Web Programming

React programming

Cookie & Session

Cookie

- 쿠키는 서버가 사용자의 웹 브라우저에 저장하는 작은 텍스트 파일이다.
- 쿠키는 이름, 값, 만료일, 경로, 도메인 등의 정보를 포함하며, 이 정보는 사용자가 웹 사이트를 다시 방문할 때마다 서버에 전송된다.
- 쿠키를 사용하면 사용자의 선호 설정, 로그인 정보 등을 기억하거나 장바구니와 같은 상태 정보를 저장할 수 있다.
- 쿠키는 사용자가 이전에 웹사이트에서 입력했던 정보(아이디, 언어 설정 등)를 기억하게 해 주어 사용자 경험을 향상시키는데 주로 사용된다.
- 장점
  - ✓ 서버가 아닌 클라이언트에서 데이터를 저장하기 때문에 서버의 부하를 줄일 수 있다.
  - ✓ 사용자 별로 개인화된 정보를 저장하고 이를 기반으로 사용자 경험을 개선할 수 있다.
- 단점
  - ✓ 보안성이 낮으며, 클라이언트 측에서 쿠키를 삭제하거나 변경할 수 있기 때문에 민감한 데이터를 저장하기에 적합하지 않다.
  - ✓ 사용자의 브라우저에 저장공간이 한정적이기 대문에 큰 데이터를 저장하기 어렵다.

Cookie & Session

Session

- 웹에서 세션은 클라이언트와 서버 간 네트워크 연결이 지속되는 동안 사용자의 상태를 유지하기 위한 방법이다.
- 즉, 사용자가 웹 사이트에 로그인할 때 세션을 시작하고, 로그아웃하거나 일정 시간 동안 활동이 없을 때 세션을 종료한다.
- 웹은 기본적으로 상태를 유지하지 않는(stateless)통신 방식이기 때문에, 세션과 같은 기술이 필요하다.
- 세션을 통해 서버는 사용자가 페이지를 이동하더라도 그 사용자의 상태를 유지할 수 있다.
- 세션 정보는 서버에 저장되며, 각 세션은 고유한 세션ID를 가지고 있다. 사용자가 로그인하면 서버는 세션을 생성하고, 그 세션ID를 사용자의 브라우저에 쿠키를 통해 전달한다.
- 사용자의 브라우저는 이후 요청마다 이 세션 ID를 서버에 보내, 사용자를 식별한다.

Cookie & Session

Session

- 장점
  - 상태 유지 : 사용자의 정보와 설정, 로그인 상태 등을 유지할 수 있다. 이는 웹 서비스에서 중요한 기능으로, 사용자의 편의를 크게 높인다.
  - 보안 : 세션 정보는 서버에 저장되므로, 클라이언트에서 직접적으로 접근하거나 수정하는 것이 어렵습니다.
  - 대용량 데이터 : 서버에 저장되므로 대용량의 데이터를 저장할 수 있다.
- 단점
  - 서버 부하 : 세션 정보는 서버에 저장되므로, 많은 사용자가 접속하면 서버의 메모리를 많이 차지하게 된다. 이로 인해 성능이 저하될 수 있으며, 스케일링이 어려울 수 있다.
  - 클라이언트 이동 : 사용자가 브라우저를 변경하거나 다른 기기로 이동하면 세션을 유지할 수 없다. 세션은 서버와 특정 클라이언트 사이의 연결을 유지하기 때문이다.
  - 세션 하이재킹 : 세션ID가 공격자에게 노출되면, 공격자가 세션ID를 이용해 사용자를 가장할 수 있다. 이를 세션 하이재킹이라고 한다. 이를 방지하기 위해 HTTPS등의 기술을 사용해 세션ID를 안전하게 전송해야 한다.

