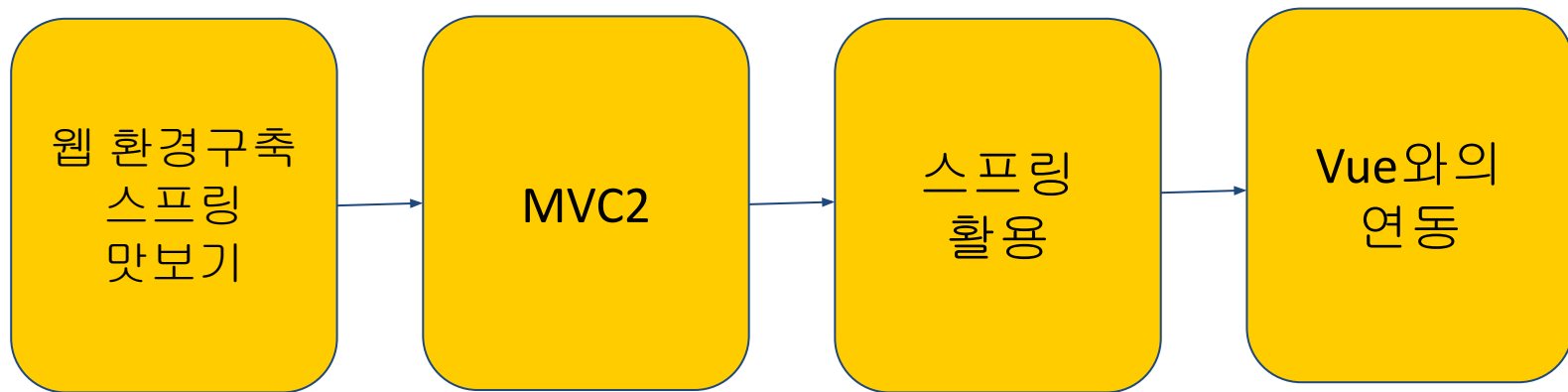


2024년 상반기 K-디지털 트레이닝

SPRING10 - Vue연동 (JWT+Paging)

[KB] IT's Your Life



+ 회원 가입

사용자 ID : ID 중복 확인 사용가능한 ID입니다.

아바타 이미지:

파일 선택 선택된 파일 없음

email

비밀번호:

비밀번호 확인:

+ 확인

로그인

사용자 ID:

비밀번호:

로그인



게시글 목록

(총 5건)

No	제목	작성자	작성일
5	테스트 제목5	user00	2024-08-25
4	테스트 제목4	user0	2024-08-25
3	테스트 제목3	user0	2024-08-25
2	테스트 제목2	user0	2024-08-25
1	테스트 제목1	user0	2024-08-25

◀ 1 ▶

글 작성

Vue연동

frontend-vue
VSCODE

```
▼ FRONTEND
> .vscode
> node_modules
> public
▼ src
  > api
  > assets
  > components
  > config
  > pages
  > router
  > stores
  > util
  ▼ App.vue
  JS main.js
  .gitignore
  <> index.html
  {} jsconfig.json
  {} package-lock.json
  {} package.json
  ⓘ README.md
  JS vite.config.js
```

backend-spring
intellij

```
▼ backend C:\Wkb_spring\Wscoula-fulls
> .gradle
> .idea
> build
> gradle
> out
▼ src
  ▼ main
    ▼ java
      ▼ org.scoula
        > board
        > common
        > config
        > controller
        > exception
        > member
        > security
      > resources
      > webapp
    > test
  .gitignore
  build.gradle
  database.sql
```

mysql8
workbench

```
▼ shop2
  ▼ Tables
    tbl_board
    tbl_board_attachment
    tbl_member
    tbl_member_auth
  Views
  Stored Procedures
  Functions
```

database

- 간단한 회원 관리 시스템과 게시판 기능을 구현

1. 데이터베이스 생성 및 사용

- 데이터베이스 생성: `CREATE DATABASE shop2;`
- 데이터베이스 사용: `USE shop2;`

2. 테이블 생성 및 데이터 삽입

- 사용자 정보 테이블 (`tbl_member`):
 - 컬럼: `username(PK)`, `password`, `email`, `reg_date`, `update_date`
 - 설명: 사용자 정보 저장
 - 데이터 삽입: 관리자 및 일반 사용자 정보 삽입
- 사용자 권한 테이블 (`tbl_member_auth`):
 - 컬럼: `username(FK, PK)`, `auth(PK)`
 - 설명: 사용자 권한 정보 저장 (복합키로 구성)
 - 데이터 삽입: 각 사용자에게 대해 권한 삽입
- 게시판 테이블 (`tbl_board`):
 - 컬럼: `no(PK)`, `title`, `content`, `writer`, `reg_date`, `update_date`
 - 설명: 게시글 정보 저장
 - 데이터 삽입: 테스트용 게시글 데이터 삽입
- 첨부파일 테이블 (`tbl_board_attachment`):
 - 컬럼: `no(PK)`, `bno(FK)`, `filename`, `path`, `content_type`, `size`, `reg_date`
 - 설명: 게시글의 첨부파일 정보 저장

3. 데이터 조회

- 사용자 조회: `SELECT * FROM tbl_member;`
- 사용자 권한 조회: `SELECT * FROM tbl_member_auth ORDER BY auth;`
- 게시글 조회: `SELECT * FROM tbl_board;`
- 첨부파일 조회: `SELECT * FROM tbl_board_attachment;`

4. 테이블 삭제

- 테이블 삭제: `DROP TABLE IF EXISTS tbl_member_auth, tbl_member, tbl_board, tbl_board_attachment;`

- database.sql

```

create database shop2;
use shop2;

drop table if exists tbl_member_auth;
drop table if exists tbl_member;

-- 사용자 정보 테이블
create table tbl_member
(
    username      varchar(50) primary key,      -- 사용자 id
    password      varchar(128) not null,        -- 암호화된 비밀번호
    email         varchar(50) not null,
    reg_date      datetime default now(),
    update_date   datetime default now()
);

-- 사용자 권한 테이블
create table tbl_member_auth
(
    username      varchar(50) not null,        -- 사용자 id
    auth          varchar(50) not null,        -- 권한 문자열 ROLE_ADMIN, ROLE_MANAGER, ROLE_MEMBER 등
    primary key(username, auth),              -- 복합키
    constraint fk_authorities_users foreign key (username) references tbl_member(username)
);

-- 테스트 사용자 추가
insert into tbl_member(username, password, email)
values
    ('admin', '2a$10$EsIMfxbJ6NuvwX7MDj4WqOYFzLU9U/lddCyn0nic5dFo3VfJYrXYC', 'admin@galapgos.org'),
    ('user0', '2a$10$EsIMfxbJ6NuvwX7MDj4WqOYFzLU9U/lddCyn0nic5dFo3VfJYrXYC', 'user0@galapgos.org'),
    ('user1', '2a$10$EsIMfxbJ6NuvwX7MDj4WqOYFzLU9U/lddCyn0nic5dFo3VfJYrXYC', 'user1@galapgos.org'),
    ('user2', '2a$10$EsIMfxbJ6NuvwX7MDj4WqOYFzLU9U/lddCyn0nic5dFo3VfJYrXYC', 'user2@galapgos.org'),
    ('user3', '2a$10$EsIMfxbJ6NuvwX7MDj4WqOYFzLU9U/lddCyn0nic5dFo3VfJYrXYC', 'user3@galapgos.org'),
    ('user4', '2a$10$EsIMfxbJ6NuvwX7MDj4WqOYFzLU9U/lddCyn0nic5dFo3VfJYrXYC', 'user4@galapgos.org');

select * from tbl_member;

```

- database.sql

```
-- 사용자 권한 추가
insert into tbl_member_auth (username, auth)
values
    ('admin', 'ROLE_ADMIN'),
    ('admin', 'ROLE_MANAGER'),
    ('admin', 'ROLE_MEMBER'),
    ('user0', 'ROLE_MANAGER'),
    ('user0', 'ROLE_MEMBER'),
    ('user1', 'ROLE_MEMBER'),
    ('user2', 'ROLE_MEMBER'),
    ('user3', 'ROLE_MEMBER'),
    ('user4', 'ROLE_MEMBER');

select * from tbl_member_auth order by auth;
```

- database.sql

```
DROP TABLE IF EXISTS tbl_board;
```

```
CREATE TABLE tbl_board
(
    no            INTEGER AUTO INCREMENT PRIMARY KEY ,
    title         VARCHAR(200) NOT NULL ,
    content       TEXT ,
    writer        VARCHAR(50) NOT NULL ,
    reg_date      DATETIME DEFAULT CURRENT_TIMESTAMP,
    update_date   DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

