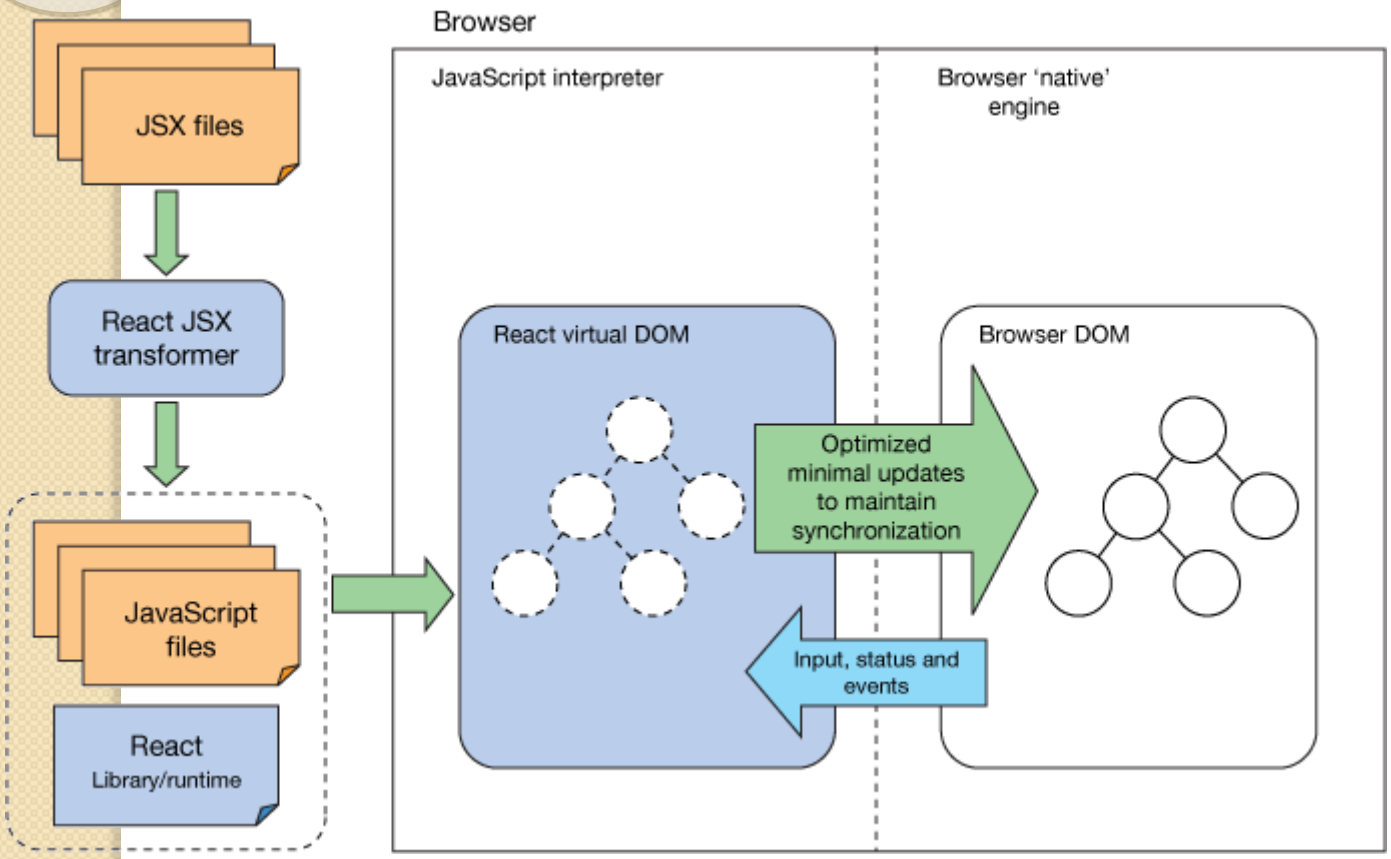
React

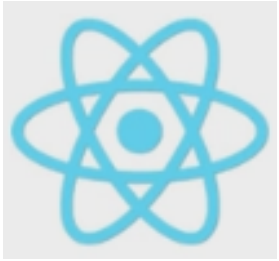

Browser
JavaScript interpreter
Browser 'native' engine
JSX files
React JSX transformer
JavaScript files
React Library/runtime
React virtual DOM
Browser DOM
Optimized minimal updates to maintain synchronization
Input, status and events


FLUX
React

Що таке ReactJS ?

- **React.js – це open-source JavaScript бібліотека для створення користувацьких інтерфейсів, яка покликана вирішувати проблеми при частковому оновлені вмісту веб-сторінки, з якими стикаються в розробці одно-сторінкових додатків.**
- **Розробляється Facebook, Instagram** і спільнотами індивідуальних розробників (з 2013 р.).
- **React – це V (view) в MVC шаблоні**.
- **Є ідеальним для** large-scale single page applications (SPA)
- **React підтримує дуже швидку Virtual DOM**, а не покладається виключно на DOM браузера.
- **JSX. React-компоненти** зазвичай пишуть на JSX – розширенні JavaScript синтаксису.

Why React ?

