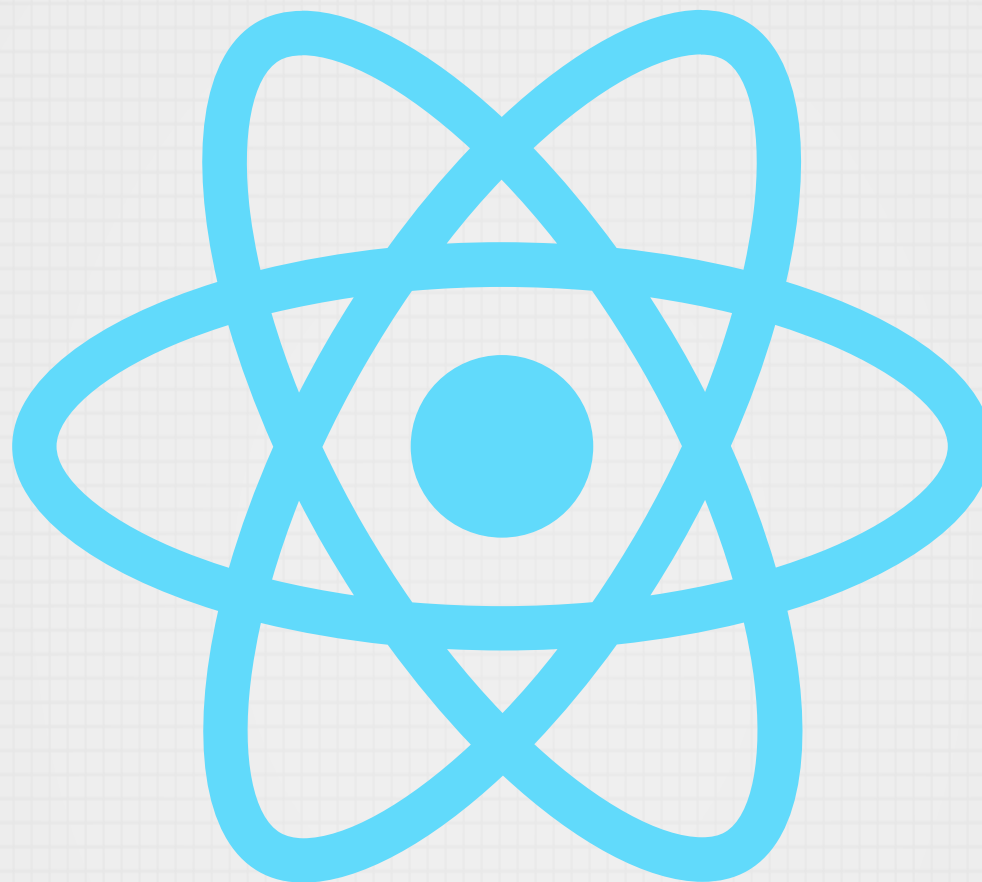


# Napredne Web Tehnologije

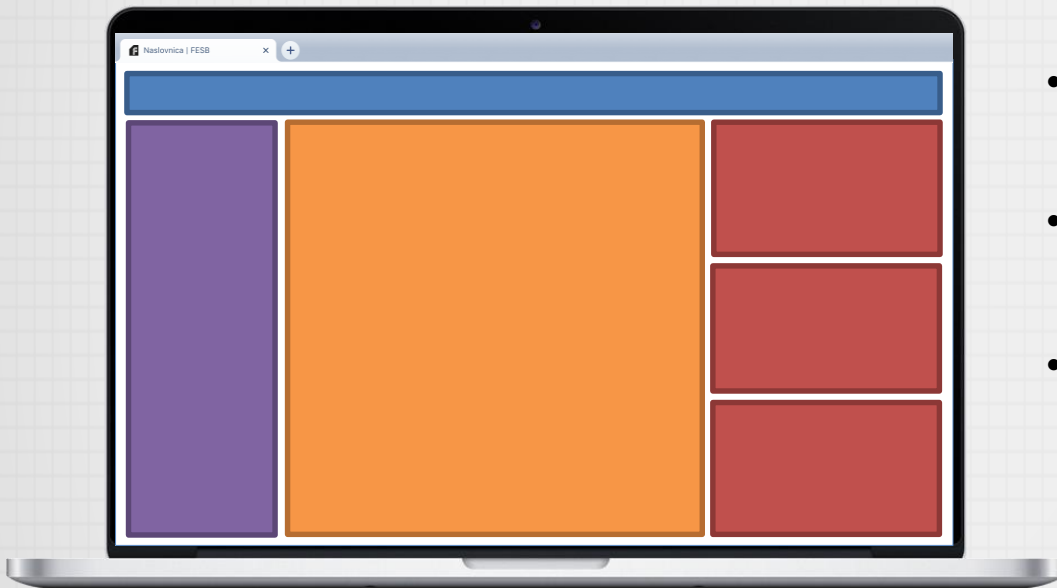
React, Redux





# React

# ŠTO JE REACT?



- Stranicu je moguće rascjepkati na **komponente**!
- Ove komponente je moguće izgraditi korištenjem **JavaScripta**.
- Komponente je moguće iskoristiti više puta unutar iste stranice (**reusable**)!

React je JavaScript biblioteka za izradu korisničkih sučelja!

# ŠTO JE REACT?

- React je “JavaScript biblioteka za kreiranje korisničkih sučelja”
  - Umjesto stvaranja HTML template-a, stvaraju se komponente.
  - React se koristi za izgradnju **View**-a (V u MVC)
- React prati „**component oriented development**” paradigmu
  - Osnovna ideja je rascjepkati UI u komponente.
  - Za svaku komponentu implementirati *reusable* react komponentu.
  - Kombinirati i ugnježdživati komponente kako bi se stvorio cjelokupni UI.
- Prednosti:
  - Ponovno iskoristivi kod (komponente).
  - Čitkiji kod – komponente se nalaze u svojim klasama.
  - Jednostavniji testovi – svaku komponentu je moguće testirati odvojeno.

# React Virtual DOM

- Prilikom svake promjene React ponovno *renderira* stranicu
  - Kreira novi virtual DOM (u memoriji)
  - *diff* - uspoređuje ga sa prethodnom verzijom
  - Računa minimalni set promjena koje je potrebno napraviti nad pravim DOM-om.
- Prednosti virtualnog DOM-a:
  - Izbjegava skupe DOM operacije
  - Minimizira pristup DOM-u
  - Dozvoljava pisanje čistog koda koji nije ovisan o karakteristikama DOM-a.

# React komponente

## props

- Skraćeno od *properties*
- Svaka komponenta posjeduje **props** objekt
- React implementira **jednosmjerni tok podataka** (*uni-directional data flow*) korištenjem props objekta
- Služi za prijenos podataka od parent komponente, prema child komponenti
- Poslana svojstva su **immutable**

u parent komponenti:

```
<ChildComponent x={10} y={20} />
```

u child komponenti:

```
...  
let combined = this.props.x + this.props.y;  
...
```

# React **komponente**

## state

- Svaka komponenta može posjedovati i **state** object
  - Ukoliko ne posjeduje, riječ je o **stateless** komponenti
- Za spremanje informacija koje se može mijenjati tijekom vremena
- Na ovaj način komponenta vodi računa o informacijama između dva render-a
- State se (za razliku od props) **može mijenjati**

Manipulativne funkcije:

Direktno **this.state** (constructor)

