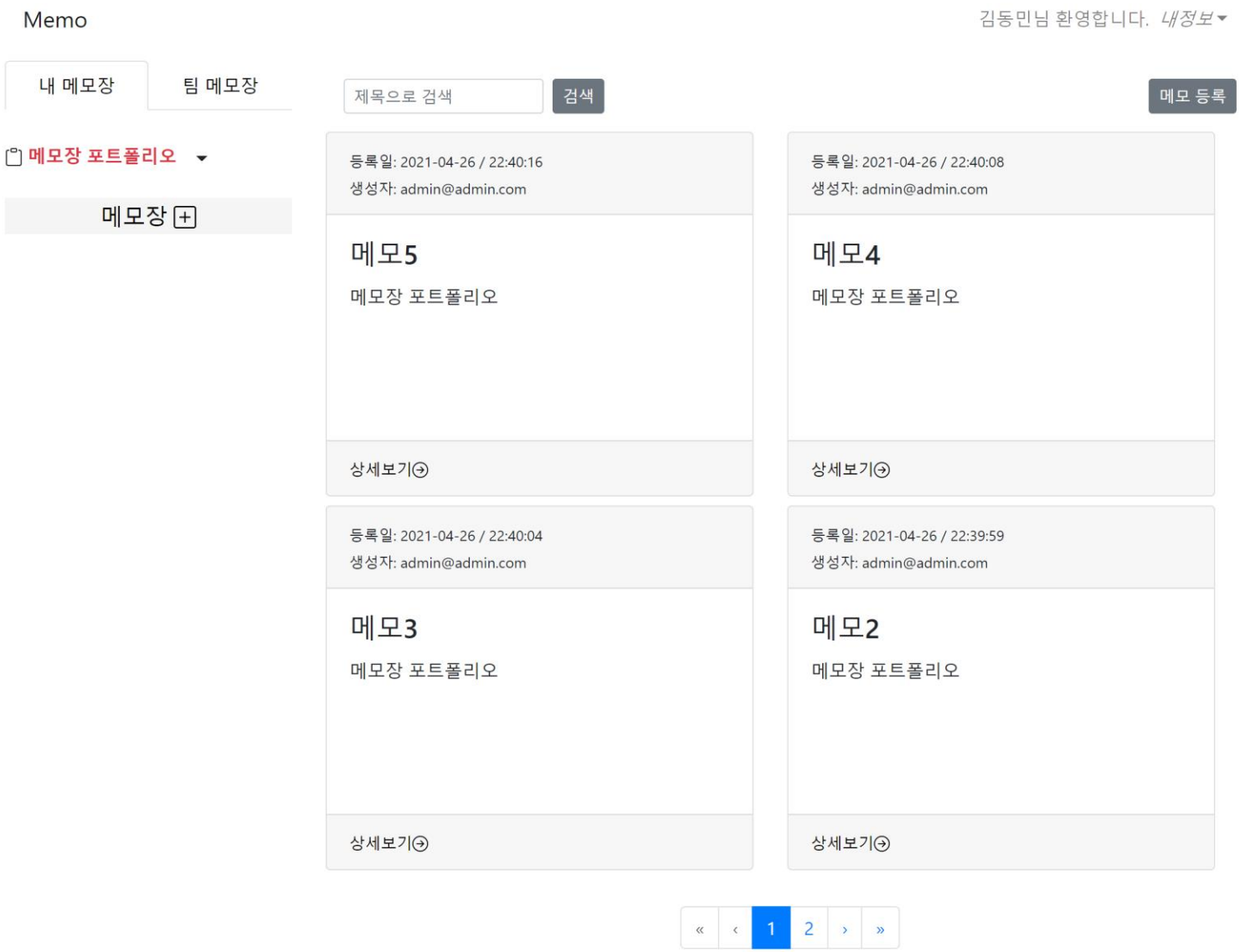
메모장 프로젝트


Memo
김동민님 환영합니다. 내정보
내 메모장
팀 메모장
제목으로 검색
검색
메모 등록
메모장 포트폴리오
메모장
등록일: 2021-04-26 / 22:40:16
생성자: admin@admin.com
메모5
메모장 포트폴리오
상세보기
등록일: 2021-04-26 / 22:40:08
생성자: admin@admin.com
메모4
메모장 포트폴리오
상세보기
등록일: 2021-04-26 / 22:40:04
생성자: admin@admin.com
메모3
메모장 포트폴리오
상세보기
등록일: 2021-04-26 / 22:39:59
생성자: admin@admin.com
메모2
메모장 포트폴리오
상세보기
« ‹ 1 2 › »

목차

프로젝트의 소스코드

- FrontEnd

https://github.com/blackcat0828/Memo-Frontend-Vue.js-

- BackEnd

https://github.com/blackcat0828/Memo-backend-Springboot-Mybatis-

개요 – 프로젝트 주제

웹 서비스 개발에 있어서 가장 기본적인 기능은 CRUD 기능이라고 생각합니다. 이 기능들을 구현하며 FrontEnd와 BackEnd 간의 통신 및 데이터 전달을 구현해 볼 수 있기 때문입니다.

저의 포트폴리오 프로젝트는 기본적인 CRUD 기능을 가진 메모장을 만들며 각 메모장마다 고유의 메모를 작성할수 있게 하였습니다. 그리고 **원하는 메모장을 다른 사용자들과 공유하여 같이 사용할수 있도록** 구현하였습니다.

이 프로젝트는 Server Side Rendering으로도 충분히 구현할 수 있지만 추후 Application의 확장성(모바일)을 위해서 BackEnd를 RESTful 방식을 사용하여 구축하였습니다. 스프링 부트를 사용하여 RESTful 방식의 서버를 구축하기로 결정하였기 때문에 FrontEnd는 SPA를 구현하기 위하여 Vue.js를 사용하였습니다.