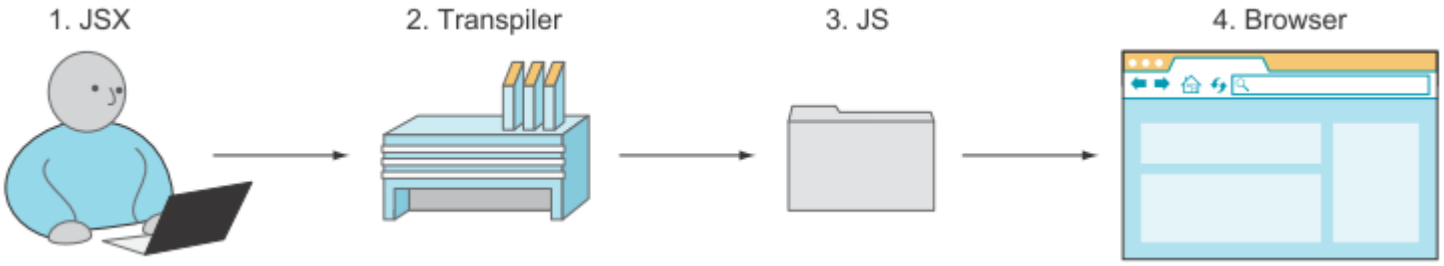


- Fast
- Composable
- Pluggable
- Isomorphic Friendly
- Simple
- Battle Proven

Основні особливості React

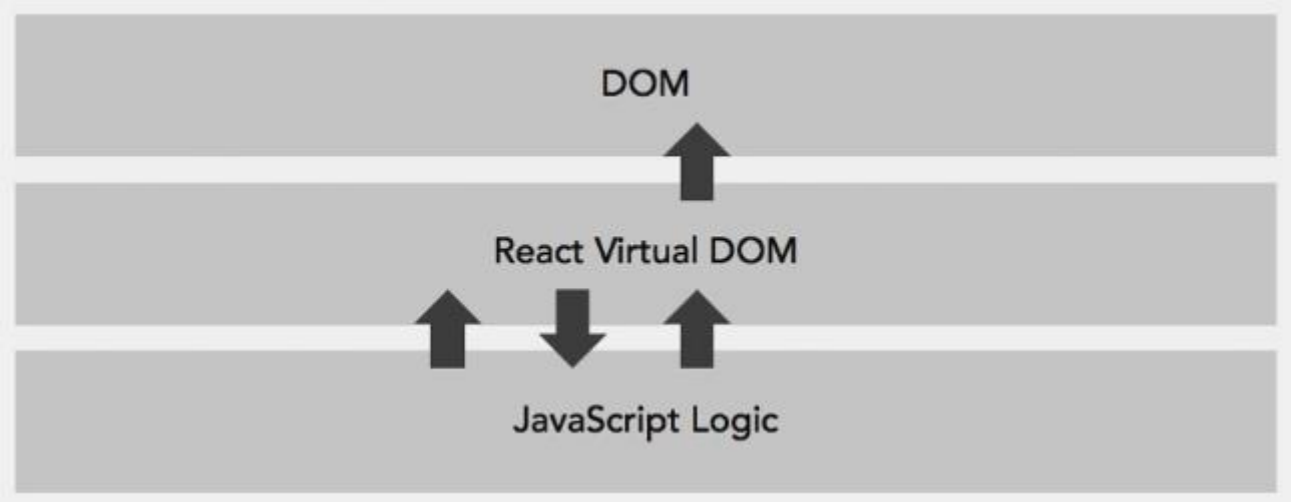
- **React** зосереджений на **компонентах**.
- Кожен **компонент** має свій **стан** (state), який може змінюватись, і коли це відбувається - компонент рендериться наново
- Кожен **компонент** має функцію **render**
- Замість того, щоб використовувати шаблонізатори, компоненти пишуться на звичайному **JS**, ну і **JSX** для рендерингу
- **React** робить по-справжньому потужним і швидким те, яким чином він рендерить компоненти, коли їхні **властивості** (props) та **стан** змінюються. Замість того, щоб проводити маніпуляції із існуючим **DOM**'ом, як інші фреймворки це роблять, він використовує так званий *віртуальний* DOM.

- Коли **стан компонента** змінюється, **React** перевіряє на наявність відмінностей *віртуальний DOM* та *наявний DOM*, та застосовує лише ці відмінності, а не перерисовує цілий DOM
- **Компоненти** можна **вкладати** один в одного, тому ви можете писати по-справжньому маленькі компоненти, які можна використовувати знову і знову, і які будуть передавати дані та їх обробку (bind event) від батьків до потомків (from parents to children)
- **Для JSX** існує ***перетворювач синтаксису*** *JavaScript **X**ML* в JavaScript. Це означає, що ви можете писати на **X**ML подібному синтаксису в своїх компонентах, і він буде перетворений у звичайні **JavaScript** функції.



Чому React такий швидкий?

- JS об’єкти більш швидкі ніж DOM об’єкти
- React Virtual DOM є JS об’єктом
- React ніколи не читає з “реальної” DOM
- React тільки записує в реальну DOM при потребі

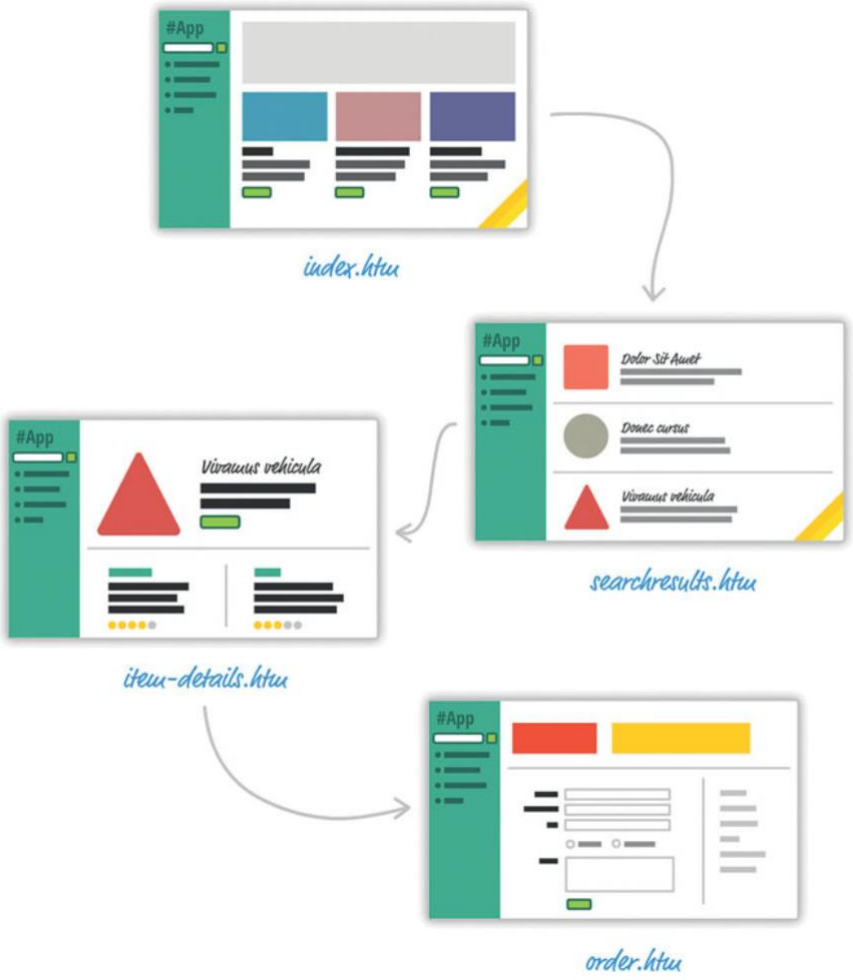


Why To Use ReactJS?

- **UI код стає більш зрозумілим і легким у супроводі**
- **Створювані UI Components дозволяють досягти модульності і повторного використання**
- **React дозволяє здійснювати рендеринг як на боці клієнта, так і на боці сервера**
- **Він використовує концепцію, звану Virtual DOM, для селективної перебудови піддерева в DOM**

Зміни в проектуванні web-додатків

Багатосторінковий дизайн старої школи



Односторінкові додатки нової школи

Труднощі при створенні односторінкових додатків

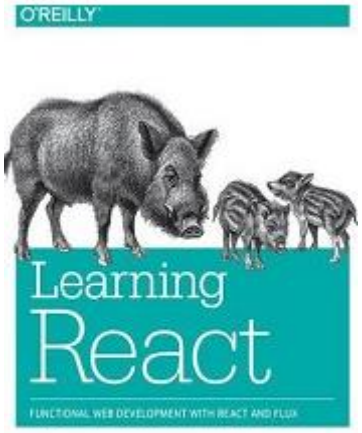
- У односторінковому додатку більша частина часу йде на синхронізацію даних з призначеним для користувача інтерфейсом.
- Маніпулювання DOM відбувається дуже повільно.
- При роботі з HTML-шаблонами можуть виникати труднощі.

