KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

로그인

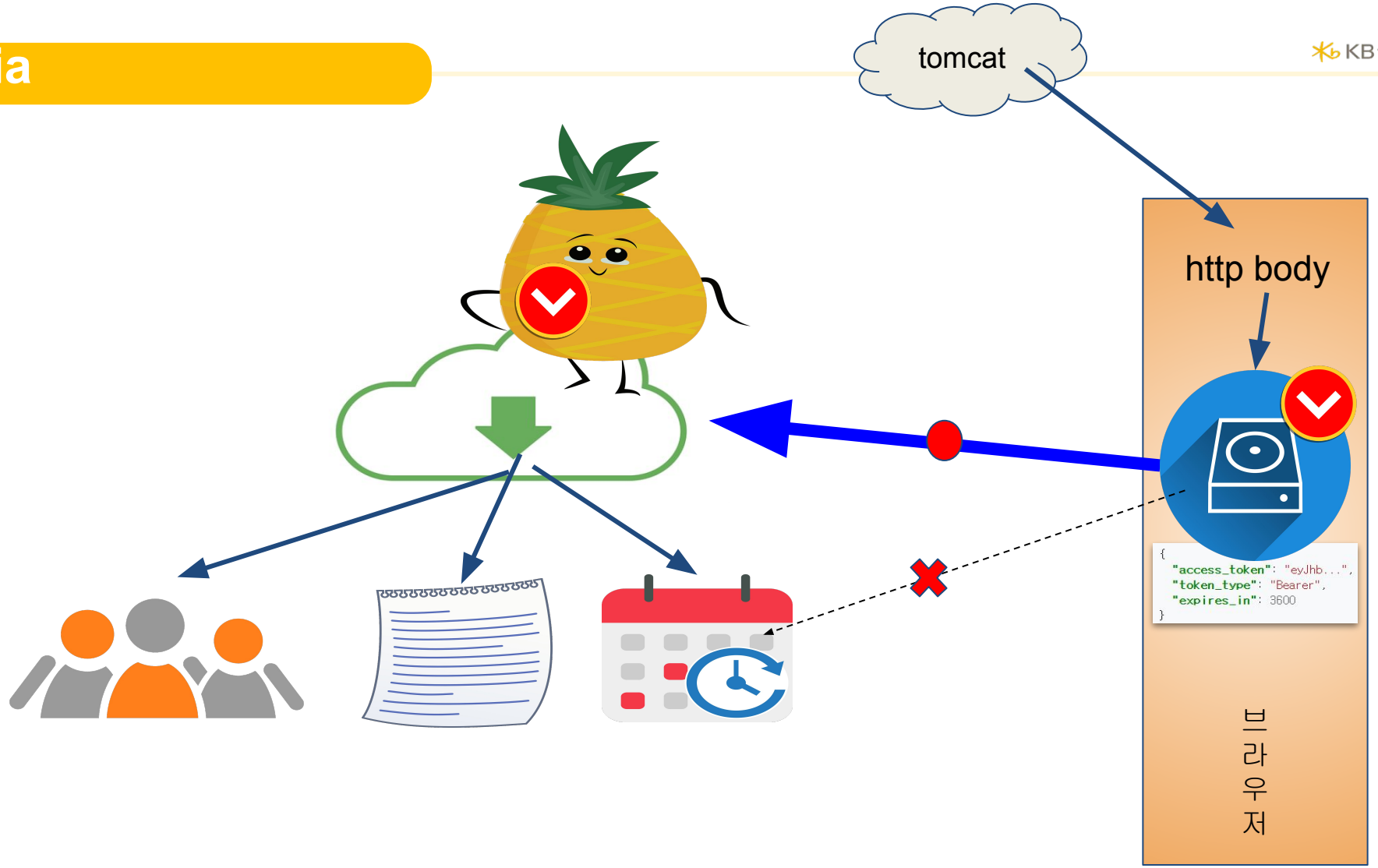
[KB] IT’s Your Life

KB 국민은행

로그인

- **API 기반 Spring Security에서 처리 함**
  - 추가 작업 없음

로컬 스토리지와 pinia



** pinia(js-중앙저장소)

- 로그인 → JWT 발급 → localStorage 저장 → 앱 재시작 시 pinia에 다시 저장해서 반응성 상태로 관리

KB금융그룹 | 국민의 평생 금융파트너

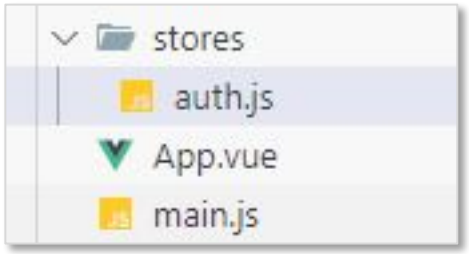
2025년 상반기 K-디지털 트레이닝

인증 스토어

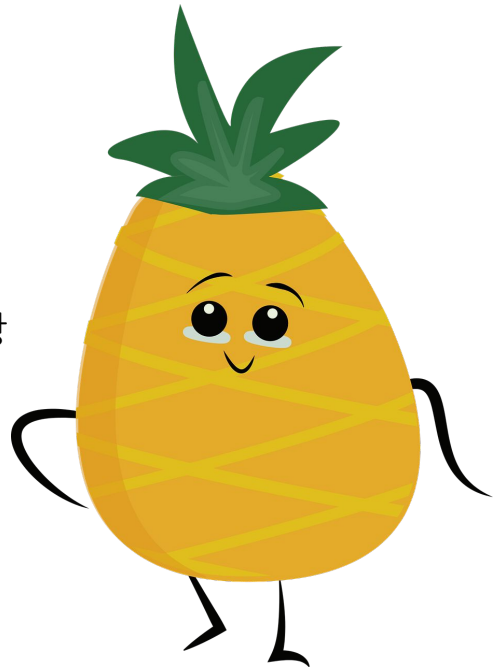
[KB] IT’s Your Life

KB 국민은행

- 인증 스토어



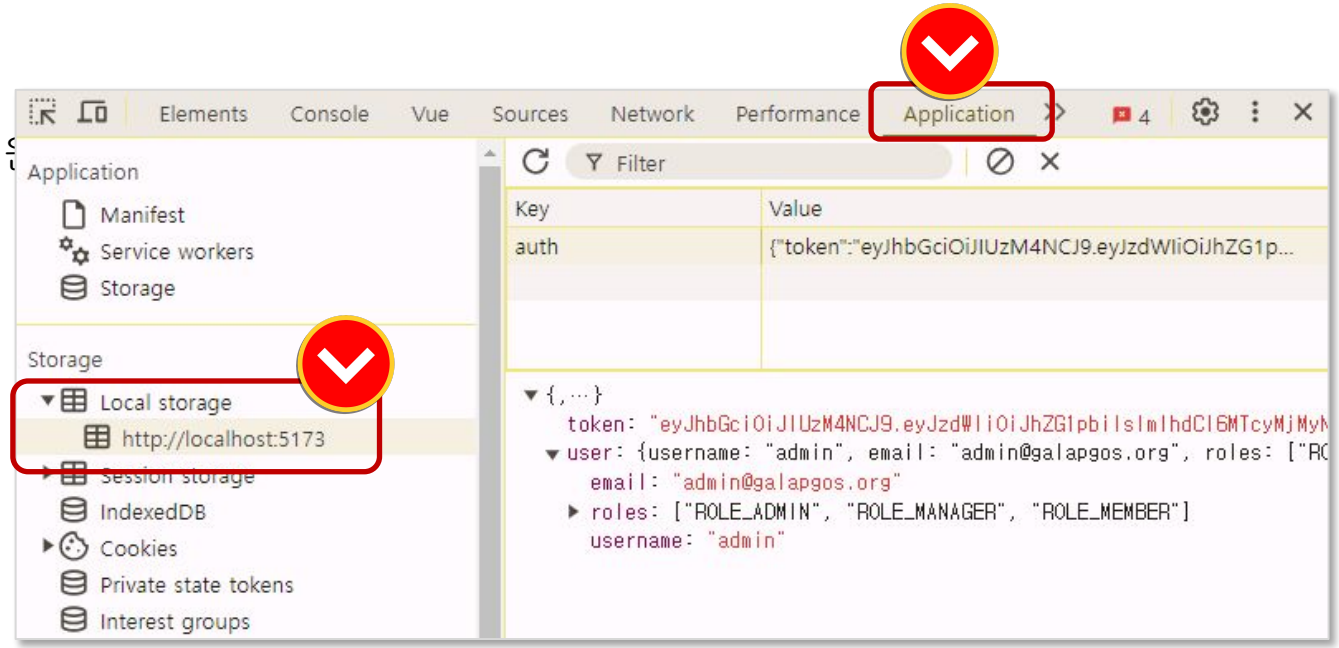
- 서버 인증은 추후에 작업
- 임시 작업 – 로그인하면 무조건 로그인 되는 것으로
- **새로 고침을 하면 스토어가 리셋 됨**
  - 상태가 변경될 때(로그인, 로그아웃)마다 localStorage에 저장
  - 첫 기동할 때 localStorage에서 상태 복원



인증 스토어

- **localStorage(git-note#55)**
  - 웹 어플리케이션 운영가능하는 저장소
  - <키, 값>의 쌍으로 정보를 저장
    - 키, 값 모두 문자열로 처리
    - 객체 정보를 저장할 때 JSON 문자열로 변환해서 저장

  - 주요 메서드
    - setItem(키, 값)
      - 값을 JSON 직렬화하여 문자열로 저장
    - getItem(키)
      - 리턴된 문자열을 JSON 역직렬화로 객체 복
    - removeItem(키)
    - clear()



인증 스토어

- **stores/auth.js**

```
import { ref, computed, reactive } from 'vue';
import { defineStore } from 'pinia';

const initState = {
  token: '',       // 접근 토큰(JWT)
  user: {
    username: '',          // 사용자 ID
    email: '',      // Email
    roles: [],      // 권한 목록
  },
};

export const useAuthStore = defineStore('auth', () => {
  const state = ref({ ...initState });

  const isLogin = computed(() => !!state.value.user.username); // 로그인 여부
  const username = computed(() => state.value.user.username);   // 로그인 사용자 ID
  const email = computed(() => state.value.user.email);   // 로그인 사용자 email
```

서버로 부터 받은 인증정보

defineStore()함수 내에 정의해놓아야 프로젝트의 다른 파일에서 언제든 접근 가능함.

