

# JWT를 이용한 인증



# 1. JWT이란?

## ❖ JWT : JSON Web Token

- 서명 검증을 통해 토큰의 위변조 유무를 확인할 수 있도록 사용하는 JSON 기반의 토큰
- 서명이 검증되어 payload가 위변조되지 않았다는 것이 확인되면 payload 정보를 신뢰하여 사용함
- JWT 구조
  - HS 방식 : HMAC-SHA, secret을 이용해 서명생성, 검증
  - RS 방식 : RSA-SHA, 서명생성은 Private Key, 서명 검증은 Public Key를 이용함

header	{ "alg" : "HS256", "typ": "JWT" }	→ ㉠
payload	{ "userid": "gdhong", "iat": 1704103645607, "exp": 3600, ..... }	→ ㉡
signature	HMAC-SHA256( secret, base64Encode(header) + '.' + base64Encode(payload) )	→ ㉢

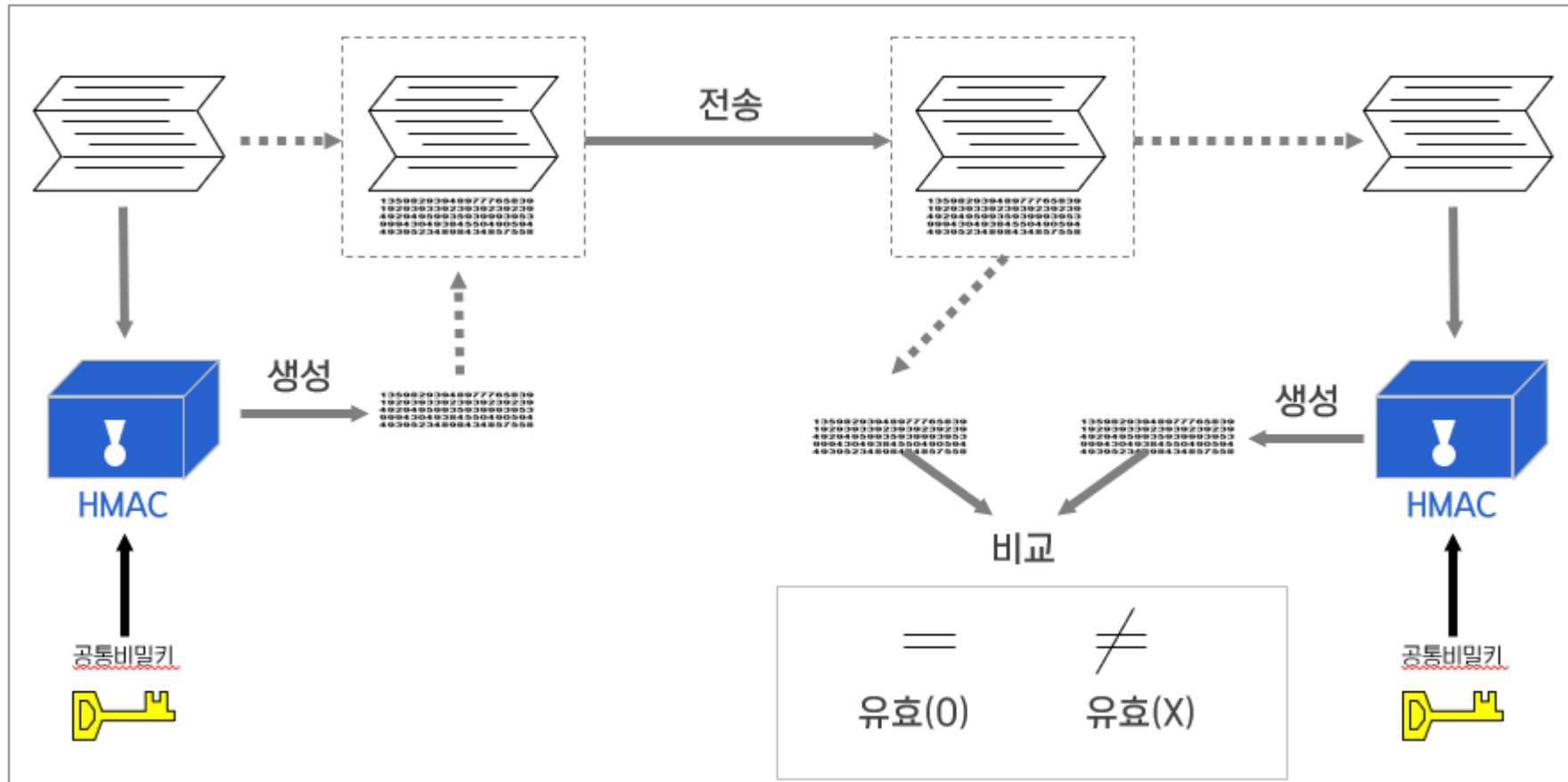
JWT

base64Encode(a) + "." +  
base64Encode(b) + "." +  
base64Encode(c)