```
INSERT INTO tbl_board(title, content, writer)
VALUES
('테스트 제목1', '테스트 내용1', 'user0'),
('테스트 제목2', '테스트 내용2', 'user0'),
('테스트 제목3', '테스트 내용3', 'user0'),
('테스트 제목4', '테스트 내용4', 'user0'),
('테스트 제목5', '테스트 내용5', 'user0');
```

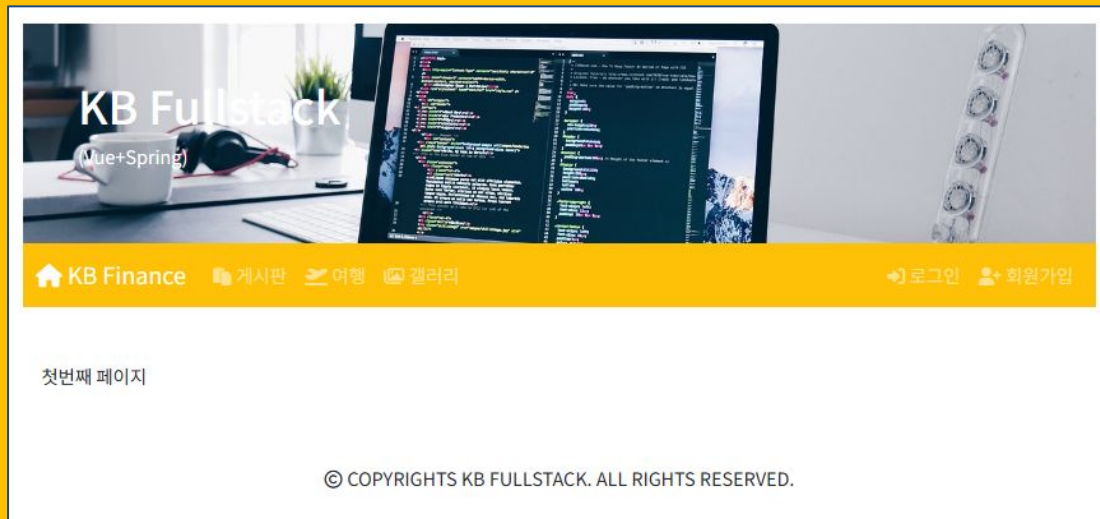
```
SELECT * FROM tbl_board;
```

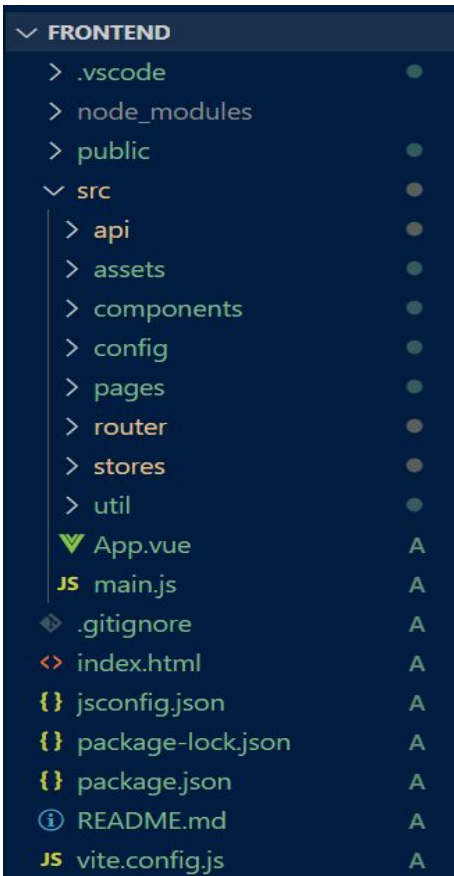
```
DROP TABLE IF EXISTS tbl_board_attachment;
```

```
CREATE TABLE tbl_board_attachment
(
    no INTEGER AUTO_INCREMENT PRIMARY KEY,
    bno INTEGER NOT NULL,           -- 게시글 번호, FK
    filename VARCHAR(256) NOT NULL, -- 원본 파일 명
    path VARCHAR(256) NOT NULL,     -- 서버에서의 파일 경로
    content_type VARCHAR(56),       -- content-type
    size INTEGER,                  -- 파일의 크기
    reg_date DATETIME DEFAULT now(),
    CONSTRAINT FOREIGN KEY(bno) REFERENCES tbl_board(no)
);
```

```
SELECT * FROM tbl_board_attachment
```

frontend





- **api** : 함수를 이용한 내부처리 및 backend api호출
- **assets** : static 파일
- **components/layouts** : layout components
- **config** : site title, subtitle, menu별 data정리 (title, url, icon)
- **pages** : template을 포함하는 ui 및 기본 computed()/유효성 검사
- **router** : routing 주소
- **stores** : 토큰 저장(로컬스토리지 이용)
- **util** : 가드 설정

- npm i
- npm i bootstrap@5 axios moment
- npm run build
- npm run dev

- package.json

```
{} package.json A ×
{} package.json > {} devDependencies
1  {
2    "name": "frontend",
3    "version": "0.0.0",
4    "private": true,
5    "type": "module",
6    "scripts": {
7      "dev": "vite",
8      "build": "vite build",
9      "preview": "vite preview"
10   },
11   "dependencies": {
12     "axios": "^1.7.5",
13     "bootstrap": "^5.3.3",
14     "moment": "^2.30.1",
15     "pinia": "^2.1.7",
16     "vue": "^3.4.21",
17     "vue-awesome-paginate": "^1.2.0",
18     "vue-router": "^4.3.0"
19   },
20   "devDependencies": {
21     "@vitejs/plugin-vue": "^5.0.4",
22     "vite": "^5.2.8"
23   }
24 }
```

- vite.config.js

```
JS vite.config.js > [?] default > [?] build
1  import { fileURLToPath, URL } from 'node:url';
2
3  import { defineConfig } from 'vite';
4  import vue from '@vitejs/plugin-vue';
5
6  // https://vitejs.dev/config/
7  export default defineConfig({
8    plugins: [vue()],
9    resolve: {
10     alias: {
11       '@': fileURLToPath(new URL('./src', import.meta.url)),
12     },
13   },
14   server: {
15     proxy: {
16       '/api': {
17         target: 'http://localhost:8080',
18         // changeOrigin: true,
19       },
20     },
21   },
22   build: {
23     outDir: 'C:/kb_spring/scoula-fullstack-0829/scoula/backend/src/main/webapp/resources',
24   },
25 });
```



- main.js

```
import './assets/main.css';
import 'bootstrap/dist/css/bootstrap.css';
import 'vue-awesome-paginate/dist/style.css';

import { createApp } from 'vue';
import { createPinia } from 'pinia';
import VueAwesomePaginate from 'vue-awesome-paginate';

import App from './App.vue';
import router from './router';

const app = createApp(App);

app.use(VueAwesomePaginate);
app.use(createPinia());
app.use(router);

app.mount('#app');
```

- App.vue

```
<script setup>
import { RouterView } from 'vue-router';
import DefaultLayout from './components/layouts/DefaultLayout.vue';
</script>

<template>
  <DefaultLayout>
    <RouterView />
  </DefaultLayout>
</template>

<style scoped></style>
```

CreateApp import

- **Router Import:** 라우터를 사용하기 위한 설정
- **Pinia Import:** 상태 관리용 Pinia 설정(인증정보 저장)
- **Paginate Import:** 페이지네이션 기능 설정

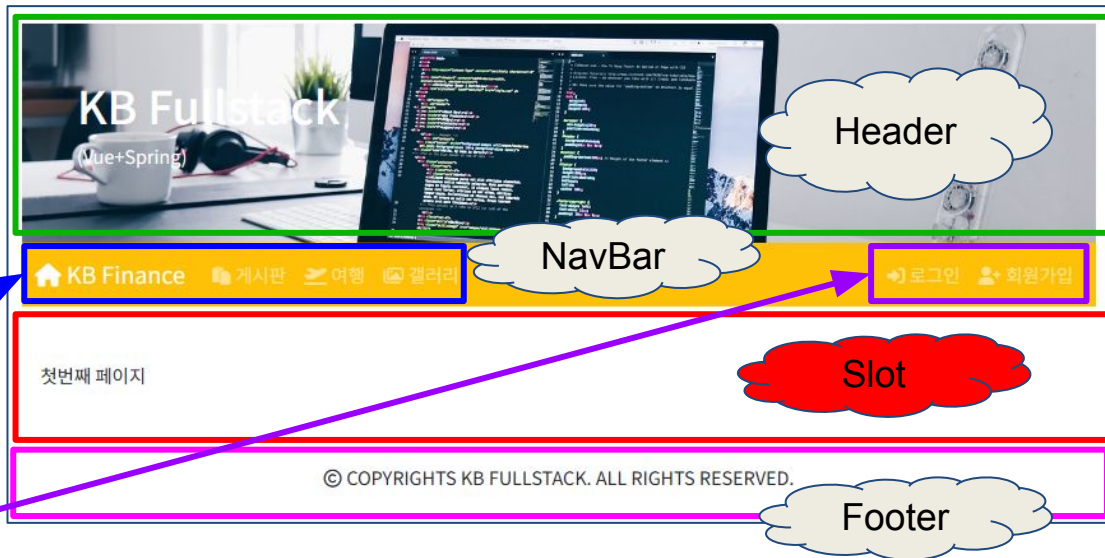
1. Layout (전체구조)

○ Default Layout

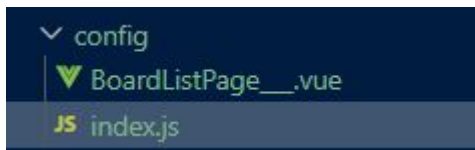
- Header (Title)
- NavBar (Menu)
- Slot (전체 내용 표시)
- Footer (Copyright)

2. Menu Structure:

- MenuGroup
 - MenuItem
- AccountMenuGroup
 - AccountMenuItem
- LogoutMenuItem

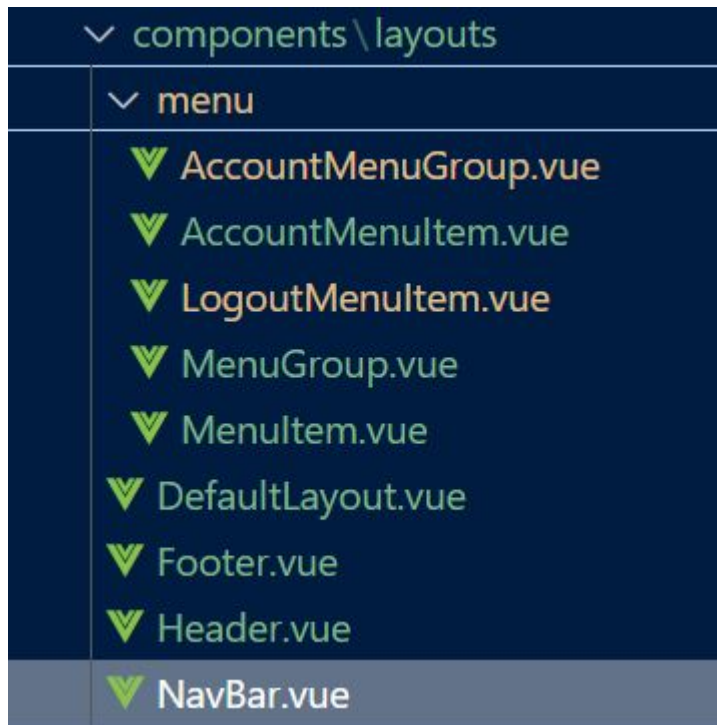


- config/index.js : site title, subtitle, menu별 data정리 (title, url, icon)