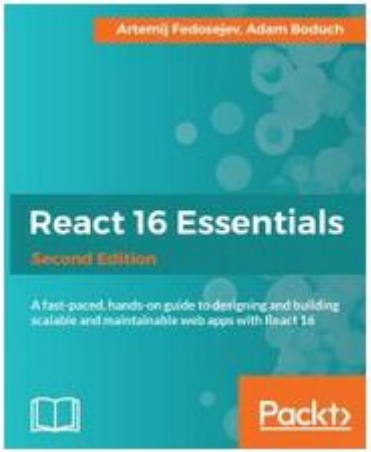
Література

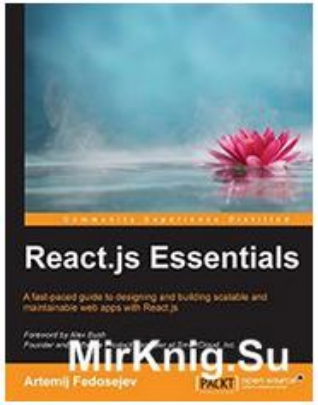
- React в действии. – Питер, 2019
- The React Workshop. – Packt, 2020
- Banks A., Porcello E. Learning React – O'Reilly Media, 2020
- Full-Stack React, TypeScript, and Node. – Packt, 2020
- Кирупа Ч. Изучаем React. – М.: Эксмо, 2019. – 368 с.
- Digesting React. – 2020

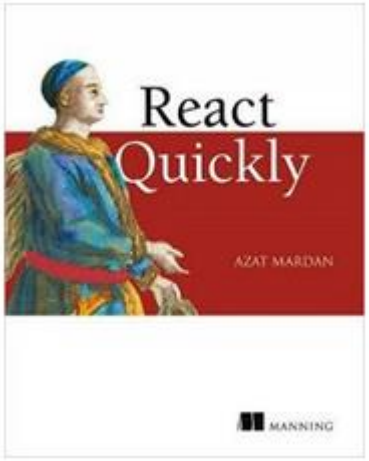
Література

- Бэнкс А., Порселло Е. React и Redux. Функциональная веб-разработка, 2018
- Хортон А., Вайс Р. Разработка веб-приложений в ReactJS, ДМК. 2016
- Стоян Стефанов React.js. Быстрый старт. – Питер, 2017
- Artemij Fedosejev, Adam Boduch React 16 Essentials, Second Edition. - Packt Publishing, 2017
- Learning React by Alex Banks and Eve Porcello. - O'Reilly Media, 2016







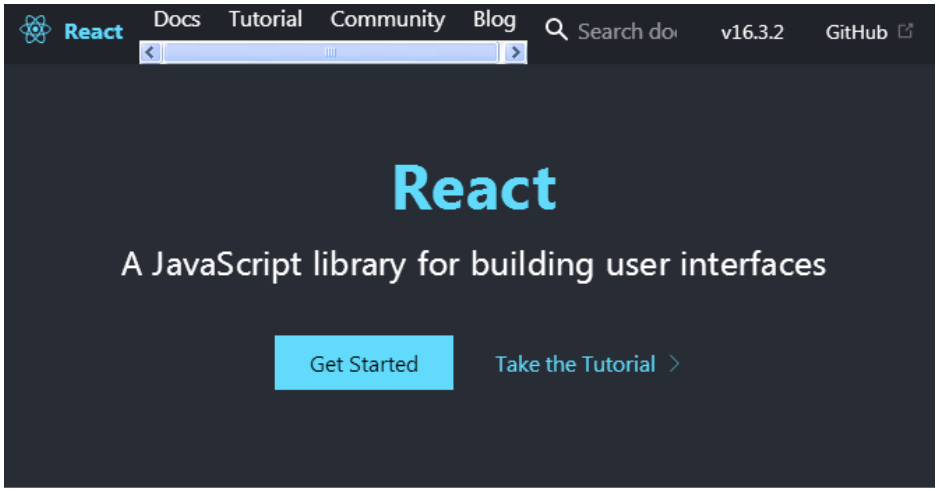


Відеокурси

- Lynda - Learning React.js (Jan 2018)
- PluralSight - React.js Getting Started (Jan 2018)
- Lynda - React for Web Designers (Apr 2018)
- Lynda – React Server-Side Rendering (Apr 2018)
- LiveLessons - React.js Fundamentals and Advanced (Apr 2018)
- Pluralsight – A Practical Start with React (Apr 2018)
- Pluralsight – Client Side React Router 4 (Mar 2018)
- Udemy - The Complete React Redux Bootcamp with ES6 (Feb 2018)
- ITVDN - ReactJS Starter (2016)
- ITVDN - React Advanced (2017)
- Специалист. JavaScript. Уровень 3. React и JSX (2017)
- [MonsterLessons] Интернет-магазин на React+Redux (2017)

Корисні посилання

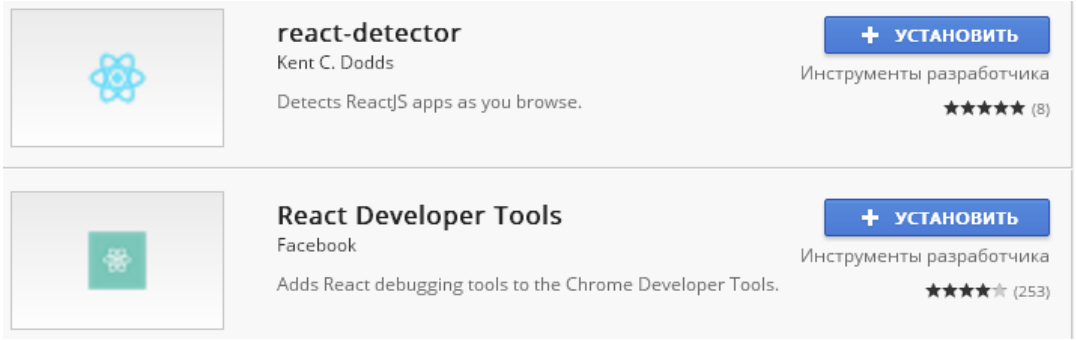
- Офіційний сайт – https://reactjs.org/



- Введение в React https://metanit.com/web/react/ (Последнее обновление: 24.02.2021)

Налаштування React tools разом з Chrome

- Зайти на Chrome Web Store (Інтернет магазин Chrome) (https://chrome.google.com/webstore/category/extensions)
- В вікні пошуку набрати react-detector



- В вікні пошуку набрати react developer tools
- Встановити обидва розширення
- Перевірити роботу можна на сайті https://airbnb.com (Ctrl+Shift+J) – SearchBar зроблена на React

“Hello World” на React (JS синтаксис)

- Щоб працювати з React в браузері потрібні дві бібліотеки: React і ReactDOM. Їх можна підключити через CDN (content delivery network)

```
<script crossorigin src="https://unpkg.com/react@16/umd/react.development.js"></script>
<script crossorigin src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
```