Mijenjanje preko **this.setState({variable: value});**

```
class ProductList extends Component {
  constructor(props) {
    super(props);
    this.state = {
      title: "unknown",
      products: [],
    }
  }

  componentDidMount(){
    const fetchedProducts = this.fetchProducts();
    this.setState({
      title: "List of products",
      products: fetchedProducts,
    })
  }

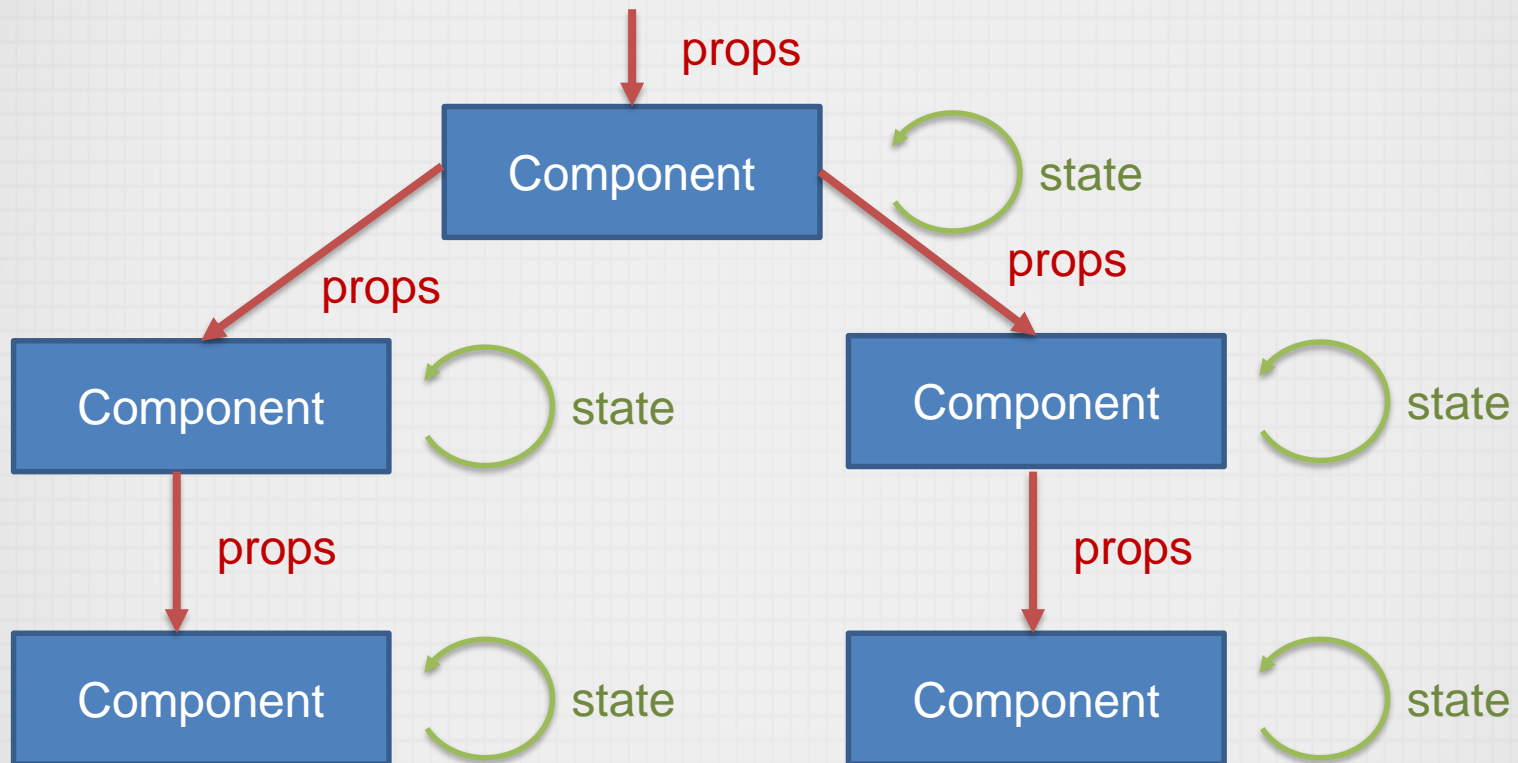
  fetchProducts(){
    return ["product1", "product2"]
  }

  render() {
    return ( <div>{this.state.products}</div> );
  }
}
export default ProductList;
```

u konstruktoru



# React – data flow



# state & props var

	props	state
Može primiti inicijalnu vrijednost od parent komponente?	Da	Da
Parent komponenta ju može promijeniti?	Da	Ne
Unutar komponente se mogu postaviti defaultne vrijednosti?	Da	Da
Može se promijeniti unutar komponente?	Ne	Da

# JSX – JavaScript eXtension

- JSX je objektno orijentiran programski jezik koji se kompajlira u JavaScript.
- Dozvoljava implementaciju UI elemenata unutar JS koda
  - Sintaksa JSX je poput XML-a
  - Dozvoljava ubacivanje HTML sintakse unutar JS funkcija.
- JSX prebacuje iz sintakse slične XML-u u nativni JavaScript
  - XML elementi se prebacuju u pozive funkcija
  - XML atributi se prebacuju u objekte

# React: JSX vs JS Syntax

```
class Hello extends React.Component {  
  render() {  
    return <div>Hello {this.props.toWhat}</div>;  
  }  
}
```

```
ReactDOM.render(  
  <Hello toWhat="World" />,  
  document.getElementById('root')  
);
```

```
class Hello extends React.Component {  
  render() {  
    return React.createElement('div', null, `Hello ${this.props.toWhat}`);  
  }  
}
```

React.createElement(component, props, ...children)

```
ReactDOM.render(  
  React.createElement(Hello, {toWhat: 'World'}, null),  
  document.getElementById('root')  
);
```

# React: JSX

```
let element =
  <h1>Hello, <b>React</b> from <i>NWT</i></h1>;
console.log(element);
```

App.js:10

```
▼ {$$typeof: Symbol(react.element), type: "h1", key: null, ref: null, props: {...}, ...} ⓘ
  $$typeof: Symbol(react.element)
  key: null
  ▼ props:
    ▼ children: Array(4)
      0: "Hello, "
      ► 1: {$$typeof: Symbol(react.element), type: ...
      2: " from "
      ► 3: {$$typeof: Symbol(react.element), type: ...
        length: 4
        ► __proto__: Array(0)
      ► __proto__: Object
    ref: null
    type: "h1"
    _owner: null
    ► _store: {validated: false}
    ► _self: App {props: {...}, context: {...}, refs: {...}...
    ► _source: {fileName: "/Users/mbugaric/Desktop/my...
    ► __proto__: Object
```

```
▼ 1:
  $$typeof: Symbol(react.element)
  key: null
  ▼ props:
    children: "React"
    ► __proto__: Object
    ref: null
    type: "b"
    _owner: null
    ► _store: {validated: true}
    ► _self: App {props: {...}, context: {...}, refs...
    ► _source: {fileName: "/Users/mbugaric/Desktop...
    ► __proto__: Object
  2: " from "
```

# React: gdje i kako početi

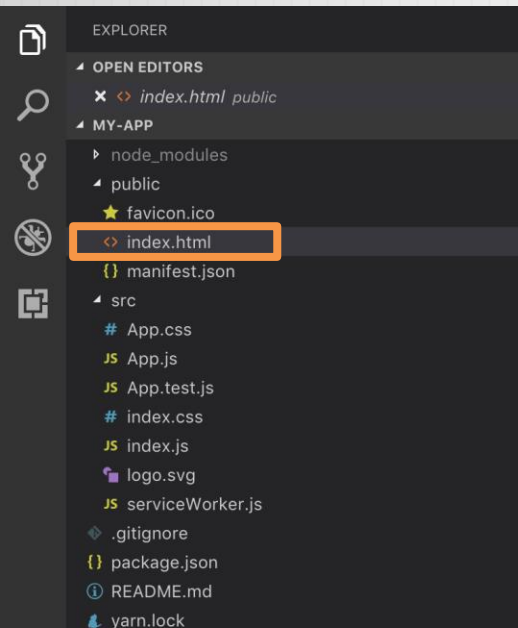
## create-react-app

- Jednostavan *environment* za:
  - učenje React-a,
  - kreiranje nove SPA aplikacije u React-u.

```
npm install -g create-react-app
```

