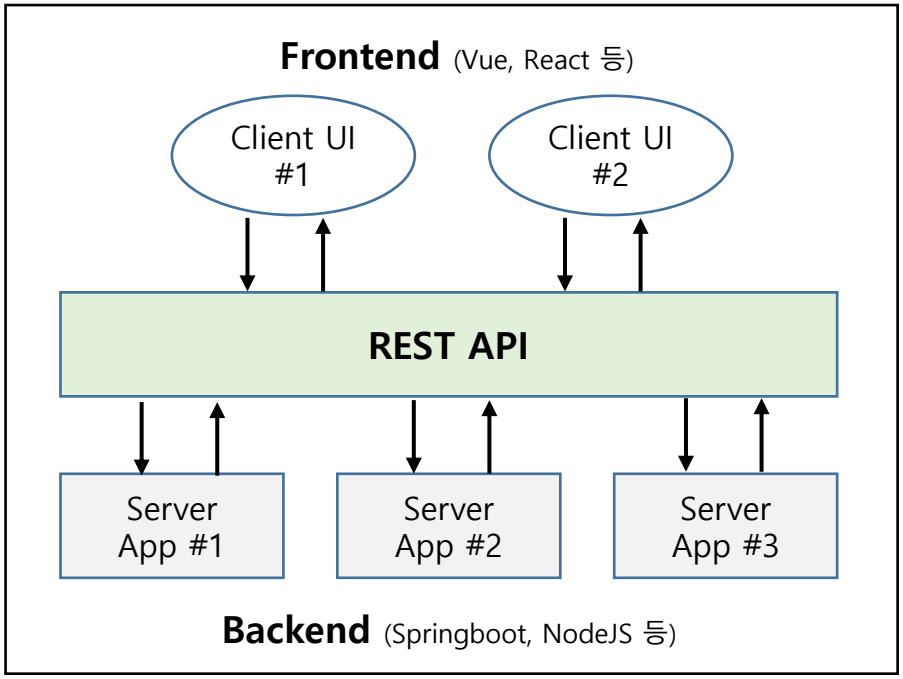
웹 UI 프레임워크 개요

FE/BE가 분리된 서비스 환경에서 웹 서비스 공통관심사항을 처리할 수 있는 독립된 UI 프레임워크가 필요

API 기반 웹 서비스

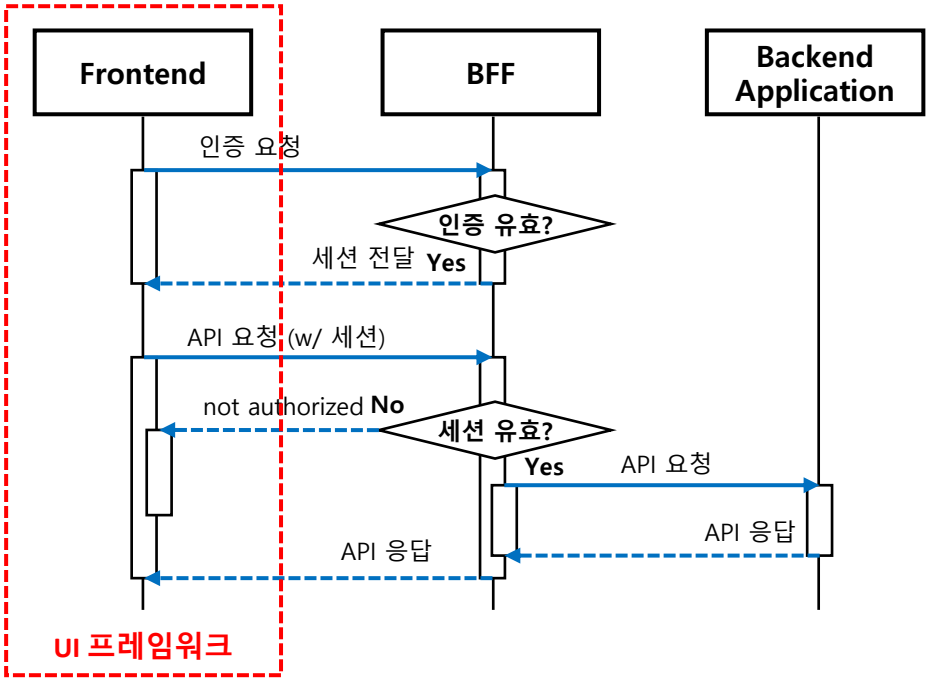
➢ API 기반 FE/BE 어플리케이션이 분리된 서비스 환경



- Frontend와 Backend가 분리되는 어플리케이션 아키텍처에서 가장 중요한 산출물은 API
- API는 Backend 어플리케이션의 최종 구현체이자 Frontend 어플리케이션이 UI를 렌더링 하기 위한 정보 그 자체

분산 환경 웹 서비스 공통관심사항 처리

"BFF는 분산환경에서 frontend 단위로 웹 서비스 공통관심사항을 제공하는 backend 어플리케이션 "



- 화면에 필요한 정보가 FE 영역에 없기 때문에 웹 서비스 공통관심사항(인증/세션/권한)을 처리하는 BFF가 필요
- UI 프레임워크는 최소한의 BFF API 사용을 전제로, 임의의 서비스에 활용 가능한 Frontend UI 어플리케이션 개발에 필요한 도구와 환경을 제공

Vite 패키저를 활용하여 Vue 프레임워크와 typescript를 기반으로 UI 어플리케이션을 구현

개발 환경

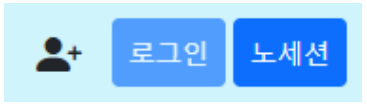
➢ 어플리케이션 개발 도구

개발도구	설명
Node(NPM)	Javascript 런타임 환경 및 모듈 패키지 (의존성) 관리
VS Code	Web UI 개발 IDE

- 프레임워크 실행 (https://github.com/lsmin625/vue-base-ui 다운로드)
- 패키지 실행 후 로그인 - "노세션"은 DEV 환경에서 BFF 로그인 없이 메인 화면 진입용

<프레임워크 패키지 실행>

```
$ npm install
$ npm run dev
```



✓ Private NW 환경에서의 NPM Repository 설정 (참고)

- 인터넷의 NPM Repository에 직접 연결할 수 없는 경우 .npmrc 파일에 NPM 프록시 접속 정보를 세팅
  - 윈도 사용자 → C:₩Users₩<your_user_id>₩.npmrc
  - 맥 사용자 → Users/<your_user_id>/.npmrc

```
registry=http://yourcompanyproxy.com
//yourcompanyproxy.com:username=...
//yourcompanyproxy.com:_password=...
strict-ssl=true
```

- 또는 npm config를 통해 세팅 가능

```
npm config set @npmcorp:registry https://your-company-nexus:80/nexus/content/repository/npm-internal
```

Why Vite? Why Vue?