- Файл index.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
  <title>Getting Started with React</title>
</head>
<body>
  <div id="react-container"></div>
  <script type="text/javascript">
    ReactDOM.render(
        React.createElement('h1', null, 'Hello World'),
        document.getElementById('react-container')                    )
  </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <script
    src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-
    dom.development.js"></script>
    <title>Getting Started with React</title>
</head>
<body>
    <div id="react-container"></div>
    <script type="text/javascript">
        ReactDOM.render(
            React.createElement('h1', null, 'Hello World'),
            document.getElementById('react-container')          )
    </script>
</body>
</html>
```

“Hello World” на React (JSX синтаксис)

- Тут потрібен Babel (https://babeljs.io/). Це транспайлер (компілятор ES6 => ES5)

Файл index2.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-
    dom.development.js"></script>
    <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
    <title>Getting Started with React</title>
</head>
<body>
    <div id="react-container"></div>
    <script type="text/babel">
            ReactDOM.render(
                    <h1>Hello World</h1>,
                    document.getElementById('react-container')                    )

    </script>
</body>
</html>
```

Файл index2.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <script
    src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-
    dom.development.js"></script>
    <script src="https://unpkg.com/babel-
    standalone@6.15.0/babel.min.js"></script>
<title>Getting Started with React</title>
</head>
<body>
    <div id="react-container"></div>
    <script type="text/babel">
            ReactDOM.render(
                    <h1>Hello World</h1>,
                    document.getElementById('react-container')                    )

    </script>
</body>
</html>
```

Компонент Hello (stateful class component)

Файл index3.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <style>
            #fancy {
                background-color: gray;
            }
            .heading {
                color: white;
            }
    </style>
<script
    src="https://unpkg.com/react@16/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom@16/umd/react-
    dom.development.js"></script>
<script src="https://unpkg.com/babel-
    standalone@6.15.0/babel.min.js"></script>
<title>Getting Started with React</title>
</head>
```

```
<body>
    <div id="react-container"></div>
    <script type="text/babel">
        class Hello extends React.Component {
            render() {
                return (
                    <div id="fancy">
                        <h1 className="heading">Hello World</h1>
                    </div>
                )
            }
        }
        ReactDOM.render(
            <Hello/>,
            document.getElementById('react-container')
        )
    </script>
</body>
</html>
```

Створення stateless-компонента

- Інший спосіб створення компонента полягає в використанні stateless functional component.
- В попередньому прикладі клас треба замінити на наступну функцію:

```
const Banner = () => <h1>Welcome To React</h1>
```

- Також треба трохи підкорегувати функцію Render:

```
ReactDOM.render( <Banner />,
document.getElementById('react-container');
```

- Можете перевірити результат в браузері
- Можна трохи ускладнити наш банер:

```
const Banner = () => (
        <div id="fancy">
            <h1 className="heading">Welcome To React</h1>
            <p>Have fun!</p>
        </div>
)
```

(Файл index4.html)

Використання властивостей

- Коли ми хочемо відобразити динамічні дані в React, ми можемо використати властивості (props).
- Можна вважати, що props це об'єкт, в якому ключі (keys) є властивостями компонента.
- Повернемось до нашого компонента Hello. Додамо динаміку:

```
class Hello extends React.Component {
        render() {
            return (<div>
                    <h1>Hello {this.props.firstName} </h1>
                    </div>
    )}}
ReactDOM.render( <Hello firstName="Nick" />,
document.getElementById('react-container');
```

- Результат можна переглянути в браузері (index5.html).

- Додамо ще одну властивість

```
<h1>Hello  {this.props.firstName} </h1>
<p>Learning React Version {this.props.version}</p>
. . . . . .
 ReactDOM.render( <Hello firstName="Nick" version={16} />,
document.getElementById('react-container');
```

- Результат можна перевірити в браузері

- Для stateless-компонентів об'єкт props можна передати через параметр функції:

```
const HelloStateless = (props) => <h1>Hello
{props.firstName}</h1>
ReactDOM.render( <HelloStateless firstName="Nick" />,
document.getElementById('react-container');
```

- Результат можна перевірити в браузері

Використання стану (state)

- Однією з найбільш важливих концепцій React є стан (state).
- Коли дані стану компонента змінюються, функція render буде викликатись повторно, щоб відобразити зміни стану.
- Розглянемо невеликий приклад.

```
class Checkbox extends React.Component {
      render() {
                return (
                <div>
                   <input type="checkbox" />
                </div>
                )
      }
}
ReactDOM.render( <Checkbox />,
document.getElementById('react-container');
```

- Перевіримо роботу в браузері.

- Додамо стан через конструктор.

```
class Checkbox extends React.Component {
      constructor(props) {
        super(props)
        this.state = {
            checked: false
        }
      }
    render() {
            return (
            <div>
                <input type="checkbox” />
            </div>
            )
    }
}
```

Коли буде відбуватися рендерінг компонента, змінна стану checked буде мати значення false.

- Далі додамо обробник зміни стану Checkbox. Для цього після конструктора додамо метод:

```
handleCheck() {
    this.setState({
      checked: !this.state.checked
    })
}
```

- Далі додамо зміни до DOM-елементу

```
<input type="checkbox“ onChange={this.handleCheck} />
```

- Далі ми повинні приєднати (bind) handleCheck. Для цього в конструкторі допишемо останнім рядком (перед “}”) :

```
this.handleCheck = this.handleCheck.bind(this)
```

- Подивимось в браузері що буде відбуватись. Ми побачимо, що все працює, але ми не побачимо, що реально відбувається за сценою.

- Для наглядності додамо маленьке повідомлення, щоб було зрозуміло, що відбувається.

```
render() {
  let msg
  if(this.state.checked) {
        msg = "checked"
  } else {
        msg = "not checked“
  }
  return (
        <div>
          <input type="checkbox"  onChange={this.handleCheck} />
          <p>This box is {msg}</p>
        </div>
  )
}
```

- Тепер можна поекспериментувати в браузері, і буде помітне оновлення при зміні стану.

- І останнє. Спробуємо в конструкторі змінити початковий стан нашого Checkbox:

```
this.state = {
    checked: true
}
```

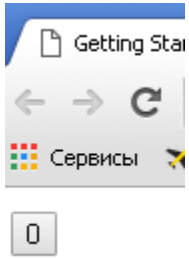
- Оновимо браузер. Checkbox залишиться невибраним, а повідомлення говоритиме протилежне. Щоб це виправити, додамо defaultChecked :

```
<input type="checkbox"  onChange={this.handleCheck}
defaultChecked={this.state.checked} />
```

- Оновимо браузер. Тепер все буде працювати вірно.
- Вихідні коди цього прикладу можна знайти в каталозі “Приклади” нашого курсу (файл checkboxComponent.html).