# 1. JWT이런?

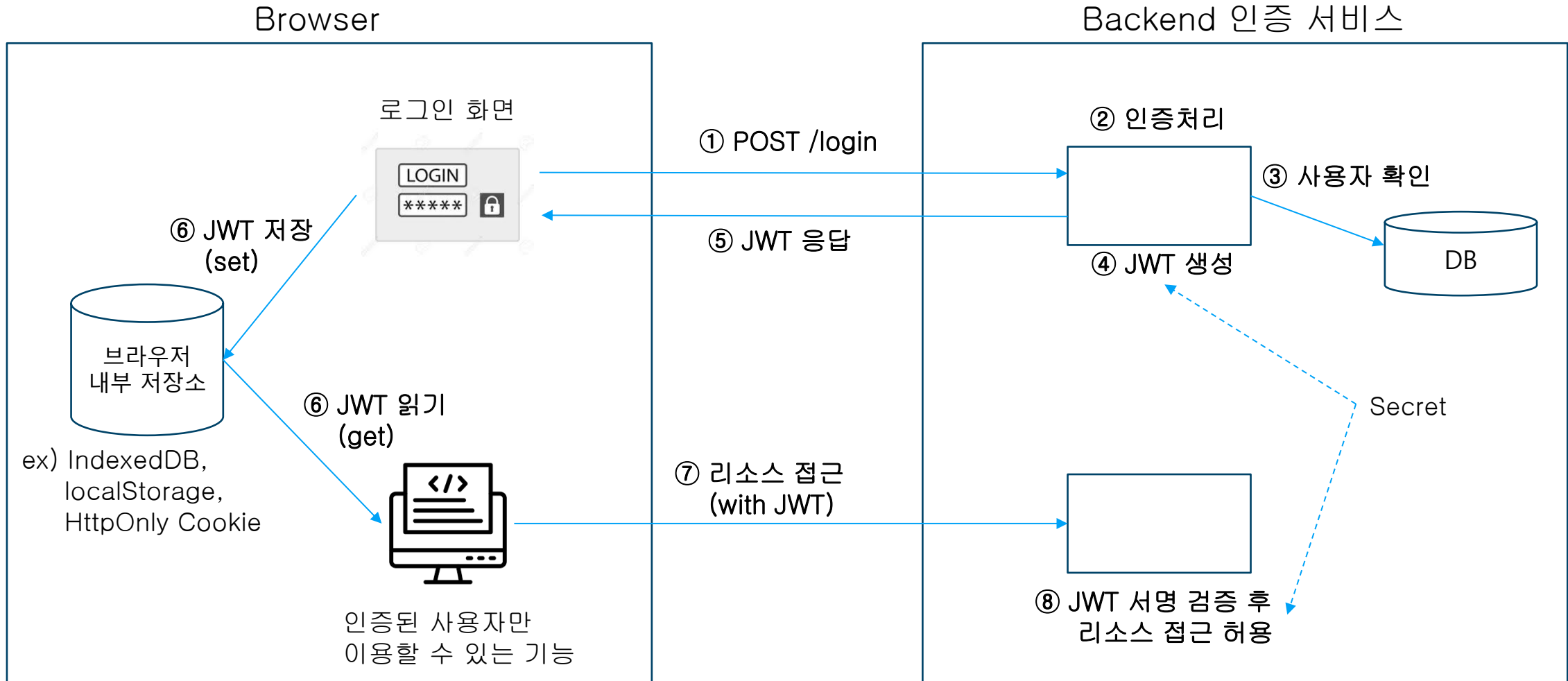
## ❖ HMAC : Hash based Message Authentication Code

- JWT 이해를 돕기 위해...



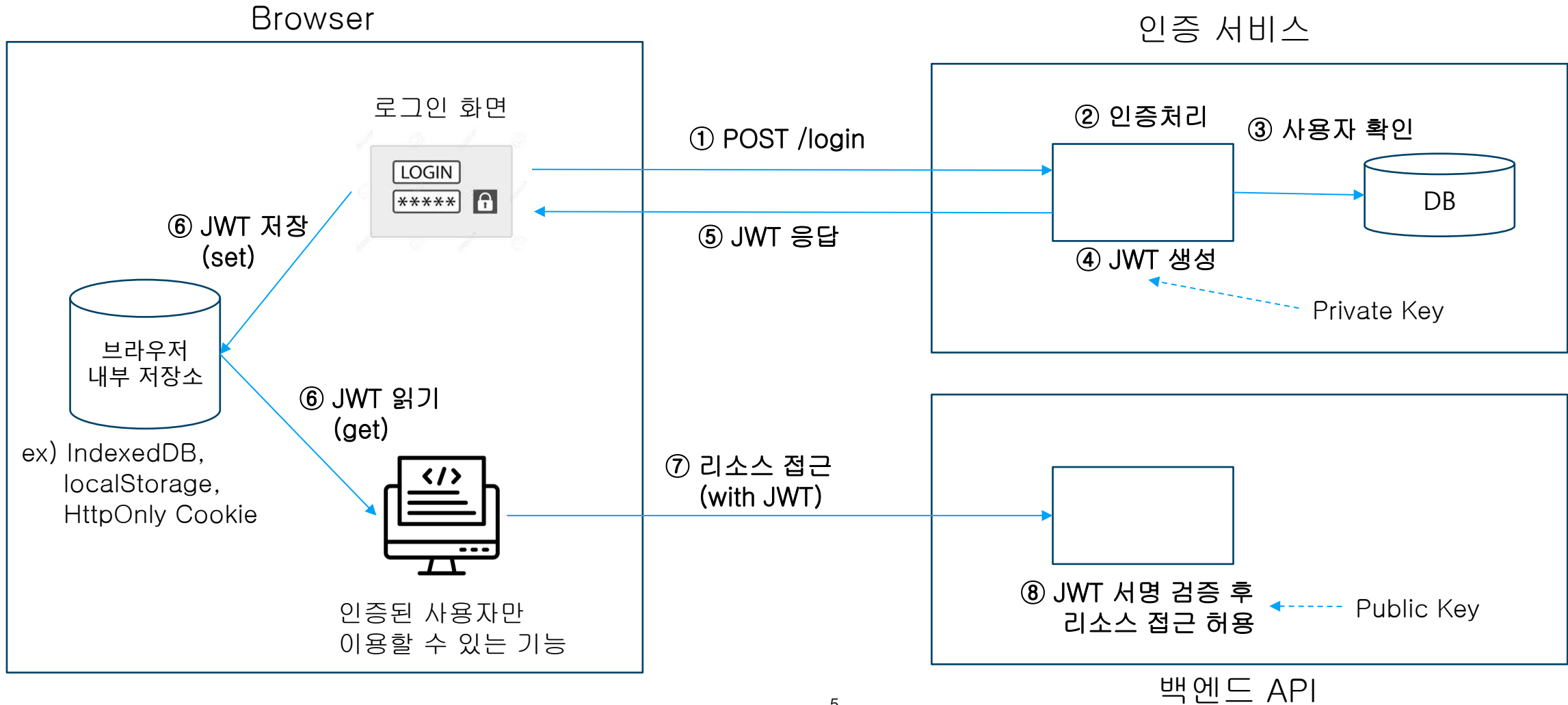
# 1. JWT이란?