❖ UI 프레임워크 고려사항

- 대규모 웹 어플리케이션을 개발하는 프로젝트
  → 다양한 기술레벨(초급/중급/고급)이 섞인 개발자가 참여
- 프레임워크 학습시간(learning curve)과 어플리케이션 운영을 고려한 일관성 있는 코드 스타일
- 공통 컴포넌트 구현 및 활용, 빌드/배포 성능

➢ Vite 패키저

1. 개발을 진행하면서 UI 기능 검증을 위해 사용하는 DEV 서버의 속도가 빠름.
2. 편집하는 소스코드에 대한 HMR(Hot Module Replacement)이 브라우저의 기능을 통해 빠르게 반응

➢ Vue 프레임워크

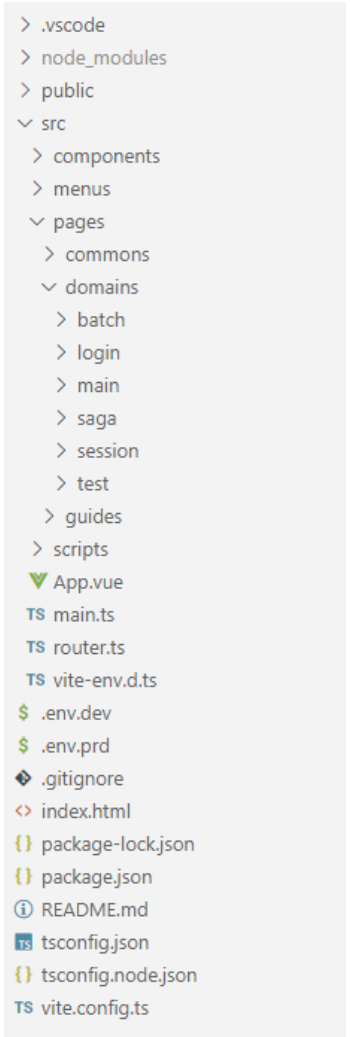
1. 직관적인 구조와 명확한 문법으로 학습시간이 작다.
2. 공식 문서가 매우 상세하고 이해하기 쉽게 작성
3. 독립적인 컴포넌트의 재사용성이 높아 유지보수 및 확장성에 유리
4. 공식 라이브러리를 통한 효과적인 어플리케이션 관리

※ 서비스 디자인(CSS, JS)이 선행되어야 UI 프레임워크에 필요한 기본 컴포넌트 구현이 가능하기 때문에 Bootstrap 스타일을 활용함

어플리케이션 패키지 구조

UI 프레임워크의 특성과 개발자 협업 구조를 고려한 어플리케이션 패키지 구조

디렉토리/파일	설명	비고
index.html	어플리케이션 진입점(entry point)	"/src/main.ts" 를 module 타입의 script로 지정
package.json	프로젝트 메타데이터와 npm 패키지 목록 정보	package-lock.json : npm 패키지 일관성 유지
tsconfig.json	TypeScript 컴파일러의 설정 정의	
tsconfig.node.json	주로 개발 환경에서 Node.js와 관련된 컴파일 설정	
vite.config.ts	Vite의 설정 파일(플러그인, 빌드 옵션, 경로 별칭 등)	
.env{.runtime}	런타임 환경 변수 정의 (--mode dev)	import.meta.env.MODE 및 "VITE_USER_XXXX"로 구분
public/	브라우저에 바로 제공되는 정적(static) 파일 폴더	3rd 파티 라이브러리 설정 위치
favicon.ico		
src/		
App.vue	어플리케이션 루트 Vue 컴포넌트	
main.ts	index.html에서 지정하는 어플리케이션 진입점	CSS, 기본 라우팅 및 공통 컴포넌트 등록
router.ts	기본 라우팅 정보	
components/	입력, 버튼, 셀렉트 등 공통 컴포넌트	main.ts에서 글로벌 컴포넌트로 등록(import 불필요)
teleports/	모든 화면에서 호출할 수 있는 팝업 또는 모달 형식의 컴포넌트	<body> 태그에 연결
menus/	그룹(도메인) 단위로 설정하는 메뉴 목록	사용자 그룹 접근 권한 연계
pages/		
commons/	도메인 공통 서비스 컴포넌트	API와 연계한 공통 서비스 제공
domains/	도메인 단위 하위 폴더 구성 및 화면 컴포넌트 작성	화면 단위 컴포넌트는 menu에 등록
guides/	공통 컴포넌트 개발 가이드	
scripts/	API 호출, 세션 및 teleports 호출 등의 스크립트 유틸리티	



※ 어플리케이션 패키지 구조는 개발 환경에 따라 변경 가능하나 기본 골격은 유지되어야 한다.

개발자 협업 구조

소스 코드가 중복으로 작업되지 않도록 개발자 별로 명확한 코딩 영역(폴더/파일) 구분이 필요

Gitflow 기반 브랜치 관리 정책

1) 신규 feature 브랜치 생성 (네비게이션 메뉴 Git 패널 선택)

1) "SOURCE CONTROL : ... > Branch > Create Branch..." 선택
2) 새로운 브랜치명 입력 : "feature/sample-241028"

2) 코드 작성

1) 소스 코드 작성 (로컬 repository)
2) Commit 메시지 작성
3) 원격 repository로 push (신규 브랜치인 경우 "Publish Branch" 클릭)

3) 브랜치 Merge (feature → develop)

1) "SOURCE CONTROL : ... > Checkout to..." 선택
2) develop 브랜치로 이동 (로컬 repository)
3) 원격 repository의 develop 브랜치 pull 요청 (다른 개발자 작업 내용 업데이트)
4) "SOURCE CONTROL : ... > Branch > Merge Branch..." 선택
5) develop 브랜치와 merge 할 feature 브랜치 선택 : "feature/ sample-241028"
6) merge가 완료된 로컬 develop 브랜치를 원격지로 push ("Sync Changes" 클릭)