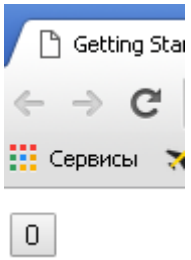
Компонент Button (при кліку лічильник збільшується)

Файл buttonCounter.html



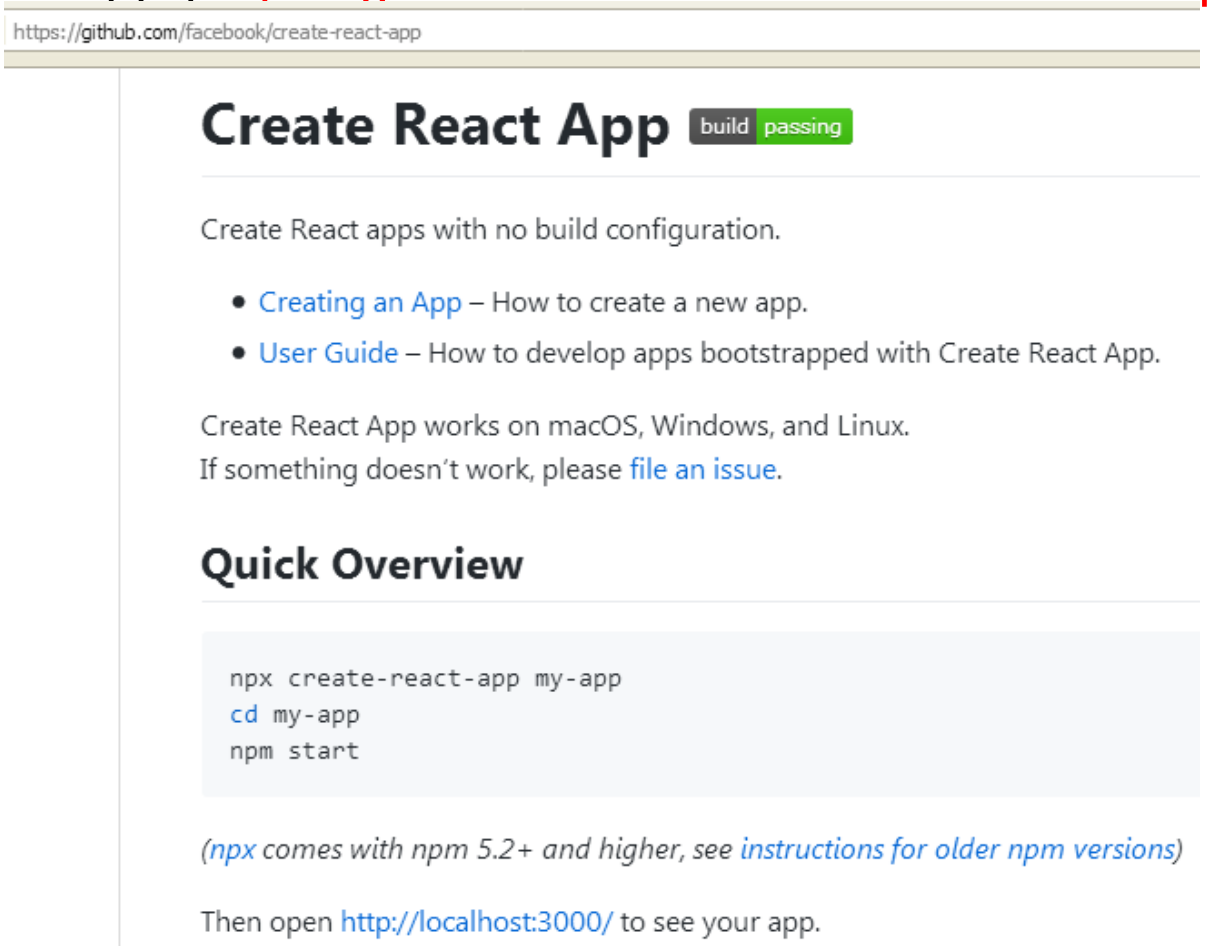

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<script
    src="https://unpkg.com/react@16/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom@16/umd/react-
    dom.development.js"></script>
<script src="https://unpkg.com/babel-
    standalone@6.15.0/babel.min.js"></script>
<title>Getting Started with React</title>
</head>
<body>
<div id="react-container"></div>
```

```
<script type="text/babel">
class Button extends React.Component{
    state = {counter: 0};
  handleClick = () => {
    this.setState((prevState) => ({
        counter: prevState.counter + 1
      })
    )
  };
  render(){
    return(
      <button onClick={this.handleClick}>
      {this.state.counter}
      </button>
    );}
}
ReactDOM.render(<Button />,
          document.getElementById('react-container')
)
</script>
</body>
</html>
```



Використання create-react-app

- Один з найцікавіших інструментів в екосистемі React - це Create React App (https://github.com/facebook/create-react-app) – 2016рік.



- Create React App дозволяє генерувати проект React з нуля без витрат часу на конфігурування інструментів типу Webpack і Babel.
- Щоб почати з Create React App ми повинні встановити його глобально за допомогою npm (на комп'ютері треба мати встановленим Node>= 6 версії):

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

- Наступним кроком буде генерація проекту

```
create-react-app  hackernews
cd hackernews
```

Далі треба відкрити застосунок в вашому редакторі.

Структура каталогів застосунку

```
hackernews/
  README.md
  node_modules/
  package.json
  .gitignore
  public/
    favicon.ico
    index.html
    manifest.json
  src/
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    registerServiceWorker.js
```

- public/: The folder holds all your files when building your project for production.

Файл src/App.js (написаний на JSX)

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

class App extends Component {
render() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <h1 className="App-title">Welcome to React</h1>
      </header>
      <p className="App-intro">
  To get started, edit <code>src/App.js</code> and save to reload.
      </p>
    </div>
);
}
}
export default App;
```

Методи життєвого циклу компонентів React

React надає розробникам безліч методів і «хуків», які викликаються під час життєвого циклу компонента, вони дозволяють нам оновлювати UI і стан додатка.

- **componentWillMount()** - зазвичай викликається перед відображенням компонента, тому його часто використовують для завантаження даних
- componentDidMount() - викликається після монтування компонента
- shouldComponentUpdate(nextProps, nextState, nextContext) - переіряє, чи потрібен re-render
- componentDidUpdate(prevProps, prevState, prevContext) – викликається після того, як відпрацювала функція render.
- . . . . . .