## ❖ JWT : JSON Web Token - HS 방식



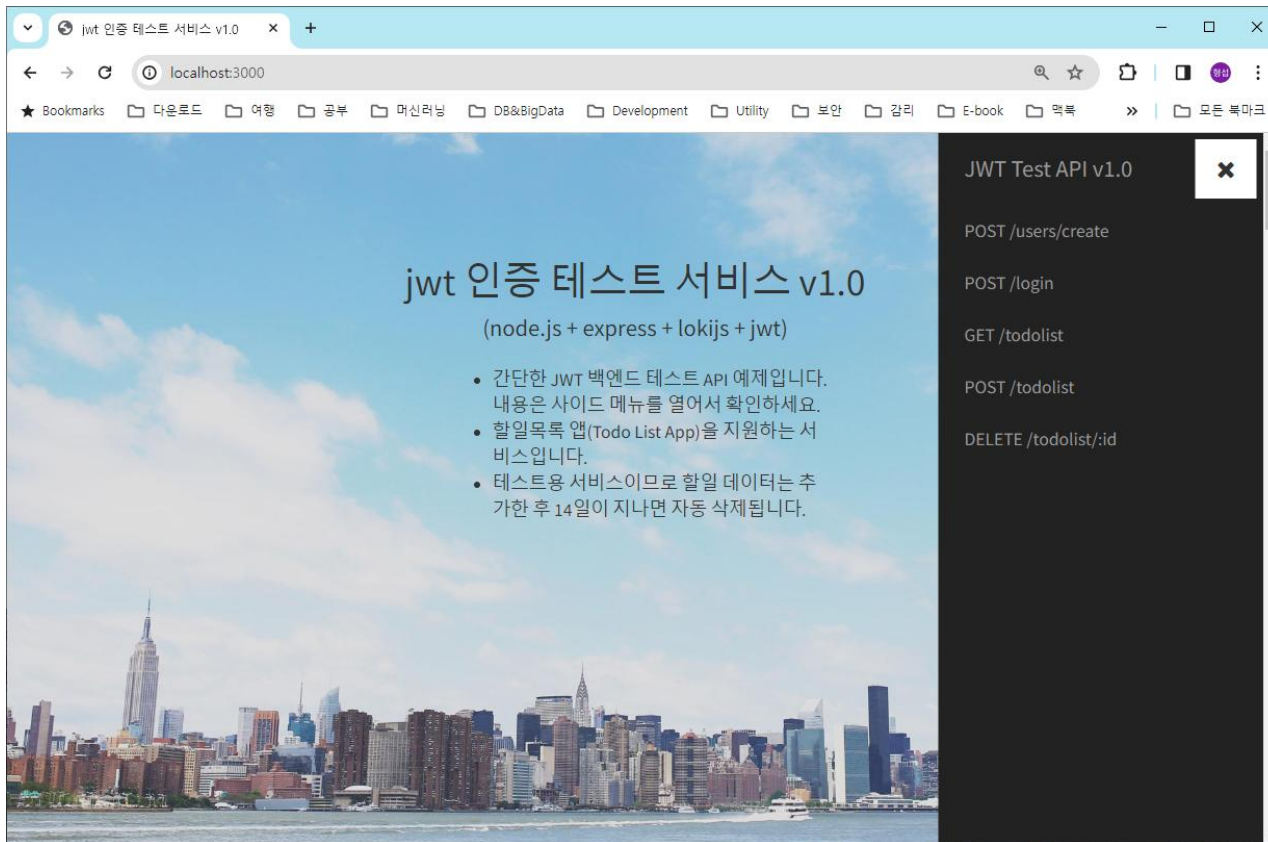
# 1. JWT이란?

## ❖ JWT : JSON Web Token - RS 방식



## 2. JWT 인증 테스트 서비스

- ❖ 강사로부터 todosvc-jwt 프로젝트 제공받아 실행할 것
  - <https://github.com/stepanowon/todosvc-jwt>
  - 실행 방법 : `npm install --> npm run start-dev`
  - <http://localhost:3000> 화면에서 도움말 확인