로그인여부/사용자id 변경될때마다 자동 계산되어 들어감.

인증 스토어

- **stores/auth.js**

```
const login = async (member) => {
  state.value.token = 'test token';
  state.value.user = {
                      username: member.username,
                      email: member.username + '@test.com' }   ;

  // api 호출
 const { data } = await axios.post('/api/auth/login', member);
 state.value = { ...data };
  localStorage.setItem('auth', JSON.stringify(state.value));
};

const logout = () => {
  localStorage.clear();
  state.value = { ...initState };
};
const getToken = () => state.value.token;
```

키

값

```
const state = ref({ ...initState });
```

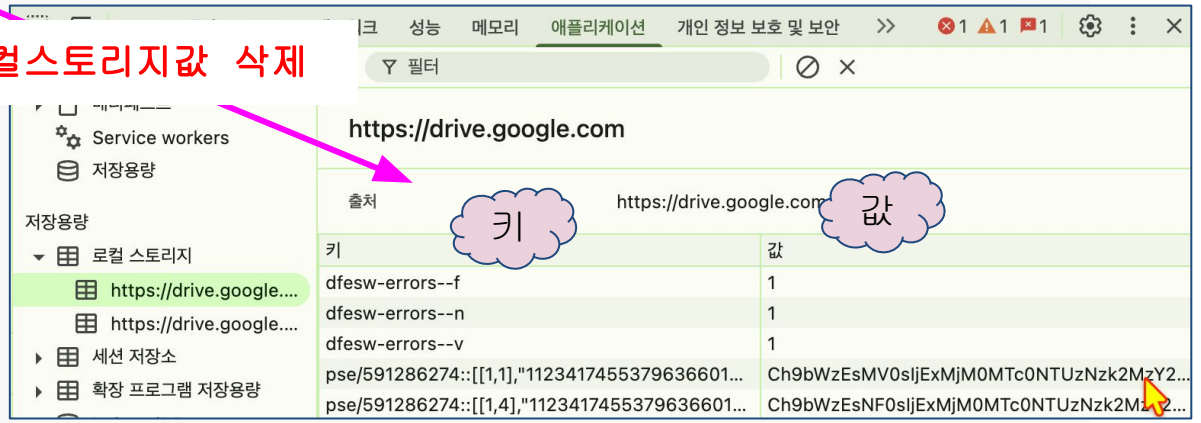
```
const initState = {
  token: '',       // 접근 토큰(JWT)
  user: {
    username: '',          // 사용자 ID
    email: '',      // Email
    roles: [],      // 권한 목록
  },
};
```

로그인시 호출할
비동기함수 정의

state값을 초기값으로
reset

token을 가지고 axios로
서버와 통신할 때
언제든 token값을 얻기
위한 함수 정의

로컬스토리지값 삭제



인증 스토어

KB 국민은행

페이지 로딩시
로컬스토리지에서
꺼내어 다시 pinia에
저장

- **stores/auth.js(git-note#56)**

```
const load = () => {
  const auth = localStorage.getItem('auth');
  if (auth != null) {
    state.value = JSON.parse(auth);
    console.log(state.value);
  }
};

load();

return { state, username, email, isLogin, login, logout, getToken };
});
```

```
localStorage.setItem('auth', JSON.stringify(state.value));
```