변경되는 공통 모듈 협업 방안

❖ 여러 개발자가 동일 어플리케이션에서 개별 feature 브랜치로 작업중 공통 모듈이 업데이트 되는 시나리오

- 업무 개발자 : "feature/order-241130"에서 작업중
- 공통 개발자 : "feature/common-241202"에서 공통 모듈 업데이트 완료

1) 공통 개발자 develop 브랜치로 업데이트된 공통 모듈 merge 하기

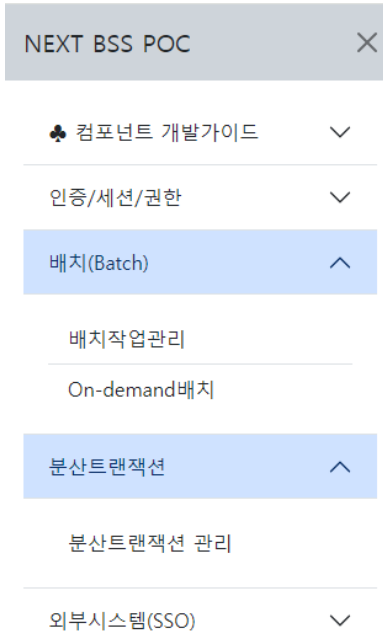
1) 로컬 develop 브랜치로 체크아웃 후 "pull" 요청으로 브랜치 변경사항 업데이트
2) feature 브랜치를 develop 브랜치로 merge
3) "Push"를 통해 원격 develop 브랜치에 작업 내용 반영(feature 브랜치 merge 내용)

2) 업무 개발자 작업중인 feature 브랜치로 공통 모듈 반영 하기

1) 작업중인 "feature/order-241130" 커밋/푸시 진행 후 develop 브랜치로 체크아웃
2) develop 브랜치에서 "pull"을 통해 원격 develop 변경 사항을 로컬에 적용
3) "feature/order-241130" 브랜치로 다시 체크아웃 후 develop 브랜치를 "merge"
4) 현재 브랜치(feature/order-241130)에서 남은 작업 계속 진행

개발 환경에서 서버 의존 없이 UI 독립적으로 화면을 구현할 수 있는 메뉴 구성

➢ 메뉴 구조 예시 (2단 형식)




```
export default {
  groupId: "batch-menus",
  groupName: "배치(Batch)",
  menuList: [
    {
      menuId: "/main/batch",
      menuName: "배치(Batch)",
      hidden: true,
      path: "/main/batch",
      component: () => import(`@/pages/domains/batch/BatchMain.vue`)
    },
    {
      menuId: "/main/batch",
      menuName: "배치작업관리",
      path: "/main/batch/BatchSchedules",
      component: () => import(`@/pages/domains/batch/.../BatchSchedules.vue`)
    },
    {
      menuId: "/main/batch",
      menuName: "On-demand배치",
      path: "/main/batch/OndemandBatch",
      component: () =>
import(`@/pages/domains/batch/components/OndemandBatch.vue`),
    },
  ]
}
```

메뉴 속성		설명
groupId		메뉴그룹 ID
groupName		메뉴그룹 명
menuList		메뉴 목록
	menuId	메뉴 ID
	menuName	메뉴 명
	hidden	메뉴 노출 여부
	path	routing 경로
	component	화면 컴포넌트
	authLevel	접근권한 (서버 제공)

- 메뉴ID(menuId)는 서버의 사용자그룹 접근 권한의 메뉴ID와 동일하게 관리
- hidden은 메뉴 노출 없이 라우팅 경로를 설정 (목록화면 → 상세화면 호출)
- authLevel을 -1로 설정하면 DEV 런타임에서 항상 노출 (접근 권한 무시)

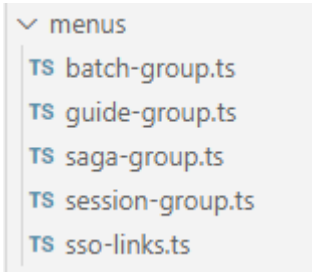
※ 메뉴 구조를 **3단**으로 설정하는 경우 도메인(domain ID, name)에 그룹목록(groupList)을 추가하는 형태로 확장 가능

➢ 메뉴 관리 기준

- UI 화면의 메뉴 체계는 서버와 독립적으로 관리 → 서버에서 사용자그룹의 메뉴 접근 권한을 제공하기 위해서는 메뉴 ID를 기준으로 권한 레벨을 제공
- 접근 권한이 없는 메뉴 그룹정보는 노출되지 않으며, 화면 라우팅 경로도 생성되지 않음. → hidden 설정을 통해 메뉴로 노출되지 않는 화면 컴포넌트 생성 가능
- 메뉴 ID는 중복 가능하며 화면 라우팅 경로(path)와 화면 컴포넌트(component)는 고유하게 관리
- 접근 권한(권한 없음 : 0, 읽기: 1, 쓰기: 2, 관리: 3)에 따라 화면 노출 및 버튼 비활성화 제어

메뉴 폴더(menus)에 그룹 단위로 파일을 작성하고 메뉴 스토어에 등록

➢ 메뉴 그룹 구조



<메뉴 폴더>

```
export default {
  groupId: "session-menus",
  groupName: "인증/세션/권한",
  menuList: [
    {
      menuId: "/main/session",
      menuName: "인증/세션/권한",
      hidden: true,
      path: "/main/session",
      component: () => import(`@/pages/.../SessionMain.vue`)
    },
    {
      menuId: "/main/menu",
      menuName: "메뉴관리",
      path: "/main/menu/MenuPane",
      component: () => import(`@/pages/.../MenuPane.vue`)
    },
    {
      menuId: "/main/menu",
      menuName: "사용자그룹 메뉴관리",
      path: "/main/menu/UserGroupMenu",
      component: () => import(`@/pages/.../UserGroupMenu.vue`)
    },
...
```

```
import guideGroup from "@/menus/guide-group";
import sessionGroup from "@/menus/session-group";
import batchGroup from "@/menus/batch-group";
import sagaGroup from "@/menus/saga-group";
import ssoLinks from "@/menus/sso-links";

export const menuGroups = [] as MenuGroup[];
menuGroups.push(guideGroup as MenuGroup);
menuGroups.push(sessionGroup as MenuGroup);
menuGroups.push(batchGroup as MenuGroup);
menuGroups.push(sagaGroup as MenuGroup);
menuGroups.push(ssoLinks as MenuGroup);
```

