2024-2 오픈소스 개발 프로젝트01

SW중심대학사업단 강재구

(학연산 742호, 010-2278-8192, kangjk@cbnu.ac.kr)

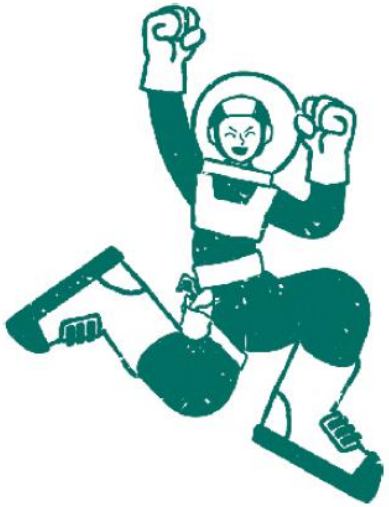
07

충북대학교
SW중심대학사업단

03. SBB 서비스 개발하기 (3)

03

SBB 서비스 개발하기



3-10 추천 기능 추가하기

3-11 앵커 기능 추가하기

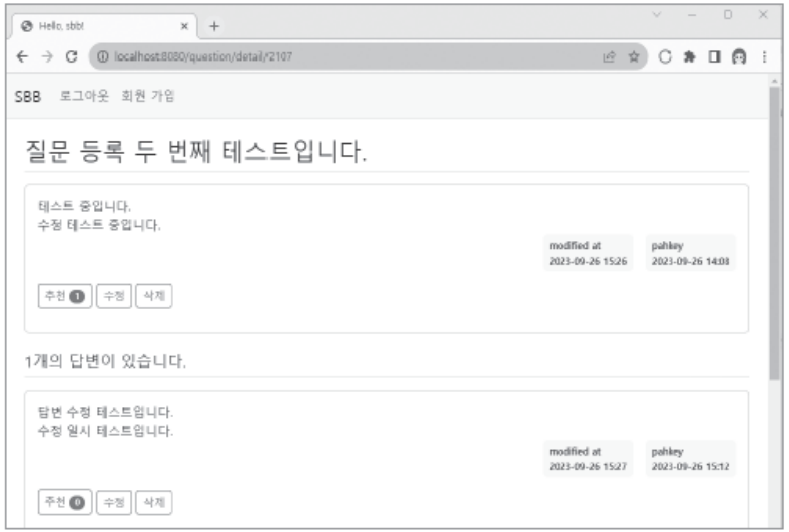
3-12 마크다운 적용하기

3-13 검색 기능 추가하기

3-10 추천 기능 추가하기

▪ 수정 일시 표시하기

- 우리는 SNS에서 마음에 드는 게시물이나 콘텐츠에 '좋아요'나, '추천' 등과 같은 표시를 남김
- SBB 게시판에도 [추천] 버튼을 통해 질문이나 답변을 본 다른 사용자들이 반응을 남길 수 있도록 '추천' 기능을 구현해 보자



3-10 추천 기능 추가하기

▪ 엔티티에 속성 추가하기

- 질문 또는 답변의 '추천' 기능을 구현하려면
  질문이나 답변을 천한 사용자(SiteUser)가 DB에 저장될 수 있도록
  관련 속성을 질문, 답변 엔티티에 추가해야 함

1. 먼저, 질문 엔티티에 추천인([추천] 버튼을 클릭한 사용자)을 저장하기 위한 voter라는 이름의 속성을 추가해 보자.

   하나의 질문에 여러 사람이 추천할 수 있고
   한 사람이 여러 개의 질문을 추천할 수 있음.

   따라서 @ManyToMany 애너테이션을 사용해야 함

3-10 추천 기능 추가하기

▪ 엔티티에 속성 추가하기

1. • /question/Question.java

```
(... 생략 ...)
import java.util.List;
import java.util.Set;

(... 생략 ...)
import jakarta.persistence.ManyToOne;
import jakarta.persistence.ManyToMany;

(... 생략 ...)
public class Question {
    (... 생략 ...)


    private LocalDateTime modifyDate;

    @ManyToMany
    Set<SiteUser> voter;
}
```

@ManyToMany 애너테이션과 함께
Set<SiteUser> voter를 작성해
voter 속성을 다대다 관계로 설정하여
질문 엔티티에 추가함.

이때 다른 속성과 달리 Set 자료형으로 작성한 이유는
voter 속성값이 서로 중복되지 않도록 하기 위해서임.

List 자료형과 달리 여기서는
Set 자료형이 voter 속성을
관리하는데 효율적임

3-10 추천 기능 추가하기

▪ 엔티티에 속성 추가하기

2. 답변 엔티티에도 같은 방법으로 voter 속성을 추가해 보자

• /answer/Answer.java

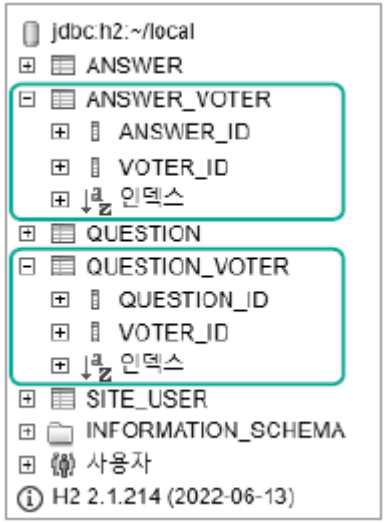
```
(... 생략 ...)
import java.time.LocalDataTime;
import java.util.Set;

(... 생략 ...)
import jakarta.persistence.ManyToOne;
import jakarta.persistence.ManyToMany;
(... 생략 ...)
public class Answer {
    (... 생략 ...)

    private LocalDateTime modifyDate;

    @ManyToMany
    Set<SiteUser> voter;
}
```

3-10 추천 기능 추가하기

▪ 엔티티에 속성 추가하기

3. 질문과 답변 엔티티에 voter 속성을 추가하였으니 H2 콘솔을 확인해 보자.

   author 속성을 추가할 때와 달리 QUESTION_VOTER, ANSWER_VOTER라는 테이블이 생성된 것을 확인할 수 있음



3-10 추천 기능 추가하기

▪ 엔티티에 속성 추가하기

3. 이렇게 @ManyToMany 애너테이션을 사용해
다대다 관계로 속성을 생성하면
새로운 테이블을 만들어 관련 데이터를 관리함.

   여기서 생성된 테이블의 인덱스 항목을 펼쳐 보면
서로 연관된 엔티티의 고유 번호(즉, ID)가 기본키로 설정되어
다대다 관계임을 알 수 있음

3-10 추천 기능 추가하기

▪ 질문 추천 기능 생성하기

1. 질문을 추천할 수 있는 버튼 위치는 질문 상세 화면이 적절함.
   질문 상세 템플릿을 다음과 같이 수정해 보자

• /templates/question_detail.html

```
(... 생략 ...)
<!-- 질문 -->
<h2 class="border-bottom py-2" th:text="${question.subject}"></h2>
<div class="card my-3">
    <div class="card-body">
        (... 생략 ...)
        <div class="my-3">
            <a href="javascript:void(0);" class="recommend btn btn-sm
btn-outline-secondary"
               th:data-uri="@{|/question/vote/${question.id}|}">
               추천
               <span class="badge rounded-pill bg-success"
               th:text="${#lists.size(question.voter)}"></span>
            </a>
            <a th:href="@{|/question/modify/${question.id}|}" class="btn
btn-sm btn-outline-secondary"
               sec:authorize="isAuthenticated()"
               th:if="${question.author != null and #authentication.
getPrincipal().getUsername() == question.author.username}"
               th:text="수정"></a>
            <a href="javascript:void(0);"
            th:data-uri="@{|/question/delete/${question.id}|}"
               class="delete btn btn-sm btn-outline-secondary"
sec:authorize="isAuthenticated()"
               th:if="${question.author != null and #authentication.
getPrincipal().getUsername() == question.author.username}"
               th:text="삭제"></a>
        </div>
    </div>
</div>
(... 생략 ...)
```

추천 버튼을 클릭하는 이벤트를 얻기 위한 클래스

3-10 추천 기능 추가하기

▪ 질문 추천 기능 생성하기

1. [추천] 버튼을 [수정] 버튼 왼쪽에 추가하기 위한 코드를 작성함.

   lists.size 메서드에 question.voter를 사용하여 추천 수도 함께 보이도록 함.