Cookie & Session

Cookie와 Session 비교

- 공통점
  - Cookie와 Session은 웹 사이트가 사용자의 브라우저를 통해 사용자의 상태를 기억하고 추적하는 데 사용되는 기술이다.
  - 이들은 HTTP의 stateless한 특성을 보완하기 위해 사용되며, 웹 사이트와 사용자의 상호작용을 개선하는데 도움을 준다.
- 주요 차이점

	Cookie	Session
저장 위치	클라이언트(브라우저)	서버
저장 형식	텍스트 형식	객체 형태
생명주기	직접 삭제 전까지 유지	서버와의 연결 종료까지

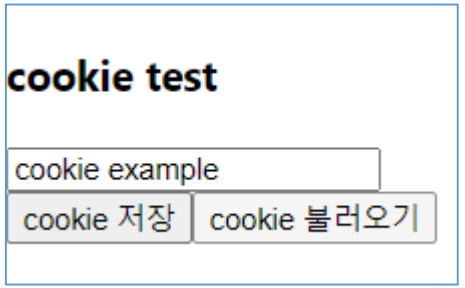
Cookie

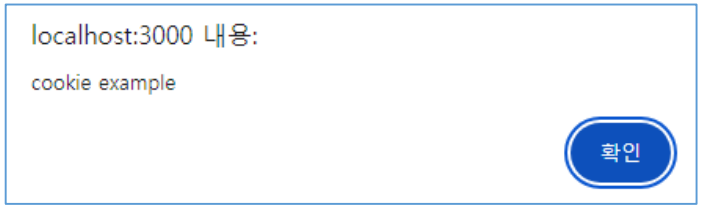
Cookie example

- cookie를 사용하기 위해서 npm install react-cookie를 실행

```
import React, {useState} from 'react';
import {useCookies} from 'react-cookie';

function App() {

  const [id, setId] = useState("");

  // 기존의 cookie를 저장
  const [cookies, setCookies] = useCookies(["id"]);

  const changeId = (e) => setId(e.target.value);

  // cookid 저장하기
  const cookieSave = () => {
    setCookies("id", id, { path:'/' }); // cookie명, 값, 경로
  }

  // cookie 불러오기
  const cookieLoad = () => {
    alert(cookies.id);
  }
```

```
  return (
    <div>
      <h3>cookie test</h3>

      <input value={id} onChange={changeId} /><br/>

      <button onClick={cookieSave}>cookie 저장</button>
      <button onClick={cookieLoad}>cookie 불러오기</button>
    </div>
  );
}

export default App;
```





F12 - 애플리케이션 - 저장용량 - 쿠키

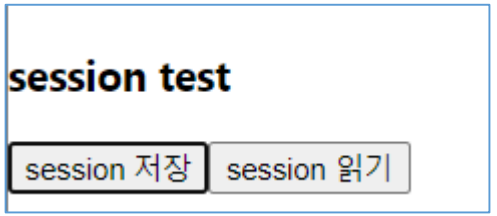
Session

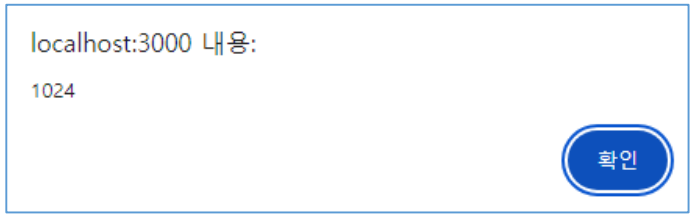
Session example1

- session을 사용하기 위해서 npm install react-session-api 를 실행

```
import React from 'react';
import Session from 'react-session-api';

function App(){

  // session 저장하는 함수 정의
  function save(){
    let count = 1024;
    Session.set("counter", count); // Session.set("세션명", 값);

  }

  // session 불러오는 함수 정의
  function load(){
    let c = Session.get("counter");
    alert(c); // 1024
  }
```

```
 return(
    <div>
      <h3>session test</h3>

      <button onClick={() => save()}>session 저장</button>

      <button onClick={() => load()}>session 읽기</button>

    </div>
  )

}

export default App;
```