<메뉴 스토어>

➢ 그룹 단위 메뉴 파일 작성 방법

① 메뉴 폴더에 메뉴 그룹 파일을 생성하고, 메뉴 그룹에 메뉴(ID, name, path, component) 목록을 작성

② 메뉴 ID는 동일한 접근 권한 특성을 가진 메뉴에 동일하게 설정하고, 화면 라우팅 경로(path)와 컴포넌트(component)는 고유하게 관리

③ 화면 컴포넌트가 작성되지 않고 메뉴만 등록할 경우에는 component에 null 지정 (import 설정 금지)

④ 메뉴로 노출되지는 않지만 화면을 통해 호출되는 화면은 hidden으로 설정하고, DEV 에서 서버에 권한 등록없이 화면을 구현하는 경우 authLevel을 -1로 설정

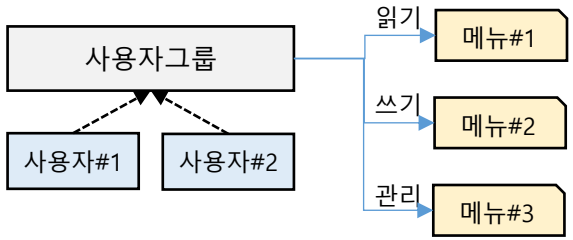
⑤ 메뉴 스토어(store-menus.ts)에 그룹 파일을 등록

메뉴 접근 권한 관리

서버에서 관리되는 사용자그룹의 메뉴권한목록을 통해 UI에서 노출하는 메뉴(화면)를 제어

사용자그룹 메뉴 접근 권한과 UI 화면

➢ 사용자그룹의 UI 메뉴 접근 권한 (서버 관리)



권한	코드	비고
없음	0	
읽기	1	
쓰기	2	읽기 포함
관리	3	쓰기 포함

- 사용자는 사용자그룹에 포함되고, 사용자그룹 단위로 메뉴 접근 권한 설정
- UI에서 로그인할 때 응답 데이터로 메뉴권한목록(authMenus)을 전달
- 메뉴권한목록을 기준으로 화면에 노출할 메뉴와 화면 라우팅 경로를 구성


```
{
    "result": 0,
    "code": 0,
    "message": null,
    "body": {
        "userId": "gdhong",
        "userGroupId": "guest",
        "userName": "홍길동",
        "userEmail": "gdhong@sk.com",
        "userPhone": "01099881234",
        "sessionId": "ebe099dc-ea91-4973-9f97-db58e37d3246",
        "authMenus": [
            {
                "systemId": "rally",
                "systemName": "랠리(RALLY)",
                "menuId": "/main/batch",
                "menuName": "배치(Batch)",
                "authLevel": 2
            }
        ],
```


<로그인 응답 데이터>

사용자그룹에 따른 메뉴 노출(예시)

➢ 로그인 응답으로 제공된 메뉴권한목록 기준으로 노출

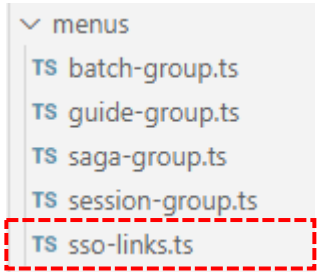
<"Admin" 사용자그룹>

<"Guest" 사용자그룹>

외부 시스템 메뉴 링크

사용자ID와 세션ID를 통해 JWT로 연동하는 외부 시스템의 메뉴 관리

➢ 메뉴 그룹 구조



<메뉴 폴더>

```ts
export default {
  groupId: "sso-links",
  groupName: "외부시스템(SSO)",
  menuList: [
    {
      menuId: "/sso/oas-portal",
      menuName: "OAS포탈(SSO)",
      path: "/sso/oas-portal",
      url: "http://localhost:5174/sso"
    }
  ]
}
```

<외부 시스템 메뉴>

```ts
import guideGroup from "@/menus/guide-group";
import sessionGroup from "@/menus/session-group";
import batchGroup from "@/menus/batch-group";
import sagaGroup from "@/menus/saga-group";
import ssoLinks from "@/menus/sso-links";

export const menuGroups = [] as MenuGroup[];
menuGroups.push(guideGroup as MenuGroup);
menuGroups.push(sessionGroup as MenuGroup);
menuGroups.push(batchGroup as MenuGroup);
menuGroups.push(sagaGroup as MenuGroup);
menuGroups.push(ssoLinks as MenuGroup);
```

