머스테치로 화면 구성하기

작성일시	@2022년 6월 19일 오전 9:59
파트	4장
자료	
작성자	
속성	

📢 템플릿 엔진

지정된 템플릿 양식과 데이터가 합쳐져 HTML 문서를 출력하는 소프트웨어

✅ 서버 템플릿 엔진

- 동작 방식 : 서버에서 자바 코드로 문자열을 만든 뒤 이 문자열을 HTML로 변환하여 브라우저에 전달
- 백엔드 영역
- ex) JSP, Freemarker, Thymeleaf

✅ 클라이언트 템플릿 엔진

- 동작 방식 : 서버는 Json, XML 형식의 데이터만 전달하고 화면 생성은 브라우저에서 수행
- 프론트엔드 영역
- ex) React, Vue

→ 간단하고 지원하는 언어가 많은 mustache를 사용!

📢 화면 구현하기

✅ 기본 화면

- 부트스트랩 외부 CDN 사용
- 각 페이지에 공통되는 부분은 별도의 파일로 분리하여 필요한 곳에서 가져다 쓰기
- 페이지 로딩 속도를 높이기 위해 CSS는 header에, JS는 footer에 둠
- 부트스트랩은 제이쿼리에 의존하므로 제이쿼리가 부트스트랩보다 먼저 호출

✅ 게시글 등록 화면

- **IndexController**
  - `/posts/save` 경로로 들어오면 “post-save”라는 이름의 뷰를 찾아서 보여줌 (경로와 확장자는 ViewResolver에서 처리)

```
@GetMapping("/posts/save")
    public String postsSave() {
        return "posts-save";
    }
```

// js로 등록 버튼 클릭 시 등록 로직 실행되도록 구현

✅ 게시글 전체 조회 화면

- **index.mustache**
  - List가 들어오면 순회하면서 객체마다 getter를 호출하여 테이블의 행을 생성한다.

```
...
<table class="table table-horizontal table-bordered">
        <thead class="thead-strong">
        <tr>
            <th>게시글번호</th>
            <th>제목</th>
            <th>작성자</th>
            <th>최종수정일</th>
        </tr>
        </thead>
        <tbody id="tbody">
        {{#posts}}
            <tr>
                <td>{{id}}</td>
                <td><a href="/posts/update/{{id}}">{{title}}</a></td>
                <td>{{author}}</td>
                <td>{{modifiedDate}}</td>
            </tr>
        {{/posts}}
        </tbody>
    </table>
...
```

- **PostsRepository**
  - JpaRepository를 상속받는 경우 findAll(), findById(), delete() 등 기본적으로 제공되는 메서드가 있는데, 이 외에 추가적인 기능을 제공하는 메서드를 만들고 싶다면 `@Query` 애노테이션으로 직접 쿼리를 지정해준다.
  - 여기서는 id를 내림차순 정렬한 Posts 리스트를 반환해줌

```
@Repository
public interface PostsRepository extends JpaRepository<Posts, Long> {

    // SpringJPA가 제공하지 않는 메서드는 쿼리문을 직접 지정해서 정의
    @Query("SELECT p FROM Posts p ORDER BY p.id DESC")
```

```
    List<Posts> findAllDesc();
}
```

- **PostsService**
  - `postsRepository.findAllDesc()` 결과를 Posts 대신 화면에 보여줄 PostsListResponseDto로 변환해서 리스트로 반환
  - `readOnly=true` : 트랜잭션 범위는 유지하되, 조회 기능만 남겨두어 조회 성능 향상 → 더 알아보기

```
// 전체 게시글 조회
    @Transactional(readOnly = true)
    public List<PostsListResponseDto> findAllDesc() {
        return postsRepository.findAllDesc().stream()
                .map(PostsListResponseDto::new) // .map(posts -> new PostsListResponseDto(posts))와 동일한 표현
                .collect(Collectors.toList());
    }
```

- **PostsListResponseDto**

```
@Getter
public class PostsListResponseDto { // 게시글 목록에 나타낼 정보

    private Long id;
    private String title;
    private String author;
    private LocalDateTime modifiedDate;

    public PostsListResponseDto(Posts entity) {
        this.id = entity.getId();
        this.title = entity.getTitle();
        this.author = entity.getAuthor();
        this.modifiedDate = entity.getModifiedDate();
    }
}
```

- **PostsApiController**
  - Model 객체에 `postsService.findAllDesc()` 결과를 “posts”라는 이름으로 담아 뷰로 전달한다.
  - “posts” 객체는 view에서 루프를 돌릴 때 사용한다.

```
@GetMapping("/")
    public String index(Model model) {
        // Model에 게시글 조회 결과 전달
        model.addAttribute("posts", postsService.findAllDesc());
        return "index";
    }
```

✅ 게시글 수정 화면

- posts-update.mustache

```
<form>
    <div class="form-group">
        <label for="id">글 번호</label>
        <input type="text" class="form-control" id="id" value="{{post.id}}" readonly>
    </div>
    <div class="form-group">
        <label for="title">제목</label>
        <input type="text" class="form-control" id="title" value="{{post.title}}">
    </div>
    <div class="form-group">
        <label for="author"> 작성자 </label>
        <input type="text" class="form-control" id="author" value="{{post.author}}" readonly>
    </div>
    <div class="form-group">
        <label for="content"> 내용 </label>
        <textarea class="form-control" id="content">{{post.content}}</textarea>
    </div>
</form>

<a href="/" role="button" class="btn btn-secondary">취소</a>
<button type="button" class="btn btn-primary" id="btn-update">수정 완료</button>
```

// js 로 수정 완료 버튼 클릭 시 저장되도록 구현

- **IndexController**
  - 수정할 게시글의 상세페이지를 보여주는 부분
  - 경로변수로 들어온 id를 찾은 결과로 반환된 PostsResponseDto를 모델에 전달

```
@GetMapping("/posts/update/{id}") // 수정 화면에서 보여줄 데이터를 모델에 전달
public String postsUpdate(@PathVariable Long id, Model model) {
    PostsResponseDto dto = postsService.findById(id);
    model.addAttribute("post", dto);

    return "posts-update";
}
```

- **PostsAPIController**
  - 실제적으로 수정이 동작하는 부분
  - 사용자가 입력한 수정 사항이 PostsUpdateRequestDto로 매핑되고 이를 사용해 update 수행

```
@PutMapping("/api/v1/posts/{id}") // 넘어온 데이터를 DTO로 받아 update 수행
  public Long update(@PathVariable Long id, @RequestBody PostsUpdateRequestDto postsUpdateRequestDto) {
      return postsService.update(id, postsUpdateRequestDto);
  }
```

✅ 게시글 삭제

```
<button type="button" class="btn btn-danger" id="btn-delete">삭제</button>
```

// js 로 삭제 버튼 클릭 시 저장되도록 구현

- **PostsService**
  - id로 Posts를 찾아 삭제 수행

```
// 게시글 삭제
  @Transactional
  public void delete(Long id) {
      Posts posts = postsRepository.findById(id)
                      .orElseThrow(() -> new IllegalArgumentException("해당 게시글이 존재하지 않습니다. id = " + id));
      postsRepository.delete(posts);
  }
```

- PostsApiController
  - 경로 변수로 들어온 아이디를 찾아 삭제 로직 수행

```
@DeleteMapping("/api/v1/posts/{id}")
    public Long delete(@PathVariable Long id) {
        postsService.delete(id);
        return id;
    }
```