키: `'auth'`

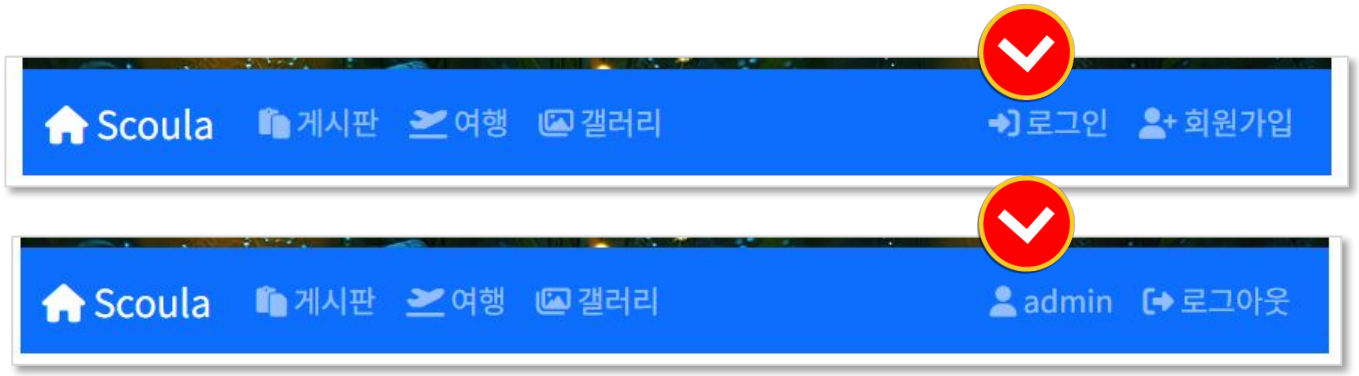
값:

```
const state = {
  token: '',                  // 접근 토큰(JWT)
  user: {
    username: '',             // 사용자 ID
    email: '',                // Email
    roles: [],                // 권한 목록
  },
};
```

- 기본적으로 Pinia에 저장된 상태(state)는 F5(페이지 새로고침)를 누르면 사라짐.
- 페이지 새로 로딩시 로컬스토리지 —> pinia 중앙저장소에 저장해주어야 함.

인증 스토어

- 로그인 페이지



인증 스토어

- **pages/auth/LoginPage.vue**

```
<script setup>
// Vue의 Composition API에서 사용하는 핵심 함수들을 임포트
import { computed, reactive, ref } from 'vue';
// 인증 관련 상태를 관리하는 Pinia 스토어 가져오기
import { useAuthStore } from '@/stores/auth';
// 라우팅 기능을 사용하기 위해 Vue Router의 useRouter 훅 가져오기
import { useRouter } from 'vue-router';



// router 인스턴스를 생성하여 페이지 이동 등에 사용
const router = useRouter();
// 인증 관련 Pinia 스토어 인스턴스를 생성하여 상태나 액션에 접근
const auth = useAuthStore();
```

인증 스토어

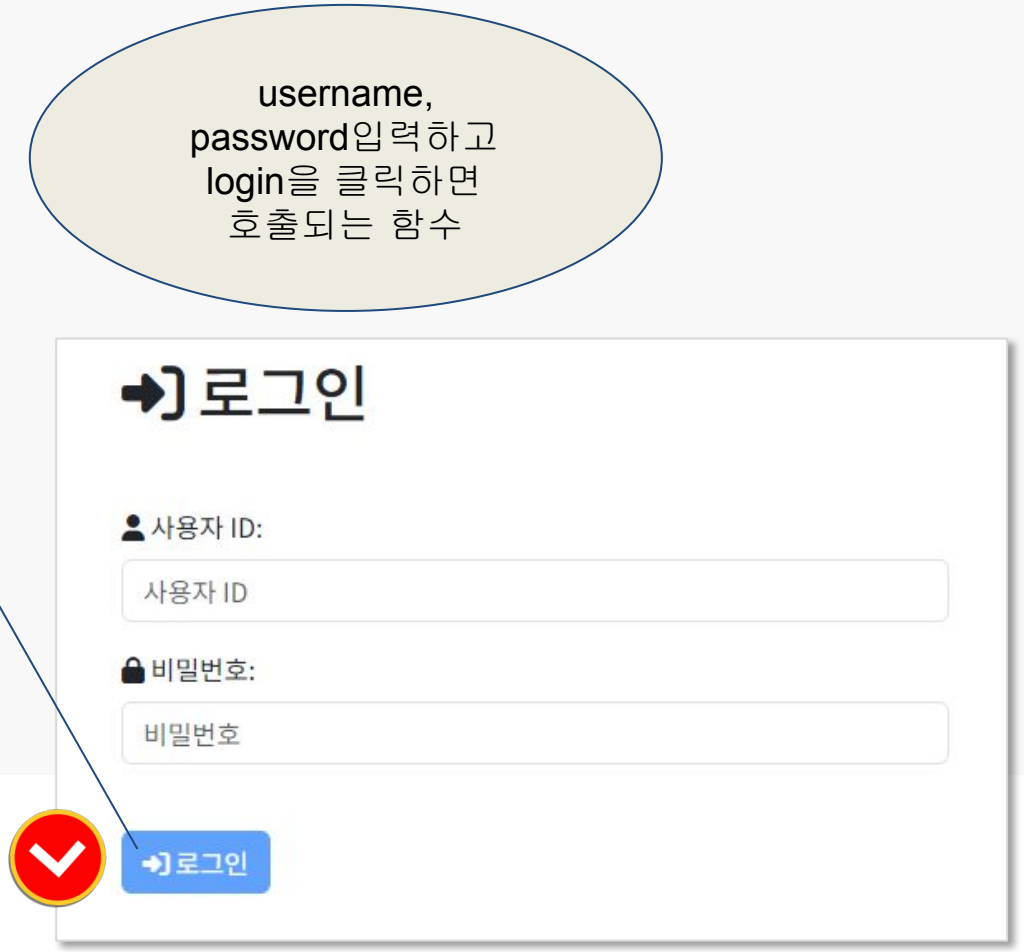
- **pages/auth/LoginPage.vue**

```
// 사용자 입력 값을 담을 반응형 객체 생성
const member = reactive({
  username: '',  // 사용자 ID 입력 필드
  password: '',  // 비밀번호 입력 필드
});

// 오류 메시지를 표시하기 위한 반응형 참조값
const error = ref('');

// '아이디와 비밀번호가 모두 입력되었는가 ?'를 판단하는 계산 속성
// 둘 다 입력되지 않으면 true → 로그인 버튼 비활성화
const disableSubmit = computed(() => !(member.username && member.password));
```