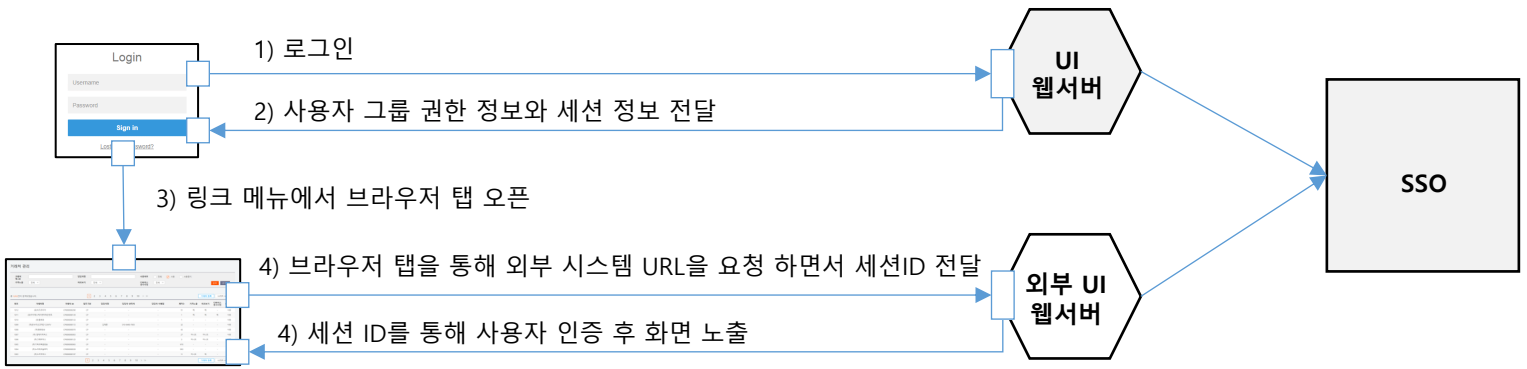
<메뉴 스토어>

➢ 외부 시스템 링크 메뉴 관리

① 메뉴 ID는 일반 메뉴와 동일하게 서버의 사용자그룹 접근 권한의 메뉴 ID와 동일하게 관리

② 화면 컴포넌트(component)를 지정하지 않고 해당 메뉴의 호출을 받을 외부 시스템의 URL을 지정

③ 외부 시스템 메뉴는 브라우저의 새로운 탭으로 열리며, 시스템이 SSO를 지원하는 경우 제공된 세션 정보를 통해 JWT로 계정 공유



도메인 단위로 독립적으로 하위 폴더를 생성하여 화면 컴포넌트를 구현

➢ 화면 폴더 구조

pages/	
commons/	도메인 공통 서비스 컴포넌트
domains/	도메인 단위 하위 폴더 구성 및 화면 컴포넌트 작성
main/	메인 화면 영역
components/	메인 화면에서 사용하는 공통 컴포넌트
MainMain.vue	POC 어플리케이션 메인 화면 (하위 라우팅 구성)
guides/	가이드 화면 영역
components/	가이드 화면에서 사용하는 공통 컴포넌트
forms/	(구분 필요시) 폼 개발 가이드 관련 화면과 컴포넌트
modals/	(구분 필요시) 모달 개발 가이드 관련 화면과 컴포넌트
OptionRadioGuide.vue	개별 가이드 화면 컴포넌트

- 소스코드 형상관리를 위해 도메인 폴더 및 도메인 하위 폴더는 개발(자) 단위로 구분되어야 한다.
- 폴더 이름은 한 단어로 소문자로 명명하고, 컴포넌트 파일의 이름은 두 단어 이상 Camel 표기법을 권고한다.
- 폴더와 파일의 이름은 직관적으로 구분되며 의미를 알 수 있도록 생성한다.

➢ 화면 컴포넌트 개발

① Vue Composition API 방식으로 타입스크립트로 구현한다.

② 개별 컴포넌트를 import 하는 경우 태그는 Camel 표기법으로 태그를 작성한다.

③ 공통 컴포넌트는 별도의 import 선언 없이 Camel 표기법으로 사용한다.

④ 업무 공통 컴포넌트와 스크립트는 절대경로("@/")로 지정하고, 도메인 컴포넌트는 상대경로("./")로 지정하는 것이 편리하다.

```vue
<script setup lang="ts">
import BatchSchedules from './components/BatchSchedules.vue';
import OndemandBatch from './components/OndemandBatch.vue';
</script>

<template>
  <div class="container-fluid">
    <div class="row">
      <div class="col border-0 border-start border-end mt-2">
        <BatchSchedules />
        <OndemandBatch class="bg-body-tertiary" />
      </div>
    </div>
  </div>
</template>
```

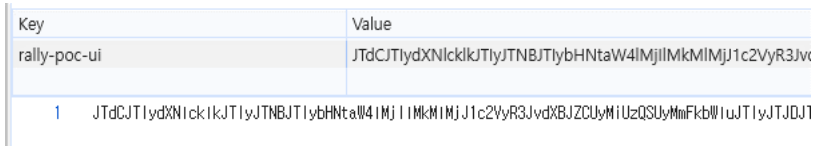
사용자 정보(로그인 세션)를 통한 권한 관리와 쿠키를 통한 API 요청 세션 유지

사용자 정보 접근

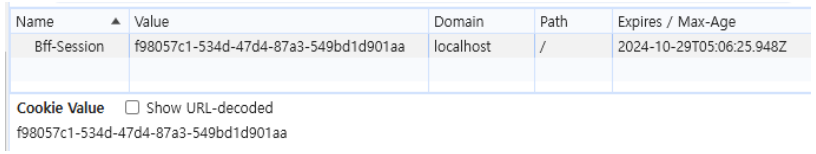
➢ 브라우저 로컬 스토리지 데이터와 쿠키 세션 비교(유효성 검사)

- 기본적으로 프레임워크에서 제공되는 상태 관리는 메모리 기반으로 휘발성
  → DEV에서 컴포넌트를 수정하면 상태정보 초기화로 세션 재생성(로그인)이 필요
- 로그인에 의해 생성된 세션을 브라우저 로컬 스토리지에 저장
- 컴포넌트가 초기화 되면 저장된 세션 정보와 쿠키의 세션ID를 비교하여 유효한 경우 세션 상태를 restore 처리

<브라우저 로컬 스토리지에 저장한 세션 정보 – UI 저장>



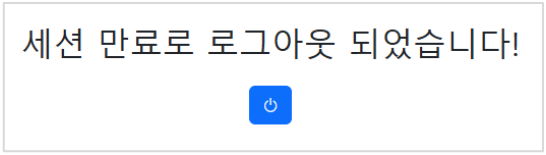
<브라우저 쿠키에 저장된 세션 ID – 서버 전달>



API 요청/응답 처리

➢ API 요청/응답 절차의 일관성 유지

- API를 요청하고 응답 데이터를 기다리는 동안 스피너를 통한 작업 진행 표시
- NW 또는 API 에러 발생 시 팝업 모달 호출 → 성공 시 응답 데이터 전달
- API 처리 과정세서 세션 만료 및 인증 오류 발생 시 로그아웃 처리




```
import { useUserSession, hasAuthMenu } from '@/scripts/session’;

const userSession = useUserSession()

const openTab = () => {
  window.open(`http://localhost:5174/sso?userId=${userSession.value?.userId}&sessionId=${userSession.value?.sessionId}`, '_blank');
}

<li v-if="hasAuthMenu('/main/session')" class="nav-item">
    <a class="nav-link" :class="isActive('session')" href="#" @click="goto('session’)”>인증/세션/권한</a>
</li>
```