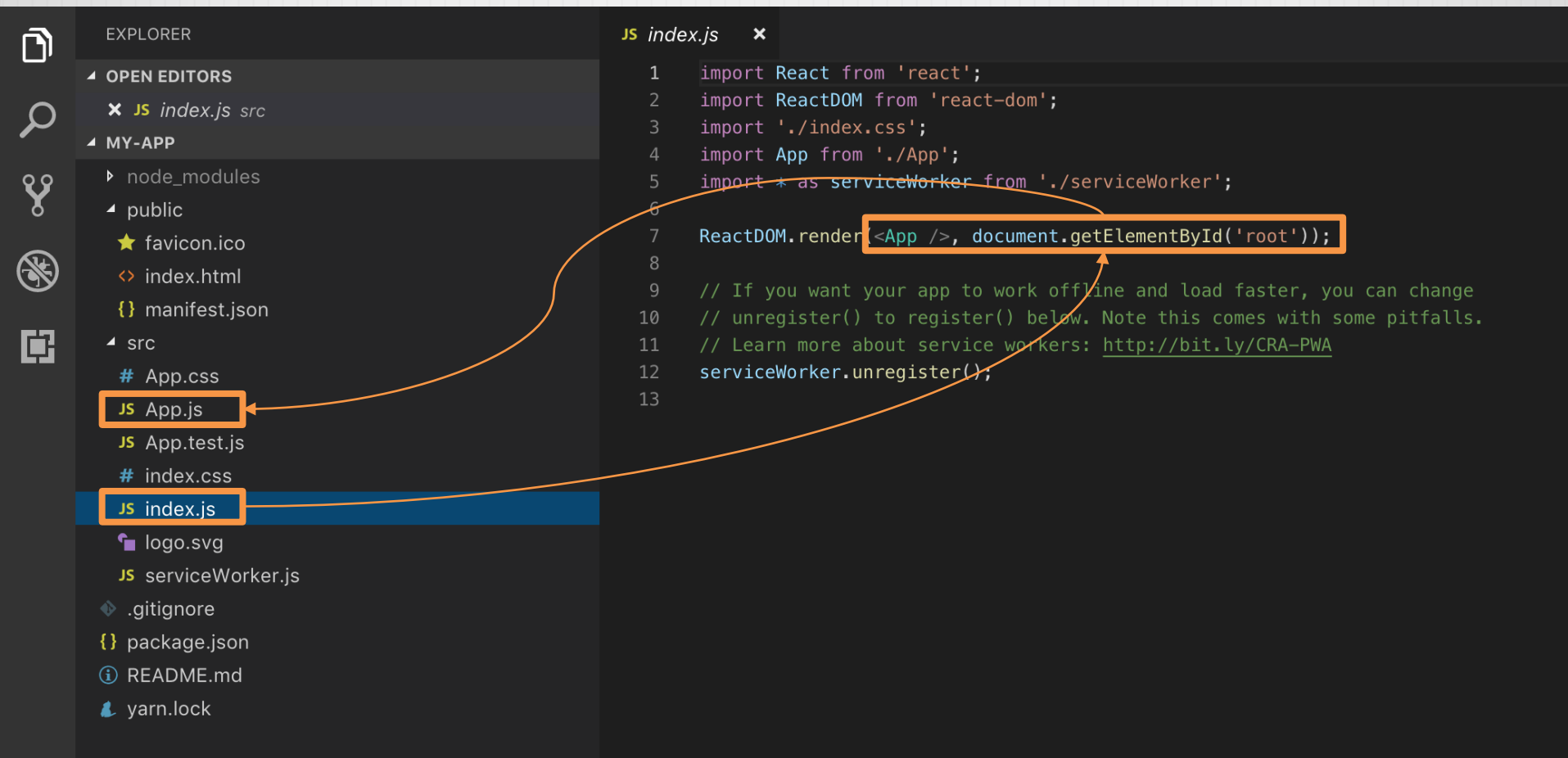
Package runner tool (npm 5.2+)

```
npx create-react-app my-app  
cd my-app  
npm start
```



```
<> index.html x
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8" />
5     <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico" />
6     <meta
7       name="viewport"
8       content="width=device-width, initial-scale=1, shrink-to-fit=no"
9     />
10    <meta name="theme-color" content="#000000" />
11    <!--
12      manifest.json provides metadata used when your web app is added to the
13      homescreen on Android. See https://developers.google.com/web/fundamentals/web-app-manifest/
14    -->
15    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
16    <!--
17      Notice the use of %PUBLIC_URL% in the tags above.
18      It will be replaced with the URL of the `public` folder during the build.
19      Only files inside the `public` folder can be referenced from the HTML.
20
21      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
22      work correctly both with client-side routing and a non-root public URL.
23      Learn how to configure a non-root public URL by running `npm run build`.
24    -->
25    <title>React App</title>
26  </head>
27  <body>
28    <noscript>You need to enable JavaScript to run this app.</noscript>
29    <div id="root"></div>
30    <!--
31      This HTML file is a template.
32      If you open it directly in the browser, you will see an empty page.
33
34      You can add webfonts, meta tags, or analytics to this file.
35      The build step will place the bundled scripts into the <body> tag.
36
37      To begin the development, run `npm start` or `yarn start`.
38      To create a production bundle, use `npm run build` or `yarn build`.
39    -->
40  </body>
41 </html>
42
```

# React: gdje i kako početi





# React: gdje i kako početi



## EXPLORER

### OPEN EDITORS

✕ JS App.js src

### MY-APP

▸ node\_modules

▸ public

★ favicon.ico

<> index.html

{ } manifest.json

▸ src

# App.css

JS App.js

JS App.test.js

# index.css

JS index.js

🖼 logo.svg

JS serviceWorker.js

📁 .gitignore

{ } package.json

📖 README.md

🔒 yarn.lock

JS App.js ✕

```
1 import React, { Component } from 'react';
2 import logo from './logo.svg';
3 import './App.css';
4
5 class App extends Component {
6   render() {
7     return (
8       <div className="App">
9         <header className="App-header">
10           <img src={logo} className="App-logo" alt="logo" />
11           <p>
12             Edit <code>src/App.js</code> and save to reload.
13           </p>
14           <a
15             className="App-link"
16             href="https://reactjs.org"
17             target="_blank"
18             rel="noopener noreferrer"
19           >
20             Learn React
21           </a>
22         </header>
23       </div>
24     );
25   }
26 }
27
28 export default App;
29
```

# React: `package.json` (*create-react-app*)

```
{
  "name": "my-app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "react": "^16.6.3",
    "react-dom": "^16.6.3",
    "react-scripts": "2.1.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": [
    ">0.2%",
    "not dead",
    "not ie <= 11",
    "not op_mini all"
  ]
}
```

# Kako kreirati komponente?

- Svaka React komponenta je definirana kao JavaScript **klasa**
  - Ova klasa nasljeđuje **React.Component** klasu
- Svaka komponenta je implementirana i vraćena unutar **render()** metode.

```
class Main extends React.Component {  
  render(){  
    return(  
      <h1>Hello world!</h1>  
    )  
  }  
}
```

# Kako kreirati komponente?

▸ node\_modules

▸ public

▸ src

▸ components

▸ person

🔗 person.jsx

# App.css

JS App.js

JS App.test.js

# index.css

JS index.js

🖼 logo.svg

JS serviceWorker.js

📄 .gitignore

{ } package.json

📖 README.md

👤 yarn.lock

```
import React from 'react';
```

```
class Person extends React.Component{
```

```
  render(){
```

```
    return(
```

```
      <div>
```

```
        <h1>{this.props.name}</h1>
```

```
        <p>Tel. {this.props.phone}</p>
```

```
      </div>
```

```
    )
```

```
  }
```

```
}
```

```
export default Person;
```

# Kako kreirati komponente?

▸ node\_modules

▸ public

▸ src

▸ components

▸ person

🔗 person.jsx

# App.css

JS App.js

JS App.test.js

# index.css

JS index.js

🖼 logo.svg

JS serviceWorker.js

📄 .gitignore

{ } package.json

📖 README.md

👤 yarn.lock

```
import React, { Component } from 'react';  
import './App.css';
```

```
import Person from './components/person/person.jsx';
```

```
class App extends Component {  
  render() {  
    return (  
      <div className="App">  
        <Person name="Ante Antić" phone="+385 91 234567"/>  
      </div>  
    );  
  }  
}  
  
export default App;
```

# React primjer

**Ante Antić**

Tel. +385 91 234567

```
import React, { Component } from 'react';
import './App.css';

import Person
  from './components/person/person.jsx';

class App extends Component {
  render() {
    return (
      <div className="App">
        <Person name="Ante Antić"
          phone="+385 91 234567"/>
      </div>
    );
  }
}

export default App;
```

```
import React from 'react';

class Person extends React.Component{
  render(){
    return(
      <div>
        <h1>{this.props.name}</h1>
        <p>Tel. {this.props.phone}</p>
      </div>
    )
  }
}

export default Person;
```

# React: capitalized components

```
import React, { Component } from 'react';
import './App.css';
import Items from './components/items/items';
import products from './components/products/products';

class App extends Component {
  render() {
    return (
      <div className="container">
        <Items />
        <products />
      </div>
    );
  }
}

export default App;
```

Zašto je ovo plave boje? Zašto ne prikazuje products?