인증 스토어

- **pages/auth/LoginPage.vue**

```
const login = async () => {
  console.log(member);
  try {
    await auth.login(member);
    router.push('/');
  } catch (e) {
    // 로그인 에러
    console.log('에러=======', e);
    error.value = e.response.data;
  }
};
</script>
```



인증 스토어

- **pages/auth/LoginPage.vue**

```
<template>
  <div class="mt-5 mx-auto" style="width: 500px">
    <h1 class="my-5">
      <i class="fa-solid fa-right-to-bracket"></i>
      로그인
    </h1>

    <form @submit.prevent="login">
      <div class="mb-3 mt-3">
        <label for="username" class="form-label">
          <i class="fa-solid fa-user"></i>
          사용자 ID:
        </label>
        <input type="text" class="form-control" placeholder="사용자 ID" v-model="member.username" />
      </div>
```


```
const member = reactive({
  username: '',  // 사용자 ID 입력 필드
  password: '',  // 비밀번호 입력 필드
});
```

인증 스토어

- **pages/auth/LoginPage.vue**

```
<div class="mb-3">
  <label for="password" class="form-label">
    <i class="fa-solid fa-lock"></i>
    비밀번호:
  </label>
  <input type="password" class="form-control" placeholder="비밀번호" v-model="member.password"/>
</div>

<div v-if="error" class="text-danger">{{ error }}</div>

<button type="submit" class="btn btn-primary mt-4" :disabled="disableSubmit">
  <i class="fa-solid fa-right-to-bracket"></i>
  로그인
</button>
</form>
</div>
</template>
```

```
const member = reactive({
  username: '',  // 사용자 ID 입력 필드
  password: '',  // 비밀번호 입력 필드
});
```

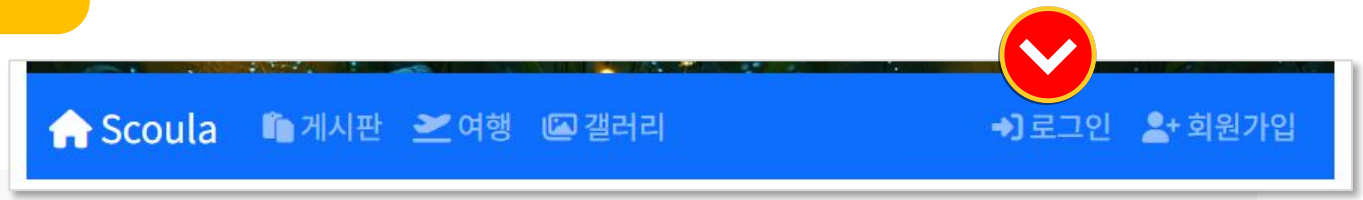
const disableSubmit = computed(() =>
!(member.username && member.password));

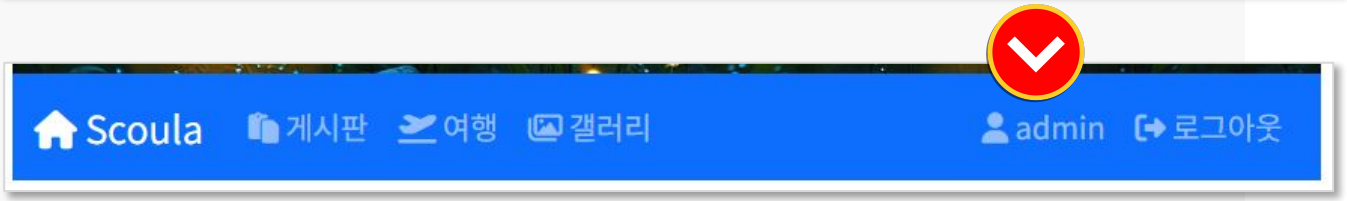
username과 pw를 모두 넣지 않으면 disableSubmit = true!로 로그인버튼 비활성

인증 스토어

- **AccountMenuGroup.vue**

```
<script setup>
…

const { login, join } = config.accoutMenus;
import { useAuthStore } from '@/stores/auth.js';

const auth = useAuthStore();

const islogin = computed(() => auth.isLogin);
const username = computed(() => auth.username);
</script>

<template>
…
</template>
```