```
export default {
  title: 'KB Fullstack',
  subtitle: '(Vue+Spring)',
  menus: [
    {
      title: '게시판',
      url: '/board/list',
      icon: 'fa-solid fa-paste',
    },
    {
      title: '여행',
      url: '/travel/list',
      icon: 'fa-solid fa-plane-departure',
    },
    {
      title: '갤러리',
      url: '/gallery/list',
      icon: 'fa-regular fa-images',
    },
  ],
  accoutMenus: {
    login: {
      url: '/auth/login',
      title: '로그인',
      icon: 'fa-solid fa-right-to-bracket',
    },
    join: {
      url: '/auth/join',
      title: '회원가입',
      icon: 'fa-solid fa-user-plus',
    },
  },
};
```

- DefaultLayout.vue



```
<script setup>
import Header from './Header.vue';
import NavBar from './NavBar.vue';
import Footer from './Footer.vue';
</script>

<template>
  <div class="container">
    <Header />
    <NavBar />
    <div class="content my-5 px-3">
      <slot></slot>
    </div>
    <Footer />
  </div>
</template>
```

- AccountMenuGroup.vue

```
<script setup>
import MenuItem from './MenuItem.vue';
import AccountMenuItem from './AccountMenuItem.vue';
import config from '@config';
import { useAuthStore } from '@stores/auth';
import LogoutMenuItem from './LogoutMenuItem.vue';
import { computed } from 'vue';

const { login, join } = config.accountMenus;
const auth = useAuthStore();

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

</script>

<template>
  <ul class="navbar-nav ms-auto">
    <template v-if="islogin">
      <AccountMenuItem :username="username" />
      <LogoutMenuItem />
    </template>
    <template v-else>
      <MenuItem :menu="login" />
      <MenuItem :menu="join" />
    </template>
  </ul>
</template>
```

- 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">
      
      {{ username }}
    </router-link>
  </li>
</template>
<style></style>
```

- LogoutMenuItem.vue

```
<script setup>
import { useAuthStore } from '@/stores/auth';
import { useRouter } from 'vue-router';

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>
```

1. Imports

- **useAuthStore**: 인증 관련 상태 관리 스토어를 가져옴.
- **useRouter**: Vue Router를 사용하여 라우팅 기능을 제공함.

2. 설정

- **store**: **useAuthStore()**를 통해 인증 스토어 인스턴스를 생성.
- **route**: **useRouter()**를 통해 라우터 인스턴스를 생성.

3. 로그아웃 함수 ('logout')

- **store.logout()**: 로그아웃 동작 수행.
- **router.push('/')**: 로그아웃 후 메인 페이지로 리다이렉트.

4. 템플릿

- **<a>** 태그에 클릭 이벤트가 바인딩되어 있으며, 로그아웃 함수를 호출함.
- 폰트 어썸 아이콘과 함께 "로그아웃" 텍스트가 표시됨.

- MenuGroup.vue

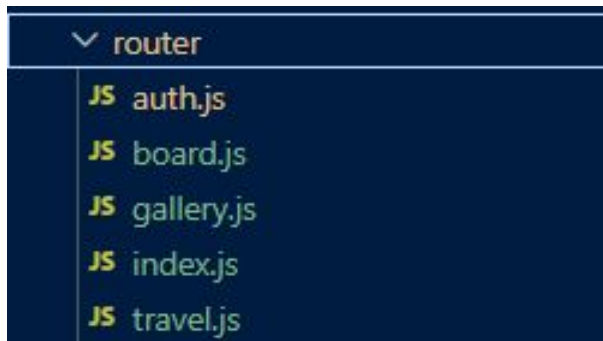
```
<script setup>
import MenuItem from './MenuItem.vue';
const props = defineProps({
  menus: { Type: Array, required: true },
});
</script>

<template>
  <ul class="navbar-nav">
    <MenuItem v-for="menu in menus" :menu="menu" />
  </ul>
</template>
```

- MenuItem.vue

```
<script setup>
const props = defineProps({
  menu: { Type: Object, required: true },
});
</script>
<template>
  <li class="nav-item">
    <router-link class="nav-link" :to="menu.url">
      <i :class="menu.icon"></i>
      {{ menu.title }}
    </router-link>
  </li>
</template>
<style></style>
```

- router



- **auth.js** - 회원정보, 로그인 routing 관리
- **board.js** - 게시판 관리 routing 관리
- **gallery.js** - 갤러리 관리 routing 관리
- **travel.js** - 트래블 routing 관리
- **index.js**
 - history
 - 프로젝트 정보
 - auth, board, gallery, travel routes 정보

- 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'),
  },
  {
    path: '/auth/profile',
    name: 'profile',
    component: () => import('../pages/auth/ProfilePage.vue'),
  },
  {
    path: '/auth/changepassword',
    name: 'changepassword',
    component: () => import('../pages/auth/ChangePasswordPage.vue'),
  },
];
```

- board.js

```
import { isAuthenticated } from '@/util/guards';

export default [
  {
    path: '/board/list',
    name: 'board/list',
    component: () => import('../pages/board/BoardListPage.vue'),
  },
  {
    path: '/board/detail/:no',
    name: 'board/detail',
    component: () => import('../pages/board/BoardDetailPage.vue'),
  },
  {
    path: '/board/create',
    name: 'board/create',
    component: () => import('../pages/board/BoardCreatePage.vue'),
    beforeEnter: isAuthenticated,
  },
  {
    path: '/board/update/:no',
    name: 'board/update',
    component: () => import('../pages/board/BoardUpdatePage.vue'),
    beforeEnter: isAuthenticated,
  },
];
```


- index.js

```
import { createRouter, createWebHistory } from 'vue-router';
import HomePage from '../pages/HomePage.vue';
import authRoutes from './auth';
import boardRoutes from './board';
import travelRoutes from './travel';
import galleryRoutes from './gallery';

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'home',
      component: HomePage,
    },
    ...authRoutes,
    ...boardRoutes,
    ...travelRoutes,
    ...galleryRoutes,
  ],
});

export default router;
```

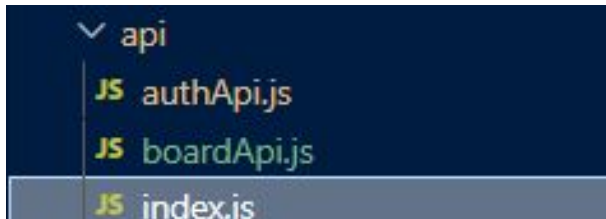
- gallery.js

```
export default [
  {
    path: '/gallery/list',
    name: 'gallery/list',
    component: () => import('../pages/gallery/GalleryListPage.vue'),
  },
];
```

- travel.js

```
export default [
  {
    path: '/travel/list',
    name: 'travel/list',
    component: () => import('../pages/travel/TravelListPage.vue'),
  },
];
```

- api : 함수를 이용한 내부처리 및 backend api호출



- authAPI.js - 회원가입과 CRUD 처리, 회원정보 확인 API 호출
- boardAPI.js - 게시판 CRUD 처리, 게시판 내용 확인 API 호출
- index.js - JWT 인증확인

- authAPI.js

```
import api from '@api';

const BASE_URL = '/api/member';
const headers = { 'Content-Type': 'multipart/form-data' };

export default {

  async checkUsername(username) {
    const { data } = await api.get(`${BASE_URL}/checkusername/${username}`);
    console.log('AUTH GET CHECKUSERNAME', data);
    return data;
  },

  /////////////// 회원 정보 조회 (username == id) ///////////////////
  async get(username) {
    const { data } = await api.get(`${BASE_URL}/${username}`);
    console.log('AUTH GET', data);
    return data;
  },

  /////////////// 회원 정보 가입 ///////////////////
  async create(member) {
    const formData = new FormData();
    formData.append('username', member.username);
    formData.append('email', member.email);
    formData.append('password', member.password);

    if (member.avatar) {
      formData.append('avatar', member.avatar);
    }

    // -----> 회원 정보 post방식 전송
    const { data } = await api.post(BASE_URL, formData, headers);

    console.log('AUTH POST: ', data);
    return data;
  },
}
```

axios
인터넷에 의해
token 정보
header에 포함됨.