   [추천] 버튼을 클릭하면 href의 속성이 javascript:void(0)으로 되어 있어서
   아무런 동작도 하지 않음.

   하지만 class 속성에 recommend를 적용해
   자바스크립트로 data-uri에 정의된 URL이 호출되도록 할 것임.

   따라서 [삭제] 버튼과 마찬가지로 [추천] 버튼을 눌렀을 때
   메시지가 적힌 팝업 창을 통해 추천을 진행할 것임

3-10 추천 기능 추가하기

▪ 질문 추천 기능 생성하기

2. 이어서 [추천] 버튼을 클릭했을 때 '정말로 추천하시겠습니까?'라는 메시지 창이 나타나도록 다음과 같이 자바스크립트 코드를 추가해 보자

• /question/question_detail.html

```
(... 생략 ...)
<script layout:fragment="script" type='text/javascript'>
const delete_elements = document.getElementsByClassName("delete");
Array.from(delete_elements).forEach(function(element) {
    element.addEventListener('click', function() {
        if(confirm("정말로 삭제하시겠습니까?")) {
            location.href = this.dataset.uri;
        };
    });
});
```

```
const recommend_elements = document.getElementsByClassName("recommend");
Array.from(recommend_elements).forEach(function(element) {
  element.addEventListener('click', function() {
      if(confirm("정말로 추천하시겠습니까?")) {
          location.href = this.dataset.uri;
      };
  });
});
</script>
</html>
```

3-10 추천 기능 추가하기

▪ 질문 추천 기능 생성하기

2. [추천] 버튼에 recommend 클래스가 적용되어 있으므로

   [추천] 버튼을 클릭하면 '정말로 추천하시겠습니까?'라는
   메시지가 담긴 팝업 창이 나타나고,

   [확인]을 선택하면 data-uri 속성에 정의한
   URL인 @{|/question/vote/${question.id}|}이 호출될 것임

3-10 추천 기능 추가하기

▪ 질문 추천 기능 생성하기

3. 추천인을 저장할 수 있도록 추천 기능을 다음과 같이 QuestionSerivce에 추가해 보자.

• /question/QuestionService.java

```
(... 생략 ...)
public class QuestionService {

    (... 생략 ...)

    public void vote(Question question, SiteUser siteUser) {
        question.getVoter().add(siteUser);
        this.questionRepository.save(question);
    }
}
```

이와 같이 로그인한 사용자를 질문 엔티티에 추천인으로
저장하기 위해 vote 메서드를 추가함

3-10 추천 기능 추가하기

▪ 질문 추천 기능 생성하기

4. [추천] 버튼을 눌렀을 때 GET 방식으로 호출되는 @{|/question/vote/${question.id}|} URL을 처리하기 위해 QuestionController에 코드를 추가해 보자

• /question/QuestionController.java

```
(... 생략 ...)
public class QuestionController {

    (... 생략 ...)

    @PreAuthorize("isAuthenticated()")
    @GetMapping("/vote/{id}")
    public String questionVote(Principal principal, @PathVariable("id") Integer
id) {
        Question question = this.questionService.getQuestion(id);
        SiteUser siteUser = this.userService.getUser(principal.getName());
        this.questionService.vote(question, siteUser);
        return String.format("redirect:/question/detail/%s", id);
    }
}
```

3-10 추천 기능 추가하기

▪ 질문 추천 기능 생성하기

4. 이와 같이 questionVote 메서드를 추가함.

   다른 기능과 마찬가지로 추천 기능도 로그인한 사람만 사용할 수 있도록 @PreAuthorize("isAuthenticated( )") 애너테이션을 적용함.

   그리고 앞서 작성한 QuestionService의 vote 메서드를 호출하여 사용자(siteUser)를 추천인(voter)으로 저장함.

   오류가 없다면 추천인을 저장한 후 질문 상세 화면으로 리다이렉트함

3-10 추천 기능 추가하기

▪ 질문 추천 기능 생성하기

5. 질문 상세 화면의 질문 내용 부분을 보면 [추천] 버튼이 생겼을 것임.
이 버튼을 클릭하여 버튼이 잘 작동하는지 확인해 보자

3-10 추천 기능 추가하기

▪ 질문 추천 기능 생성하기

6. [추천] 버튼을 클릭하면 다음과 같이 메시지 창이 등장함.
메시지 창의 [확인] 버튼을 클릭하면 다시 질문 상세 화면으로 돌아가고
[추천] 버튼에 추천인 숫자가 변경됨

3-10 추천 기능 추가하기

▪ 답변 추천 기능 생성하기

1. 답변의 추천 수를 표시하고, 답변을 추천할 수 있는 버튼을 질문 상세 템플릿에 추가해 보자

• /templates/question_detail.html

```
(... 생략 ...)
<!-- 답변 반복 시작 -->
<div class="card my-3" th:each="answer : ${question.answerList}">
    <div class="card-body">
        (... 생략 ...)
        <div class="my-3">
            <a href="javascript:void(0);" class="recommend btn btn-sm
btn-outline-secondary"
               th:data-uri="@{|/answer/vote/${answer.id}|}">
               추천
               <span class="badge rounded-pill bg-success"
               th:text="${#lists.size(answer.voter)}"></span>
            </a>
            <a th:href="@{|/answer/modify/${answer.id}|}" class="btn btn-sm
btn-outline-secondary"
               sec:authorize="isAuthenticated()"
               th:if="${answer.author != null and #authentication.getPrincipal().
getUsername() == answer.author.username}"
               th:text="수정"></a>
            <a href="javascript:void(0);" th:data-uri="@{|/answer/delete/${answer.
id}|}"
               class="delete btn btn-sm btn-outline-secondary"
sec:authorize="isAuthenticated()"
               th:if="${answer.author != null and #authentication.getPrincipal().
getUsername() == answer.author.username}"
               th:text="삭제"></a>
        </div>
    </div>
</div>
<!-- 답변 반복 끝  -->
(... 생략 ...)
```

3-10 추천 기능 추가하기

▪ 답변 추천 기능 생성하기

1. 질문 추천 기능을 만들 때와 마찬가지로 답변 영역의 상단에
답변을 추천할 수 있는 버튼을 생성함.

   이 역시 추천 버튼에 class="recommend"가 적용되어 있으므로
추천 버튼을 클릭하면 '정말로 추천하시겠습니까?'라는 메시지가 적힌
팝업 창이 나타나고 [확인]을 선택하면 data-uri 속성에 정의한 URL이 호출될 것임

3-10 추천 기능 추가하기

▪ 답변 추천 기능 생성하기

2. 답변을 추천한 사람을 저장하기 위해 다음과 같이 AnswerService를 수정해 보자.

• /answer/AnswerService.java

```
(... 생략 ...)
public class AnswerService {

    (... 생략 ...)

    public void vote(Answer answer, SiteUser siteUser) {
        answer.getVoter().add(siteUser);
        this.answerRepository.save(answer);
    }
}
```

AnswerService에 추천인을 저장하는 vote 메서드를 추가함

3-10 추천 기능 추가하기

▪ 답변 추천 기능 생성하기

3. [추천] 버튼을 눌렀을 때 GET 방식으로 호출되는 @{|/answer/vote/${answer.id}|} URL을 처리하기 위해 다음과 같이 AnswerController에 코드를 추가해 보자

• /answer/AnswerController.java

```
(... 생략 ...)
public class AnswerController {

    (... 생략 ...)

    @PreAuthorize("isAuthenticated()")
    @GetMapping("/vote/{id}")
    public String answerVote(Principal principal, @PathVariable("id") Integer id)
{
        Answer answer = this.answerService.getAnswer(id);
        SiteUser siteUser = this.userService.getUser(principal.getName());
        this.answerService.vote(answer, siteUser);
        return String.format("redirect:/question/detail/%s", answer.getQuestion().
getId());
    }
}
```

3-10 추천 기능 추가하기

▪ 답변 추천 기능 생성하기

3. 이와 같이 answerVote 메서드를 추가함.