Типи методів життєвого циклу
(Types of lifecycle methods)

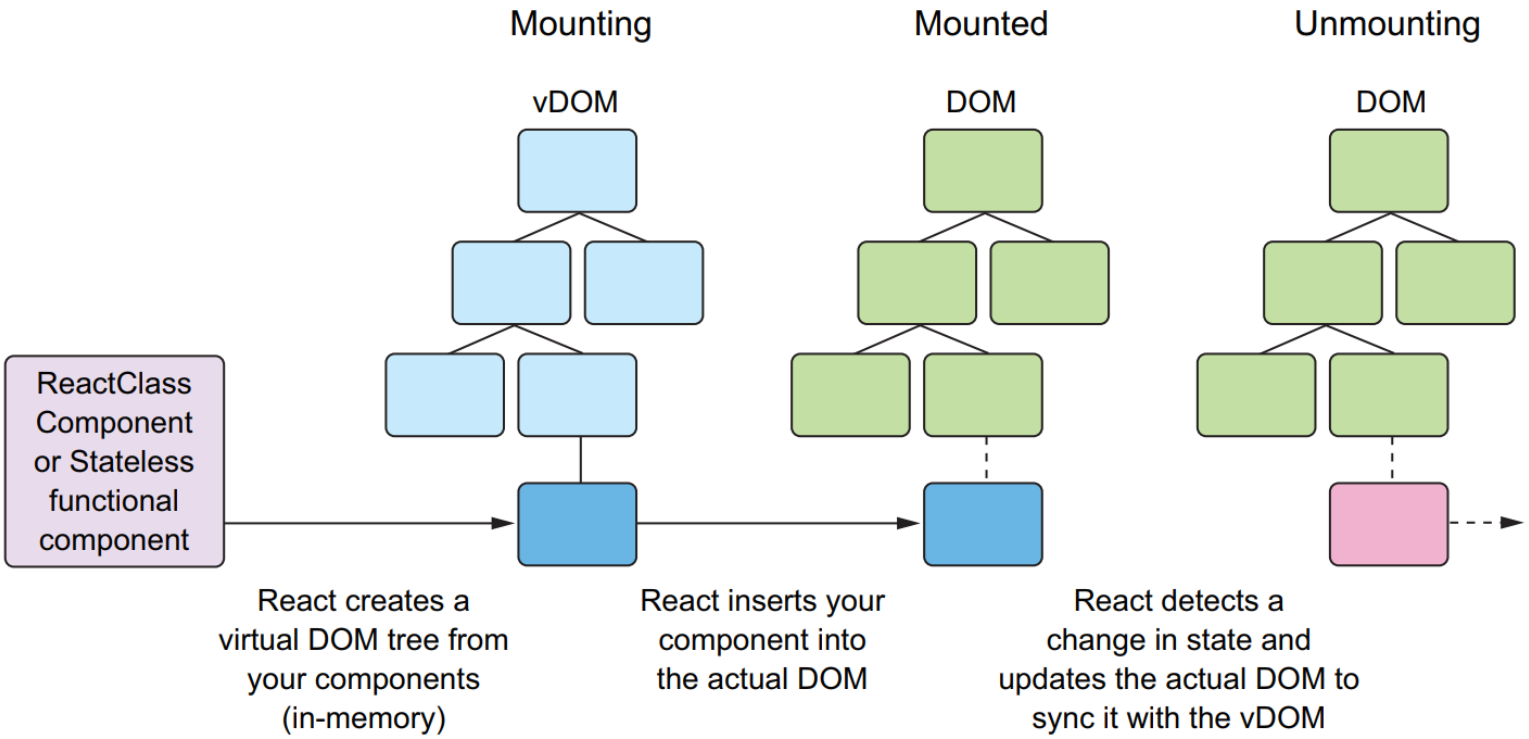
Методи життєвого циклу можна розділити на дві основні групи:

- **Will methods** - Called right before something happens
- **Did methods** - Called right after something happens

Існує також кілька інших методів, які не входять до жодної з цих груп.

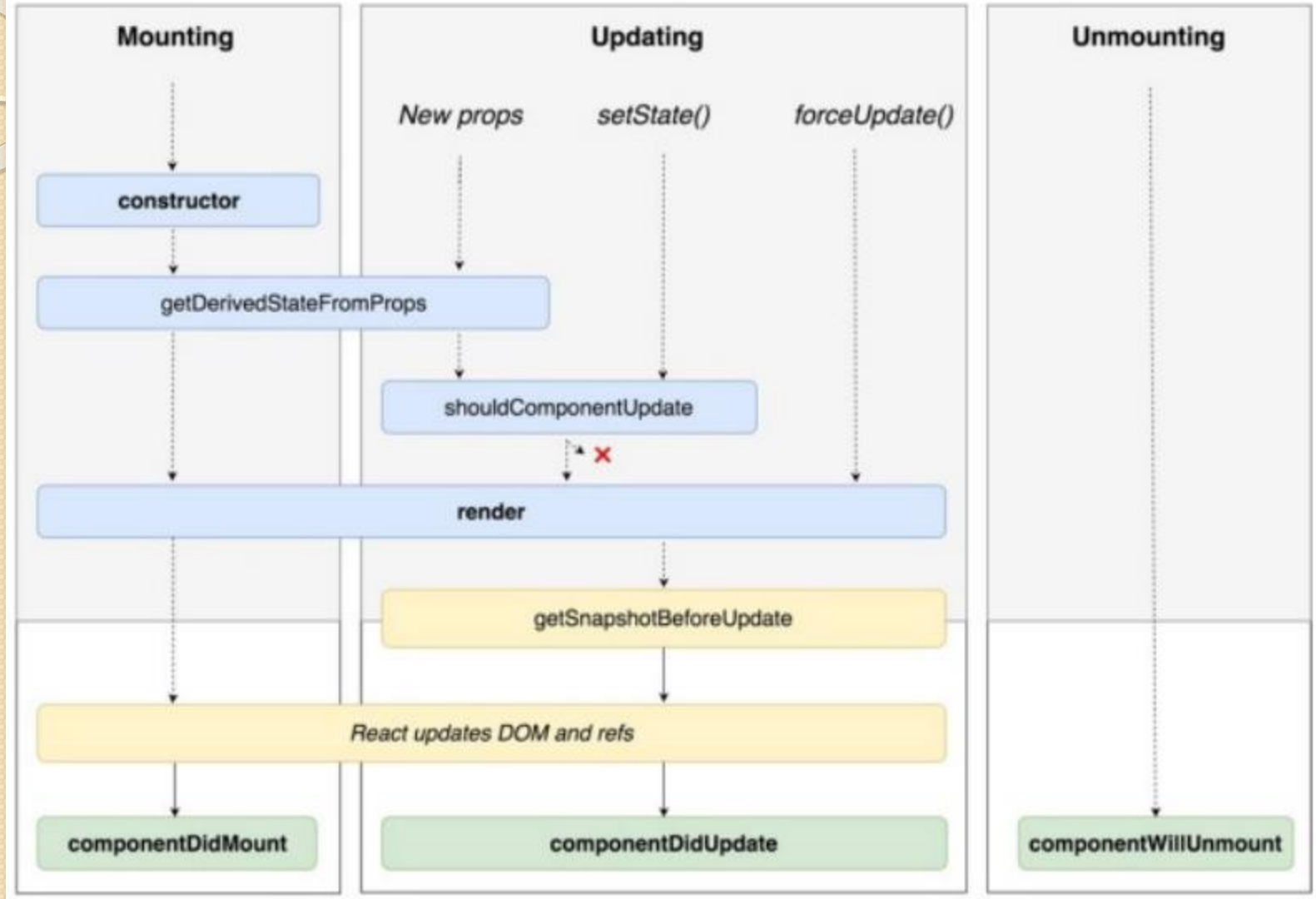
Життєвий цикл компонента ділиться на чотири основні частини:

- **Initialization** — When a component class is being instantiated.
- **Mounting** — A component is being inserted into the DOM.
- **Updating** — A component is being updated with new data via state or props.
- **Unmounting** — A component is being removed from the DOM.