REST API 요청 및 응답 처리는 표준 라이브러리 스크립트를 통해 관리

➢ 표준 API 호출 라이브러리 (apiCall)

- 웹 표준 API인 fetch를 사용하여 REST API에서 필요한 GET, POST, PUT, DELETE 및 파일 업로드/다운로드를 위한 함수를 제공
- API 요청에서 데이터 응답 과정에 필요한 일련의 작업을 일괄적으로 처리
- HTTP 일반 에러 및 API 표준 응답을 반영하여 에러 발생 시 오류 메시지 팝업을 자동 노출 (직접 에러 처리를 하고자 하는 경우 just~ 함수 사용)

get, post, put, delete	라이브러리에서 에러 처리
justGet, justPost, justPut, justDelete	화면 단위로 직접 에러 처리
download, upload	파일 다운로드, 업로드

➢ API 표준응답 (Response)

Property	Type	설명
result	int	API 응답 결과 (성공: 0, 실패: 1)
code	int	결과가 실패인 경우 에러 코드
message	string	결과가 실패인 경우 에러 메시지
body	object	API 응답 데이터 (데이터가 없는 경우 null)

<로그인 API 호출>

```
import apiCall from '@/scripts/api-call’

const account = reactive({
  userId: '',
  userPassword: ''
})

const login = async () => {
  const inputs = [
    { label: '사용자ID', value: account.userId },
    { label: '비밀번호', value: account.userPassword },
  ]
  if (validateAndNotify(inputs)) {
    const url = '/api/account/login'
    const { body: session } = await apiCall.post(url, null, account)
    if (session) {
      setUserSession(session)
    }
  }
}
```

<파일 다운로드 및 업로드 API 호출>

```
import apiCall from '@/scripts/api-call’

const downloadExcel = async () => {
    const url = `/api/apis/excel/download?keyword=${keyword.value}`
    await apiCall.download(url, null, null, `api-download-${getTimestamp()}`)
}

const selectFile = async () => {
    const file = fileInput.value?.files?.[0]
    if (file) {
        notifyConfirm('API 목록을 업로드 할까요?', async (confirmed: boolean) => {
            if (confirmed) {
                const url = '/api/apis/excel/upload'
                const response = await apiCall.upload(url, null, file)
                if (response.result === apiCall.Response.SUCCESS) {
                    notifySuccess('목록이 업로드 되었어요.')
                }
            }
        })
    }
}
```

※ API URL은 상대 경로로 호출 (개발 환경에는 Vite의 proxy를 통해 상용 환경에서는 웹서버의 reverse proxy 설정을 통해 CORS 해결)

목록형 API 페이징 처리

목록형 데이터는 페이지 단위 처리를 위해 API 요청과 응답에서 페이지 처리에 필요한 속성을 전달

➢ 목록형 API 요청 파라미터

Property	Type	설명
count	int	페이지당 수신하고자 하는 데이터의 개수
offset	int	페이지의 오프셋 (0부터 시작)

➢ 목록형 API 응답 : 페이지 구분 목록 객체 (PagedList)

Property	Type	설명
total	int	조회 대상 데이터의 총 개수 (총 개수가 미정인 경우 나머지 존재 -1, 미존재 0)
count	int	해당 페이지에서 전달되는 데이터의 개수
offset	int	이번 페이지의 오프셋 (0부터 시작)
list	array	조회된 데이터 객체의 배열 (Array구조)

<목록형 데이터 페이징 처리 화면 예시>

<목록형 API 호출 및 응답 데이터 페이징 처리 스크립트>

```
const page = reactive({
    total: 0,
    current: 1,
    count: 10,
})

const getCodeGroups = async () => {
    table.items.length = 0

    const url = '/api/code-group/list'
    const queryParams = {
        keyword: keyword.value,
        count: page.count,
        offset: page.current - 1
    }
    const { body: pagedList } = await apiCall.get(url, null, queryParams)
    if (pagedList) {
        page.total = pagedList.total
        page.current = pagedList.offset + 1
        table.items = pagedList.list
        ascendArray(table.items, 'codeGroupName')
        setSequence(table.items, (page.current - 1) * page.count + 1)
    }
}
```

<목록형 데이터(테이블) 및 페이징 컴포넌트 처리 예시>

```
<ItemsTable :headers="table.headers" :items="table.items“>
...
</ItemsTable>
<PageNavigator :totalCount="page.total" v-model:current="page.current" v-
model:count="page.count" />
```

<총 개수가 미정인 데이터 페이징 처리를 위한 버튼 구현 예시>

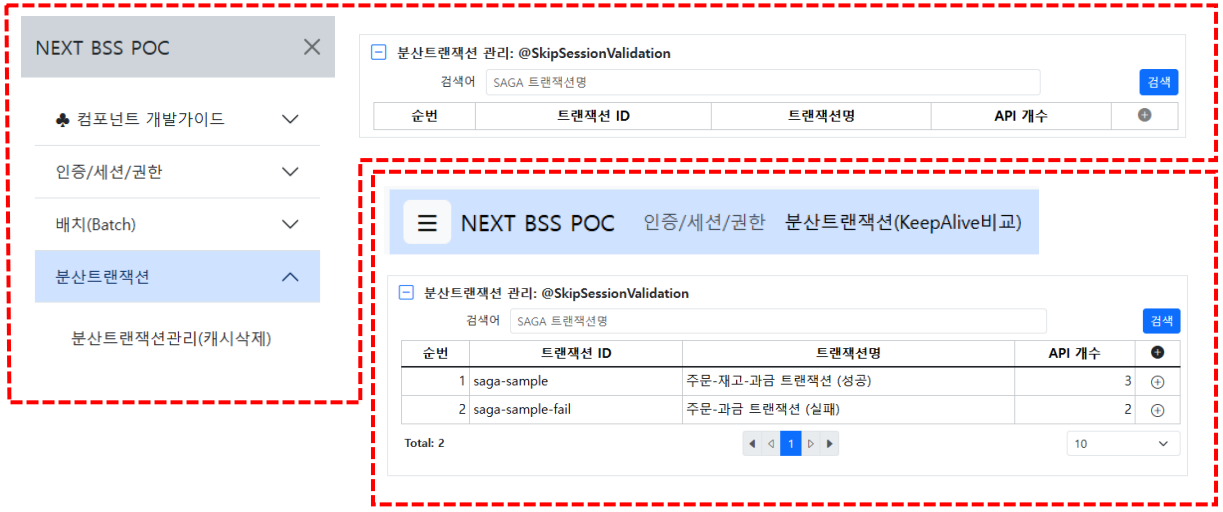
```
<button v-if="page.total === -1" class="btn btn-sm btn-outline-primary"
@click="appendBatchHistory"><i class="bi bi-caret-down"></i></button>
```