```
///////////////// 회원 정보 수정 ///////////////////

async update(member) {
  const formData = new FormData();
  formData.append('username', member.username);
  formData.append('password', member.password);
  formData.append('email', member.email);

  if (member.avatar) {
    formData.append('avatar', member.avatar);
  }

  const { data } = await api.put(`${BASE_URL}/${member.username}`, formData, headers);
  console.log('AUTH PUT: ', data);
  return data;
},

///////////////// 회원 탈퇴 ///////////////////

async delete(username) {
  const { data } = await api.delete(`${BASE_URL}/${username}`);
  console.log('AUTH DELETE: ', data);
  return data;
},

///////////////// 회원 암호 수정 ///////////////////

async changePassword(formData) {
  const { data } = await api.put(`${BASE_URL}/${formData.username}/changepassword`, formData);
  console.log('AUTH PUT: ', data);
  return data;
},
}
```

- boardAPI.js

```
import api from '@api';

const BASE_URL = '/api/board';

const headers = { 'Content-Type': 'multipart/form-data' };

export default {
  async getList(params) {
    const { data } = await api.get(BASE_URL, { params });
    console.log('BOARD GET LIST: ', data);
    return data;
  },

  async get(no) {
    const { data } = await api.get(`${BASE_URL}/${no}`);
    console.log('BOARD GET: ', data);
    return data;
  },

  async create(article) {
    const formData = new FormData();

    formData.append('title', article.title);
    formData.append('writer', article.writer);
    formData.append('content', article.content);

    if (article.files) {
      for (let i = 0; i < article.files.length; i++) {
        formData.append('files', article.files[i]);
      }
    }

    const { data } = await api.post(BASE_URL, formData, { headers });
    console.log('BOARD POST: ', data);
    return data;
  },
};
```

```
async update(article) {
  const formData = new FormData();
  formData.append('no', article.no);
  formData.append('title', article.title);
  formData.append('writer', article.writer);
  formData.append('content', article.content);

  if (article.files) {
    for (let i = 0; i < article.files.length; i++) {
      formData.append('files', article.files[i]);
    }
  }

  const { data } = await api.put(`${BASE_URL}/${article.no}`, article, { headers });
  console.log('BOARD PUT: ', data);
  return data;
},

async delete(no) {
  const { data } = await api.delete(`${BASE_URL}/${no}`);
  console.log('BOARD DELETE: ', data);
  return data;
},

async deleteAttachment(no) {
  const { data } = await api.delete(`${BASE_URL}/deleteAttachment/${no}`);
  console.log('ATTACHMENT DELETE: ', data);
  return data;
},
};
```

- index.js

```
import axios from 'axios';

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

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

// 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) => {
    console.log(error);
    return Promise.reject(error);
  }
);
```

- **Axios** 요청 인터셉터를 설정, **Axios** 인스턴스를 설정하고, 요청 인터셉터를 통해 **JWT** 토큰을 요청 헤더에 추가하는 기능을 구현
1. **Axios** 인스턴스 생성:
 - 1000ms의 타임아웃 설정을 가진 새로운 **Axios** 인스턴스를 생성
 2. 요청 인터셉터:
 - **useAuthStore()**와 **getToken()**을 사용하여 **JWT** 토큰을 가지고 온다.
 - **토큰이 있을 경우, 이 토큰을 요청의 Authorization 헤더에 추가**
 - 요청 중 에러가 발생하면, 이를 로그에 기록하고, 해당 요청을 거부

**** Axios 인터셉터** 는 HTTP 요청이나 응답이 애플리케이션에서 처리되기 전에 가로채서 원하는 작업을 수행할 수 있도록 해주는 기능임. 예를 들어, 요청 인터셉터를 사용하면 모든 HTTP 요청에 인증 토큰을 자동으로 추가하거나, 응답 인터셉터를 사용해 특정한 응답 처리를 공통으로 할 수 있음.

- 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.' });
      // 로그인 필요
    } else if (error.response?.status === 403) {
      return Promise.reject({ error: '권한이 부족합니다.' });
    }
    return Promise.reject(error);
  }
);

export default instance;
```

- **Axios** 응답 인터셉터를 설정, 서버로부터 받은 응답을 검토하고, 특정 조건에 따라 에러를 처리하거나 리다이렉션하는 기능

1. 응답 처리:
 - 응답 상태 코드가 **200**일 경우 정상적으로 응답을 반환
 - 상태 코드가 **404**이면 "페이지 없음" 에러를 반환
2. 오류 처리:
 - 상태 코드가 **401** (인증 실패)인 경우, 로그아웃을 수행하고 로그인 페이지로 리다이렉션
 - 상태 코드가 **403** (권한 부족)인 경우, 권한 부족 에러를 반환

- **지드**

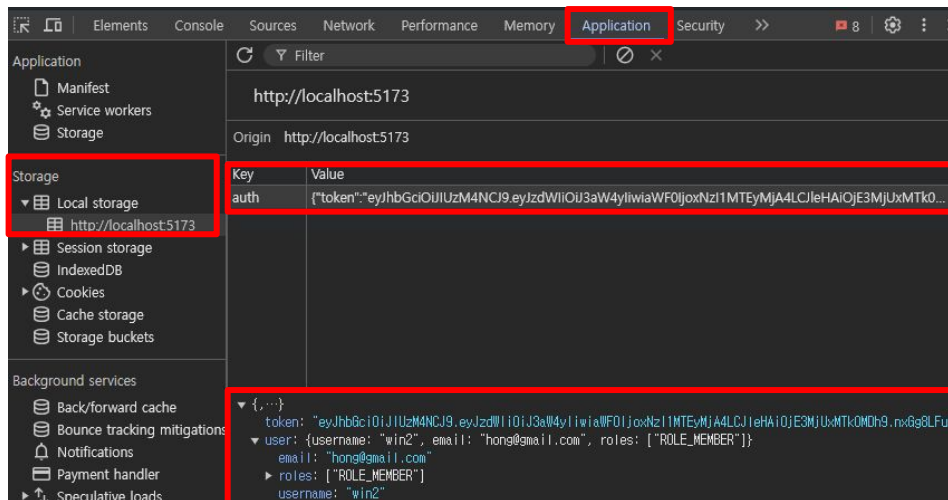
- API 호출
- local storage에 저장된 인증정보 불러오기, state에 저장
- auth 체크 호출 (state 값으로 확인)
- 저장된 정보 불러서 세션 체크, 페이지/게시판 화면으로 진입 사용 체크

- 로그인

- `axios.post()`
- `state.value`에 저장
- `local storage`에 저장

- 로그아웃

- `logout` storage clear
- `state.value`에 `null` 처리
- `local storage` 설정 초기화



- stores/auth.js : 토큰 저장(로컬스토리지 이용)

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

```
const initState = {
  token: '',
  user: {
    username: '',
    email: '',
    roles: [],
  },
};
```

```
export const useAuthStore = defineStore('auth', () => {
```

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

```
  const isLogin = computed(() => !!state.value.user.username);
```

```
  // state.value.user.username이 존재하면 그 값은 truthy이므로,
  // ---> !!state.value.user.username은 true를 반환
  // 만약 state.value.user.username이 null, undefined, ''(빈 문자열)과 같이 falsy한 값이라면,
  // ---> !!state.value.user.username은 false를 반환
  // 따라서 computed 함수는 사용자가 로그인 상태인지 (username이 있는지) 여부를 계산하여
  // isLogin이라는 계산된 속성에 할당
```


- 사용자의 로그인 상태와 관련된 정보를 관리하고, 사용자가 로그인했는지 여부를 쉽게 확인할 수 있도록 돕는 **Vue.js**의 상태 관리 로직

1. 초기 상태 설정 (`initState`)

- token: 사용자의 인증 토큰을 저장.
- user: 사용자 정보를 담은 객체로, `'username'`, `'email'`, `'roles'`라는 속성을 가짐

2. `'useAuthStore'` 정의

- `useAuthStore`는 인증 상태를 관리하는 스토어
- `state`는 `initState`로 초기화된 반응형(ref) 상태. 즉, 이 상태가 변경되면 **Vue** 컴포넌트도 자동으로 업데이트

3. `'isLogin'` 계산된 속성

- `isLogin`은 사용자가 로그인했는지 확인하는 계산된 속성
- `state.value.user.username`이 존재하면(즉, 사용자의 `'username'`이 있으면) `'true'`를 반환하고, 그렇지 않으면 `'false'`를 반환

- stores/auth.js : 토큰 저장(로컬스토리지 이용)

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

// localStorage에 저장된 사용자 인증 정보를 불러와서 state에 저장하는 역할
// localStorage에서 auth라는 항목을 가져와, 그 값이 존재하면 이를 파싱하여 state.value에 저장하는 함수
// localStorage.getItem('auth'): localStorage에서 'auth'라는 키에 저장된 값을 가져온다.
// if (auth !== null): auth 값이 존재하면 (null이 아니면),
// state.value = JSON.parse(auth); auth 문자열을 JSON 객체로 변환한 후 state.value에 할당

- stores/auth.js : 토큰 저장(로컬스토리지 이용)

```
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));  
};  
  
// 로그인 요청을 보내고, 서버로부터 받은 인증 정보를 상태와 localStorage에 저장하는 역할  
// axios.post('/api/auth/login', member): member 정보를 사용해 /api/auth/login API에 로그인 요청을 보냄.  
// { data } = await axios.post(...): API 요청이 완료되면 서버에서 받은 응답 데이터를 data에 저장  
// state.value = { ...data };; 받은 데이터를 state.value에 저장  
// localStorage.setItem('auth', JSON.stringify(state.value));: state.value를 문자열로 변환해 localStorage에 'auth'라는 키로 저장
```

