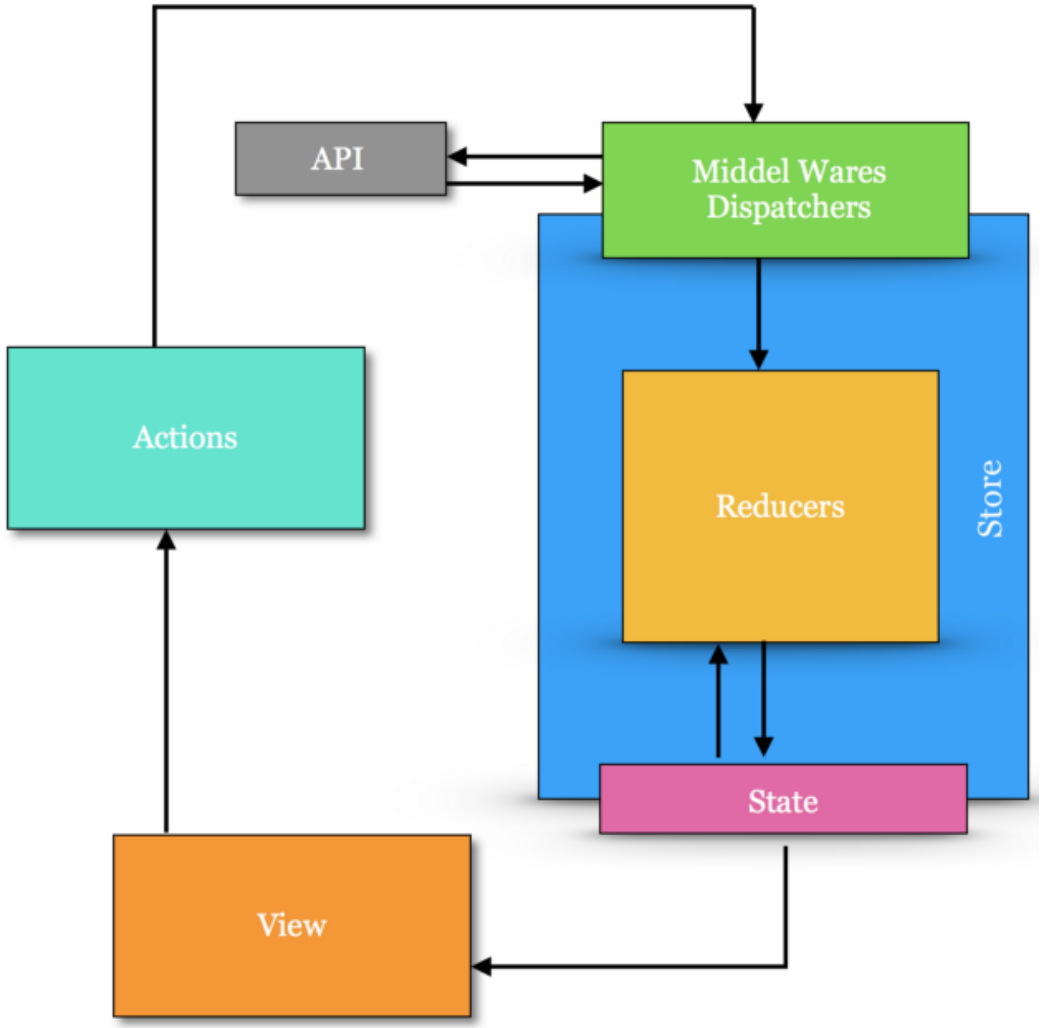


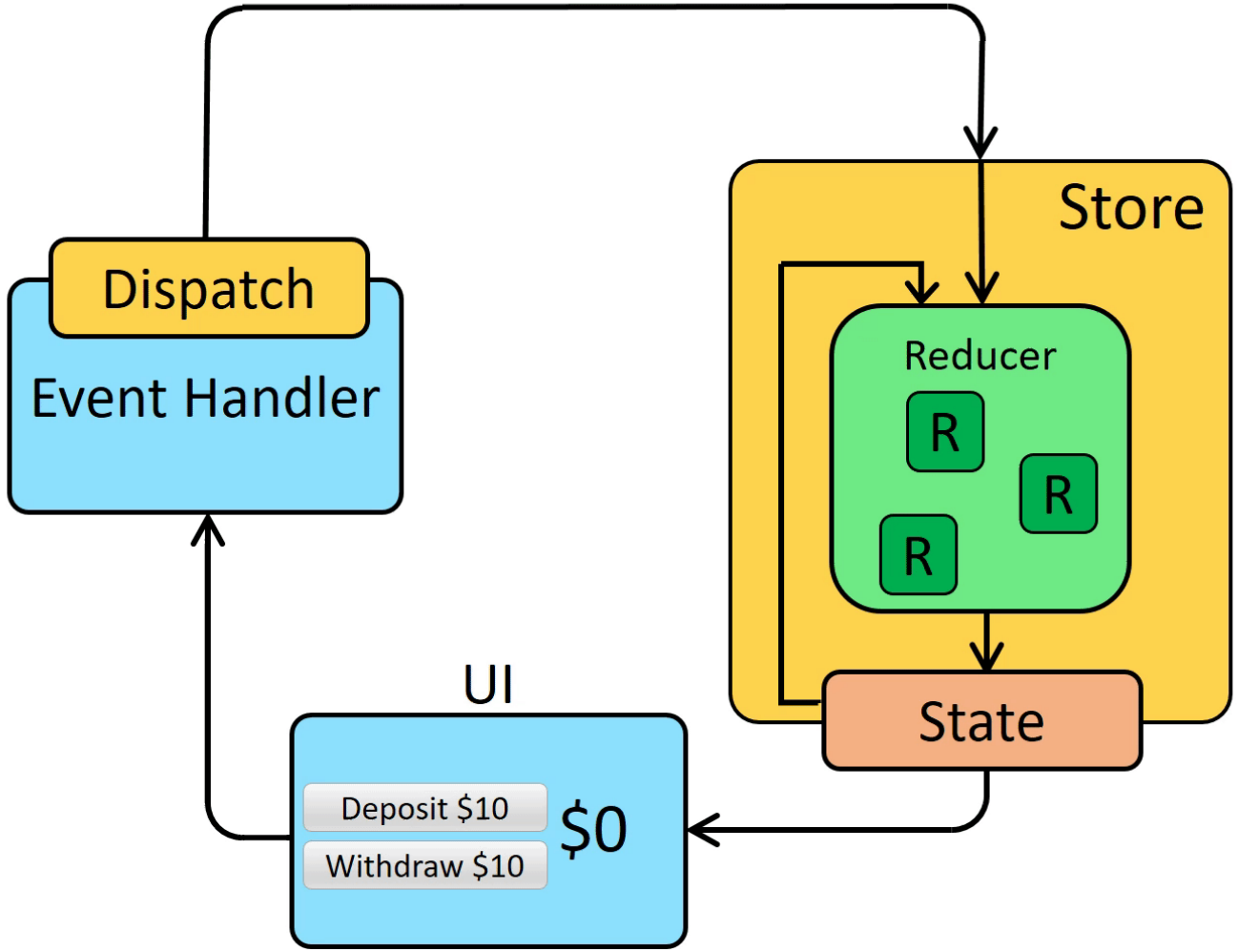
MVC base Model



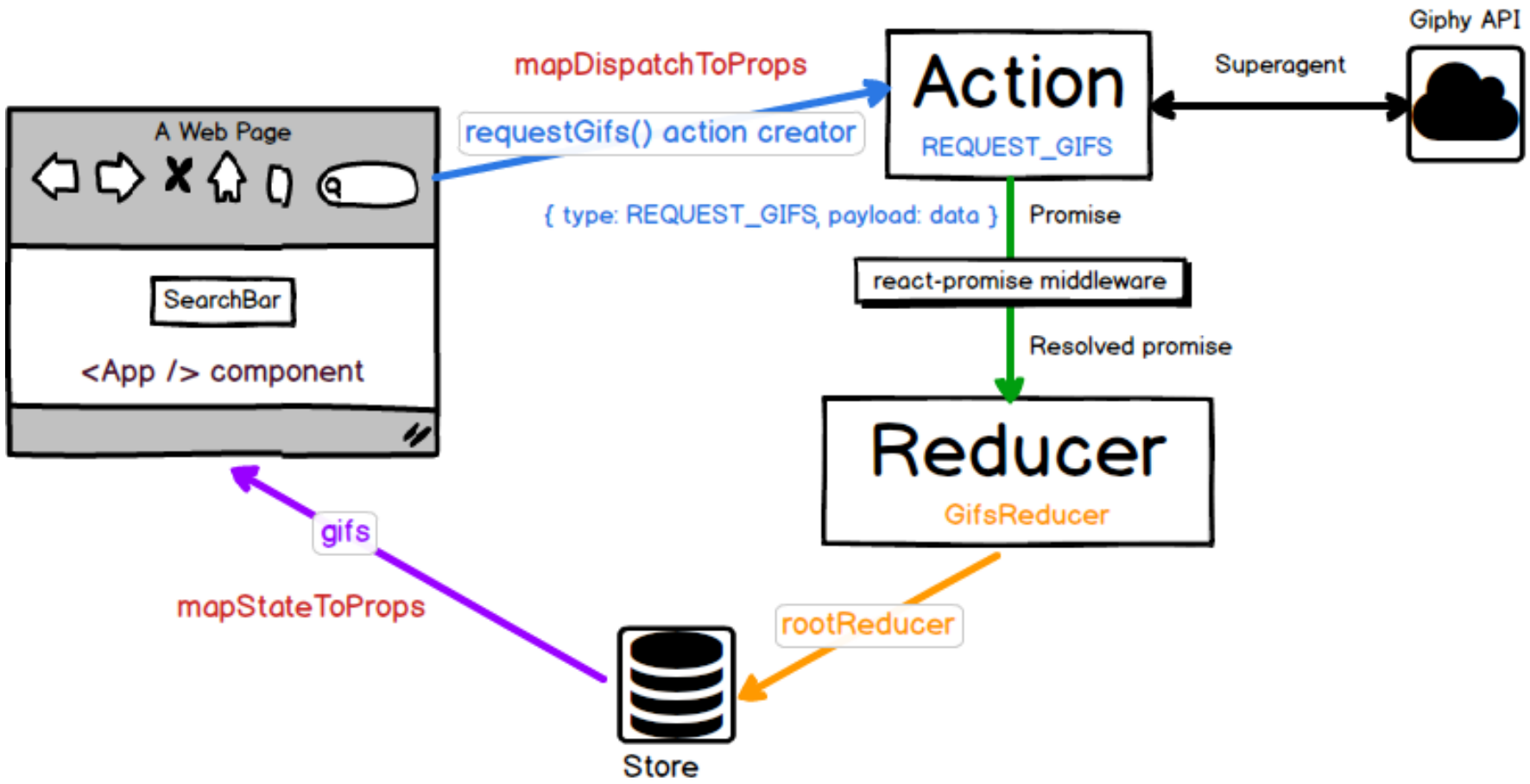
React's MVC model

React MVC


API
Middel Wares
Dispatchers
Actions
Reducers
Store
State
View


Dispatch
Event Handler
Store
Reducer
R
R
R
State
UI
Deposit $10
Withdraw $10
$0

Redux


Giphy API
mapDispatchToProps
Action
REQUEST_GIFS
Superagent
A Web Page
requestGifs() action creator
{ type: REQUEST_GIFS, payload: data }
Promise
SearchBar
react-promise middleware
<App /> component
Resolved promise
Reducer
GifsReducer
gifs
mapStateToProps
rootReducer
Store

Redux

```
.store
├── actionCreators
│   ├── action_1.js
│   └── action_2.js
├── actions
│   ├── action_1.js
│   └── action_2.js
├── reducers
│   ├── reducer_1.js
│   ├── reducer_2.js
│   └── rootReducer.js
├── initialState.js
└── store.js
```

- Используем папку /src/store в которой хранится всё то, что связано с Redux и хранилищем приложения