로그인여부에 따라 계정 정보가 달라짐.
중앙저장소에서 login여부와 로그인된 username가지고 와서 계성메뉴 구성함

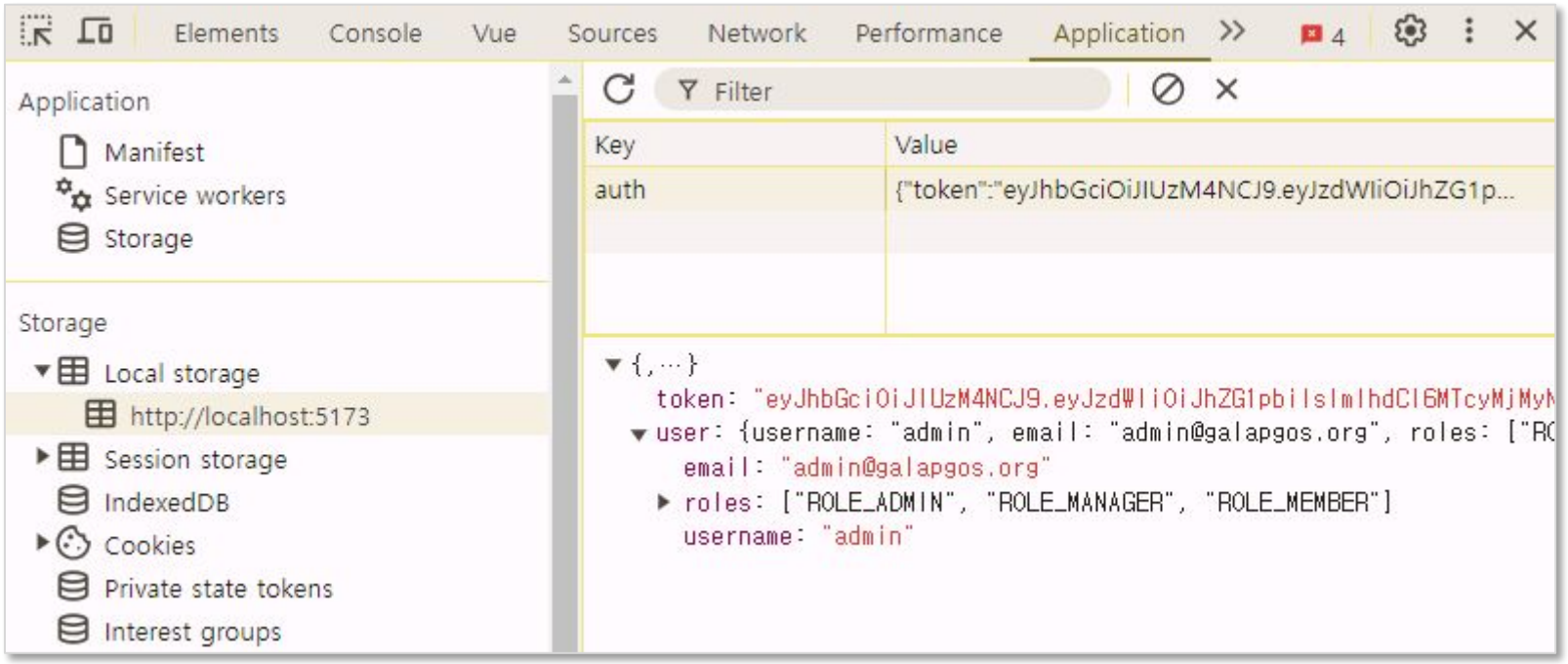
인증 스토어

- router/auth.js

```
export default [
 {
   path: '/auth/login',
   name: 'login',
   component: () => import('../pages/auth/LoginPage.vue'),
 },
 {
   path: '/auth/join',
   name: 'join',
   component: () => import('../pages/auth/JoinPage.vue'),
 },
];
```