- stores/auth.js : 토큰 저장(로컬스토리지 이용)

```
const getToken = () => state.value.token;

const changeProfile = (member) => {
  state.value.user.email = member.email;
  localStorage.setItem('auth', JSON.stringify(state.value));
};

load();

// 토큰을 가져오고, 사용자의 이메일을 업데이트하며, 초기 상태를 불러오는 기능을 수행
// getToken(): 현재 상태(state.value)에서 token 값을 반환합니다.
// changeProfile(member): 사용자의 이메일을 주어진 member.email로 변경하고, 변경된 상태를 localStorage에 저장합니다.
// load(): 페이지가 로드될 때 localStorage에서 저장된 인증 정보를 불러와 state에 설정

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

- util/guards.js : 가드 설정

- `isAuthenticated()`로 인증 확인 절차
- 인증되지 않은 경우, 로그인 페이지로 리다이렉션
- 로그인 -> 기존 페이지 이동
- 로그아웃 -> 로그인 화면 이동

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

export const isAuthenticated = (to, from) => {
  const auth = useAuthStore();

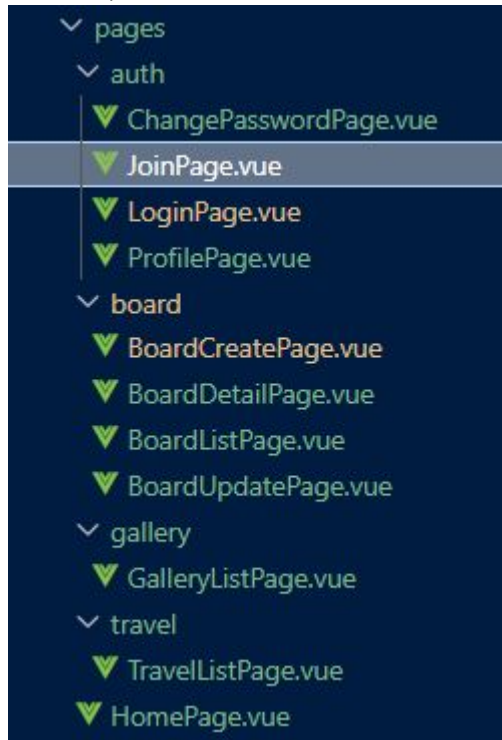
  if (!auth.isLogin) {
    console.log('로그인 필요.....');

    return { name: 'login', query: { next: to.name } };
  }

  console.log('로그인 인증');
};
```


- 사용자가 인증되지 않은 상태에서 특정 페이지로 접근하려고 할 때, 로그인 페이지로 리다이렉션하는 가드(guard)
- `'isAuthenticated'` 함수
 - 역할: 사용자가 인증(로그인)되어 있는지 확인하는 역할
 - 동작:
 1. `'useAuthStore()'`를 사용하여 `'auth'` 객체를 가지고 온다.
 2. `'auth.isLogin'`이 `'false'`일 경우(즉, 사용자가 로그인하지 않은 경우), 로그인 페이지로 리다이렉션함. 이때 현재 이동하려던 페이지(`'to.name'`) 정보를 쿼리로 함께 전달하여, 로그인 후에 다시 그 페이지로 돌아갈 수 있게 함
 3. 만약 `'auth.isLogin'`이 `'true'`라면, 현재 페이지로 계속 진행함

- **pages : template**을 포함하는 **ui** 및 기본 **computed()/유효성 검사**




1. **auth** - 회원가입/로그인
 - 회원가입 화면
 - 로그인 화면
 - 프로필 보기 화면
2. **board** - 게시판관리 화면
 - 게시판 목록 화면
 - 게시판 작성하기 화면
 - 게시판 목록 화면
 - 게시판 수정 화면
3. **gallery**
4. **travel**
5. **HomePage.vue** - 기본페이지


회원 가입

 사용자 ID : ID 중복 확인 사용가능한 ID입니다.

 아바타 이미지:

 email

 비밀번호:

 비밀번호 확인:

로그인

 사용자 ID:

 비밀번호:

- JoinPage.vue

```
<script setup>
import { reactive, ref } from 'vue';
import { useRouter } from 'vue-router';
import authApi from '@/api/authApi';

const router = useRouter();
const avatar = ref(null);
const checkError = ref('');

////////////////////////////////////
const member = reactive({
  username: 'hong',
  email: 'hong@gmail.com',
  password: '1234',
  password2: '1234',
  avatar: null,
});
////////////////////////////////////

const disableSubmit = ref(true);
const checkUsername = async () => {
  if (!member.username) {
    return alert('사용자 ID를 입력하세요.');
```

```
  }

  if (member.password !== member.password2) {
    return alert('비밀번호가 일치하지 않습니다.');
```

```
  if (avatar.value.files.length > 0) {
    member.avatar = avatar.value.files[0];
  }
```

```
  try {
    await authApi.create(member);
    router.push({ name: 'home' });
  } catch (e) {
    console.error(e);
  }
```

```
};
</script>
```


- JoinPage.vue

```
<template>
  <div class="mt-5 mx-auto" style="width: 500px">
    <h1 class="my-5">
      <i class="fa-solid fa-user-plus" ></i>
      회원 가입
    </h1>

    <form @submit.prevent="join">
      <div class="mb-3 mt-3">
        <label for="username" class="form-label">
          <i class="fa-solid fa-user"></i>
          사용자 ID :
          <button type="button" class="btn btn-success btn-sm py-0 me-2" @click="checkUsername">ID 중복 확인</button>
          <span :class="disableSubmit.value ? 'text-primary' : 'text-danger'">{{ checkError }}</span>
        </label>
        <input type="text" class="form-control" placeholder="사용자 ID" id="username" @input="changeUsername" v-model="member.username" />
      </div>

      <div>
        <label for="avatar" class="form-label">
          <i class="fa-solid fa-user-astronaut"></i>
          아바타 이미지 :
        </label>
        <input type="file" class="form-control" ref="avatar" id="avatar" accept="image/png, image/jpeg" />
      </div>

      <div class="mb-3 mt-3">
        <label for="email" class="form-label">
          <i class="fa-solid fa-envelope"></i>
          email
        </label>
        <input type="email" class="form-control" placeholder="Email" id="email" v-model="member.email" />
      </div>
    </form>
  </div>
```

- JoinPage.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" id="password" v-model="member.password" />
</div>

<div class="mb-3">
  <label for="password" class="form-label">
    <i class="fa-solid fa-lock"></i>
    비밀번호 확인:
  </label>
  <input type="password" class="form-control" placeholder="비밀번호 확인" id="password2" v-model="member.password2" />
</div>

  I
  <button type="submit" class="btn btn-primary mt-4" :disabled="disableSubmit">
    <i class="fa-solid fa-user-plus"></i>
    확인
  </button>
</form>

</div>
</template>
```

- LoginPage.vue

```
<script setup>
import { computed, reactive, ref } from 'vue';
import { useAuthStore } from '@stores/auth';
import { useRoute, useRouter } from 'vue-router';

const cr = useRoute();
const router = useRouter();
const auth = useAuthStore();

///////////////////////////////////////////////////
const member = reactive({
  username: '',
  password: '',
});

const error = ref('');

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

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

```
<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" required />
    </div>

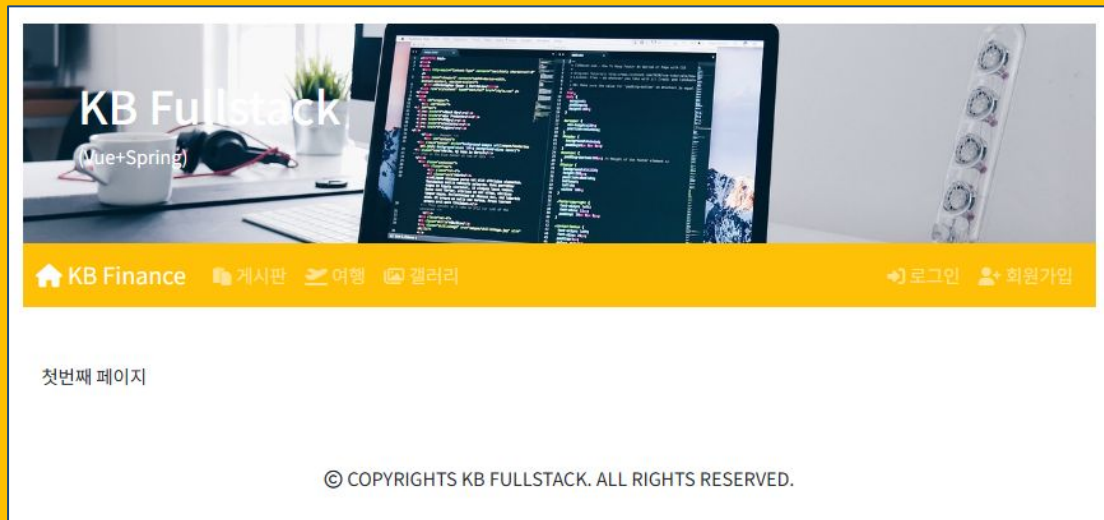
    <div class="mb-3">
      <label for="password" class="form-label">
        <i class="fa-solid fa-lock"></i>
        비밀번호:
      </label>
      <input type="password" class="form-control" placeholder="비밀번호" v-model="member.password" required />
    </div>

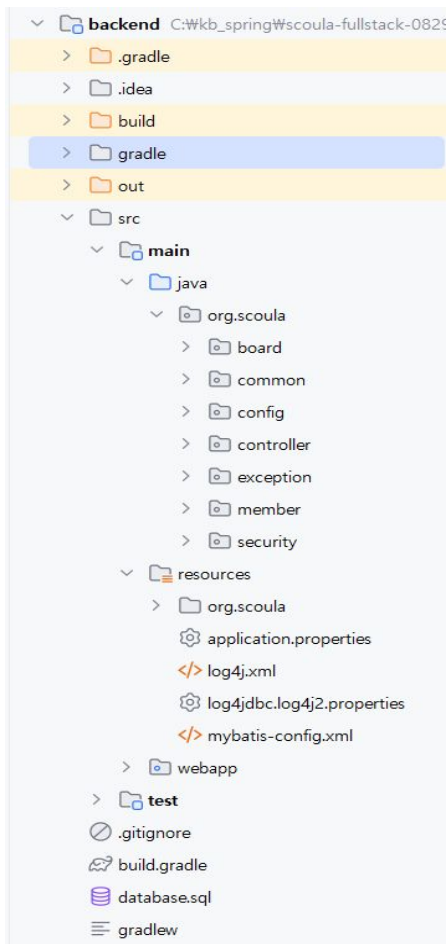
    <div v-if="error" class="text-danger">{{ error }}</div>

    <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>
```