## 2. JWT 인증 테스트 서비스

### ❖제공 엔드포인트

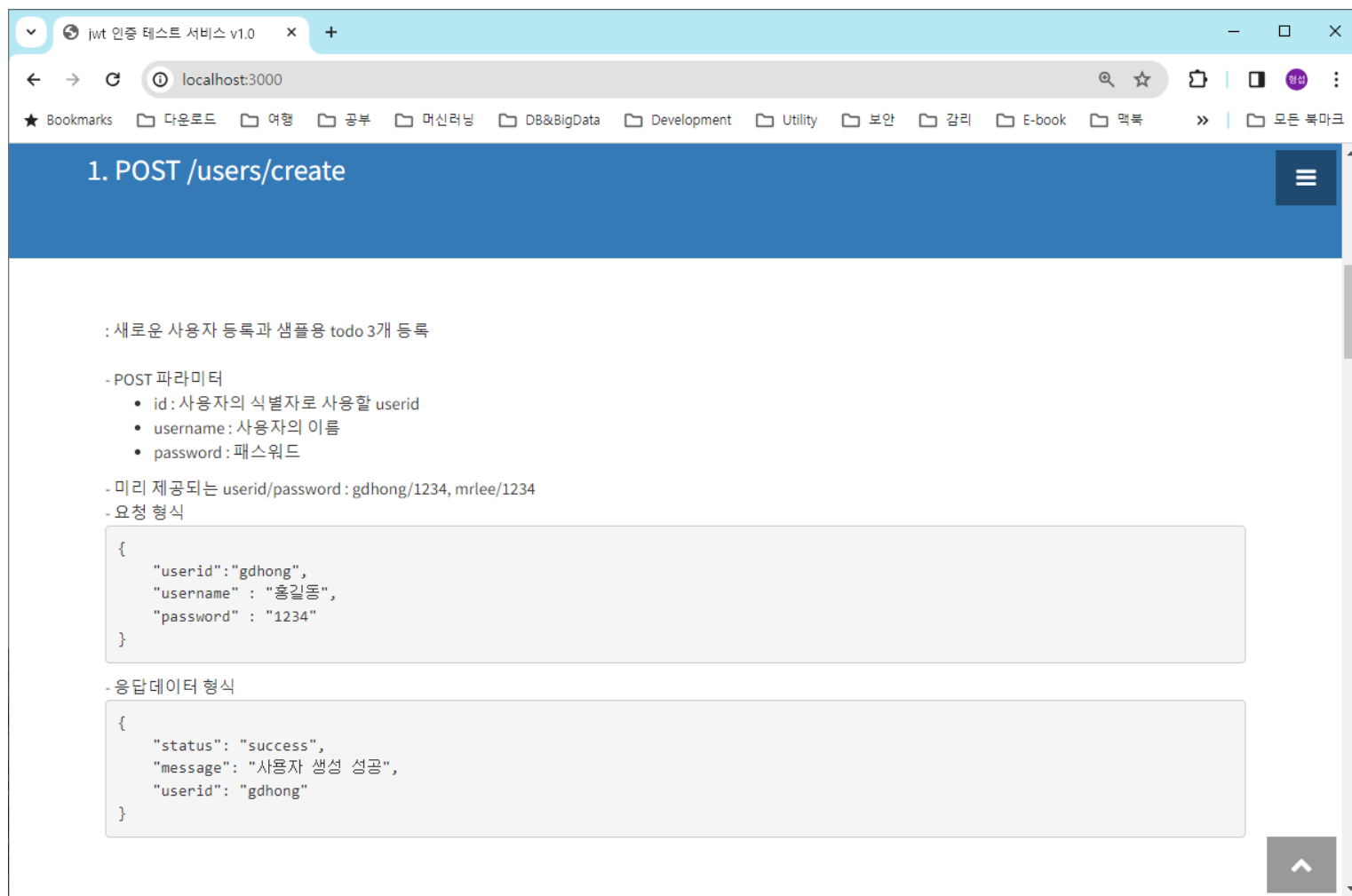
- 사용자 로그인, token 기능
  - POST /users/create : 새로운 사용자 등록
  - POST /login : 로그인후 access\_token, refresh\_token 발급
  - POST /token : refresh\_token을 이용해 access\_token 재발급
- 할일 서비스 : Authorization 요청 헤더로 access\_token을 함께 전송해야 함
  - GET /todolist: 사용자의 todolist 정보 획득
  - POST /todolist : 사용자의 todolist에 새로운 todo 추가
  - DELETE /todolist/:id : id를 이용해 todo 한건 삭제

### ❖미리 제공되는 계정

- admins 역할 부여
  - admin/1234 사용자
- users 역할
  - gdhong/1234(홍길동), mrlee/1234(이몽룡)
  - 새롭게 추가하는 사용자

## 2. JWT 인증 테스트 서비스

### ❖ 서비스의 도움말 페이지





## 2. JWT 인증 테스트 서비스

### ❖ Postman을 이용한 테스트

#### ■ 사용자 생성

The screenshot displays the Postman web interface. On the left, a sidebar shows the 'History' tab with a 'New' button and an 'Import' button. Below this is a cartoon astronaut and the text 'Time to send your first request'. The main area shows a POST request to 'http://localhost:3000/users/create'. The request body is a JSON object: 

```
{  "userid": "gdhong2",  "username": "홍길동2",  "password": "1234"}
```

. The 'Body' tab is selected, and the 'JSON' format is chosen. A 'Send' button is visible. Below the request, the response is shown in the 'Body' tab, indicating a '200 OK' status with a message: 

```
{  "status": "success",  "message": "사용자 등록 성공!"}
```

