

BApp Dev Lab 1

Frontend Programming Part 2

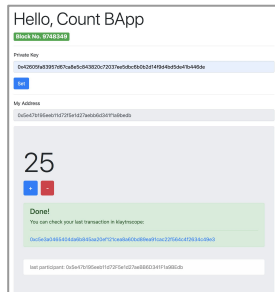
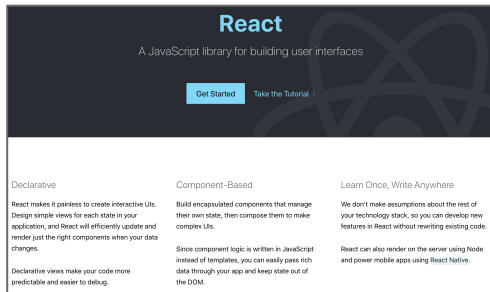
Previously...

리액트 (자바스크립트 라이브러리)

위키백과, 우리 모두의 백과사전.

컴퓨팅에서 리액트(React, React.js 또는 ReactJS)는 자바스크립트 라이브러리의 하나로서^[2] 사용자 인터페이스를 만들기 위해 사용된다. 페이스북과 개별 개발자 및 기업들 공동체에 의해 유지보수된다.^{[3][4][5]}

리액트는 싱글 페이지나 모바일 애플리케이션의 개발 시 토대로 사용될 수 있다. 복잡한 리액트 애플리케이션들은 상태 관리, 라우팅, API와의 통신을 위한 추가 라이브러리의 사용이 일반적인^{[6][7]}로 요구된다.

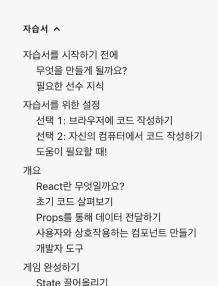


자습서: React 시작하기

이 자습서는 React에 대한 어떤 지식도 가정하지 않습니다.

자습서를 시작하기 전에

우리는 이 자습서에서 작은 게임을 만들겁니다. 게임을 만들고 싶지 않아서 자습서를 건너뛰고 싶을 수 있습니다. 다, 그래도 한번 해보세요! 자습서를 통해 React 앱을 만드는 기본적인 사항들을 배웁니다. 이것을 마스터하면 React를 더 깊게 이해할 수 있습니다.



Hello, Count BApp

Block No. 9756952

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

```
$ npx create-react-app count
$ cd count
$ npm install caver-js
$ rm -f src/*
```



개발 순서

1. 리액트 프로젝트 생성하기
2. 첫번째 컴포넌트 만들기
3. 컴포넌트 추출하기
4. Klaytn과 통신하는 컴포넌트 만들기
5. 비밀키 입력폼 만들기
6. Count 컨트랙트 호출하기
7. Assembly

5. 비밀키 입력폼 만들기

Hello, Count BApp

Block No. 9748349

Private Key

0x42605fa83957d67ca8e5c843820c72037ee5dbc6b0b2d14f9d4bd5de41b446de

Set

`export(key)`

1d27aebb6d341f1a9bedb

25



Done!

You can check your last transaction in klaytnscope:

0xc5e3a0465404da6b845aa20ef121cea8a60bd89ea91cac22f564c4f2634c49e3

last participant: 0x5e47b195eeb11d72F5e1d27aeBB6D341F1a9BEdb

```
import React from 'react';  
import caver from '../klaytn/caver';
```

```
class PrivateKeyInput extends React.Component { ... }
```

```
class KeyAndAddress extends React.Component { ... }
```

```
export default KeyAndAddress
```

5-1. PrivateKeyInput 컴포넌트 목적

- 사용자로부터 비밀키를 입력 받는 목적
 - <input> 태그와 onChange 이벤트 핸들링 사용
- 입력받은 비밀키를 부모 컴포넌트, KeyAndAddress에게 전달
 - 부모 컴포넌트가 자식 컴포넌트에 함수 전달 (function pointer 개념)
 - 입력된 비밀키의 검증은 부모 컴포넌트에서 수행

5-1. PrivateKeyInput 컴포넌트 1/2

```
// import statements
class PrivateKeyInput extends React.Component {
  constructor(props) {
    super(props); // assume parent component has passed a function `export(key)`
    this.state = {
      privateKey: ''
    };
  }

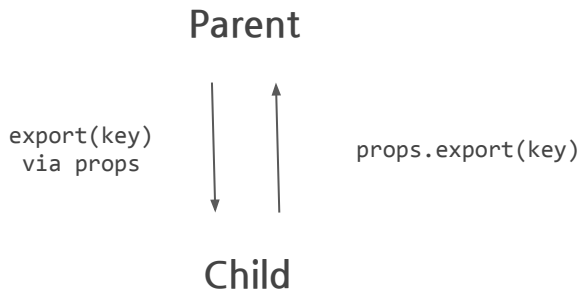
  handleChange = (e) => {
    this.setState({
      privateKey: e.target.value
    });
  }

  // more on next slide
}
// omitted for brevity
```