- DEFINITION Mounting is the process of React inserting your components into the real DOM.

Методи життєвого циклу



State in React

Давайте розглянемо приклад компонента, який перемикає стан компонента, щоб відображати привітальне повідомлення при натисканні кнопки (The React Workshop, p.170):

```
class App extends Component {
    constructor(props) {
        super(props);
        this.state = { isActive: false};
    }
    render() {
        const { isActive } = this.state;
        return (
            <div><button onClick={() => {this.setState({ isActive: !isActive
});}}>
                    Say Hello
                </button>
                {isActive && <div>Hello World</div>}
            </div>
        );
    }
}
```

Initializing and Using State

Setting State

Стан можна визначити та ініціалізувати для компонента класу лише наступними способами:

- Як властивість класу Component:

```
class App extends Component {
    state = {
        count: 1
    };
```

- У конструкторі класу:

```
class App extends Component {
    constructor (props) {
        super(props);
        this.state = { count: 1 };
    }
```

setState

Стан можна змінити, використовуючи метод setState() у React. Синтаксис такий:

setState(updater, [callback])

- Перший параметр, Updater, - це значення, яке встановлюється як об’єкт, тоді як другий параметр - це необов’язкова функція зворотного виклику, яка викликається відразу після встановлення значення та повторного відображення компонента (re-rendered).
- Зміна стану завжди призведе до рендерингу компонента, і цим можна керувати, використовуючи метод життєвого циклу React shouldComponentUpdate.

Props in React

Props in React is a way of passing data from parent to child.

```
import React, { Component } from "react";
export class HelloMessage extends Component {
    render() {
        return <h1>Hello {this.props.name}!</h1>;
    }
}
. . . . . .
```

Setting props in our JSX is similar to setting an attribute in XML or HTML.

```
<div>
    <HelloMessage name="John" />
</div>
```

Children Prop

- Вміст у тегах відкриття та закриття (тобто між > та <) компонента в JSX передається як спеціальна властивість, яка називається **children**.
- Давайте модифікуємо попередній компонент, щоб використовувати children prop. Прикладом надсилання повідомлення Hello як children prop є таке:

```
<div>
  <HelloMessage name="John">Hello</HelloMessage>
</div>
```

An example of using the **children prop** is as follows:

```
import React from "react";
export const HelloMessage = props => <h1>{props.children}
{props.name}!</
h1>;
```

Props are Immutable

- React має односпрямований потік даних, що означає, що дані надходять від батьків до нащадка, а ніколи не навпаки.
- Іншим застереженням React є те, що властивості (props) не можуть бути змінені всередині компонента.
- Властивості схожі на параметри функції.
- Якби властивості були змінними, вони могли би викликати нескінченний цикл і витік пам'яті, що було б небажано.
- Оскільки властивості передаються від батьків до дитини, лише батько зможе змінити властивості до того, як вони будуть передані.

Communication between Components

(The React Workshop, p.195)

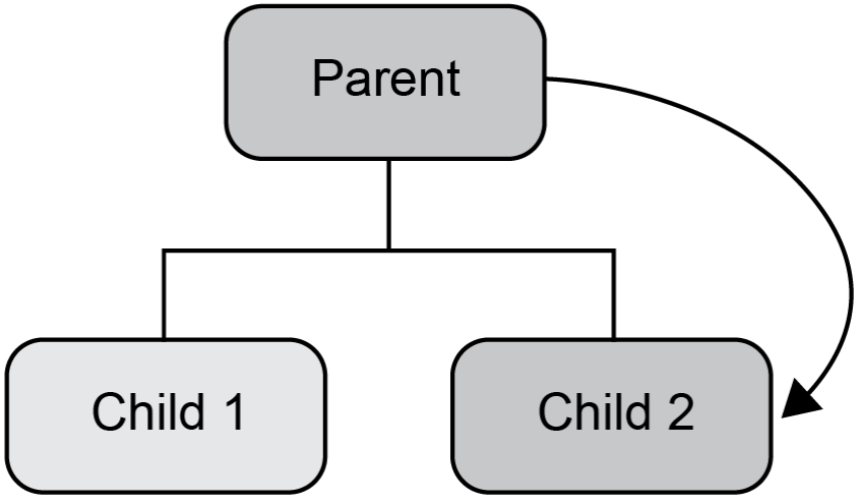
Існує багато способів передачі даних між компонентами. Ми розглянемо чотири основні способи передачі даних:

- Від батьківського до дочірнього компонента.
- Від дитини до батьківського компонента.
- Між будь-якими компонентами, наприклад, компонентами брата та сестри.

Примітка

У ієрархічному порядку React батьківський компонент посилається на компонент, розташований вгорі, тоді як *дочірні* компоненти посилаються на компоненти, розташовані *під батьківським компонентом в ієрархії*.

Passing Data to Direct Child Components



Дерево компонентів

Щоб надіслати дані прямому дочірньому компоненту, ми надаємо значення **props** в батьківському компоненті, яке отримує дочірній компонент.

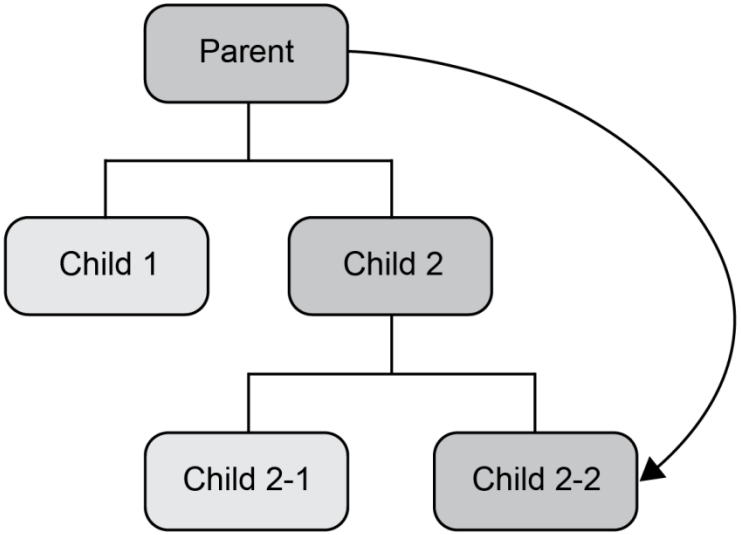
- Parent component

```
class App extends Component {
    render() {
        return <Animal name="Tiger" />;
    }
}
```

- Direct Child Component

```
class Animal extends Component {
    render() {
        return <div>Animal: {this.props.name}</div>;
    }
}
```