Store.js

- Глобальное хранилище приложения создаётся в отдельном файле, который как правило называется store.js

```
// Код файла store.js
import { createStore } from 'redux';

const store = createStore(reducer);

export default store;
```

reducer

- reducer — чистая функция которая будет отвечать за обновление состояния.
- Здесь реализовывается логика в соответствие с которой будет происходить обновление полей store

```
function reducer(state, action) {
    switch(action.type) {
        case ACTION_1: return { value: action.value_1 };
        case ACTION_2: return { value: action.value_2 };

        default: return state;
    }
}
```

- Редьюсер как бы спрашивает: что произошло?
- actio.type равен «ACTION_1», значит произошло событие номер 1. Дальше его нужно как то обработать и обновить состояние.
- То, что вернёт редьюсер и будет новым состоянием.

dispatch

- Что бы обновить store необходимо вызвать метод dispatch().
- Эта функция вызовет функцию reducer который обработает событие и обновит соответствующие поля хранилища
- Он вызывается у объекта store который вы создаёте в store.js

- ACTION_1 это константа события о которой речь пойдет дальше

```
store.dispatch({ type: ACTION_1, value_1: "Some text" });
```

actionCreator

- На самом деле передавать объект события напрямую в dispatch() является признаком плохого тона.
- Для этого нужно использовать функцию под названием actionCreator.
- Вызов этой функции нужно передавать как аргумент в dispatch а в actionCreator передавать необходимое значение (value).

```
function action_1(value) {
    return {
        type: ACTION_1,
        value_1: value
    };
}

export default action_1;
```

- Таким образом вызов dispatch должен выглядеть так:

```
store.dispatch(action_1("Some value"));
```

Actions

- Actions это константы, описывающие событие.
- Обычно это просто строка с названием описывающее событие.