개요 – 개발 스팩 및 상세 기능

- **개발 스팩**

BackEnd

- Java8
- Gradle
- Spring boot
- Spring Security
- Mybatis
- MariaDB

FrontEnd

- Vue.Js
- Bootstrap-Vue
- Axios

- **상세 기능**

로그인 & 회원가입 페이지

- JWT (Json Web Token) 사용

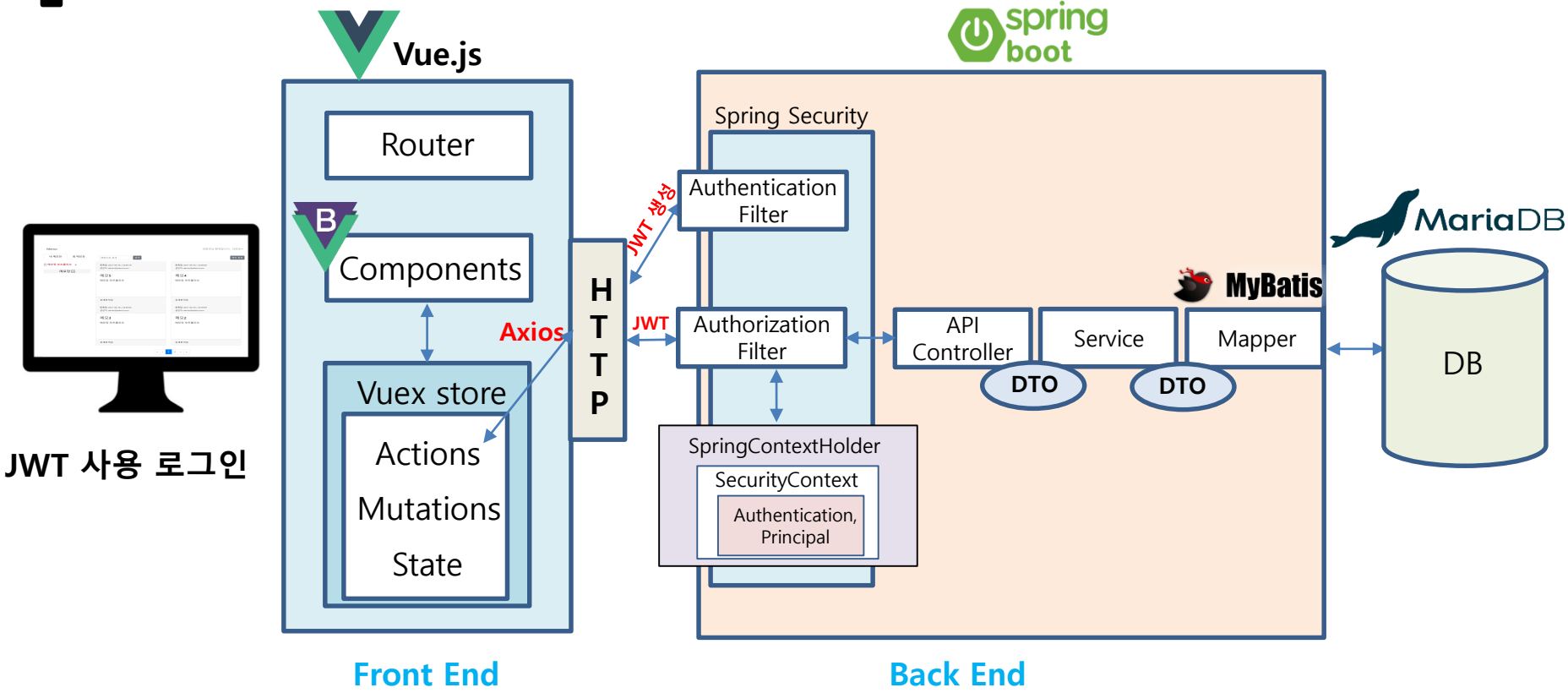
메인 페이지

- 메모장, 메모 CRUD
  - 메모리스트 페이징 처리
  - 메모 제목으로 검색 기능

메모장 멤버 공유 기능

- 해당 메모장을 공유할 멤버 추가, 삭제 기능
- 공유된 멤버가 메모장 멤버에서 탈퇴할수 있는 기능

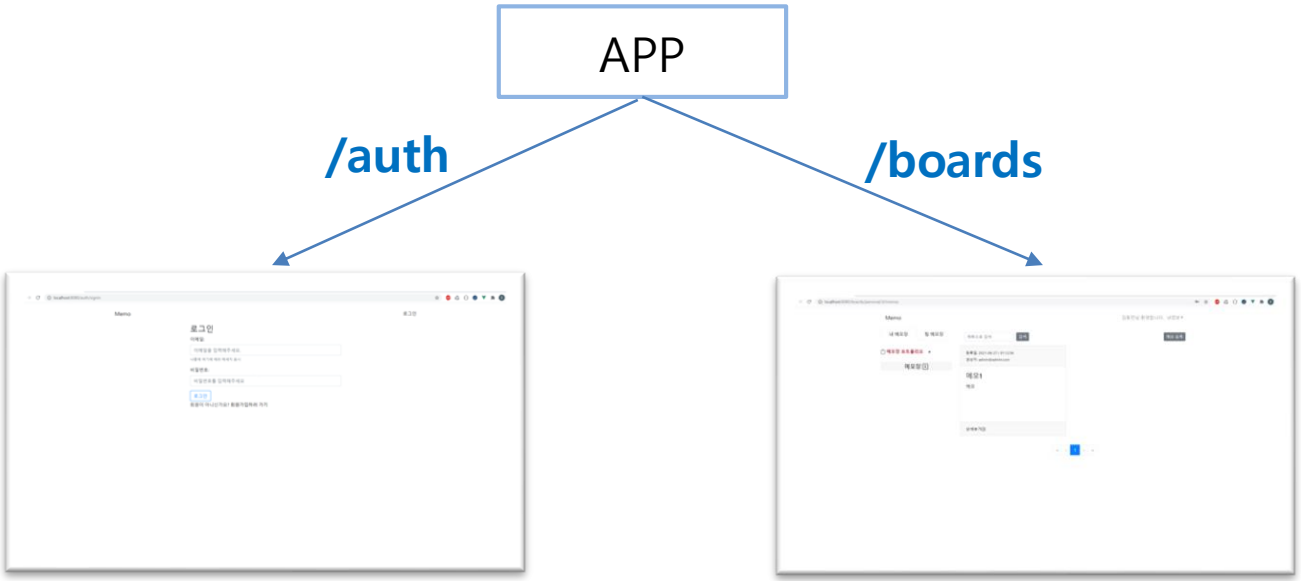
Project Achitecture



- JWT 통한 로그인 관리
- Vue.js Framework를 이용한 FrontEnd
  - Bootstrap-vue를 이용한 component구성
  - Vuex를 이용한 상태 관리
- Axios를 이용한 API 호출

- Spring Security를 이용한 인증 및 권한 제어
- Spring Boot Framework를 이용한 BackEnd

FrontEnd – Vue Router 구성(1)



- **Vue Router:** SPA는 화면이동시 기본적으로 URL을 사용하는 것이 아닌 한페이지 안에서 모듈을 바꿔서 필요한 부분만 다시 랜더링 하는 방식임으로 URL을 이용해서 특정페이지를 공유를 하는 것 같은 것들이 어려움으로 라우팅을 쉽게 해주는 Vue Router 라이브러리를 사용하여 각 페이지 마다 고유의 URL를 부여하였습니다.
- 가장 기본적인 UI 구성으로 위와 같은 2개의 기본 페이지를 생각하였고, 이 기본 페이지를 최상위 컴포넌트로 구성하였습니다.
- Vue Router를 이용하여 2개의 최상위 컴포넌트(여러 하위 컴포넌트들로 구성)로 이동할 수 있습니다.

FrontEnd – Vue Router 구성(2)

```
Vue.use(Router)
const router = new Router({
  mode: 'history',
  routes: [
    {
      path: '/auth/signup',
      name: 'Signup',
      components: {
        header: AppHeader,
        auth: Signup
      }
    },
    {
      path: '/auth/signin',
      name: 'Signin',
      components: {
        header: AppHeader,
        auth: Signin
      }
    },
    {
       path: '/',
       alias: '/boards',
       name: 'home',
      components: {
          header: AppHeader,
          boards: Boards
      },
    },
    {
      path: '/boards/personal/:boardId/momos',
      name: 'MemoLists',
      components: {
         header: AppHeader,
         boards: Boards,
         Memos: Memos
      },
      props: {
       Memos: true
      },
  },
```