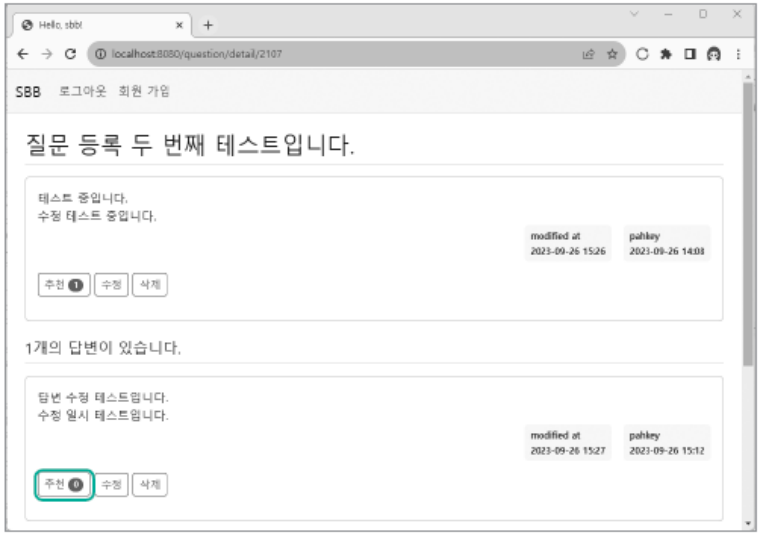
   추천은 로그인한 사람만 가능해야 하므로
   @PreAuthorize("isAuthenticated( )") 애너테이션을 적용함.

   그리고 앞서 작성한 Answer Service의 vote 메서드를 호출하여 추천인을 저장함.
   오류가 없다면 추천인을 저장한 후 질문 상세 화면으로 리다이렉트함

3-10 추천 기능 추가하기

▪ 답변 추천 기능 생성하기

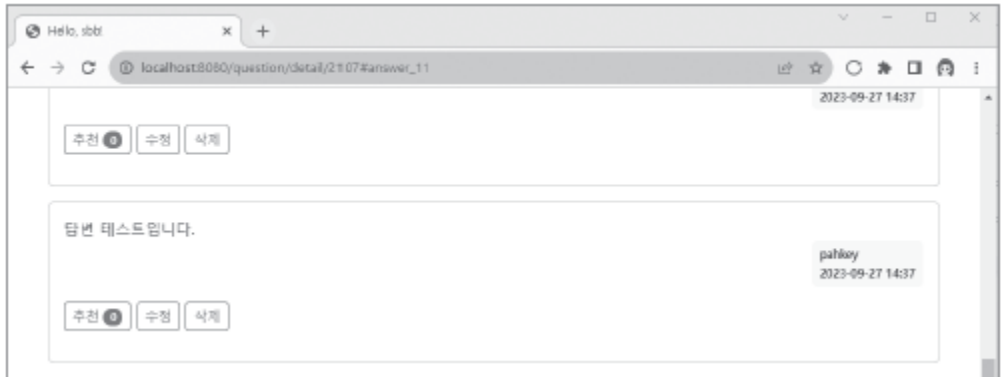
4. 질문 상세 화면에서 답변 추천 기능도 확인해 보자.
   답변의 [추천] 버튼을 누르면 메시지 창이 등장하고,
   [확인] 버튼을 누르면 [추천] 버튼의 숫자가 변경됨



3-11 앵커 기능 추가하기

- 이번에는 SBB의 문제점을 해결하려고 함
- 발견된 문제점은 답변을 작성하거나 수정하면 페이지 상단으로 스크롤이 이동해서 자신이 작성한 답변을 확인하려면 다시 스크롤을 내려서 확인해야 한다는 점임
- 이 문제는 답변을 추천한 경우에도 동일하게 발생함

- HTML에는 URL 호출 시 원하는 위치로 이동해 주는 앵커(anchor) 태그
  즉, <a> 태그가 있는데,
  이를 활용하면 답변 등록, 답변 수정, 답변 추천 시 앵커 태그를 이용하여 원하는 위치로 이동할 수 있음



3-11 앵커 기능 추가하기

▪ 답변 앵커 추가하기

- 앵커 태그인 <a> 태그를 활용해 사용자가 다른 웹 페이지로 이동하거나 동일한 페이지 내에서 특정 위치로 스크롤하도록 만들 수 있음
- 먼저 답변 작성, 수정 시에 이동해야 할 앵커 태그를 질문 상세 템플릿에 추가해 보자

• /templates/question_detail.html

```
(... 생략 ...)
<!-- 답변의 개수 표시 -->
<h5 class="border-bottom my-3 py-2"
    th:text="¦${#lists.size(question.answerList)}개의 답변이 있습니다.¦"></h5>
<!-- 답변 반복 시작 -->
<div class="card my-3" th:each="answer : ${question.answerList}">
    <a th:id="¦answer_${answer.id}¦"></a>
    <div class="card-body">
(... 생략 ...)
```

3-11 앵커 기능 추가하기

▪ 답변 앵커 추가하기

- 답변이 반복되어 표시되도록 하는 th:each 문장 바로 다음에 <ath:id="|answer_${answer.id}|">와 같이 앵커 태그를 추가함
- 앵커 태그의 id 속성은 유일한 값이어야 하므로 답변의 id값을 사용함

▪ 리다이렉트 수정하기

- 이제 답변을 등록하거나 수정할 때 앞서 지정한 앵커 태그를 사용해 원하는 화면 위치로 이동할 수 있도록 코드를 수정하려고 함

3-11 앵커 기능 추가하기

▪ 리다이렉트 수정하기

- 먼저 답변 컨트롤러에서 답변 등록 또는 답변 수정을 하고 난 뒤, URL을 리다이렉트하는 코드를 살펴보자

```
return String.format("redirect:/question/detail/%s", answer.getQuestion().getId());
```

- 이와 같은 코드에 앵커를 포함하여 다음과 같이 수정할 수 있음

```
return String.format("redirect:/question/detail/%s#answer_%s",
    answer.getQuestion().getId(), answer.getId());
```

3-11 앵커 기능 추가하기

- **리다이렉트 수정하기**
  - 리다이렉트되는 질문 상세 페이지 URL에 #answer_%s를 이와 같이 삽입하여 앵커를 추가한 것임
  - 이때 수정해야 하는 곳은 총 3곳으로 답변 등록, 수정, 추천 부분에서 리다이렉트와 관련된 코드를 수정하면 됨
  - 이와 관련된 코드를 수정하기 전에 답변 서비스를 잠시 먼저 수정한 후 진행하도록 하자

3-11 앵커 기능 추가하기

▪ 답변 서비스 수정하기

- 답변 컨트롤러에서 답변이 등록된 위치로 이동하려면
  반드시 답변 객체, 즉 Answer 객체가 필요함

- 그동안 AnswerService에서는 답변 등록 시 답변 객체를 리턴하지 않으므로
  다음과 같이 AnswerService를 먼저 수정해 보자

• /answer/AnswerService.java

```
(... 생략 ...)
public class AnswerService {

    (... 생략 ...)

    public Answer create(Question question, String content, SiteUser author) {
        Answer answer = new Answer();
        answer.setContent(content);
        answer.setCreateDate(LocalDateTime.now());
        answer.setQuestion(question);
        answer.setAuthor(author);
        this.answerRepository.save(answer);
        return answer;
    }

    (... 생략 ...)
}
```