backend





- **java/org/scoula/board** : board
- **java/org/scoula/common**: pagination, util(file upload)
- **java/org/scoula/config** : spring project java 설정
- **java/org/scoula/exception**: project 전체 exception
- **java/org/scoula/member**: member
- **java/org/scoula/security**: security, JWT 설정/처리
- **resources/org.scoula/mapper** : mapper xml 파일
- **webapp/resources/assets** : static 파일

- build.gradle sync
- tomcat setting
- WEB-INF/views 삭제

- 의존성 추가

- 주요 버전

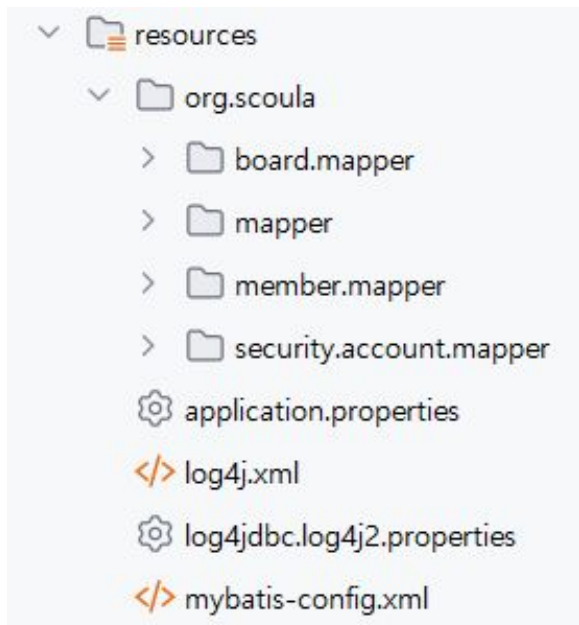
- JDK-17
- JUnit - 5.9.2
- Spring - 5.3.29
- Lombok - 1.18.30
- Spring Security - 5.8.13
- JWT - 0.11.5

- Spring

- tx
- jdbc
- context
- webmvc
- inject

- AOP - 1.9.20
- JSP/Servlet4/JSTL1.2
- slf4j - 2.0.9 / log4j - 1.2.17
- logback - 1.4.6
- Lombok - 1.18.30
- mysql 8.1 / HikariCP (2.7)
- Security - 5.6.13

- application.properties



```
jdbc.driver=net.sf.log4jdbc.sql.jdbcapi.DriverSpy
jdbc.url=jdbc:log4jdbc:mysql://localhost:3306/shop2
jdbc.username=root
jdbc.password=1234

upload.maxFileSize=1024
```

- log4jdbc.log4j2.properties

```
log4jdbc.spylogdelegator.name=net.sf.log4jdbc.log.slf4j.Slf4jSpyLogDelegator
log4jdbc.auto.load.popular.drivers=false
log4jdbc.drivers=com.mysql.cj.jdbc.Driver
```

- mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
    PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>
  <settings>
    <setting name="mapUnderscoreToCamelCase" value="true" />
  </settings>

  <typeAliases>
    <package name="org.scoula.security.account.domain" />
    <package name="org.scoula.board.domain" />
  </typeAliases>

</configuration>
```


- RootConfig.java

```
@Configuration
@PropertySource({"classpath:/application.properties"})
@ComponentScan(basePackages = {
    "org.scoula.board.mapper",
    "org.scoula.member.mapper"
})
@ComponentScan(basePackages = {
    "org.scoula.board.service",
    "org.scoula.member.service"
})
@Slf4j
@EnableTransactionManagement
public class RootConfig {
    @Value("${net.sf.log4jdbc.sql.jdbcapi.DriverSpy}") String driver;
    @Value("${jdbc:log4jdbc:mysql://localhost:3306/shop2}") String url;
    @Value("${root}") String username;
    @Value("${1234}") String password;

    @Bean
    public DataSource dataSource() {
        HikariConfig config = new HikariConfig();

        config.setDriverClassName(driver);
        config.setJdbcUrl(url);
        config.setUsername(username);
        config.setPassword(password);

        HikariDataSource dataSource = new HikariDataSource(config);
        return dataSource;
    }

    @Autowired
    ApplicationContext applicationContext;

    @Bean
    public sqlSessionFactory sqlSessionFactory() throws Exception {
        sqlSessionFactoryBean sqlSessionFactory = new sqlSessionFactoryBean();
        sqlSessionFactory.setConfigLocation(
            applicationContext.getResource("classpath:/mybatis-config.xml"));
        sqlSessionFactory.setDataSource(dataSource());
        return sqlSessionFactory.getObject();
    }

    @Bean
    public DataSourceTransactionManager transactionManager(){
        DataSourceTransactionManager manager = new DataSourceTransactionManager(dataSource());
        return manager;
    }
}
```

1. 데이터 소스 설정 (dataSource):

- HikariCP를 사용하여 데이터베이스 연결 풀(DataSource)을 설정
- 데이터베이스 연결 정보는 application.properties에서 로드

2. SqlSessionFactory 설정 (sqlSessionFactory):

- MyBatis의 SQL 세션 팩토리를 생성
- mybatis-config.xml 파일을 사용해 MyBatis 설정을 로드

3. 트랜잭션 매니저 설정 (transactionManager):

- 트랜잭션 관리를 위해 DataSourceTransactionManager를 설정

4. 어노테이션 사용:

- @MapperScan으로 MyBatis Mapper 인터페이스의 위치를 지정
- @ComponentScan으로 서비스 클래스들을 스캔하여 빈으로 등록
- @EnableTransactionManagement로 트랜잭션 관리 기능을 활성화

- ServletConfig.java

```
@EnableWebMvc 1 usage
@ComponentScan(basePackages = {
    "org.scoula.controller",
    "org.scoula.exception",
    "org.scoula.member.controller",
    "org.scoula.board.controller",
})
public class ServletConfig implements WebMvcConfigurer {

    @Override 2 usages
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController(urlPathOrPattern: "/").setViewName("forward:/resources/index.html");
    }

    @Override 2 usages
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler(pathPatterns: "/resources/**")
            .addResourceLocations("/resources/");
        registry.addResourceHandler(pathPatterns: "/assets/**")
            .addResourceLocations("/resources/assets/");
    }

    // Servlet 3.0 파일 업로드 사용시
    @Bean
    public MultipartResolver multipartResolver() {
        StandardServletMultipartResolver resolver = new StandardServletMultipartResolver();
        return resolver;
    }
}
```

1.Spring MVC 활성화 (@EnableWebMvc):

- Spring MVC를 활성화하여 웹 애플리케이션을 구성

2.뷰 컨트롤러 설정 (addViewControllers):

- 기본 경로(/)로 접근 시 index.html로 포워딩

3.정적 리소스 핸들링 (addResourceHandlers):

- /resources/와 /assets/ 경로의 정적 리소스를 처리

4.파일 업로드 설정 (multipartResolver):

- Servlet 3.0의 파일 업로드 기능을 위한 멀티파트 리졸버를 설정

- WebConfig.java

```
@Slf4j
@Configuration
//@PropertySource({"classpath:/application.properties"})
public class WebConfig extends AbstractAnnotationConfigDispatcherServletInitializer {
    final String LOCATION = "c:/upload"; 1 usage
    final long MAX_FILE_SIZE = 1024 * 1024 * 10L; 1 usage
    final long MAX_REQUEST_SIZE = 1024 * 1024 * 20L; 1 usage
    final int FILE_SIZE_THRESHOLD = 1024 * 1024 * 5; 1 usage

    // @Value("${upload.maxFileSize}") Long maxFileSize;
    @Override 5 usages
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] { RootConfig.class, SecurityConfig.class };
    }

    @Override 4 usages
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { ServletConfig.class };
    }