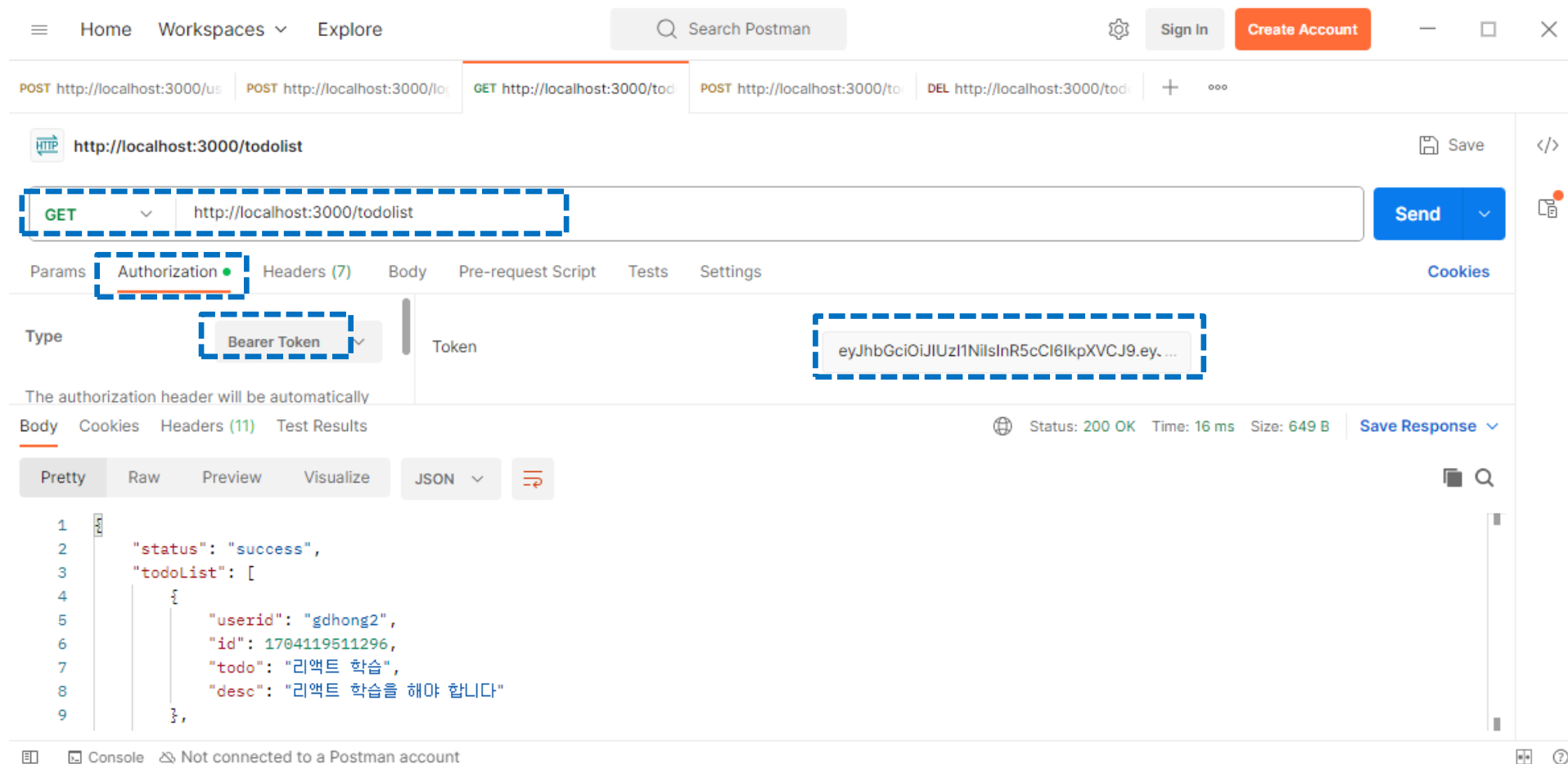
. The interface also includes a search bar, 'Sign In', and 'Create Account' buttons at the top.



## 2. JWT 인증 테스트 서비스

### ❖ Postman을 이용한 테스트

- 획득한 JWT를 이용해 리소스 접근



### 3. React App에서 구현해야 할 기능

#### ❖ 백엔드와의 통신을 위해

- 로그인후 받아온 JWT를 브라우저 내부 저장소에 저장
  - IndexedDB
  - LocalStorage
  - HttpOnly Cookie
- 인증된 사용자만을 위한 리소스 API에 접근할 때
  - Authorization 요청 헤더를 이용해 Bearer 토큰으로 전달
    - Authorization: Bearer XXXXXXXXXXXX
- 이것으로 끝이 아니다.

#### ❖ 프론트엔드 앱 내부에서의 필요한 처리?

- 저장된 JWT에 접근하는 코드
- 리액트 앱 내부에서의 접근 제어
  - 로그인하지 않은 채 리소스 이용화면에 접근하는 경우 로그인 화면으로 이동시켜야 함
  - react-router를 이용

## 4. JWT 저장소 선택

### ❖ Web Storage

- localStorage
  - 동기식 key-value 텍스트로 저장하며 사용이 간편함
  - 브라우저 종료 후 재시작해도 정보가 남아있음
  - 다른 창, 탭을 통해서도 접근할 수 있음'
- sessionStorage
  - localStorage와 마찬가지로 동기식 key-value 텍스트 저장소
  - 브라우저가 종료되면 사라짐
  - 다른 탭, 창에서 접근 불가
- IndexedDB
  - localStorage, sessionStorage와는 달리 텍스트가 아닌 객체 타입을 저장할 수 있음
  - 비동기 방식

## 4. JWT 저장소 선택

### ❖ WebStorage 방식의 단점

- 자바스크립트 코드로 접근하기 때문에 XSS 공격의 위험이 존재함
- 자동 네트워크 전송을 지원하지 않기 때문에 클라이언트 애플리케이션이 직접 전송해야 함

### ❖ Httponly Cookie

- 자바스크립트 코드로 쿠키를 열람할 수 없기 때문에 XSS 공격에 대해 상대적으로 안전함
  - HTTPS를 반드시 사용해야 하고 domain, sameSite와 같은 설정이 번거롭고 테스트가 힘들
  - XSS 취약점을 노려서 외부 API를 호출하는 방법으로 XSS 공격이 가능함

### ❖ Google Firebase

- IndexedDB 사용
- XSS 공격에 대한 방어를 철저히 하는 것이 가장 중요함

## 4. JWT 저장소 선택

### ❖ 저장소 선택 방안

- 1안
  - refresh\_token은 HttpOnly Secure Cookie
  - access\_token은 유효시간을 짧게 하여 localStorage, IndexedDB, 브라우저 메모리 중 한 곳에 저장
- 2안
  - refresh\_token은 SessionStorage, IndexedDB
  - access\_token은 유효시간을 짧게 하여 localStorage, IndexedDB, 브라우저 메모리 중 한 곳에 저장
- XSS 공격을 방어할 수 있는 처리가 필수
  - JSX는 기본적으로 HTML 태그 문자열을 자동으로 Escape 처리함
- 결론적으로 1안,2안 모두 사용해도 무방함