3-11 앵커 기능 추가하기

▪ 답변 컨트롤러 수정하기

- 답변 컨트롤러의 리다이렉트와 관련된 코드를 다음과 같이 직접 수정해 보자

• /answer/AnswerController.java

```
(... 생략 ...)
public class AnswerController {

    (... 생략 ...)

    @PreAuthorize("isAuthenticated()")
    @PostMapping("/create/{id}")
    public String createAnswer(Model model, @PathVariable("id") Integer id,
            @Valid AnswerForm answerForm, BindingResult bindingResult, Principal
principal) {
        Question question = this.questionService.getQuestion(id);
        SiteUser siteUser = this.userService.getUser(principal.getName());
        if (bindingResult.hasErrors()) {
            model.addAttribute("question", question);
            return "question_detail";
        }
        Answer answer = this.answerService.create(question, answerForm.getCon
tent(), siteUser);
        return String.format("redirect:/question/detail/%s#answer_%s",
                answer.getQuestion().getId(), answer.getId());
    }
```

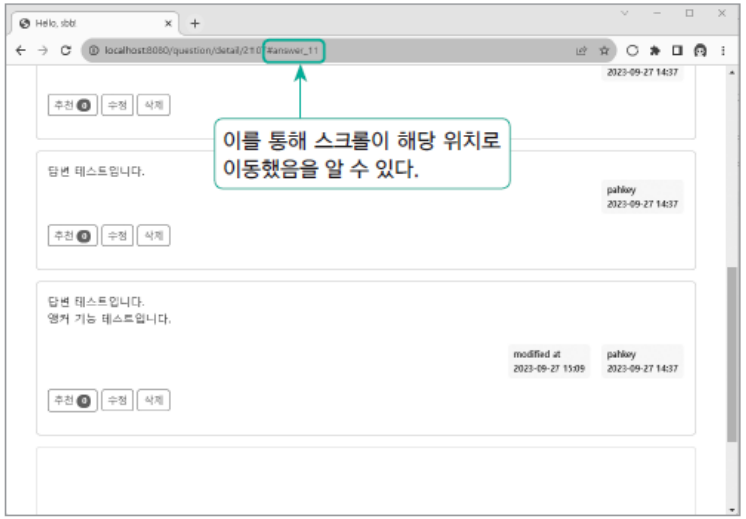
```
    (... 생략 ...)
    @PreAuthorize("isAuthenticated()")
    @PostMapping("/modify/{id}")
    public String answerModify(@Valid AnswerForm answerForm, @PathVariable("id")
Integer id,
            BindingResult bindingResult, Principal principal) {
        if (bindingResult.hasErrors()) {
            return "answer_form";
        }
        Answer answer = this.answerService.getAnswer(id);
        if (!answer.getAuthor().getUsername().equals(principal.getName())) {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "수정 권한이
없습니다.");
        }
        this.answerService.modify(answer, answerForm.getContent());
        return String.format("redirect:/question/detail/%s#answer_%s",
                answer.getQuestion().getId(), answer.getId());
    }

    (... 생략 ...)
```

```
    @PreAuthorize("isAuthenticated()")
    @GetMapping("/vote/{id}")  ← 답변 추천 부분

    public String answerVote(Principal principal, @PathVariable("id") Integer id)
{
        Answer answer = this.answerService.getAnswer(id);
        SiteUser siteUser = this.userService.getUser(principal.getName());
        this.answerService.vote(answer, siteUser);
        return String.format("redirect:/question/detail/%s#answer_%s",
                answer.getQuestion().getId(), answer.getId());
    }
}
```

3-11 앵커 기능 추가하기

▪ 답변 앵커 기능 확인하기

- 질문 상세 페이지에서 답변을 등록, 수정, 추천을 실행하면 자신이 작업한 답변 부분으로 돌아옴
- 즉, 스크롤이 지정한 앵커로 이동함을 확인할 수 있음

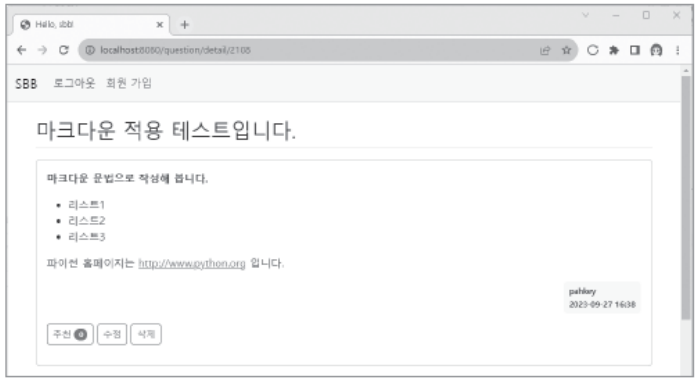


3-12 마크다운 적용하기

- 깃허브(Github), 노션(Notion)과 같이 우리가 자주 사용하는 서비스에서는 글을 작성할 때 마크다운(markdown)이라는 도구를 사용함

- 마크다운은 텍스트 기반의 마크업 언어로, HTML과 달리 쉽고 간단한 문법을 사용하며 텍스트 편집기를 통해 웹상에서 글자를 강조하거나 제목, 목록, 이미지, 링크 등을 추가할 때도 유용하게 활용할 수 있음

- SBB 서비스에도 질문이나 답변 등의 글쓰기 작성 도구로 마크다운을 적용해 보자



3-12 마크다운 적용하기

▪ 마크다운 문법 살펴보기

▪ 목록 표시하기

- SBB에서 내용에 여러 내용을 나열한 목록을 표시하기 위해 다음과 같이 작성했다고 가정

```
* 자바
* 스프링 부트
* 알고리즘
```

- 이 문자열을 마크다운 해석기가 HTML로 변환하면 실제 화면에서는 다음과 같이 보임

```
• 자바
• 스프링 부트
• 알고리즘
```

- 만약 순서가 있는 목록을 표시하고 싶다면 다음과 같이 작성함

```
1. 하나
1. 둘
1. 셋
```

- 그러면 마크다운 해석기는 다음과 같은 결과를 출력함

```
1. 하나
2. 둘
3. 셋
```

3-12 마크다운 적용하기

▪ 마크다운 문법 살펴보기

▪ 강조 표시하기

- 작성한 글자에 강조 표시를 하려면 강조할 텍스트 양쪽에 **를 넣어 감싸 보자

```
스프링 부트는 **자바**로 만들어진 웹 프레임워크이다.
```

- 출력 결과는 다음과 같음

스프링 부트는 **자바**로 만들어진 웹 프레임워크이다.

3-12 마크다운 적용하기

▪ 마크다운 문법 살펴보기

▪ 링크 표시하기

- 질문이나 답변 내용에 링크를 추가하고 싶다면
  다음과 같이 '[링크명](링크주소)' 문법을 적용하여 생성할 수 있음

```
스프링 홈페이지는 [https://spring.io](https://spring.io) 입니다.
```

- 출력 결과는 다음과 같음

스프링 홈페이지는 https://spring.io입니다.

3-12 마크다운 적용하기

▪ 마크다운 문법 살펴보기

▪ 소스 코드 표시하기

- 소스 코드는 백쿼트 ` 3개를 연이어 붙여 위아래로 감싸면 생성할 수 있음

필요한 소스 코드는 다음과 같다.

```
package com.mysite.sbb;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
```

```
@Controller
public class HelloController {
    @GetMapping("/hello")
    @ResponseBody
    public String hello() {
        return "Hello Spring Boot Board";
    }
}
```

3-12 마크다운 적용하기

▪ 마크다운 문법 살펴보기

▪ 소스 코드 표시하기

- 출력 결과는 다음과 같음

필요한 소스 코드는 다음과 같다.

```
package com.mysite.sbb;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class HelloController {
    @GetMapping("/hello")
    @ResponseBody
    public String hello() {
        return "Hello Spring Boot Board";
    }
}
```

3-12 마크다운 적용하기

▪ 마크다운 문법 살펴보기

▪ 인용 표시하기

- 질문이나 답변 내용에서 어떤 문장을 인용했다는 표시를 하려면
  다음과 같이 >를 문장 맨 앞에 입력하고 1칸 띄어쓰기를 한 다음 인용구를 입력함

```
> 마크다운은 Github에서 사용하는 글쓰기 도구이다.
```

- 출력 결과는 다음과 같을 것임

> 마크다운은 Github에서 사용하는 글쓰기 도구이다.