    @Override 2 usages
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }

    @Override 2 usages
    protected void customizeRegistration(ServletRegistration.Dynamic registration) {
        log.info("Max File Size: " + maxFileSize);

        registration.setInitParameter( name: "throwExceptionIfNoHandlerFound", value: "true");
        MultipartConfigElement multipartConfig =
            new MultipartConfigElement(
                LOCATION, // 업로드 처리 디렉토리 경로
                MAX_FILE_SIZE, // 업로드 가능한 파일 하나의 최대 크기
                MAX_REQUEST_SIZE, // 업로드 가능한 전체 최대 크기(여러 파일 업로드 하는 경우)
                FILE_SIZE_THRESHOLD // 메모리 파일의 최대 크기(이보다 작으면 실제 메모리에서만 작업)
            );
        registration.setMultipartConfig(multipartConfig);
    }
}
```

1.Spring MVC 활성화 (@EnableWebMvc):

- Spring MVC를 활성화하여 웹 애플리케이션을 구성

2.뷰 컨트롤러 설정 (addViewControllers):

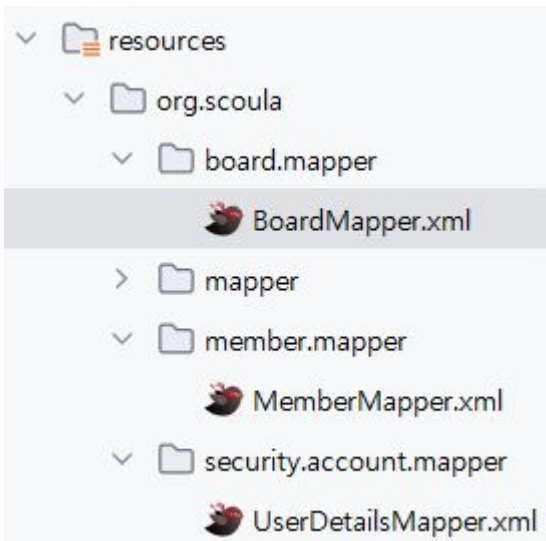
- 기본 경로(/)로 접근 시 index.html로 포워딩

3.정적 리소스 핸들링 (addResourceHandlers):

- /resources/와 /assets/ 경로의 정적 리소스를 처리

4.파일 업로드 설정 (multipartResolver):

- Servlet 3.0의 파일 업로드 기능을 위한 멀티파트 리졸버를 설정



```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
PUBLIC "-//mybatis.org/DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="org.scoula.board.mapper.BoardMapper">
  <insert id="create">
    insert into tbl_board (title, content, writer)
    values (#{title}, #{content}, #{writer})
    <selectKey resultType="Long" keyProperty="no" keyColumn="no" order="AFTER">
      SELECT LAST_INSERT_ID()
    </selectKey>
  </insert>

  <update id="update">
    update tbl_board set
      title = #{title},
      content = #{content},
      writer = #{writer},
      update_date = now()
    where no = #{no}
  </update>

  <delete id="delete">
    delete from tbl_board where no = #{no}
  </delete>

  <select id="getTotalCount" resultType="java.lang.Integer">
    select count(*) from tbl_board
  </select>

  <select id="getPage" resultType="BoardVO">
    <![CDATA[
      select * from tbl_board
      order by no desc
      limit #{offset}, #{amount}
    ]]>
  </select>

  <select id="getList" resultType="BoardVO">
    <![CDATA[
      select * from tbl_board
      order by no desc
    ]]>
  </select>

  <resultMap id="attachmentMap" type="org.scoula.board.domain.BoardAttachmentVO">
    <id column="ano" property="no"/>
    <result column="bno" property="bno"/>
    <result column="filename" property="filename"/>
    <result column="path" property="path"/>
    <result column="contentType" property="contentType"/>
    <result column="size" property="size"/>
    <result column="a_reg_date" property="regDate"/>
  </resultMap>

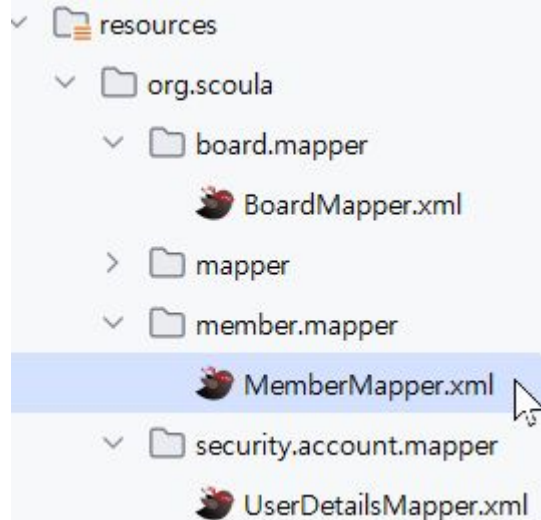
  <resultMap id="boardMap" type="org.scoula.board.domain.BoardVO">
    <id column="no" property="no"/>
    <result column="title" property="title"/>
    <result column="content" property="content"/>
    <result column="writer" property="writer"/>
    <result column="reg_date" property="regDate"/>
    <result column="update_date" property="updateDate"/>
    <collection property="attaches" resultMap="attachmentMap"/>
  </resultMap>

  <!-- select * from tbl_board where no = #{no} -->
  <select id="get" resultMap="boardMap">
    select b.*, a.no as ano, a.bno, a.filename, a.path, a.content_type, a.size, a.reg_date a_reg_date
    from tbl_board b
    left outer join tbl_board_attachment a
    on b.no = a.bno
    where b.no = #{no}
    order by filename
  </select>

  <insert id="createAttachment">
    insert into tbl_board_attachment(filename, path, content_type, size, bno)
    values(#{filename}, #{path}, #{contentType}, #{size}, #{bno})
  </insert>

  <select id="getAttachmentList" resultType="BoardAttachmentVO">
    select * from tbl_board_attachment
    where bno = #{bno}
    order by filename
  </select>

  <select id="getAttachment" resultType="BoardAttachmentVO">
    select * from tbl_board_attachment
    where no = #{no}
  </select>
</mapper>
```



```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="org.scoula.member.mapper.MemberMapper">

  <insert id="insert">
    INSERT INTO tbl_member
    VALUES(#{username}, #{password}, #{email}, now(), now() )
  </insert>

  <update id="update">
    UPDATE tbl_member
    SET
      email = #{email},
      update_date = now()
    WHERE username =#{username}
  </update>

  <update id="updatePassword">
    UPDATE tbl_member
    SET
      password = #{newPassword},
      update_date = now()
    WHERE username =#{username}
  </update>

  <delete id="delete">
    DELETE FROM tbl_member
    WHERE username =#{username}
  </delete>

  <resultMap id="authMap" type="Authvo">
    <result property="username" column="username" />
    <result property="auth" column="auth" />
  </resultMap>

  <resultMap id="memberMap" type="MemberVO">
    <id property="username" column="username" />
    <result property="username" column="username" />
    <result property="password" column="password" />
    <result property="email" column="email" />
    <result property="regDate" column="reg_date" />
    <result property="updateDate" column="update_date" />

    <collection property="authList" resultMap="authMap" />
  </resultMap>
```

```
<select id="get" resultMap="memberMap">
  SELECT m.username, password, email, reg_date, update_date, auth
  FROM
    tbl_member m
  LEFT OUTER JOIN tbl_member_auth a
    ON m.username = a.username
  where m.username = #{username}
</select>

<select id="getList" resultMap="memberMap">
  SELECT m.username, password, email, reg_date, update_date, auth
  FROM
    tbl_member m
  LEFT OUTER JOIN tbl_member_auth a
    ON m.username = a.username
</select>

<select id="findByusername" resultType="org.scoula.security.account.domain.MemberVO">
  SELECT * FROM tbl_member WHERE username = #{username}
</select>

<insert id="insertAuth">
  INSERT INTO tbl_member_auth(username, auth)
  VALUES(#{username}, #{auth})
</insert>

<delete id="deleteAuth">
  DELETE FROM tbl_member_auth
  WHERE username = #{username} AND auth = #{auth}
</delete>
</mapper>
```


- ServletConfig.java

```
@ControllerAdvice
@Slf4j
public class CommonExceptionHandler {
    @ExceptionHandler({NoHandlerFoundException.class})
    public String handle404(NoHandlerFoundException ex) {
        log.error(ex.getMessage());
        return "/resources/index.html";
    }
}
```

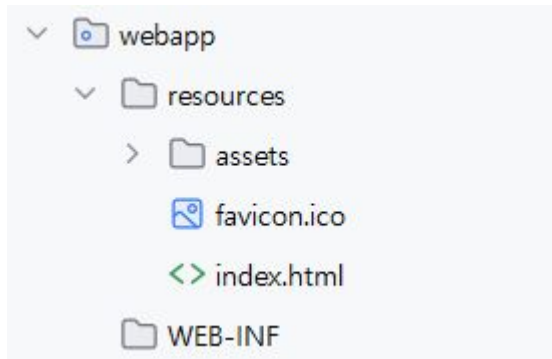
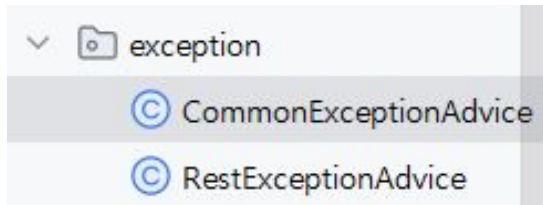
404
에러처리

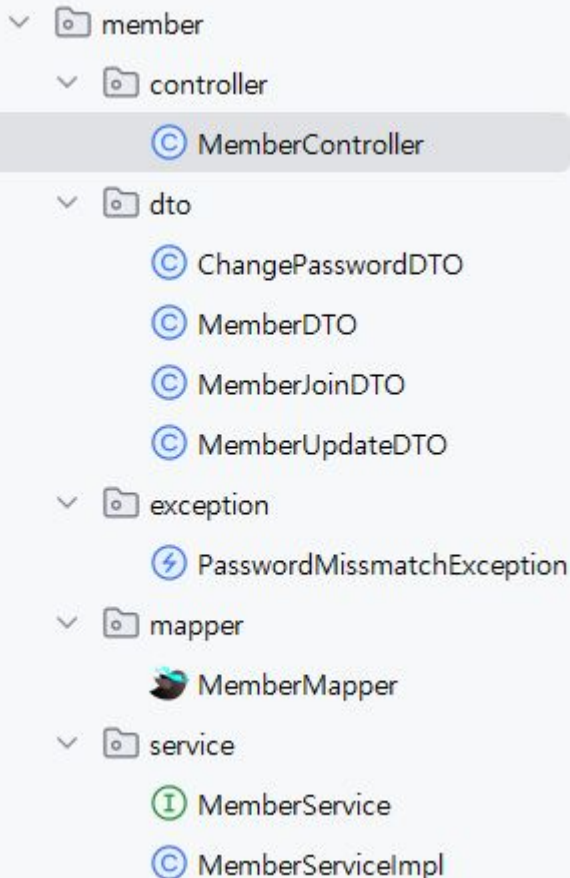
- RestExceptionHandler.java