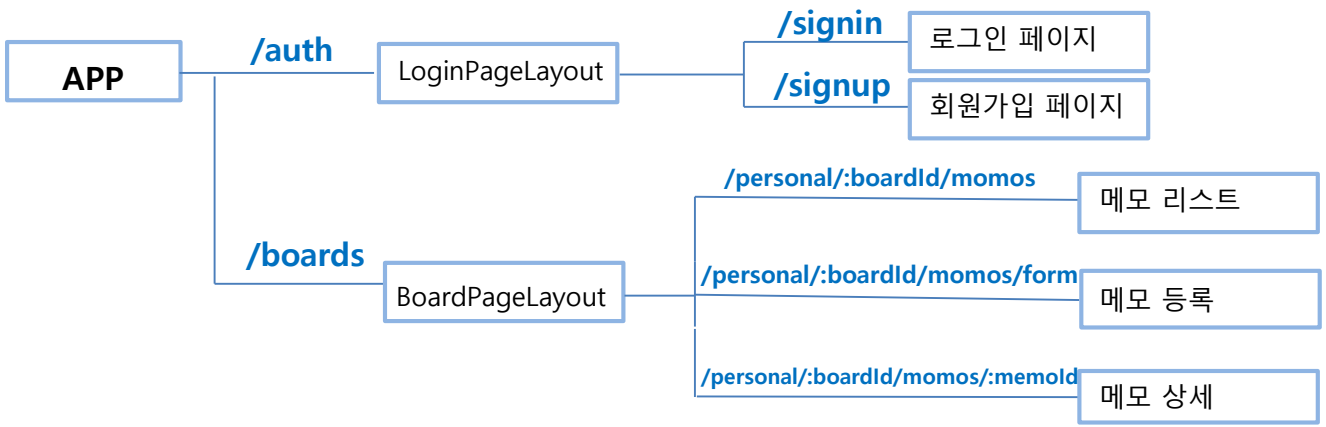
Router 설정을위한 index.js

```
<template>
  <div id="app">
    <b-container fluid="sm" class="container">

      <router-view name="header"/>

    <div id="main">
      <b-row class="side" >
        <b-col id="boards" cols="3"><router-view name="boards" /></b-col>
        <router-view name="auth" />
        <b-col cols="9"><router-view name="Memos" :key="$route.fullPath" /></b-col>
      </b-row>
    </div>

    </b-container>
  </div>
</template>
```

최상위 Component인 App.vue

중첩 라우트 구성

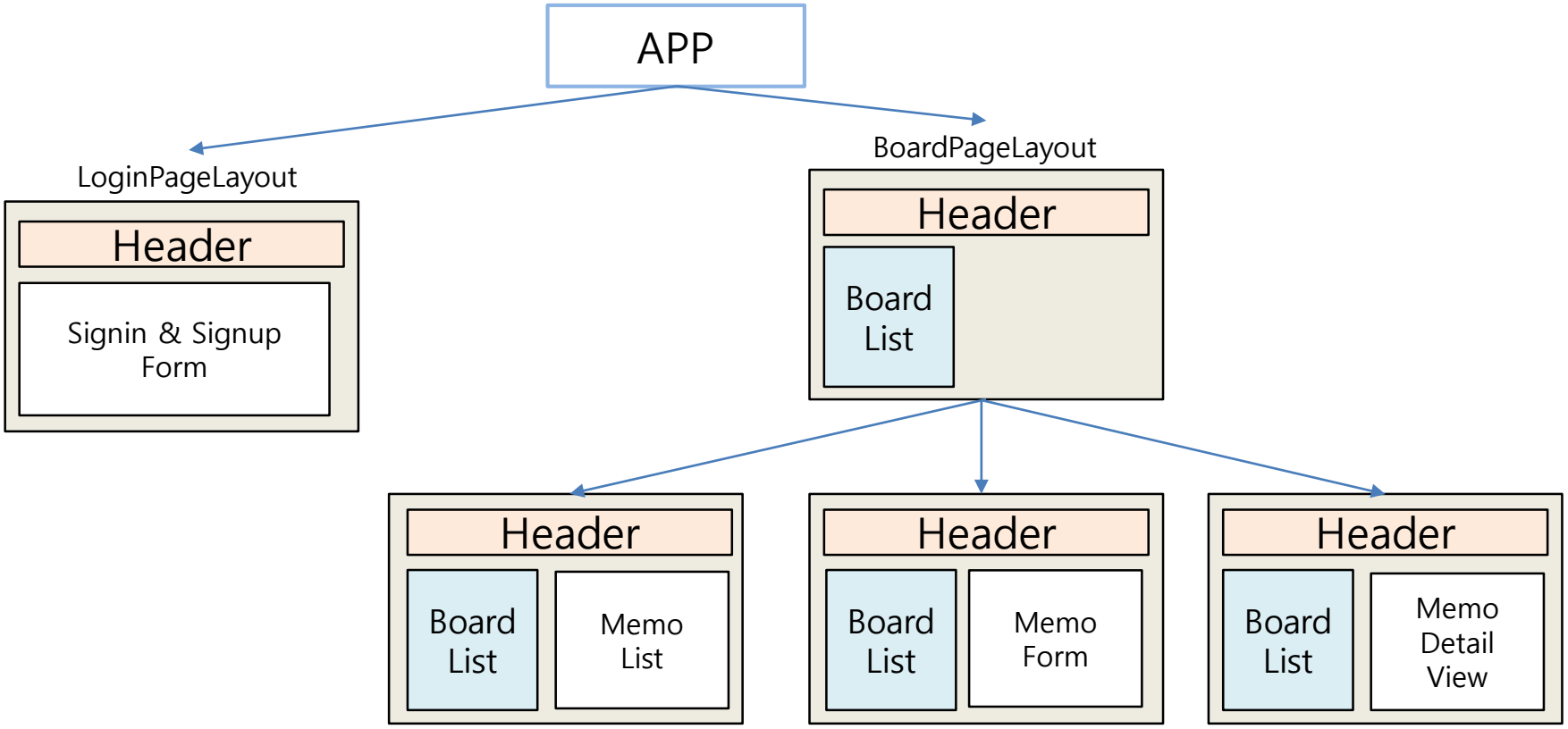
Components를 사용해 <router-view>내에 원하는 Component를 랜더링



FrontEnd – 컴포넌트 구성

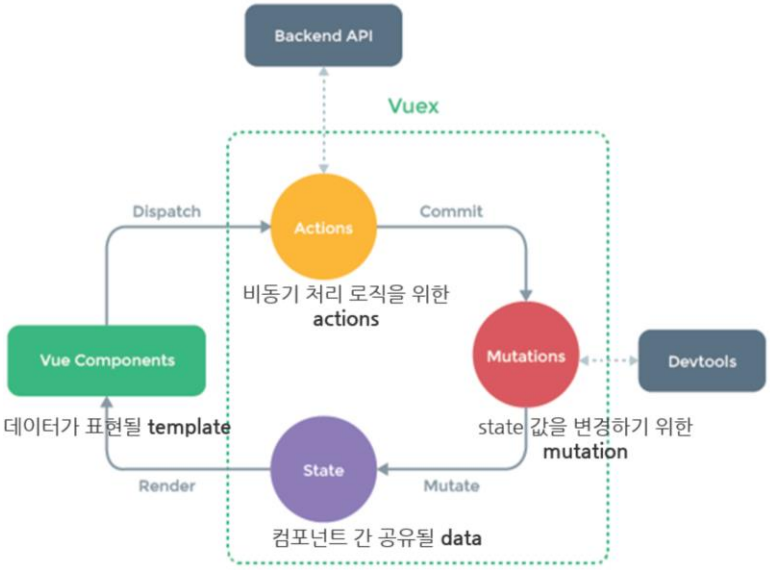
2개의 최상위 컴포넌트에 대한 구체적인 컴포넌트 구성은 아래와 같습니다.