- 사용자 입장에서 이와 같이 마크다운 문법을 사용하면
  질문이나 답변을 남길 때 간단하면서 가독성 좋은 글을 작성할 수 있음

3-12 마크다운 적용하기

▪ 마크다운 설치하기

- SBB에 마크다운 기능을 추가하려면 마크다운 라이브러리를 설치해야 함
- 다른 라이브러리를 설치할 때와 마찬가지로
  다음과 같이 build.gradle 파일을 수정하여 마크다운을 설치하자

• build.gradle

```
(... 생략 ...)

dependencies {
    (... 생략 ...)
    implementation 'org.thymeleaf.extras:thymeleaf-extras-springsecuri
ty6:3.1.1.RELEASE'
    implementation 'org.commonmark:commonmark:0.21.0'
}

(... 생략 ...)
```

- [Gradle → Refresh Gradle Project]를 클릭하여 변경 사항을 적용하면
  0.21.0 버전의 commonmark 라이브러리가 설치됨. 로컬 서버도 한번 재시작하자

3-12 마크다운 적용하기

▪ 마크다운 컴포넌트 작성하기

- 라이브러리 설치를 마쳤으니 이제 질문이나 답변의 '내용' 부분에 마크다운을 적용해 보자
- 사실 컨트롤러에서 질문이나 답변을 조회한 후에 마크다운 라이브러리를 적용하면 변환된 HTML을 얻을 수 있음

- 하지만 여기서는 개별적으로 사용하기보다 좀 더 범용적으로 사용할 수 있는 마크다운 컴포넌트를 만들고 타임리프 템플릿에서 작성한 마크다운 컴포넌트를 사용하는 방법을 알아보자

- 그러기 위해 먼저 다음과 같이 CommonUtil이라는 이름으로 마크다운 컴포넌트를 작성해보자
- com.mysite.sbb 패키지에 CommonUtil.java를 만들어 보자

3-12 마크다운 적용하기

▪ 마크다운 컴포넌트 작성하기

• CommonUtil.java

```
package com.mysite.sbb;

import org.commonmark.node.Node;
import org.commonmark.parser.Parser;
import org.commonmark.renderer.html.HtmlRenderer;
import org.springframework.stereotype.Component;

@Component
public class CommonUtil {
    public String markdown(String markdown) {
        Parser parser = Parser.builder().build();
        Node document = parser.parse(markdown);
        HtmlRenderer renderer = HtmlRenderer.builder().build();
        return renderer.render(document);
    }
}
```

3-12 마크다운 적용하기

▪ 마크다운 컴포넌트 작성하기

- @Component 애너테이션을 사용하여 CommonUtil 클래스를 생성함
- 이렇게 하면 이제 CommonUtil 클래스는 스프링 부트가 관리하는 빈으로 등록됨
- 빈으로 등록된 컴포넌트는 템플릿에서 사용할 수 있음

- CommonUtil 클래스에는 markdown 메서드를 생성함
- markdown 메서드는 마크다운 텍스트를 HTML 문서로 변환하여 리턴함
- 즉, 마크다운 문법이 적용된 일반 텍스트를 변환된 HTML로 리턴함

3-12 마크다운 적용하기

▪ 템플릿에 마크다운 적용하기

- 마크다운 문법은 질문 상세 페이지에서 사용하므로 question_detail.html에 코드를 추가해 보자

• /templates/question_detail.html

```
(... 생략 ...)
<!-- 질문 -->
<h2 class="border-bottom py-2" th:text="${question.subject}"></h2>
<div class="card my-3">
    <div class="card-body">
        <div class="card-text" th:utext="${@commonUtil.markdown(question.
content)}"></div>
        <div class="d-flex justify-content-end">
        (... 생략 ...)
```

```
<!-- 답변 반복 시작 -->
<div class="card my-3" th:each="answer : ${question.answerList}">
    <a th:id="|answer_${answer.id}|"></a>
    <div class="card-body">
        <div class="card-text" th:utext="${@commonUtil.markdown(answer.
content)}"></div>
        <div class="d-flex justify-content-end">
        (... 생략 ...)
```

3-12 마크다운 적용하기

▪ 템플릿에 마크다운 적용하기

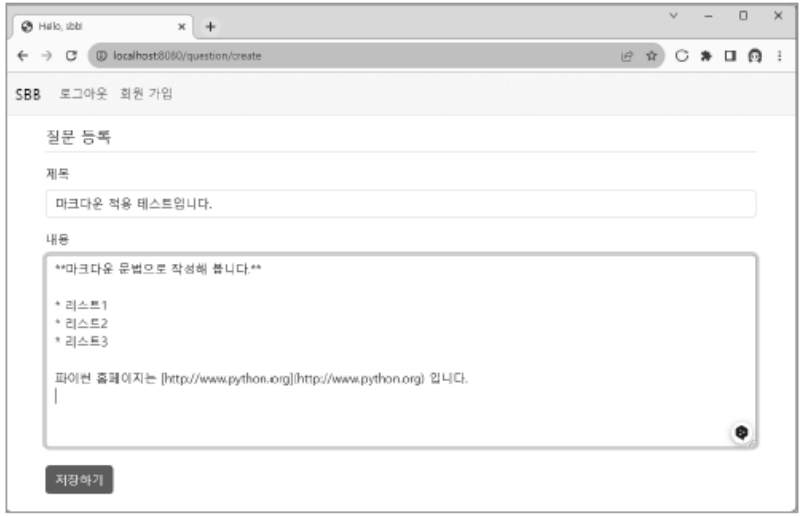
- 질문과 답변 영역에 마크다운을 각각 적용하기 위해 줄 바꿈을 표시하려고 사용한 기존의 style="white-space: pre-line;" 스타일을 삭제하고 ${@commonUtil.markdown (question.content)}와 같이 마크다운 컴포넌트를 적용함
- 이때 th:text가 아닌 th:utext를 사용한 부분에 주의하자
- 만약 th:utext 대신 th:text를 사용할 경우 HTML의 태그들이 이스케이프(escape)처리되어 화면에 그대로 보일 것임
- 마크다운으로 변환된 HTML 문서를 제대로 표시하려면 이스케이프 처리를 하지 않고 출력하는 th:utext를 사용해야 함

3-12 마크다운 적용하기

▪ 마크다운 확인하기

질문 또는 답변을 마크다운 문법으로 작성하면 브라우저에서 어떻게 보이는지 확인해 보자

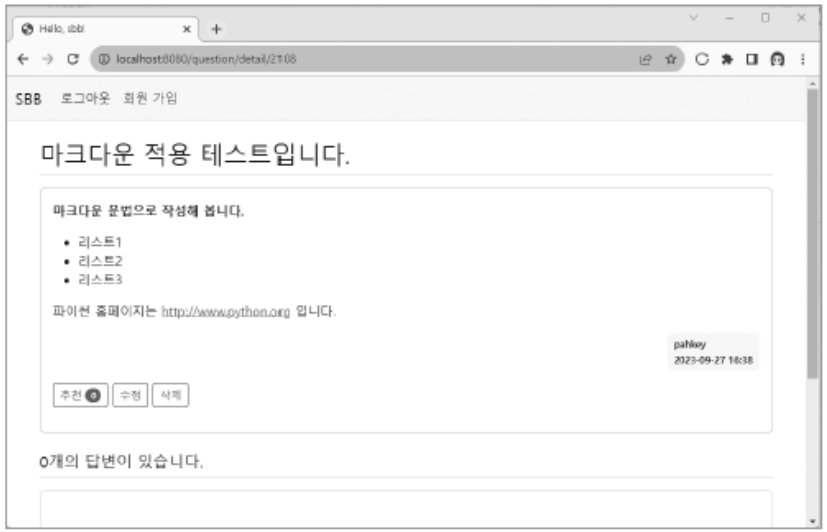
1. 질문 내용에 마크다운 문법을 적용하여 등록해 보자



3-12 마크다운 적용하기

▪ 마크다운 확인하기

2. 그러면 다음과 같이 마크다운 문법이 적용된 것을 확인할 수 있을 것임



3-13 검색 기능 추가하기

- SBB의 질문 목록 화면에 찾고자 하는 내용을 키워드로 검색하여
  해당 검색어와 관련 있는 질문을 조회할 수 있는 기능을 만들어 보자

- 그러기 위해 검색어를 입력할 수 있는 텍스트 창을 만들고
  검색어를 입력하여 조회하면 검색어에 해당되는 질문들이 보여야 함

3-13 검색 기능 추가하기

▪ 검색 기능 구현하기

- SBB는 질문과 답변 데이터가 계속 쌓이는 게시판이므로
  검색 기능은 필수라고 할 수 있음

- 검색 대상으로는 질문 제목, 질문 내용, 질문 작성자, 답변 내용,
  답변 작성자 정도로 정하면 좋음

- 예를 들어 '스프링'을 검색하면 '스프링'이라는 문자열이 제목, 내용, 질문 작성자,
  답변, 답변 작성자에 존재하는지 찾아보고 그 결과를 화면에 보여 주도록 하자