## 5. Protected route

### ❖ Protected Route (Private Route)란?

- react-router를 이용해 권한이 있어야만 접근할 수 있는 Route

```
const ProtectedRoute = ({ children }: { children: JSX.Element }) => {  
  const auth = useAuth();           //인증정보 처리를 위한 사용자 정의 훅  
  const location = useLocation();  
  
  if (!auth.isAccessibleToPath(location.pathname)) {  
    return <Navigate to="/login" state={{ from: location }} replace />;  
  }  
  
  return children;  
}
```

```
<Routes>  
  <Route path="/" element={<PublicPage />} />  
  <Route path="/login" element={<LoginPage />} />  
  <Route path="/users" element={ <ProtectedRoute><UserPage /></ProtectedRoute> } />  
  <Route path="/admins" element={ <ProtectedRoute><AdminPage /></ProtectedRoute> } />  
</Routes>
```

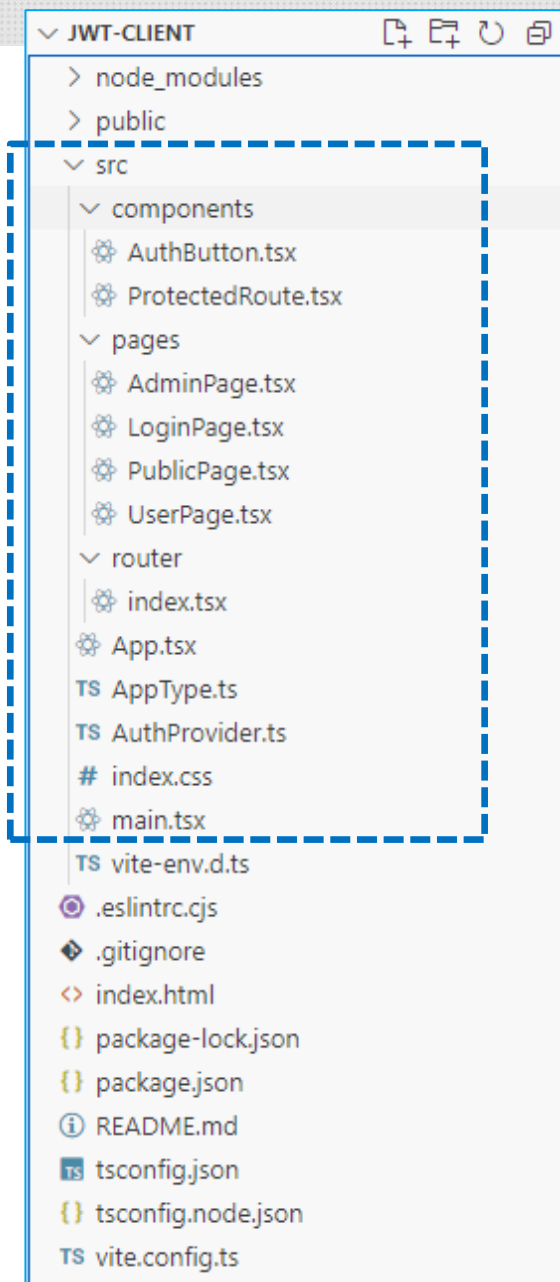


## 6. localStorage + JWT + react-router 예제

### ❖ 프로젝트 생성

- `npm init vite jwt-client -- --template react-swc-ts`
- `cd jwt-client`
- `npm install`
- `npm install axios react-router-dom`

### ❖ 프로젝트 디렉토리 구조



## 6. localStorage + JWT + react-router 예제

### ❖useAuth 사용자 정의 훅

- localStorage에 토큰 저장/읽기 기능 : saveToken, loadToken
- JWT 토큰 파싱 기능 : parseAccessToken
- 토큰의 유효성 여부 확인 기능 : isValidAccessToken
- 현재 로그인한 유저 정보 리턴 기능 : getCurrentInfo
- 로그인/로그아웃 처리 기능 : loginProcess, logoutProcess
- 리프레시 토큰을 이용한 액세스토큰 재발급 : refreshTokenProcess
- 인자로 전달된 경로가 현재 토큰으로 접근가능한지 여부 확인 기능 : isAccesibleToPath

## 6. localStorage + JWT + react-router 예제

### ❖src/AuthProvider.ts

```
import axios from "axios";

const useAuth = () => {
  //각 요청 경로별 권한(Role 기반)
  const pathToRoles = [
    { path: "/users", role: "users" },
    { path: "/admins", role: "admins" },
  ];

  const saveToken = (access_token:string, refresh_token:string) => {
    window.localStorage.setItem("access_token", access_token);
    window.localStorage.setItem("refresh_token", refresh_token);
  }

  const loadToken = () => {
    const access_token = window.localStorage.getItem("access_token");
    const refresh_token = window.localStorage.getItem("refresh_token");
    return { access_token, refresh_token };
  }
}
```

## 6. localStorage + JWT + react-router 예제

### ❖src/AuthProvider.ts (이어서2)

```
const parseAccessToken = () => {
  try {
    const { access_token } = loadToken();
    if (!access_token) throw new Error("유효한 토큰이 존재하지 않습니다.");
    const arr = access_token.split(".");
    const claimSet = JSON.parse(atob(arr[1]))
    return claimSet;
  } catch (e) {
    return "토큰 구문 분석 오류";
  }
}

const isValidAccessToken = () => {
  const result = { valid: true, message: "정상적인 토큰" };
  const claimSet = parseAccessToken();
  if (typeof(claimSet) !== "string") {
    result.valid = false;
    result.message = claimSet;
  }
  const currentTimeStamp = new Date().getTime() / 1000 ;
  if (claimSet.exp < currentTimeStamp) {
    result.valid = false;
    result.message = "파기된 토큰, refresh_token을 이용해 access_token을 재발급받으세요";
  }
  return result;
}
```