부모 컴포넌트가 자식
컴포넌트에게 함수 전달

입력필드에 연결할
이벤트 핸들러

setState가
render를 호출



5-1. PrivateKeyInput 컴포넌트 2/2

```
// import statements
class PrivateKeyInput extends React.Component {
  // things from previous slide
  render() {
    const { privateKey } = this.state;
    return (<div><form><div>
      <label for='inputKey'>Private Key</label>
      <input type='text' placeholder="your private key starting with 0x"
        onChange={this.handleChange} id="inputKey" />
      </div>
      <button onClick={() => this.props.export(privateKey)}>Set</button>
    </form></div>);
  }
  // more on next slide
}
// omitted for brevity
```

state에 저장된 (최신)값

입력필드에 변경이 발생할 경우 onChange에 등록된 함수를 실행

Parent가 전달한 함수를 실행

5-2. KeyAndAddress 컴포넌트 목적


- 비밀키를 받아 주소를 추출, 표시하는 것이 목적
 - PrivateKeyInput 컴포넌트로부터 비밀키를 공급받기 위해 함수를 전달
 - 공급받은 비밀키는 검증
 - caver-js를 사용하여 비밀키 → 주소 추출

5-2. KeyAndAddress 컴포넌트 1/4

```
// import statements  
class KeyAndAddress extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      privateKey: '',  
      address: ''  
    };  
  }  
}
```

```
  reloadAddress = (key) => { ... }
```

PrivateKeyInput 컴포넌트에
전달할 함수



```
  render() { ... }  
}  
// omitted for brevity
```

5-2. KeyAndAddress 2/4

```
// import statements
class KeyAndAddress extends React.Component {
  // things from previous slide
  reloadAddress = (key) => {
    let account;
    try {
      account = caver.klay.accounts.privateKeyToAccount(key);
    } catch (e) {
      console.error(e);
    }
    // more on next slide
  }
}
// omitted for brevity
```

try { ... } catch (e) { ... }:

try 블록에서 실행하던 중 오류가 발생하면 catch 블록을 실행 (오류 객체를 e로 전달)

account 변수에 `caver.klay.accounts.privateKeyToAccount`를 실행한 결과를 저장

5-2. KeyAndAddress 3/4

```
// import statements
class KeyAndAddress extends React.Component {
  // things from previous slide
  reloadAddress = (key) => {
    // continued from the previous slide
    if (account) {
      this.setState({ privateKey: key, address: account.address });
      this.props.propagateKey(key);
    } else {
      this.setState({ privateKey: '', address: '' });
      this.props.propagateKey(false);
    }
  }
}
// omitted for brevity
```

try 블록이 성공적으로 실행되었다면 account 변수 비밀키로 추출한 어카운트 정보가 저장되어 있을 것. 따라서 if 블록이 실행됨. state를 변경 후 (re-render 실행) 키값을 props.propagateKey를 실행하여 부모에게 전달.

try 블록이 실패할 경우 실행될 else 블록. 올바른 키가 입력되지 않았으므로 propagateKey에 false를 전달.

5-2. KeyAndAddress 4/4

```
// import statements
class KeyAndAddress extends React.Component {
  // things from previous slide
  render() {
    const { address } = this.state;
    return (<div>
      <PrivateKeyInput export={this.reloadAddress} /><hr />
      <form><div>
        <label for='addressField'>My Address</label>
        <input id='addressField' type='text' disabled value={address} />
      </div></form></div>
    );
  }
}
// omitted for brevity
```

PrivateKeyInput 컴포넌트에 export 라는 이름으로 reloadAddress 함수 전달

State에 저장된 address 값을 표시

6. Count 컴포넌트

Hello, Count BApp

Block No. 9748349

Private Key

0x42605fa83957d67ca8e5c843820c72037ee5dbc6b0b2d14f9d4bd5de41b446de

Set

My Address

0x5e47b195eeb11d72f5e1d27aebb6d341f1a9bedb

25

+ -

Done!