3-13 검색 기능 추가하기

▪ 검색 기능 구현하기

- 이런 조건으로 검색하려면 다음과 같은 SQL 쿼리가 실행되어야 함

```
select
    distinct q.id,
    q.author_id,
    q.content,
    q.create_date,
    q.modify_date,
    q.subject
from question q
left outer join site_user u1 on q.author_id=u1.id
left outer join answer a on q.id=a.question_id
left outer join site_user u2 on a.author_id=u2.id
where
    q.subject like '%스프링%'
    or q.content like '%스프링%'
    or u1.username like '%스프링%'
    or a.content like '%스프링%'
    or u2.username like '%스프링%'
```

3-13 검색 기능 추가하기

▪ 검색 기능 구현하기

- 이 쿼리문은 question, answer, site_user 테이블을 대상으로 '스프링'이라는 문자열이 포함된 데이터를 검색함
- question 테이블을 기준으로 answer, site_user 테이블을 아우터 조인(outer join)하여 문자열 '스프링'을 검색함
- 만약 아우터 조인 대신 이너 조인(inner join)을 사용하면 합집합이 아닌 교집합으로 검색되어 데이터 검색 결과가 누락될 수 있음
- 총 3개의 테이블을 대상으로 아우터 조인하여 검색하면 중복된 결과가 나올 수 있어서 select 문에 distinct를 함께 적어 중복을 제거함
- 이 쿼리문 그대로 사용하지 않고 이전과 마찬가지로 JPA를 사용하여 자바 코드로 만들 것임

3-13 검색 기능 추가하기

▪ 검색 기능 구현하기

▪ JPA의 Specification 인터페이스 사용하기

- 앞의 쿼리에서 본 것과 같이 여러 테이블에서 데이터를 검색해야 할 경우에는 JPA가 제공하는 Specification 인터페이스를 사용하는 것이 편리함
- 이 인터페이스는 DB 검색을 더 유연하게 다룰 수 있고, 복잡한 검색 조건도 처리할 수 있음
- Specification 인터페이스를 어떻게 사용할 수 있는지 예제를 통해서 더 자세히 알아보자

3-13 검색 기능 추가하기

▪ 검색 기능 구현하기

▪ JPA의 Specification 인터페이스 사용하기

- 검색 기능을 구현하기 위해 다음과 같이 QuestionService에 search 메서드를 추가해 보자

• /question/QuestionService.java

```
(... 생략 ...)
import com.mysite.sbb.answer.Answer;
import jakarta.persistence.criteria.CriteriaBuilder;
import jakarta.persistence.criteria.CriteriaQuery;
import jakarta.persistence.criteria.Join;
import jakarta.persistence.criteria.JoinType;
import jakarta.persistence.criteria.Predicate;
import jakarta.persistence.criteria.Root;
import org.springframework.data.jpa.domain.Specification;
(... 생략 ...)

public class QuestionService {

    private final QuestionRepository questionRepository;
```

```
    private Specification<Question> search(String kw) {
        return new Specification<>() {
            private static final long serialVersionUID = 1L;
            @Override
            public Predicate toPredicate(Root<Question> q, CriteriaQuery<?>
query, CriteriaBuilder cb) {
                query.distinct(true);  // 중복을 제거
                Join<Question, SiteUser> u1 = q.join("author", JoinType.LEFT);
                Join<Question, Answer> a = q.join("answerList", JoinType.LEFT);
                Join<Answer, SiteUser> u2 = a.join("author", JoinType.LEFT);
                return cb.or(cb.like(q.get("subject"), "%" + kw + "%"), // 제목
                        cb.like(q.get("content"), "%" + kw + "%"),  // 내용
                        cb.like(u1.get("username"), "%" + kw + "%"), // 질문 작성자
                        cb.like(a.get("content"), "%" + kw + "%"),    // 답변 내용
                        cb.like(u2.get("username"), "%" + kw + "%")); // 답변 작성자
            }
        };
    }
    (... 생략 ...)
}
```

3-13 검색 기능 추가하기

▪ 검색 기능 구현하기

▪ JPA의 Specification 인터페이스 사용하기

- search 메서드는 검색어를 가리키는 kw를 입력받아 쿼리의 조인문과 where문을 Specification 객체로 생성하여 리턴하는 메서드임

- 앞서 살펴본 쿼리를 자바 코드로 그대로 재현한 것임을 알 수 있음

- 위 코드에서 사용한 변수들에 대해서 자세히 살펴보자

- **q**: Root 자료형으로, 즉 기준이 되는 Question 엔티티의 객체를 의미하며 질문 제목과 내용을 검색하기 위해 필요하다.
- **u1**: Question 엔티티와 SiteUser 엔티티를 아우터 조인(여기서는 JoinType.LEFT로 아우터 조인을 적용한다.)하여 만든 SiteUser 엔티티의 객체이다. Question 엔티티와 SiteUser 엔티티는 author 속성으로 연결되어 있어서 q.join("author")와 같이 조인해야 한다. u1 객체는 질문 작성자를 검색하기 위해 필요하다.
- **a**: Question 엔티티와 Answer 엔티티를 아우터 조인하여 만든 Answer 엔티티의 객체이다. Question 엔티티와 Answer 엔티티는 answerList 속성으로 연결되어 있어서 q.join("answerList")와 같이 조인해야 한다. a 객체는 답변 내용을 검색할 때 필요하다.
- **u2**: 바로 앞에 작성한 a 객체와 다시 한번 SiteUser 엔티티와 아우터 조인하여 만든 SiteUser 엔티티의 객체로 답변 작성자를 검색할 때 필요하다.

3-13 검색 기능 추가하기

▪ 검색 기능 구현하기

▪ JPA의 Specification 인터페이스 사용하기

- 검색어(kw)가 포함되어 있는지를 like 키워드로 검색하기 위해
  제목, 내용, 질문 작성자, 답변 내용, 답변 작성자 각각에 cb.like를 사용함

- 최종적으로 cb.or로 OR 검색(여러 조건 중 하나라도 만족하는 경우
  해당 항목을 반환하는 검색 조건을 말한다.)이 되게 함

- 앞서 살펴본 쿼리문과 비교해 보면
  이 JPA 코드가 어떻게 구성되었는지 좀 더 쉽게 이해할 수 있을 것임

3-13 검색 기능 추가하기

▪ 검색 기능 구현하기

▪ 질문 리포지터리 수정하기

- 앞서 작성한 Specification을 통해
  질문을 조회하려면
  QuestionRepository를
  다음과 같이 수정해야 함

• /question/QuestionRepository.java

```
package com.mysite.sbb.question;

import java.util.List;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.data.jpa.repository.JpaRepository;
public interface QuestionRepository extends JpaRepository<Question, Integer> {
    Question findBySubject(String subject);
    Question findBySubjectAndContent(String subject, String content);
    List<Question> findBySubjectLike(String subject);
    Page<Question> findAll(Pageable pageable);
    Page<Question> findAll(Specification<Question> spec, Pageable pageable);
}
```