KeepAlive 빌트인 컴포넌트를 통한 작업 화면 데이터 유지(캐시) 및 선별적 캐시 삭제 처리

➢ 선별적 화면 데이터 유지 또는 삭제 처리

- 화면이 전환될 때 이전 화면에서 작업중인 데이터(입력 및 결과)의 유지 또는 삭제 처리를 선별적으로 관리
- 예시 : 사이드 메뉴를 통해 화면이 노출되는 경우 초기화(캐시 삭제)하고 네비게이션 탭을 통해 노출되는 경우는 데이터 유지(캐시 유지)

<사이드 메뉴를 통한 화면 호출 (캐시 삭제)>



<네비게이션 탭을 통한 화면 호출 (캐시 유지)>

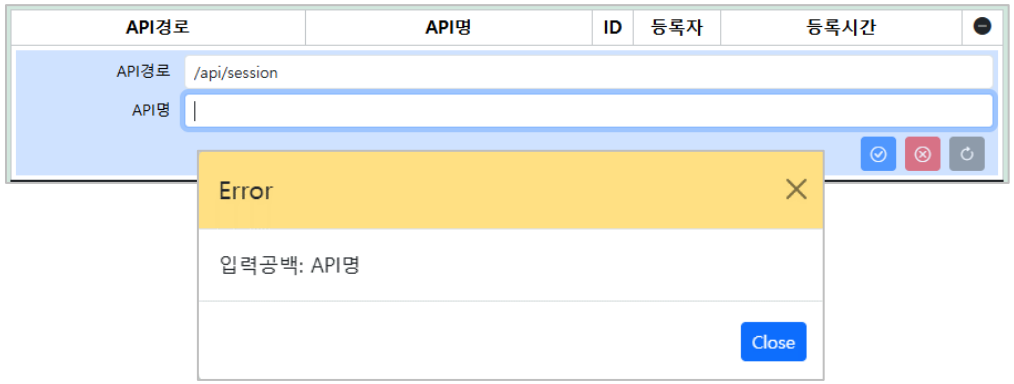
<화면 노출 요청 컴포넌트(caller) 판단>

```
import { isCalledByMenu } from '@/scripts/store-cache'

const AUTH_MENU = '/main/saga'

onActivated(() => {
    if (isCalledByMenu()) {
        // clear cache
        keyword.value = ''
        table.items.length = 0
        page.total = 0
        page.current = 1
    }
})
```

입력 데이터 검증 (스타일 의존)

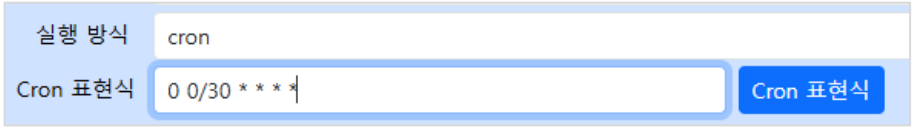
필수 입력 항목에 대한 검증 방식

➢ 팝업 처리를 통한 필수 항목 검증




```
import { validateAndNotify } from '@/scripts/validater'

const emitSave = () => {
    const inputs = [
        { label: 'API경로', value: watchTargets.apiPath },
        { label: 'API명', value: watchTargets.apiName },
    ]
    if (validateAndNotify(inputs)) {
        const setting = {} as ApiSetting
        setting.id = props.setting.id
        setting.apiPath = watchTargets.apiPath
        setting.apiName = watchTargets.apiName

        emit('save', setting)
    }
}
```

➢ 입력 폼 스타일 변경을 통한 데이터 형식 검증



<입력 데이터 형식 유효>

<입력 데이터 형식 오류>

```
import { isValidCronExpression } from '@/scripts/batch'

<InlineInput class="mb-1" label="Cron 표현식" v-model="watchTargets.jobCron" type="text"
    placeholder="Cron Expression" :warning="!isValidCronExpression(watchTargets.jobCron)" :append="true">
    <button class="btn btn-sm btn-primary" @click="cronExpressionModal.show()">
      Cron 표현식
    </button>
</InlineInput>
```

정규 표현식을 활용한 입력 데이터 형식 검증

➢ 입력 유형 검증

검증 유형	검증을 위한 정규 표현식 (RegExp)	비고
비밀번호	/^(?=.*[a-zA-Z0-9])(?=.*[a-zA-Z!@#$%^&*₩₩(₩₩)_+-=])(?=.*[0-9!@#$%^&*₩₩(₩₩)_+-=]).{10,15}$/	영문/숫자/특수 문자 조합 10~15자
이메일	/^[a-zA-Z0-9]([-_.]?[a-zA-Z0-9])*@[a-zA-Z0-9]([-_.]?[a-zA-Z0-9])*.[a-zA-Z]{2,3}$/	
유선전화	/^(0[2-8][0-5]?)-?([1-9]{1}[0-9]{2,3})-?([0-9]{4})$/	숫자 입력 검증 후 형식 변환 적용
무선전화	/^(01[01346-9])-?([1-9]{1}[0-9]{2,3})-?([0-9]{4})$/	
주민번호	/^([0-9]{6})-?([1-4]{1}[0-9]{6})$/	
사업자번호	/^([0-9]{3}) -?([0-9]{2})-?([0-9]{5})$/	
영문숫자	/^[a-zA-Z0-9]+$/	
한글	/^[ㄱ-힣]+$/	
숫자	/^[0-9]+$/	
돈	/^[0-9,.]$/	

➢ 입력 형식 변환 – 주민 번호 예시

```
let trimmedInput = input.trim().replaceAll( ‘-’ , ‘’ )
trimmedInput.replace(/(₩d{6})(₩d{7}), ‘$1-$2’ )
```

모달 컴포넌트 호출 (스타일 의존)