인증 스토어

- 로그인 후 localStorage 확인



인증 스토어

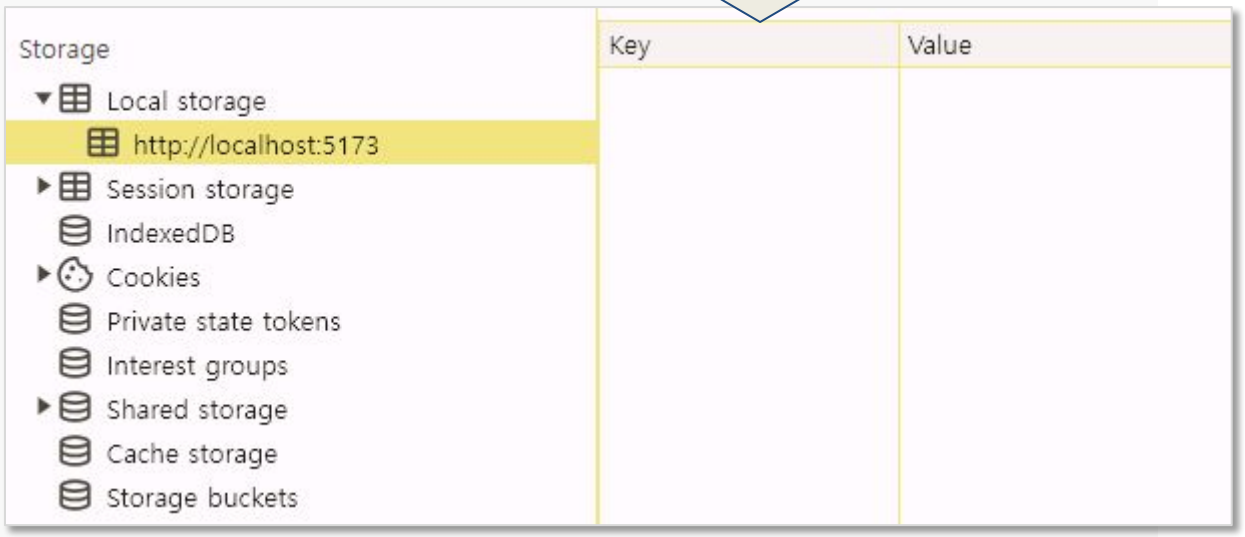
- 로그아웃
  - auth 스토어의 logout 호출
    - localStorage 클리어
    - 상태를 초기 상태로 설정

인증 스토어

- **LogoutMenuItem.vue**

```
<script setup>
import { useRouter } from 'vue-router';
import { useAuthStore } from '@/stores/auth';
const store = useAuthStore();
const router = useRouter();
const logout = (e) => {
  // 로그아웃
  store.logout();
  router.push('/');
};
</script>
<template>
  <a href="#" class="nav-link" @click.prevent="logout">
    <i class="fa-solid fa-right-from-bracket"></i>
    로그아웃
  </a>
</template>
```

로컬스토리지 삭제함



KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

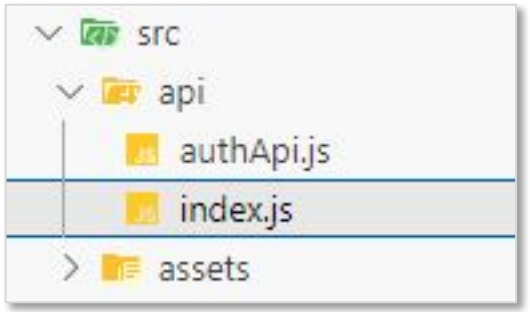
로그인 – axios 인터셉터

[KB] IT’s Your Life

KB 국민은행

- **axios 인터셉터 (interceptor)**
  - axios의 요청과 응답에 대한 필터 역할
    - 모든 요청을 가로채서 요청을 수정할 수 있음.
    - 모든 응답을 가로채서 응답을 수정할 수 있음
  - backend와의 모든 통신은 인증 토큰(jwt)를 포함해야 함
  - Axios Interceptor 운영해야 함
    - **자동으로 모든 요청의 request 헤더에 인증 헤더 추가**
      - **Authentication: Bearer <토큰 문자열>**
  - **인터셉터가 설정된 axios 인스턴스를 작성하고, 백엔드 통신시 이 인스턴스를 사용**

axios 인터셉터



[사용자 컴포넌트]

↓

axios.request(config)

↓

요청 인터셉터 (Request Interceptor)
(예: 헤더에 토큰 추가 등)

↓

📡 서버로 요청 전송

↓

🌐 서버 응답 도착

↓

응답 인터셉터 (Response Interceptor)
(예: 공통 에러 처리, 데이터 후처리)

↓

[사용자 컴포넌트로 응답 전달]

axios 인터셉터

- **api/index.js (기본 골격)**

```
import axios from 'axios';

const instance = axios.create({
  timeout: 1000,
});

// 요청 인터셉터
instance.interceptors.request.use(
  (config) => {
    // config.headers : 요청 헤더
    return config;
  },
  (error) => {
    // 요청 중 에러가 난 경우
    return Promise.reject(error);
  }
);
```

axios객체 생성

request 인터셉터 설정

모든 axios호출시 적용됨

- jwt를 http header로 포함시킴.

axios 인터셉터

- **api/index.js (기본 골격)**

```
// 응답 인터셉터
instance.interceptors.response.use(
  (response) => {
    // 정상 응답인 경우 (200, 404)

    return response;
  },
  async (error) => {
    // 에러 응답인 경우(401, 403, 305, 500 등)

    return Promise.reject(error);
  }
);

export default instance;  // 인터셉터가 적용된 axios 인스턴스
```

response 인터셉터 설정

모든 axios응답시 적용됨

- 에러에 대한 처리

axios 인터셉터

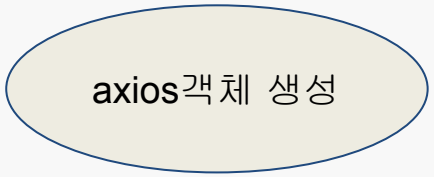
- 요청 인터셉터에서 할일
  - auth store에서 토큰을 추출
  - 토큰이 있다면 요청 헤더에 추가
    - **Authentication: Bearer <토큰 문자열>**