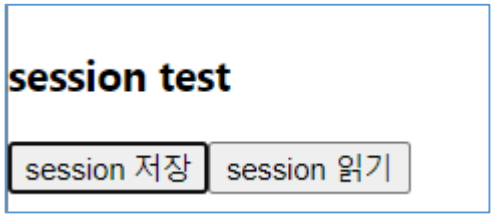


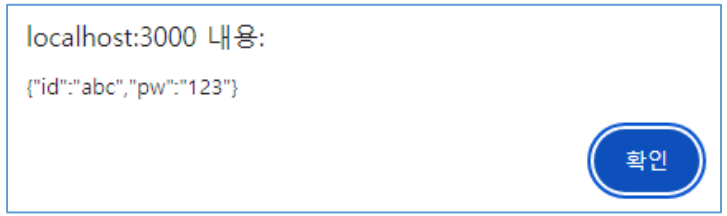
Session

Session example2

```
import React from 'react';
import Session from 'react-session-api';

function App(){

  // session 저장하는 함수 정의
  function save(){
    let member = { "id":"abc", "pw":"123"};
    Session.set("member", member); // Session.set("세션명", 값);

  }

  // session 불러오는 함수 정의
  function load(){
    let member = Session.get("member");
    // alert(member);
    alert(JSON.stringify(member));
  }
```

```
  return(
    <div>
      <h3>session test</h3>

      <button onClick={() => save()}>session 저장</button>

      <button onClick={() => load()}>session 읽기</button>

    </div>
  )

}

export default App;
```



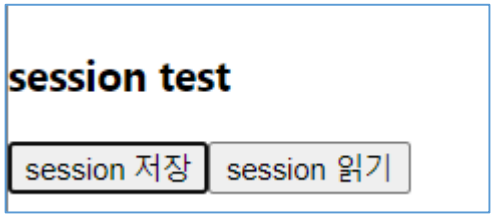


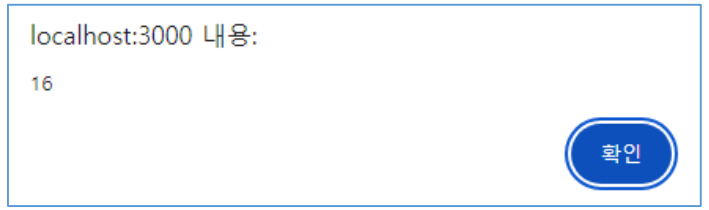
Session

Session example3

```
import React from 'react';
import Session from 'react-session-api';

function App(){

  // session 저장하는 함수 정의
  function save(){
    let jsonData = [ { "name":"홍길동", "age":24}, { "name":"성춘향", "age":16} ];
    Session.set("jsonData", jsonData); // Session.set("세션명", 값);

  }

  // session 불러오는 함수 정의
  function load(){
    let jsonData = Session.get("jsonData");
    // alert(JSON.stringify(jsonData));
    alert(jsonData[1].age); // 16
  }
```

```
  return(
    <div>
      <h3>session test</h3>

      <button onClick={() => save()}>session 저장</button>

      <button onClick={() => load()}>session 읽기</button>

    </div>
  )

}

export default App;
```





Login

Login

- express를 비롯한 Node.js의 모듈들과, React로 구현한 로그인, 회원가입 예제
- 필요 모듈
    1. mysql2(mysql2 대신 mysql 모듈을 사용 무방)
    2. express(서버)
    3. express-session(세션)
    4. express-mysql-session(세션)
    5. body-parser(편의를 위한 것으로, 소스코드에서 req.body 부분만 수정하시고 사용해도 됨)
    6. Bcrypt(비밀번호를 해시알고리즘으로 암호화해서 저장하는데 필요한 모듈)

```
npm install express
npm install express-mysql-session
npm install bcrypt npm
npm install express-session
npm install mysql2
install body-parser
```