- Vue Component: 화면에 비춰지는 Vue를 쪼개어 재활용이 가능한 형태로 관리하는 단위
- 색이 칠해진 컴포넌트는 재사용된 컴포넌트를 의미합니다.



FrontEnd – Vuex, 상태관리 도구

Vue에서의 기본적인 컴포넌트 통신 방법



Vuex의 상태관리 방법

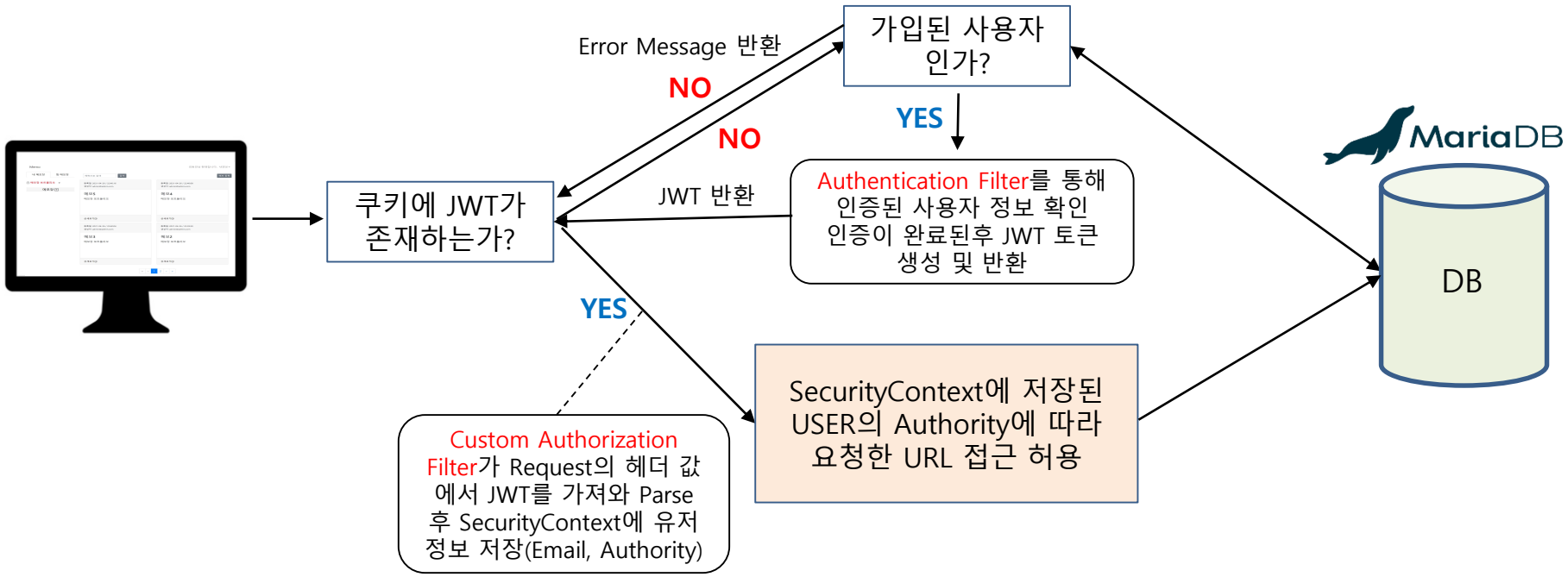
- Vue.js는 데이터가 단방향으로 데이터가 흐르는 구조입니다. 그래서 컴포넌트들은 자식 컴포넌트에게는 props로 데이터를 전달하고 부모 컴포넌트에게는 emit로 이벤트 발생을 알려야 합니다. 그래서 데이터가 어떻게 흘러갈지 예측은 쉬우나 컴포넌트들이 많아지면 많아질수록 컴포넌트 간에 데이터 전달이 어려워집니다.

- 이러한 문제를 해결하기위해 Vuex를 통해 중앙 집중식으로 상태 정보를 관리할 수 있게 되어 모든 컴포넌트들이 동일한 조건에서 접근할 수 있게 되었고 이에 따라 효율적으로 상태값(data)을 조작할 수 있었습니다.

- 소규모의 프로젝트에서 Vuex를 도입하는 것은 오히려 시간을 더 할애해야 하는 경우가 생길 수도 있지만 추후 프로젝트가 확장 될 것을 생각해 Vuex를 도입 해야겠다고 판단하였습니다.

BackEnd – 사용자 인증 과정

- Spring Security와 JWT를 적용한 인증 과정 흐름은 다음과 같습니다.



BackEnd – 접근 권한 제어

- **Spring Security 설정을 통한 URL 접근 제어**

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
    .cors()
    .and()
    .csrf()
    .disable()
    // JWT 인증 필터 보안 컨텍스트에 추가
    .addFilter(new JwtAuthenticationFilter(authenticationManager()))
    // JWT 인가 필터 보안 컨텍스트에 추가
    .addFilter(new JwtAuthorizationFilter(authenticationManager()))
    //세션 관리 비활성화
    .sessionManagement()
    .sessionCreationPolicy(SessionCreationPolicy.STATELESS);


    http.authorizeRequests()
    .antMatchers("/auth/signin").permitAll()
    .antMatchers("/auth/signup/**").permitAll()
    .antMatchers("/**").hasAnyAuthority("ADMIN","MEMBER")
    .antMatchers("/admin/**").hasAuthority("ADMIN");


}
```

기본적인 Client 요청 처리 과정 (1)

1. Client의 API 호출

- **사용자(MEMBER) 권한을 가진 Client가 메모장을 생성하기 위해 Server에 아래와 같이 요청합니다.**
  - ✓ POST http://localhost:8080/boards/personal
  - ✓ Client가 작성한 메모장 정보가 담긴 json 형태의 text를 RequestBody에 담습니다.

```
async addPersonalBoard({commit}, payload){
  const result = await api.post('/boards/personal', payload)

  if(result.status === 201){
      payload.pboardid = result.data.pboardid;
      await commit(ADD_PERSONALBOARD, payload)
  }
},
```

2. Client 권한 체크 및 URL Mapping 처리

- Server에서는 Client가 인증된 Client인지 권한을 가진 Client인지 접근 권한 제어 설정에 따라 확인한 후 해당 Method를 수행합니다.

기본적인 Client 요청 처리 과정 (2)

3. API Controller 역할

- Controller는 Client요청으로부터 Request Body 데이터를 DTO로 받아 적절한 Service에 인자로 넘겨 해당 Service를 호출합니다. Method가 정상적으로 실행되었다면 지정된 데이터를 Response의 Body로 지정된 Http Status와 함께 전송합니다.

```
@PostMapping("/boards/personal")
public ResponseEntity<Object> addPersonalBoards(@RequestBody PersonalBoard board){
    personalMemoService.addPersonalBoard(board);
    PersonalBoard returnboard = board;
    return new ResponseEntity<>(returnboard, HttpStatus.CREATED);
}
```

기본적인 Client 요청 처리 과정 (3)

3. **Service의 역할**
   - 요청에 따른 Business Logic을 수행합니다.