You can check your last transaction in klaytnscope:

0xc5e3a0465404da6b845aa20ef121cea8a60bd89ea91cac22f564c4f2634c49e3

last participant: 0x5e47b195eeb11d72f5e1d27aebb6d341f1a9BEDb

INDIRECTLY

setPrivateKey(key)

```
import React from 'react'
import caver from "../klaytn/caver"
const ABI = require("../assets/abi.json");
class Count extends React.Component {
  constructor() { ... }
  setPrivateKey(key) { ... }
  callPlus = () => { ... }
  callMinus = () => { ... }
  getCount = async () => { ... }
  componentDidMount() { ... }
  componentWillUnmount() { ... }
  render() { ... }
}
export default Count
```

6. Count 컴포넌트 목적

- 컨트랙트 함수를 호출하여 값을 불러오거나 값을 변경하는 것
 - 컨트랙트는 이미 배포되어 있다고 가정
 - 컨트랙트 ABI 또한 이미 준비되어 있다고 가정
- KeyAndAddress 컴포넌트로부터 사용자 비밀키를 받아 TX 생성 및 전송
 - KeyAndAddress → App으로 키 전달
 - App → Count로 키 전달 (ref 사용)
- Lifecycle을 사용하여 주기적인 컨트랙트 함수 호출 구현

```
import React from 'react'
import caver from "../klaytn/caver"
const ABI = require("../assets/abi.json");
class Count extends React.Component {
  constructor() { ... }
  setPrivateKey(key) { ... }
  callPlus = () => { ... }
  callMinus = () => { ... }
  getCount = async () => { ... }
  componentDidMount() { ... }
  componentWillUnmount() { ... }
  render() { ... }
}
export default Count
```

6. Count 컴포넌트 1/7

```
// import statements and ABI
```

```
class Count extends React.Component {
```

```
  constructor() {
```

```
    super();
```

```
    // assume .env file contains REACT_APP_CONTRACT_ADDRESS variable
```

```
    this.countContract = process.env.REACT_APP_CONTRACT_ADDRESS && ABI &&
```

```
      new caver.klay.Contract(ABI, process.env.REACT_APP_CONTRACT_ADDRESS);
```

```
    this.state = { count: '', lastParticipant: '', privateKey: '' };
```

```
    this.wallet = caver.klay.accounts.wallet;
```

```
  }
```

```
  // more on next slide
```

```
}
```

```
export default Count
```

컨트랙트 객체 생성

- 환경변수(.env) 파일에 저장된 컨트랙트 주소를 사용
- import한 ABI를 사용

6. Count 컴포넌트 2/7

```
class Count extends React.Component {  
  // exposed function allowing anyone with the right reference can call this function  
  setPrivateKey(key) {  
    this.wallet.clear();  
    if (key) {  
      this.wallet.add(key);  
    }  
    this.setState({  
      privateKey: key  
    });  
  }  
  
  // more on next slide  
}  
export default Count
```

외부에 공개할 함수

- 공개된 함수는 reference로 접근 가능
- 올바른 키가 들어올 때만 월렛에 키를 추가

6. Count 컴포넌트 3/7

```
class Count extends React.Component {
```

```
  callPlus = () => {
```

```
    const walletInstance = this.wallet[0];
```

```
    if (!walletInstance) return;
```

```
    this.countContract.methods.plus().send({
```

```
      from: walletInstance.address,
```

```
      gas: '200000',
```

```
    })
```

```
    .once('transactionHash', (txHash) => { console.log(`Sending a transaction... ('plus') txHash: ${txHash}`); })
```

```
    .once('receipt', (receipt) => {
```

```
      console.log(`Received receipt!`, receipt);
```

```
      this.setState({ txHash: receipt.transactionHash, });
```

```
    })
```

```
    .once('error', (error) => { alert(error.message); });
```

```
  }
```

```
  // more on next slide
```

```
}
```

```
export default Count
```

월렛에 적어도 하나의 키가 설정되어 있을때만 실행

plus 함수를 실행하는 TX를 생성 및 서명
(from과 gas 필드를 반드시 기입)