## 6. localStorage + JWT + react-router 예제

### ❖src/AuthProvider.ts (이어서3)

```
const getCurrentUserInfo = () => {
  const { userid, role } = parseAccessToken();
  return { userid, role };
}

const loginProcess = async (
  userid:string,
  password:string ,
  callback:(cbArgs: { status:string; message:string}) => void
) => {
  //BASEURL은 main.tsx 참조
  const LOGIN_URL = "/login";
  try {
    const response = await axios.post(LOGIN_URL, { userid, password });
    if (response.data.status === "success") {
      saveToken(response.data.access_token, response.data.refresh_token);
      axios.defaults.headers.common["Authorization"] = "Bearer " + response.data.access_token;
      callback({ status: "ok", message:"로그인 성공"})
    } else {
      callback({ status: "fail", message: response.data.message});
    }
  } catch(e) {
    callback({ status: "fail", message:"로그인 실패 - 서버 오류" });
  }
}
```

## 6. localStorage + JWT + react-router 예제

### ❖src/AuthProvider.ts (이어서4)

```
const getCurrentUserInfo = () => {
  const { userid, role } = parseAccessToken();
  return { userid, role };
}

const loginProcess = async (
  userid:string,
  password:string,
  callback:(cbArgs: { status:string; message:string}) => void
) => {
  //BASEURL은 main.tsx 참조
  const LOGIN_URL = "/login";
  try {
    const response = await axios.post(LOGIN_URL, { userid, password });
    if (response.data.status === "success") {
      saveToken(response.data.access_token, response.data.refresh_token);
      axios.defaults.headers.common["Authorization"] = "Bearer " + response.data.access_token;
      callback({ status: "ok", message:"로그인 성공"})
    } else {
      callback({ status: "fail", message: response.data.message});
    }
  } catch(e) {
    callback({ status: "fail", message:"로그인 실패 - 서버 오류" });
  }
}
```

## 6. localStorage + JWT + react-router 예제

### ❖src/AuthProvider.ts (이어서5)

```
const logoutProcess = (callback:()=>void) => {
  saveToken("", ""); //토큰 삭제
  axios.defaults.headers.common["Authorization"] = "";
  callback();
}

const refreshTokenProcess = async (
  refresh_token:string,
  callback:(cbArgs: { status?:string; message?:string}) => void
) => {
  const TOKEN_URL = "/token";
  try {
    const response = await axios.post(TOKEN_URL, { refresh_token });
    if (response.data.status === "success") {
      saveToken(response.data.access_token, response.data.refresh_token);
      axios.defaults.headers.common["Authorization"] = "Bearer " + response.data.access_token;
      callback({ status: "ok", message:"토큰 재발급 성공"})
    } else {
      callback({ status: "fail", message: response.data.message});
    }
  } catch(e) {
    callback({ status: "fail", message:"토큰 재발급 실패 - 서버 오류" });
  }
}
```

## 6. localStorage + JWT + react-router 예제

### ❖src/AuthProvider.ts (이어서6)

```
const isAccessibleToPath = (path:string) => {  
  //pathToRoles 에 있는 경로 중 path와 일치하는 경로를 가진 것을 찾음  
  const pathToRole = pathToRoles.find((p)=> p.path === path)  
  //매칭되는 경로가 없다면 권한이 없어도 접근 가능한 것으로 간주함  
  if (!pathToRole) return true;  
  
  //JWT가 유효하지 않으므로  
  if (isValidAccessToken().valid === false) return false;  
  
  //JWT에서 사용자 정보 payload 파싱  
  const claimSet = parseAccessToken();  
  //요청된 경로에서 요구하는 Role  
  const requiredRole = pathToRole.role;  
  
  if (claimSet.role === requiredRole || claimSet.role === "admins") return true;  
  else return false;  
}  
  
return { parseAccessToken, isValidAccessToken, getCurrentUserInfo, loginProcess, logoutProcess,  
        refreshTokenProcess, isAccessibleToPath }  
}  
  
export { useAuth };
```



## 6. localStorage + JWT + react-router 예제

### ❖src/components/ProtectedRoute.tsx

```
import { Navigate, useLocation } from "react-router-dom";
import { useAuth } from "../AuthProvider";


const ProtectedRoute = ({ children }: { children: JSX.Element }) => {
  const auth = useAuth();
  const location = useLocation();

  if (!auth.isAccessibleToPath(location.pathname)) {
    return <Navigate to="/login" state={{ from: location }} replace />;
  }

  return children;
}

export default ProtectedRoute;
```

children



```
<ProtectedRoute>
  <PrivateComponent />
</ProtectedRoute>
```

## 6. localStorage + JWT + react-router 예제

## ❖src/components/AuthButton.tsx

- 로그인 여부에 따라 다르게 보여줄 로그인/로그아웃 버튼 컴포넌트

```
import { useNavigate } from 'react-router-dom';
import { useAuth } from '../AuthProvider';

const AuthButton = () => {
  const navigate = useNavigate();
  const auth = useAuth();
  const userInfo = auth.getCurrentUserInfo();

  return userInfo.userid ? (
    <span style={{ marginLeft : "100px", color: "blue" }}>
      Logged in : (Role : {userInfo.role}) &nbsp;&nbsp;&nbsp;
      <button onClick={() => { auth.logoutProcess(() => navigate("/")); }}>logout </button>
    </span>
  ) : (
    <span style={{ marginLeft : "100px", color: "red" }}>Not Logged in</span>
  );
};

export default AuthButton;
```