```java
@Service
public class PersonalMemoServiceImpl {
    @Autowired
    private PersonalMemoMapper personalMemoMapper;

    public void addPersonalBoard(PersonalBoard board) {
        personalMemoMapper.addPersonalBoard(board);
    }
```

기본적인 Client 요청 처리 과정 (4)

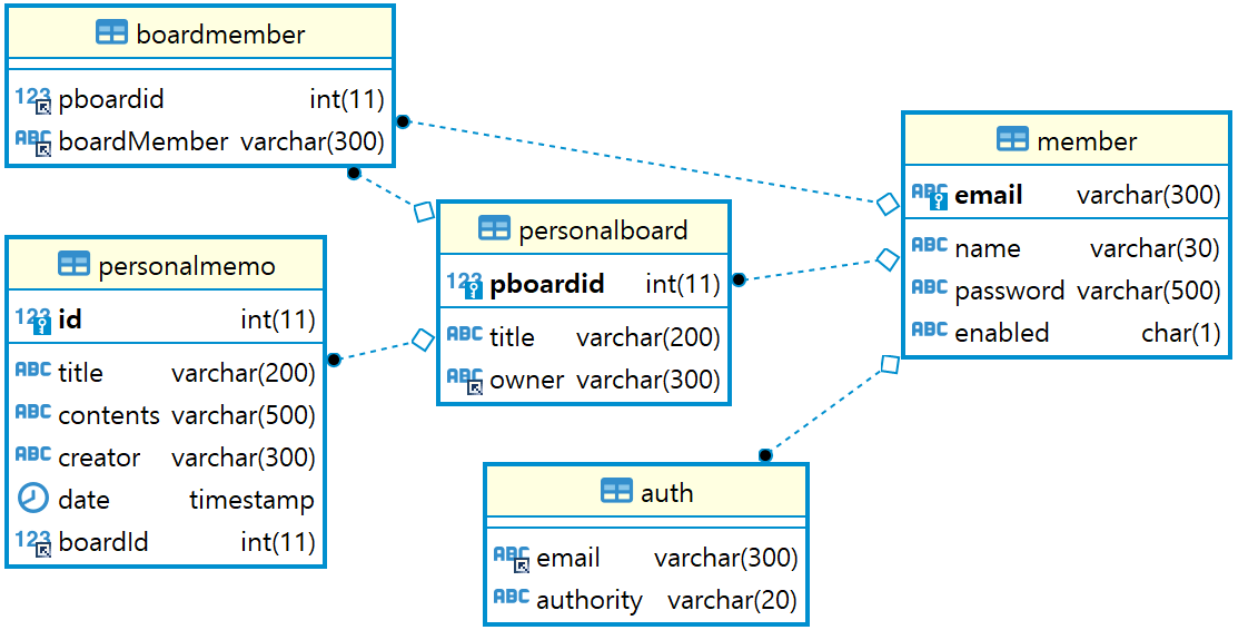
3. Mabatis(SQL Mapper)의 역할

- 실제 DB에 접근하여 적절한 결과값을 반환하는 역할을 수행합니다.

```java
@Mapper
public interface PersonalMemoMapper {

    List<PersonalBoard> findPersonalBoardByEmail(String email);
    List<PersonalBoard> findTeamBoardByEmail(String email);
    void addPersonalBoard(PersonalBoard board);
```