Login

Login

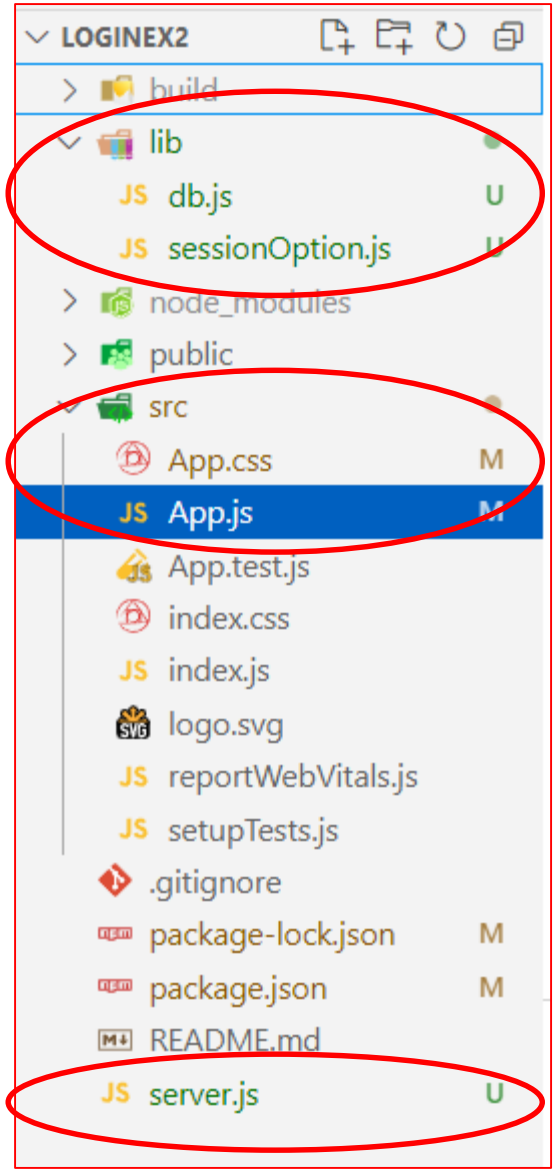
- express를 비롯한 Node.js의 모듈들과, React로 구현한 로그인, 회원가입 예제
- 필요 모듈
    1. mysql2(mysql2 대신 mysql 모듈을 사용 무방)
    2. express(서버)
    3. express-session(세션)
    4. express-mysql-session(세션)
    5. body-parser(편의를 위한 것으로, 소스코드에서 req.body 부분만 수정하시고 사용해도 됨)
    6. Bcrypt(비밀번호를 해시알고리즘으로 암호화해서 저장하는데 필요한 모듈)

```
npm install express
npm install express-mysql-session
npm install bcrypt npm
npm install express-session
npm install mysql2
install body-parser
```

Login

파일 생성 및 정리

- npx create-react-app . 명령어를 입력해서 리액트를 설치
- 프로젝트 최상단 폴더에 server.js와 lib 폴더 생성
- /lib 폴더 내에는 db.js와 sessionOption.js 파일 생성
- /src 폴더 내에는 App.css와 App.js 파일 수정



Login

server.js

- express로 서버를 실행시키는 server.js 파일이며 이름은 main.js 등 원하는 대로 작성
- 페이지의 root에 접속했을 때 react로 작성된 build/index.html을 보여줌
- 로그인과 회원가입을 위해 이곳에서 mysql 데이터베이스에 접근해 로그인 과정을 처리하며, 회원가입 과정도 처리
- 세션은 express-mysql-session 모듈을 이용하여 MySQL 데이터베이스에 저장하는 방식으로 구현

Login

server.js

```
const express = require('express')
const session = require('express-session')
const path = require('path');
const app = express()
const port = 3001

const db = require('./lib/db');
const sessionOption = require('./lib/sessionOption');
const bodyParser = require("body-parser");
const bcrypt = require('bcrypt');

app.use(express.static(path.join(__dirname, '/build')));
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

var MySQLStore = require('express-mysql-session')(session);
var sessionStore = new MySQLStore(sessionOption);
app.use(session({
        key: 'session_cookie_name',
    secret: '~',
        store: sessionStore,
        resave: false,
        saveUninitialized: false
}))
```

```
app.get('/', (req, res) => {
    req.sendFile(path.join(__dirname, '/build/index.html'));
})

app.get('/authcheck', (req, res) => {
    const sendData = { isLogin: "" };
    if (req.session.is_logined) {
        sendData.isLogin = "True"
    } else {
        sendData.isLogin = "False"
    }
    res.send(sendData);
})

app.get('/logout', function (req, res) {
    req.session.destroy(function (err) {
        res.redirect('/');
    });
});

app.post("/login", (req, res) => { // 데이터 받아서 결과 전송
    const username = req.body.userId;
    const password = req.body.userPassword;
    const sendData = { isLogin: "" };
```