## 6. localStorage + JWT + react-router 예제

### ❖src/pages/LoginPage.tsx

- 로그인 화면 제공, 로그인 기능

```
import { useState } from 'react';
import { useNavigate, useLocation } from "react-router-dom";
import { useAuth } from '../AuthProvider';

const LoginPage = () => {
  const navigate = useNavigate();
  const location = useLocation();
  const auth = useAuth();
  const [userid, setUserId] = useState<string>("");
  const [password, setPassword] = useState<string>("");

  const { from } = location.state || { from: { pathname: '/' } };
  const login = () => {
    auth.loginProcess(userid, password, ({ status, message })=>{
      if (status === "ok") {
        alert("로그인 성공");
        navigate(from);
      } else {
        alert("로그인 실패 : " + message);
      }
    })
  };
};
```

## 6. localStorage + JWT + react-router 예제

### ❖src/pages/LoginPage.tsx(이어서)

```
return (  
  <div style={{ margin:"20px" }}>  
    <div style={{ marginTop:"10px", marginBottom:"10px" }}>  
      아이디 : <input type="text" value={userid} onChange={(e)=>setUserid(e.target.value)} /> <br/>  
      암호 : <input type="password" value={password} onChange={(e)=>setPassword(e.target.value)} /> <br />  
    </div>  
    <button onClick={login}>로그인</button>  
  </div>  
);  
};  
  
export default LoginPage;
```

## 6. localStorage + JWT + react-router 예제

### ❖src/pages/PublicPage.tsx

- 로그인하지 않아도 접근가능한 컴포넌트

```
const PublicPage = () => {  
  return (  
    <div>  
      <h3>Home : 로그인하지 않아도 접근가능한 페이지</h3>  
    </div>  
  );  
};  
  
export default PublicPage;
```

### ❖src/ApiType.ts

- axios 로 요청/응답할 때 사용하는 Type

```
export type TodoItemType = { id: number; userid: string; todo: string; desc: string; }  
export type TodoListResponseType = { status: string; todoList : TodoItemType[] }
```

## 6. localStorage + JWT + react-router 예제

### ❖src/pages/UserPage.tsx

- src/pages/AdminPage.tsx 는 볼드체 부분만 다르게 작성할 것

```
import { useState } from 'react';
import { useAuth } from '../AuthProvider';
import axios from 'axios';
import { TodoItemType, TodoListResponseType } from '../AppType';

// src/main.tsx의 baseURL 설정 확인할 것
const TODOLIST_URL = "/todolist";

const UserPage = () => {
  const auth = useAuth();
  const userInfo = auth.getCurrentUserInfo();
  const [todoList, setTodoList] = useState<TodoItemType[]>([]);

  const getTodoList = async () => {
    const response = await axios.get<TodoListResponseType>(TODOLIST_URL);
    if (response.data.status === "success") {
      setTodoList(response.data.todoList);
    } else {
      setTodoList([]);
    }
  }
}
```

## 6. localStorage + JWT + react-router 예제

### ❖src/pages/UserPage.tsx (이어서)

```
return (  
  <>  
    <div>  
      <h3>사용자 페이지 : users 역할이 필요함.</h3>  
      <p>사용자 : {userInfo.userid}, 역할 : {userInfo.role}</p>  
    </div>  
    <div>  
      <button onClick={getTodoList}>TodoList 조회</button>  
      <hr />  
      <ul>  
        {  
          todoList.length === 0 ? <li>데이터 없음</li> :  
            todoList.map((todoItem)=>(<li key={todoItem.id}>{todoItem.todo}</li>))  
        }  
      </ul>  
    </div>  
  </>  
>);  
};  
  
export default UserPage;
```

## 6. localStorage + JWT + react-router 예제

### ❖src/App.tsx

```
import { Link, Outlet } from 'react-router-dom';

import AuthButton from './components/AuthButton';

const App = () => {
  return (
    <div style={{ margin:'10px' }}>
      <div>
        <span style={{}}>
          <Link to="/">Home</Link> &nbsp; | &nbsp; &nbsp; &nbsp;
          <Link to="/users">Users</Link> &nbsp; | &nbsp; &nbsp; &nbsp;
          <Link to="/admins">관리자페이지</Link>
          <AuthButton />
        </span>
      </div>
      <hr/>
      <Outlet />
    </div>
  );
};
export default App;
```



## 6. localStorage + JWT + react-router 예제

### ❖src/router/index.tsx

#### ▪ 볼드체로 표현된 부분 검토

```
import { createBrowserRouter } from "react-router-dom";
import App from "../App";
import PublicPage from "../pages/PublicPage";
import LoginPage from "../pages/LoginPage";
import ProtectedRoute from "../components/ProtectedRoute";
import UserPage from "../pages/UserPage";
import AdminPage from "../pages/AdminPage";

const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
    children : [
      { index:true , element: <PublicPage /> },
      { path:"login", element: <LoginPage /> },
      { path:"users", element: <ProtectedRoute><UserPage /></ProtectedRoute> },
      { path:"admins", element: <ProtectedRoute><AdminPage /></ProtectedRoute> },
    ]
  },
]);
export default router;
```

## 6. localStorage + JWT + react-router 예제

### ❖src/index.css

```
body {  
  margin: 10px;  
}
```

### ❖src/main.tsx

```
import React from 'react'  
import ReactDOM from 'react-dom/client'  
import './index.css'  
import axios from 'axios';  
import { RouterProvider } from 'react-router-dom';  
import router from './router/index.tsx';  
  
axios.defaults.baseURL = "http://localhost:3000";  
  
ReactDOM.createRoot(document.getElementById('root')!).render(  
  <React.StrictMode>  
    <RouterProvider router={router} />  
  </React.StrictMode>,  
)
```

## 6. localStorage + JWT + react-router 예제

### ❖ 실행 결과