```xml
<insert id="addPersonalBoard">
    INSERT into personalBoard(title, owner) values(#{title}, #{owner})
    <!-- LAST_INSERT_ID가 추가된 object를 반환 -->
    <selectKey resultType="int" keyProperty="pboardid" order="AFTER">
        SELECT LAST_INSERT_ID()
    </selectKey>
</insert>
```

Database



MariaDB 를 선택한 이유

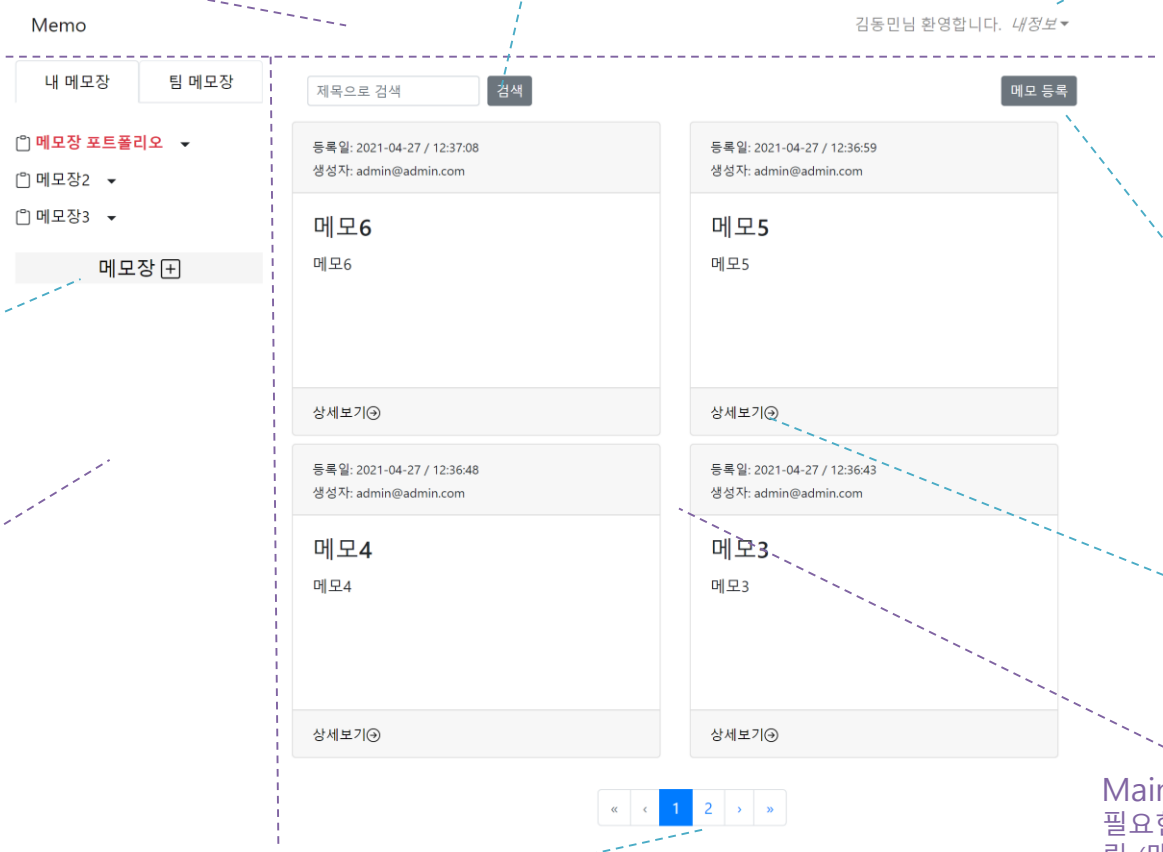
- **MySQL이 여전히 무료이지만 Oracle로 넘어간후로 Java처럼 언제 유료로 전환할지 알 수 없습니다.**
- **Oracle을 배우기는 했으나 MySQL쪽이 더 친숙했고 MariaDB는 MySQL의 개발자들이 개발한 DB이기에 MySQL과 사용법이 거의 흡사합니다.**

프로젝트 설명
(메인 화면)

Header
모든 페이지 재사용

로그인
로그아웃

로그인
로그아웃



메모 등록 페이지로 이동

메모장 추가

메모 내용 상세보기 페이지로 이동 (수정, 삭제 가능)

메모장 리스트
모든 페이지 재사용

Main
필요한 Component들을 랜더링 (메모 리스트, 메모 등록폼, 메모 상세페이지)

페이징

프로젝트 설명
(회원가입, 로그인)

Memo

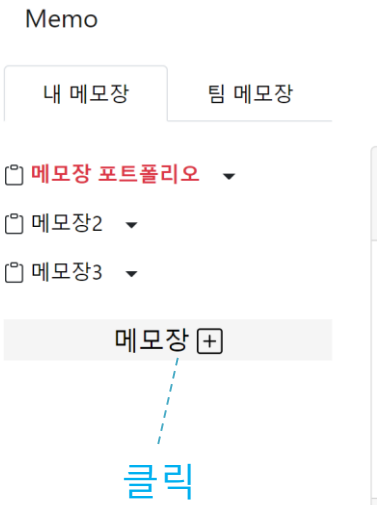
로그인

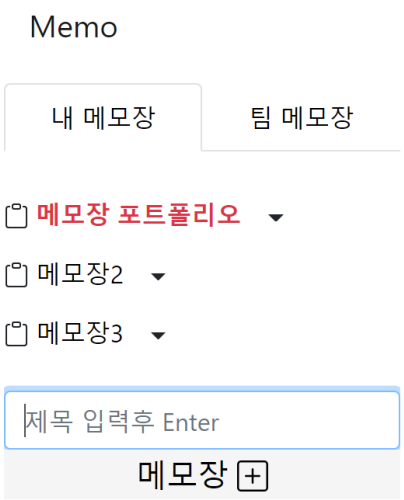
Memo

로그인

필요한 정보를 입력후 가입, 로그인 버튼을 누르면 완료

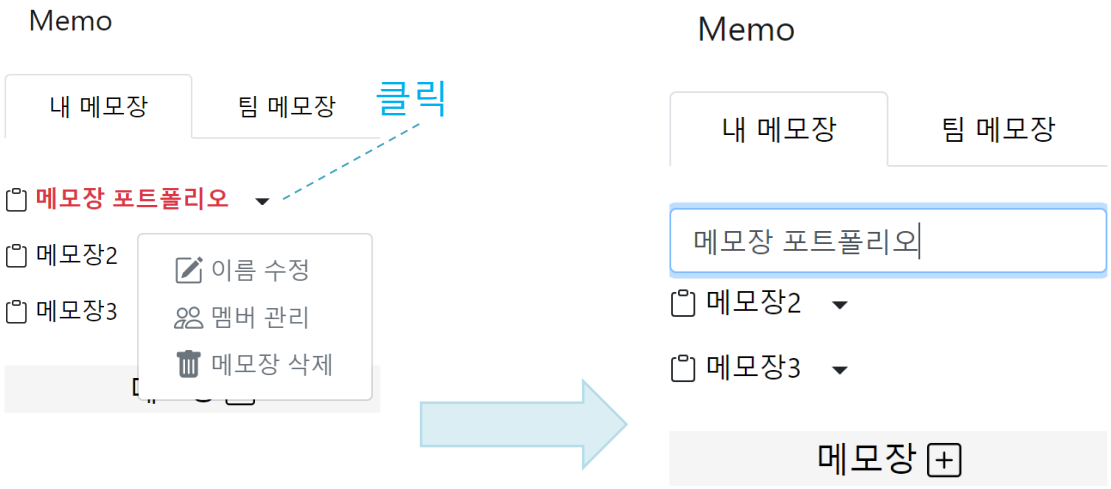
프로젝트 설명
(메모장 추가)





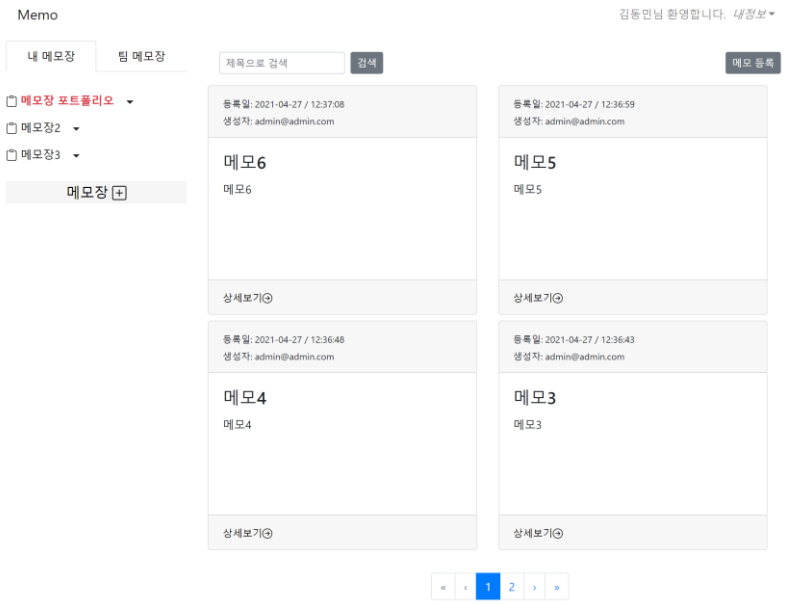
메모장+ 버튼을 클릭하면 메모장을 생성할수 있는 Form이 자동으로 생성됩니다. 제목을 입력후 Enter를 입력하면 새로운 메모장이 생성됩니다.

프로젝트 설명
(메모장 수정, 삭제)



각 메모장 옆의 세모 버튼을 누르면 메모장의 이름 변경, 삭제를 할 수 있는 메뉴바가 드롭됩니다.
이름 수정을 클릭하면 해당 메모장은 편집모드로 바뀌며 오른쪽 사진과 같이 이름을 수정할 수 있습니다.