Login

server.js

```
if (username && password) {              // id와 pw가 입력되었는지 확인
    db.query('SELECT * FROM usertable WHERE username = ?', [username], function (error, results, fields) {
        if (error) throw error;
        if (results.length > 0) {       // db에서의 반환값이 있다 = 일치하는 아이디가 있다.

            bcrypt.compare(password , results[0].password, (err, result) => {    // 입력된 비밀번호가 해시된 저장값과 같은 값인지 비교

                if (result === true) {                  // 비밀번호가 일치하면
                    req.session.is_logined = true;      // 세션 정보 갱신
                    req.session.nickname = username;
                    req.session.save(function () {
                        sendData.isLogin = "True"
                        res.send(sendData);
                    });
                    // db.query(`INSERT INTO logTable (created, username, action, command, actiondetail) VALUES (NOW(), ?, 'login' , ?, ?)`
                    //      , [req.session.nickname, '-', `React 로그인 테스트`], function (error, result) { });
                }

                else{                                   // 비밀번호가 다른 경우
                    sendData.isLogin = "로그인 정보가 일치하지 않습니다."
                    res.send(sendData);
                }
```

Login

server.js

```
 });
    } else {            // 아이디, 비밀번호 중 입력되지 않은 값이 있는 경우
        sendData.isLogin = "아이디와 비밀번호를 입력하세요!"
        res.send(sendData);
    }
});

app.post("/signin", (req, res) => {  // 데이터 받아서 결과 전송
    const username = req.body.userId;
    const password = req.body.userPassword;
    const password2 = req.body.userPassword2;

    const sendData = { isSuccess: "" };

    if (username && password && password2) {
        db.query('SELECT * FROM userTable WHERE username = ?', [username], function(error, results, fields) {
       // DB에 같은 이름의 회원아이디가 있는지 확인
 if (error) throw error;
            if (results.length <= 0 && password === password2) {
       // DB에 같은 이름의 회원아이디가 없고, 비밀번호가 올바르게 입력된 경우
                const hasedPassword = bcrypt.hashSync(password, 10);    // 입력된 비밀번호를 해시한 값
                db.query('INSERT INTO usertable (username, password) VALUES(?,?)', [username, hasedPassword], function (error, data) {
```

Login

server.js

```
if (error) throw error;
                    req.session.save(function () {
                        sendData.isSuccess = "True"
                        res.send(sendData);
                    });
                });
            } else if (password !== password2) {                    // 비밀번호가 올바르게 입력되지 않은 경우
                sendData.isSuccess = "입력된 비밀번호가 서로 다릅니다."
                res.send(sendData);
            }
            else {                                                  // DB에 같은 이름의 회원아이디가 있는 경우
                sendData.isSuccess = "이미 존재하는 아이디 입니다!"
                res.send(sendData);
            }
        });
    } else {
        sendData.isSuccess = "아이디와 비밀번호를 입력하세요!"
        res.send(sendData);
    }

});
```

Login

App.js

- 프론트엔드 영역에 해당하는 App.js
- /src 폴더 안에 파일로, 리액트를 이용하여 작성
- fetch 함수로 Node.js 서버와 통신해서 로그인과 회원가입을 진행
- mode라는 이름의 state를 이용해서 로그인/회원가입/로그인 성공 페이지 전환
- 처음 페이지가 로드될 때, 로그인 여부를 체크한 다음, 로그인되어 있으면 mode를 'WELCOME'으로, 아니면 'LOGIN'으로 설정
- 로그아웃은 /logout 페이지로 이동하는 방식으로 구현

Login

App.js

```
import './App.css';
import { useState } from 'react';
import { useEffect } from 'react';

function Login(props) {
  const [id, setId] = useState("");
  const [password, setPassword] = useState("");

  return <>
    <h2>로그인</h2>

    <div className="form">
      <p><input className="login" type="text" name="username" placeholder="아이디" onChange={event => {
        setId(event.target.value);
      }} /></p>
      <p><input className="login" type="password" name="pwd" placeholder="비밀번호" onChange={event => {
        setPassword(event.target.value);
      }} /></p>

      <p><input className="btn" type="submit" value="로그인" onClick={() => {
        const userData = {
          userId: id,
          userPassword: password,
        };
```

Login

App.js

```
fetch("http://localhost:3001/login", { //auth 주소에서 받을 예정
          method: "post", // method :통신방법
          headers: {       // headers: API 응답에 대한 정보를 담음
            "content-type": "application/json",
          },
          body: JSON.stringify(userData), //userData라는 객체를 보냄
        })
          .then((res) => res.json())
          .then((json) => {
            if(json.isLogin==="True"){
              props.setMode("WELCOME");
            }
            else {
              alert(json.isLogin)
            }
          });
      }} /></p>
    </div>

    <p>계정이 없으신가요?  <button onClick={() => {
      props.setMode("SIGNIN");
    }}>회원가입</button></p>
  </>
}
```