Receipt를 받음 = TX가 최근 블록에 포함됨

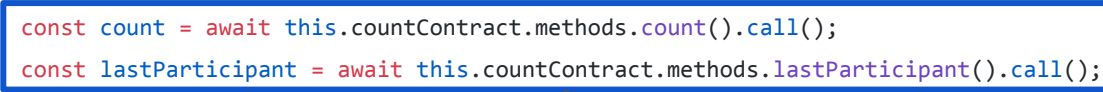
6. Count 컴포넌트 4/7

```
class Count extends React.Component {  
  callMinus = () => {  
    const walletInstance = this.wallet[0];  
    if (!walletInstance) return;  
    this.countContract.methods.minus().send({  
      from: walletInstance.address,  
      gas: '200000',  
    })  
    .once('transactionHash', (txHash) => { console.log(`Sending a transaction... (minus) txHash: ${txHash}`); })  
    .once('receipt', (receipt) => {  
      console.log(`Received receipt!`, receipt);  
      this.setState({ txHash: receipt.transactionHash, });  
    })  
    .once('error', (error) => { alert(error.message); });  
  }  
  // more on next slide  
}  
  
export default Count
```

plus → minus로 변경된 것 외 변경 없음

6. Count 컴포넌트 5/7

```
class Count extends React.Component {  
  getCount = async () => {  
    const count = await this.countContract.methods.count().call();  
    const lastParticipant = await this.countContract.methods.lastParticipant().call();  
    this.setState({  
      count,  
      lastParticipant,  
    });  
  }  
  
  // more on next slide  
}  
  
export default Count
```



컨트랙트 state를 읽기 위해 RPC 호출 수행

- await를 사용하여 동기화

6. Count 컴포넌트 6/7

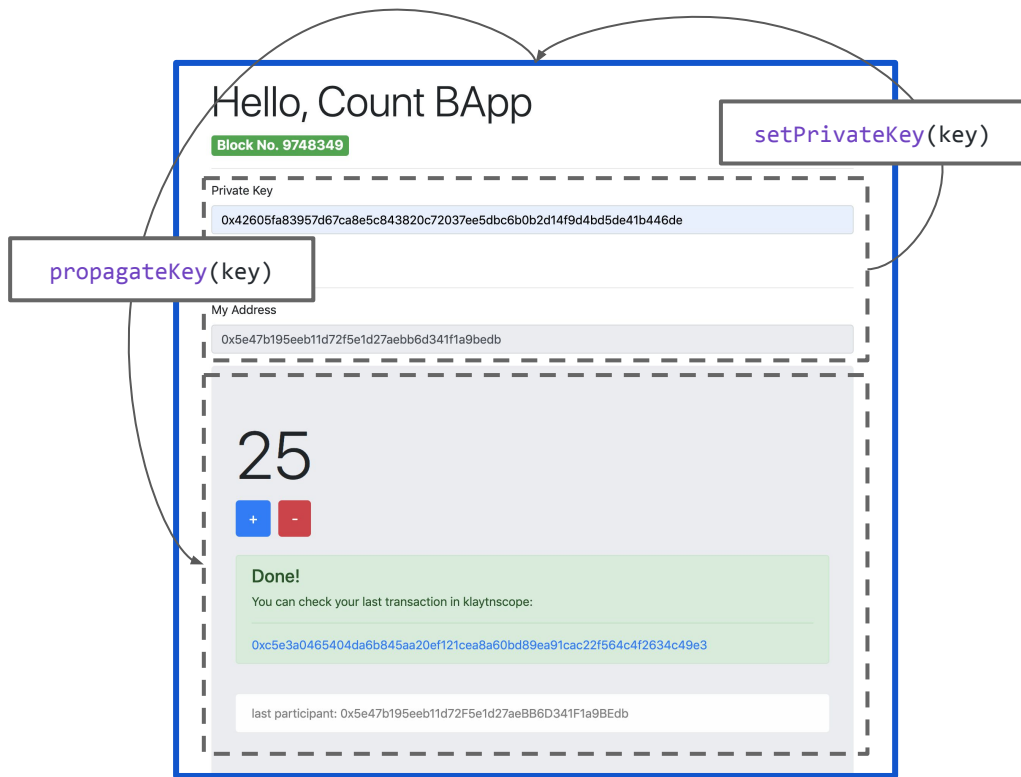
```
class Count extends React.Component {  
  intervalId = null  
  
  componentDidMount() {  
    this.intervalId = setInterval(this.getCount, 1000)  
  }  
  
  componentWillUnmount() {  
    clearInterval(this.intervalId)  
  }  
  
  // more on next slide  
}  
export default Count
```

BlockNumber 컴포넌트에서 구현한 것과 동일하게 컴포넌트 lifecycle을 고려하여 1초마다 getCount를 실행하도록 setInterval 설정