프로젝트 설명
(메모 리스트)

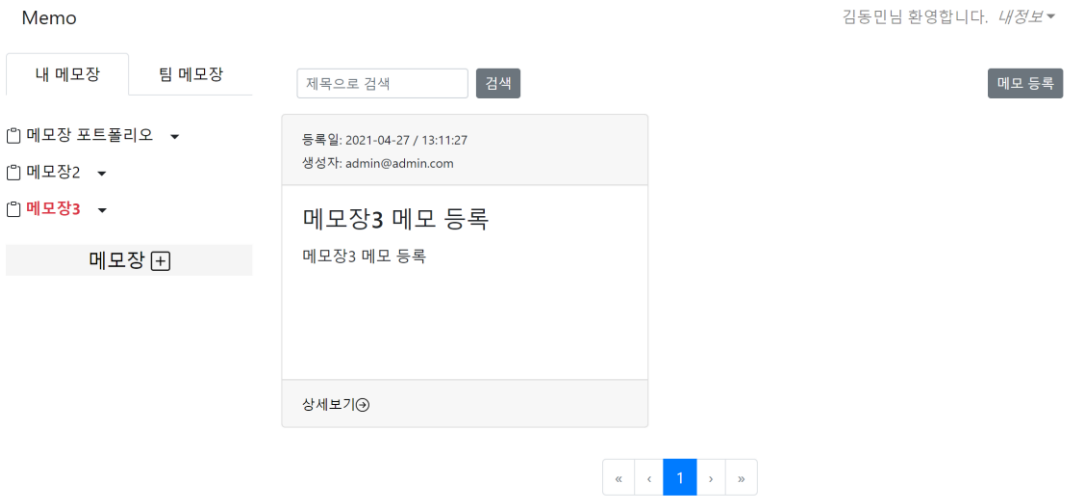


메모장이 선택되면 선택된 메모장은 빨간색으로 글자색이 바뀝니다. 해당 메모장에 메모가 등록되어있다면 리스트를 불러옵니다. 한 페이지에 4개의 메모가 표시되게 했으며 4개가 넘어가면 페이징처리가 됩니다.

제목으로 검색을 실행하면 해당 메모장의 메모의 제목중 검색 키워드에 부합하는 메모들의 리스트를 가져옵니다.

등록된 메모가 없다면 아래와 같이 "메모를 등록해 주세요."가 표시됩니다.

프로젝트 설명
(메모 등록 페이지)



메모장이 선택되면 선택된 메모장은

메모 리스트 페이지에서 메모등록 버튼을 누르면 해당 메모장에 메모를 등록할 수 있는 페이지로 이동합니다.

메모 정보를 입력후 등록을 클릭하면 메모가 등록되고 목록으로를 클릭하면 메모 리스트 페이지로 돌아갑니다.

프로젝트 설명
(메모 상세 페이지 – 수정, 삭제)

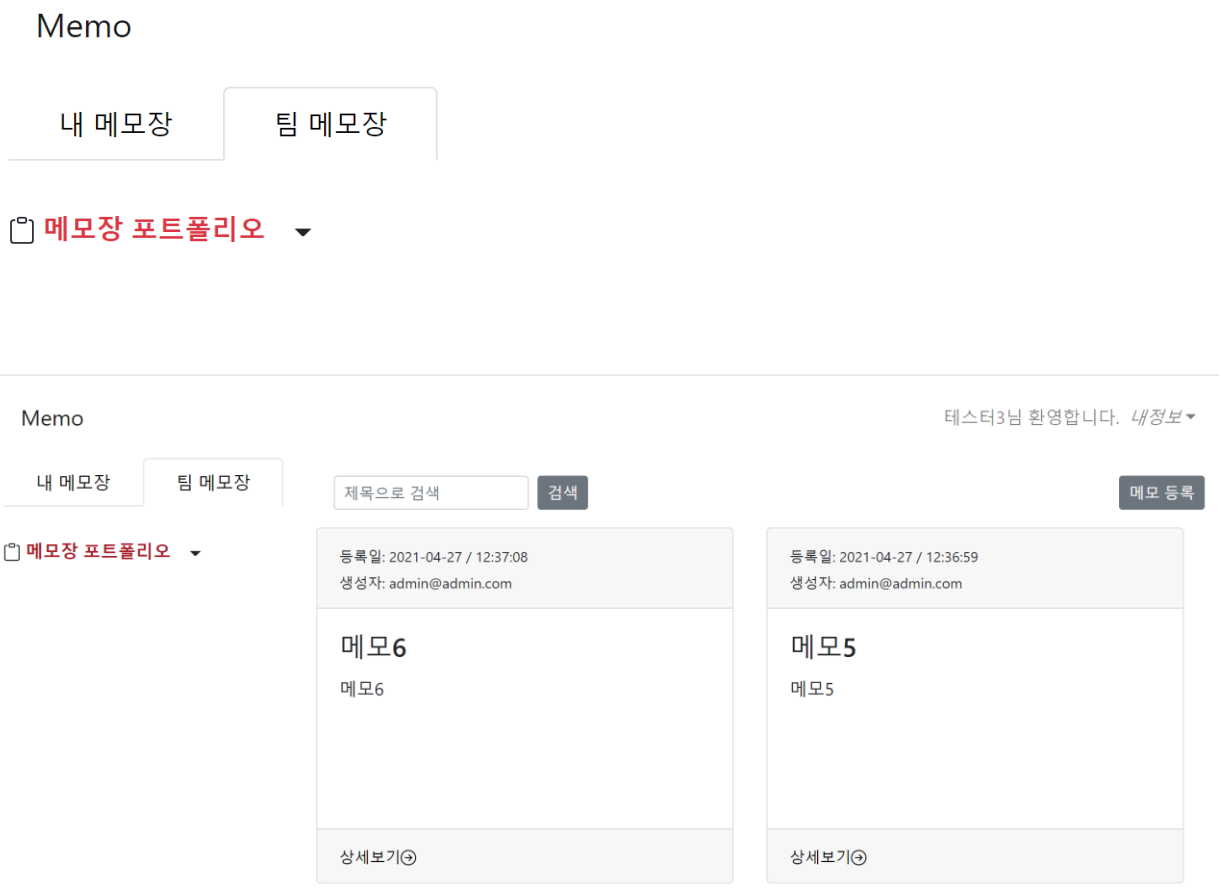
메모 리스트에서 상세보기를 클릭하면 메모 상세보기 페이지로 이동하며 해당 메모의 정보를 수정, 삭제 할 수 있습니다. 수정, 삭제는 작성자만이 가능합니다. 수정 버튼을 누르면 편집 모드로 변하며 버튼의 메뉴도 아래 사진과 같이 바뀝니다.

프로젝트 설명
(메모장 공유 - 1)

멤버 관리를 선택하면 오른쪽과 같이 메모장의 멤버를 추가, 삭제 할수 있는 모달창이 열립니다.

현재 공유된 멤버의 목록이 확인되며 공유할 멤버를 추가 할 수 있고 멤버를 삭제해서 공유를 취소시킬 수 있습니다.

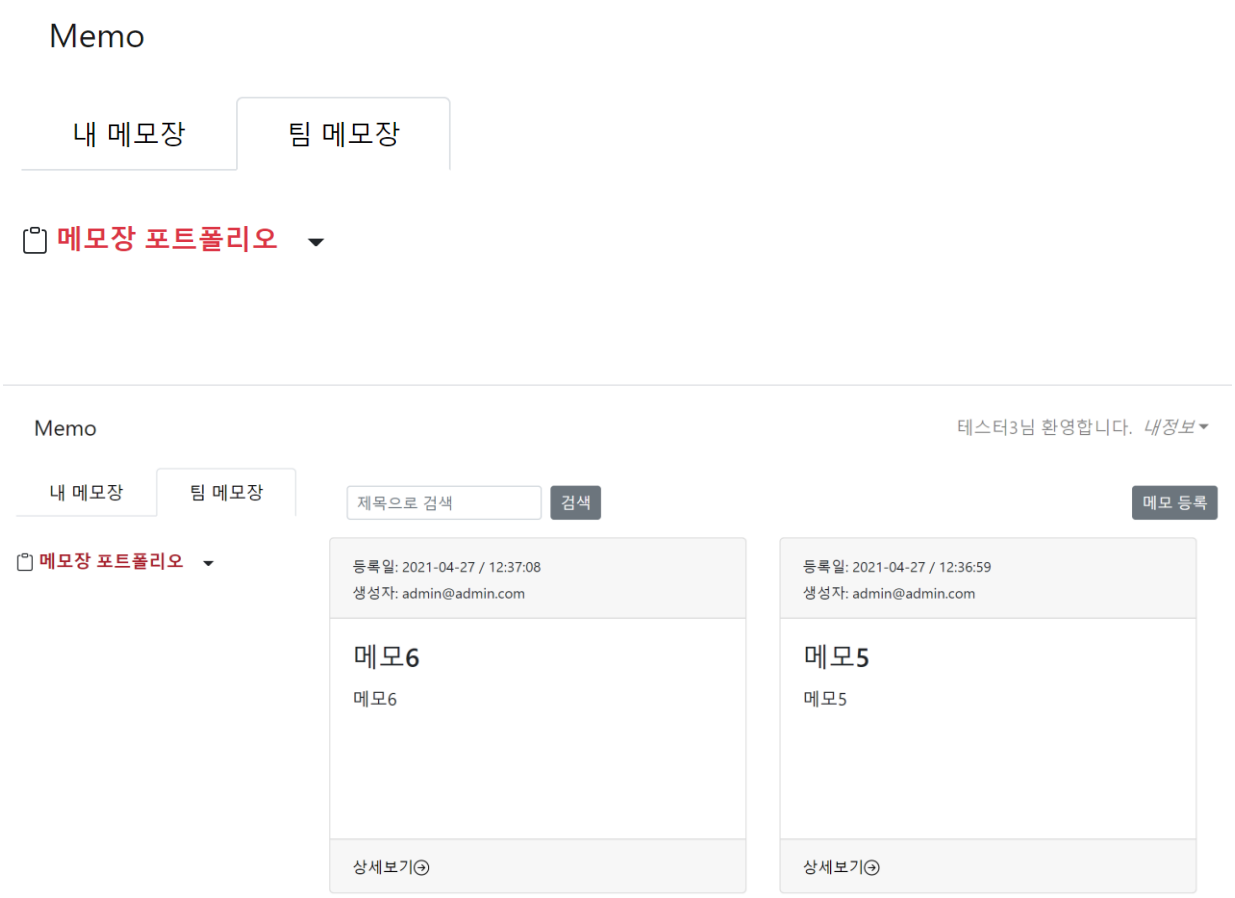
프로젝트 설명
(메모장 공유 - 1)



내가 메모장의 생성자가 아닌 멤버인 메모장은 팀 메모장 탭에서 확인 할 수 있습니다.

메모장의 생성자인 admin@admin.com의 메모장이 공유된 것을 확인할 수 있습니다.

프로젝트 설명
(메모장 공유 - 2)

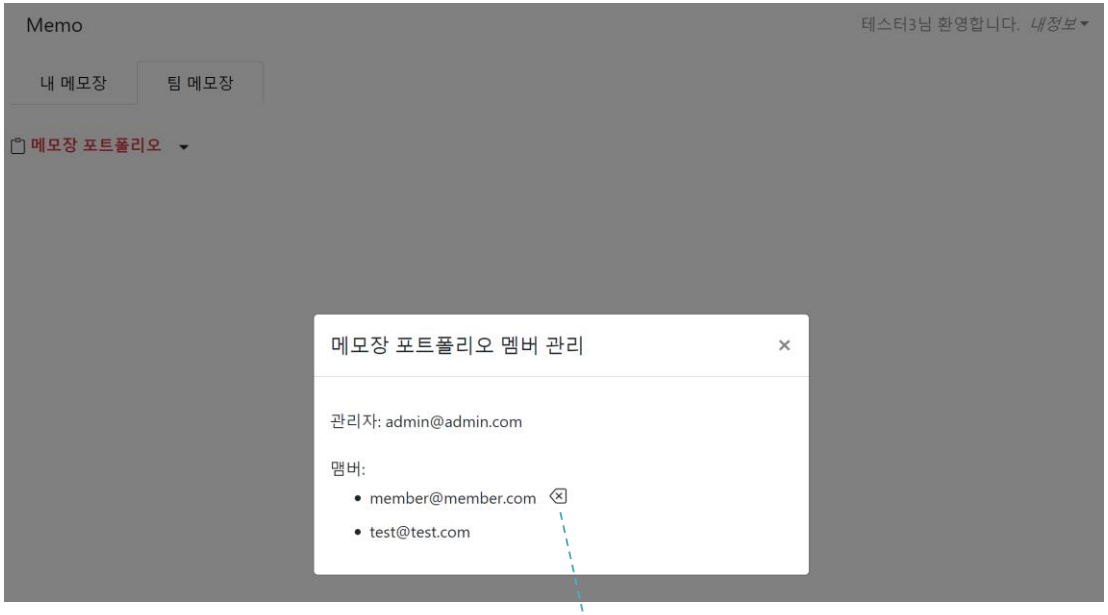


내가 생성자가 아닌 멤버인 메모장은 팀 메모장 탭에서 확인 할 수 있습니다.

메모장의 생성자인 admin@admin.com의 메모장이 공유된 것을 확인할 수 있습니다.

공유된 메모장에 메모를 등록 할 수 있습니다.