Передача даних дочірньому компоненту на кілька рівнів вниз

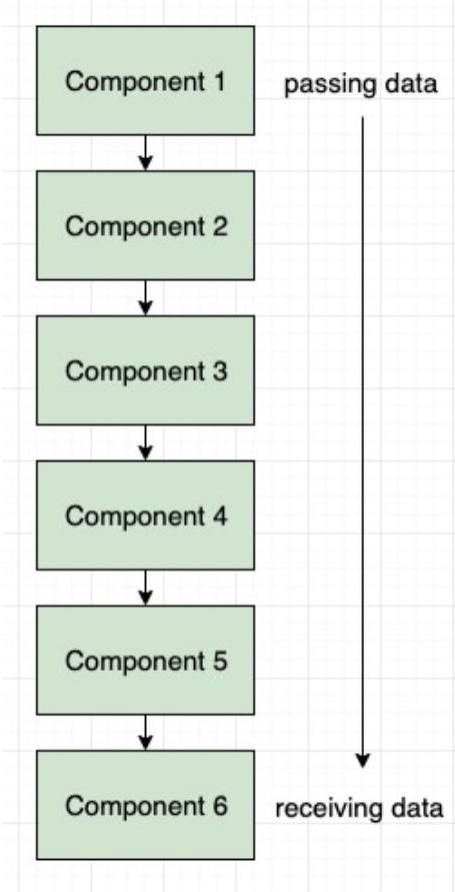


Існує кілька способів надсилання даних до таких дочірніх компонентів:

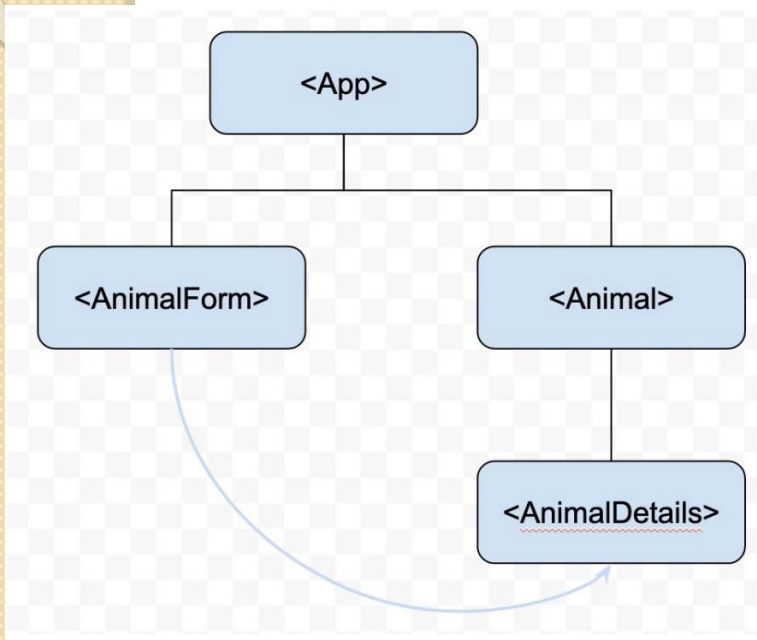
- Props
- React Context API
- Redux

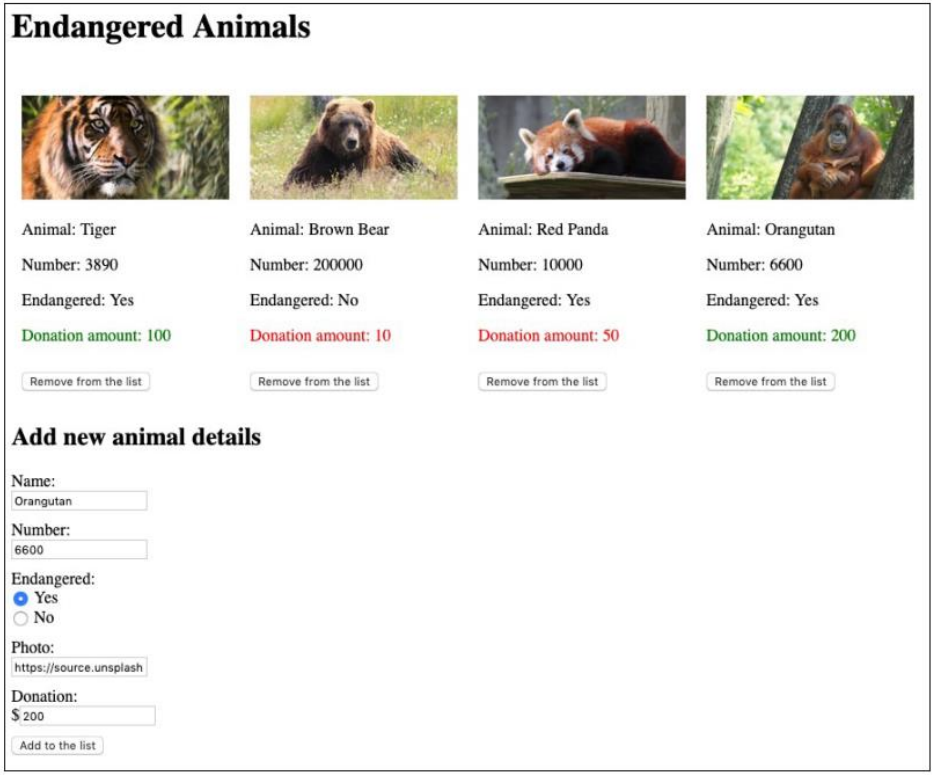
Проблема Prop-drilling

- Prop-drilling стосується процесу передачі значень дочірнім компонентам на кілька рівнів вниз.
- Тут деякі дочірні компоненти посередині можуть насправді не використовувати дані.



Passing Data Between Components at Any Level
(The React Workshop, p.253)





Знайомство з React Router

Так виглядає рендеринг одного компонента:

```
var Home = React.createClass({
   render: function() {
         return (<h1>Welcome to the Home Page</h1>);
   }
});
ReactDOM.render(( <Home /> ),
   document.getElementById('root'));
```

Відмальовування компонента за допомогою React Router

```
...
ReactDOM.render((
   <Router>
        <Route path="/" component={Home} />
   </Router>
), document.getElementById('root'));
```

Компоненти не завжди створюють елементи DOM, іноді вони просто координують роботу внутрішніх компонентів.

В прикладі вище компонент задає правило, згідно якому на сторінці / в елементі root буде відображатися компонент Home

Декілька маршрутів

```
ReactDOM.render((
<Router>
   <Route path="/" component={Home} />
   <Route path="/users" component={Users} />
   <Route path="/widgets" component={Widgets} />
</Router>
 ), document.getElementById('root'));
```

Питання до теми “React.js”

1. Що таке React?
2. Що таке JSX?
3. Для чого потрібен Babel?
4. Що таке компонент в React? Які типи компонентів ви знаєте?
5. Що таке Props? Для яких цілей використовують Props?
6. Що таке State?
7. Які типи методів життєвого циклу в React ви знаєте?
8. Що таке React Hooks?