3-13 검색 기능 추가하기

- **검색 기능 구현하기**
  - **질문 리포지터리 수정하기**
    - 추가한 findAll 메서드는 Specification과 Pageable 객체를 사용하여 DB에서 Question 엔티티를 조회한 결과를 페이징하여 반환함
  - **질문 서비스 수정하기**
    - 검색어를 포함하여 질문 목록을 조회하기 위해 다시 QuestionService로 돌아와 getList 메서드를 다음과 같이 수정해 보자

3-13 검색 기능 추가하기

▪ 검색 기능 구현하기

▪ 질문 서비스 수정하기

• /question/QuestionService.java

```
(... 생략 ...)
public class QuestionService {

    (... 생략 ...)

    public Page<Question> getList(int page, String kw) {
        List<Sort.Order> sorts = new ArrayList<>();
        sorts.add(Sort.Order.desc("createDate"));
        Pageable pageable = PageRequest.of(page, 10, Sort.by(sorts));
        Specification<Question> spec = search(kw);
        return this.questionRepository.findAll(spec, pageable);
    }

    (... 생략 ...)
}
```

검색어를 의미하는 매개변수 kw를
getList 메서드에 추가하고
kw값으로 Specification 객체를 생성하여
findAll 메서드 호출 시 전달함

3-13 검색 기능 추가하기

▪ 검색 기능 구현하기

▪ 질문 컨트롤러 수정하기

QuestionService의 getList 메서드의 입력 항목이 변경되었으므로 QuestionController도 다음과 같이 수정해야 함

• /question/QuestionController.java

```
(... 생략 ...)
public class QuestionController {

    (... 생략 ...)

    @GetMapping("/list")
    public String list(Model model, @RequestParam(value = "page", defaultValue =
"0") int page, @RequestParam(value = "kw", defaultValue = "") String kw) {
        Page<Question> paging = this.questionService.getList(page, kw);
        model.addAttribute("paging", paging);
        model.addAttribute("kw", kw);
        return "question_list";
    }

    (... 생략 ...)
}
```

3-13 검색 기능 추가하기

▪ 검색 기능 구현하기

▪ 질문 컨트롤러 수정하기

- 검색어에 해당하는 kw 매개변수를 추가했고 기본값으로 빈 문자열을 설정함
- 그리고 화면에서 입력한 검색어를 화면에 그대로 유지하기 위해 model.addAttribute(" kw " ,kw)로 kw값을 저장함
- 이제 화면에서 검색어가 입력되면 kw값이 매개변수로 들어오고 해당 값으로 질문 목록이 검색되어 조회될 것임

3-13 검색 기능 추가하기

▪ 검색 화면 구현하기

▪ 검색창 만들기

검색어를 입력할 수 있는 텍스트 창을 다음과 같이 질문 목록 템플릿에 추가해 보자

• /templates/question_list.html

```
<html layout:decorate="~{layout}">
<div layout:fragment="content" class="container my-3">
    <div class="row my-3">
        <div class="col-6">
            <a th:href="@{/question/create}" class="btn btn-primary">질문 등록하기</a>
        </div>
        <div class="col-6">
            <div class="input-group">
                <input type="text" id="search_kw" class="form-control"
th:value="${kw}">
                <button class="btn btn-outline-secondary" type="button"
id="btn_search">찾기</button>
            </div>
        </div>
    </div>
    <table class="table">
        (... 생략 ...)
    </table>
    <!-- 페이징 처리  시작 -->
    (... 생략 ...)
    <!-- 페이징 처리  끝 -->
    <a th:href="@{/question/create}" class="btn btn-primary">질문 등록하기</a>
</div>
</html>
```

기존 코드를 삭제한다.

3-13 검색 기능 추가하기

▪ 검색 화면 구현하기

▪ 검색창 만들기

- 질문 목록 위에 검색창이 노출되도록 이와 같이 입력하여
  <table> 태그 상단 오른쪽에 검색어를 입력할 수 있는 텍스트 창을 생성함

- 기존에 아래에 있던 [질문 등록하기] 버튼은 검색창의 왼쪽에 배치되도록 수정함

- 자바스크립트에서 이 검색창에 입력된 값을 읽을 수 있도록
  검색창의 id 속성에 'search_kw'라는 값을 추가한 점을 한번 더 살펴보자

```
<input type="text" id="search_kw" class="form-control" th:value="${kw}">
```

3-13 검색 기능 추가하기

▪ 검색 화면 구현하기

▪ 검색 폼 만들기

- page와 kw를 동시에 GET 방식으로 요청하기 위해
  searchForm을 앞서 코드를 삭제한 자리에 다음과 같이 추가해 보자

• /templates/question_list.html

```
(... 생략 ...)
    <!-- 페이징 처리  끝 -->
    <form th:action="@{/question/list}" method="get" id="searchForm">
        <input type="hidden" id="kw" name="kw" th:value="${kw}">
        <input type="hidden" id="page" name="page" th:value="${paging.number}">
    </form>
</div>
</html>
```

3-13 검색 기능 추가하기

▪ 검색 화면 구현하기

▪ 검색 폼 만들기

- GET 방식으로 요청해야 하므로 method 속성에 'get'을 설정함
- kw와 page는 이전에 요청했던 값을 기억하고 있어야 하므로 value에 값을 유지할 수 있도록 함
- 이전에 요청했던 kw와 page의 값은 컨트롤러로부터 다시 전달받음
- action 속성에는 폼이 전송되는 URL이므로 질문 목록 URL인 /question/list를 지정함

3-13 검색 기능 추가하기

- **검색 화면 구현하기**
  - **페이징 수정하기**
    - 페이징을 처리하는 부분도 기존의 ?page=1과 같이 직접 URL을 링크하는 방식이 아니라 값을 읽어 폼에 설정할 수 있도록 다음과 같이 변경해야 함
    - 왜냐하면 검색어가 있을 경우 검색어와 페이지 번호를 함께 전송해야 하기 때문

3-13 검색 기능 추가하기

▪ 검색 화면 구현하기

▪ 페이징 수정하기

• /templates/question_list.html

```
(... 생략 ...)
<!-- 페이징 처리 시작 -->
<div th:if="${!paging.isEmpty()}">
    <ul class="pagination justify-content-center">
        <li class="page-item" th:classappend="${!paging.hasPrevious} ? 'disabled'">
            <a class="page-link" href="javascript:void(0)"
            th:data-page="${paging.number-1}">
                <span>이전</span>
            </a>
        </li>
        <li th:each="page: ${#numbers.sequence(0, paging.totalPages-1)}"
            th:if="${page >= paging.number-5 and page <= paging.number+5}"
            th:classappend="${page == paging.number} ? 'active'" class="page-item">
            <a th:text="${page}" class="page-link" href="javascript:void(0)"
            th:data-page="${page}"></a>
        </li>
        <li class="page-item" th:classappend="${!paging.hasNext} ? 'disabled'">
            <a class="page-link" href="javascript:void(0)"
            th:data-page="${paging.number+1}">
                <span>다음</span>
            </a>
        </li>
    </ul>
</div>
<!-- 페이징 처리 끝 -->
(... 생략 ...)
```

3-13 검색 기능 추가하기

▪ 검색 화면 구현하기

▪ 페이징 수정하기

- 모든 페이지 링크를 href 속성에 직접 입력하는 대신 data-page 속성으로 값을 읽을 수 있도록 함

```
<a class="page-link" th:href="@{|?page=${paging.number-1}|}">
```

- 이와 같은 형태의 페이지 링크를 다음과 같이 변경함

```
<a class="page-link" href="javascript:void(0)" th:data-page="${paging.number-1}">
```