```
@RestControllerAdvice
@Slf4j
public class RestExceptionHandler {
    @ExceptionHandler({PasswordMismatchException.class})
    public ResponseEntity<?> handlePasswordError(Exception ex) {
        return ResponseEntity.status(400)
            .header(HttpHeaders.CONTENT_TYPE, "application/json; charset=utf-8")
            .body(ex.getMessage());
    }

    @ExceptionHandler({Exception.class})
    public ResponseEntity<?> handleError(Exception ex) {
        log.error(ex.getMessage());
        return ResponseEntity.status(500)
            .header(HttpHeaders.CONTENT_TYPE, "application/json; charset=utf-8")
            .body(ex.getMessage());
    }
}
```

400/500
에러처리





- MemberController.java

```
@Slf4j
@RestController
@RequiredArgsConstructor
@RequestMapping("/api/member")
public class MemberController {
    final MemberService service;

    @GetMapping("/{checkusername/{username}}")
    public ResponseEntity<Boolean> checkUsername(@PathVariable String username) {
        return ResponseEntity.ok().body(service.checkDuplicate(username));
    }

    @GetMapping("/{username}")
    public ResponseEntity<MemberDTO> get(@PathVariable String username) {
        return ResponseEntity.ok(service.get(username));
    }

    @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()) {
            file = new File(avatarPath + "unknown.png");
        }
        uploadFiles.downloadImage(response, file);
    }

    @PostMapping("")
    public ResponseEntity<MemberDTO> join(MemberJoinDTO member) { return ResponseEntity.ok(service.join(member)); }

    @PutMapping("/{username}/changepassword")
    public ResponseEntity<?> changePassword(@RequestBody changePasswordDTO changePasswordDTO) {
        service.changePassword(changePasswordDTO);
        return ResponseEntity.ok().build();
    }

    @PutMapping("/{username}")
    public ResponseEntity<MemberDTO> changeProfile(MemberUpdateDTO member) {
        return ResponseEntity.ok(service.update(member));
    }

    @DeleteMapping("/{username}")
    public ResponseEntity<MemberDTO> delete(@PathVariable String username) {
        return ResponseEntity.ok(service.delete(username));
    }
}
```

- MemberController.java

1. 사용자명 중복 확인 (`//checkusername/{username}`):

- 주어진 사용자명이 중복되었는지 확인하고, 결과를 반환

2. 회원 정보 조회 (`//{username}`):

- 주어진 사용자명에 해당하는 회원 정보를 반환

3. 아바타 이미지 조회 (`//{username}/avatar`):

- 주어진 사용자명의 아바타 이미지를 반환하며, 이미지가 없을 경우 기본 이미지를 반환

4. 회원 가입 (`POST /api/member`):

- 새로운 회원을 등록하고, 등록된 회원 정보를 반환

5. 비밀번호 변경 (`PUT /api/member/{username}/changepassword`):

- 주어진 사용자명의 비밀번호를 변경

6. 프로필 수정 (`PUT /api/member/{username}`):

- 회원의 프로필 정보를 업데이트하고, 업데이트된 정보를 반환

7. 회원 삭제 (`DELETE /api/member/{username}`):

- 주어진 사용자명을 가진 회원을 삭제하고, 삭제된 회원 정보를 반환

- MemberDTO.java

```
@Data 19 usages
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class MemberDTO {
    private String username;
    // private String password;
    private String email;
    private Date regDate;
    private Date updateDate;

    MultipartFile avatar;

    private List<String> authList; // 권한 목록, join 처리 필요

    public static MemberDTO of(Membervo m) {
        return MemberDTO.builder()
            .username(m.getUsername())
            .password(m.getPassword())
            .email(m.getEmail())
            .regDate(m.getRegDate())
            .updateDate(m.getUpdateDate())
            .authList(m.getAuthList().stream().map(a->a.getAuth()).toList())
            .build();
    }

    public Membervo toVO() { no usages
        return Membervo.builder()
            .username(username)
            .password(password)
            .email(email)
            .regDate(regDate)
            .updateDate(updateDate)
            .build();
    }
}
```

- MemberJoinDTO.java

```
@Data 6 usages
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class MemberJoinDTO {
    String username;
    String password;
    String email;

    MultipartFile avatar;

    public Membervo toVO() { 1 usage
        return Membervo.builder()
            .username(username)
            .password(password)
            .email(email)
            .build();
    }
}
```

builder
DTO <-> VO

- ChangePasswordDTO.java

```
@Data 8 usages
@NoArgsConstructor
@AllArgsConstructor
public class ChangePasswordDTO {
    String username;
    String oldPassword;
    String newPassword;
}
```

- MemberUpdatedDTO.java

```
@Data 6 usages
@NoArgsConstructor
@AllArgsConstructor
public class MemberUpdatedDTO {
    private String username;
    private String password;
    private String email;

    MultipartFile avatar;

    public MemberVO toVO() { 1 usage
        return MemberVO.builder()
            .username(username)
            .email(email)
            .build();
    }
}
```

- MemberServiceImpl.java

```
@Slf4j
@Service
@RequiredArgsConstructor
public class MemberServiceImpl implements MemberService {
    final PasswordEncoder passwordEncoder;
    final MemberMapper mapper;

    @Override @1 usage
    public boolean checkDuplicate(String username) {
        MemberVO member = mapper.findByUsername(username);
        return member != null ? true : false;
    }

    @Transactional @1 usage
    @Override
    public MemberDTO join(MemberJoinDTO dto) {
        MemberVO member = dto.toVO();

        member.setPassword(passwordEncoder.encode(member.getPassword()));
        mapper.insert(member);

        AuthVO auth = new AuthVO();
        auth.setUsername(member.getUsername());
        auth.setAuth("ROLE_MEMBER");

        mapper.insertAuth(auth);
        saveAvatar(dto.getAvatar(), member.getUsername());

        return get(member.getUsername());
    }

    private void saveAvatar(MultipartFile avatar, String username) { @2 usages
        //아바타 업로드
        if(avatar != null && !avatar.isEmpty()) {
            File dest = new File( parent: "c:/upload/avatar", child: username + ".png");
            try {
                avatar.transferTo(dest);
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

1. 중복 확인 (checkDuplicate):

- 주어진 사용자 이름으로 회원이 이미 존재하는지 확인하는 메서드.

2. 회원 가입 (join):

- DTO를 MemberVO로 변환
- 비밀번호를 암호화한 후 회원 정보를 데이터베이스에 저장
- 사용자에게 "ROLE_MEMBER" 권한을 부여하고, 해당 정보를 데이터베이스에 추가
- 아바타 이미지를 저장
- 회원 정보를 반환

3. 아바타 저장 (saveAvatar):

- 업로드된 아바타 파일을 지정된 경로에 저장하는 메서드.

- MemberServiceImpl.java

```
@Override 1 usage
public MemberDTO update(MemberUpdatedDTO member) {
    MemberVO vo = mapper.get(member.getUsername());
    if(!passwordEncoder.matches(member.getPassword(),vo.getPassword())) {
        throw new PasswordMismatchException();
    }

    mapper.update(member.toVO());
    saveAvatar(member.getAvatar(), member.getUsername());
    return get(member.getUsername());
}

@Override
public MemberDTO delete(String username) {
    MemberVO member = mapper.get(username);
    mapper.delete(username);
    return MemberDTO.of(member);
}

@Override 1 usage
public void changePassword(ChangePasswordDTO changePassword) {
    MemberVO member = mapper.get(changePassword.getUsername());
    if(!passwordEncoder.matches(
        changePassword.getOldPassword(),
        member.getPassword()
    )) {
        throw new PasswordMismatchException();
    }
    changePassword.setNewPassword(passwordEncoder.encode(changePassword.getNewPassword()));
    mapper.updatePassword(changePassword);
}

@Override
public MemberDTO get(String username) {
    MemberVO member = optional.ofNullable(mapper.get(username))
        .orElseThrow(NoSuchElementException::new);
    return MemberDTO.of(member);
}
```

1.회원 정보 업데이트 (update):

- 주어진 회원 정보를 업데이트.
- 비밀번호가 일치하지 않으면 예외를 발생

2.회원 삭제 (delete):

- 사용자 이름으로 회원 정보를 찾아 삭제

3.비밀번호 변경 (changePassword):

- 기존 비밀번호가 일치하는지 확인하고, 일치하면 새 비밀번호로 변경

4.회원 정보 조회 (get):

- 사용자 이름으로 회원 정보를 조회하고, 없을 경우 예외를 발생

1. full stack project

- a. database 준비
- b. frontend project
- c. backend project

2. frontend jwt localStorage 저장

3. backend jwt 발행

4. board paging

5. board fileupload