프로젝트 설명
(메모장 공유 - 3)



클릭시 메모장 탈퇴

내가 멤버로 속한 메모장의 멤버 관리 모달에서는 메모장의 관리자와 멤버를 확인 할 수 있고 생성자의 멤버관리 화면과 다르게 본인 아이디 옆에만 X 버튼으로 메모장을 탈퇴할 수 있게 설정하였습니다.

메모장을 탈퇴하면 해당 메모장은 생성자가 다시 멤버로 추가 하기 전까지 팀 메모장에서 사라집니다.

맺음말

포트폴리오를 마치면서 든 생각과 앞으로의 포부에 대해 짧게나마 맺음말을 적어보려 합니다.

우선 FrontEnd부터 BackEnd까지 대학 시절이나 국비 학원에서 배우지 않은 기술들을 도입하며 하나의 서비스를 온전히 구현해야 되는 일이 쉽지 않았습니다. 특히 이전 까지 Client Side Rendering(SPA)를 제대로 경험해 본적이 없는 저로서는 처음에는 화면을 구현하는 모든 것들이 어렵게 느껴졌습니다. 특히 Vuex를 사용한 중앙집중식 상태관리의 개념은 더욱 생소했고 한정된 시간에서 괜히 생소한 스팩으로 프로젝트 준비를 시작했나 하는 생각도 있었습니다.

그럼에도 불구하고 Vue, JWT 인증방식, RESTful API서버를 고집한 것은 현재 개발 생태계가 제가 대학을 다니던 10년 전과는 너무나 다르고 해당 기술들의 확장성과 트랜드를 쫓고 싶었던 마음이 컸습니다. 특히 RESTful을 꼭 구현하고 싶었는데 그 이유는 웹 뿐만 아니라 모바일 HTTP 프로토콜을 이용하는 어떤 Device라도 같은 서버에서 통신할 수 있다는 점 때문 이었습니다.

해당 프로젝트는 공개된 서버 API를 만드는 것은 아니었기에 REST의 성숙도면에서 보자면 높지 않지만 RESTful이라는 것이 어떻게 동작하고 사용될 수 있을지 고민해보는 과정에서 보람이 있었습니다. 프로젝트를 개발하는 과정에서 막히는 부분이 생기면 책도 여러권 구매하고 공식 사이트도 매일매일 찾아보며 습득 속도가 조금씩 더 빨라졌고 어느정도 제가 원하는 기능을 구현 할 수 있게 되는 재밌는 경험을 할 수 있었습니다.

해당 프로젝트가 모든 면에서 완벽한 프로그램은 아닐지라도 실력 있는 개발자가 되기 위해서 안주하지 않고 좀 더 효율적이고 좀 더 성능적으로 우수한 프로그램을 만들 수 있도록 계속해서 정진해야겠다는 마음가짐으로 시작했고 조금은 더 성장하지 않았나 하는 자신감도 얻게 해준 의미 있는 프로젝트 였다고 생각합니다.

감사합니다.