팝업 형식의 모달 컴포넌트는 텔레포트(teleport)로 등록되어 스크립트를 통해 호출

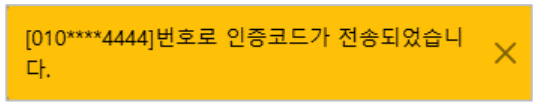
➢ 메시지 유형별로 스타일이 구분된 메시지 알림 모달 컴포넌트

호출 함수	설명
notifySuccess	이벤트 처리 성공 메시지
notifyInfo	이벤트 공지 메시지
notifyError	이벤트 처리 에러 메시지
notifyConfrim	사용자 의사(확인/취소)에 따른 후속 작업 처리

<notifyConfirm을 통한 모달 컴포넌트 호출>

```
import { notifyConfirm } from '@/scripts/store-popups'

const startJob = async () => {
    notifyConfirm(`[${props.operation.jobName}] 작업을 시작 할까요?`, async (yes: boolean) => {
        if (yes) {
            const requestBody = { ...props.operation }
            requestBody.jobParameters = null
            const url = '/batch/api/operation/start'
            await apiCall.post(url, null, requestBody)
        }
    })
}
```

➢ 토스트 팝업 스타일의 모달 컴포넌트 (팝업 3초 유지후 사라짐)




```
import {popToast } from '@/scripts/store-popups'

popToast(`[${userSetting.value.userPhone}]번호로 인증코드가 전송되었습니다.`)
```

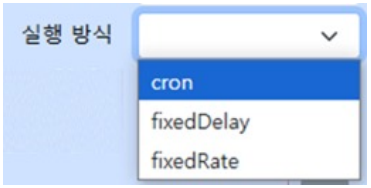
공통코드를 통한 선택 목록 구성 (스타일 의존)

셀렉트 박스 또는 라디오 버튼의 목록 구성을 API 호출을 통해서도 구성 가능

➢ 목록 선택 컴포넌트 구성

- 셀렉트 박스, 라디오 버튼 그룹 등의 컴포넌트의 목록을 직접 지정하거나 API를 통해 전달된 데이터를 이용해서 구성 가능
- 공통코드 API를 사용하는 경우 코드그룹ID를 지정하면 컴포넌트 내부적으로 목록이 관리 될 수 있도록 구현

<일반 옵션 셀렉트>




```
const jobTypes = ref(['cron', 'fixedDelay', 'fixedRate'])

<OptionSelect class="mb-1" label="실행 방식" :values="jobTypes" v-model="watchTargets.jobType" />
```

<라디오 버튼 그룹>


```
const jobStyles = reactive([
    { label: '일반', value: 'normal' },
    { label: '병렬', value: 'parallel' },
    { label: '쉘', value: 'shell' },
    { label: 'OnD', value: 'ondemand' }
])

<OptionRadio :options="jobStyles" v-model="watchTargets.jobStyle" />
```

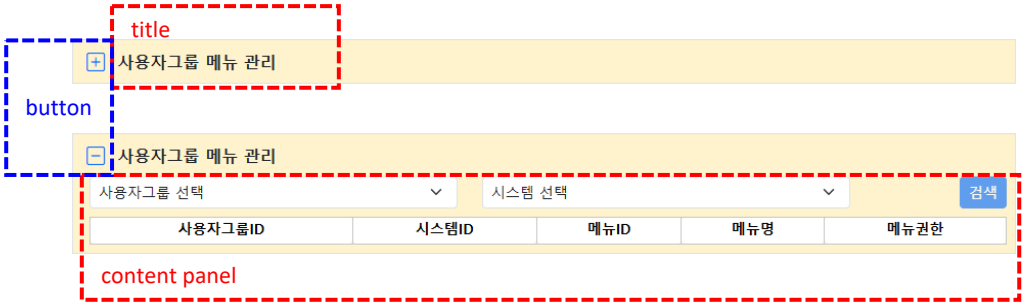
<공통코드 옵션 셀렉트>


```
const userGroup = reactive({
    codeGroupId: 'user_auth_group',
    defaultOption: { label: '사용자그룹 선택', value: '' },
    id: '',
    selected: {} as any
})

<div class="col-4">
  <OptionSelectByCode :codeGroupId="userGroup.codeGroupId" v-model="userGroup.id"
      :defaultOption="userGroup.defaultOption" @optionSelected="selectUserGroup" :withValue="true" />
</div>
```

스타일 의존적인 공통 컴포넌트 구현 방법 (예시)

UI 디자인이 확정되면 이를 기반으로 재사용성을 고려하여 공통 컴포넌트로 구현

➢ PanelCollapse 컴포넌트 구현 예시



- 컴포넌트의 프로퍼티(props)로 title을 지정
- 버튼 토글 이벤트(toggled)를 부모 컴포넌트에게도 전달하기 위한 emits 지정하고, watchEffect를 통해 상태 변동 시 이벤트 전달
- Toggle 버튼이 대상 컴포넌트를 개별적으로 구분하기 위해 elementId를 고유한 key로 생성
- 내용(content)로 들어갈 임의의 컴포넌트는 slot으로 지정

```
<script setup lang="ts">
import { ref, watchEffect } from 'vue';

const props = defineProps<{
    title: string;
}>()

const emit = defineEmits<{
 (event: 'toggled', value: boolean): void
}>()

const toggle = ref(false)
const elementId = crypto.randomUUID()

watchEffect(() => {
    emit('toggled', toggle.value)
})
</script>
```

```
<template>
    <div class="container-fluid mt-1 border p-0">
        <div class="d-flex justify-content-start align-items-center">
            <button class="btn btn-link" type="button" data-bs-toggle="collapse" :data-bs-target="'#' + elementId"
                @click="toggle = !toggle">
                <i :class="['bi', toggle ? 'bi-dash-square' : 'bi-plus-square']" style="font-size: 1.1em;"></i>
            </button>
            <h6 class="fw-bold m-0">{{ props.title }}</h6>
        </div>
        <div class="collapse ms-2 p-2 pt-0" :id="elementId">
            <slot></slot>
        </div>
    </div>
</template>
```

<PanelCollapse 사용>

```
<PanelCollapse title="사용자그룹 메뉴 관리" @toggled="toggle">
  <div>슬롯으로 설정한 컴포넌트가 보입니다.</div>
</PanelCollapse>
```