- В проекте вам стоит называть константы в соответствии с событием, которое она описывает: onClick, createUserSesion, deleteItem, addItem и т.д.

```
const ACTION_1 = "ACTION_1";

export default ACTION_1;
```

getState

- С помощью dispatch() обновили, а как теперь посмотреть новое значение store?
- Есть метод getState().
- Он также, как и метод dispatch вызывается на экземпляре объекта store

```
store.getState()
```

- К примеру что бы посмотреть значение поля value_1 необходимо будет вызвать

```
store.getState().value_1
```

subscribe

- А как же узнать, когда состояние обновилось?
- Для этого есть метод subscribe(). Он также вызывается на экземпляре store.
- Данный метод принимает функцию, которая будет вызывается каждый раз после обновления store

```
store.subscribe(() => console.info(store.getState()))
```

combineReducers

- Если логика обновления компонентов довольно сложна и\или необходимо обрабатывать большое количество различных типов событий, то корневой reducer может стать слишком громоздким
- Название редьюсера (value_1) показывает какое свойство он будет обновлять в store

```
function value_1(state, action) {
    switch(action.type) {
        case ACTION_1: return action.value_1;

        default: return state;
    }
}

export default value_1;
```

```
case ACTION_1: return { value_1: action.value_1 };
```

```
case ACTION_1: return action.value_1;
```

initialState

- initialState — объект, представляющий начальное состояние хранилища.
- Он является вторым не обязательным аргументом метода createStore()
- Также если вы не передаёте объект initialState в createStore вы можете вернуть его из редьюсера.
- В обоих случаях будет инициализировано начальное состояние для store

```
const initialState = {
    date_1: "value...",
    date_2: "value..."
};

export default initialState;
```

React-redux. Provider

- Для использование store в компоненте вам необходимо передавать его в пропсы

```
ReactDOM.render(<Main store={store} />, document.getElementById('root'));
```

- И после использовать в компоненте: this.props.state. Для этого react-redux предостовляет метод Provider

```
ReactDOM.render(
    <Provider store={store}>
        <Main />
    </Provider>,
document.getElementById('root'));
```

mapStateToProps

- Этот метод вызывается всякий раз, когда происходит обновление store и именно он передаёт необходимые свойства из store в компонент.
- К примеру компонент, должен реагировать и обновлять UI каждый раз, когда поле номер один (value_1) обновилось.

```
function (state) {
    return {
        value_1: state.value_1
    };
}
```

```
function mapStateToProps(component) {
    switch(component) {
        case "Component_1": {
            return function (state) {
                return {
                    value_1: state.value_1
                };
            }
        }
        case "Component_2": {
            return function(state) {
                return {
                    value_2: state.value_2
                };
            }
        }
        default: return undefined;
    }
}

export default mapStateToProps;
```

mapDispatchToProps

- Эта функция передаёт в компонент методы для обновления необходимого поля store.
- Что бы не вызывать dispatch напрямую из компонента

```
function (dispatch) {
    return {
        changeValue_1: bindActionCreators(action_1, dispatch)
    };
};
```

connect

- Она связывает mapStateToProps и mapDispatchToProps с компонентом и передает необходимые поля и методы в него
- Возвращает она новый компонент-обёртку для вашего компонента.

```
const COMPONENT_1_W = connect(mapStateToProps("Component_1"),
                              mapDispatchToProps("Component_1"))(Component_1);
```

Пример без Hooks

```
import React from 'react';
import { connect } from 'react-redux';
import { incrementCount } from './store/counter/actions';

export function AwesomeReduxComponent(props) {
    const { count, incrementCount } = props;

    return (
        <div>
            <p>Count: {count}</p>
            <button onClick={incrementCount}>Add +1</button>
        </div>
    );
}

const mapStateToProps = state => ({ count: state.counter.count });
const mapDispatchToProps = { incrementCount };

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

Hooks

- Выглядит более просто, чем с использованием функции connect
- *props* компонента не смешиваются со свойствами из редакса

```
import React from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { incrementCount } from './store/counter/actions';

export const AwesomeReduxComponent = () => {
    const count = useSelector(state => state.counter.count);
    const dispatch = useDispatch();

    return (
        <div>
            <p>Count: {count}</p>
            <button onClick={() => dispatch(incrementCount())}>Add +1</butt
        </div>
    );
};
```