6. Count 컴포넌트 7/7

```
class Count extends React.Component {
  render() {
    const { lastParticipant, count, txHash, privateKey } = this.state;
    return (<div>
      {!privateKey && (<div>User must provide a private key to start</div>)}
      <div><h1>{count}</h1></div>
      {privateKey && (<div><div>
        <span><button onClick={this.callPlus}>+</button></span>
        <span><button onClick={this.callMinus} disabled={count === 0}>-</button></span></div><br />
        {txHash && (<div><h4>Done!</h4>
          <p>You can check your last transaction in klaytnscope:</p><hr />
          <a target="_blank" rel="noopener noreferrer" href={`https://baobab.scope.klaytn.com/tx/${txHash}`}>open</a>
        </div>)}
      </div>)}<br />
      {Number(lastParticipant) !== 0 && (<div>last participant: {lastParticipant}</div>)}</div>);
  }
}
```

7. Assembly (App 컴포넌트)



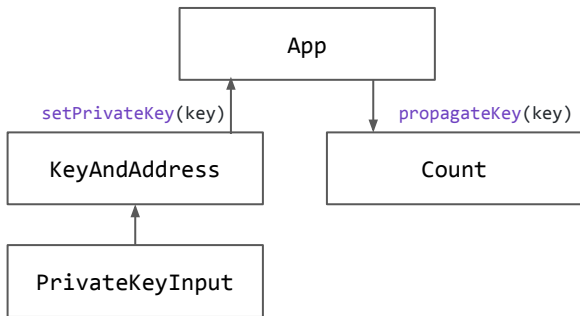
```
import React from 'react';
import ReactDOM from 'react-dom';
import BlockNumber from './components/BlockNumber'
import Count from './components/Count'
import KeyAndAddress from './components/KeyAndAddress'
```

```
class App extends React.Component {
  constructor(props) { ... }
  propagateKey = (key) => { ... };
  render() { ... }
}
```

```
ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

7. App 컴포넌트 목적

- 지금까지 만든 컴포넌트들을 한 곳에 묶는 역할 수행
 - BlockNumber, KeyAndAddress, Count ...
- 컴포넌트 간 정보전달을 돕는 역할 수행



- DOM에 컴포넌트 렌더링을 담당

```
import React from 'react';
import ReactDOM from 'react-dom';
import BlockNumber from './components/BlockNumber'
import Count from './components/Count'
import KeyAndAddress from './components/KeyAndAddress'
```

```
class App extends React.Component {
  constructor(props) { ... }
  propagateKey = (key) => { ... };
  render() { ... }
}
```

```
ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

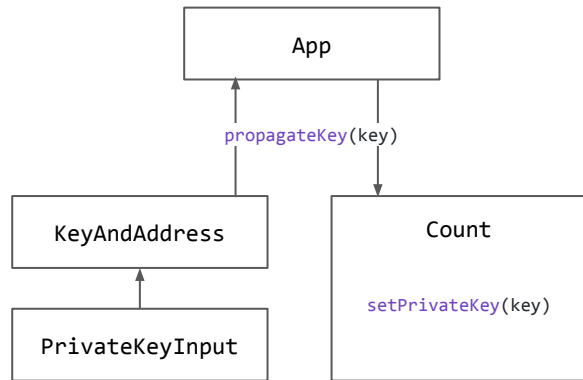
7. App 컴포넌트 1/2

```
// import statements
class App extends React.Component {
  constructor(props) {
    super(props);
    this.countRef = React.createRef();
  }

  // note that the following definition must be declared as a variable
  propagateKey = (key) => {
    this.countRef.current.setPrivateKey(key);
  };

  // more on next slide
}
// DOM render omitted
```

자식 컴포넌트의 함수를 실행하기 위해
컴포넌트에 부여할 레퍼런스 생성

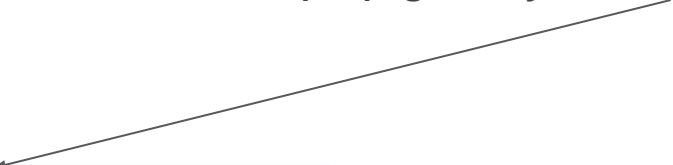


자식 컴포넌트의 함수 `setPrivateKey`를 실행

7. App 컴포넌트 2/2

```
// import statements
class App extends React.Component {
  // things from previous slide
  render() {
    return (
      <div className="App">
        <Greeting />
        <BlockNumber /><hr />
        <KeyAndAddress propagateKey={this.propagateKey} />
        <Count ref={this.countRef} />
      </div>
    );
  }
  // more on next slide
}
```

KeyAndAddress 컴포넌트에 prop으로
propagateKey 함수를 전달



Count 컴포넌트를 this.countRef 레퍼런스로
연결 → Count 컴포넌트에 선언된 함수들을
this.countRef.current로 접근 가능