# React: capitalized components

- Sve komponente koje započinju malim slovom React tretira kao built-in komponente
  - poput `<div>` ili `<span>`
- Sve komponente koje započinju velikim slovom React tretira kao user-defined komponente
  - i one se kompajliraju u `React.createElement(...)`



# React: razlike u atributima

## className

- obzirom da je `class` ključna riječ u JavaScriptu, u JSX se koristi `className`

```
class App extends Component {  
  render() {  
    return (  
      <div className="container">  
        <ProductList />  
      </div>  
    );  
  }  
}
```

```
▼ <div id="root">  
  ▶ <div class="container">...</div>  
  </div>
```

# React: razlike u atributima

## checked

- `<input>` komponente poput checkbox ili radio podržavaju ovaj atribut
- korisno za kreiranje „controlled” komponenti
- `defaultChecked` je „uncontrolled” alternativa koja postavlja check samo prilikom mountanja komponente

# React: razlike u atributima

```
class CheckboxExample extends Component {
  constructor(props) {
    super(props);
    this.state = {
      isChecked: false
    }
  }

  toggleCheck(){
    this.setState({
      isChecked: !this.state.isChecked
    })
  }

  render() {
    return (
      <div>
        <input checked={this.state.isChecked}
          type="checkbox" onChange={()=>this.toggleCheck()} />
        Checkbox checked: {this.state.isChecked.toString()}
      </div>
    );
  }
}
```

☐ Checkbox checked: false

☒ Checkbox checked: true

controlled attribute

# React: razlike u atributima

## dangerouslySetInnerHTML

- React zamjena za innerHTML
- Kada želite ubaciti neki HTML direktno u komponentu (ne JSX)
- Namjerno nazvana dangerously...
  - cross-site scripting napadi

```
function createMarkup() {  
  return {__html: 'First &middot; Second'};  
}  
function MyComponent() {  
  return <div dangerouslySetInnerHTML={createMarkup()} />;  
}
```

# React: razlike u atributima

## htmlFor

- `for` je ključna riječ u JavaScriptu

## selected

- može se koristiti za `<option>` komponente

# React: komunikacija child → parent

30

```
...  
clickedProduct(id){console.log(id)};  
render() {  
  return (  
    <div>  
    {  
      this.state.products.map(product =>  
        <Product  
          key={uniqueId}  
          id={uniqueId}  
          title={product}  
          clickedProduct={this.clickedProduct}  
        />  
      )  
    }  
    </div> );  
  }  
}
```

parent

```
render() {  
  return (  
    <div onClick={() => this.props.clickedProduct(this.props.id)}>  
      {this.props.title}  
    </div> );  
  }  
}
```

child

# React: komunikacija child->parent

31

```
...  
clickedProduct(id){console.log(id)};  
render() {  
  return (  
    <div>  
    {  
      this.state.products.map(product =>  
        <Product  
          key={uniqueId}  
          id={uniqueId}  
          title={product}  
          clickedProduct={this.clickedProduct}  
        />  
      )  
    }  
    </div> );  
  }  
}
```

parent

A što je key?!

```
render() {  
  return (  
    <div onClick={()=>this.props.clickedProduct(this.props.id)}>  
      {this.props.title}  
    </div> );  
  }  
}
```

child

# React: keys

- Korisno prilikom rada sa dinamički kreiranim komponentama
  - posebno ukoliko liste mijenjaju sami korisnici
- Postavljanje `key` vrijednosti omogućuje React-u da jedinstveno identificira komponentu nakon bilo kakve promjene
- Često se zanemaruju `key`-s, što nije dobro!
- Može imati značajnog utjecaja na brzinu rada aplikacije



# React: keys

✖ ▶ Warning: Encountered two children with the same key, `1`. Keys index.js:1446 should be unique so that components maintain their identity across updates. Non-unique keys may cause children to be duplicated and/or omitted – the behavior is unsupported and could change in a future version.

- in div (at productList.js:26)
- in ProductList (at App.js:11)
- in div (at App.js:10)
- in App (at src/index.js:7)

Ne zanemarivati ova upozorenja!

# React: keys

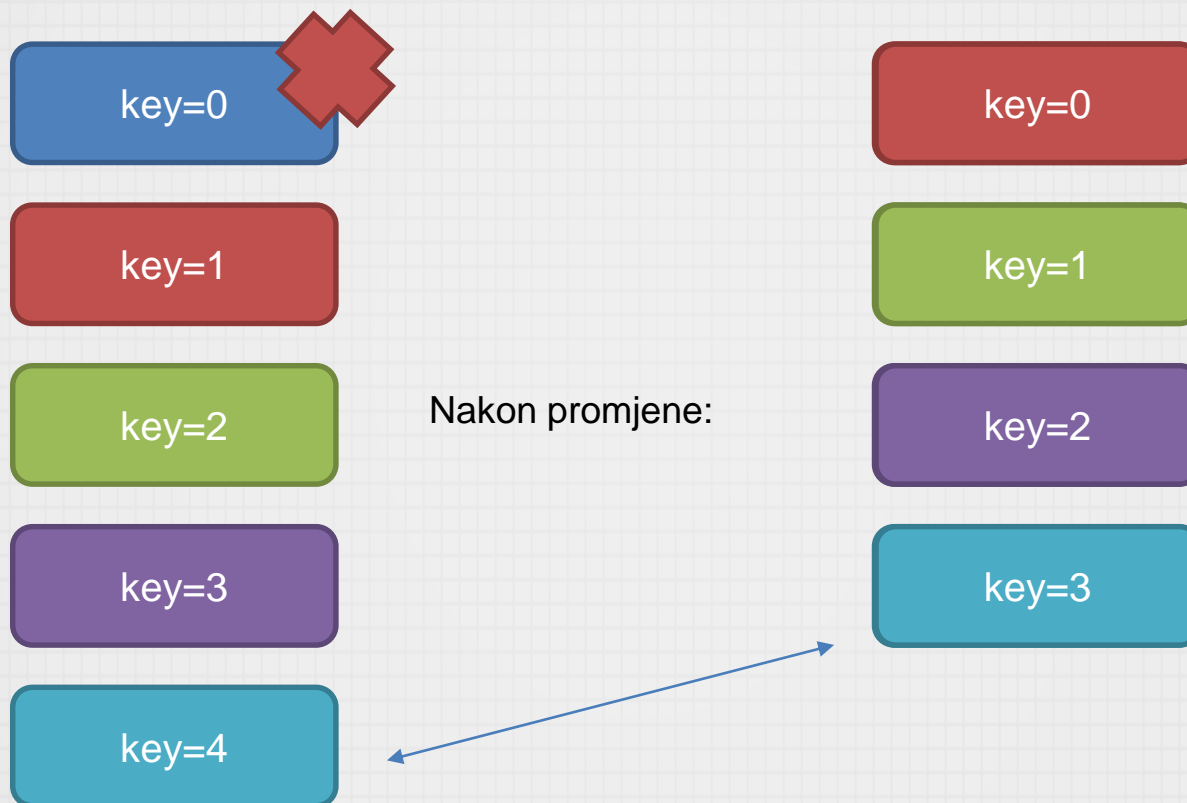
```
fetchProducts(){
  return [
    "product1", "product2", "product3",
    "product4", "product5", "product5"
  ]
}

render() {
  return (
    <div>
      {this.state.products.map((product,i) =>
        <Product
          key={i}
          title={product}
        />
      )}
    </div>
  );
}
```

Najčešći način postavljanja key-a, ali loš!

# React: keys

Ako je key vrijednost postavimo u index:



React misli da su ovo različiti elementi

# React **komponente** (React pre 16.3)

## Initialization

Setup props and state

## Mounting

componentWillMount 

render

componentDidMount

## Updating

props

componentWillRecieveProps 

shouldComponentUpdate

true

false 

componentWillUpdate 

render

componentDidUpdate

states

shouldComponentUpdate

true

false 

componentWillUpdate 

render

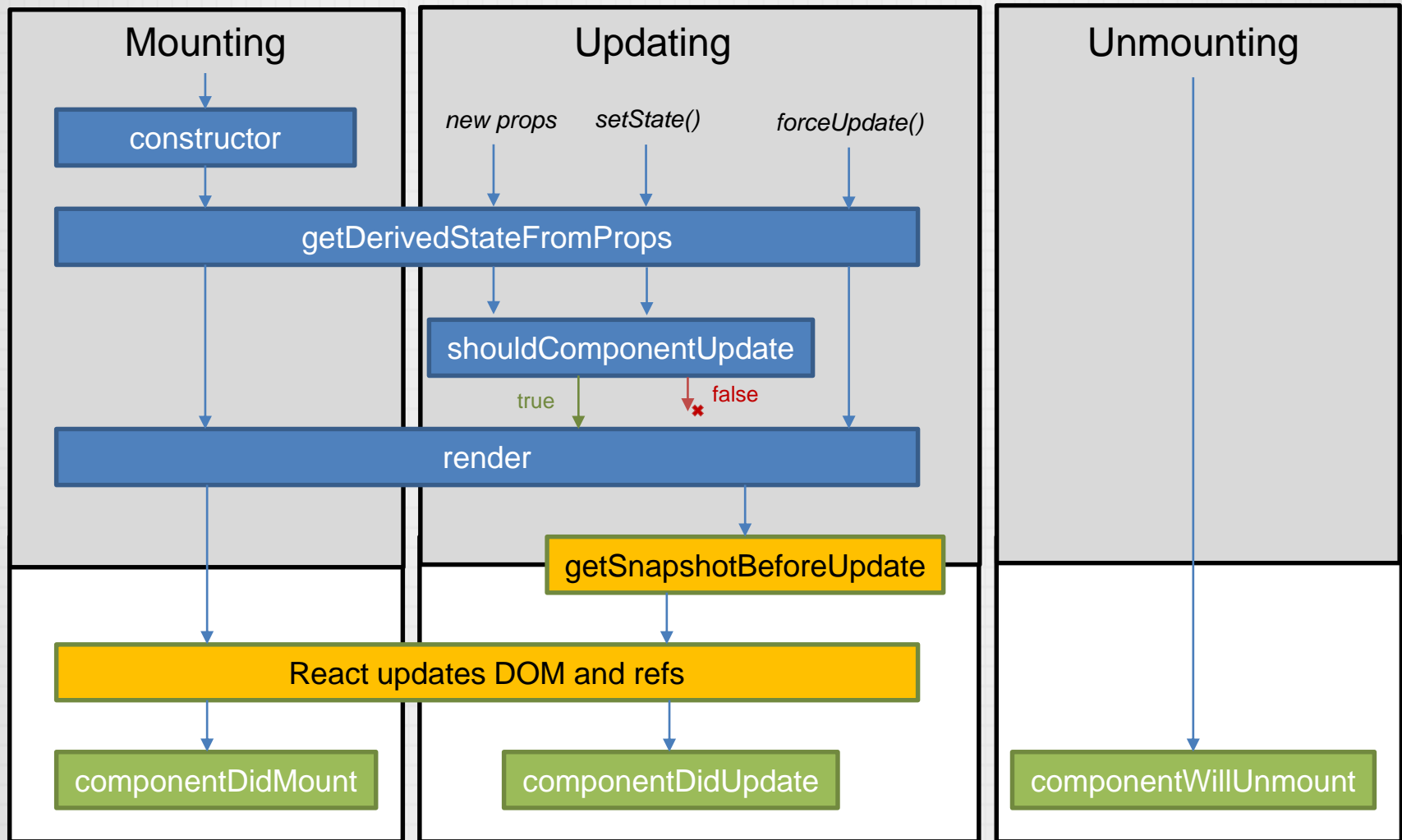
componentDidUpdate

## Unmounting

componentWillUnmount

**React 16.3 uvodi promjene!**

# React **komponente** (React 16.3+)



# React **komponente**

## getDerivedStateFromProps

```
static getDerivedStateFromProps(props, state) {  
  if (state.value !== props.value) {  
    return {  
      derivedValue: deriveValueFromProps(props),  
      mirroredProp: props.value  
    }  
  }  
  // state se ne mijenja ako se vrati null  
  return null;  
}
```

# React **komponente**

shouldComponentUpdate

```
shouldComponentUpdate(nextProps, nextState) {  
  let shouldUpdate = this.props.status !== nextProps.status;  
  return shouldUpdate;  
}
```

# React komponente

## getSnapshotBeforeUpdate & componentDidUpdate

```
class ScrollingList extends React.Component {
  constructor(props) {
    super(props);
    this.listRef = React.createRef();
  }
  getSnapshotBeforeUpdate(prevProps, prevState) {
    if (prevProps.list.length < this.props.list.length) {
      const list = this.listRef.current;
      return list.scrollHeight - list.scrollTop;
    }
    return null;
  }
  componentDidUpdate(prevProps, prevState, snapshot) {
    if (snapshot !== null) {
      const list = this.listRef.current;
      list.scrollTop = list.scrollHeight - snapshot;
    }
  }
  render() {
    return (
      <div ref={this.listRef}>{/* ...contents... */}</div>
    );
  }
}
```



```
class ProductList extends Component {
  constructor(props) {
    super(props);
    this.state = {
      title: "unknown",
      products: [],
    }
  }
  componentDidMount(){
    const fetchedProducts = this.fetchProducts();
    this.setState({
      title: "List of products",
      products: fetchedProducts,
    })
  }
  componentDidUpdate(){
    console.log("COMPONENT DID UPDATE")
  }
  fetchProducts(){
    return ["product1", "product2"]
  }
  render() {
    return ( <div>{this.state.products}</div> );
  }
}
```

hoće li se ovo ikad izvršiti?

```

class ProductList extends Component {
  constructor(props) {
    super(props);
    this.state = {
      title: "unknown",
      products: [],
    }
  }
  componentDidMount(){
    const fetchedProducts = this.fetchProducts();
    this.setState({
      title: "List of products",
      products: fetchedProducts,
    })
  }
  componentDidUpdate(){
    this.setState({title: "List of products"})
  }
  fetchProducts(){
    return ["product1", "product2"]
  }
  render() {
    return ( <div>{this.state.products}</div> );
  }
}

```

Maximum update depth exceeded. This can happen when a component repeatedly calls setState inside componentWillUpdate or componentDidUpdate. React limits the number of nested updates to prevent infinite loops.

► 4 stack frames were collapsed.

ProductList.componentDidUpdate  
C:/Users/nbugaric/Desktop/react/nwt1/src/components/productList/productList.js:19

```

16 |   })
17 | }
18 | componentDidUpdate(){
19 |   this.setState({title: "List of products"})
20 | }
21 | fetchProducts(){
22 |   return ["product1", "product2"]

```

Problem!

# React **komponente**

## `componentWillUnmount`

- clean up nakon što se komponenta uništi
- ne smije se zvati `setState`

# React **komponente**

## forceUpdate

- ukoliko želimo da se komponenta prisilno update-a
- preskače se shouldComponentUpdate()
  - samo za trenutnu komponentu, za child elemente će se pozvati
- treba izbjegavati korištenje ove metode

# Presentational vs container components

- Samo prijedlog kako graditi komponente, nije obavezno

## presentational

- Vode brigu o tome kako stvari izgledaju
- mogu sadržavati druge presentational i container komponente unutar sebe
- nemaju dependency-je na ostatak aplikacije (flux, redux)
- ne specificiraju kako se podaci učitavaju ili mijenjaju
- primaju podatke i callback-e preko props
- rijetko imaju svoj state

# Presentational vs container components

- Samo prijedlog kako graditi komponente, nije obavezno

## presentational

```
//definiranje komponente kao React komponente
class Image extends Component {
  render() {
    return <img src={this.props.image} />;
  }
}

export default Image
//definiranje komponente kao konstanta
const Image = props => (
  <img src={props.image} />
)

export default Image
```

# Presentational vs container components

- Samo prijedlog kako graditi komponente, nije obavezno

## container

- Vode brigu o tome kako stvari funkcioniraju
- mogu sadržavati druge presentational i container komponente unutar sebe
- pružaju podatke i funkcionalnost presentational i ostalim container komponentama
- Vrlo često imaju state, služe kao izvori podataka
- Često se generiraju korištenjem higher order components (npr. `connect()` iz Redux-a)

# Presentational vs container components

## container

```
class Collage extends Component {
  constructor(props) {
    super(props);
    this.state = {
      images: []
    };
  }
  componentDidMount() {
    const fetchedImages = fetch('/api/image_list');
    this.setState({images: fetchedImages});
  }
  render() {
    return (
      <div className="image-list">
        {this.state.images.map(image => {
          <div className="image">
            <img src={image.image_url} />
          </div>
        })}
      </div> )
  }
}
```



# TypeScript ?

```
npm install --save typescript @types/node @types/react @types/react-dom @types/jest
```

# React + typescript

```
import React, { Component } from 'react';
export interface ProductItemProps {
  product: IProduct;
}
export interface ProductItemState {
  title: string;
}
class ProductItem extends React.Component<ProductItemProps, ProductItemState> {
  constructor(props: ProductItemProps) {
    super(props);
    this.state = { title: '' };
  }
  render() {
    return ( <div></div> );
  }
}
export default ProductItem;
```



# Redux

# Redux (<https://redux.js.org>)



A predictable state container for JavaScript apps.

GET STARTED



## Predictable

Redux helps you write applications that **behave consistently**, run in different environments (client, server, and native), and are **easy to test**.



## Centralized

Centralizing your application's state and logic enables powerful capabilities like **undo/redo**, **state persistence**, and much more.