Login

App.js

```
<p><input className="btn" type="submit" value="회원가입" onClick={() => {
  const userData = {
    userId: id,
    userPassword: password,
    userPassword2: password2,
  };
  fetch("http://localhost:3001/signin", { //signin 주소에서 받을 예정
    method: "post", // method :통신방법
    headers: {       // headers: API 응답에 대한 정보를 담음
      "content-type": "application/json",
    },
    body: JSON.stringify(userData), //userData라는 객체를 보냄
  })
    .then((res) => res.json())
    .then((json) => {
      if(json.isSuccess==="True"){
        alert('회원가입이 완료되었습니다!')
        props.setMode("LOGIN");
      }
      else{
        alert(json.isSuccess)
      }
    });
 }} /></p>
</div>
```

Login

App.js

```
    <p>로그인화면으로 돌아가기  <button onClick={() => {
      props.setMode("LOGIN");
    }}>로그인</button></p>
  </>
}

function App() {
  const [mode, setMode] = useState("LOGIN");

  useEffect(() => {
    fetch("http://localhost:3001/authcheck")
      .then((res) => res.json())
      .then((json) => {
        if (json.isLogin === "True") {
          setMode("WELCOME");
        }
        else {
          setMode("LOGIN");
        }
      });
  }, []);
```

```
  let content = null;

  if(mode==="LOGIN"){
    content = <Login setMode={setMode}></Login>
  }
  else if (mode === 'SIGNIN') {
    content = <Signin setMode={setMode}></Signin>
  }
  else if (mode === 'WELCOME') {
    content = <>
                <h2>메인 페이지에 오신 것을 환영합니다</h2>
                <p>로그인에 성공하셨습니다.</p>
                <a href="/logout">로그아웃</a>
    </>
  }

  return (
    <>
      <div className='background'>
        {content}
      </div>
    </>
  );
}

export default App;
```

Login

db.js

- MySQL에 접속하기 위한 정보를 담고 있는 db.js 파일
- mysql2 모듈을 사용하고 있으나 mysql 모듈을 사용하셔도 변경하실 내용은 없다.
- 본인이 원하는 대로 host, user, password, database의 내용을 채워서 사용

```
var mysql = require('mysql2');
var db = mysql.createConnection({
    host: '',
    user: '',
    password: '',
    database: ''
});
db.connect();

module.exports = db;
```

Login

sessionOption.js

- MySQL에 세션을 저장하기 위한 옵션을 저장하는 파일
- db.js와 마찬가지로 원하는 대로 host, user, password, database의 내용을 채워서 사용
- 세션의 지속시간 등은 편의에 따라 수정하면 됨
- db.js와 sessionOption.js을 별도의 파일로 저장하는 이유는 두 파일이 저장하는 내용이 보안에 민감한 내용이기 때문

```
// MySQL에 저장할 세션의 옵션
var options = {
  host: '',
  user: '',
  password: '',
  database: '',
  port: 3306,

  clearExpired : true ,              // 만료된 세션 자동 확인 및 지우기 여부
  checkExpirationInterval: 10000,    // 만료된 세션이 지워지는 빈도 (milliseconds)
  expiration: 1000*60*60*2,          // 유효한 세션의 최대 기간 2시간으로 설정
(milliseconds)
};

module.exports = options;
```

Login

실행하기

- 위의 소스코드를 모두 작성한 다음, npm run build를 이용해서 리액트 소스를 빌드하면 build 폴더에 index.html 등의 파일이 생성
- 그런 다음 node server.js를 입력하여 서버를 실행시키면 위의 코드들로 구현된 로그인/회원가입 페이지가 http://localhost:3001/ 에서 열림

```
>npm run build
>node server.js
```

레포트(optional)

로그인 화면 추가하기
게시판 추가하기
공공데이터 API 사용하기

발표주제

rest api
Fetch 함수

Reference

- https://itshootingstar.tistory.com/100
- https://sirius7.tistory.com/101

TL;DR