3-13 검색 기능 추가하기

▪ 검색 화면 구현하기

▪ 검색 스크립트 추가하기

- page, kw 매개변수를 동시에 요청할 수 있는 자바스크립트를 다음과 같이 추가해 보자

• /templates/question_list.html

```
(... 생략 ...)
    <!-- 페이징 처리 끝 -->
    <form th:action="@{/question/list}" method="get" id="searchForm">
        <input type="hidden" id="kw" name="kw" th:value="${kw}">
        <input type="hidden" id="page" name="page" th:value="${paging.number}">
    </form>
</div>
<script layout:fragment="script" type='text/javascript'>
const page_elements = document.getElementsByClassName("page-link");
Array.from(page_elements).forEach(function(element) {
    element.addEventListener('click', function() {
        document.getElementById('page').value = this.dataset.page;
        document.getElementById('searchForm').submit();
    });
});
const btn_search = document.getElementById("btn_search");
btn_search.addEventListener('click', function() {
    document.getElementById('kw').value = document.getElementById('search_kw').
value;
    document.getElementById('page').value = 0;  // 검색 버튼을 클릭할 경우 0페이지부터
조회한다.
  document.getElementById('searchForm').submit();
});
</script>
</html>
```

3-13 검색 기능 추가하기

▪ 검색 화면 구현하기

▪ 검색 스크립트 추가하기

- 만약 다음과 같이 class 속성값으로 'page-link'라는 값이 있는 페이지 링크를 질문 목록 화면에서 클릭하는 경우를 보자

```
<a class="page-link" href="javascript:void(0)" th:data-page="${paging.number-1}">
```

3-13 검색 기능 추가하기

▪ 검색 화면 구현하기

▪ 검색 스크립트 추가하기

- 이 링크의 data-page 속성값을 읽어 searchForm의 page 필드에 설정하여 searchForm을 요청하기 위해 다음과 같은 스크립트를 추가함

```
const page_elements = document.getElementsByClassName("page-link");
Array.from(page_elements).forEach(function(element) {
    element.addEventListener('click', function() {
        document.getElementById('page').value = this.dataset.page;
        document.getElementById('searchForm').submit();
    });
});
```

3-13 검색 기능 추가하기

▪ 검색 화면 구현하기

▪ 검색 스크립트 추가하기

- 그리고 [검색] 버튼을 클릭하면 검색창에 입력된 값을 searchForm의 kw 필드에 설정하여 searchForm을 요청하도록 다음과 같은 스크립트를 추가함

```
const btn_search = document.getElementById("btn_search");
btn_search.addEventListener('click', function() {
    document.getElementById('kw').value = document.getElementById('search_kw').
value;
    document.getElementById('page').value = 0;  // 검색 버튼을 클릭할 경우 0페이지부
터 조회한다.
    document.getElementById('searchForm').submit();
});
```

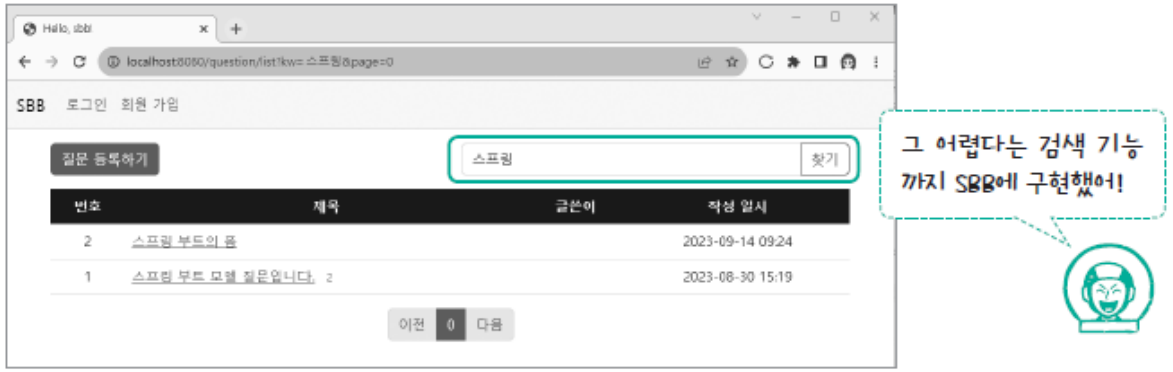
3-13 검색 기능 추가하기

- **검색 화면 구현하기**
  - **검색 스크립트 추가하기**
    - 그리고 [검색] 버튼을 클릭하는 경우는 새로운 검색에 해당되므로 page에 항상 0을 설정하여 첫 페이지로 요청하도록 함

3-13 검색 기능 추가하기

▪ 검색 기능 확인하기

- 이제 질문 목록 화면으로 돌아가 검색창에 '스프링'을 검색어로 조회하면 다음과 같이 해당 검색어가 포함된 게시물만 조회될 것임



3-13 검색 기능 추가하기

- **@Query 애너테이션 사용하기**
  - 쿼리에 익숙하다면 이와 같이 자바 코드로 쿼리를 생성하는 방식보다 직접 쿼리를 작성하는 게 훨씬 편하게 여겨질 것임
  - 이번에는 Specification 대신 쿼리를 직접 작성하여 검색 기능을 구현하는 방법을 간단히 알아보자

3-13 검색 기능 추가하기

▪ @Query 애너테이션 사용하기

1. QuestionRepository에 다음과 같은 메서드를 추가해 보자

• /question/QuestionRepository.java

```
(... 생략 ...)
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

public interface QuestionRepository extends JpaRepository<Question, Integer> {
    (... 생략 ...)

    @Query("select "
            + "distinct q "
            + "from Question q "
            + "left outer join SiteUser u1 on q.author=u1 "
            + "left outer join Answer a on a.question=q "
            + "left outer join SiteUser u2 on a.author=u2 "
            + "where "
            + "    q.subject like %:kw% "
            + "    or q.content like %:kw% "
            + "    or u1.username like %:kw% "
            + "    or a.content like %:kw% "
            + "    or u2.username like %:kw% ")
    Page<Question> findAllByKeyword(@Param("kw") String kw, Pageable pageable);
}
```

3-13 검색 기능 추가하기

▪ @Query 애너테이션 사용하기

1. 여기서는 @Query 애너테이션이 적용된 findAllByKeyword 메서드를 추가함.
앞에서 살펴본 쿼리를 @Query로 구현한 것임.

   이때 @Query는 반드시 테이블 기준이 아닌 엔티티 기준으로 작성해야 함.
즉, site_user와 같은 테이블명 대신 SiteUser처럼 엔티티명을 사용해야 하고,

   조인문에서 보듯이 q.author_id=u1.id와 같은 컬럼명 대신
q.author=u1처럼 엔티티의 속성명을 사용해야 함

3-13 검색 기능 추가하기

- **@Query 애너테이션 사용하기**

1. 그리고 @Query에 매개변수로 전달할 kw 문자열은
   메서드의 매개변수에 @Param("kw")처럼 @Param 애너테이션을 사용해야 함.

   검색어를 의미하는 kw 문자열은 @Query 안에서 :kw로 참조됨

3-13 검색 기능 추가하기

▪ @Query 애너테이션 사용하기

2. 작성한 findAllByKeyword 메서드를 사용하기 위해 QuestionService를 다음과 같이 수정하자

• /question/QuestionService.java

```
(... 생략 ...)
public class QuestionService {

    (... 생략 ...)

    public Page<Question> getList(int page, String kw) {
        List<Sort.Order> sorts = new ArrayList<>();
        sorts.add(Sort.Order.desc("createDate"));
        Pageable pageable = PageRequest.of(page, 10, Sort.by(sorts));
        return this.questionRepository.findAllByKeyword(kw, pageable);
    }
    (... 생략 ...)
}
```

3-13 검색 기능 추가하기

- **@Query 애너테이션 사용하기**

2. Specification 인터페이스를 사용하기 위해 작성했던 내용 대신
   이와 같이 작성해도 동일하게 동작할 것임.

   이와 같이 SQL을 알고, @Query 애너테이션을 사용한다면
   검색 기능을 좀 더 간단하게 구현할 수 있음

되 / 새 / 김 / 문 / 제

P.316~318

Q1. 회원 가입 기능 업그레이드하기

Q2. 권한 변경해 보기

Q3. 버튼 클릭 막아두기