## Debuggable

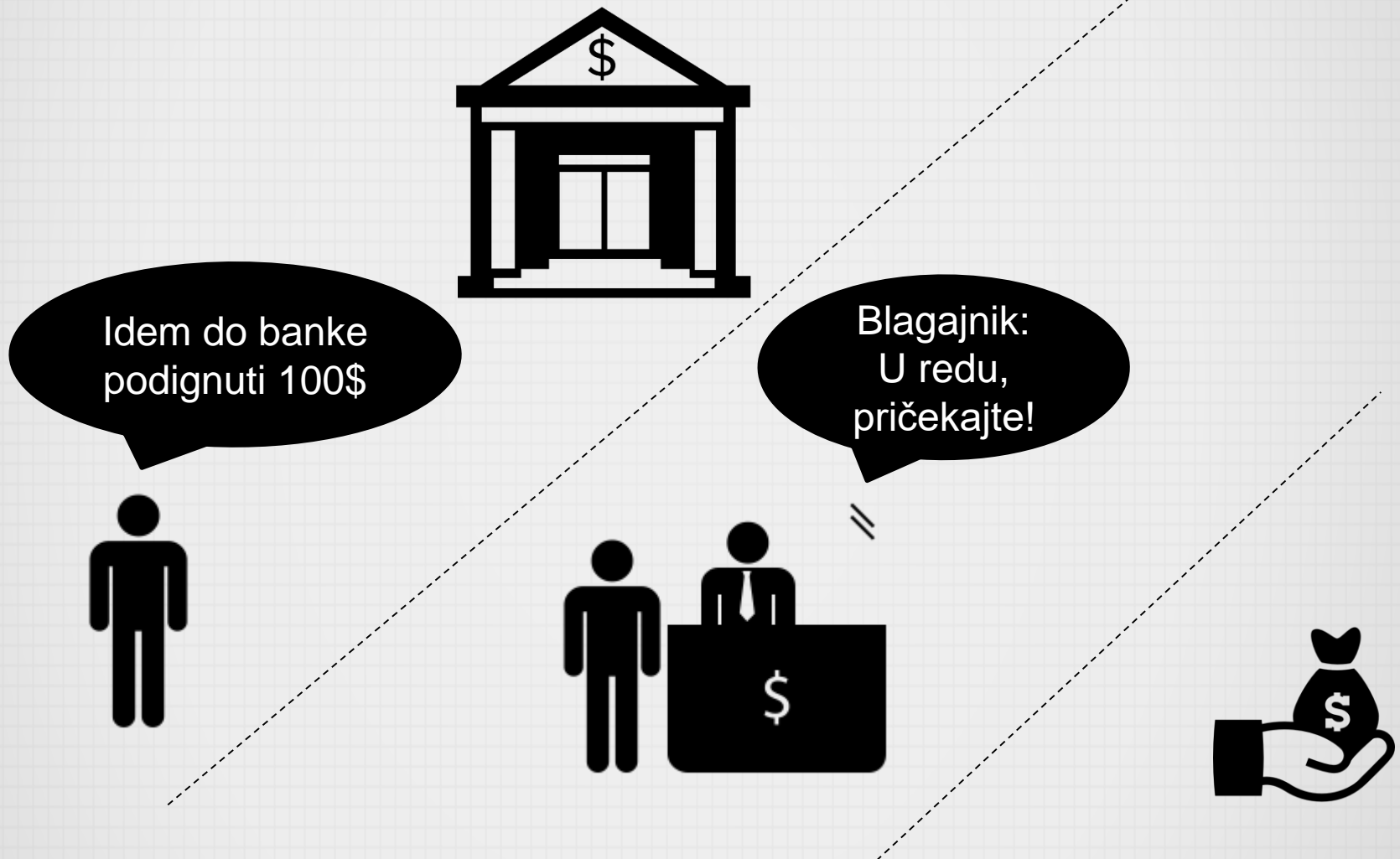
The Redux DevTools make it easy to trace **when, where, why, and how your application's state changed**. Redux's architecture lets you log changes, use "time-travel debugging", and even send complete error reports to a server.



## Flexible

Redux works with any UI layer, and has a **large ecosystem of addons** to fit your needs.

# Redux - analogija



# Redux - analogija




## REDUX STORE:

- banka (trezor) == redux store
- novac je dobro zaštićen u trezoru banke, slično kao i state vaše aplikacije.
- Single source of truth – podaci o novcu i novac su zapisani u banci, ne negdje drugo.

## Pravilo 1:

Jedan **STATE** objekt aplikacije pod kontrolom jednog **STORE**-a

# Redux - analogija



Idem do banke  
podignuti 100\$

## ACTION:

- Donijeli smo odluku što želimo – podignuti novac
- Također, odredili smo i koliko novca želimo podignuti (dodatne informacije za ovu akciju)

```
{  
  type: "WITHDRAW_MONEY",  
  amount: "$100"  
}
```

## Pravilo 2:

State je read-only.

Jedini način da se promijeni **STATE** je tako da se emitira **ACTION**  
(u biti radimo kopiju state-a koji sad postaje važeći)

# Redux - analogija



## REDUCER:

- Banka neće dopustiti da sami ušetate u sef !!!
- Za nas će to napraviti netko drugi, u kontroliranim uvjetima

## Pravilo 3:

Da bi specificirali kako se **STATE** mijenja na temelju pojedine **ACTION**, pišemo **PURE REDUCERS**.



# Redux – three principles

1. ONE IMMUTABLE STORE
2. ACTION TRIGGERS CHANGES
3. REDUCERS UPDATE STATE

# Redux

## Store

Objekt koji sadrži aplikacijski state tree. U aplikaciji bi trebao biti samo jedan store.

## State

State (ili state tree) u Redux API-ju predstavlja vrijednost state-a kojim upravlja store i koji se može dohvatiti pomoću `getState()`. Predstavlja cjelokupni state Redux aplikacije.

## Action

Action je jednostavni JavaScript objekt koji predstavlja intenciju (namjeru) da se promijeni state. Akcije su jedini način na koji se može ubaciti podatke u store. Sve podaci moraju se eventualno dispatchati kao akcije.

Akcije moraju imati polje koje definira tip akcije koja se obavlja, a to se najčešće radi preko konstanti.

# Redux

## Reducer

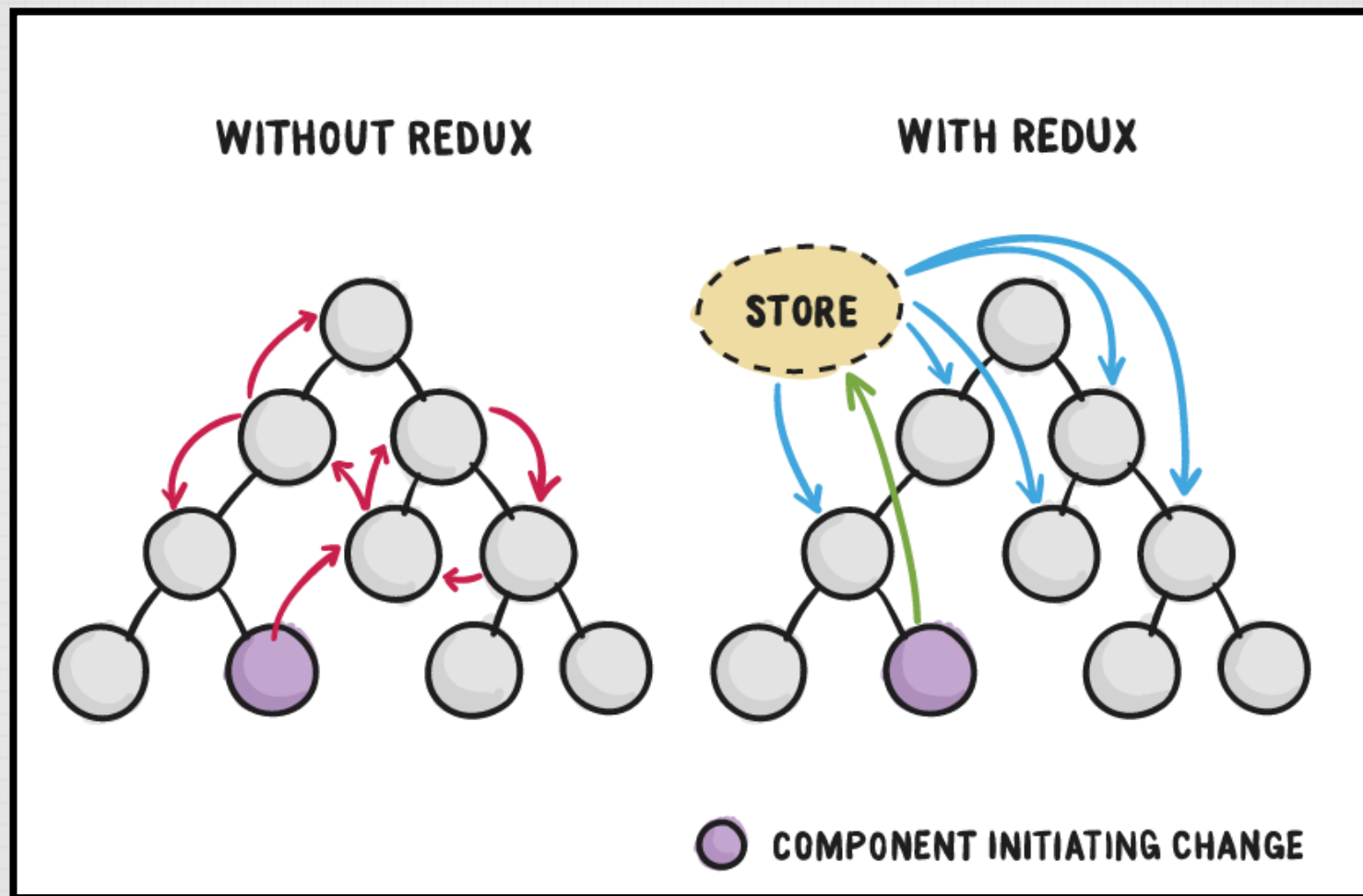
Reducer (općenito) je funkcija koja prima akumuliranu vrijednost i trenutnu vrijednost i vraća novu akumuliranu vrijednost. U Redux-u, reduceri uzimaju akciju, te na temelju te akcije modificiraju state (u biti rade njegovu modificiranu kopiju).