- 응답 인터셉터에서 할일
  - 401 에러인 경우
    - 로그인 페이지로 이동

axios 인터셉터

- **api/index.js**

```
import axios from 'axios';

import { useAuthStore } from '@/stores/auth';
import router from '@/router';

const instance = axios.create({
  timeout: 1000,
});
```



axios 인터셉터

- **api/index.js**

```
// 요청 인터셉터
instance.interceptors.request.use(
  (config) => {
    // JWT 추출
    const { getToken } = useAuthStore();
    const token = getToken();
    if (token) {
      // 토큰이 있는 경우
      config.headers['Authorization'] = `Bearer ${token}`;
      console.log(config.headers.Authorization);
    }
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);
```

요청 인터셉터

pinia에서 토큰값을 가지고 와서
header를 생성

axios 인터셉터

- **api/index.js**

```
// 응답 인터셉터
instance.interceptors.response.use(
  (response) => {
    if (response.status === 200) {
      return response;
    }
    if (response.status === 404) {
      return Promise.reject('404: 페이지 없음 ' + response.request);
    }
    return response;
  },
  async (error) => {
    if (error.response?.status === 401) {
      const { logout } = useAuthStore();
      logout();
      router.push('/auth/login?error=login_required');
      return Promise.reject({ error: '로그인이 필요한 서비스입니다.' });
    }
    return Promise.reject(error);
  }
);
export default instance;
```

응답 인터셉터

응답과 응답 에러처리

- 200 : 받은 데이터 전달
- 404 : 404페이지없음 전달
- 401 : logout()후, 로그인페이지로 push
  로그인이 필요한 서비스입니다. 전달

axios 인터셉터

- **실제 api 서버로 로그인하기**
  - axios를 이용해서 '/api/auth/login' 요청

axios 인터셉터

- **store/auth.js**

```
import { ref, computed, reactive } from 'vue';
import { defineStore } from 'pinia';
import axios from 'axios';
…

export const useAuthStore = defineStore('auth', () => {
  …
 const login = async (member) => {
   // state.value.token = 'test token';
   // state.value.user = { username: member.username, email: member.username + '@test.com' }   ;

   // api 호출
   const { data } = await axios.post('/api/auth/login', member);
   state.value = { ...data };
   localStorage.setItem('auth', JSON.stringify(state.value));
 };
 …
});
```

axios 인터셉터

- **localStorage 확인**

KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

로그인 – 아바타 처리 (백엔드)

[KB] IT’s Your Life

KB 국민은행

아바타 처리(백엔드)

- **UploadFiles.java**

```
package org.scoula.common.util;
…
public class UploadFiles {
    …
    public static void downloadImage(HttpServletResponse response, File file) {
        try {
            Path path = Path.of(file.getPath());
            String mimeType = Files.probeContentType(path);
            response.setContentType(mimeType);
            response.setContentLength((int) file.length());

            try (OutputStream os = response.getOutputStream();
                 BufferedOutputStream bos = new BufferedOutputStream(os)) {
                 Files.copy(path, bos);
            }
        }
        catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

아바타 처리 (백엔드)

- **MemberController.java**

```
public class MemberController {
    …

    @GetMapping("/{username}/avatar")
    public void getAvatar(@PathVariable String username, HttpServletResponse response) {
        String avatarPath = "c:/upload/avatar/" + username + ".png";
        File file = new File(avatarPath);
        if(!file.exists()) {  // 아바타 등록이 없는 경우, 디폴트 아바타 이미지 사용
            file = new File("C:/upload/avatar/unknown.png");
        }

        UploadFiles.downloadImage(response, file);
    }
}
```

KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

로그인 – 아바타 처리 (프론트엔드)

[KB] IT’s Your Life

KB 국민은행

아바타 처리 (프론트엔드)

- **assets/main.css**

```
…
.avatar {
  border-radius: 50%;
  border-color: gray;
  background-color: #d8d8d8;
}

.avatar-sm {
  width: 20px;
  height: 20px;
}

.avatar-md {
  width: 40px;
  height: 40px;
}

.avatar-lg {
  width: 80px;
  height: 80px;
}
```

아바타 처리 (프론트엔드)

- **AccountMenuItem.vue**

```
<script setup>
const props = defineProps({ username: String });

const avatar = `/api/member/${props.username}/avatar`;
</script>

<template>
  <li class="nav-item">
    <router-link class="nav-link" to="/auth/profile">
      <img :src="avatar" class="avatar avatar-sm" />
      {{ username }}
    </router-link>
  </li>
</template>
<style></style>
```