Moraju biti pure funkcije (koje za isti input uvijek daju isti output). Nikad ne pozivati neki API unutar reducera.

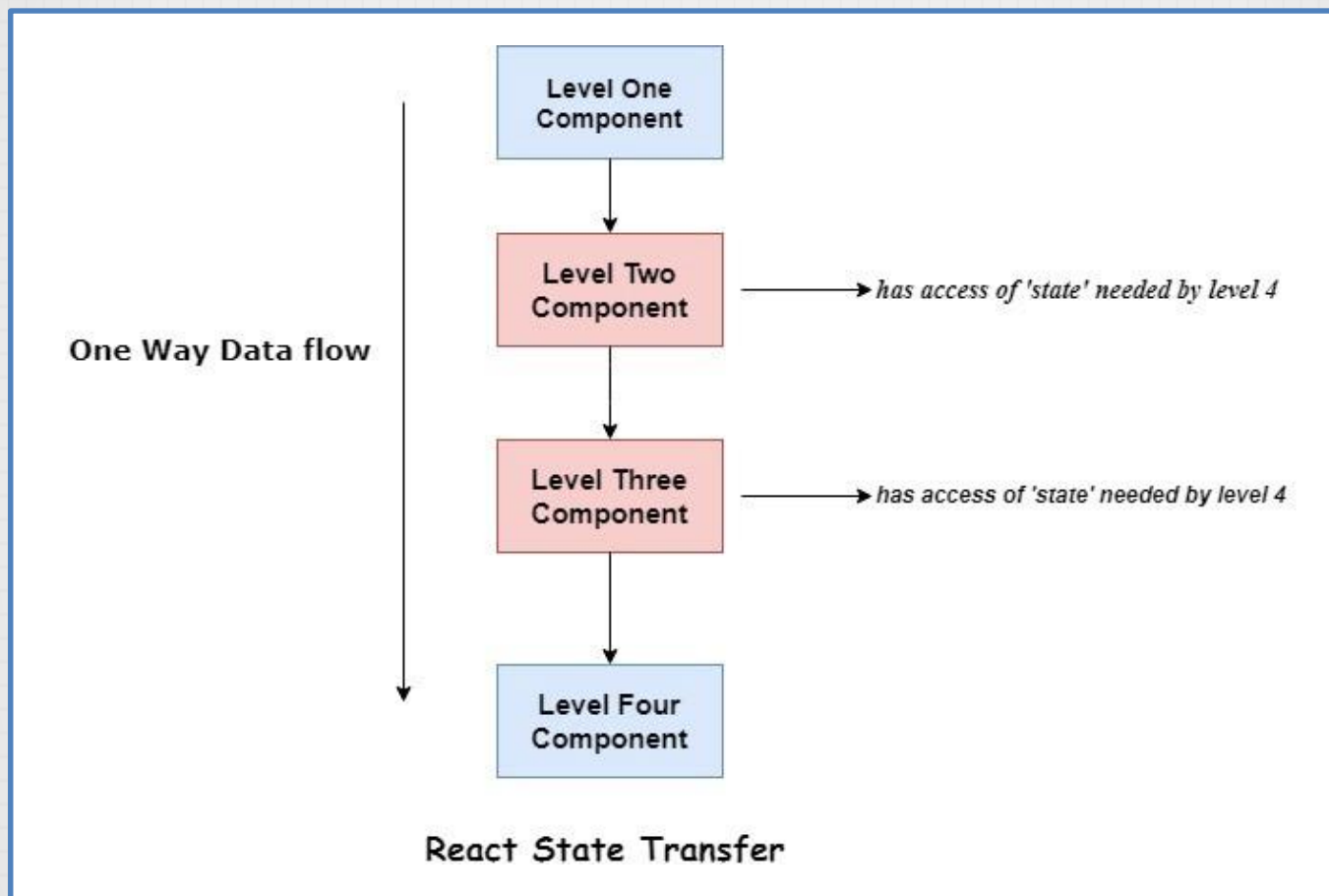
## Action creator

Akcije su jednostavni Javascript objekti. Action creatori su funkcije koje kreiraju action-e.

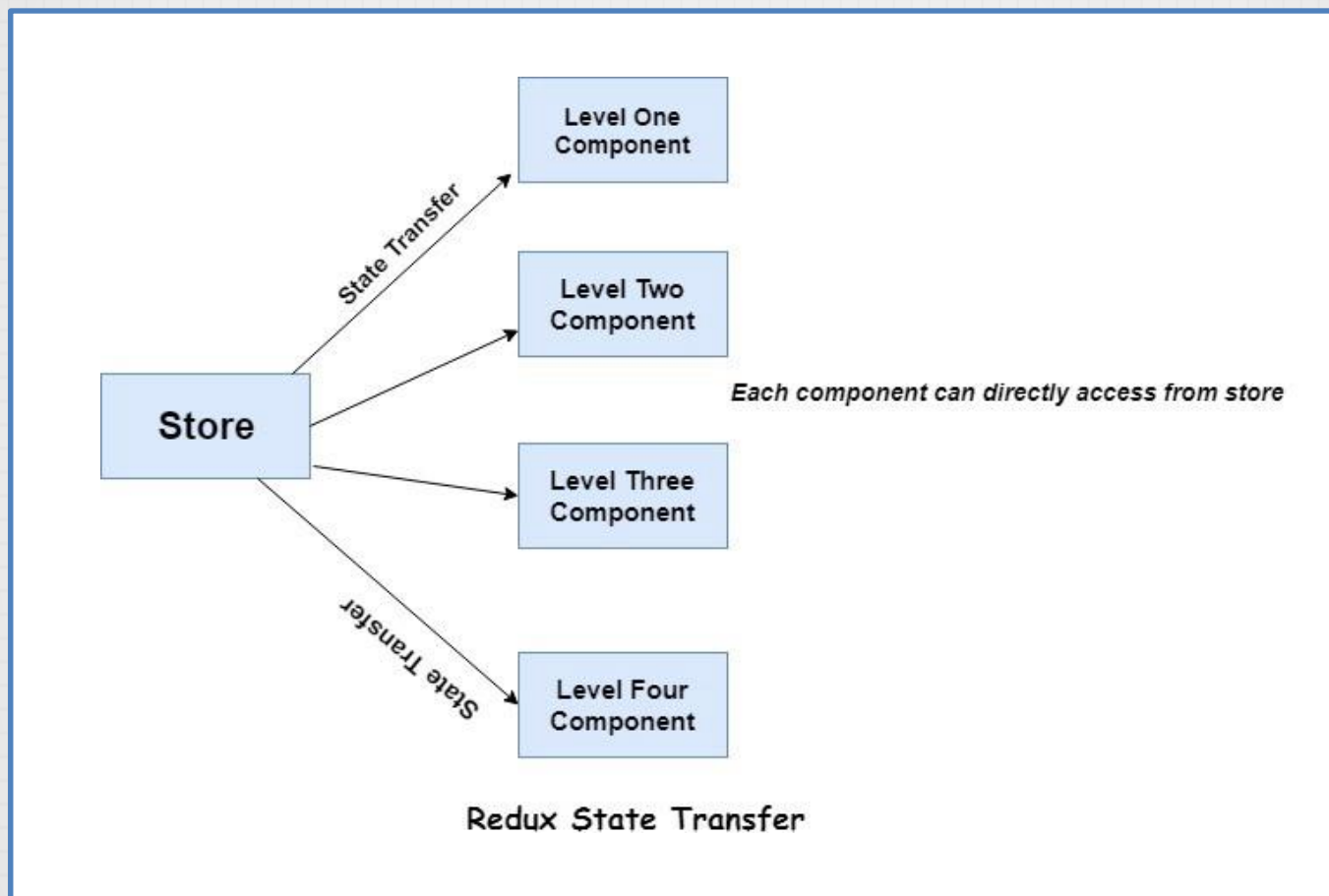
# Redux



# Nedostaci Reacta



# Nedostaci Reacta



# Redux – ostale prednosti

## Predictable state

Reducers su pure functions, ako se isti state i ista akcija pošalju reduceru, uvijek ćemo dobiti isti rezultat.

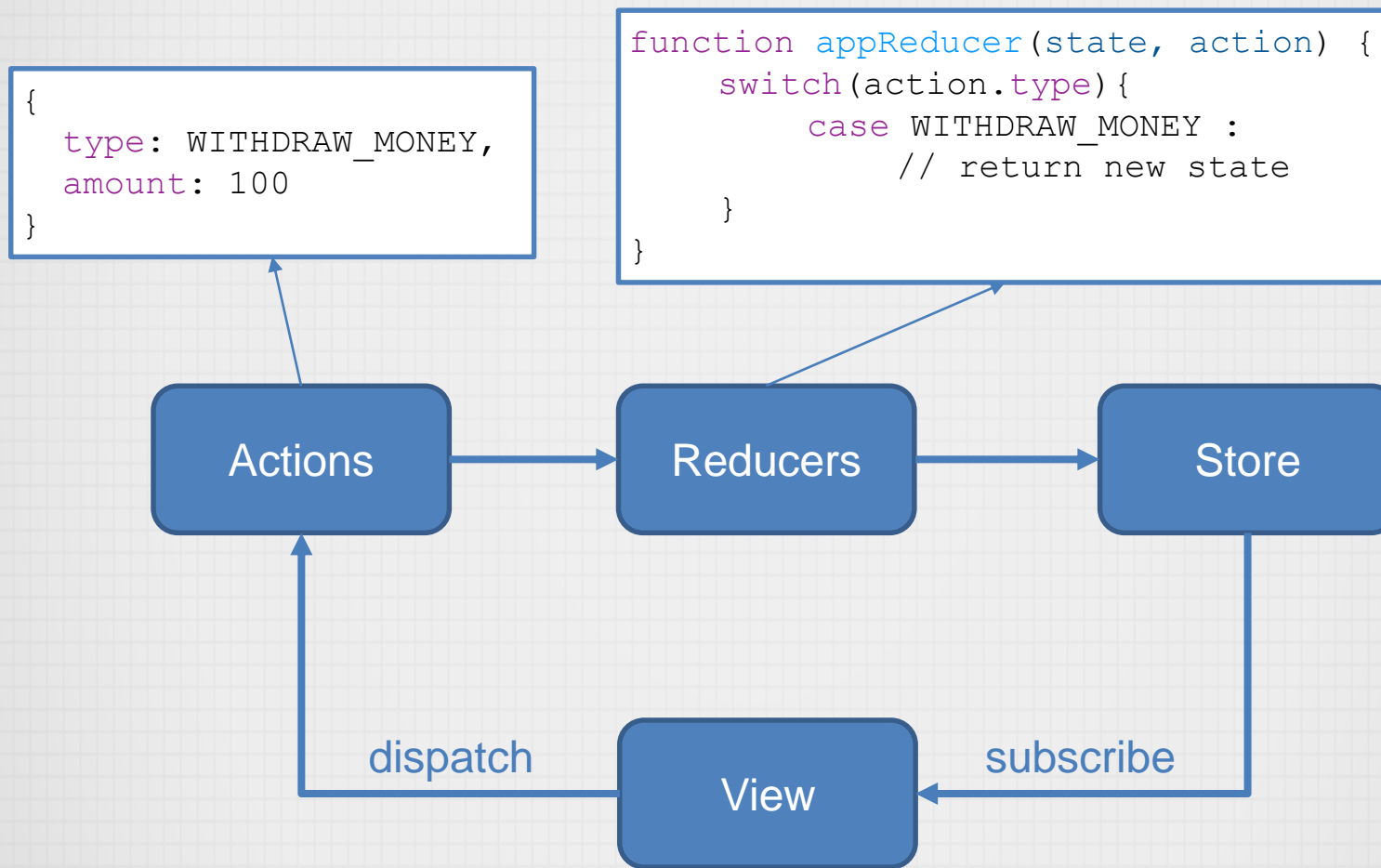
## Održavanje

Redux je strog oko načina kako je kod organiziran i svi ga se moraju pridržavati.

## Debugging & time travel

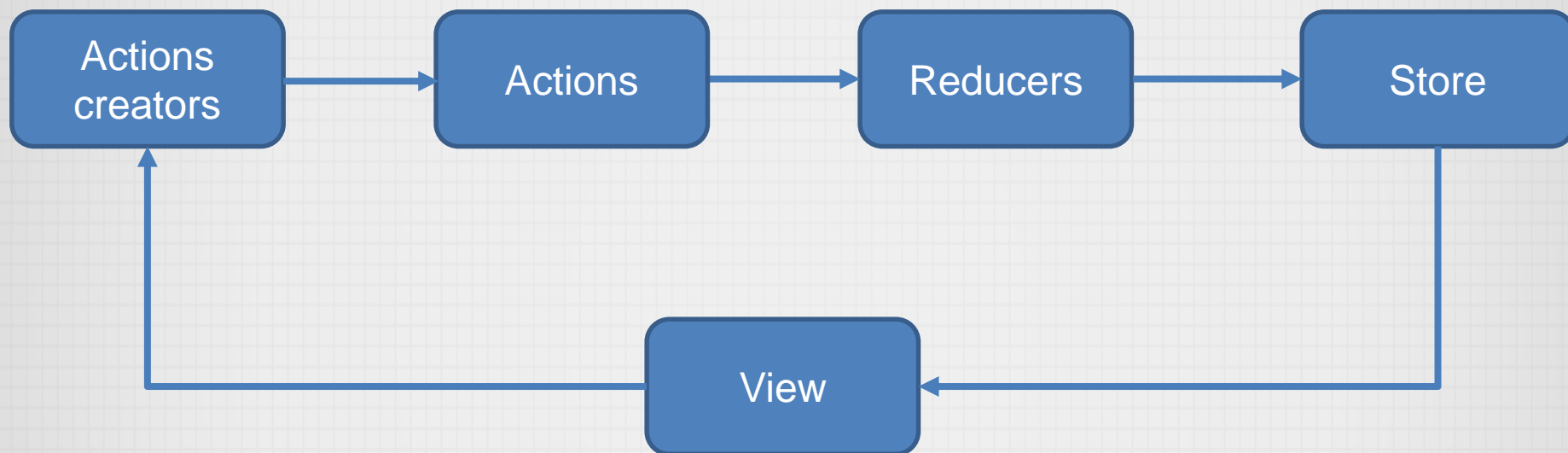
State i akcije se logiraju, olakšava debugiranje.

# Redux





# Redux



## Actions:

- Jednostavni JavaScript objekti

## Action creators:

- Funkcije koje kreiraju akcije

```
export const withdrawMoneyAction =  
  (amount) => dispatch => {  
    dispatch({  
      type: WITHDRAW_MONEY,  
      amount: amount  
    })  
  }  
}
```

# Redux - primjer

Todos

# Redux - primjer

## Store

```
export default function configureStore(initialState = {}) {  
  return createStore(  
    rootReducer,  
    initialState,  
    applyMiddleware(thunk)  
  );  
}
```

## rootReducer

```
import todos from './todoReducer';  
  
export default combineReducers({  
  todos  
});
```

# Redux - primjer

## Action creator example

```
export const editTodoAction = (data) => dispatch => {  
  dispatch({  
    type: EDIT_TODO,  
    payload: {  
      id: data.id,  
      text: data.text,  
    }  
  })  
}
```

## Reducer example (mora biti pure)

```
case EDIT_TODO: {  
  return state.map(todo => {  
    return todo.id === action.payload.id ? {  
      ...todo,  
      text: action.payload.text  
    } : todo;  
  });  
}
```

# Higher order components

- Riječ je o funkciji koja uzima komponentu i vraća novu komponentu.

```
const EnhancedComponent = higherOrderComponent(WrappedComponent);
```

- Obične komponente pretvaraju props u UI
- HOC komponente pretvaraju jednu komponentu u drugu
- Najpoznatija HOC komponenta je connect()

# Redux - primjer

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

const mapDispatchToProps = dispatch => ({
  completeTodo: (id) => dispatch(completeTodoAction(id)),
  deleteTodo: (id) => dispatch(deleteTodoAction(id)),
  editTodo: (id, text) => dispatch(editTodoAction(id, text))
})

class TodoItem extends React.Component {
  ...

  render() {
    ...
    this.props.todos.map(...)
    ...
    this.props.completeTodo()
    ...
  }
}

export default connect(mapStateToProps, mapDispatchToProps)(TodoItem);
```

# Redux - primjer

# DEMO

# Za domaći rad

rutiranje

middleware in Redux

...